О. В. Редич

ВЕЛИКІ ДАНІ ТА МАШИННЕ НАВЧАННЯ

Навчальний посібник



ДЕРЖАВНИЙ ПОДАТКОВИЙ УНІВЕРСИТЕТ

ВЕЛИКІ ДАНІ ТА МАШИННЕ НАВЧАННЯ

Навчальний посібник

Ірпінь
2025

УДК 004.4(075.8)
ББК 32.97я73
Р 33

Рекомендовано до друку Вченою радою
Державного податкового університету
(протокол № 17 від 26 червня 2025 року)

Рецензенти:

Виклюк В. І., доктор технічних наук, професор кафедри систем штучного інтелекту Національного університету «Львівська політехніка»; **Путренко В. В.,** доктор технічних наук, старший науковий співробітник, професор кафедри інформаційних технологій ТОВ ВНЗ «Американ Юніверсіті Київ»; **Омельчук А. А.,** кандидат технічних наук, доцент кафедри комп'ютерних та інформаційних технологій і систем Державного податкового університету.

Автор:

Редич Олександр Володимирович, кандидат економічних наук, доцент кафедри комп'ютерних та інформаційних технологій і систем.

Редич О. В.

Р 33 **Великі дані та машинне навчання** : навчальний посібник / О. В. Редич. – Ірпінь : Державний податковий університет, 2025. – 468 с.
ISBN 978-966-337-771-1

У навчальному посібнику розглянуто теоретичні відомості, основні поняття та практичні аспекти використання інструментарію великих даних (Big Data), машинного навчання (Machine Learning) та штучного інтелекту (AI). Наведені приклади та практичні рекомендації для застосування алгоритмів машинного навчання на основі бібліотек Python мають прикладний характер і можуть бути застосовані в різних задачах аналітики даних та під час розробки компонентів штучного інтелекту.

Посібник рекомендується для викладачів і здобувачів ступеня бакалавра й магістра спеціальностей F3 «Комп'ютерні науки», F2 «Інженерія програмного забезпечення», для всіх, хто вивчає та досліджує технології галузі науки про дані (Data Science).

УДК 004.4(075.8)
ББК 32.97я73

© Редич О. В., 2025
ISBN 978-966-337-771-1
©Державний податковий університет, 2025

ЗМІСТ

ВСТУП.....9

ТЕМА 1. ОСНОВНІ ПОНЯТТЯ ПРО ВЕЛИКІ ДАНІ 13
1.1. Великі дані, інтернет речей, всеохоплюючий інтернет.....13
1.1.1. Поняття Великих даних (Big Data) 13
1.1.2. Інтернет речей 19
1.1.3. Всеохоплюючий інтернет 20
1.1.4. Відкриті дані як джерело Big Data.....25
Питання для самоконтролю.....30
1.2. Формати даних, підготовка даних для аналізу.....31
1.2.1. Формати даних XLS / XLSX.....31
1.2.2. Графічні формати JPEG, PNG, TIFF 31
1.2.3. Текстові формати даних TXT, CSV, TSV, Markdown, PDF 34
1.2.4. Формати даних JSON, XML, RDF.....39
1.2.5. Формати геопросторових даних.....43
Питання для самоконтролю.....50
1.3. Огляд нереляційних БД (NoSQL DB).....51
1.3.1. Причини появи NoSQL баз даних.....51
1.3.2. Бази даних виду «ключ-значення» (key-value DB) 51
1.3.3. Документо-орієнтовані бази даних.....52
1.3.4. Графові бази даних 54
1.3.5. Bigtable-схожі бази даних (стовпчикові БД).....56
Питання для самоконтролю.....62

ТЕМА 2. ЕКОСИСТЕМА BIG DATA 63
2.1. Технологія Hadoop.....63
2.1.1. Архітектура екосистеми Big Data на базі Hadoop.....63
2.1.2. Особливості використання Hadoop.....68
2.1.3. Pig – високорівнева мова опису потоків даних 71
2.1.4. Hive: SQL структура для організації сховищ даних 72
2.1.5. HBase – розподілена СУБД зі зберіганням за стовпцями.....73

2.1.6. Приклад розгортання віртуальної
машини Oracle Big Data 76
Питання для самоконтролю 80
2.2. Збір і оброблення Big Data: HDFS та Map Reduce 82
2.2.1. Розподілена файлова система HDFS 82
2.2.2. Kafka – розподілена потокова платформа 88
2.2.3. Файлова система Cassandra
та її порівняння з Hadoop 90
2.2.4. MapReduce як технологія
розподілених обчислень 92
2.2.5. Програмна модель MapReduce 97
2.2.6. Приклад отримання та оброблення Big Data 99
Питання для самоконтролю 108
2.3. Spark: розподілена система обробки даних 110
2.3.1. Огляд компонентів Spark 110
2.3.2. Переваги Spark над MapReduce 112
2.3.3. Основні терміни Spark 113
2.3.4. Компоненти Spark 114
2.3.5. Лямбда-архітектура обробки даних 117
Питання для самоконтролю 119
2.4. Платформи та інструменти Big Data 121
2.4.1. Oracle Big Data SQL 121
2.4.2. Amazon Web Services 123
2.4.3. Microsoft Azure 124
2.4.4. Google Cloud Platform 125
Питання для самоконтролю 126

**ТЕМА 3. МЕТОДИ ТА ІНСТРУМЕНТАРІЙ
МАШИННОГО НАВЧАННЯ** 128
3.1. Data Science: наука про дані і машинне навчання 128
3.1.1. Штучний інтелект та інтелектуальний аналіз 128
3.1.2. Життєвий цикл машинного
навчання й Data Science 130
3.1.3. Навчання «з учителем» і «без учителя» 135
3.1.4. Алгоритми машинного навчання 137
Питання для самоконтролю 142

3.2. Дослідження й розуміння даних на прикладах 144
3.2.1. Основні статистики розвідувального аналізу даних 144
3.2.2. Підготовка даних: очищення,
навчальна, тестова вибірики 155
3.2.3. Формування наборів даних для навчання 159
3.2.4. Нормалізація (стандартизація) даних 161
Питання для самоконтролю 167
3.3. Прикладні бібліотеки та хмарні платформи для МН 169
3.3.1. Бібліотеки Python 169
3.3.2. Oracle Machine Learning 172
3.3.3. Microsoft Azure Machine Learning 176
3.3.4. Amazon Web Servises Machine Learning 177
3.3.5. Google Colaboratory 180
3.3.6. Matlab Machine Learning 182
Питання для самоконтролю 185

ТЕМА 4. МАШИННЕ НАВЧАННЯ
В КЛАСИЧНИХ МЕТОДАХ АНАЛІЗУ 187
4.1. Регресійно-кореляційний аналіз 187
4.1.1. Основи регресійно-кореляційного аналізу 187
4.1.2. Приклад регресійно-кореляційного аналізу 192
4.1.3. Оцінювання якості регресійної моделі 203
Питання для самоконтролю 213
4.2. Машинне навчання в класифікаційних задачах 215
4.2.1. Основні методи класифікації 215
4.2.2. Основи застосування
інструментарію дерев рішень 218
4.2.3. Алгоритми дерев рішень ID3, C4.5 222
Питання для самоконтролю 230
4.3. Оцінювання якості класифікаційних моделей 231
4.3.1. Основні показники класифікаційних
якостей моделей 231
4.3.2. Класифікація з урахуванням втрат 235
4.3.3. Lift, Profit, ROC криві 237
4.3.4. Приклад побудови дерева рішень 252
Питання для самоконтролю 264

4.4. Кластерний аналіз 266
4.4.1. Сутність методів кластерного аналізу 266
4.4.2. Алгоритм кластеризації k-means 268
4.4.3. Математичне пояснення дендрограми у кластерному аналізі 271
4.4.4. Приклад кластерного аналізу: KMeans 275
Питання для самоконтролю 285
4.5. Асоціативні правила 286
4.5.1. Основні відомості про асоціативні правила 286
4.5.2. Пошук асоціативних правил 293
4.5.3. Стратегії ведення пошуку правил 296
4.5.4. Приклад побудови асоціативних правил: *apriori* 297
Питання для самоконтролю 306
4.6. Прогнозування часових рядів 307
4.6.1. Методи та інструментарій прогнозування часових рядів 307
4.6.2. ARIMA. Авторегресійна інтегрована модель ковзного середнього 309
4.6.3. SARIMA. Сезонна авторегресійна інтегрована модель ковзного середнього 311
4.6.4. Моделі експоненційного згладжування 313
4.6.5. Приклади аналізу функцій автокореляції (АКФ, ЧАКФ) 316
4.6.6. Приклади використання моделей ARIMA, SARIMA 331
4.6.7. Приклад використання моделі Холта-Вінтерса 336
Питання для самоконтролю 339
4.7. Алгоритми пошуку аномалій в даних 341
4.7.1. Огляд задач пошуку аномалій у різних сферах 341
4.7.2. Основні методи пошуку аномалій в даних 344
4.7.3. Приклад використання методу Евклідової відстані 346
Питання для самоконтролю 352

ТЕМА 5. МЕТОДИ ТА ІНСТРУМЕНТАРІЙ ШТУЧНОГО ІНТЕЛЕКТУ 354
5.1. Штучні нейронні мережі 354
5.1.1. Сучасний стан та актуальність штучних нейронних мереж 354
5.1.2. Біологічний нейрон і його штучна модель 359
5.1.3. Функції активації та їх використання в ШНМ 364
5.1.4. Навчання ШНМ як задача мінімізації помилки 367
5.1.5. Градієнтний спуск і алгоритм зворотного поширеня помилки 374
5.1.6. Методи та інструменти оптимізації функції помилки 379
Питання для самоконтролю 381
5.2. Згорткові нейронні мережі (CNN) 383
5.2.1. Структура згорткових нейронних мереж 383
5.2.2. Процедури згортання (convolution) 384
5.2.3. Приклад використання згорткових нейромереж 392
Питання для самоконтролю 410
5.3. Рекурентні штучні нейромережі (RNN) та мережі LSTM 411
5.3.1. Основні відомості про RNN 411
5.3.2. Використання, обмеження і покращення RNN 413
5.3.3. LSTM архітектура нейронних мереж 415
5.3.4. Приклад у прогнозування часових рядів: RNN+LSTM 420
Питання для самоконтролю 425
5.4. Інструментарій AI для аналізу геопросторових даних 427
5.4.1. Актуальність просторового аналізу 427
5.4.2. Oracle Spatial AI для задач геопросторового аналізу 428
5.4.3. Приклади блокнотів моделей Spatial AI на Oracle ML 430
5.4.4. Інтеграція Spatial AI з OML 433

5.4.5. API Python для просторового
штучного інтелекту 434
5.4.6. Приклад використання Oracle Spatial AI 435
Питання для самоконтролю 444

ТЕРМІНОЛОГІЧНИЙ ДОВІДНИК (ГЛОСАРІЙ) 445

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 449

ДОДАТКИ 454
ДОДАТОК А. Порівняльна таблиця класичних методів
машинного навчання 454
ДОДАТОК Б. Методи класифікації 459
ДОДАТОК В. Довідник переднавчених моделей
штучних нейронних мереж 461
ДОДАТОК Г. Опис набору ImageNet
для комп'ютерного зору 464
ДОДАТОК Д. Опис та архітектура моделі VGG16 466

ВСТУП

Комп'ютерна інтелектуалізація реального сектору економіки на сьогодні є одним із визначальних процесів, що сприяє як економічному зростанню загалом, так і розвитку самої цифрової індустрії, яка пропонує інноваційні технології.

Розвиток технологій великих даних, штучного інтелекту (далі – ШІ), машинного навчання та потенційна поява загального штучного інтелекту обіцяють стати одними із найвпливовіших чинників, що визначатимуть траєкторію розвитку індустрії та суспільства в найближчі роки. Їхній вплив буде багатогранним і глибоким. В епоху динамічних змін Індустрії 5.0, технічною основою якої стають штучний інтелект, Web3, блокчейн, метавсесвіти та NFT, які революціонізують взаємодію суспільств з інформацією, актуальність вивчення великих даних, машинного навчання та штучного інтелекту стає критично важливою. Основними аргументами, які це підтверджують, є:

GPT та інші інструменти генеративного ШІ демонструють, як аналіз великих текстових даних може привести до створення нового контенту, програмного коду чи реалістичного діалогу;

Індустрія 5.0 використовує ШІ для підвищення ефективності, гнучкості та стійкості виробництв, де люди та роботи співпрацюють, аналізуючи дані з датчиків і виробничих ліній;

Web3 та *блокчейн* покладаються на криптографію, але ШІ може використовуватися для аналізу патернів транзакцій, виявлення шахрайства, оптимізації роботи децентралізованих додатків (dApps) та управління децентралізованими автономними організаціями (DAO) і децентралізованими фінансами (DF).

Метавсесвіти неможливі без ШІ, який створює реалістичні аватари, інтелектуальних неігрових персонажів (NPC), динамічні середовища та аналізує поведінку користувачів для персоналізації. Дані про взаємодії тут є ключовими.

NFT хоч і є унікальними токенами, їхня цінність, автентичність і навіть створення (наприклад, AI-generated art) можуть аналізуватися або здійснюватися за допомогою ML.

Отже, вивчення про великі дані, машинне навчання та ШІ – це не просто тренд, а фундаментальна потреба для розуміння та активної участі у формуванні майбутнього. Ці технології є сполучною ланкою та рушійною силою для всіх згаданих інновацій, відкриваючи спосіб до створення більш інтелектуальних, ефективних і захопливих цифрових та фізичних світів.

Українським інститутом майбутнього (Ukrainian Institute for the Future (UIF)) здійснено прогноз змін в Україні та світі та змоделювано можливі сценарії розвитку подій. Експерти UIF формулюють практичні рекомендації уряду щодо дій як доповіді, аналітичні записки, законопроєкти тощо. Одними із основних доробків UIF є Стратегія України до 2030 року та книга «Україна 2030Е – країна з розвинутою цифровою економікою», у якій передбачається побудова так званого нейронету, тобто мережі, де комунікації між людьми, біосистемами та речами здійснюватимуться на принципах нейрокомунікації, використання штучного інтелекту та всеосяжного інтернету людей, речей, даних, процесів тощо [1].

У липні 2022 року в Швейцарському місті Лугано відбулась Міжнародна конференція з питань відновлення України, у якій взяли участь понад 40 країн світу й приблизно 20 міжнародних організацій, і на якій була ухвалена спільна декларація про підтримку України на шляху відновлення після військової агресії рф. На конференції було представлено «План Маршалла» відбудови України, розрахований на 10 років, який орієнтовно оцінюється в 750 млрд доларів. Голова Міністерства цифрової трансформації України Михайло Федоров презентував діджитал-частину плану повоєнного відновлення України, яку назвали «Digital for Freedom». До цієї програми пропонується долучитися провідним технологічним компаніям світу та перетворити її в програму «цифрового ленд-лізу» провідних технологічних компаній для України. Програма «Digital for Freedom» поділена на ряд масштабних проєктів [2].

Отже, державні установи України, бізнес-структури та їхні промислові технології повинні бути готовими функціонувати в середовищі цифрових інфраструктур як з погляду організаційно-технічної інтероперабельності, доступу, так і з інноваційного

погляду – через створення умов та стимулів заохочувати бізнес і державу до цифровізації. Розвиток такої «відкритої держави» буде сприяти модернізації економіки, її оздоровлення та підвищення конкурентоспроможності.

Метою посібника є формування в здобувачів вищої освіти цілісного розуміння великих даних, технологій їхнього накопичення та обробки, надання системних знань про процеси аналізу даних загалом, особливості застосування в цьому контексті методів та інструментарію машинного навчання, оволодіння основами застосування даних технологій у прикладних завданнях аналізу даних.

Посібник орієнтований для використання в дисциплінах підготовки бакалаврів і магістрів спеціальності «Комп'ютерні науки», які вивчають технології аналізу даних, зокрема великі дані, методи та інструментарій машинного навчання, інтелектуального аналізу, ШІ. Матеріал посібника допоможе здобувачам освіти під час підготовки до лабораторних практикумів виконання курсових і дипломних робіт, проведення наукових досліджень, підготовки доповідей на науково-практичних конференціях.

Завдання посібника: висвітлити особливості різних підходів і технологій формування масивів та розвитку інфраструктури великих даних, застосування різних форматів даних, сформувати навички застосування методів та інструментів обробки й візуалізації даних, ознайомити здобувачів вищої освіти із особливостями оброблення даних, технологіями програмування штучних нейронних мереж.

Зміст розділів посібника охоплює всі теми розділів предмета «Великі дані та машинне навчання» для підготовки бакалаврів і окремі теми предметів «Програмування штучних нейронних мереж», «Технології баз даних та Big Data» для підготовки магістрів спеціальності «Комп'ютерні науки» за освітньою програмою «Комп'ютерні інтелектуальні технології», що здійснюється на кафедрі «Комп'ютерних та інформаційних технологій і систем» Державного податкового університету.

У навчальному процесі використовуються ресурси програм освітнього партнерства Oracle Academy та Amazon Web Services

Academy (AWS Academy), інституціональним учасником яких є Державний податковий університет.

Для виконання наведених у посібнику лабораторних робіт з технологій Big Data, практичних завдань для самостійного виконання використовується віртуальна машина Oracle Big Data Lite 4.11, розгорнута на серверних платформах Державного податкового університету і яку можна розгортати на власних ресурсах здобувачів у разі наявності відповідних обчислювальних ресурсів.

Для виконання лабораторних робіт з технологій машинного навчання використовуються хмарні платформи Google Colaboratory, Oracle Cloud Machine Learning, AWS Academy та інші. Для пошуку помилок коду та отримання довідкової інформації з питань програмування нейронних мереж здобувачам дозволяється використовувати засоби генеративного штучного інтелекту з послідуючим авторським поясненням сутності методів, алгоритмів і результатів.

ТЕМА 1
ОСНОВНІ ПОНЯТТЯ ПРО ВЕЛИКІ ДАНІ

1.1. Великі дані, інтернет речей, всеохоплюючий інтернет

1.1.1. Поняття великих даних (Big Data)

Термін Big Data *(чит. англ. «Біг Дейта»)* перекладається українською мовою як «великі дані». Вперше термін Big Data з'явився в пресі 2008 року, коли редактор журналу «*Nature» Кліффорд Лінч* випустив статтю на тему розвитку майбутньої науки за допомогою технологій роботи з великою кількістю даних. До 2009 року цей термін розглядався лише з погляду наукового аналізу, проте після виходу кількох статей преса стала широко використовувати поняття Big Data, і продовжує використовувати його сьогодні.

2010 року стали з'являтися перші спроби вирішити зростаючу проблему великих даних. Були випущені програмні продукти, дія яких була направлена на те, щоб мінімізувати ризики під час використання значних інформаційних масивів.

З 2011 року великими даними зацікавилися такі компанії, як Microsoft, Oracle, EMC і IBM – вони стали першими використовувати напрацювання Big Data в своїх стратегіях.

Як поняття «великі дані», загалом будемо використовувати термін «*Big Data».* Дамо декілька визначень.

Big Data – набори структурованої та неструктурованої інформації настільки великих розмірів, що традиційні способи та підходи, засновані на рішеннях класу бізнесової аналітики та системах управління реляційними базами даних, не можуть бути застосовані до них.

Big Data – часто під цим поняттям у різних контекстах можуть матися на увазі як дані великого обсягу, так і набори методів і засобів їхнього оброблення, наприклад масово-паралельної обробки даних системами категорії NoSQL, алгоритмами MapReduce та програмними каркасами проєкту Hadoop.

Одним із найбільш популярних визначень *великих даних* є підхід, що базується на їхніх визначальних характеристиках – так званих 4V:

Volume (обсяг) – дані великого обсягу. Виклики сьогодення полягають у тому, щоб виявити найбільш ефективні методи та засоби оброблення зростаючих обсягів даних. За прогнозами IDC (International Data Corporation) та Statista, 2025 року обсяг глобального створення даних досяг 181 зетабайта. Це значне збільшення порівняно з 97 зетабайтами даних, створеними 2022 року. Зростання зумовлене збільшенням кількості підключених пристроїв, зокрема поширенням пристроїв інтернету речей і зростаючою тенденцією зберігання даних у хмарі [3; 4].

Velocity (швидкість) – дані експоненційно зростають, і, відповідно, є потреба у високошвидкісних технологіях обробки та отриманні результатів. Повідомлення в соціальних мережах розходиться по всьому інтернету за лічені секунди. Сучасні технології дають змогу аналізувати дані на льоту, навіть не розміщуючи їх у базах даних. Наприклад, дані про ціни акцій, що продаються на Нью-Йоркській фондовій біржі, мають бути доступними для аналізу миттєво. Інфраструктура даних повинна мати можливість негайно реагувати на вимоги програм, які мають доступ до потоку даних.

Variety (різноманітність) – можливість одночасної обробки різних типів структурованих і неструктурованих даних. Ця характеристика означає дані, які рідко бувають у стані цілком готовому для оброблення та аналізу. У недавньому минулому розглядалися тільки структуровані дані, акуратно вбудовані в таблиці реляційних баз даних, наприклад фінансові дані. Сьогодні у формування обсягу *великих даних* значний внесок дають неструктуровані дані, які за оцінюванням становлять від 70 до 90 % світових даних (текст, зображення, відео, голос та ін.) З технологіями *великих даних* наразі є можливість проаналізувати й звести воєдино дані різних типів, як-от повідомлення, розмови в соціальних мережах, фотографії, дані з датчиків, відео- або голосові записи.

Veracity (достовірність) – процес запобігання наповнення наборів даних неточними даними, що зменшують точність аналізу. Наприклад, коли люди підписуються на онлайн-обліковий запис, вони часто використовують неправдиві контактні дані. Збільшення

достовірності під час збору даних зменшує обсяг відповідної очистки даних. Для значної більшості даних їхня якість і точність є слабоконтрольованими (повідомлення в соціальній мережі, скорочення й помилки розмовної мови, ненадійність і неточності контенту). Технологія аналізу *великих даних* дають змогу нині працювати і з такими типами даних.

Великі дані мають чималі обсяги даних у фізичному значенні. Той обсяг даних, який раніше накопичувався роками, тепер генерується щохвилини.

Наприклад, Airbus A380 Engine генерує 1 петабайт даних під час рейсу з Лондона в Сінгапур. Великий адронний колайдер (Large Hadron Collider, LHC) генерує 1 гігабайт даних щосекунди. Квадратний кілометровий масив (Square Kilometre Array, SKA) є найбільшим радіотелескопом у світі. Він генерує 20 екзабайтів даних на день. Це еквівалентно 20 мільярдам гігабайт на день [5].

Інтернет речей (IoT) використовує датчики для генерації даних. Дані можуть надходити від датчиків температури та вологості, які є в сільському господарстві. Датчики нині є у всьому: від смартфонів до автомобілів, від реактивних двигунів до побутової техніки. Список речей з датчиками зростає з кожним роком, це також сприяє експоненційному зростанню Big Data.

У березні 1999 року Міжнародна електротехнічна комісія ввела новий стандарт іменування двійкових чисел. Приставки МЕК схожі з СІ: вони починаються на ті самі склади, але другий склад у всіх двійкових приставок – *бі* (англ. *binary* – двійковий). Стандарт був затверджений міжнародно, але введені назви практично не використовуються, очевидно, через їх неблагозвучність: *кілобайт* звучить приємніше, ніж *кібібайт*.

Варто зазначити, що набір даних, або бізнес-проблема, належить до класифікації *Big Data,* якщо дані є настільки великими, динамічно змінними та складними, що їх стає неможливим зберігати, обробляти та аналізувати за допомогою традиційних додатків і баз даних.

Відповідно, все це свідчить про те, що для обробки *великих даних* потрібні економічні та інноваційні методи обробки інформації. Для розуміння масштабів *великих даних* їхня співрозмірність подається на рис. 1.1.

Кількість байтів									
Десяткова система				Префікси СІ		Двійкові префікси МЕК			
Назва	Скорочення		Степінь	Назва	Степінь	Назва	Скорочення		Степінь
байт	Б	(B)	10^0	-	10^0	байт	Б	(B)	2^0
кілобайт	кБ	(kB)	10^3	кіло-	10^3	кібібайт	КіБ	(KiB)	2^{10}
мегабайт	МБ	(MB)	10^6	мега-	10^6	мебібайт	МіБ	(MiB)	2^{20}
гігабайт	ГБ	(GB)	10^9	гіга-	10^9	гібібайт	ГіБ	(GiB)	2^{30}
терабайт	ТБ	(TB)	10^{12}	тера-	10^{12}	тебібайт	ТіБ	(TiB)	2^{40}
петабайт	ПБ	(PB)	10^{15}	пета-	10^{15}	пебібайт[en]	ПіБ	(PiB)	2^{50}
ексабайт	ЕБ	(EB)	10^{18}	екса-	10^{18}	ексбібайт	ЕіБ	(EiB)	2^{60}
зетабайт	ЗБ	(ZB)	10^{21}	зета-	10^{21}	зебібайт[en]	ЗіБ	(ZiB)	2^{70}

Рисунок 1.1 – Дані в десятковій системі, СІ
та з двійковими префіксами

Нові інструменти оброблення *Big Data* використовують розподілені системи, тому що дані можна зберігати й аналізувати в географічно розподілених базах даних. На основі звіту Visual Networking Index (Cisco VNI), підготовленому корпорацією Cisco, подана динаміка глобального IP-трафіка в 2017–2022 рр. 2022 року щомісячний трафік становив 396 ексабайт щомісячно [5]. Генеративний ШІ за результатами аналізу різних відкритих джерел прогнозує, що 2025 року трафік може збільшитися орієнтовно до 480 ексабайт.

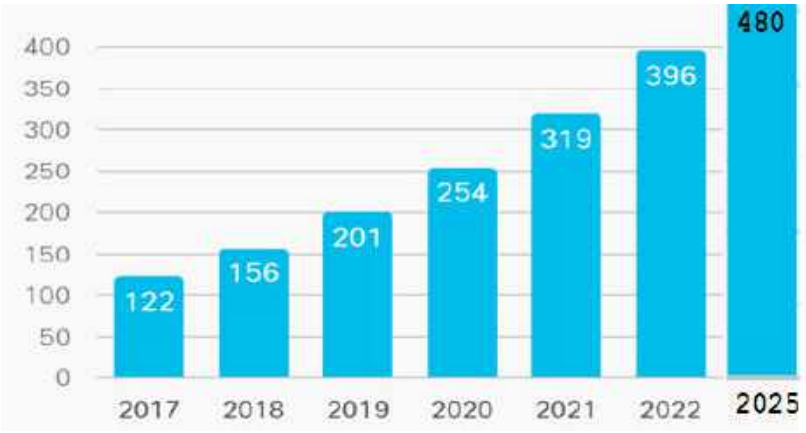


Рисунок 1.2 – Динаміка глобального IP-трафіка 2017–2025 рр.
(ексабайт щомісячно)

Джерело: Cisco VNI Global IP Traffic Forecast, 2017–2022, Gemini: 2025

Виникають запитання, скільки потрібно мати даних, щоб вони називалися *великими даними*? Чи є достатнім для цього 100 терабайт, чи вони повинні бути обсягом не менше 100 петабайт? Проте обсяг є лише одним із критеріїв, оскільки існує потреба в обробці даних у режимі реального часу, що відомо під терміном «Дані в русі». Потреба в інтеграції структурованих і неструктурованих даних також може кваліфікуватись як проблема *великих даних*.

Наприклад, International Data Corporation (IDC) використовує 100 терабайт як розмір набору даних, який кваліфікується як *великі дані*. І знову ж, якщо дані потокові, розмір наборів даних може бути меншим, ніж 100 терабайт. Але якщо такий набір збільшуватиметься в розмірі понад 60 % на рік, такі дані вважатимуться *великими даними*.

Національний інститут стандартів і технологій США (The National Institute of Standards and Technology) дає таке визначення: «Парадигма *великих даних* складається з розподілених систем даних через горизонтально пов’язані, незалежні ресурси для досягнення масштабованості, потрібної для ефективної обробки великих наборів даних».

Прискорення швидкості

За даними *Ookla Speedtest Global Index*, станом на січень 2025 року (це відображає дані, зібрані до кінця 2024 року або на самому початку 2025) глобальна медіанна швидкість завантаження для фіксованого широкосмугового доступу становила 97,61 Мбіт/с, а швидкість вивантаження – 52,84 Мбіт/с.

Швидкість в Україні 2024 року: за даними, опублікованими на початку 2024 року (ймовірно, відображають кінець 2023 або саме початок 2024), середня швидкість фіксованого інтернету в Україні становила приблизно 73,68 Мбіт/с. (Джерело: Mezha.Media, посилаючись на Ookla [5]).

Швидкість мобільного (стільникового) зв’язку зросла більш ніж утричі. Середня швидкість з’єднання з мобільною мережею 2018 року становила 13,2 Мбіт/с, а 2023 досягла 43,9 Мбіт/с.

Глобальна швидкість мобільного інтернету 2024 року: за даними Ookla [5], опублікованими на початку 2025 року (аналізують дані за попередній період, включно із 2024 роком), глобальна

медіанна швидкість завантаження мобільного інтернету сягнула 91,24 Мбіт/с. Це є значним зростанням, практично подвоєнням порівняно з показниками приблизно 50 Мбіт/с роком раніше (на початку 2024 року, аналізуючи дані 2023). Лідерами за швидкістю мобільного інтернету в світі стабільно залишаються такі країни, як ОАЕ, Катар, Кувейт. Наприклад, в ОАЕ середня швидкість завантаження в мобільних мережах на початку 2025 року сягала вражаючих 545,94 Мбіт/с.

Швидкість мобільного інтернету в Україні 2024 року: за даними аналізу за січень 2025 року (і станом на кінець 2024), Україна посіла *81 місце* у світі за швидкістю мобільного інтернету з показником 42,68 Мбіт/с (медіанна швидкість завантаження) [5]. Це певне покращення порівняно з попередніми періодами.

Мобільні додатки

2025 року прогнозується завантаження майже 300 мільйонів мобільних додатків. Найпопулярнішими будуть соціальні мережі, ігри та бізнес-додатки (Statista).

Wi-Fi

2025 року кількість точок доступу Wi-Fi зросла до більше 600 мільйонів порівняно із 169 мільйонами точок 2018 року. У 2024–2025 роках триває активний розвиток інфраструктури Wi-Fi. Відбувається швидкий перехід на новіші стандарти: Wi-Fi 6 та Wi-Fi 6E стають усе більш поширеними, а Wi-Fi 7 починає активно завойовувати ринок, особливо в сегментах, що вимагають високої пропускної здатності та низьких затримок. Прогнозувалось, що кількість точок доступу Wi-Fi 6 повинна була зрости в 13 разів з 2020 по 2023 рік і до кінця 2023 року мала становити 11 % від усіх публічних точок доступу Wi-Fi [6]. Станом на червень 2025 р. Cisco не надає такі дані після 2023 р.

Аналіз безпеки

Кількість порушень і загальний обсяг даних, викритих на одне порушення, продовжує зростати. У всьому світі кількість атак у діапазоні від 100 до 400 Гбіт/с зросла на 776 % порівняно з 2018–2019 роками, а загальна кількість DDoS-атак подвоїлася із 7,9 мільйона 2018 року до 15,4 мільйона для прогнозів 2023 року [6].

1.1.2. Інтернет речей

Наявність правильних даних, які можна перетворити на інформацію, а потім на бізнес-дані, є критичним чинником для успіху. Джерела даних, доступні підприємствам, зростають експоненційно. Дані можуть надходити із різних прикладних систем, наприклад, це можуть бути:

– телеметричні системи моніторингу автомобіля;

– інтелектуальне вимірювання;

– управління запасами та відстеження активів;

– управління флотом і логістика.

Але основним джерелом *Big Data* будуть слугувати під'єднані до мережі різні сенсори, датчики, пристрої під загальним терміном *інтернет речей* (англ. *Internet Of Things, IoT*).

Інтернет речей – концепція мережі взаємозв'язаних фізичних пристроїв, які мають вбудовані сенсори й програмне забезпечення, що дає змогу здійснювати передачу й обмін даними між фізичним світом і комп'ютерними системами в автоматичному режимі за допомогою використання стандартних протоколів зв'язку.

1990 року Джон Ромки, один з творців протоколу TCP/IP, підключив свій тостер до інтернету та змусив його включатися й вимикатися дистанційно. Цей пристрій і став першою в світі «інтернет-річчю». У період з 2008 по 2009 рік (точну дату вказати неможливо), за оцінкою аналітиків корпорації Cisco, кількість пристроїв, підключених до Всесвітньої павутини, перевищило чисельність населення Землі. Термін «інтернет речей» був запропонований 1999 року Кевіном Ештоном, одним з трьох засновників Центру автоматичної ідентифікації Массачусетського університету (Auto-ID Center). Складовою частиною інтернету речей є Індустріальний (або Промисловий) інтернет речей (Industrial Internet of Things, IIoT).

Крім сенсорів, мережа може мати виконавчі пристрої, вбудовані у фізичні об'єкти та пов'язані між собою через дротові чи бездротові мережі. Ці взаємопов'язані пристрої мають можливість зчитування й приведення в дію функцію програмування та

ідентифікації, а також дають змогу вилучити потребу участі людини за рахунок використання інтелектуальних інтерфейсів. Інтернет речей не тільки наділяє сенсорами існуючі речі, він створює середовище нових пов'язаних речей. Усі ці пов'язані речі генерують дані. Це призводить до формування неймовірної кількості даних, які є основою *Big Data*.

1.1.3. Всеохоплюючий інтернет

На сьогодні вже з'явився й новий термін: Internet of Everything, IoE – дослівний переклад «інтернет всього», проте більш вживаним є термін *«всеохоплюючий інтернет»*. Технологічні аналітики та дослідницькі фірми часто розглядають *всеохоплюючий інтернет* як наступний етап *інтернету речей*. Термін був запроваджений, щоб запропонувати більш широке уявлення про підключення речей до інтернету. Багато джерел стверджують, що цей термін придумала компанія Cisco, але сама компанія заперечує цю інформацію. Cisco стверджує, що кілька технологічних компаній, включно із Gartner та Qualcomm, почали використовувати цю комбінацію слів самостійно приблизно тоді, коли й Cisco.

Всеохоплюючий інтернет (далі – IoE) є розширенням технологій інтернету речей IoT насамперед через збільшення кількості самих речей у мережі, а також швидкості мережі й кількості та якості послуг. Власне, це філософія, в якій наступна ера інформаційно-мережевих технологій складається з багатьох різних типів приладів, сенсорів, пристроїв і предметів, підключених до глобальної мережі «Інтернет», а разом вони утворюють екосистему, де все системно взаємопов'язано: речі, люди, дані, бізнес, процеси.

Отже, *всеохоплюючий інтернет* – це більш широке поняття, яке охоплює не лише взаємодію машини з машиною (Machine-to-Machine, M2M), але й взаємодію людей з машинами (Peole-to-Machine, P2M) та взаємодію людей з людьми (People-to-People, P2P) через інформаційно-мережеві технології. Хоча зв'язок M2M завжди важливий, *всеохоплюючий інтернет* робить акцент на інших двох типах зв'язку (рис. 1.3).

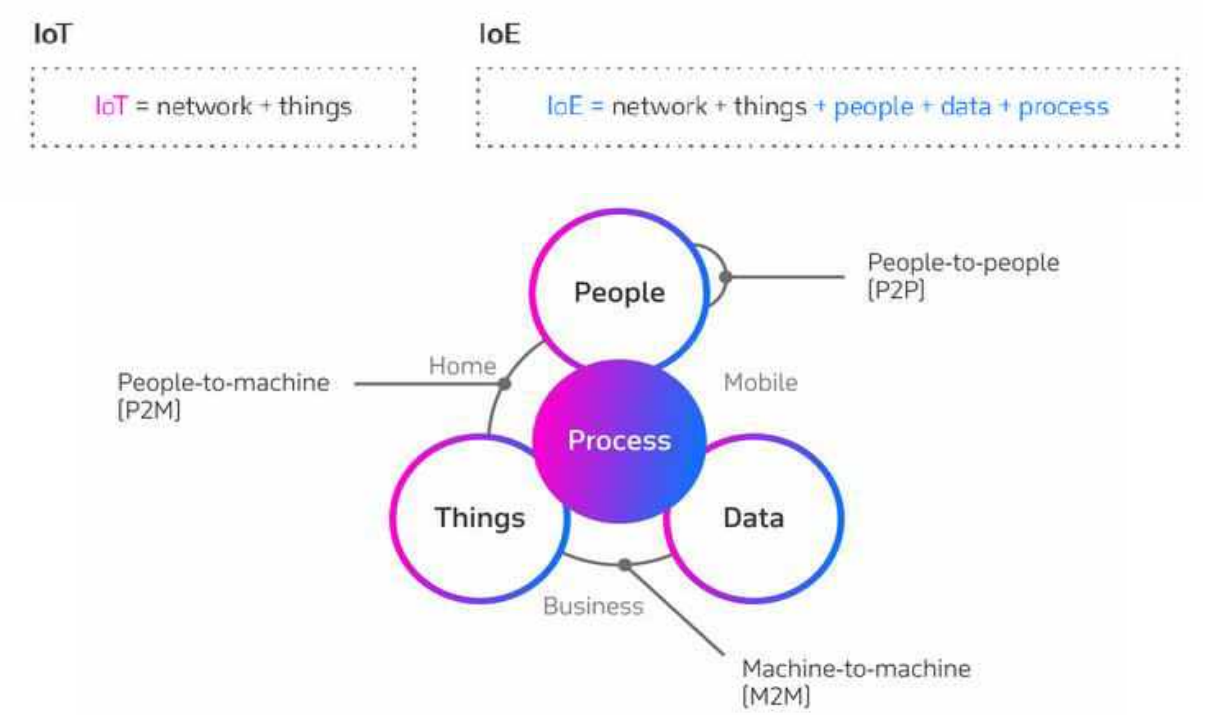


Рисунок 1.3 – Всеохоплюючий інтернет
(Internet of Everything, IoE)

Сучасний інтернет складається з тисяч корпоративних, наукових, урядових і домашніх комп'ютерних мереж. Об'єднання мереж різної архітектури й топології здійснюється за допомогою протоколу IP. Кожному учаснику мережі (або групі учасників) присвоюється IP-адреса постійна або тимчасова (динамічна).

Аналогічно *інтернет речей* як основа *всеохоплюючого інтернету* сьогодні складається із безлічі слабо пов'язаних між собою мереж, кожна з яких вирішує свої завдання. Наприклад, в офісній будівлі може бути розгорнуто відразу кілька мереж: для управління кондиціонерами, системою опалення, освітленням, безпекою тощо. Ці мережі можуть працювати за різними стандартами, об'єднання їх в одну мережу являє собою нетривіальне завдання. Крім того, четверта версія протоколу IP (IPv4) дає змогу використовувати всього лише 4,22 мільярда адрес, через що виникла проблема їхнього вичерпання. Тому на сьогодні паралельно із 32 бітними IP-адресами широко впроваджується 128 бітний протокол IPv6, який дасть змогу вирішити бурхливе зростанням IP-адрес інтернету речей.

Кількість адрес IPv6 відповідає 340 секстильйонам ($3{,}4 \times 10^{38}$) або приблизно по 5×10^{28} адрес на кожну людину на планеті й можливості адресації протоколу IPv6 дадуть змогу практично без обмежень ідентифікувати в мережі будь-яку річ упродовж десятиліть.

Засоби ідентифікації

Кожен об'єкт фізичного світу, який бере участь в інтернеті речей, навіть непідключений до мережі, повинен мати унікальний ідентифікатор. Для автоматичної ідентифікації предметів можуть використовуватися різні вже існуючі системи: радіочастотна, під час використання якої до кожного об'єкта прикріплюється радіочастотна мітка, оптична (штрих-коди, Data Matrix, QR-коди), інфрачервоні мітки тощо. Але для забезпечення унікальності ідентифікаторів різних типів доведеться провести роботу щодо їхньої стандартизації.

Засоби вимірювання

Завдання засобів вимірювання – забезпечити перетворення інформації про зовнішнє середовище в дані, придатні для передачі їх засобам обробки. Це можуть бути як окремі датчики температури, освітленості тощо, так і складні вимірювальні комплекси. Для досягнення автономності засобів вимірювання потрібно забезпечити електроживлення датчиків за рахунок засобів альтернативної енергетики (сонячні батареї та ін.), для мінімізації витрат часу й коштів на підзарядку акумуляторів або заміну батарей.

Засоби передачі даних

Для передачі даних може бути використана будь-яка з наявних технологій. У разі використання бездротових мереж особлива увага приділятиметься підвищенню надійності передачі даних. Під час використання дротових мереж планується активно використовувати технологію передачі даних лініями електропередач.

Засоби обробки даних

Десятки мільярдів пристроїв, під'єднані до інтернету, за прогнозами [3], повинні були згенерувати 2025 року 120 зетабайт даних (120ZB). Це приблизно в 20 разів перевищує кількість оцифрованої інформації в усьому світі станом на 2010-ті роки.

Тому в компанії Microsoft вважають, що ключовим компонентом інтернету речей є не сенсори та датчики й засоби передачі даних, а хмарні системи, які повинні забезпечувати високу пропускну здатність і спроможні швидко реагувати на визначені стани процесів, об'єктів тощо. Впоратися з чималими потоками інформації допоможуть *туманні обчислення*, серверна частина яких буде розміщена ближче до джерел інформації, що будуть не конкурувати з хмарними, а ефективно їх доповнювати.

Виконавчі пристрої

Це пристрої, здатні перетворювати цифрові електричні сигнали, які надходять від інформаційних мереж, у дії. Наприклад, для того щоб через смартфон можна було увімкнути систему опалення в будинку, вона повинна мати відповідний пристрій. Виконавчі пристрої часто конструктивно поєднуються з датчиками.

Перспективи інтернету речей

Відповідно до прогнозів різних організацій і за даними звітів DataReportal («Digital 2025») та Міжнародного союзу електрозв'язку (ITU, «Facts and Figures 2024») на початок 2025 року (відображає дані за 2024 рік):

– кількість користувачів інтернету становила майже 5,5–5,56 мільярда осіб;

– рівень проникнення інтернету сягнув 67–68 % від загальної чисельності населення світу;

– прогнози на кінець 2025 року вказують на подальше зростання до приблизно 5,6–5,7 мільярда користувачів, а рівень проникнення може наблизитися до 69–70 %;

– пристрої та підключення. Загальна кількість підключених пристроїв (включно із IoT / M2M) та пристроїв на душу населення.

За даними Ericsson Mobility Report (листопад 2024) вказувалось, що на кінець 2024 року очікується 17,7 мільярда IoT-з'єднань.

Прогноз на 2025 рік

Statista (дані оновлюються, але прогнози на лютий 2024 року) передбачала, що кількість пристроїв інтернету речей у всьому світі сягне 29,4 мільярда до 2030 року. На 2025 рік ця цифра, відповідно до їхніх графіків зростання, мала б бути в діапазоні приблизно 19–21 мільярда IoT-пристроїв.

IoT Analytics у звіті «IoT Device & Connectivity Market Sizing & Forecast 2024–2030» (опубліковано в березні 2025) прогнозує, що кількість підключених IoT-пристроїв досягне 20,5 мільярдів до кінця 2025 року.

Ericsson Mobility Report (листопад 2024) прогнозує, що до кінця 2030 року кількість IoT-з'єднань сягне 39,9 мільярда. На 2025 рік з огляду на їхні темпи зростання можна очікувати приблизно 20–22 мільярдів IoT-з'єднань.

Домінування споживчого сегмента за кількістю пристроїв

Споживчий сегмент, що охоплює смартфони, планшети, ПК, носимі пристрої, пристрої для «розумного» дому (smart TV, розумні колонки, побутова техніка тощо), традиційно становить більшу частину за кількістю підключених пристроїв. Ця тенденція, ймовірно, збережеться й у 2025 році.

Згідно з деякими звітами (наприклад, Data Insıghts Market або ResearchAndMarkets.com, які аналізують ринок підключених пристроїв) споживчий сегмент продовжує утримувати найбільшу частку ринку.

Швидке зростання бізнес-сегмента (Enterprise / Industrial IoT)

Водночас бізнес-сегмент (включно із промисловим IoT, підключеними автомобілями, рішеннями для «розумних» міст, сільського господарства, охорони здоров'я тощо) демонструє досить швидкі темпи зростання. IoT Analytics у звіті «State of IoT Spring 2025» зазначає, що після деякого сповільнення 2024 року ринок корпоративного IoT (Enterprise IoT) має прискорити зростання у 2025 році з прогнозованим середньорічним темпом зростання (CAGR) 14 % до 2030 року.

Крім того, інтернет речей буде застосовуватися в торгівлі, логістиці, громадському харчуванні, готельному бізнесі, банківській системі.

1.1.4. Відкриті дані як джерело Big Data

Одним із джерел *Big Data* стають не лише пристрої IoT а й *відкриті дані* державних відомств, корпорацій, наукових установ, соціальних мереж, яких стає все більше, що акумулюються на відкритих, вільно доступних ресурсах, і до них є вільний доступ.

Відкриті дані (англ. *Open Data*) – це концепція, яка відображує ідею, що дані, визначені законодавством, мають бути доступні для легкої обробки програмними засобами, тобто бути доступними для читання комп'ютером (*Machine Readable*) та подальшого використання й розповсюдження без жодних обмежень і контролю, зокрема й для комерційного використання.

Відкриті дані є не просто інформацією, а концепцією, тобто системою поглядів, підходів, процесів, які мають одну ідею та мету – вільне використання й розповсюдження даних про діяльність державних органів та органів місцевого самоврядування через мережу «Інтернет».

Згідно із загальноприйнятою в світі концепцією обов'язково безоплатними можуть бути лише ті дані, які перебувають у власності держави («ліцензійно чисті») і якщо вони подаються в первинному необробленому вигляді. Додаткова обробка чи доступ до API досить часто лімітуються або коштують грошей. Крім того, хто завгодно може (з обов'язковим посиланням на джерело) використати дані в комерційний спосіб, створити на базі них власну програму чи обробити та надати нову цінність (розкласифікувати, встановити зв'язки тощо). Безумовно, всі витрати на додаткову роботу редакторів, програмістів та інше оплачує замовник (або кінцевий користувач).

Відкриті дані дають змогу повторно й необмежено використовувати інформацію, поєднувати її між собою, зменшити або вилучити зайві витрати на дублювання та опрацювання великих масивів даних, реєстрів, довідників, баз даних тощо, створених у різних органах влади.

Закон України від 19.02.2015 № 2171 «Про внесення змін до деяких законів України щодо доступу до публічної інформації у формі відкритих даних» (ухвалений 09.04.2015) визначає такий

термін, як «*Публічна інформація у формі відкритих даних»*. Це є публічною інформацією у форматі, що дає змогу на її автоматизоване оброблення електронними засобами, вільний та безоплатний доступ до неї, а також на її подальше використання.

У Законі використовуються поняття «форма» та «формат», що не є тотожними й важливими для розуміння можливості їхнього використання для аналізу. Для розуміння форматів відкритих даних звернемося до відомої класифікації «5 зірок Open Data» [4], де якість даних і рівень відкритості визначається кількістю зірок від 1 до 5, чим більше – тим краще. Відкритість даних залежить від способів доступу, форматів і кількості додаткових дій, які потрібні для отримання кінцевої інформації, її обробки та збереження у власному сховищі або базі даних.

Одну зірку (*) отримує будь-яка інформація, вільно доступна через інтернет в будь-якому форматі. Під цю класифікацію підпадає файл у форматі PDF або інша (сканована) копія документа, який вказує на пряме посилання на офіційному сайті державного органу. Якщо цей файл можна відкрити на власному екрані, прочитати, роздрукувати та отримати звідти потрібну інформацію, то це відкриті дані з однією зіркою.

Дві зірки (**) отримує структурована інформація, яку можна обробляти автоматично, наприклад у форматах для веббраузерів чи офісних програм (відкриті формати – TXT, HTML, RSS; пропрієтарні формати, Excel – XLS, Word – DOC, RTF). Якщо дані перебувають у тілі вихідної вебсторінки, але не мають чіткої структури, містять зайві елементи оформлення, навігації, потребують додаткових дій – спеціального розбору (*парсингу*), то вони вважаються з двома зірками.

Три зірки (***) може отримати інформація, представлена у відомих, добре описаних відкритих структурованих форматах (наприклад, CSV, JSON, XML, YAML), та автоматизована її обробка не потребує від користувача особливих ліцензій і додаткових плат. До відкритих форматів також належать пов’язані дані (HTML+RDFa) з узгодженою розміткою елементів в атрибутах (*сніппет* для пошукових систем) або текстові файли таблиць, поля яких розділені табуляцією, комами, крапками з комою або іншими символами.

Чотири зірки (****) надаються в разі, якщо можна отримати первинні необроблені набори відкритих даних як файли (довідники, списки, таблиці у відкритому форматі, зліпок бази даних, архів документів тощо) або фільтровані дані в запиті до API за вказаними параметрами.

API (*Application Programming Interface*, *API*) – *прикладний програмний інтерфейс* (інтерфейс програмування застосунків, інтерфейс прикладного програмування) – набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

Це дає змогу отримувати тільки потрібну інформацію, актуальну на момент запиту, заощаджує ресурси та час користувача. Безумовно, API має бути описаний так само, як і формати даних, а доступ до нього може бути анонімний без обмежень або з реєстрацією за вказаним ідентифікатором, лімітами на кількість одночасних запитів тощо.

Останній рівень – *п'ять зірок (*****)* – надається інформації, коли набори відкритих даних пов'язані між собою (мають спільні довідники, класифікатори, ідентифікатори, посилання між документами та іншими елементами тощо) і являють семантичну мережу, що постійно оновлюється й змінюється відповідно до сучасних запитів.

На сьогодні в Україні вже працює значна кількість ресурсів відкритих даних, не враховуючи ресурси міністерств та відомств, зокрема:

– державний портал відкритих даних https://data.gov.ua ;

– вебсервіс «Дія» https://id.diia.gov.ua/ ;

– державний портал бюджету для громадян https://openbudget.gov.ua/ ;

– проєкт «Суд на долоні» https://conp.com.ua/about ;

– інформаційно-аналітична система «Прозорий бюджет» https://edata.gov.ua/ ;

– доходи бюджету України http://old.cost.ua/budget/revenue/#1;

– єдиний вебпортал використання публічних коштів https://spending.gov.ua/new/ ;

– тощо.

Усі вищезазначені ресурси зазвичай мають API інтерфейси, які дають змогу здійснювати міжсерверну взаємодію, розробку прикладних додатків комерційними структурами, громадськими організаціями та ін.

Одним із аспектів інтеграції України до Європейського Союзу є формування екосистеми відкритих даних (далі – ВД). З огляду на те, що Україна 2024 року в загальноєвропейському рейтингу *Зрілість відкритих даних (Open Data Maturity)* зайняла третю позицію після Франції та Польщі, (2022-го, відповідно, другу позицію після Франції), цей напрям цифрової трансформації нашої держави є важливим аспектом і у формуванні великих даних. Ці дані будуть використовуватись для систем штучного інтелекту, машинного навчання, інтелектуальних систем управління під час ухвалення тактичних і стратегічних рішень в економіці та бізнесі. На рис. 1.4 показано графік, що відображає стабільну позицію України в загальноєвропейське середовище відкритих даних, починаючи лише з 2021 року.

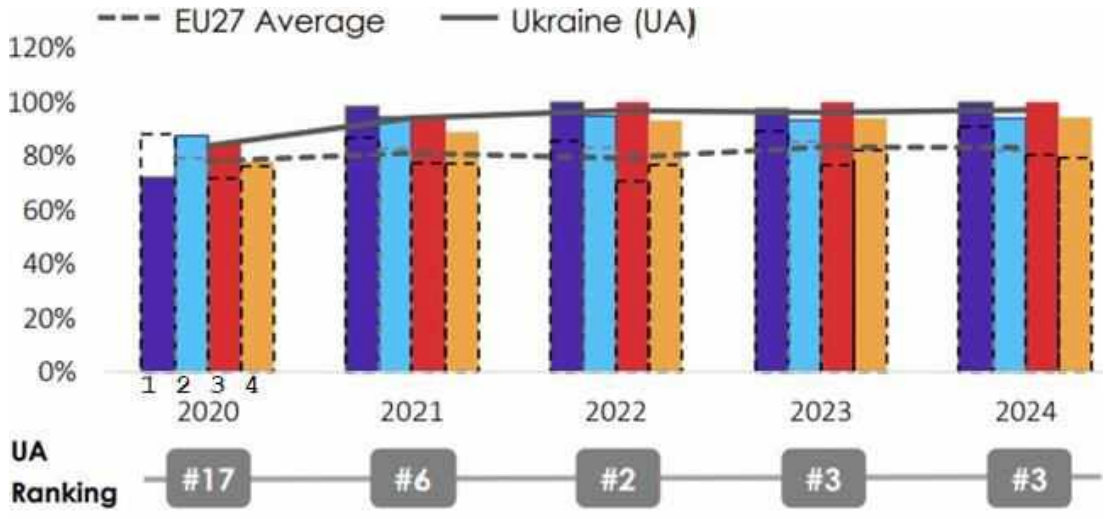


Рисунок 1.4 – Зрілість відкритих даних України у 2020–2024 роках

(1 – політика ВД; 2 – портал ВД; 3 – вплив ВД; 4 – якість ВД)

Методика оцінювання зрілості ВД використовує чотири виміри:

Політика (Policy): зосередження уваги на політиці та стратегіях країн щодо відкритих даних.

Вплив (Impact): аналіз діяльності з моніторингу й вимірювання повторного використання відкритих даних та їхнього впливу.

Портал (Portal): оцінювання функцій і можливостей порталу, які дають змогу користувачам отримати доступ до відкритих даних через національний портал і підтримують взаємодію в рамках відкритих даних у національному порталі та підтримувати взаємодію всередині спільноти відкритих даних.

Якість (Quality): зосередження на механізмах, що забезпечують якість (мету) даних.

Особливістю методології формування звітів «Зрілість відкритих даних» є врахування положення Директиви про відкриті дані [7] щодо впровадженням наборів даних високої цінності.

Високоцінні набори даних – це набори даних з високим потенційним економічним і соціальним впливом. Директива про відкриті дані заохочує держави-члени ЄС сприяти повторному використанню даних державного сектору, приділяючи особливу увагу публікації таких категорій даних: геопросторові, спостереження за Землею та навколишнім середовищем, метеорологічні, статистичні, дані про компанії та власників компаній, а також дані про їх мобільність.

На сьогодні 96 % держав-членів ЄС уже працюють над визначенням високоцінних даних, які мають бути пріоритетними для публікації. 85 % країн ЄС-27 готуються до моніторингу та вимірювання рівня повторного використання наборів даних високої цінності, і всі вони мають на меті просувати або вже просувають набори даних високої цінності. 63 % країн ЄС готуються забезпечити інтероперабельність високоцінних наборів даних поряд з наявними наборами даних з інших країн.

Важливо, що, незважаючи на війну, Україні вдається тримати високий рівень зрілості своїх ВД і досягти значного успіху в політичному вимірі. Потенціал ВД використовується країнами Європи для реагування на війну, зокрема деякі країни використали ВД для моніторингу рівня використання енергії та для сприяння інтеграції українських біженців на своєму ринку праці.

Питання для самоконтролю

1. Поняття великих даних, відмінності від інших.
2. Опишіть коротко характеристики та значення даних в IoT.
3. У чому полягає цінність даних?
4. Які існують тенденції зростання даних (подайте в основних показниках)?
5. Як великі дані впливають на повсякденне життя?
6. Опишіть основні положення концепції великих даних.
7. Вкажіть основні джерела великих даних.
8. Дайте порівняльну характеристику відкритих даних і приватних даних.
9. У чому полягає відмінність структурованих даних і неструктурованих даних?
10. Що означають терміни «дані в спокої» й «дані в русі»?
11. Опишіть підходи до управління даними IoT.
12. Опишіть основні технології управління даними.
13. Опишіть показники й характеристики відкритих даних.
14. Опишіть поняття інтернету речей та всеохопного інтернету.

1.2. Формати даних, підготовка даних для аналізу

Вагоме значення для накопичення й оброблення відкритих даних, які із зростанням їхніх обсягів стають обов'язково великими даними, має розуміння форматів, у яких вони подаються на інтернет-ресурси. Розглянемо формати детальніше.

1.2.1. Формати даних XLS / XLSX

Спочатку формат XLSX створювався як заміна раніше бінарного формату документів (.xls), який використовували додатки Microsoft Office. Але 2006 року формат Office Open XML був оголошений вільним і відкритим форматом.

Файл XLSX – електронна таблиця, створена в Microsoft Excel. Дані в документі зберігаються в комірках, кожна з яких має певну адресу (колонки нумеруються англійськими літерами, рядки – цифрами, наприклад: A1 – ліва верхня клітинка). Кожна клітинка може містити як фіксовані дані, так і формули, часто пов'язані з даними в інших осередках. Формат XLSX дає змогу користувачеві змінювати форматування тексту: його шрифт, колір, вирівнювання (в комірці) та інші параметри. Крім того, документ може містити зображення, а також діаграми, побудовані на основі даних у певних комірках.

Програми, які дають змогу працювати з XLSX-файлами: File Viewer Lite, Microsoft Excel, Microsoft Excel Viewer, Microsoft Works, Corel WordPerfect Office X7, Apache OpenOffice, Kingsoft Spreadsheets, Nuance OmniPage Ultimate, Gnumeric, LibreOffice.

1.2.2. Графічні формати JPEG, PNG, TIFF

JPEG (Joint Photographic Experts Group)

*JPEG (*англ. *Joint Photographic Experts Group)*, за назвою організації-розробника), – один з популярних графічних форматів, застосовуваний для зберігання фотозображень і схожих до них зображень. Файли даних JPEG зазвичай мають розширення *.jpg, .jfif, .jpe* або .jpeg. Однак з них .jpg є найпопулярнішим на всіх платформах.

Алгоритм JPEG дає змогу стискати зображення як із втратами, так і без втрат (режим стиснення *lossless* JPEG). Підтримуються зображення з лінійним розміром не більше 65 535 × 65 535 пікселів.

Алгоритм JPEG найбільшою мірою придатний для стиснення фотографій і картин, що містять реалістичні сцени з плавними переходами яскравості й кольору. Найбільшого поширення JPEG отримав у цифровій фотографії та для зберігання й передачі зображень з використанням мережі «Інтернет». Формат JPEG у режимі стиснення із втратами малопридатний для стиснення креслень, текстової та знакової графіки, де різкий контраст між сусідніми пікселями призводить до появи помітних артефактів. Такі зображення доцільно зберігати у форматах без втрат, як-от JPEG-LS, TIFF, GIF, PNG, або використовувати режим стиснення Lossless JPEG.

JPEG не підходить для стиснення зображень багатоетапної обробки, оскільки спотворення в зображенні будуть вноситися щоразу під час збереження проміжних результатів обробки. JPEG не повинен використовуватись і в тому разі, коли недопустимі навіть мінімальні втрати, наприклад під час стиснення астрономічних або медичних зображень.

PNG (Prtable Network Graphics)

*PNG (*англ. *Portable Network Nraphics)* – растровий формат зберігання графічної інформації, що використовує стиснення без втрат за алгоритмом *Deflate.* Формат PNG розроблений для заміни форматів GIF та TIFF. Він позиціонується передусім для використання в інтернеті й редагування графіки. Формат PNG зберігає графічну інформацію в стислому вигляді, причому це стиснення графічної інформації проводиться без втрат.

PNG підтримує три основних типи растрових зображень:

1. Півтонування (з глибиною кольору 16 біт).

2. Кольорові індексовані зображення (палітра 8 біт для кольору глибиною 24 біт).

3. Повнокольорове зображення (з глибиною кольору 48 біт).

Формат має такі основні переваги:

– практично необмежену кількість кольорів у зображенні;

– опціональну підтримку альфа-каналу;

– можливість гамма-корекції;
– двовимірну черезстрокову розгортку;
– можливість розширення формату для користувача блоками (на цьому заснований, зокрема, APNG).

Будь-яке збережене зображення PNG може бути прочитано в будь-якому іншому додатку, що підтримує PNG.

PNG є хорошим форматом для редагування зображень, навіть для зберігання проміжних стадій редагування, оскільки відновлення й повторне збереження зображення здійснюється без втрат в якості.

TIFF (Tagged Image File Format)

*TIFF (*англ. *Tagged Image File Format)* – формат зберігання растрових графічних зображень. TIFF став популярним форматом для зберігання зображень з великою глибиною кольору. Він використовується під час сканування, відправлення факсів, розпізнавання тексту, в поліграфії широко підтримується графічними додатками. TIFF був обраний як основний графічний формат операційної системи NeXTSTEP і з неї підтримка цього формату перейшла в Mac OS X. Спочатку формат підтримував стиснення без втрат, згодом був доповнений для підтримки стиснення із втратами у форматі JPEG. Файли формату TIFF зазвичай мають розширення .tiff або .tif.

Структура формату гнучка й дає змогу зберігати зображення в режимі кольорів з палітрою, а також у різних колірних просторах:

– бінарному (двокольоровому, іноді неправильно званому чорно-білим);
– напівтоновому;
– з індексованої палітрою ;
– RGB;
– CMYK;
– YCbCr ;
– CIE Lab.

Підтримуються режими 8, 16, 32 і 64 біт на канал у разі цілочисельного, а також 32 і 64 біт на канал під час подання значення пікселя числами з плаваючою комою.

1.2.3. Текстові формати даних TXT, CSV, TSV, Markdown, PDF

TXT (Textfile)

TXT (Text File) – це формат .txt, що містить текстові дані, які зазвичай організовані як рядки; слугують основою для багатьох більш спеціалізованих форматів, як-от CHM, PHP, XML, CSV. Може бути переглянутий будь-яким текстовим редактором.

Текстові файли розбиваються на кілька рядків. На сучасних платформах розбивка на рядки кодується керуючим символом переведення рядка (\n), а іноді послідовністю двох керуючих символів. Взагалі текстові файли можуть містити друковані символи, як-от букви, цифри та розділові знаки й деяку кількість керуючих символів, як-от знаки табуляції та переведення рядка.

Файли TXT містять досить мало елементів форматування, однак дають змогу зіставляти прийняті набори елементів форматування із системним терміналом або простим текстовим редактором. Файли TXT універсальні за своєю природою, тому що відкрити їх зможе будь-який текстовий редактор. Водночас ці файли може використовувати Unicode, що дає змогу полегшити користувачам їхнє застосування, які пишуть різними мовами. Файли TXT, що використовують текст тільки в кодуванні ASCII, можна переносити на різні комп'ютери та пристрої на ОС Unix, Mac і Windows.

CSV (Comma Separated Values), TSV (Tab-Separated Values)

CSV (Comma Separated Values) – формат простого тексту з розділювачами, які позначають структуру таблиці. Відповідно, є розділювач стовпців і рядків таблиці. Найпоширеніший формат .csv, де стандартне налаштування:

– колонки розділені комами;

– рядки розділені знаком нового рядка \n;

– значення колонок у подвійних лапках (щоб коми всередині тексту не руйнували структуру файлу);

– нецілі числа розділені крапками: 12.345;

– кодування UTF-8.

TSV (Tab-Separated Values) поширений формат .tsv, в якому колонки розділені знаком табуляції \t. Цей формат зручний для текстових даних з комами.

Під час збереження файлу у форматі CSV Excel за стандартним налаштуванням зберігає файл .csv у кодуванні вашої операційної системи. Це, зокрема, впливає на десятковий розділювач та розділювач стовпчиків. Для локалізованих українською мовою версій MS Office цим кодуванням є *Windows-1251,* як розділювач, виступає крапка з комою «;», а як десятковий розділювач – кома. Часто Excel спотворює дати або неправильно читає числа з десятковими значеннями. Краще скористатись *Libre Office Calc* або *Google Таблицями*, щоб зберегти файл у загальноприйнятому форматі .csv у кодуванні *UTF-8*. У такому форматі коми розділяють стовпці, текстові значення в подвійних лапках, десятковий розділювач – крапка. Конвертувати таблицю з Excel у .csv можна й за допомогою онлайн-конвертерів (наприклад, https://convertio.co/xls-csv/) та ін.

Для того щоб дізнатись кодову сторінку, встановлену на ПЕОМ, можна скористатись командою *chcp: Start/Права-клавіша-миші/Run/Open/cmd*. У командному рядку для прогляду та зміни встановленої кодової сторінки використовуємо команду *chcp* (рис. 1.5).

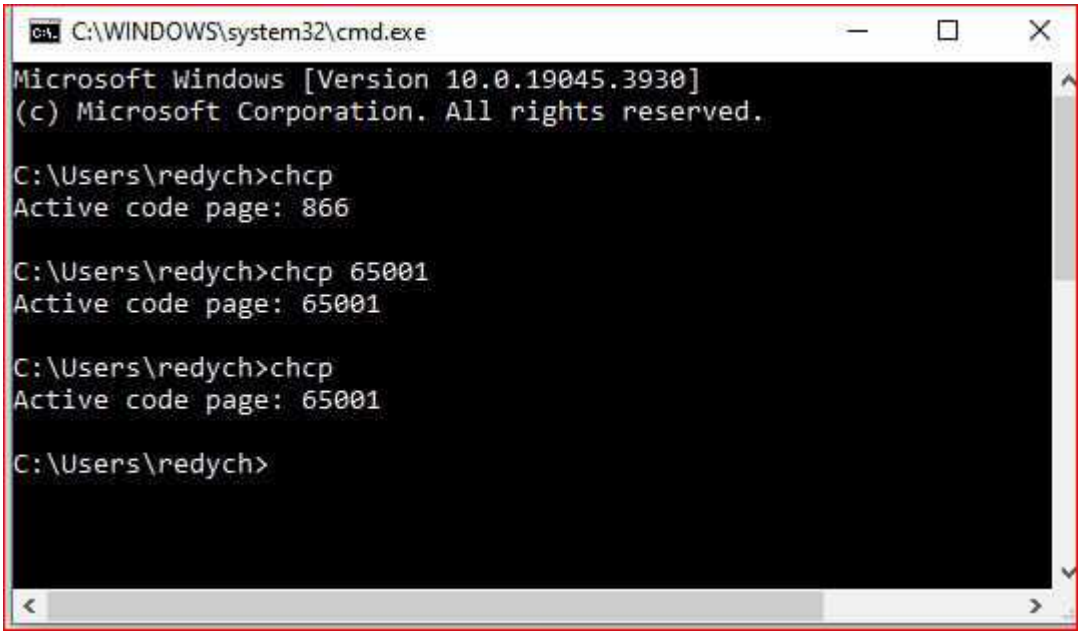


Рисунок 1.5 – Використання команди кодової сторінки *chcp*

Отже, для формування в операційній системі Windows у пакеті Excel коректних даних у форматі CSV потрібно:

– встановити кодову сторінку операційної системи UTF-8;

– встановити знаком, що розділяє десяткові значення, – крапку;

– встановити розділювачем десяткових значень числа – кому.

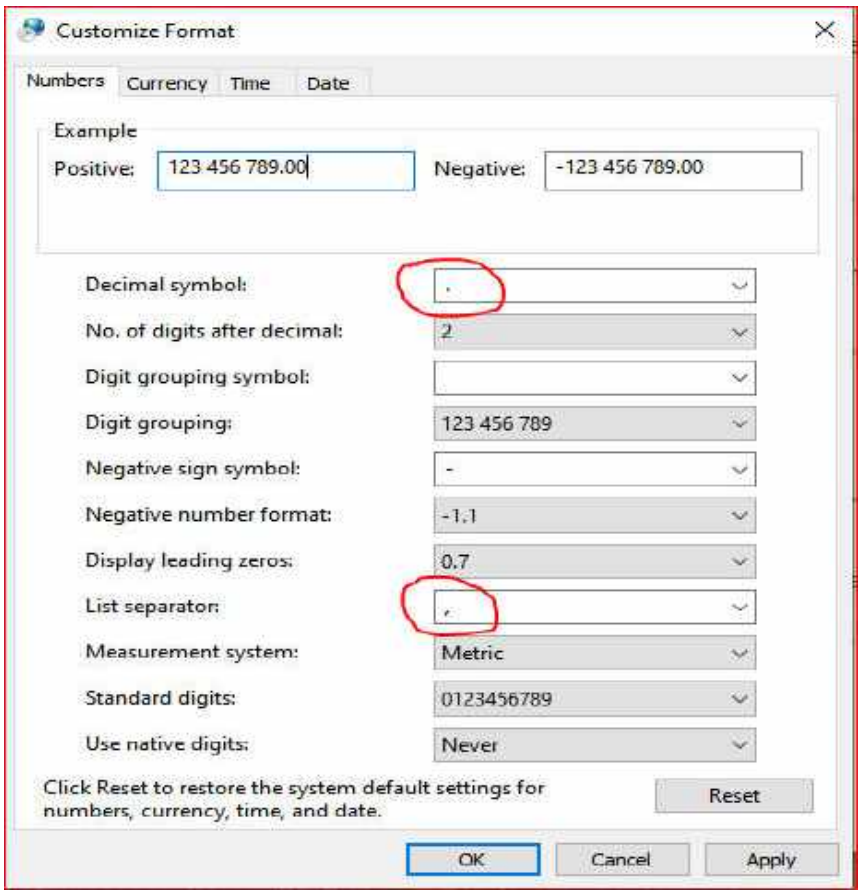


Рисунок 1.6 – Налаштуваннях операційної системи для отримання коректних даних у форматі CSV

Для встановлення кодової сторінку *UTF-8* вводимо в командному рядку (рис. 1.5) *chcp 65001* (встановити кодову сторінку *UTF-8*) і змінюємо в налаштуваннях операційної системи (*«Control Panel» / «Region» / «Additional setting»)* параметри *Decimal symbol* на крапку та *List separator* на кому (рис. 1.6).

Для довідки основні кодові сторінки:

866 – DOS кодування (OEM);

1251 – ANSI;

65001 – UTF-8;

1201 – UTF-16 LE (Big endian), прямий порядок байтів;

1200 – UTF-16 LE (Little endian), відповідно, «справа наліво».

Порядок від старшого до молодшого (англ. *Big-endian* – з великого кінця): «зліва направо», наприклад число сто двадцять три буде записано в такому порядку, як 123. У цьому самому порядку прийнято записувати байти в технічній та навчальній літературі, якщо інший порядок явно не позначений.

Markdown

Markdown – полегшена мова розмітки, створена з метою написання максимально читабельного та зручного для редагування тексту, але придатного для перетворення в мови для публікацій (HTML, Rich Text та ін.).

Отже, Markdown – це дві речі:

1) простий синтаксис форматування тексту;

2) програмний інструмент, написаний на Perl, який перетворює зазвичай форматований текст у HTML.

Першочергова мета розробки для форматування синтаксису Markdown – зробити його максимально читабельним, наскільки це можливо. Ідея полягає в тому, що Markdown-форматований документ повинен бути опублікований як звичайний текст, використовуючи теги та інструкції форматування.

Спочатку формат створювався як заміна раніше бінарного формату документів, який використовували додатки Microsoft Office аж до версії Office 2003 включно. 2006 року формат Office Open XML був оголошений вільним і відкритим форматом Ecma International. Він є форматом за стандартним налаштуванням для додатків Microsoft Office 2007 та пізніших.

Файл з розширенням .DOCX – документ, створений Microsoft Word, програмою обробки тексту; містить текст документа, зображення, форматування, стилі, намальовані об’єкти та інші параметри документа.

Файли DOCX створюються за допомогою відкритого формату XML, в якому зберігаються документи як збори окремих файлів і папок у стислому пакеті. DOCX-файли містять XML-файли і три папки *docProps, Word і _rels,* які охоплюють властивості документа, зміст і відносини між файлами.

OpenDocument format (LibreOffice / OpenOffice)

Формат *OpenDocument* – це формат для текстових, табличних документів і презентацій, що був створений з метою підвищення інтероперабельності між різними офісними застосунками (програмами, офісними пакетами). Результатом зусиль зі стандартизації стало створення двох конкуруючих форматів – Office Open XML (Microsoft) та OpenDocument (Sun Microsystems).

Файли формату OpenDocument мають розширення ODT (OpenDocument Text), ODS (OpenDocument Spreadsheet). Його підтримують такі пакети програм:

– Apache OpenOffice;

– LibreOffice;

– MS Office (2013 – тільки читання, 2016 – читання та редагування);

– Google Docs (тільки імпорт / експорт).

PDF (Portable Document Format)

PDF (Portable Document Format) – формат переносного документа – це формат файлу, який використовується для надійного відображення й обміну документами, незалежно від програмного й апаратного забезпечення або операційної системи. Сьогодні формат PDF, винайдений компанією Adobe, є відкритим стандартом, підтримуваним Міжнародною організацією зі стандартизації (ISO). Файли PDF можуть містити посилання й кнопки, поля форм, аудіо- та відеоелементи, бізнес-логіку. Вони також підтримують можливість електронного підпису й можуть переглядатися за допомогою безкоштовного програмного забезпечення *Acrobat Reader DC*.

Характеристики формату:

– мають такий самий вигляд, як вихідні документи, незалежно від використовуваної платформи або пристрою;

– відповідають стандартам ISO32000 з обміну електронними документами, включно із спеціальними стандартами для PDF;

– можуть бути підписані електронно будь-яким користувачем за допомогою безкоштовного програмного забезпечення Adobe Acrobat Reader DC або мобільного застосування Acrobat DC;

– можуть захищатися паролем для запобігання копіювання та редагування;

– можуть виправлятися для видалення конфіденційної інформації без можливості відновлення;

– зберігають усю інформацію вихідного файлу, охоплюючи текст, зображення, аудіо, 3D-карти та багато іншого, об’єднану в одному файлі;

– забезпечують зручний пошук, включно із пошуком щодо сканованого тексту, який був перетворений за допомогою технології оптичного розпізнавання символів (OCR);

– працюють з допоміжними технологіями, які роблять файли PDF доступнішими для людей з обмеженими можливостями, як-от слабкий зір або сліпота.

1.2.4. Формати даних JSON, XML, RDF

JSON (JavaScript Object Notation)

JSON (англ. *JavaScript Object Notation*) – текстовий відкритий формат, оснований на *Javascript*, представлений і призначений для обміну даними в інтернеті між сервером і клієнтом або сервером і сервером. Хоча він позиціонується, як незалежний від системи й мови програмування, частіше за все використовується за допомогою програм на Javascript, але, як і інші текстові формати, легко читається людиною.

Найбільшу популярність JSON набув після створення інтерактивних вебсторінок, дані до яких через API передавались під час взаємодії користувача з елементами інтерфейсу (технологія AJAX). За рахунок своєї лаконічності, на відміну від XML, простоті й швидкості використання саме в програмах на Javascript, широким можливостям в серіалізації даних – рекурсивного перетворення в текстовий вигляд складних об’єктів, формат активно використовується для формування «на льоту» та передачі структур даних в інтернет у різних інформаційних системах і сервісах.

Синтаксис JSON являє собою текст як програмний код *Javascript*, який описує кілька варіантів структур:

1) масив у квадратних дужках (значення через кому);

2) іменований масив (чи об’єкт) у фігурних дужках (пари ключ-значення, розділені двома крапками, через кому);

3) просте значення (число, рядок, true, false, null), функція.

Для перетворення формату JSON в об’єкт у пам’яті (десеріалізації) в Javascript використовується функція eval(), яка виконує завантажений у текстовий рядок програмний код.

Якщо для обробки використовувати тільки мову Javascript, виникають як мінімум дві проблеми. По-перше, щоб обробити дані, потрібно мати приблизно в два рази більше вільної пам'яті, ніж файл (спочатку завантажується повний текст JSON, а після його виконання дані об'єкта завантажуються повторно як відповідні структури).

По-друге, можливість передачі програмного коду, який автоматично стартує, дає змову зловмисникам отримати несанкціонований доступ до системи. Цю вразливість також можна уникнути, якщо спочатку перевірити й очистити текст JSON від заборонених об'єктів, а дані перетворити на коректні значення.

Через нераціональне використання пам'яті й ресурсів у JSON краще працювати з об'єктами будь-якої складності, але не надто великих розмірів, у швидкому режимі «запит-відповідь».

Приклад формату JSON:

```
 {
"firstName": "Дмитро", "lastName": "Маковський",
"address": {
"streetAddress": "Ділова вул., 24, кв. 205",
"city": "Київ",
"postalCode": 03150
},
"phoneNumbers": [ "380 207-1730",
"380 207-1736"
]
}
```

XML (eXtensible Markup Language)

XML (англ. *eXtensible Markup Language* – мова розмітки, що розширюється) – мабуть, найстаріший текстовий відкритий формат, створений 1994 року та рекомендований Консорціумом Всесвітньої павутини (W3C) як основний для обміну інформацією в інтернеті. Гіпертекстова розмітка (HTML) – це один із різновидів XML. Разом з таблицями каскадних стилів CSS, які формують зовнішній вигляд документів, вони є тими основними форматами, що обумовлюють розвиток технологій.

Насправді XML – універсальний зручний для людини формат для збереження, передачі структурованих даних і їхньої автоматичної обробки у формі машиночитаних документів. З нього

починали розвиток інші відкриті формати, на ньому вдосконалювались підходи обміну даними. Сама мова розмітки XML є похідною від ще старішого та більш складного формату *SGML* (*Standard Generalized Markup Language* – стандартна узагальнена мова розмітки), стандартизованого за ISO 8879:1986.

Перевагами XML є простота та гнучкість розмітки, яка не вимагає формальних, фіксованих назв тегів чи параметрів, і будь-який розробник може доповнювати й змінювати формат, створювати власну схему XSD (XML Schema Definition). Фактично це мова, яка описує сама себе та будь-які за розміром і складністю структури даних.

Безумовно, в цей формат можна конвертувати інші формати (наприклад, XLS). Крім того, формат не залежіть від операційної системи чи бази даних. Для простого перегляду або редагування достатньо текстового редактора. Але є й недоліки – дані в форматі XML займають значно більше місце, ніж це потрібно, за рахунок повторення тегів і відступів, а парсинг складніший, ніж CSV чи JSON.

За довгий час існування XML на його базі було розроблено багато форматів і стандартів зі схожим синтаксисом (приклади стандартних схем на http://schema.org).

Зазвичай цю групу форматів називають загальною назвою – XML, тому що вони мають єдині механізми опису схем XSD, перевірки правильності даних (валідації), перегляд *DOM (Document Object Model)*, доступу до елементів XPath та трансформації для автоматичного конвертування в інші схеми чи формати (наприклад, альтернативні JSON та YAML) за допомогою мови перетворення *XSLT (eXtensible Stylesheet Language Transformations)*.

Використання формату XML (а саме LegalXML) як відкритого стандарту нормативно-правового документа – це сучасний спосіб забезпечити обмін інформацією (документами, картками, довідниками тощо) між інформаційними системами або в межах однієї системи під час підготовки (опрацювання) документів.

В одному файлі XML у текстовому вигляді, крім основних даних і тексту електронного нормативного документа, можна розміщати метадані (характеристики, реквізити, опис, класифікацію

тощо), вкладені файли (картинки, стилі тексту, таблиці Excel, документи Word та ін.), відповідні структури чи довідники. Це дає змогу зручно не тільки зберігати, передавати, обробляти документ, отримувати PDF-версію для друку, формувати зміст чи робити посилання на конкретну главу, статтю, пункт, підпункт тощо, а й автоматизовано вносити зміни, підготовлені у вигляді, що дає змогу їхній програмній обробці.

Приклад даних у форматі XML:

```
<person firstName="Степан" lastName="Іванченко">
<address streetAddress="Московское ш., 101,
кв.101" city="Дніпро"
postalCode="101101" />
<phoneNumbers>
<phoneNumber>812 123-1234</phoneNumber>
<phoneNumber>916 123-4567</phoneNumber>
</phoneNumbers>
</person>
```

RDF (Resource Description Framework)

RDF (Resource Description Framework) – розроблена консорціумом Всесвітньої павутини модель для представлення даних, особливо метаданих. RDF представляє твердження про ресурси у вигляді, придатному для машинної обробки. RDF є частиною концепції семантичної павутини. Ресурсом у RDF може бути будь-яка сутність: як інформаційна (наприклад, вебсайт або зображення), так і неінформаційна (наприклад, людина, місто або якесь абстрактне поняття). Твердження, що висловлюється про ресурс, має вигляд «суб'єкт – предикат – об'єкт» і називається триплетом. Затвердження «небо блакитного кольору» в RDF-термінології можна представити так: суб'єкт – «небо», предикат – «має колір», об'єкт – «блакитний». Для позначення суб'єктів, відносин і об'єктів у RDF використовуються URI.

RDF сам собою є не форматом файлу, а лише абстрактною моделлю даних, тобто описує пропоновану структуру, способи обробки та інтерпретації даних. Для зберігання й передачі інформації, покладеної в модель RDF, існує ряд форматів запису.

Для обробки RDF-даних пропонується реалізувати мови запитів: SPARQL (стандарт W3C), RQL, RDQL.

Для запису й передачі RDF використовується кілька форматів, зокрема:

– RDF / XML – запис як XML-документ;

– RDF / JSON – запис як JSON-дані;

– RDFa – запис всередині атрибутів довільного HTML або XHTML-документа;

– N-Triples, Turtle, N3 – компактні форми запису тверджень.

1.2.5. Формати геопросторових даних

Формати GIS

GIS – файли геопросторових даних (геоінформаційних систем, ГІС) можуть містити таку інформацію:

– просторова або геометрична інформація, яка забезпечує положення й форми конкретних географічних об’єктів;

– інформація про атрибути, яка забезпечує описову інформацію (числову, текстову, логічну) про кожний об’єкт;

– інформація про відображення, яке може описувати спосіб оформлення об’єкта.

Деякі цифрові карти не містять усі три типи інформації. Наприклад, растрові карти зазвичай не містять інформацію про атрибути. Зі свого боку багато джерел векторних даних не охоплюють інформацію про оформлення.

Просторова або геометрична інформація

Географічна інформація описує в цифровій формі положення та форму кожного зображуваного об’єкта. На векторній карті властива об’єкту позиція зазвичай виражається як набір координат x, y або x, y, z з використанням системи координат, визначеної для набору геоданих. Більшість векторних географічних інформаційних систем підтримують такі основні геометричні об’єкти:

– крапка: одна пара координат, 0-вимірний об’єкт;

– лінія: дві або більше точок у певній послідовності, одновимірний об’єкт;

– полігон: площа, обмежена лінією, двовимірний об’єкт;

– тіло: обсяг, обмежений гранями, тривимірний об’єкт.

Векторні дані характеризуються топологією, яка задає правила просторової взаємодії між векторними об'єктами. Залежно від масштабу даних один і той самий об'єкт може зберігатися як точка, лінія, полігон або тіло.

На растровій карті властива об'єкта позиція визначається комірками з відповідними атрибутами. Кожна комірка характеризується розміром і значенням атрибутів, яке може бути відображене як значення кольорової моделі (супутниковий знімок) або значення атрибуту (GRID), на основі яких растр може бути класифікований за кількісними або якісними характеристиками. Деякі системи також підтримують більш складні об'єкти, як-от регіони, кола, еліпси, дуги та криві.

Інформація про атрибути

Атрибути описують специфічні характеристики об'єктів, які не є графічними елементами. Наприклад, атрибутом, що характеризує дорогу, може бути її назва або дата, коли її побудували. Зазвичай атрибути зберігаються окремо від графічної частини карти за допомогою геореляційних структур даних. Атрибути є частиною опису векторних карт і рідко застосовуються разом із растровими зображеннями.

Інформація про відображення

Інформацію, що відображається, може описувати цифрова карта або накреслена (план). Загальна інформація на карті відображається за допомогою різних кольорів: озера, моря, вершини, гори, рівнин; ширини й типів ліній: назви доріг, вулиць, нанесення висотної поясності на планах тощо.

Інформація про відображення задає графічні стилі, за допомогою яких відображаються дані на карті. Основними графічними стилями є графічні перемінні, до яких належить колір, товщина, форма, орієнтація, символізація, підписи. Разом графічні перемінні утворюють способи умовних позначень на карті географічних об'єктів і формують легенду карти.

GIS-формати створені для стандартного кодування географічної інформації у файл. Файли цього формату створюються державними агентствами з картографування, комерційними організаціями або розробниками програмного забезпечення.

Ці формати є розповсюдженими, але існує програмне забезпечення, що не підтримує файли цього формату. У таких випадках шукають способи перетворення GIS-файлу або використовують інший файл. Майже кожен GIS має свій власний внутрішній формат файлу. Ці формати призначені для оптимального використання розробником і часто є запатентованими. Вони не призначені для використання поза системою, що їх розробляла.

Векторні формати

Багато додатків GIS засновані на векторній технології через її широке застосування. Вони є складними, тому що інформацію можливо описати багатьма способами: для зберігання координат, атрибутів, зв'язків, структур баз даних і відображення інформації.

Найбільш поширеним форматом гепросторових даних є формат **Shapefile (shp)**. Він розроблений компанією *ESRI* та є пропрієтарним форматом даних. Проте сьогодні цей формат став дефакто стандартом обміну та публікації даних. Більшість програмних продуктів працюють з цим форматом, зокрема ПЗ QGIS, яке є найбільш популярним OpenSource додатком для ГІС.

Специфікація формату визначає набір файлів із загальним префіксом імені файлу, які зберігаються в тому самому каталозі. Три обов'язкових файли мають розширення файлів *shp, .SHX* та *.dbf.*

.shp – головний файл, містить набори геометричних об'єктів;

.shx – індексний файл, використовується для зв'язку між файлами *.dbf* та *.shp*;

.dbf – атрибутивний файл, містить атрибути об'єктів, описаних в *.shp* файлі у форматі *dBase IV*.

На сучасному етапі більшість геоданих зберігаються в СУБД, які підтримують збереження просторових даних. Наразі до них належать Oracle, SQLserver, Postgres та ін. ПЗ ArcGis підтримує два типи баз даних: персональну та геобазу даних з відповідними форматами: *mdb, gdt*.

Файли AutoCAD не є ГІС-форматом, проте сучасні професійні ГІС підтримують пряму конвертацію з формату DXF та навпаки. Доволі рідкими форматами в Україні є формати National Transfer Format (NTF), SOSI, Spatial Data File, TIGER, Vector Product Format (VPF).

Для обміну просторовими даними також використовують формат ASCII, який у більшості випадків використовують для збереження растрових даних.

Для прикладу наведено найпоширеніші формати: AutoCAD DXF, Cartesian coordinate system (XYZ), Digital Line Graph (DLG), Esri TIN, Geography Markup Language (GML), GeoJSON, GeoMedia, ISFC, Keyhole Markup Language (KML), MapInfo TAB format, National Transfer Format (NTF), Spatialite, Shapefile, Simple Features, SOSI, Spatial Data File, TIGER, Vector Product Format (VPF).

Растрові формати

Растрові файли зазвичай використовують для зберігання інформації зображення як скановані паперові карти чи аерофотозйомку. Їх також застосовують для передачі даних дистанційного зондування Землі. На відміну від інших растрових файлів, які характеризуються розрізненням у розмірах точок на дюйм, розрізнення на зображеннях дистанційного зондування виражається в метрах на піксел растру, що вказує розмір земельної ділянки певної області. Прикладами растрових форматів є: ADRG, RPF, CADRG, CIB, Digital raster graphic (DRG), ECRG, ECW, Esri grid, GeoTIFF, IMG – ERDAS IMAGINE, JPEG2000, MrSID, netCDF-CF.

Детальніше про формати й GIS-формат: https://en.wikipedia.org/wiki/GIS_file_formats

KML – геоінформаційний формат Google Maps

KML (англ. *Keyhole Markup Languag*e) – мова розмітки на основі XML для представлення тривимірних геопросторових даних у програмі «Google Earth». Підмножина мови KML 2.0 може використовуватись і для відображення двовимірних карт у сервісі «Карти Google».

KML-файли зазвичай поширюються в ZIP-архіві: KMZ.

Зміст файлів KML визначає один або кілька об’єктів для відображення в Google Earth. Цими об’єктами можуть бути:

– позначки на карті. Задається знаковий символ, який відображається в певному місці на карті, і його параметри: величина, текст і колір, величина написів, яка відображається біля знаку.

Є можливість створити позначку без знаку. Наприклад, поставити номери будинків району. Для цього потрібно просто в стилі створити порожнє посилання на значок: <Icon> <href> </ href> </ Icon>;

– багатокутник або набір ліній. Визначається колір ліній і колір підпису;

– зображення. Визначається положення зображення на поверхні Землі, а також його масштаб. Також можна розмістити зображення на екрані, що не переміщається разом з картою, наприклад логотип;

– тривимірна модель. Версія мови KML 2.1 (яка відповідає четвертій версії програми Google Earth) дає змогу підключати опис тривимірних об'єктів (наприклад, будівель і споруд). Тривимірну модель можна задати двома способами: завданням висоти плоских фігур (витягуванням) і посиланням на повноцінну модель у форматі Collada.

Крім згаданих характеристик, для кожного об'єкта також задаються основні геоінформаційні властивості (географічна широта й довгота, а також висота над рівнем моря або над рівнем поверхні Землі). Може бути наведено короткий опис об'єкта (який у подальшому відображається в підказці за запитом користувача). Може бути зазначений рекомендований ракурс спостереження вказаного на карті місця, тобто рекомендована висота, азимут і кут нахилу «віртуальної камери».

Об'єкти всередині KML-файлу можна організувати в ієрархічні структури папок і підпапок, щоб було зручніше спільно ввімкнути або вимкнути відображення логічно взаємопов'язаних груп об'єктів.

KML-файл може містити (в URL-формі) посилання на інші файли у форматі KML або KMZ, розташовані будь-де в мережі, і задавати умови й регулярність завантаження та відображення даних з цих зовнішніх джерел. Таке мережеве посилання також видно як своєрідну підпапку.

Мова KML багато в чому наслідує структуру мови GML – географічної мови розмітки. Приклад KML-розмітки:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.1">
<Placemark>
    <name>Київ</name>
<description><![CDATA[<p>Київ, Київська область,
Україна.</p>Столиця України. Місто побудоване на
березі річки Дніпро.]]></description>
    <LookAt id="khLookAt540_copy0">
        <longitude>38.0576198113139</longitude>
        <latitude>44.56963150481845</latitude>
            <altitude>0</altitude>
            <range>14693.40972993507</range>
            <tilt>49.10268313434742</tilt>
        <heading>37.85562764777833</heading>
    </LookAt>
    <Style>
            <IconStyle>
                <scale>0.9</scale>
                <Icon>
                    <href>root://icons/palette-
                    4.png</href>
                        <x>32</x>
                        <y>128</y>
                        <w>32</w>
                        <h>32</h>
              </Icon>
          </IconStyle>
          <LabelStyle>
              <scale>0.9</scale>
          </LabelStyle>
  </Style>
    <Point id="khPoint541_copy0">
<coordinates>38.06284424434902,44.56842733252498,
0</coordinates>
    </Point>
</Placemark>
</kml>
```

Формати GeoJSON та TopoJSON

GeoJSON – відкритий формат, призначений для зберігання графічних структур даних, заснований на форматі JSON.

Формат може зберігати примітивні типи для опису географічних об'єктів, як-от: точки (адреси та місця розташування), лінії (вулиці, шосе, кордони), полігони (країни, штати, ділянки землі). Також можуть зберігатися так звані мультитипи, які являють собою об'єднання декількох примітивних типів.

Формат GeoJSON відрізняється від інших стандартів ГІС тим, що він був написаний і підтримується не певною організацією зі стандартизації, а за допомогою робочої групи розробників.

Подальшим розвитком GeoJSON є TopoJSON, розширення GeoJSON, яке кодує геопросторову топологію та зазвичай забезпечує менший розмір файлів. Сьогодні ці формати підтримуються більшістю ГІС, а також безпосередньо застосовуються під час публікації інтерактивних картографічних даних у веб, зокрема засобами бібліотек OpenLayers та Leaflet.

```
{ "type": "FeatureCollection",
"features": [
 { "type": "Feature",
 "geometry": {"type": "Point", "coordinates":
[102.0, 0.5]},
 "properties": {"prop0": "value0"}
 },
 { "type": "Feature",
   "geometry": {
 "type": "LineString",
 "coordinates": [
 [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0,
1.0]
   ]
 },
 "properties": {
  "prop0": "value0",
   "prop1": 0.0
 }
 },
 { "type": "Feature",
   "geometry": {
   "type": "Polygon",
```

```
   "coordinates": [
   [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
[100.0, 1.0], [100.0, 0.0] ]
  ]
},
   "properties": {
   "prop0": "value0",
"prop1": {"this": "that"}
}
   }
]
}
```

Питання для самоконтролю

1. Покажіть на прикладі структуру даних у форматі XLS / XLSX (Document Office Open XML).

2. Поясніть графічні формати: JPEG, PNG.

3. Поясніть відмінність текстових форматів: TXT, CSV, Markdown, OpenDocument.

4. Поясніть особливості форматів представлення даних через API: JSON, XML.

5. Приклади форматів даних для роботи з геопросторовими даними.

6. Поясніть на прикладі особливість формату GIS.

7. Опишіть порядок очищення й підготовки даних для аналізу в Open Refine.

8. Наведіть приклади відкритих в інтернеті засобів підготовки даних для аналізу.

9. Структура та особливість графічного формату TIFF.

10. Структура та особливість текстових форматів TSV, PDF.

11. Структура та особливість форматів представлення даних RDF.

1.3. Огляд нереляційних БД (NoSQL DB)

1.3.1. Причини появи NoSQL баз даних

Концепція великих даних вимагає появи нової моделі даних, яка буде більше націлена на швидкість доступу й масштабованість. Виникала потреба в простіших, ніж існуючі, реляційні бази даних (далі – БД), які водночас не поступалися б їм у ефективності. Насамперед це завдання стосувалось побудови хмарних сховищ, де кінцевому користувачеві важлива швидкість доступу й можливий обсяг інформації, що там зберігатиметься. Проте потрібно критично оцінювати довгострокові перспективи впровадження нереляційних баз даних у конкретному бізнесі, на підприємстві. Як було вищезазначено, розрізняють чотири основні типи NoSQL баз даних. Вони відрізняються моделлю даних, підходом до розподіленості та реплікації, завдяки чому можуть у різній мірі підходити під певні види конкретних завдань [9].

1.3.2. Бази даних виду «ключ-значення» (key-value DB)

Бази даних «ключ-значення» являють собою найпростіший вид бази даних, будучи, по суті, асоціативним масивом, де кожному значенню відповідає свій унікальний ключ. Простота баз даних цього типу дає змогу широкій масштабованості. Під час їхньої розробки не потрібні схеми баз даних, вони не мають зв’язків між значеннями, кількість елементів асоціативного масиву обмежена лише обчислювальними потужностями. Саме тому цей вид баз даних цікавий насамперед компаніям, які надають хмарні послуги (рис. 1.7).



Рисунок 1.7 – Подання даних у базах даних «ключ-значення» (key-value DB)

З іншого боку, простота сховищ «ключ-значення» ускладнює або повністю унеможливлює більшість звичних операцій. Якщо ключами можна ще оперувати як завгодно, то спроба виконати пошук за значеннями може тривати на довше, ніж у реляційній базі даних. А разом з обмеженим набором маніпуляцій над значеннями різних частин сховища зменшується можливість швидкого аналізу збору статистики наявної в базі даних інформації. Тому схожі бази даних використовують у тому разі, коли конкретний вміст окремої комірки не є цікавим. Бази даних типу «ключ-значення» не можуть повністю замінити реляційні БД. Проте вони знайшли своє застосування як кеші для об'єктів, адже між кешуванням об'єктів різних користувачів також немає зв'язків – важлива лише швидкість доступу до кешу, а також можливість швидко змінювати масштаб системи.

Найбільш відомими прикладами СУБД цього типу є Amazon DynamoDB, Berkeley DB, MemcacheDB, Redis і Riak.

1.3.3. Документо-орієнтовані бази даних

Документо-орієнтована БД є системою зберігання ієрархічних структур даних (документів), що має структуру дерева або лісу. Структура дерева починається з кореневого вузла й може мати кілька внутрішніх і зовнішніх вузлів. Зовнішні вузли містять кінцеві дані, які під час додавання заносяться в індекси бази, завдяки яким можна здійснювати швидкий пошук навіть у разі досить складної загальної структури сховища. Фактично документо-орієнтовані БД є більш складною версією сховищ «ключ-значення», що дають змогу здійснювати вибірку за запитом без повного завантаження окремих документів у оперативну пам'ять. Деталі визначення цієї основної концепції різняться в різних реалізаціях документо-орієнтованих баз даних, але загалом усі вони мають на увазі, що документ ізолює дані та кодує їх (або інформацію) у певному форматі або кодуванні. Це кодування можуть використовувати XML, YAML, JSON, BSON, а також бінарні формати, як-от PDF чи документи Microsoft Office (MS Word, Excel тощо). Документи всередині документо-орієнтованої бази даних у певному розумінні схожі до записів або рядків реляційних баз даних, але вимоги до них не такі жорсткі. Вони не потребують задовольняти ні стандартній схемі, ні мати однакові секції, частини або ключі (рис. 1.8).

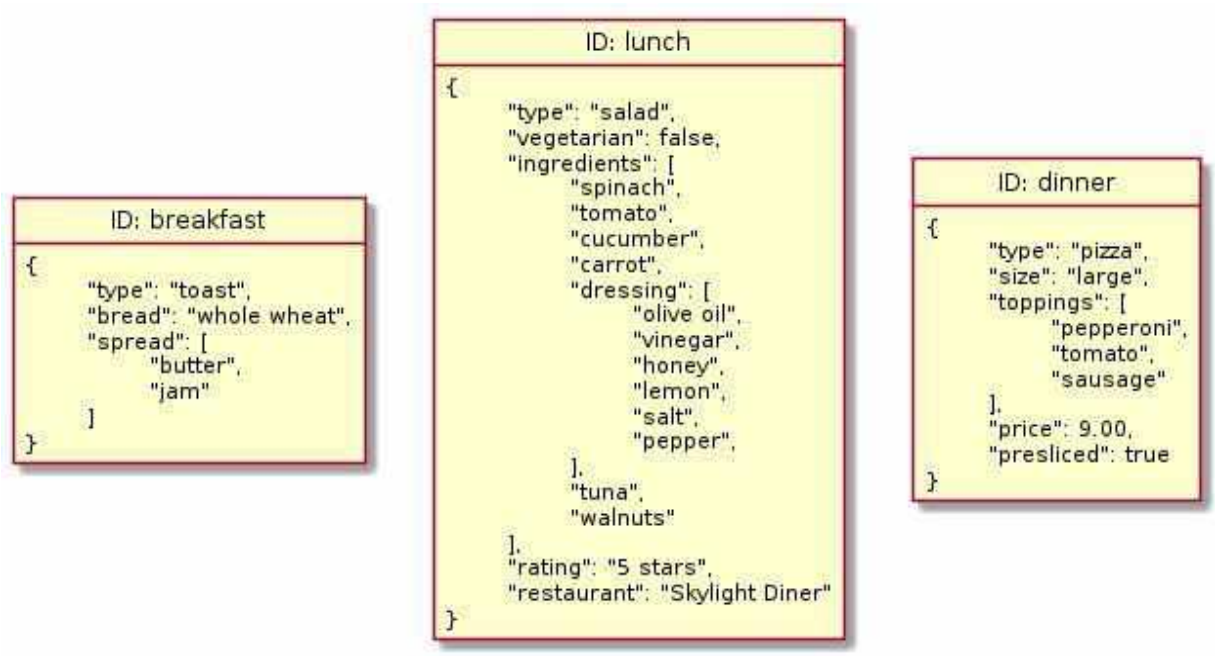


Рисунок 1.8 – Схема документо-орієнтованої БД

Характеристики:

– база даних не виділяє окремий формат або схему;

– кожен документ може мати свою внутрішню структуру;

– документні БД є гарним вибором для швидкої розробки;

– у будь-який момент можна змінювати властивості даних, не змінюючи структуру або самі дані.

Приклад

Перший документ може бути таким:

```
{
    Ім’я: "Микола",
    Адреса: "Коцюбинського, 5",
    Захоплення: "рибалка"
}
```

Другий документ може бути:

```
{
    Ім’я: "Петро",
    Адреса: "Панаса Мирного, 15",
    Діти: [
        {Ім’я: "Михайло", Вік: 10},
        {Ім’я: "Євгенія", Вік: 8},
        {Ім’я: "Одарка", Вік: 5},
        {Ім’я: "Олена", Вік: 2}
    ]
}
```

Ці два документи мають спільні структурні елементи, але кожен має і свої власні унікальні елементи. На відміну від реляційних баз даних, де кожен запис містить однакові поля, з яких невикористані залишаються порожніми, у документо-орієнтованих базах немає пустих полів у жодному документі (запису), як у прикладі вище. Такий підхід дає змогу додавати нову інформацію в запис без потреби, щоб кожен інший аналогічний запис бази даних мав таку саму структуру.

Наприклад, під час створення музичного сховища можна виготовити колекцію музики 80-х років, у ній зробити окремі колекції за роками, а всередині них – окремі документи з треками, випущених у цей рік альбомів. Але якщо користувач побажає побачити рейтинг найпопулярніших композицій певного десятиліття, цей запит буде виконуватися досить довго, адже доведеться переглянути кожен документ усієї бази даних. Отже, можна зробити висновок, що документо-орієнтовані БД знайдуть своє застосування в завданнях, де треба впорядковане зберігання інформації, але без зв’язків між даними і де не потрібно постійно збирати статистику щодо них. Документи не вимагають визначення схеми, це означає, що кожен окремий документ може складатися з будь-якої кількості унікальних полів, на відміну від реляційних баз даних, в яких у разі спроби зберігати різнорідні дані неминуче з’являються порожні поля.

Приклади СУБД цього типу: CouchDB, Couchbase, MarkLogic, MongoDB, eXist.

1.3.4. Графові бази даних

Графова модель бази даних являє собою узагальнення мережевої моделі даних і відрізняється сильними зв’язками між вузлами. Графові бази даних найкраще підходять для реалізації проєктів, які передбачають природну графову структуру даних, насамперед соціальних мереж, а так само для створення семантичної павутини. В аналогічних завданнях вони значно випереджають реляційні БД щодо продуктивності, простоти внесення змін і наочності подання інформації (рис. 1.9).

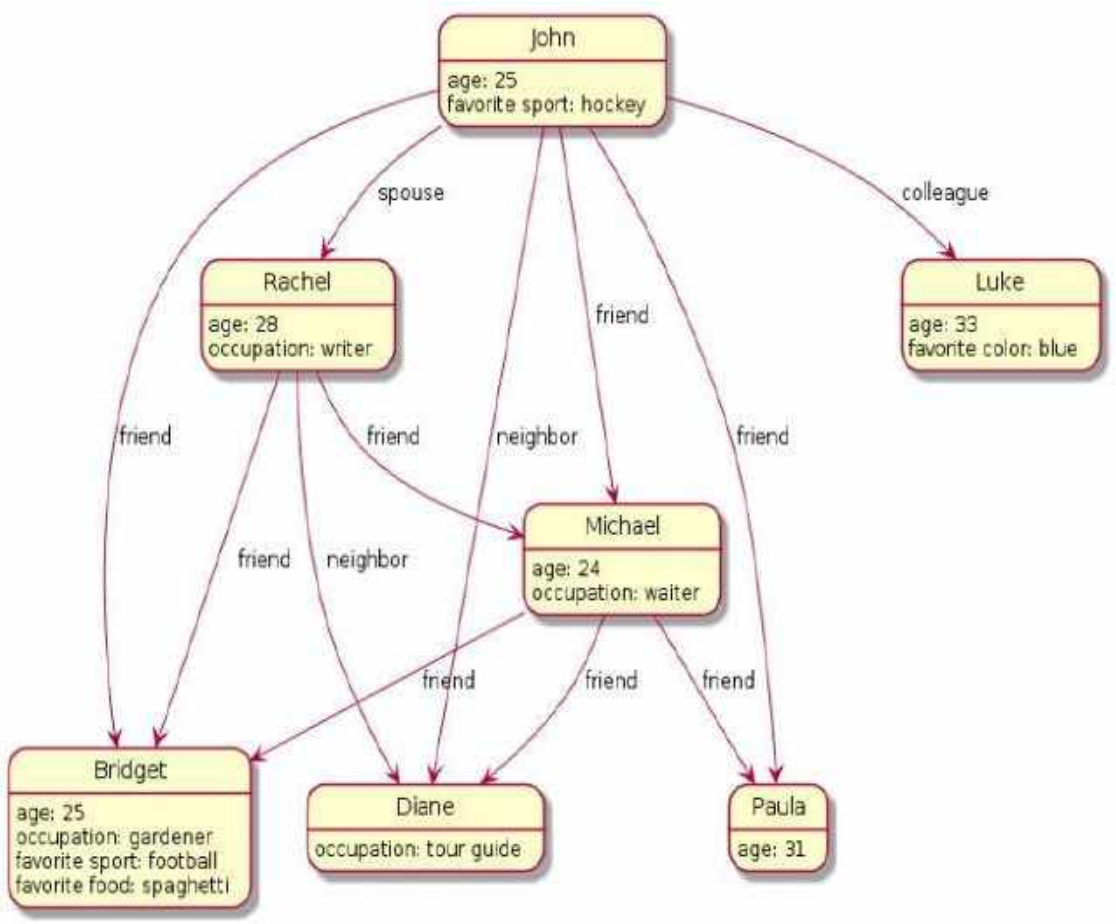


Рисунок 1.9 – Графова модель бази даних

Графові бази представляють дані як окремі вузли, які можуть мати будь-яку кількість пов’язаних з ними властивостей.

Характеристики:

– фокусуються на зв’язках між елементами;

– чітко відображають зв’язки між типами даних;

– не вимагають покрокового обходу для переміщення між елементами;

– немає обмежень у типах зв’язків.

У деяких базах даних існують механізми спеціальної оптимізації для роботи з SSD-накопичувачами. Для роботи з досить великими графами використовуються алгоритми, які передбачають часткове приміщення графа в оперативну пам’ять. Найбільш відомими графовими СУБД є ArangoDB, FlockDB, Giraph, HyperGraphDB, Neo4j, OrientDB.

1.3.5. Bigtable – схожі бази даних (стовпчикові БД)

Bigtable – схожі БД (або стовпчикові БД) містять дані, впорядковані як розріджена матриця, рядки та стовпці якої використовуються як ключі. Ці сховища мають багато спільного з документо-орієнтованими базами даних – системами керування вмістом (CRM), Event-агрегаторами реєстрації на події, блоги. Не варто плутати bigtable – схожі бази даних з лінійними сховищами, які є, по суті, реляційними БД з роздільним зберіганням колонок.

У стовпчикових базах замість таблиць є структури – «стовпчик сімейства». Сімейства містять рядки, кожна з яких визначає власний формат. Рядок складається з унікального ідентифікатора, що використовується для пошуку, за яким слідують набори імен і значень стовпців (рис. 1.10).

Характеристики:

– стовпчикові БД зручні під час роботи з додатками, що вимагають високої продуктивності;

– дані та метадані доступні по одному ідентифікатору;

– гарантоване розміщення всіх даних з рядка в одному кластері, що спрощує сегментацію та масштабування даних.

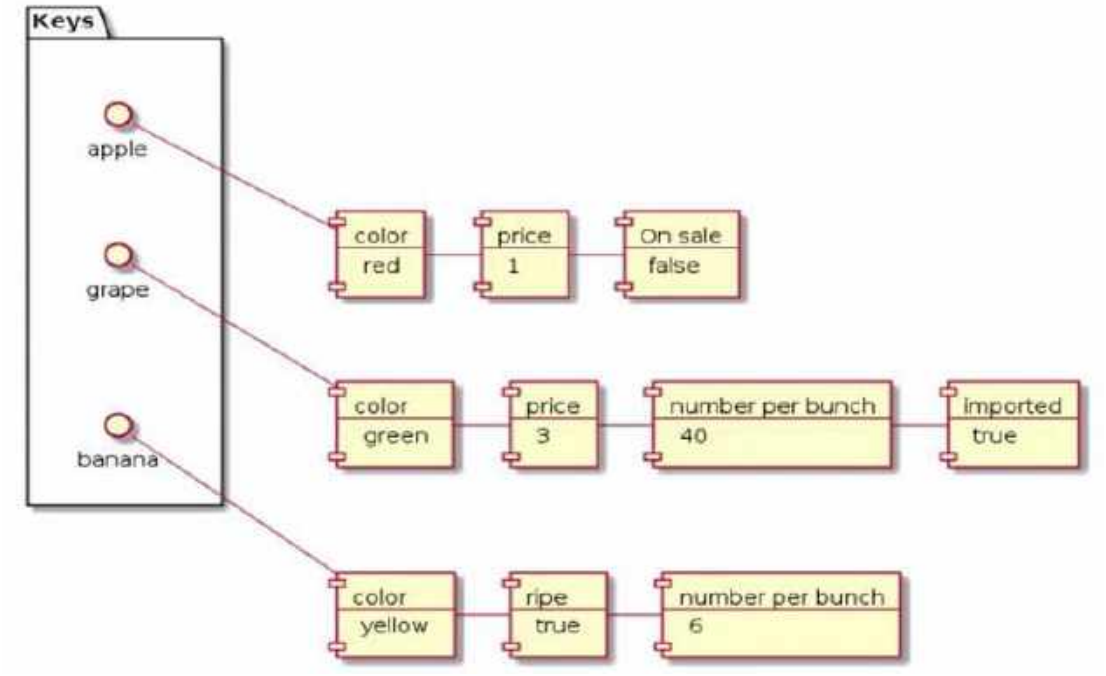


Рисунок 1.10 – Модель Bigtable – схожої БД

Зазвичай ці бази даних застосовуються для вебіндексування й рішення інших завдань, які передбачають значні обсяги даних.

Прикладами СУБД цього типу є: HBase, Cassandra, Hypertable, SimpleDB.

Сильні та слабкі сторони NoSQL баз даних. Сильні сторони NoSQL БД

Зазвичай NoSQL БД – це розподілені системи. Вагомими перевагами під час роботи з розподіленими системами мають усі типи нереляційних БД, за винятком графових БД – вони за визначенням передбачають значну кількість зв'язків між вузлами даних.

Лавиноподібне зростання кількості даних у світовій мережі загострило проблему вертикальної масштабованості – обчислювальні потужності процесорів можуть зростати нескінченно та й ціна кількох простих серверів є меншою, ніж одного високопродуктивного. У такій ситуації правильним виходом стане *горизонтальне масштабування*, коли кілька незалежних машин з'єднуються разом і кожна з них обробляє лише частину запитів. Така архітектура робить можливим швидке збільшення потужності кластера через додавання нового сервера. Розраховані на роботу в розподілених системах NoSQL БД спочатку проєктуються з таким розрахунком, щоб усі процедури реплікації, розподілу даних і забезпечення відмовостійкості виконувалися самою NoSQL БД.

Ключові переваги NoSQL БД у розподілених системах полягають також у процедурах *шарингу й реплікації*.

Реплікація – це копіювання даних під час їхнього оновлення на інші сервера. Цей механізм дає змогу домогтися більшої відмовостійкості та масштабованості системи. Прийнято виділяти два види реплікації: *master-slave* та *peer-to-peer*.

Перший тип *master-slave* має на увазі наявність одного майстер-сервера й кількох дочірніх серверів. Запис може проводитися тільки на майстер-сервер, а він зі свого боку передає зміни на дочірні машини. Цей тип реплікації дає хорошу масштабованість на читання (читання може відбуватися з будь-якого вузла мережі), але не дає змогу масштабувати операції записів. Запис іде тільки на один майстер-сервер. Також такий варіант організації реплікації передбачає складності в разі несправності майстер-сервера.

У такому разі повинен відбуватися автоматичний або ручний вибір нового майстер-сервера з решти.

Другий тип *peer-to-peer* припускає, що всі вузли рівні в можливості обслуговувати запити на читання й запис. Інформація про оновлення даних передається від сервера до сервера по колу.

Шаринг – поділ масиву інформації з різних вузлів мережі, коли кожен вузол відповідає тільки за певний набір даних і обробляє запити на читання й запис, що належать лише до цього набору даних.

Досить популярним аргументом для вибору NoSQL БД має той факт, що соціальні дані не є реляційними й реалізація великих соціальних мереж засобами реляційного SQL-сховища пов'язана з певними складнощами. Дійсно, формування стрічки новин для користувача засобами реляційної бази даних являє собою операцію з'єднання декількох таблиць. Пости новинної стрічки, лайки та коментарі, аватари користувачів і безліч інших даних, потрібних для формування стрічки, зазвичай зберігаються окремо, тому зведення цих даних воєдино часто проходить не так швидко, як хотілося б.

Збереження всієї стрічки користувача як єдиної ненормалізованої структури є правильним рішенням. Але реалізовані в цей момент графові NoSQL-бази мають проблему з масштабуванням, що не дає змогу використовувати їх для великих соціальних мереж. Крім того, реляційні сховища мають і ряд інших переваг, серед яких надійність, гарантія цілісності інформації та безпеки даних користувача. Під час реалізації дійсно складних багатокористувальних проєктів блискавична швидкість і нескінченна масштабованість системи рідко виходять на перший план. Насамперед компанії прагнуть зробити надійний продукт.

Можна зазначити також такі NoSQL. БД мають позитивні сторони.

Простота роботи. Багато NoSQL баз даних, загалом сховища виду «ключ-значення», мають, порівняно з реляційними базами даних, досить урізану функціональність, яка їм просто не потрібна для виконання поставлених завдань. Тому оператору бази даних не треба глибоких знань потужного та гнучкого механізму

роботи з SQL-запитами. Це надто знижує вхідний поріг для початку роботи з NoSQL-сховищами.

Простіший синтаксис запитів – менше помилок. Для спрощення роботи з базою даних деякими розробниками використовується технологія, яка дає змогу автоматично транслювати операції з об'єктами в запити до бази даних (Object-Relational Mapping, ORM). Найчастіше схожі рішення працюють неефективно та плодять безліч непотрібних або відверто помилкових запитів. Не можна зазначити, що розробники ORM погано виконують свою роботу, просто завдання занадто складне. Мова SQL універсальна, проте досить ємна, для повноцінної роботи з нею потрібний певний багаж знань. Водночас власні мови запитів сучасних NoSQL-сховищ набагато більше підходять для виконання простих маніпуляцій з базою даних.

Приклади успішних реалізацій на NoSQL БД. NoSQL-сховища досить добре прижилися як основна база даних для соціальних мереж Instagram і Facebook, і це вагомі історії успіху для всього сімейства NoSQL.

Недоліки NoSQL БД

NoSQL-рішення не вимагають визначати схему бази даних перед початком роботи. Але без етапу первісного тестування й планування можна в процесі розробки наштовхнутися на непередбачені труднощі, які можуть навіть повністю відсікти варіант роботи з конкретним NoSQL-рішенням. А з огляду на вищезазначені труднощі швидкого переходу з однієї нереляційної бази даних на іншу ціна помилки стає досить високою.

Обмеження структури реляційних БД певною мірою гарантують цілісність даних – інформація, яка не задовольняє обмеженням щодо типу, просто не зможе потрапити в базу даних. У разі використання, наприклад, сховищ типу «ключ-значення» завдання контролю цілісності даних повністю лягає на додаток. Також процес створення реляційного сховища містить етап проєктування моделі даних. На цій стадії можна оцінити вузькі місця обраної стратегії та спроєктувати дійсно надійну та зручну систему.

Для використання на довгострокову перспективу потрібно враховувати такі особливості NoSQL БД:

1. Додатки значно прив'язуються до конкретної БД. Наприклад, мова SQL універсальна для всіх реляційних баз даних і користувачеві не доведеться кардинально переписувати весь код у разі зміни СУБД. Якщо дві NoSQL-системи концептуально практично не відрізняються, вони матимуть досить мало загальних стандартів у API і в специфіці запитів.

2. Обмежена ємність вбудованої мови запитів. SQL має багату історію та безліч стандартів. Це потужний і складний інструмент для операцій з даними та складанням звітів. Практично всі мови запитів і методи API сховищ NoSQL були створені на основі певних функцій SQL – як наслідок, вони мають значно меншу функціональність.

3. Низька цінність і вузька профільність знань. Фахівців з хорошим знанням SQL набагато простіше знайти тоді, коли специфікою роботи API деяких NoSQL-рішень на серйозному рівні мало хто захоплюється, це означає, що багато специфічних моментів оператору бази даних доведеться освоювати «на ходу».

У хмарних сховищах і розроблених для них NoSQL-рішеннях часто використовується принцип множинної оренди. Він полягає в тому, що значна кількість користувачів одночасно використовує одну й ту саму систему. Щоб запобігти її перевантаження в рішеннях, розрахованих на значну масштабованість, застосовують політику обмеження запитів. Наприклад, у SimpleDB час виконання запиту не може перевищувати 5 секунд, а в Google AppEngine Datastore один запит до бази не дозволяє отримати більше 1 000 записів.

Ці обмеження не страшні для простої роботи програми, але значно позначаються на можливостях аналітики даних. Компанії часто отримують вигоду від великих обсягів даних користувача. На основі їхнього аналізу можна виділяти найбільш популярні продукти, складати «на льоту» списки рекомендацій для конкретних груп користувачів ґрунтуючись на їхніх особистих даних або історії. Для багатьох NoSQL-рішень схожі функції складно реалізувати, бо вони є просто неможливі.

Цікавим варіантом для вирішення цієї проблеми стає створення окремого сховища, куди дані періодично дублюються для аналізу. Звичайно, перенесення мільйонів записів частинами по 1 000 штук за запит може сильно затягнутися, так що цей момент варто продумати заздалегідь.

Пік популярності NoSQL-рішень був багато в чому пов'язаний з амбітною заявою, що надійшли від соціальної мережі Twitter. Вони бачили певні недоліки роботи з реляційним сховищем MySQL для своїх твітів і виявили бажання перейти на нову NoSQL СУБД Cassandra. Але ця ідея так і не була втілена в життя. Як коментують цей момент самі співробітники Twitter, компанія оцінила свої пріоритети, після чого рішення було визнано занадто ризикованим.

Отже, найбільш вдалі NoSQL-рішення насамперед націлені на вирішення специфічних завдань і створюються гігантами IT-індустрії для власних потреб – це продукти від Google, Amazon, Microsoft і Apache, які обслуговують конкретні проєкти. Google AppEngine Data Store можливо використовувати тільки з вебсервісами Google, сховище SQL Data Services є частиною платформи Microsoft Azure, а SimpleDB входить до складу Amazon WebServices.

Нехмарні бази даних, якими можна користуватися, просто завантаживши та встановивши в себе на ПК, часто є досить молодими відкритими проєктами, які так само створювалися під конкретні потреби. Вони не були створені, щоб вирішувати весь обсяг завдань, який дають змогу вирішувати SQL бази даних. До того ж призначення найчастіше не є ключовою перешкодою для масового впровадження NoSQL-систем, вони мають певний набір недоліків, який некритичний для гігантів IT-індустрії, але стають вирішальним для більшої частини звичайних компаній.

Багато нереляційних баз даних відмінно працюють, виконуючи мету свого створення, але водночас вони не є універсальним продуктом, яким є реляційні БД. NoSQL бази даних показують себе з добре в симбіозі з реляційними базами даних. Наприклад, у системах, де основний обсяг інформації зберігає SQL, а за кеш відповідає NoSQL. Для захоплення більш істотних позицій на ринку нереляційним системам усе ще не вистачає багатьох базових речей: універсальності, надійності, цілісності, передбачуваності.

Питання для самоконтролю

1. Поясніть причини появи NoSQL баз даних.
2. Опишіть і покажіть на прикладі використання баз даних виду «ключ-значення» (key-value DB).
3. Опишіть і покажіть на прикладі використання документо-орієнтованої бази даних.
4. Опишіть і покажіть на прикладі використання графової бази даних.
5. Опишіть і покажіть на прикладі використання Bigtable – схожої бази даних (стовпчикової БД).
6. Опишіть і покажіть на прикладі сильні та слабкі сторони NoSQL БД.
7. Опишіть і покажіть на прикладі реалізовані проєкти на NoSQL базах даних.
8. Поясніть ключові переваги NoSQL БД у розподілених системах.
9. Наведіть приклад коду документо-орієнтованої бази даних.
10. Наведіть характеристики графових баз даних.
11. Опишіть модель Bigtable-подібних баз даних (стовпчикові БД).

ТЕМА 2
ЕКОСИСТЕМА BIG DATA

2.1. Технологія Hadoop

2.1.1. Архітектура екосистеми Big Data на базі Hadoop

Підприємства потребують два основних типи інформації – транзакційної інформації та аналітичної інформації.

Транзакційна інформація фіксується та зберігається, коли відбуваються події. Транзакційна інформація використовується для аналізу щоденних звітів про продажі та виробничих графіків, щоб визначити, скільки інвентаризації потрібно здійснити.

Аналітична інформація забезпечує виконання завдань з аналізу управлінських процесів.

Початково в сукупність підходів і технологій Big Data вносилися засоби масово-паралельної обробки невизначено-структурованих даних, як-от СУБД *NoSQL*, алгоритми *MapReduce* та інструменти проєкту *Hadoop*. У подальшому до технологій *великих даних* почали відносити й інші рішення, які забезпечують схожі за характеристиками можливості обробки надвеликих масивів даних, а також деякі апаратні засоби. На сьогодні це компоненти та технології, які умовно можна продати такими взаємодіючими рівнями: джерела даних; інтерфейсний рівень; рівень додатків великих даних; засоби аналітики; рівень збереження; фізичний рівень (рис. 2.1) [10].

На рис. 2.1 подані складові взаємодіючих рівнів екосистеми *Big Data*.

Hadoop – набір утиліт, бібліотек і фреймворків, що вільно розповсюджуються для розробки й виконання розподілених програм, які працюють на кластерах із сотень і тисяч вузлів. Hadoop вважається однією із основоположних технологій великих даних, забезпечує розподілене зберігання значних обсягів даних (у масштабі петабайт) на недорогих комп'ютерах. За допомогою Hadoop додатки Big Data взаємодіють з вузлами зберігання без безпосереднього звернення до фізичних серверів. Hadoop складається із двох основних компонентів *HDFS* і *MapReduce.*

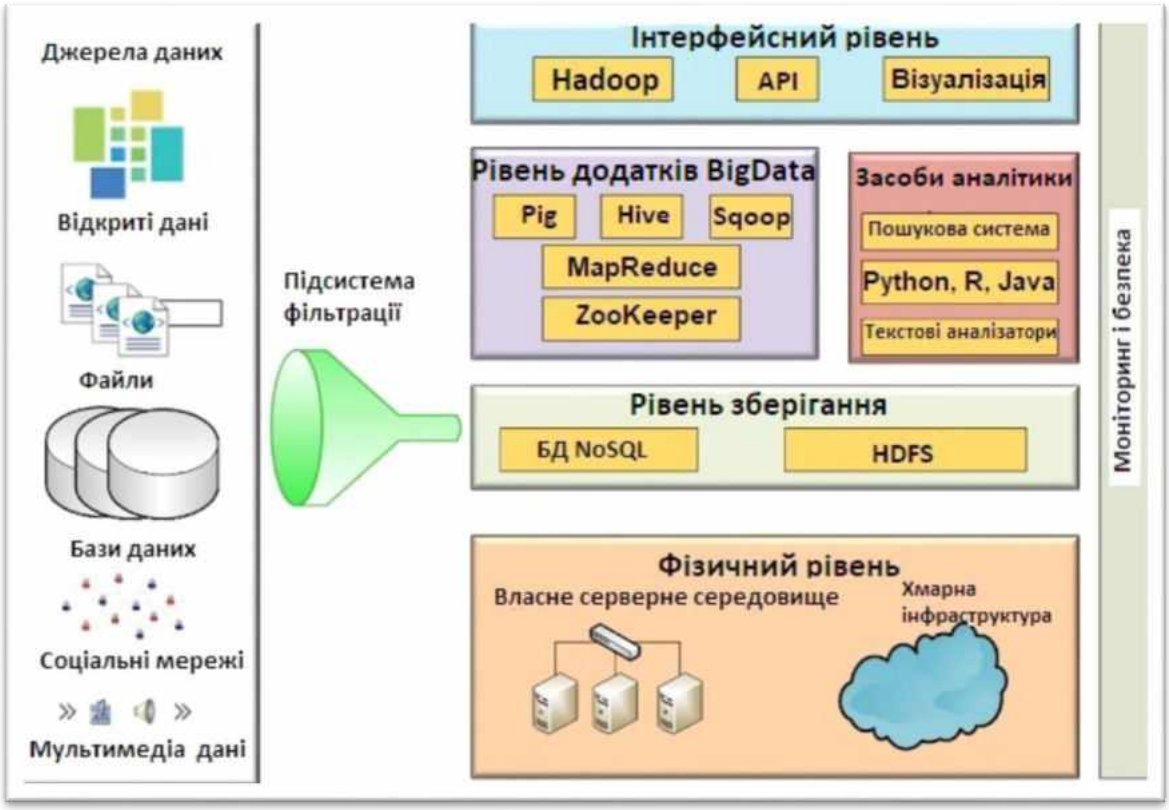


Рисунок 2.1 – Базова архітектура екосистеми Big Data

HDFS (Hadoop Distributed File System) – розподілена й у високому ступені масштабована файлова система. HDFS побудована за принципом однократного запису й багаторазового читання (*write-once-read-many*) і має блокову структуру, в кожному блоці якої можна зберігати файл або частину файлу. Варто зазначити, що до файлів HDFS можна звернутися безпосередньо, як це робиться в традиційних файлових системах. Для ефективної роботи в HDFS потрібно розробляти досить складні спеціальні програми.

MapReduce – модель розподілених обчислювань у комп'ютерних кластерах, представлена компанією Google. Згідно з цією моделлю додаток розділяється на значну кількість однакових елементарних завдань, що виконуються на вузлах кластера й потім зводяться в кінцевий результат.

NoSQL (англ. Not Only SQL, не лише SQL) – загальний термін для різних нереляційних баз даних і сховищ, не означає якусь конкретну технологію чи продукт. Звичайні реляційні бази даних добре підходять для досить швидких і однотипних запитів, а на складних і гнучко побудованих запитах, характерних для великих

даних, навантаження перевищує розумні межі й використання реляційних СУБД стає неефективним Виділяють чотири основні класи NoSQL БД:

1. Бази даних з доступом до значень за ключем (key-value БД) належать до найпростішого типу NoSQL БД і можуть зберігати об'єкти будь-яких типів.

2. Документо-орієнтовані БД дає змогу зберігати ієрархічні структури даних.

3. Бази даних на основі графів (графові БД) використовуються в ситуації, коли важливі не стільки самі дані, скільки зв'язки між ними. Цей клас NoSQL БД може застосовуватися в мережному моделюванні, а також під час створення сервісів рекомендацій.

4. Бази даних на основі стовпців (BigTable – схожі БД) надають інструмент для індексування неструктурованої інформації та слугує загалом у вебдодатках.

Фізичний рівень

Фізичний рівень представляє інфраструктуру для роботи рівня зберігання. Інфраструктура великих даних масштабується горизонтально, що, з одного боку, забезпечує високу надійність, а з іншого – дешевизну щодо традиційних вертикально інтегрованих рішень.

Високий ступінь горизонтальної інтеграції передбачає застосування архітектури без поділу ресурсів (*share-nothing architecture*), коли вузли не мають спільних ресурсів і не конфліктують за них. У системах *Big Data* це досягається за рахунок того, що дані й функції для управління ними перебувають на одному вузлі.

Рівень додатків

Рівень додатків, який також називають рівнем платформи, регулює доступ до даних і керує запитами. Ключовою технологією на цьому рівні є *MapReduce*.

*MapReduc*е є технологією для пакетного виконання процедур з обробки великих обсягів даних. Незалежно від конкретної реалізації типове рішення MapReduce працює так:

– під час надходження запиту на обробку даних з боку клієнтського додатку функція Map розділяє вихідні дані на частини й розподіляє їх по вузлах системи з урахуванням вимог до балансування й відмовостійкості;

– після завершення завдань на вузлах функція Reduce отримує результат обробки з кожної частини та об'єднує всі результати в один, який і повертає до клієнтського додатка.

На рівні додатків можуть застосовуватися такі інструменти:

Hive – сховище даних, реалізоване під Hadoop. Hive має sql-подібний інтерфейс і може робити агрегацію для значних обсягів.

Pig – скриптова мова програмування, за допомогою якої можна розробляти завдання для MapReduce більш ефективно, ніж на мові Java. Варто зазначити, що Pig призначений для обробки великих наборів даних і на малих обсягах від нього не варто чекати хороших результатів.

HBase – NoSQL СУБД, що реалізована на основі стовпців. На відміну від аналогів HBase найбільш адаптована для роботи в Hadoop.

Sqoop – інструмент на базі командного рядка, що призначений для перенесення великих обсягів даних з неструктурованого сховища в структуроване (наприклад, у реляційну СУБД).

ZooKeeper – координатор для різних екземплярів Hadoop. Він забезпечує синхронізацію та захист від збоїв для вузлів системи.

Рівень моніторингу та інформаційної безпеки

Щоб звести до мінімуму час простою для компонентів Big Data і всієї системи загалом, потрібна система моніторингу, що реалізована за принципом автономності кожного зі своїх компонентів. Найбільшою популярністю в середовищі Big Data користуються системи моніторингу з відкритим кодом Ganglia і Nagios.

Що стосується безпеки, то організація ефективної системи захисту в розподіленій системі Big Data є набагато складнішим завданням, ніж у традиційних системах. Основна складність полягає у відстеженні всієї системи цілком, коли логи формуються на багатьох вузлах, а формування кінцевого результату відбувається без будь-якої можливості перевірки з боку користувача.

Апаратні рішення

Корпорації Teradata, EMC тощо пропонують апаратно-програмні комплекси, призначені для обробки великих даних. Ці комплекси поставляються як готові до установки в телекомунікаційні шафи, що містять кластер серверів і адміністративне програмне

забезпечення для масово-паралельної обробки. Сюди іноді відносять апаратні рішення для аналітичної обробки в оперативній пам'яті, зокрема апаратно-програмні комплекси Hana компанії SAP і комплекс Exalytics компанії Oracle, незважаючи на те, що така обробка початково не є масово-паралельною, а обсяги оперативної пам'яті одного вузла обмежуються кількома терабайтами.

Підсистема фільтрації (*Ingestion Layer*) слугує бар'єром для значної кількості даних, що потрапляють у систему Big Data з різних джерел. Основним призначенням підсистеми фільтрації є відокремлення сміття від релевантної інформації й тим самим підготування даних для подальшої обробки всередині платформи Big Data.

Обробка даних у підсистемі фільтрації передбачає виконання таких процедур:

1. Ідентифікація відомих форматів даних або призначення форматів за стандартним налаштуванням для неструктурованих даних.
2. Фільтрація вхідної інформації на предмет релевантності для цієї організації. Релевантність оцінюється на підставі системи управління даними (Master Data Management).
3. Валідація та аналіз даних.
4. Зменшення шуму охоплює очищення даних і мінімізацію різних похибок.
5. Трансформація передбачає за потреби розділення наборів даних на складові, зведення даних, денормалізацію та підсумовування.
6. Стиснення даних зменшує розмір даних без втрати результатів попереднього аналізу.
7. Інтеграція забезпечує вивантаження відфільтрованої інформації на рівень зберігання Hadoop.

Консалтингова компанія McKinsey, крім технологій NoSQL, Map Reduce, Hadoop, вносить у контекст придатності для обробки *великих даних* також технології *Business Intelligence* та реляційні системи управління базами даних з підтримкою мови SQL.

Вважається, що *великі дані* і є технологією **Hadoop**. Проте *великі дані* є дещо більше, ніж Hadoop. Однією з ключових вимог є розуміння й навігація за розрізненими джерелами великих даних.

У квітні 2008 року було оновлено світовий рекорд зі швидкості сортування терабайта даних із застосуванням технології Hadoop. Час сортування становив 209 секунд (попередній рекорд – 297 секунд). Кластер складався із 910 вузлів. У листопаді того самого року компанія Google представила власну реалізацію Map Reduce, здатну відсортувати 1 терабайт за 68 секунд.

Після цього технологія Hadoop стрімко увійшла в повсякденну діяльність великих компаній. Ця технологія виконувала роль платформи зберігання й аналізу даних загального призначення. Наразі існують дистрибутиви Hadoop як від великих загальновизнаних фірм, включно із EMC, IBM, Microsoft та Oracle, так і від компаній з вузькою спеціалізацією, як-от Cloudera, Hortonworks і MapR.

2.1.2. Особливості використання Hadoop

Екосистема *Hadoop* використовується в таких компаніях: IBM, ebaY, Twitter, Amazon, Adobe, Yahoo, Facebook, LinkedIn та ін.

Особливості цієї платформи:

– працює з Big Data на звичайних серверах;

– потужне відкрите співтовариство;

– безліч різних продуктів і засобів використовують Hadoop.

Для розгортання кластера Hadoop зазвичай використовують звичайні сервера (не суперкомп’ютери або десктопи), з’єднані в мережі та розташовані в одному місці.

Переваги Hadoop

Можливість зберігати й обробляти значну кількість даних. З розвитком соціальних мереж та інших рішень доводиться мати справу з постійно зростаючими обсягами даних. Тому це досить важливий чинник.

Обчислювальна потужність. Модель розподілених обчислень Hadoop обробляє великі дані швидко. Чим більше обчислювальних вузлів, які використовують, тим більше обчислювальної потужності є.

Відмовостійкість. Дані й обробка додатків захищені від відмови обладнання. Якщо вузол втрачає працездатність, то завдання автоматично перенаправляються до інших вузлів, щоб

упевнитися, що розподілені обчислення не перестали працювати. Багаторазові копії (реплікація) всіх даних виробляються автоматично.

Гнучкість. На відміну від традиційних реляційних баз даних не потрібно попередньо обробляти дані, перш ніж зберегти їх. Можете зберігати стільки даних, скільки хочете (різних типів) і вирішуєте, як використовувати їх пізніше. Це охоплює неструктуровані дані як текст, зображення й відео.

Низька вартість. Платформа з відкритим вихідним кодом безкоштовна та використовує товарні апаратні засоби, щоб зберігати значну кількість даних.

Масштабованість (горизонтальна). Можна легко наростити свою систему, щоб обробити більше даних, просто додавши вузли. І для цього не потрібно глибокого адміністрування.

Інкапсуляція складності реалізації. Hadoop приховує багато складності розподілених і багатопоточних систем. Звільняє розробника від турботи про проблеми системного рівня, як-от очікування даних, організація передачі даних, розподіл даних, доставка коду. Дає змогу розробнику сфокусуватися на розробці додатків і реалізації бізнес-логіки.

Опис компонентів ядра Hadoop

Основними компонентами (ядром) Hadoop є [10; 11]:

– Hadoop Distributed File System (HDFS) – розподілена файлова система, що дає змогу зберігати інформацію практично необмеженого обсягу;

– YARN – програмна модель для управління ресурсами й менеджменту завдань, зокрема, містить програмну модель MapReduce (використовуються в нових версіях Hadoop замість MapReduce);

– Hadoop common – бібліотеки й утиліти, які використовуються іншими модулями Hadoop;

– Hadoop MapReduce – програмний каркас для програмування розподілених обчислень у рамках парадигми MapReduce.

На рис. 2.2 зображено схему основних компонентів екосистеми Hadoop.

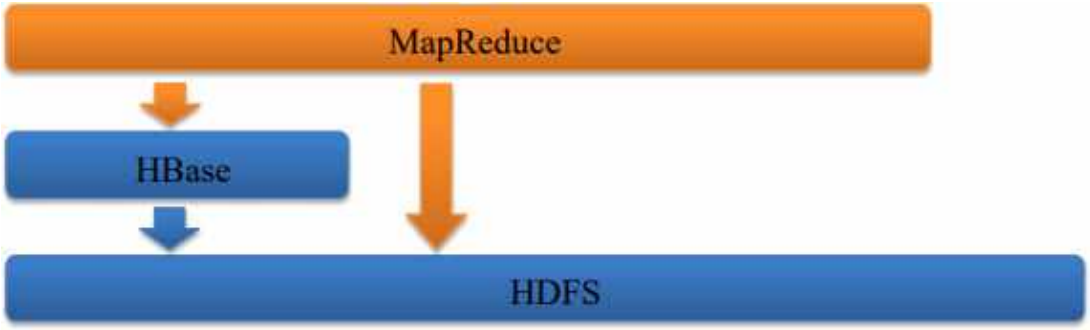


Рисунок 2.2 – Схема основних компонентів
ядра екосистеми Hadoop

Також існує значна кількість проєктів, безпосередньо пов'язаних з Hadoop, але які не входять до набору основних компонентів Hadoop [10]:

– *Hive* – інструмент для «SQL-like запитів» над великими даними (перетворює «SQL-запити» в серію Map Reduce-завдань);

– *Pig* – мова програмування для аналізу даних на високому рівні. Один рядок коду цією мовою може перетворитися в послідовність MapReduce-завдань;

– *HBase* – колонкова база даних, яка реалізує парадигму BigTable;

– *Cassandra* – високопродуктивна розподілена «key – value» база даних;

– *ZooKeeper* – сервіс для розподіленого зберігання конфігурацій і синхронізації цих змін;

– *Mahout* – бібліотека й движок машинного навчання на великих даних.

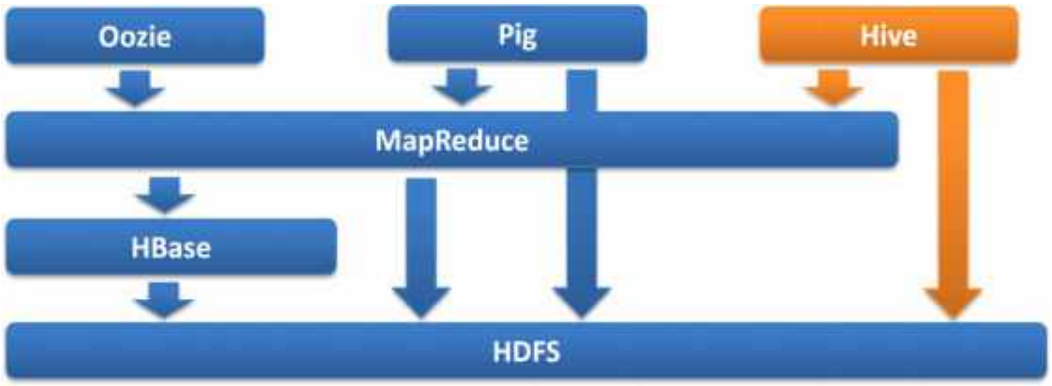


Рисунок 2.3 – Схема основних компонентів екосистеми Hadoop

Такі модулі розширюють можливості платформи, що дає змогу розробникам зберігати й аналізувати різні типи даних.

2.1.3. Pig – високорівнева мова опису потоків даних

Pig підвищує рівень абстракції під час обробки великих наборів даних. *MapReduce* дає змогу програмісту задати функцію відображення, а потім функцію згортки, але для адаптації механізму обробки даних до цієї схеми часто доводиться використовувати кілька стадій *MapReduce,* а це ускладнює завдання. Під час використання *Pig* структури даних набагато складніше, з безліччю значень і багаторівневої ієрархією, а перетворення, застосовувані до даних, набагато потужніше.

Pig складається з двох основних частин:

1) мова для опису потоків даних, званий Pig Latin;

2) виконавче середовище для запуску програм Pig Latin. Сьогодні доступні два варіанти: локальне виконання на одній JVM і розподілене виконання в кластері Hadoop.

Програма *Pig Latin* складається із серії операцій (перетворень), які застосовуються до вхідних даних для отримання вихідних даних. Загалом ці операції описують потік даних, який перетворюється виконавчим середовищем Pig, де виконувалось уявлення, а потім запускається для виконання. У внутрішній реалізації Pig трансформує перетворення в серію завдань *MapReduce*, однак ця трансформація загалом залишається прихованою від програміста, що дає змогу йому зосередитися на даних, а не на природі виконання.

Однією з цілей розробки *Pig* була хороша розширюваність. Налаштовуються практично всі стадії обробки: завантаження, зберігання, фільтрація, угруповання, з'єднання – кожна операція може бути змінена за допомогою призначених для користувача функцій (*UDF, User-Defined Function*). Ці функції працюють із вкладеною моделлю даних Pig, тому вони можуть досить глибоко інтегруватися з операторами *Pig*. Крім того, призначені для користувача функції зазвичай краще підходять для повторного використання, ніж бібліотеки, розроблені для написання програм *MapReduce*.

Основний об'єкт у *Pig Latin* – це «відношення». Саме з відносинами працюють усі оператори мови. У формі відносин представляються вхідні й вихідні дані.

Кожне відношення являє собою набір однотипних об’єктів – *кортежів (tuples)*. Аналоги в БД: кортеж – це рядок, ставлення – це таблиця. Кортежі зі свого боку об’єднаються в колекції, які мають назву bag.

Кортежі, відповідно, складаються з нумерованих або іменованих об’єктів: «полів», довільних базових типів (число, рядок, булева змінна тощо).

2.1.4. Hive: SQL структура для організації сховищ даних

Hive – це пакет для організації сховищ даних, побудований на базі Hadoop. Орієнтований він на аналітиків, які впевнено володіють SQL і потребують засобів для виконання довільних запитів, агрегування й аналізу даних, обсяг яких такий, що потрібно Hadoop. Для взаємодії з Hive застосовується SQL-схожа мова запитів HiveQL.

Hive видає із себе движок, який перетворює SQL-запити в ланцюжок map-reduce завдань. Движок мчстить такі компоненти, як Parser (розбирає SQL-запити, які отримує на вхід), Optimizer (оптимізує запит для досягнення більшої ефективності), Planner (планує завдання на виконання) Executor (запускає завдання на фреймворку MapReduce (рис. 2.4).

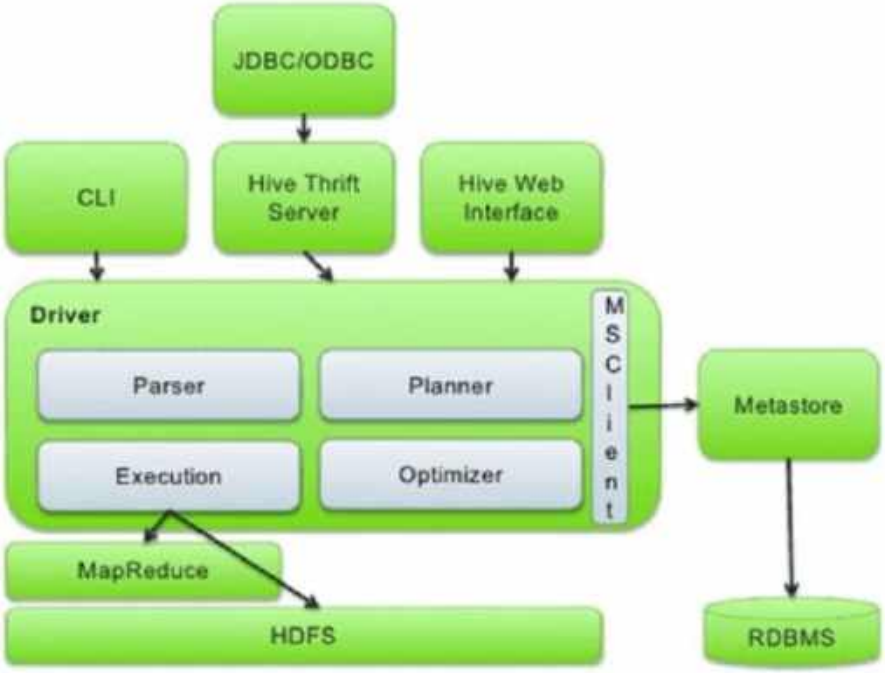


Рисунок 2.4 – Структурна схема Apache Hive

Для роботи *Hive* також потрібно сховище метаданих. Справа в тому, що SQL передбачає роботу з такими об'єктами, як база даних, таблиця, колонки, рядки, клітинки. Оскільки самі дані, які використовує *Hive*, зберігаються просто як файли на HDFS, то потрібно десь зберігати відповідність між об'єктами Hive і реальними файлами. Зазвичай мета-сховище розміщується в реляційній базі даних.

Pig і Hive мають приблизно однакові можливості й створені для однієї і тієї саме мети – абстрагувати розробника від безпосередньої розробки програм у рамках *MapReduce*. Але, однак, кілька відмінностей існує:

– Pig реалізує процедурний підхід, тоді як Hive використовує декларативний SQL-підхід. Тому під час використання складних вибірок запити в HQL стають досить громіздкими, а Pig дає змогу розбивати логіку на блоки, кожен крок можна розгорнуто описувати коментарями;

– розробка на Pig складніша та передбачає вивчення Pig Latin, тоді як HQL практично ідентичний усьому знайомому SQL.

Оскільки розвиток системи передбачає збільшення джерел даних і складність запитів буде тільки зростати, було вирішено спочатку використовувати *Apache Pig*.

2.1.5. HBase – розподілена СУБД зі зберіганням за стовпцями

HBase – набори даних розподілені на стовпці та орієнтовані на них і побудовані на основі HDFS. Цей компонент Hadoop використовується в ситуаціях, коли потрібно організувати довільний доступ для читання / запису до досить значних наборів даних у реальному часі [11; 12].

Додатки зберігають дані в таблицях, що складаються з рядків і стовпців. Для елементів таблиці (перетину рядків і стовпців) діє контроль версії. За стандартним налаштуванням як версії використовується тимчасова мітка, автоматично призначається HBase на момент вставки.

Вміст клітинки представляє неінтерпретувальний масив байтів. Ключі рядків таблиці також є байтовими масивами, тому теоретично ключем рядка може бути будь-що – від рядків до довічних уявлень long і навіть серіалізованих структур даних. Рядки таблиці упорядковані відповідно до ключу рядків (первинному ключу таблиці). Сортування здійснюється в порядку проходження байтів. Усі звернення до таблиці виконуються за первинним ключем.

Колонки організовані в групи колонок, звані Column Family. Зазвичай в одну Column Family об'єднують колонки, для яких однаковий патерн використання й зберігання. Записи фізично зберігаються у відсортованому за RowKey порядку. Водночас дані відповідні різним Column Family зберігаються окремо, що дає змогу в разі потреби читати дані тільки з відповідного сімейства колонок (рис. 2.5).

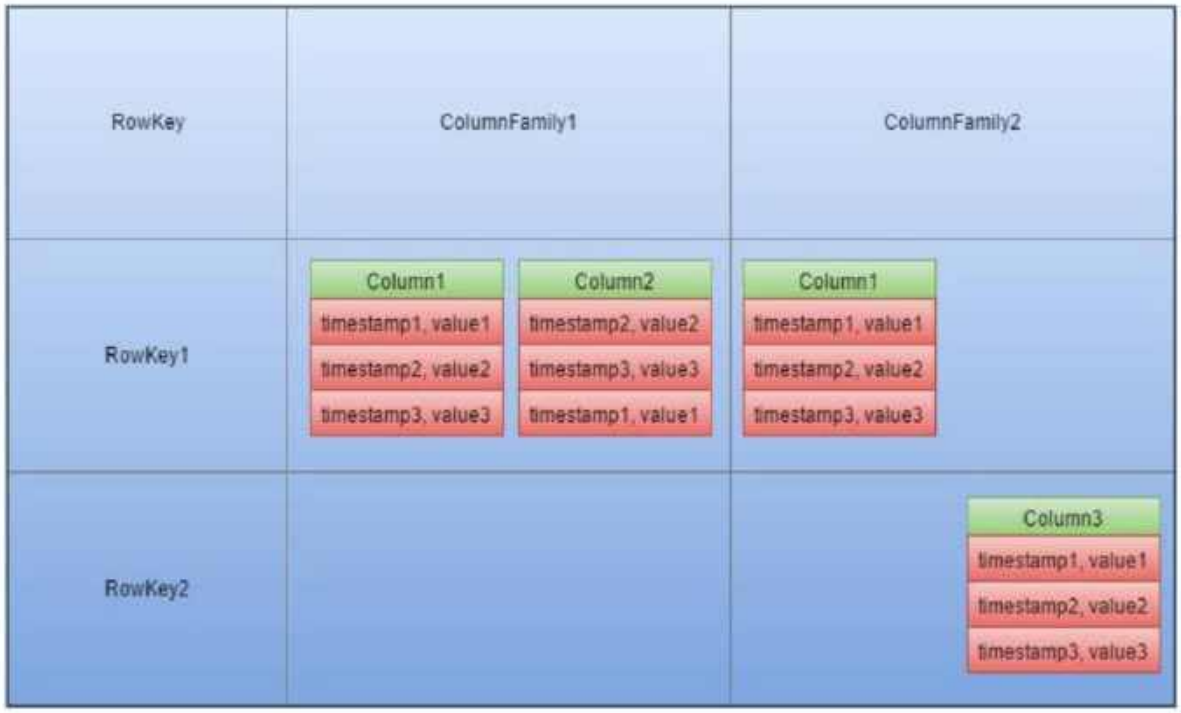


Рисунок 2.5 – Схема організації даних у HBase

HBase є розподіленою базою даних, яка може працювати на десятках і сотнях фізичних серверів, забезпечуючи безперебійну роботу навіть під час виходу з ладу деяких з них. Тому архітектура HBase є доволі складною порівняно з класичними реляційними базами даних.

HBase для своєї роботи використовує дві основні сутності.

Region Server – обслуговує один або кілька діапазонів записів, які відповідають певному діапазону ключів (Регіонів). Кожен регіон містить:

– основне сховище даних. HBase працює поверх HDFS, причому використовує власний формат файлів – HFile, в якому дані зберігаються у відсортованому за ключем у лексеграфічному порядку;

– буфер на запис. Оскільки дані зберігаються в HFile у відсортованому порядку, оновлювати ці файли на кожен запис досить дорого, тому дані під час запису потрапляють у спеціальну сферу пам'яті, де накопичуються та через деякий час записуються в основне сховище;

– кеш на читання. Дає змогу суттєво економити час на даних, які читаються часто;

– під час використання буфера на запис існує деякий ризик втрати даних через збій, тому для забезпечення відмовостійкості всі операції перед виконанням потрапляють у спеціальний журнальний файл – так званий Write Ahead Log.

Master Server – основний сервер у кластері HBase. Master управляє розподілом регіонів, веде їх реєстр, управляє запусками регулярних завдань.

Для координації дій між сервісами HBase використовує *Apache ZooKeeper* – спеціальний сервіс, призначений для управління конфігураціями та синхронізацією сервісів (рис. 2.6).

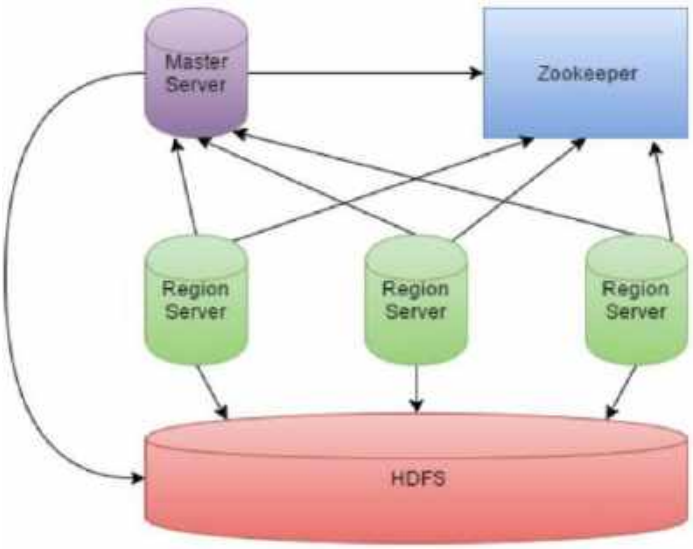


Рисунок 2.6 – Функціональна схема побудови кластера HBase

Під час збільшення кількості даних у регіоні й досягнення певного розміру HBase запускає split – операцію, що розбиває регіон на 2. Для того щоб уникнути постійних поділів регіонів, можна заздалегідь задати кордон регіонів і збільшити їхній максимальний розмір. HBase призначений загалом для читання й доступу до великих даних. Використання HBase виправдано, коли дані часто оновлюються та видаляються й потрібен доступ до даних за певними ключами.

2.1.6. Приклад розгортання віртуальної машини Oracle Big Data

У цьому прикладі показано розгортання віртуальної машини Oracle Big Data Lite 4.11 на локальному комп'ютері.

Відповідні ресурси. Для виконання лабораторних робіт з великих даних потрібні такі ресурси: Oracle Virtual Box; Oracle Big Data Lite VM; мінімум 16 ГБ оперативної пам'яті; увімкнена функція Virtual Assist в BIOS; 95 ГБ дискового простору для завантаження та встановлення VM; завантажений 7Zip.

Для розгортання виконаємо такі процедури:

1. Завантаження та інсталяція віртуальної машини.
2. Налаштування віртуальної машини.
3. Робота з образом віртуальної машини.
4. Налаштування Hive в образі віртуальної машини.
5. Після виконання роботи дати відповідь на питання самоконтролю.

Порядок виконання

1. Завантаження та встановлення віртуальної машини

Завантажити віртуальну машину Oracle Big Data Lite VM та інструкції за посиланням [13]: Oracle Big Data Lite Virtual Machine v 4.11.

Розпакуйте заархівовані файли в каталог на локальній машині.

Для цього потрібен Virtual Box:

Oracle VM VirtualBox – Downloads | Oracle Technology Network | Oracle

Встановіть Oracle Virtual Box на машину та пакет розширень. Коли з'явиться відповідний запит, завантажте файли уроків. Розпакуйте BigData_Lesson_files.7z на флешку або в локальну папку на комп'ютері.

Інсталяція віртуальної машини. Відкрийте Oracle Virtual Box, клацніть по файлу образу віртуальної машини й перейдіть до Preferences. На вкладці «General» встановіть каталог віртуальної машини за стандартним налаштуванням. Потрібно буде 65 ГБ вільного місця. Натисніть *ОК*.

Початок роботи з образом віртуальної машини. У вікні Oracle VM VirtualBox Manager / File натискаємо на Import Appliance і переходимо до місця, куди розпакували файл Big Data Lite 411. Потім виберіть імпорт; натисніть *Далі*. Вибираємо налаштування за стандартним налаштуванням і *імпорт*. Погоджуємось із ліцензією та натискаємо *ОК*.

2. Налаштування віртуальної машини

У вікні, що з'явиться, виберіть образ Big Data Lite 411.ova і натисніть *Setting* (рис. 2.7).

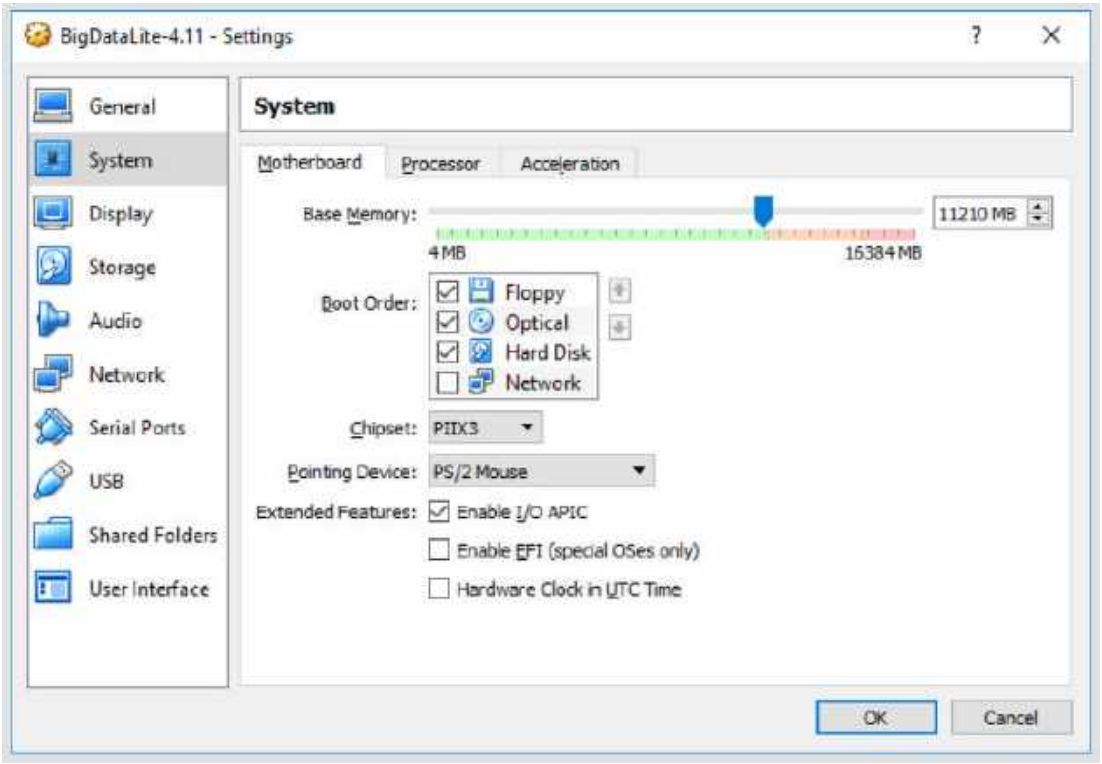


Рисунок 2.7 – Налаштування віртуальної машини

У розділі System Setting у Virtual box збільшуємо обсяг оперативної пам’яті до максимально дозволеного обсягу, що не впливає на продуктивність системи хосту. Потрібно мінімум 11 000 МБ (рис. 2.7).

Для роботи із зовнішніми файлами потрібно налаштувати доступ до зовнішніх файлів (оригінальна інструкція за посиланням вище [9]).

3. Робота з образом віртуальної машини

Натисніть кнопку *Start*, щоб завантажити образ у Virtual box, під час входу вкажіть ім’я користувача: *oracle*, пароль: *welcome1* (якщо не вказано інше). Після цього потрапите на робочий стіл Linux.

Оновіть гостьові додатки, коли з’явиться відповідний запит.

Двічі клацніть документ *Start Here* на робочому столі, у якому буде показано детальну інформацію про встановлене програмне забезпечення, ідентифікатори, паролі тощо.

Натисніть на запуск / зупинку послуг Start / Stop Services.

Переконайтесь, що служби відповідають рис. 2.8. Використовуйте пробіл для ввімкнення / вимкнення послуг.

Натисніть кнопку *OK*.

Натисніть на кнопку *Refresh Samples*, щоб оновити файли. На запит про продовження введіть «*y*». У відповідь на запит про налаштування проксі повинні мати можливість ввести «*n*». Нові приклади будуть завантажені в /home/oracle/src. Натисніть *Enter*, щоб вийти з терміналу.

Щоб зміни набули чинності, потрібно перезапустити віртуальну машину: System / Shut Down / Restart.

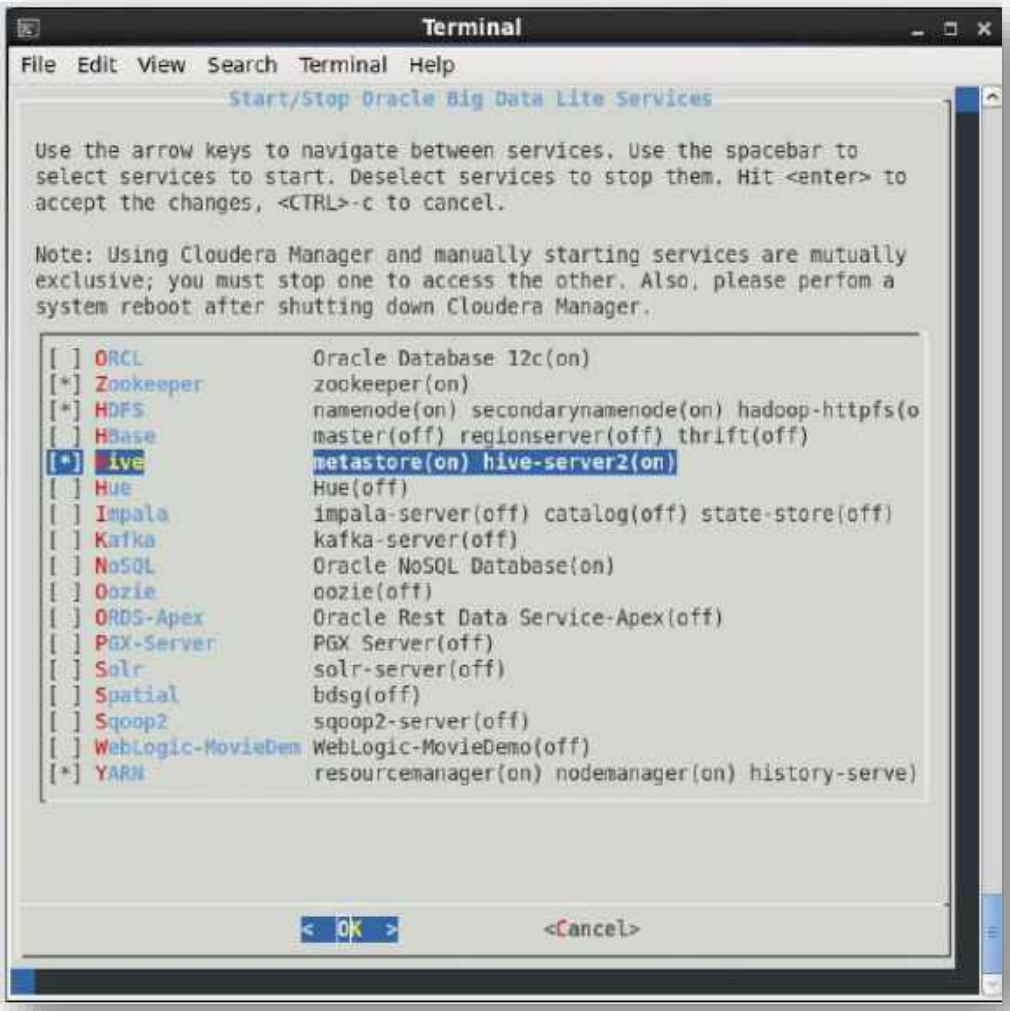


Рисунок 2.8 – Служби Oracle Big Data Lite 4.11

4. Налаштування Hive в образі віртуальної машини

Для функціонування Hive потрібно встановити драйвери JDBC у системі. Якщо ви скопіювали файли уроків на флешку, то:

– двічі клацніть *Computer* на робочому столі. Двічі клацніть по іконці USB-накопичувача. Клацніть правою кнопкою миші та скопіюйте каталог Cloudera_HiveJDBC4_2.5.15.1040 і можете вставити його в домашній каталог Oracle;

– клацніть правою кнопкою миші на скопійованому файлі й виберіть *extract here.* Перейдіть до нового каталогу 2.5.15.1040 GA і виберіть Cloudera_HiveJDBC4_2.5.15.1040 і знову розпакуйте *extract here.*

Для використання драйверів Cloudera Hive JDBC потрібно оновити Oracle SQL Developer.

Інструкція щодо завантаження та інсталяції Oracle SQL Developer: https://docs.oracle.com/cd/E12151_01/doc.150/e12153/install.htm#CIHFHIAH

Дистрибутив Oracle SQL Developer: https://www.oracle.com/database/sqldeveloper/technologies/download/

Запустіть Oracle SQL Developer. Перейдіть до Preferences / Navigate to Database / Third Party JDBC Drivers.



Рисунок 2.9 – Додавання jar-файлів

Перейдіть до Database / third Party JDBC Drivers.

Додайте всі jar-файли окремо, які містяться в папці Cloudera_HiveJDBC4_2.5.15.1040, до каталогу Third-party JDBC Driver Path. Це має виглядати, як показано на рис. 2.9.

Ваша віртуальна машина готова. Тепер можете виконувати лабораторні роботи, щоб отримати практичний досвід роботи з Big Data.

Питання для самоконтролю

1. Поясніть сутність і особливості технології великих даних Hadoop.

2. Опишіть особливості Hadoop Distributed File System – розподіленої масштабованої файлової системи.

3. Опишіть своїми словами особливість сховища даних, реалізованого для Hadoop на Hive.

4. Опишіть особливості скриптової мови програмування Pig.

5. Опишіть особливості інструменту на базі командного рядка Sqoop.

6. Опишіть особливості координатора для різних екземплярів Hadoop ZooKeeper.

7. Опишіть особливості використання Hadoop.

8. Опишіть особливості високорівневої мови опису потоків даних Pig.

9. Опишіть особливість: SQL-схожої інфраструктури для організації сховищ даних.

10. Опишіть особливості розподіленої СУБД зі зберіганням за стовпцями HBase.

11. Опишіть особливості розподіленої системи обробки даних Spark.

12. Які і для чого потрібні ресурси для виконання лабораторних робіт з великих даних?

13. Що являє собою віртуальна машина? Як вона працює із операційною системою комп'ютера?

14. Для чого налаштовується Hive?

15. Які служби Oracle Big Data Service повинні бути задіяні для цього практикуму?

16. Що таке Oracle SQL Developer? Які його функціональні можливості?

2.2. Збір і оброблення Big Data: HDFS та Map Reduce

2.2.1. Розподілена файлова система HDFS

Експоненціальне зростання інформації привело до потреби в появі файлових систем, орієнтованих на забезпечення високої продуктивності, масштабованості, надійності й доступності. З'явилася розподілена файлова система *HDFS* – Hadoop Distributed File System (аналог GSF від Google). Проєкт активно підтримується й розвивається компанією Yahoo.

В основу *HDFS* закладено концепцію одноразового запису / багатократного читання як найефективнішу схему обробки даних. Набір даних зазвичай генерується або копіюється із джерела, після чого з ним виконуються різні аналітичні операції. У кожній операції задіюється більша частина набору даних (або весь набір), тому час читання всього набору даних важливіший за затримку читання першого запису.

Hadoop не потребує коштовного обладнання високої надійності. Система спроєктована для роботи на стандартному обладнанні з досить високою ймовірністю відмови окремих вузлів у кластері. Технологія *HDFS* спроєктована так, щоб у разі відмови система продовжувала роботу без помітного переривання.

У *HDFS* існує концепція блоку, розмір якого за стандартним налаштуванням становить 64 Мбайт. Файли *HDFS* розбиваються на блокові фрагменти, які зберігаються незалежно один від одного.

Абстракція блоків у розподіленій файловій системі має кілька переваг:

– розмір файл може перевищувати розмір окремого диска в мережі; блоки файлу мають можливість використання будь-яких дисків у кластері. Крім того, коли файл зберігається в кластері *HDFS*, блоки цього файлу можна розподілити по всім дискам кластера;

– блоки спрощують підсистему зберігання та мають фіксований розмір. Це дає змогу системі обчислити, яка кількість блоків зберігається на певному диску. Також блоки запобігають проблемам, пов'язаним зі зберіганням метаданих;

– блоки підходять для реплікації. За рахунок блоків відбувається поліпшення відмовостійкості та доступності системи. Кожен блок має кілька копій на окремих машинах (за стандартним налаштуванням 3). У разі недоступності будь-якого блоку відбувається зчитування копії цього блоку з іншого доступного місця.

Кластер *HDFS* має 2 види вузлів (рис. 2.10): *Name Node* (вузол імен) і *Data Node* (вузол даних). *Name Node* є керуючим вузлом, а *Data Nod*e – керованим.

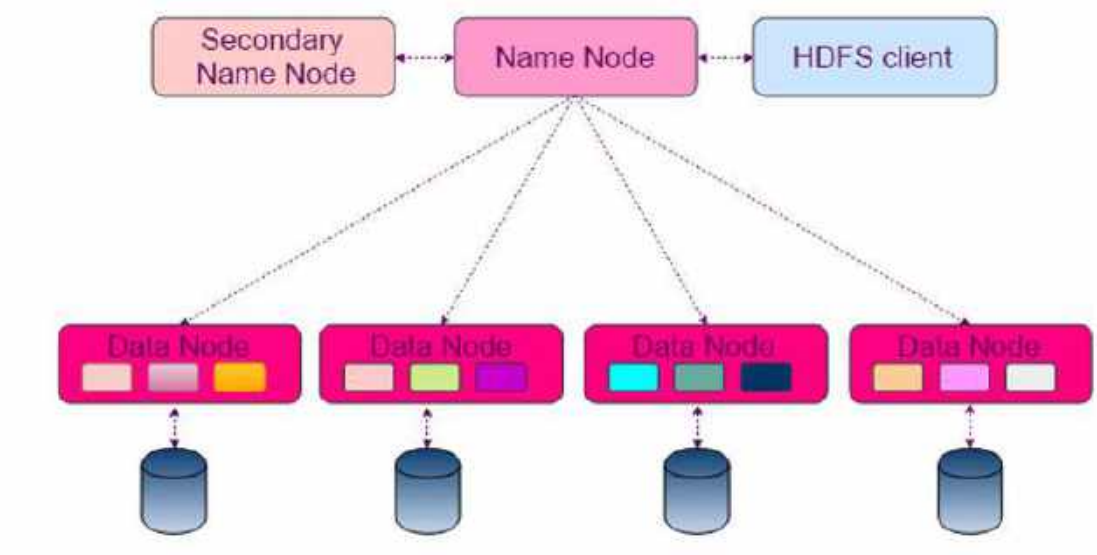


Рисунок 2.10 – Базова архітектура HDFS

Name Node займається управлінням простором вузлів імен файлової системи, підтримкою метаданих і каталогів у дереві файлової системи. Локальний диск зберігає інформацію у вигляді 2 файлів: журналу змін і образу файлової системи. Name Node містить інформацію про те, на яких саме вузлах перебувають усі блоки файлу, водночас інформація про місцезнаходження блоків будується заново під час запуску системи, а не зберігається постійно.

NameNode – основний сервер для управління простором імен файлової системи і доступом клієнтів до даних. Цей сервер під час першого запуску фіксує всі транзакції, пов’язані зі зміною метаданих файлової системи, у файлі під назвою *EditLog* і застосовує ці зміни до образу *HDFS*, розташованого у файлі *FsImage*. Потім записується вже новий образ із змінами, і система починає роботу з чистим *EditLog*.

Для звернення до файлової системи клієнт зв'язується з *Name Node* і *Data Node* вузлами. Інтерфейс файлової системи схожий з інтерфейсом Portable Operating System Interface. Отже, користувацький код може не володіти жодною інформацією про вузли *Name Node і Data Nod*e.

DataNode – сервер для безпосереднього зберігання даних. Передача даних відбувається безпосередньо між *DataNode* і клієнтом, щоб розвантажити *NameNode.*

Вузли даних виконують запис і читання блоків, коли цього вимагають клієнти або вузол імен. Крім того, ці вузли періодично передають список збережених ними блоків безпосередньо вузлу імен. Вузли, будучи частиною системи, впливають на працездатність файлової системи. У разі ліквідації машини, на якій працює *Name Node*, відбудеться втрата всіх даних. Саме з цієї причини стійкість вузла імен до можливих збоїв важлива.

Hadoop надає два способи для забезпечення стійкості:

1. Архівація файлів, які визначають стійкий стан метаданих файлової системи. Є можливість налаштувати Hadoop так, щоб Name Node був здатний на запис свого стійкого стану в кілька файлових систем. Операції запису виконуються синхронно та атомарно.

2. Запуск вторинного вузла імен (Secondary NameNode), який не виконує функції вузла імен. Його основною функцією є періодичне злиття образу простору імен із журналом змін, щоб уникнути надмірного розростання останнього.

Secondary NameNode – сервер, який виконує функції *NameNode*, що стосуються файлів *EditLog* та *FsImage*. Водночас ці файли зберігаються на основному сервері.

Вторинний вузол імен зазвичай працює на окремій фізичній машині, оскільки для того, щоб виконати злиття, потрібні значна обчислювальна потужність і така сама кількість пам'яті, як на вузлі імен. Вторинний вузол імен має копію об'єднаного образу простору імен, який може бути використаний у разі збою основного вузла. Однак стан вторинного вузла імен дещо відстає від стану основного вузла. У разі повної відмови основного вузла не є можливим запобігти втраті даних.

Дані в *HDFS* зберігаються як блоки (блок – одиниця зберігання файлів) на *DataNode* та управляються через *NameNode.* Причому під час запису файлу відбувається процес реплікації (копіювання) цього файлу. Стандартний розмір блоків у HDFS – 64 МБ або 128 МБ (рис. 2.11).

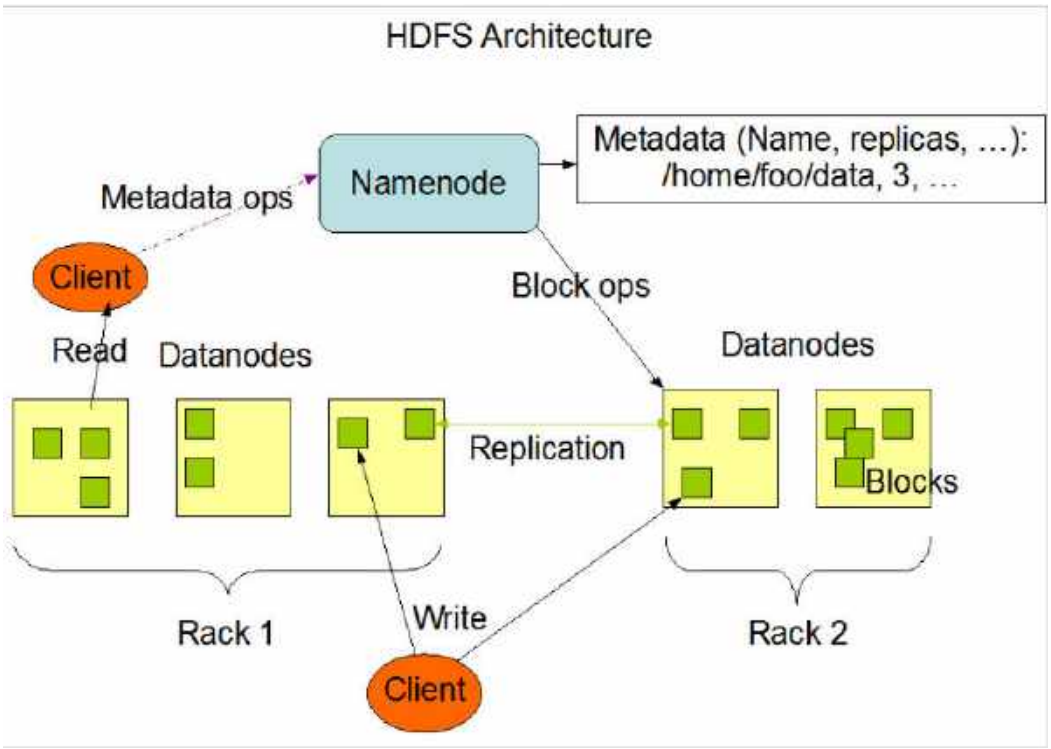


Рисунок 2.11 – Архітектура й блоки HDFS

Основним мотивом для визначення такого розміру є зменшення часу пошуку даних порівняно з їхньою передачею. Водночас відбувається процес реплікації (копіювання) даних.

Копії блоків зберігаються на декількох серверах, за стандартним налаштуванням – на трьох. Їхнє розміщення відбувається так:

– перша репліка – на локальному DataNode;

– друга репліка – на іншому DataNode в цій самій стійці;

– третя репліка – на довільній DataNode іншої стійки;

– решта репліки розміщується довільним способом.

Під час читання даних клієнт вибирає найближчий до нього *DataNode* з реплікою.

На початку 2000-х корпорація Google зрозуміла, що їй буде потрібна нова система керування базами даних, щоб каталогізувати всі дані у Всесвітній павутині. Щоб виконати це завдання, Google вирішила розробити розподілену обчислювальну систему,

що складається із кластера серверів, використовуючи єдину файлову систему, розподілену на кількох серверах, де кожен сервер ділиться частиною обробки.

Відповідно, *HDFS* – це файлова система, у якій Hadoop зберігає дані. *HDFS* не замінює файлову систему Linux на окремих серверах, а розташовується поверх кластера серверів як єдина файлова система, що охоплює весь кластер. *HDFS* зберігає дані у фрагментах по 64 Мб, використовуючи принаймні три сервери *DataNode*, як показано на рис. 2.12.

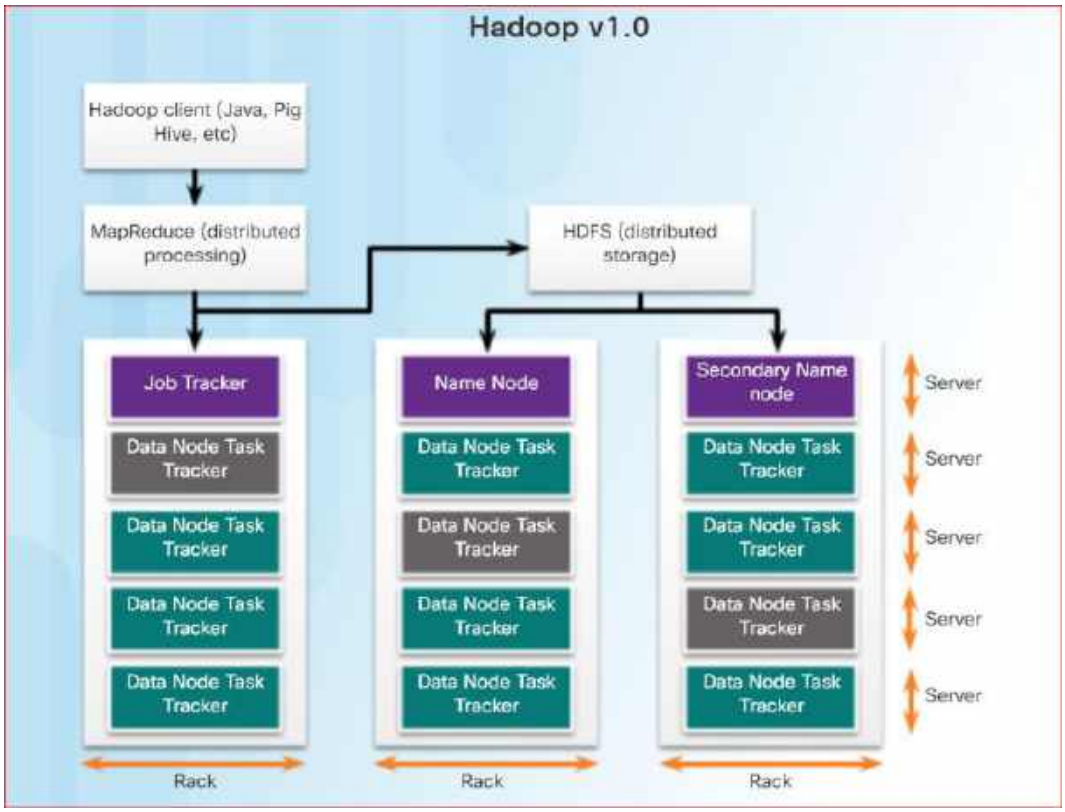


Рисунок 2.12 – Мінімальна архітектура Hadoop 1.0 на базі HDFS

HDFS керує інформацією в кластері з централізованого координаційного сервера під назвою *NameNode.* Сервер NameNode відстежує, які дані зберігаються на різних серверах *DataNode.* Коли дані надходять у систему, вони імпортуються до *NameNode*, потім *NameNode* розділяє дані на фрагменти по 64 Мб, які будуть дублюватися в трьох або більше *DataNode* залежно від конфігурації. Ця *надлишковість* забезпечує відмовостійкість, схожу до дзеркального масиву RAID. У разі якщо один *DataNode* вийшов з ладу, сервер *DataNode* може бути замінений, а файлова система та дані будуть відновлені з повторюваних *DataNodes.*

Проблема прийому даних

У своїй основній формі конвеєр великих даних складається з трьох компонентів: прийом даних (*ingest*), зберігання даних (*store*) і обробка даних (обчислення, *compute*), як показано на рис. 2.13.



Рисунок 2.13 – Проблеми збору й обробки Big Data

Для кожного з цих компонентів існує багато програмних платформ, які використовуються для виконання кожного завдання.

Залежно від типу даних, типу надходження та вимог до обчислень компоненти кожного конвеєра даних є унікальними, їх часто об’єднують і налаштовують для роботи один з одним.

Перший етап конвеєра великих даних, прийом даних, викликає багато проблем, які потрібно подолати. Великі дані часто надходять із різних джерел. Значна частина цих даних передається в режимі реального часу. Масштабованість також є проблемою. Якщо більше пристроїв підключаються, то більше даних потрібно отримати. Якість цих даних теж є проблемою. Дані можуть бути структурованими, неструктурованими або мати досить складний формат.

2.2.2. Kafka – розподілена потокова платформа

Щоб отримати дані в режимі реального часу, потрібно використовувати розподілену потокову платформу, зокрема *Kafka.*

Деякі з характеристик *Kafka*:

– потоки записів публікуються та підписуються на них (pub-sub) аналогічно до корпоративної системи обміну повідомленнями;

– потоки записів є відмовостійкими завдяки розподіленому сховищу;

– потоки записів обробляються по мірі їхнього виникнення в режимі реального часу;

– *Kafka* використовується для передачі потокових даних у реальному часі між різними системами та програмами;

– *Kafka* також використовується для перетворення та реагування на потоки даних через програми реального часу;

– *Kafka* була розроблена *LinkedIn*, але 2011 року стала програмним забезпеченням з відкритим вихідним кодом.

Cisco Systems, Netflix і eBay – це лише деякі з компаній, які використовували або нині використовують Kafka.

Ось кілька важливих основних понять про *Kafka*.

Kafka працює на одному або кількох серверах як кластер. Сервери в кластері мають назву брокери. Кластер зберігає потоки записів у групах, які називаються темами. Кожен запис має ключ, значення та мітку часу. Kafka має чотири основні API, як показано на рис. 2.14.

Producers (Виробники) – це API, де програма публікує потік записів у темах *Kafka;*

Stream Processors (Потокові процесори) – це API, де програма споживає вхідні потоки з тем і може створювати вихідні потоки для тем;

Connectors (Конектори) – це API, де теми *Kafka* підключаються до існуючих систем і програм;

Consumers (Споживачі) – це API, де створюється потік записів у темах.

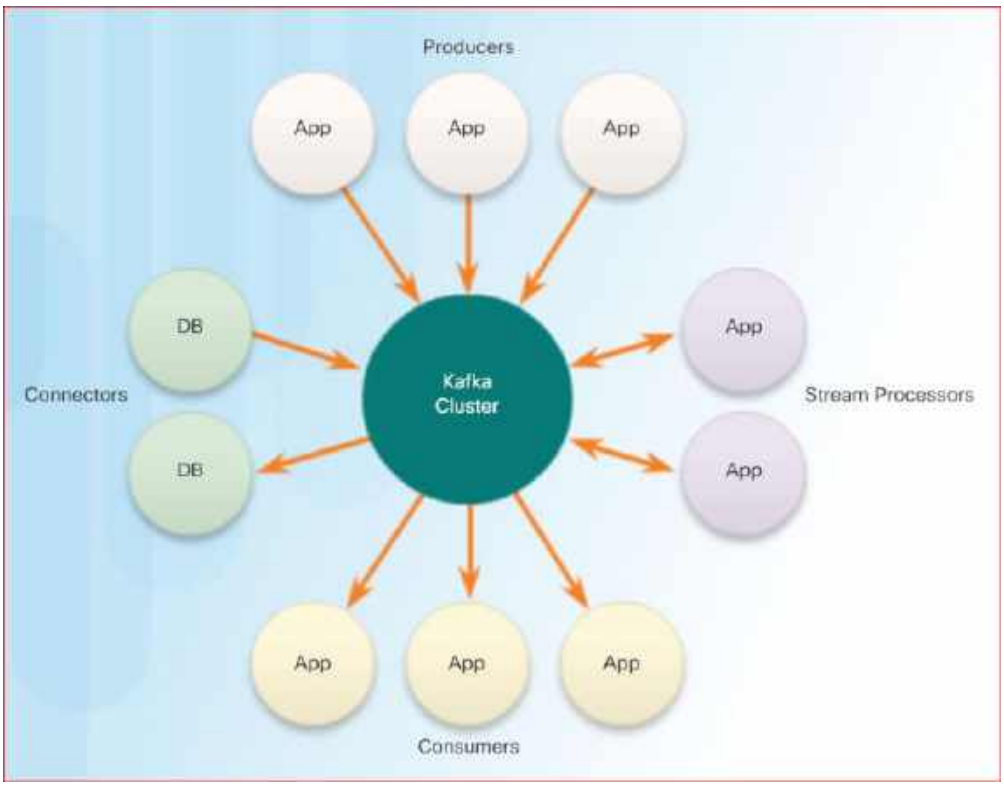


Рисунок 2.14 – Чотири основні API розподіленої потокової платформи Kafka

Kafka діє як система обміну повідомленнями для централізації зв'язку між усіма виробниками даних і всіма споживачами даних у системі. *Kafka* має розподілений дизайн, що дає змогу обробляти значні обсяги даних і легко масштабуватися. Він також стійкий до збоїв обладнання, програмного забезпечення та мережі.

Переваги *Kafka* порівняно з іншими підходами.

Kafka – це набагато більше, ніж просто сервер обміну повідомленнями. Kafka працює так, як розподілена база даних. Повідомлення, написані *Kafka*, копіюються на багато серверів і записуються на диск. За бажанням сховище даних може тривати вічно. Завдяки такому розподіленому дизайну *Kafka* має високу доступність, підтримує автоматичне відновлення та досить стійкий до збоїв мережі.

Те, що відрізняє *Kafka* від традиційних брокерів повідомлень, це використання журналів транзакцій. Кожна тема складається з набору журналів, які називаються розділами. Виробники додають ці журнали, а споживачі можуть читати журнали, коли їм це потрібно. Теми – це дані, які тиражуються багатьом брокерам для досягнення відмовостійкості. У *Kafka* немає жодної точки провалу.

Kafka здатний керувати досить високою пропускною здатністю. Це пов'язано з тим, що багато обов'язків покладено на виробників і споживачів. Це робить брокерів легкими, а *Kafka* – бажаним інструментом для багатьох платформ IoT. Завдяки значній кількості даних, що надходять від датчиків та інших пристроїв у режимі реального часу, і бажанню проаналізувати ці дані без втрати інформації *Kafka* може регулювати це надходження та надавати дані кільком споживачам одночасно.

2.2.3. Файлова система Cassandra та її порівняння з Hadoop

Проєкт *Apache Hadoop* визначає HDFS як «основну систему зберігання, що використовується програмами *Hadoop*, яка забезпечує «надійні, надзвичайно швидкі обчислення». HDFS використовує архітектуру головний-підлеглий. Основний *NameNode* у кластері регулює файлову систему та доступ до даних клієнтами. У кластері також є *DataNodes*, часто одна фізична машина для обробки підключеного сховища. Дані зберігаються у файлах, розділених на блоки та розподілених по *DataNodes*. За стандартним налаштуванням HDFS копіює ці блоки на два додаткові сервери. Той факт, що HDFS має такий тип автоматичного резервування, робить його популярним вибором для пакетної аналітики сховищ даних.

Cassandra використовує файлову систему Cassandra (CFS). CFS не є архітектурою головний-підлеглий як HDFS. Кожен вузол у кластері однаковий, однорангова реалізація. Кластери зберігають дані в реальному часі, і над ними можна виконувати аналітичні операції. Вбудована реплікація *Cassandra* копіює дані між усіма вузлами реального часу, аналітичними та пошуковими вузлами.

У CFS аналітичні метадані зберігаються в просторі ключів, схожому до бази даних у RDBMS. Є два сімейства стовпців, схожі до таблиць RDBMS, які містять дані. Дані в цих сімействах стовпців реплікуються в кластері для відмовостійкості та захисту даних. Дві родини стовпців схожі до двох основних служб HDFS. Сімейство стовпців *inode* замінює службу *NameNode* HDFS. Інформація в цьому сімействі даних – це такі речі, як користувач, батьківський шлях, ім'я файлу та ідентифікатори блоку, з яких

складається файл. Сімейство стовпців *sblocks* займає місце служби HDFS *DataNode*. Це сімейство стовпців зберігає вміст будь-якого файлу. Модель даних CFS показана на рис. 2.15.

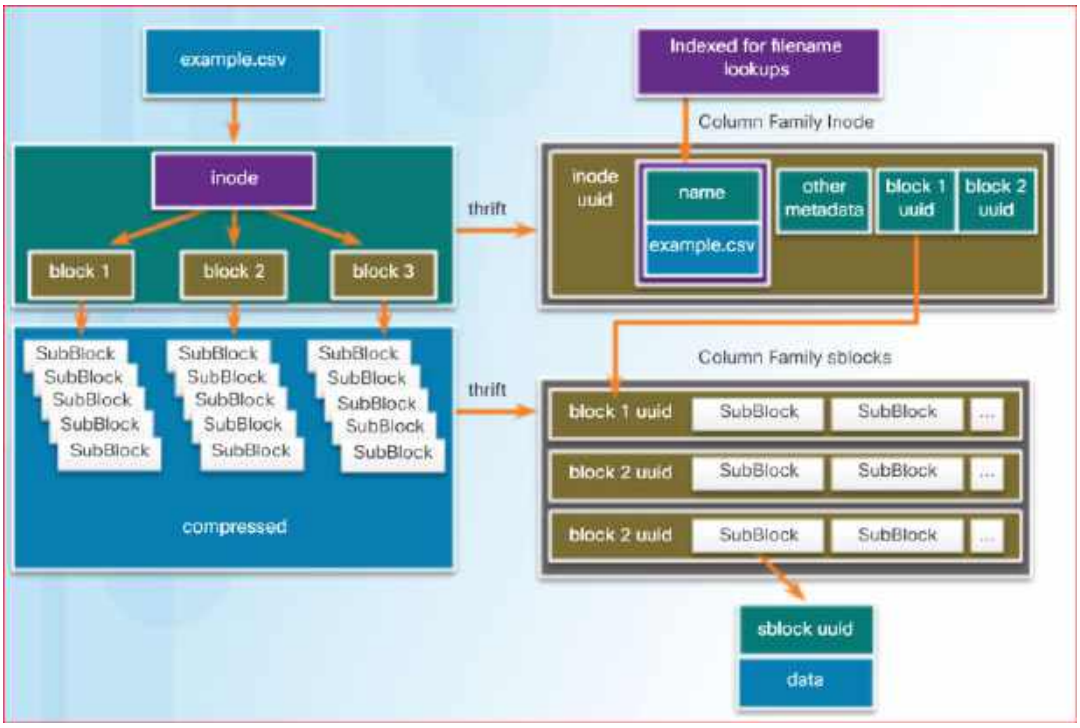


Рисунок 2.15 – Файлова система Cassandra (CFS)

Переваги використання CFS над HDFS

Краща доступність – не потрібне рішення для спільного зберігання. Чим більше вузлів і кластерів, тим краща доступність. Це також можна покращити через збільшення коефіцієнта реплікації, який визначає, скільки вузлів отримають репліковані дані.

Базова апаратна підтримка – для CFS не потрібні спеціальні сервери та спеціальні мережеві пристрої.

Автоматичне перемикання після відмови – як і у випадку з доступністю, перемикання після збоїв відбувається автоматично через реплікацію. Вузли або кластери, навіть центри обробки даних можуть вийти з ладу, а дані на вузлах і кластерах залишаться доступними в інших місцях.

Інтеграція даних – усі дані, які записуються в Cassandra, реплікуються як на аналітичні, так і на пошукові вузли. Це дає змогу виконувати аналітику, завдання корпоративного пошуку та завдання в реальному часі одночасно, не впливаючи один на одного.

Простіше розгортання – кластери легко налаштувати і їх можна запустити лише за кілька хвилин. Немає складних вимог до зберігання та конфігурації головний-підлеглий.

Підтримує кілька центрів обробки даних – CFS може запускати одну базу даних у кількох центрах обробки даних. Доступні інструменти, які дають змогу кожному центру обробки даних мати локальну копію всіх даних у базі даних. Аналітичні завдання можна виконувати в кількох центрах обробки даних одночасно.

HDFS є чудовим вибором, коли для програми Hadoop потрібне недороге рішення для зберігання даних із зосереджених на сховищі даних. CFS може запускати аналітику даних, що надходять із великих програм, які інтегруються в бази даних і СУБД.

2.2.4. MapReduce як технологія розподілених обчислень

MapReduce є технологією для розподілених обчислень, яка була реалізована в декількох системах, зокрема у внутрішній реалізації компанії Google і популярній системі з відкритим кодом Hadoop, який може бути отриманий разом із файловою системою *HDFS* [11; 12].

Можна використовувати реалізацію MapReduce для управління багатьма великомасштабними обчисленнями так, щоб апаратні несправності не були проблемою. Усе, що потрібно розробнику, це написати дві функції, які називаються Map та Reduce, тоді як система самостійно управляє паралельним виконанням, координацією завдань, які виконуються, а також контролює ситуації, коли одне з цих завдань не має змоги бути виконаним. Схема обчислень зображена на рис. 2.15.

Загальний порядок обчислень *MapReduce* можна пояснити так.

Частина даних разом з функцією Map відправляється на один з обчислювальних вузлів на виконання. Виходом цієї програми з одного кластеру є пари ключ-значення.

Ці пари ключ-значення з кожного завдання збираються на основному контролері й упорядковуються відповідно до ключа.

Функція Reduce виконує обробку даних, упорядкованих до ключа. Спосіб обробки визначає код, написаний користувачем для функції Reduce.

Зрозуміло, що компоненти мережі можуть давати збій, виходити з ладу, і чим більше обчислювальних вузлів і з'єднань має система, тим більша ймовірність збоїв у системі. А важливі розрахунки можуть виконуватись хвилини або й навіть години на тисячах обчислювальних вузлах. Зрозуміло, що перезапуск обчислення щоразу стосовно появи неполадок одного із елементів унеможливив би цей підхід. Вирішення цієї проблеми має дві форми:

1) файли повинні мати резервні копії. Відсутність дублювання файлів на декількох обчислювальних вузлах призвело б до втрати файлів назавжди;

2) розрахунки повинні бути розділені на завдання так, що якщо одне завдання не може бути виконаним до кінця, воно може бути перезапущена, не впливаючи на інші завдання.

Для того щоб використовувати кластерні обчислення, файли повинні виглядати трохи інакше, ніж у звичайних файлових системах, які розміщені на окремих комп'ютерах. Ця нова файлова система, яку часто називають розподіленою файловою системою або DFS, зазвичай використовується так [10; 11].

Файли розділені на частини за розміром у 64 МБ. Ці частини реплікуються, наприклад три рази в три різних обчислювальних вузли. Крім того, кожна частина повинна бути розташована на різних стійках, тому не втратимо всі копії через збій. Зазвичай такі параметри, як розмір блоку й ступінь реплікації задаються користувачем. Для того щоб знайти частини файлу, існує ще один невеликий файл, який називається головний вузол. Головний вузол самостійно реплікується, і каталог для файлової системи знає, де знайти його копію. Сам каталог може бути відтворений, і всі учасники, що використовують DFS, знають, де перебуває копія каталогу (рис. 2.16).

Вхідні файли для завдання *Map* складаються з елементів, які можуть бути будь-якого вигляду: масив, документ тощо. Частина являє собою сукупність елементів, і жоден з елементів не зберігається в двох частинах даних. Технічно всі входи до завдання *Map* і виходи з *Reduce* завдань мають вигляд пар ключ-значення, але зазвичай ключі вхідних елементів не мають значення і їх можна ігнорувати. Ця форма входів і виходів мотивована бажанням дати

змогу композиції з декількох процесів *MapReduce.* Функція *Map* приймає вхідний елемент як аргумент і виробляє нуль або більше пар ключ-значення. Типи ключів і значень кожен вузол генерує довільно. Крім того, ключі не повинні бути унікальними. Найчастіше за одне завдання функція *Map* може привести кілька пар ключ-значення з тим самим ключем, навіть з того самого елемента. Варто зазначити, що одне завдання *Map* зазвичай має обробляти значну кількість документів – всі документи перебувають в одній або декількох частинах даних.

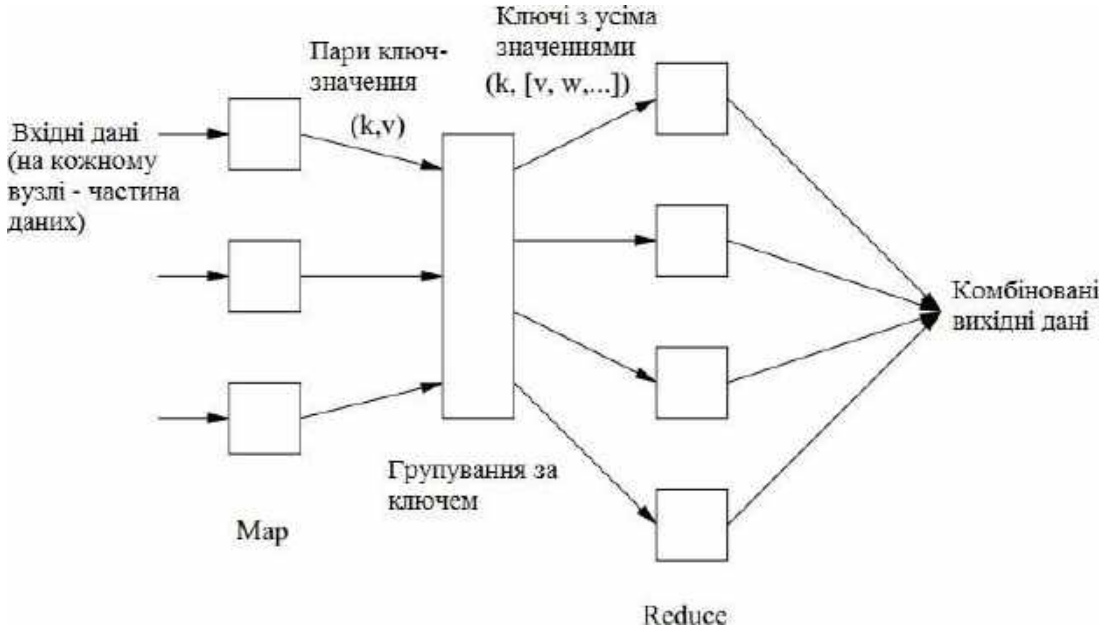


Рисунок 2.16 – Схема обчислень з використанням MapReduce

Як тільки завдання *Map* успішно завершилось, пари ключа-значення групуються по ключу, а значення, пов’язані з кожним ключем, формуються в список значень. Угруповання здійснюється за допомогою системи, незалежно від того, що роблять *Map* та *Reduce* завдання. Процес головного контролера знає, скільки буде Reduce-завдань. Користувач зазвичай вказує системі *MapReduce*, що має бути певна кількість завдань (позначимо *r*), і тоді головний контролер вибирає хеш-функцію, яка застосовується до ключів, і виробляє ряд значень від 0 до r – 1. Кожен ключ, який виводиться за допомогою завдання *Map*, хешується і його пари ключ-значення поміщаються в одному з r-локальних файлів. Кожен файл призначений для одного з *Reduce*-завдань. Щоб виконати

угруповання по ключу і розподілити файли до *Reduce*-завдань, головний контролер об'єднує файли з кожного завдання *Map*, які призначені для конкретного *Reduce*-завдання, і подає об'єднаний файл у цей процес як послідовність пар ключ-список значень.

Аргументом функції *Reduce* є пара, що складається з ключа і його списку відповідних значень. Виходом функції *Reduce* є послідовність з нуля або більше пар ключ-значення. Ці пари ключ-значення можуть бути іншого типу, як і послані від завдань *Map*, але часто вони мають однаковий тип. Завдання *Reduce* отримує один або кілька ключів і пов'язані з ними списки значень. Тобто *Reduce* виконує одне або декілька завдань. Виходи з усіх *Reduce* завдань об'єднуються в один файл. Іноді функція *Reduce* має властивості асоціативності та комутативності. Тобто значення можуть бути об'єднані в будь-якому порядку, а результат буде незмінним.

На рис. 2.17 зображено схему, яка містить процеси, завдання й файли та їх взаємодію між собою. Користуючись бібліотеками, наданими системою *Hadoop MapReduce*, користувач програми має розгалуження процесів, а саме основний процес і деяку кількість робочих процесів на різних обчислювальних вузлах. Зазвичай на вузлах обробляються завдання *Map* або *Reduce*, але не обидва одночасно.

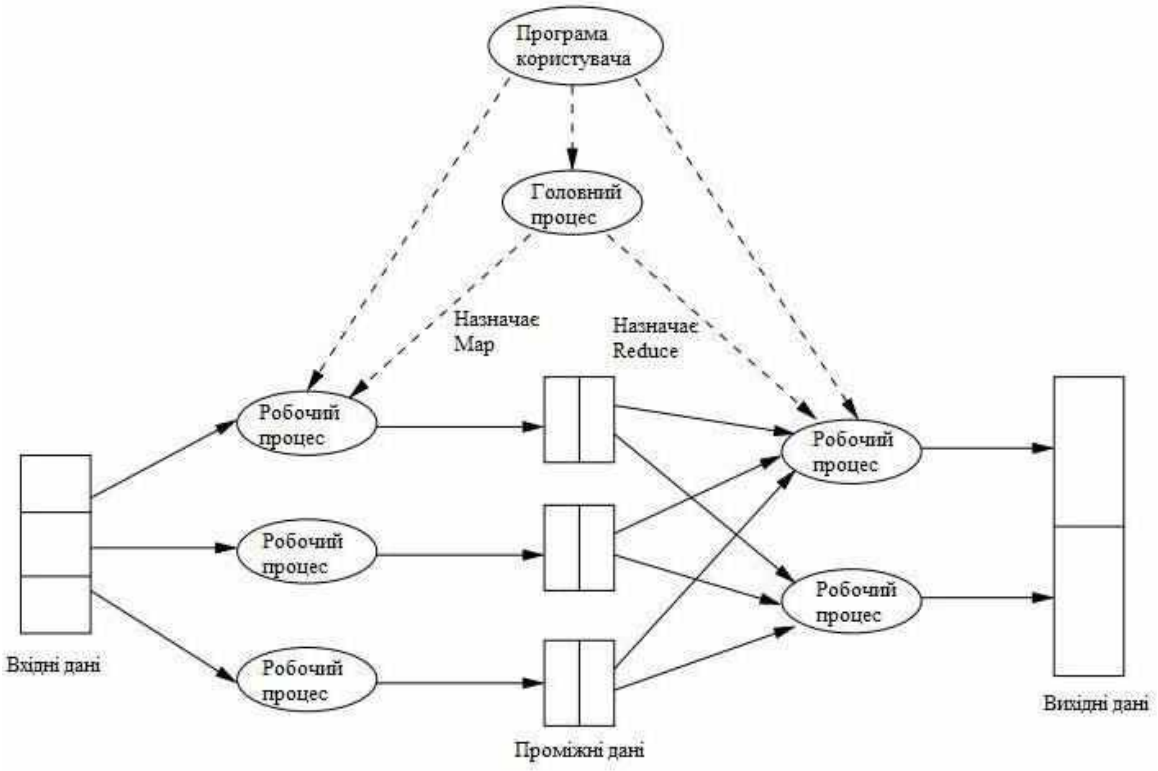


Рисунок 2.17 – Загальний огляд запуску програми MapReduce

Основний вузол має багато обов'язків. Одним з них є створення деякого числа завдань *Map* і деякого числа завдань *Reduce*. Ці цифри обираються в програмі користувачем. Завдання будуть призначені на робочі процеси за допомогою основного вузла. Розумно створити одне *Map*-завдання для кожного фрагмента вхідного файлу, але тоді варто буде створити менше *Reduce*-завдань. Причиною обмеження числа завдань є те, що потрібно для кожного завдання *Map* створити проміжний файл на вхід до завдання *Reduce*, і якщо є занадто багато *Map*-завдань, кількість проміжних файлів буде значно більша за кількість *Reduce*-завдань. Тому треба перед тим, як задати кількість завдань *Map* проаналізувати можливості завдань *Reduce.*

Основний процес відстежує стан кожного завдання *Map* і *Reduce* (очікування виконання на конкретному вузлі). Робочий процес підпорядковується головному, коли він закінчує завдання, тоді нове завдання може бути надане цьому робочому процесу.

Кожному завданню *Map* задається одна або кілька частин вхідного файлу(ів) і після цього на ньому виконується код, написаний користувачем. Завдання *Map* створює файл для кожного *Reduce*-завдання на локальному диску кластеру, який виконує завдання. Майстру повідомляється про місце й розмір кожного із цих файлів. Коли завдання *Reduce* призначається майстром до робочого процесу, то завданню даються всі відповідні файли, які формують його вхід. Завдання *Reduce* виконує код, написаний користувачем, і записує свій результат у файл, який є частиною розподіленої файлової системи.

Найгірше, що може статися, що обчислювальний вузол, на якому працює основний процес, перестає працювати. У цьому випадку вся *MapReduce* робота повинна бути перезапущена. Але тільки цей один вузол може завдати шкоду всьому процесу; інші відмови будуть управлятися цим вузлом, і робота *MapReduce* буде завершена без значних затримок.

Припустимо, що обчислювальний вузол, на якому виконується завдання *Map*, перестає працювати. Ця зупинка буде виявлена основним вузлом, тому що він періодично перевіряє стан робочих процесів. Усі завдання *Map*, які були призначені на цей

вузол, мають бути перероблені, навіть якщо вони були завершені. Причиною переробки завершених завдань *Map* є те, що їхній результат збережений саме на цьому вузлі. Основний вузол встановлює статус кожного із цих завдань *Map* як готовий до виконання на іншому вузлі.

Випадок з відмовою у вузлі *Reduce* обробляти простіше. Основний вузол просто перериває виконання на цьому вузлі та назначає виконання на іншому вузлі, як тільки той стає доступним.

2.2.5. Програмна модель MapReduce

MapReduce – програмна модель для виконання розподілених обчислень для значних обсягів даних, що являє набір Java-класів і виконуваних утиліт для створення й обробки завдань на паралельну обробку.

Основні принципи *MapReduce* можна сформулювати так:

– обробка та обчислення значних обсягів даних;

– масштабованість;

– автоматичне розпаралелювання завдань;

– робота на простому обладнанні;

– автоматична обробка відмов виконання завдань.

Роботу *MapReduce* можна поділити на такі етапи:

1. *Читання вхідних даних.* Вхідні дані діляться на блоки даних визначеного розміру (від 16 Мб до 128 Мб), які називаються сплити. *MapReduce* закріплює за кожною функцією *Map* певний сплит.

2. *Функція Map.* Кожна функція *Map* отримує на вхід список пар «ключ / значення», обробляє їх і на виході, отримує нуль або більше пар «ключ / значення» і є проміжним результатом. Усі операції *Map* виконуються паралельно та не залежать від результатів роботи один одного. Кожна функція *Map* отримує на вхід свій унікальний набір даних.

3. *Функція Reduce.* Програмна модель викликає функцію *Reduce* для кожного унікального ключа в списку значень. Операції *Reduce* виконуються паралельно й не залежать від результатів роботи один одного. Так, результати роботи кожної функції *Reduce* пишуться в окремий блок *HDFS.*

4. *Запис вихідних значень*. Результати, отримані на етапі *Reduce*, записуються у файлові блоки в HDFS. Кожен вузол *Reduce* пише у власний блок. На рис. 2.18 зображена схема роботи *MapReduce.*

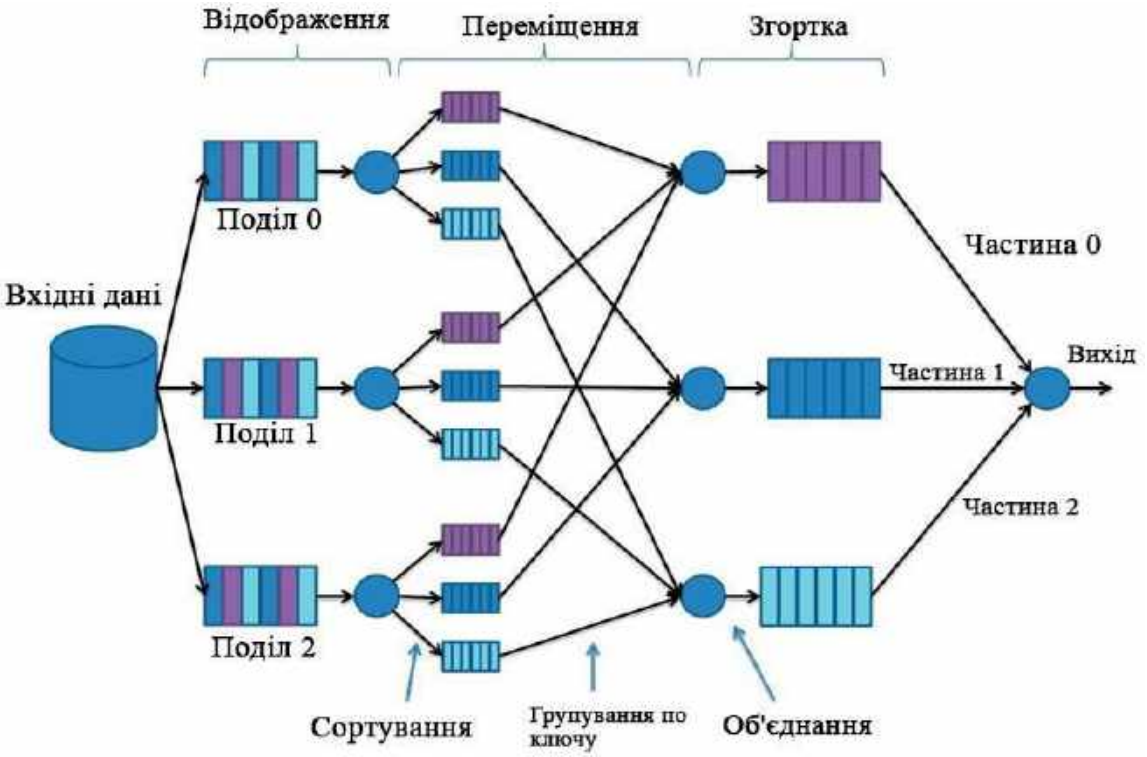


Рисунок 2.18 – Схема роботи MapReduce

Розробнику додатків для *Hadoop MapReduce* потрібно написати базовий обробник, який на кожному обчислювальному вузлі забезпечить освіту початкових пар «ключ / значення» в проміжний набір пар «ключ / значення» (клас, який реалізує інтерфейс *Mapper*) і обробник зводить проміжний набір пар в остаточний набір пар (клас, який реалізує інтерфейс *Reducer)*.

Усі інші етапи виконуються програмною моделлю без додаткових зусиль розробника. *Map Reduce* виконує такі функції:

– планування завдань;
– розпаралелювання завдань;
– перенесення завдань до даних;
– синхронізація виконання завдань;
– обробка відмов виконання завдань і їхній перезапуск;
– оптимізація мережевих взаємодій.

На рис. 2.19 показано приклад використання MapReduce для підрахунку кількості слів у потоці даних.

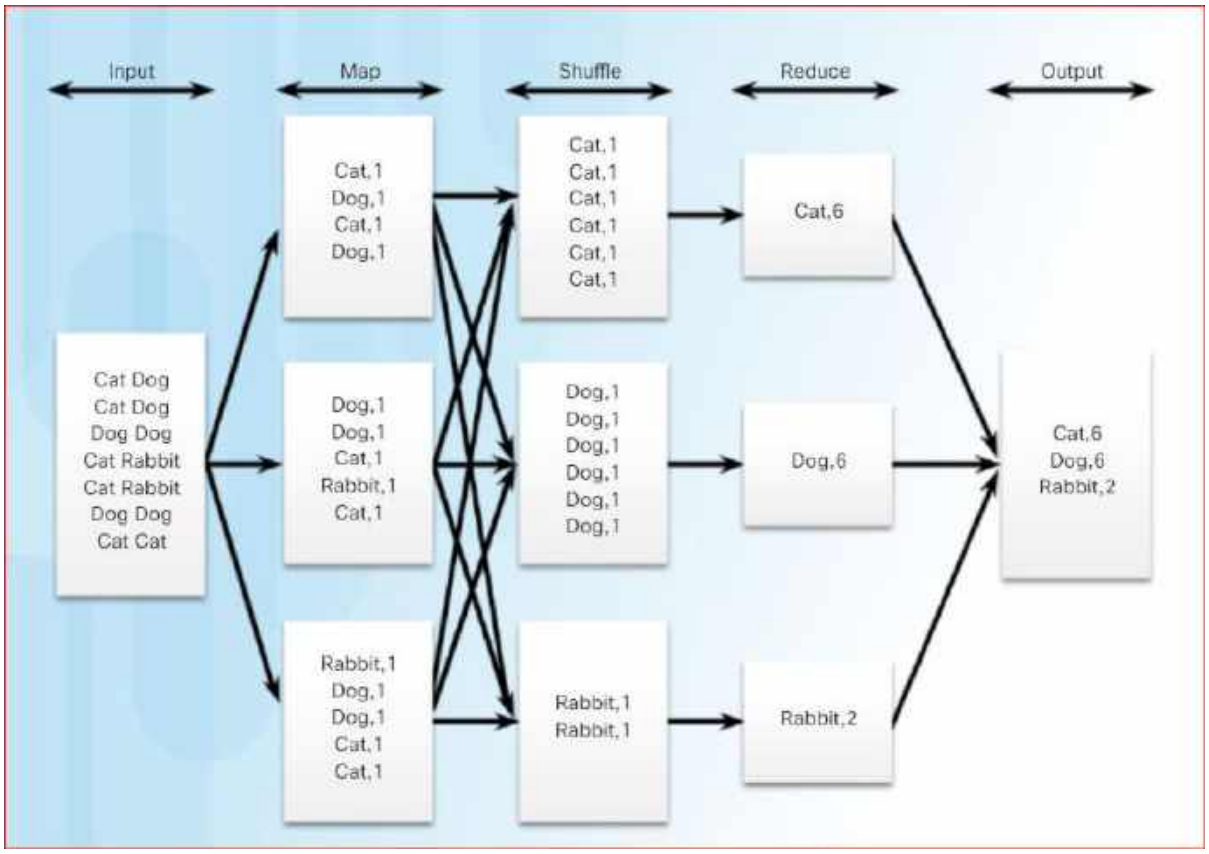


Рисунок 2.19 – Приклад використання MapReduce
для підрахунку кількості слів у потоці даних

2.2.6. Приклад отримання та оброблення Big Data

У цьому прикладі показано процес організації потоків зовнішніх даних і знайомство із HDFS.

Розглянемо сценарій: є інтернет-магазин, який дає змогу своїм клієнтам бронювати авіаквитки онлайн. З'ясуємо, як можна використовувати інструменти Big Data для аналізу даних, створених клієнтами, коли вони переглядають і купують товари на сайті магазину. Зокрема, які опції були доступні клієнтам, коли вони купували авіаквитки.

У прикладі використовуються такі інструменти:

– *Flume* – дає змогу брати великі дані та зберігати їх у HDFS-сховищі;

– *HDFS sink* – розподілена файлова система Hadoop дає потім змогу аналізувати отримані дані;

– зовнішня таблиця *Hive* – надає просту мову, схожу до SQL, для проведення аналізу.

Отже:

1) потрібно налаштувати *Flum*;

2) використати джерела даних: *Flume, HDFS sink, Hive external table*;

3) отримати інформацію, використовуючи *Hive*;

4) сформувати структуровану таблицю.

Налаштування Flum

Розглянемо конфігураційний файл *Flum.* Нижче дано пояснення рядків конфігураційного файлу *Flum*, який буде використовуватися в цьому прикладі.

Визначення джерела, поглинача (sink) та каналу:

Джерело збору даних Netcat

```
hdfs-agent.sources= netcat-collect
```

HDFS writers sink

```
hdfs-agent.sinks = hdfs-write
```

Канал пам’яті memoryChannel буде використовуватися для переміщення інформації між джерелом і каналом

```
hdfs-agent.channels= memoryChannel
```

Налаштування джерела

Встановлення типу колекції – netcat

```
hdfs-agent.sources.netcat-collect.type = netcat
```

Присвоєння netcat адреси localhost

```
hdfs-agent.sources.netcat-collect.bind = 127.0.0.1
```

Використання для netcat порту 11111

```
hdfs-agent.sources.netcat-collect.port = 11111
```

Визначення, що використовуються для Sink

Налаштування сховища на використання HDFS

```
hdfs-agent.sinks.hdfs-write.type = hdfs
```

Шлях, куди будете записувати дані у HDFS

```
hdfs-agent.sinks.hdfs-write.hdfs.path =
hdfs://localhost:8020/user/oracle/lesson1/flume_
example
```

Періодичність створення нових файлів (кожні 30 секунд)

```
hdfs-agent.sinks.hdfs-write.rollInterval = 30
```

Писати звичайним текстом замість файлу послідовності

```
hdfs-agent.sinks.hdfs-write.hdfs.writeFormat=Text
```

Призначення типу файлу як потоку даних

```
hdfs-agent.sinks.hdfs-write.hdfs.fileType=DataStream
```

Налаштування каналу

Вказуємо, що канал пам'яті матиме тип memory

```
hdfs-agent.channels.memoryChannel.type = memory
```

Призначаємо обсяг пам'яті в оперативній пам'яті

```
hdfs-agent.channels.memoryChannel.capacity=10000
```

Щоб звести все це воєдино, підключаємо канал до джерела

```
hdfs-agent.sources.netcat-
collect.channels=memoryChannel
```

Підключаємо канал до Sink

```
hdfs-agent.sinks.hdfs-write.channel=memoryChannel
```

Виконання пошуку та збору даних

Запускаємо й входимо до віртуальної машини Big DataLite 4.11.

Для виконання лабораторної роботи потрібно буде використання декількох термінальних вікон одночасно. Щоб додати новий термінал, натискаємо на *File*, відкрийте вкладку. Створюємо новий каталог у домашньому каталозі Oracle з іменем lesson1. Копіюємо вміст каталогу lesson_1 у новий каталог lesson1. Відкриваємо нове вікно термінала командного рядка, вводимо команду переходу до каталогу lesson1, який створили раніше: cd lesson1.

Ознайомлюємось із вмістом файлу *example.xml* з поточного каталогу.

Можемо отримати інформацію про клієнта в межах заданого виведення. Розглянемо дані та з'ясуємо, з яким змістом / типом інформації будемо мати справу (рис. 2.20).

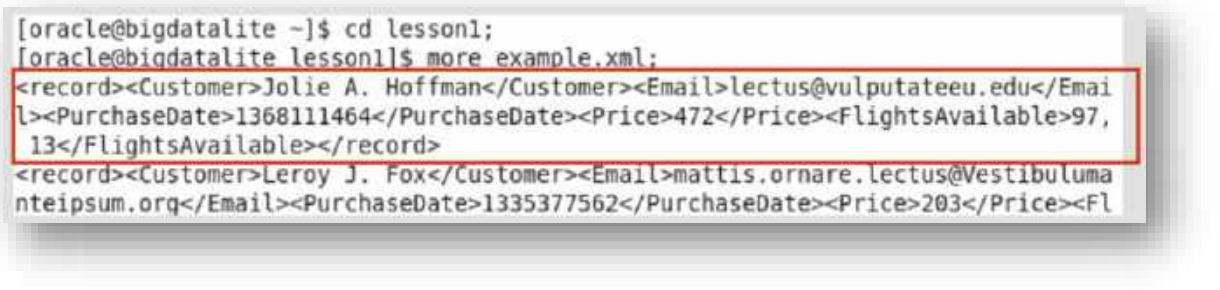


Рисунок 2.20 – Вміст файлу *example.xml*

Запускаємо агента потоку на основі файлу flume_example.conf з поточного каталогу: *flume-ng agent -n hdfs-agent -f ./flume_example.conf*.

Цей код запускає flume-агент. Ім'я (hdfs-agent) має відповідати імені, вказаному у файлі .conf, зазначеному в команді. Можна побачити ініціалізацію потоку, але останні кілька рядків показують налаштування джерела, приймача та каналу (рис. 2.21).

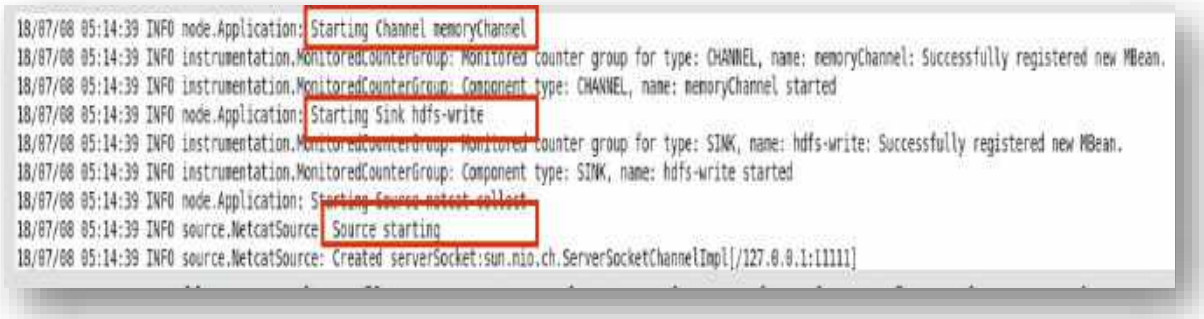


Рисунок 2.21 – Налаштування джерела, приймача та каналу

Відкриваємо нове вікно термінала Terminal 2. Відправляємо перші 10 рядків .xml-файлу через *netcat* до *Flum*:

```
head example.xml | nc localhost 11111
```

Список повідомлень «OK» (по одному для кожного рядка) – це підтвердження того, що *netcat* підключився й відправив дані (рис. 2.22).

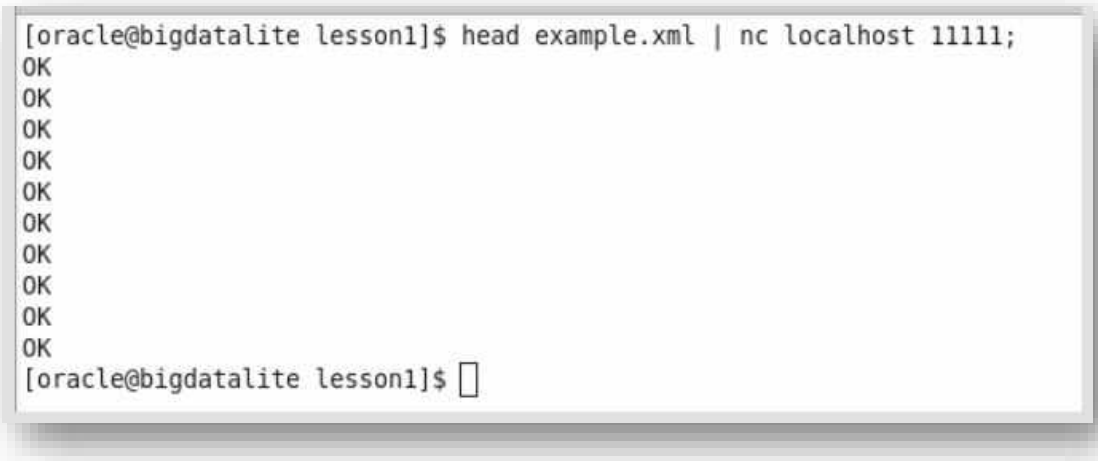


Рисунок 2.22 – Підтвердження підключення netcat і відправки даних

Натискаємо на Terminal 1, щоб подивитися на агента *Flum* (рис. 2.23).



Рисунок 2.23 – Агент Flum

Можна побачити, що створений файл закритий через 30 секунд і з присвоєним ім'ям, яке було зафіксовано.

У вікні Terminal 2 надсилаємо решту файлу через *netcat* до *Flume*:

```
cat example.xml | nc localhost 11111
```

Виконуємо ті самі перевірки, що й для перших 10 рядків, щоб показати, що решта файлу була оброблена через *Flume* (перевірте Terminal 1).

Використання Hive для отримання інформації

Відкриваємо нове вікно термінала Terminal 3. Перейдемо в Hive, щоб можна було працювати з даними більш структурованим способом:

```
hive
```

Створимо зовнішню таблицю:

```
CREATE EXTERNAL TABLE flume_example (record string)
ROW FORMAT DELIMITED
LINES TERMINATED BY '\n'
LOCATION '/user/oracle/lesson1/flume_example'
```

Тепер, коли в нас є таблиця, яка зберігає дані в одному стовпчику, можна продовжити й вибрати інформацію з неї:

```
SELECT record
FROM flume_example limit 10
```

Цей запит надасть перші 10 рядків таблиці відповідно до обмеження запиту (рис. 2.24). Наразі не досить важливо, що відображають рядки.



Рисунок 2.24 – Перші 10 рядків таблиці відповідно до обмеження запиту

Тепер використаємо *xpath* для отримання інформації окремо з одного стовпця з назвою *record*, який існує в таблиці:

```
SELECT xpath_int(record, "/record/PurchaseDate"),
xpath_string(record, "/record/FlightsAvailable")
FROM flume_example
```

Можна помітити, що фаза *Map Reduce*, яку проходить *Hive*, цього разу триває довше. Це тому, що *xpath* має бути запущено двічі для кожного рядка, який обробляється, щоб отримати дату придбання (*PurchaseDate)* та інформацію про доступні рейси *(FlightsAvailable).*

Це надає нам таку інформацію, яка показує мітку часу UNIX для дати й доступні рейси на цю дату (рис. 2.25).

Мітку часу UNIX можна замінити на фактичну датою та часом купівлі за допомогою функції FROM_UNIXTIME.

Для цього перезапускаємо відредагований запит:

```
SELECT FROM_UNIXTIME(xpath_int(record,
"/record/PurchaseDate")),
xpath_string(record, "/record/FlightsAvailable")
FROM flume_example
```

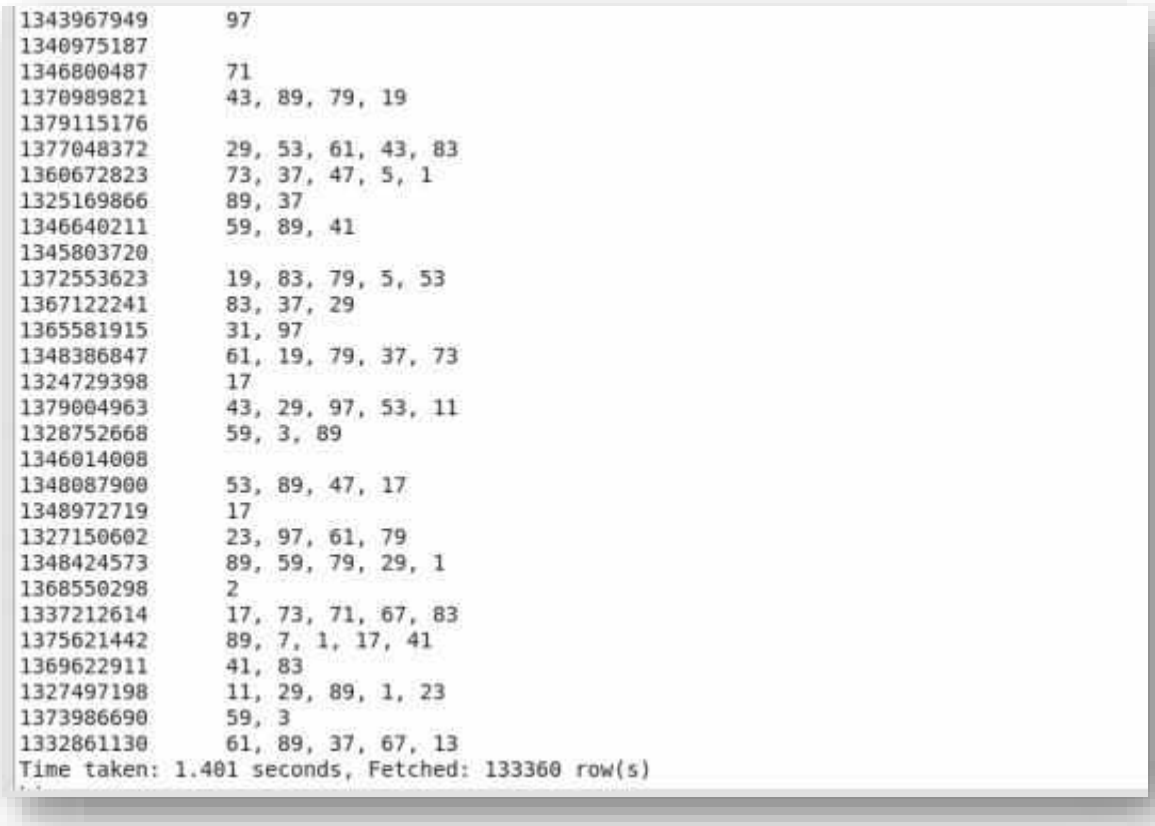


Рисунок 2.25 – Мітки часу UNIX для дати й доступних рейсів

Такий варіант дає нам набагато простіший зміст для розуміння результатів (рис. 2.26).

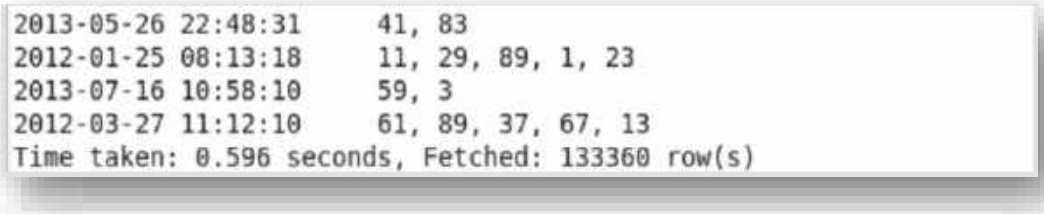


Рисунок 2.26 – Формування структурованої таблиці для аналізу

Тепер витягнемо інформацію, яка буде корисною для аналізу. Створимо таблицю, яка зберігатиме:

– ім’я клієнта;

– дату придбання авіаквитка;

– доступні рейси на той момент;

– дані про доступні рейси, які будуть розділені комою.

Для цього використовуємо команда *split*, яка створює масив у таблиці рейсів.

Hive може зберігати складні типи даних в одному стовпчику.

Вводимо в *Hive* такий скрипт:

```
CREATE TABLE flight_arrays AS
SELECT xpath_string(record, "/record/Customer") as
customer,
FROM_UNIXTIME( xpath_int(record,
"/record/PurchaseDate")) as purchase_date,
split(xpath_string(record,
"/record/FlightsAvailable"), ",") as
available_flights
FROM flume_example
```

Це займе хвилину або дві, оскільки є досить складним завданням. Тепер можна використати таблицю *flights_arrays* для отримання інформації, яку можна використовувати для аналізу.

Задача. Можемо написати запити, які повернуть, наприклад, ім'я клієнта та всі доступні рейси на момент купівлі; ім'я клієнта та дату купівля, впорядковані за датою купівлі.

Уся інформація про продажі з 2012 року впорядкована за датою (не обмежуйтесь однією датою, проте обмежуйте кількість даних (не більше 20 рядків).

Варіант 1. Вивести імена клієнтів і всіх доступних рейсів на момент купівлі:

```
SELECT customer, available_flights
FROM flight_arrays
LIMIT 20;
```

Якщо хочете бачити заголовок стовпця під час виконання запиту, виконайте оператор *set*, а потім повторно запустіть запит *Hive*:

```
set hive.cli.print.header=true;
```

Варіант 2. Вивести ім'я клієнта та дату купівлі, впорядковані за датою купівлі:

```
SELECT customer, purchase_date
FROM flight_arrays
ORDER BY purchase_date
LIMIT 20;
```

Варіант 3. Вивести всю інформацію про продажі з 2012 року в порядку дат (не обмежуйтесь однією датою):

```
SELECT customer, purchase_date, available_flights
FROM flight_arrays
WHERE purchase_date BETWEEN '2012-01-01' AND
'2012-12-31’
ORDER BY customer;
```

Можемо використати таблицю *flights_arrays* для отримання більш детальної інформації для аналізу. Використаємо функцію *count* для підрахунку кількості рейсів, зокрема *LATERAL VIEW* – це один з операторів *Hive* (складової Hadoop), який використовується для «розгортання» стовпців або елементів у структурах даних, які містяться в стовпці.

Варіант 4. У наступному прикладі *LATERAL VIEW* використовується з функцією *explode()*. Функція *explode()* здійснюється для «розгортання» масиву або структури даних у вихідні рядки. Тобто коли використовується *LATERAL VIEW explode(available_flights)*, кожен елемент масиву або структури в стовпці «available_flights» буде розгорнутий у окремий рядок, і цей рядок матиме новий стовпець з назвою «flight_no», який буде містити значення розгорнутого елемента.

Після цього використовується *GROUP BY* для групування даних за «purchase_date» і «flight_no», тобто рядки будуть групуватись за цими двома стовпцями. Отже, в результаті отримуємо груповані дані за датою покупки та номером рейсу. Тобто отримаємо невелику підтаблицю рейсів, яка буде прив’язана до клієнта і використовуватиметься для підрахунку:

```
SELECT purchase_date, flight_no, COUNT(flight_no)
FROM flight_arrays
LATERAL VIEW explode(available_flights)
flight_table as flight_no
GROUP BY purchase_date, flight_no;
```

У результаті у відповіді на запит отримаємо дату та кількість рейсів, які було запропоновано для кожного з них (рис. 2.27).

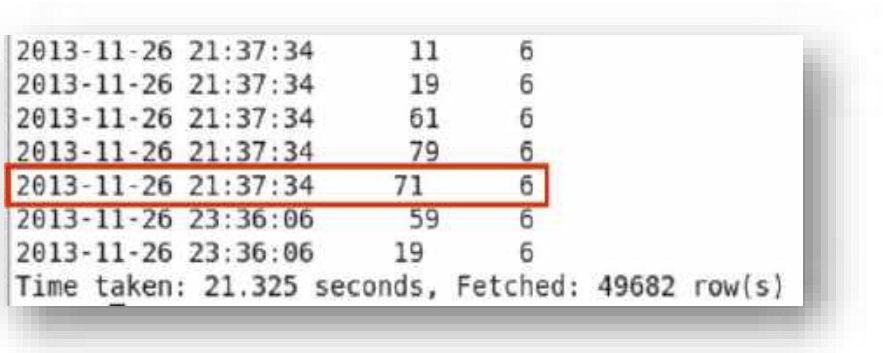


Рисунок 2.27 – Результат виконання
вибірки дат і кількості рейсів

Виділений рядок показує, що 26 листопада 2013 року рейс № 71 пропонувався 6 разів.

Приклад дав нам змогу відчути, як *Flume* може бути використаний у системах Big Data. *Flume* ідеально підходить для роботи з великими обсягами високошвидкісних даних. У поєднанні з *xpath* та зовнішньою таблицею *Hive* це досить корисно, оскільки можна аналізувати великі дані з погляду потреб бізнесу. Також є можливість вибрати, які елементи Big Data хочемо аналізувати, та працювати тільки з цією конкретною інформацією. Це дає змогу використовувати навички роботи з SQL у середовищі Big Data.

Питання для самоконтролю

1. Поясніть, у чому полягає ключова відмінність між традиційними системами обробки даних і підходом, що використовує *HDFS* та *MapReduce* для роботи з Big Data.

2. Опишіть основну роль *HDFS* та *MapReduce* в екосистемі Hadoop. Які завдання виконує кожна з цих технологій?

3. Назвіть і поясніть функції двох основних компонентів архітектури *HDFS: NameNode ma DataNode*.

4. Чому *HDFS* оптимізована для великих файлів і послідовного доступу до даних, а не для роботи зі значною кількістю маленьких файлів і довільного доступу?

5. Що таке «фактор реплікації» в HDFS і яке його основне призначення?

6. Дайте визначення поняттям «Producer», «Consumer», «Topic» та «Broker» у контексті *Apache Kafka.*

7. Поясніть, чим *Kafka* відрізняється від традиційних систем обміну повідомленнями (наприклад, *RabbitMQ).* Які переваги вона надає для потокової обробки даних?

8. Який тип бази даних являє Cassandra? Опишіть її основні архітектурні принципи, що забезпечують високу доступність і відмовостійкість.

9. Порівняйте *Cassandra* та *HDFS* (Hadoop). В яких сценаріях використання доцільніше обрати *Cassandra*, а в яких – *HDFS*?

10. Опишіть загальний принцип роботи MapReduce. Які два основні етапи обробки даних він містить?

11. Що відбувається на етапі *Shuffle and Sort* і чому він є критично важливим для коректної роботи моделі MapReduce?

12. Поясніть, як *MapReduce* забезпечує паралелізм та відмовостійкість під час виконання обчислень на кластері.

13. Які ключові пари (*key-value pairs*) є вхідними та вихідними для функції *Map*? Наведіть приклад.

14. Які ключові пари є вхідними та вихідними для функції *Reduce*? Наведіть приклад, який є логічним продовженням прикладу для функції *Map*.

15. Опишіть роль об’єкта *Context* (або аналогічного) у функціях *Map* та *Reduce.* Для чого він використовується?

16. Які функції виконує *Flume*?

17. У чому основні відмінності між реляційною БД і HDFS?

18. Які функції виконує *Hive*?

19. Як використовується *Hive* для отримання даних із *HDFS*?

20. За допомогою якої функції мітку часу UNIX можна замінити на фактичні дати?

21. Чуму для аналізу із неструктурованого масиву даних потрібно формувати структуровану таблицю?

22. Поясніть використання LATERAL VIEW explode (available_flights). Що означає «розгортання» масиву або структури даних?

23. Для чого у прикладі 2 використовується GROUP BY?

2.3. Spark: розподілена система обробки даних

2.3.1. Огляд компонентів Spark

Spark – це розподілена система обробки даних з відкритим вихідним кодом, яка використовується для роботи з *великими даними*. Вона була розроблена в Каліфорнійському університеті в Берклі 2009 року і стала доступною з відкритим вихідним кодом 2010 року. *Spark* використовує кешування в пам'яті для досягнення досить високої продуктивності за рахунок обмеження читання й запису на диск. Ця система може використовувати HDFS і досягати кращої продуктивності, ніж *MapReduce.* Вона також може використовувати інші типи кластеризації, як-от *Cassandra*. *Spark* підтримує багато різних мов програмування, зокрема Python, Scala, R та Java, структуровані бази даних на основі SQL. З усім цим розмаїттям підтримуваних мов і систем можна реалізувати багато різних рішень на *Spark*.

Spark містить багато бібліотек з коробки, які допомагають створювати значну кількість типів додатків, а саме:

Shark SQL – бібліотека SQL, яка досить ефективно підтримує складну аналітику, залишаючись водночас відмовостійкою. Підтримує запити даних або за допомогою SQL, або *Hive Query Language*. Бібліотека виникла як порт *Apache Hive* для роботи поверх *Spark* (замість *Map Reduce*), а нині вже інтегрована зі стеком *Spark*. Вона не тільки забезпечує підтримку різних джерел даних, а й дає змогу переплітати SQL-запити з трансформаціями коду; виходить надто потужний інструмент;

Spark Streaming – бібліотека обробки потоків, яка масштабується, підтримує високу пропускну здатність і відмовостійкість; обробку потокових даних у реальному часі; такими даними можуть бути файли логів робочого вебсервера (наприклад, *Apache Flume* і *HDFS/S3*), інформація із соціальних мереж, зокрема *Twitter*, а також різні черги повідомлень, наприклад *Kafka*. *Spark Streaming* отримує вхідні потоки даних і розбиває дані на пакети. Далі вони обробляються рушієм Spark, після чого генерується кінцевий потік даних (також у пакетній формі);

MLib – масштабована, високопродуктивна бібліотека машинного навчання, що надає різні алгоритми, розроблені для горизонтального масштабування на кластері з метою класифікації, регресії, кластеризації, спільної фільтрації тощо. Деякі з цих алгоритмів працюють і з потоковими даними, наприклад лінійна регресія з використанням звичайного методу найменших квадратів або кластеризація за методом k-середніх;

GraphX – швидка графічна бібліотека, яка містить багато алгоритмів побудови графіків. Бібліотека для маніпуляцій над графами й виконання з ними паралельних операцій. Бібліотека надає універсальний інструмент для ETL, дослідницького аналізу та ітераційних обчислень на основі графів. Крім вбудованих операцій для маніпуляцій над графами, тут також надається бібліотека звичайних алгоритмів для роботи з графами, наприклад *PageRank*.

Однією з причин швидкої роботи *Spark* є те, що він використовує стійкі розподілені набори даних (*Resilient Distributed Data sets, RDDs*) для зберігання вхідних, вихідних та проміжних даних у пам'яті, а не на диску. Це усуває витрати на введення / виведення. *RDD* є стійкими, тому що вони відстежують історію на випадок, якщо їм потрібно буде відновити себе після збою. Вони розподілені, оскільки розподілені на багатьох вузлах кластера. Це забезпечує надмірність, а також вищу ефективність завдяки паралельній обробці. Коли додаток робить запит, *Spark* створює та завантажує дані в *RDD*. Дані можуть надходити практично з будь-якого джерела, включно із *HDFS, CFS, AWS S3* або навіть базою даних *SQL*. Після створення *RDD* Spark може виконувати два різних типи операцій над *RDD*:

Transformations (перетворення) – це будь-яка маніпуляція з даними, охоплюючи відображення або фільтрацію. Будь-яке перетворення приведе до створення нового RDD. Оригінальний RDD не змінюється. Це дає змогу Spark виконувати контроль версій для RDD. Перетворення виконуються лише тоді, коли вони потрібні, наприклад для виконання дії.

Actions (дії) – взаємодіють з даними, але не створюють змін. Приклади охоплюють агрегування, підрахунок або пошук певного елемента в *RDD*. Після запиту на дію створюється новий *RDD*, на основі якого виконується дія (рис. 2.28).

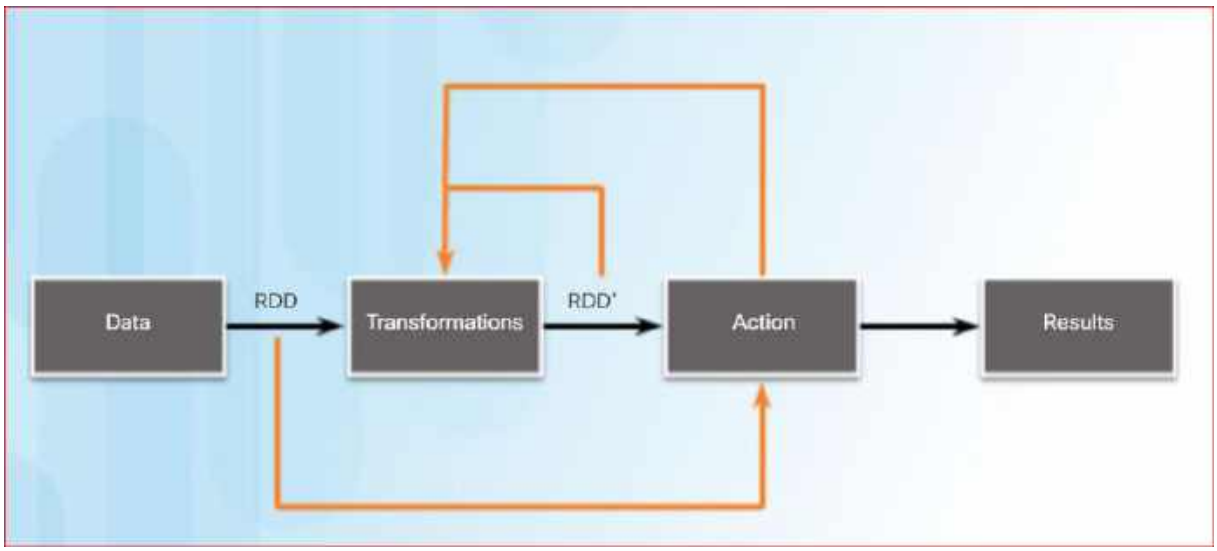


Рисунок 2.28 – Потік даних Spark

2.3.2. Переваги Spark над MapReduce

Spark може працювати прямо поверх екземпляру *Hadoop*, використовуючи HDFS для зберігання та YARN для управління кластером. *Spark* взагалі не потребує використання *Hadoop*. Можна використовувати інші рішення для зберігання даних, як-от *CFS* або *AWS S3*, а також інші кластерні менеджери, а саме *Mesos*. *Spark* є досить незалежною платформою, оскільки підтримує чимало різних технологій і мов програмування. Це корисно, оскільки існують будь-які потреби, коли йдеться про рішення для *великих даних* та аналіз у різних галузях.

Підхід до рішення для *великих даних* полягає у виборі правильних інструментів для роботи. Іноді поєднання *Spark* і *MapReduce* є найкращим рішенням для конкретного завдання. Використання *Hadoop* з *MapReduce* все ще є життєздатним під час виконання пакетної обробки, використання програми, яка застосовує виключно HDFS, коли персонал уже має досвід роботи з ними, або існуючі програми, які використовують цю технологію, вже запущені у виробництво.

Spark став досить популярним завдяки своїй продуктивності, легкості адміністрування, простоті й тому факту, що додатки можна створювати швидше під час його використання. Ось кілька причин використовувати *Spark* замість *MapReduce* в процесі створення рішення для роботи з великими даними:

Потокові дані. Spark досить добре справляється зі значними обсягами даних у реальному часі. Дані можуть надходити, наприклад, з мобільних пристроїв, соціальних мереж або датчиків інтернету речей.

Heterogeneous data (різнорідні дані) – багато рішень для роботи з великими даними отримують дані з різних джерел. Завдяки гнучкості Spark та його здатності виходити за межі цих різних сховищ даних, він є хорошим вибором для рішення.

Machine learning (машинне навчання) – завдяки вбудованій бібліотеці машинного навчання *Spark* може обслуговувати набагато більшу аудиторію, ніж попередні рішення.

Real-time applications (додатки в реальному часі) – обробка в пам'яті дає змогу *Spark* повертати результати обчислень набагато швидше, ніж *MapReduce*. Це важливо у всіх ситуаціях, але особливо, коли програма потребує результатів у реальному часі.

Less code (менше коду) – *Spark* підтримує багато різних мов, а це означає, що потрібно писати й підтримувати менше коду.

Developer experience (досвід розробника) – *Spark* набагато легше вивчити й він менше лякає, ніж *MapReduce.* Це дає змогу досить швидко створювати надійніший код для проєкту.

2.3.3. Основні терміни Spark

Job: Завдання: фрагмент коду, який зчитує деякі вхідні дані з HDFS або локальної пам'яті, виконує деякі обчислення над цими даними й записує деякі вихідні дані.

Stages: Етапи: завдання поділяються на етапи. Етапи класифікуються як карта або скорочені етапи (це легше зрозуміти, якщо ви працювали з Hadoop і хочете порівняти). Етапи поділяються на основі обчислювальних меж, усі обчислення (оператори) не можуть бути оновлені в одному Етапі. Це відбувається впродовж багатьох етапів.

Tasks: Завдання: кожна стадія має декілька завдань, по одному завданню на розділ. Одне завдання виконується на одному розділі даних на одній машині.

DAG. DAG розшифровується як *Directed Acyclic Graph*, у цьому контексті це DAG операторів.

Executor: Виконавець: процес, відповідальний за виконання завдання.

Master: Майстер: комп'ютер, на якому виконується програма-драйвер.

Slave: Підлеглий: комп'ютер, на якому виконується програма Executor: (Виконавець).

2.3.4. Компоненти Spark

1. *Spark Driver* окремий процес для виконання програм користувача створює *SparkContext* для планування виконання завдань і узгодження з менеджером кластера (рис. 2.29).

2. Executors: Виконавці:

– виконують завдання, заплановані Executors;

– зберігають результати обчислень у пам'яті, на диску;

– взаємодіють із системами зберігання.

3. Cluster Manager: кластер менеджерів:

– Mesos;

– YARN;

– Spark Standalone.

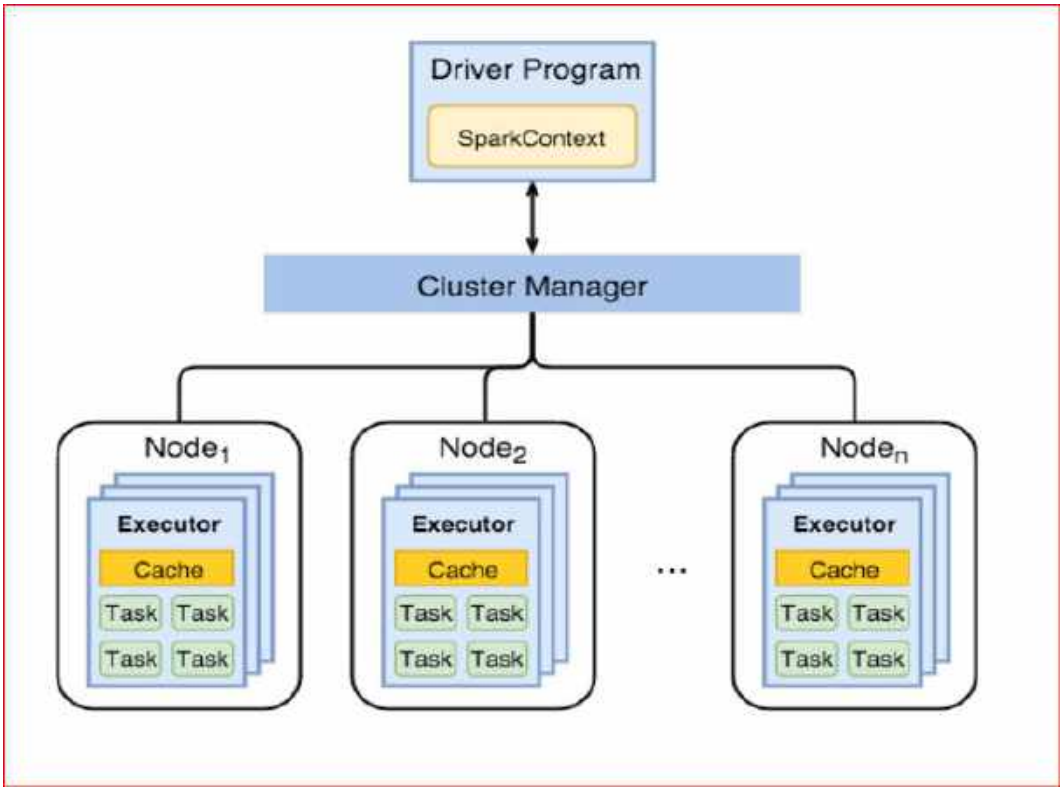


Рисунок 2.29 – Компоненти Spark

Драйвер *Spark* містить більше компонентів, відповідальних за трансляцію коду користувача в реальні завдання, що виконуються на кластері:

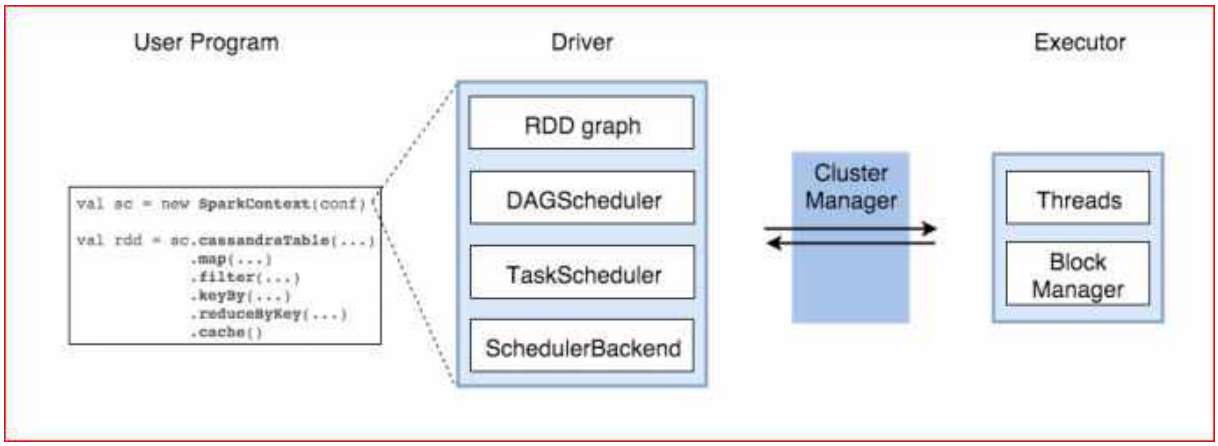


Рисунок 2.30 – Spark Driver

RDD, Resilient Distributed Dataset – представляє стійкий розподілений набір даних. *RDD у Spark* – це просто незмінна розподілена колекція наборів об'єктів. Кожен RDD розбивається на кілька секцій (схожий шаблон із меншими наборами), які можуть бути обчислені на різних вузлах кластера.

SparkContext – представляє підключення до кластера Spark і може використовуватися для створення RDD, накопичувачів і широкомовних змінних у цьому кластері.

DAGScheduler – обчислює DAG етапів для кожного завдання та надсилає їх у TaskScheduler, визначає бажані місця для завдань (на основі стану кешу або розташування файлів у випадковому порядку) і знаходить мінімальний графік для виконання завдань.

TaskScheduler, Планувальник завдань – відповідає за відправку завдань у кластер, їхнє виконання, повторні спроби, якщо є збої, і пом'якшення відсталих.

Scheduler Backend – внутрішній інтерфейс для систем планування, який дає змогу підключати різні реалізації (Mesos, YARN, Standalone, local).

BlockManager – надає інтерфейси для розміщення та отримання блоків як локально, так і віддалено в будь-які сховища (пам'ять, диск тощо).

Spark має невелику кодову базу, система розділена на різні рівні. Кожен шар має певні обов'язки. Шари незалежні один від одного (рис. 2.31).

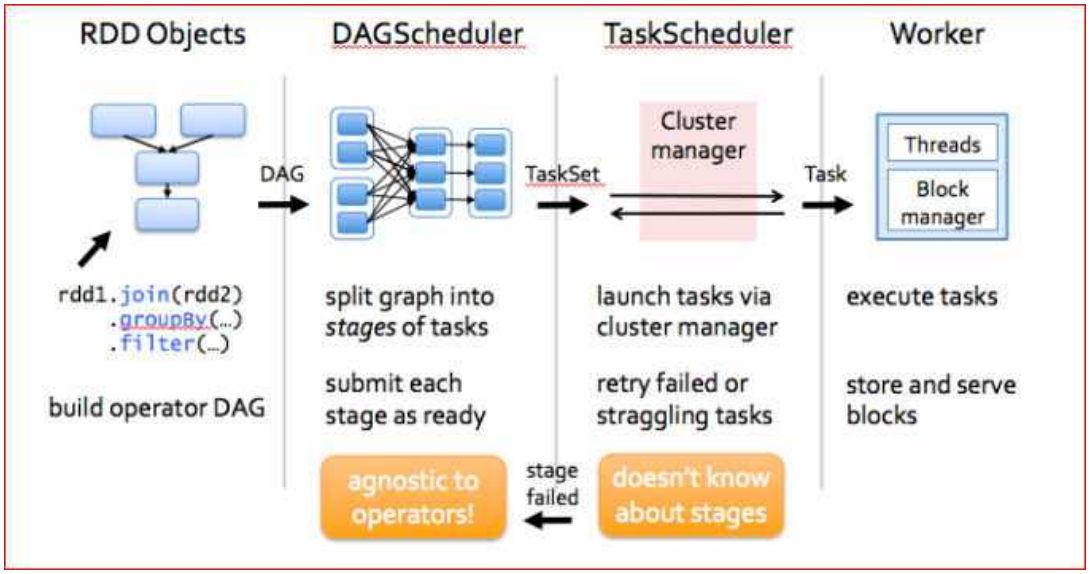


Рис. 2.31 – Шари Spark

Перший рівень – це інтерпретатор, Spark використовує інтерпретатор Scala з деякими модифікаціями. Коли вводимо код у консоль spark (створюючи RDD і застосовуючи оператори), Spark створює графік операторів. Коли користувач виконує дію (наприклад, збір), графік надсилається до планувальника DAG. Планувальник DAG розділяє граф оператора на етапи (відображення та зменшення). Етап складається із завдань на основі розділів вхідних даних. Планувальник DAG об'єднує оператори разом для оптимізації графіка. Багато операторів карт можна запланувати в один етап. Ця оптимізація є ключем до продуктивності Sparks. Останнім результатом планувальника DAG є набір етапів. Етапи передаються до *Планувальника завдань (Resilient Distributed Dataset)*. Планувальник завдань запускає завдання через менеджер кластерів (*Spark Standalone/Yarn/Mesos*). *Планувальник завдань* не знає про залежності між етапами (рис. 2.31).

2.3.5. Лямбда-архітектура обробки даних

Щоб зблизити аналіз історичних даних і даних у реальному часі, була створена *архітектура Lambda (Лямбда)*. *Лямбда* – це архітектура обробки даних, яка використовує як потокову, так і пакетну обробку для отримання точних уявлень як про «живі» дані, так і про пакетні дані. Лямбда-архітектура має чотири шари, як показано на рис. 2.32:

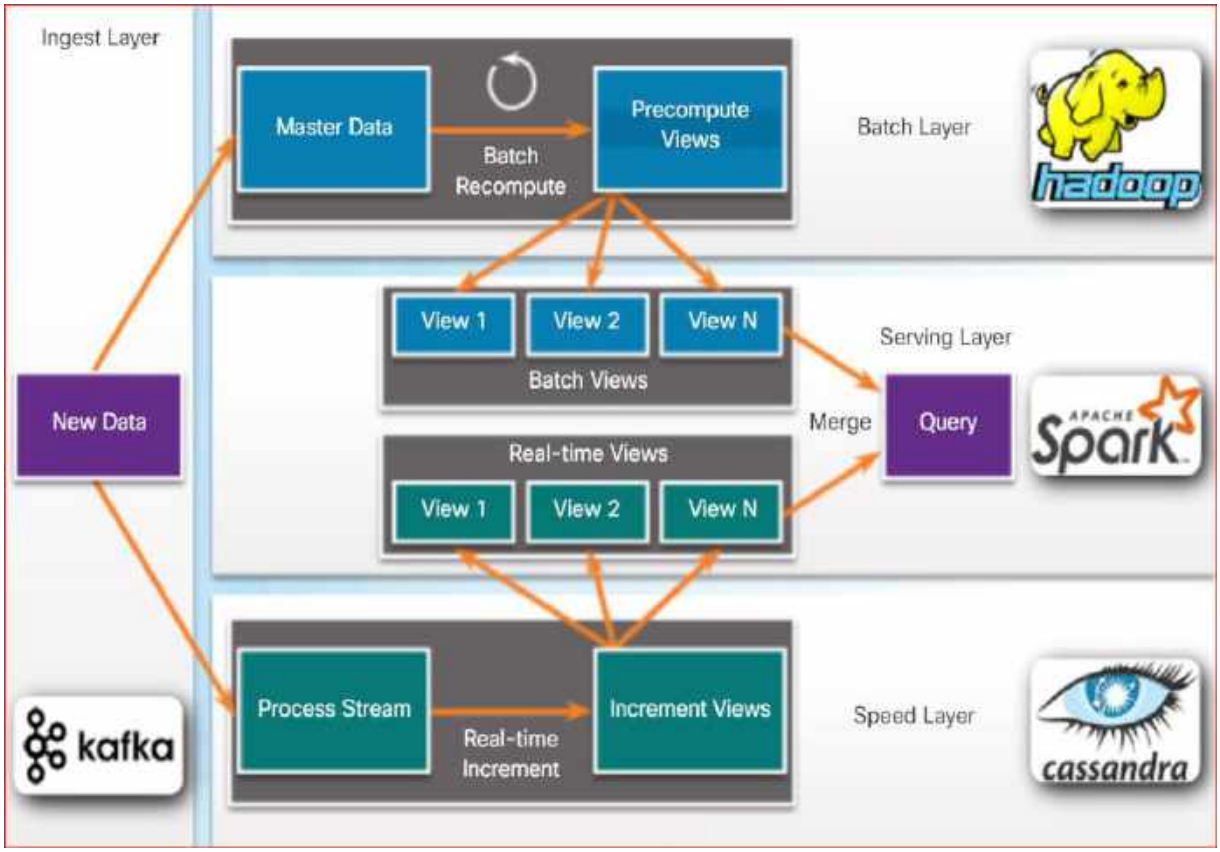


Рис. 2.32 – Лямбда-архітектура обробки даних

Ingestion (поглинання) – на цьому рівні відбувається імпорт даних. Це можуть бути дані з різних джерел, включно із потоками даних.

Batch (пакетний) – це рівень даних у стані спокою. Дані тут часто будуються за розкладом і охоплюють імпорт даних з рівня потоку. Точність тут важливіша за швидкість.

Stream (потік) – це складний рівень, що підтримує інкрементне оновлення. Низька затримка тут є пріоритетом над точністю. Ці дані часто відстають від генерації даних лише на кілька секунд.

Presentation (представлення) – цей рівень керує роботою. Він приймає всі запити й вирішує, чи використовувати швидкісний або пакетний рівень. Найчастіше перевага надається пакетному рівню, оскільки дані є чистішими. Якщо запитуються молоді дані, то використовується потоковий рівень.

Усі дані, які надходять в архітектуру, надсилаються одночасно на пакетний і швидкісний рівні, як показано на рис. 2.32. Пакетний рівень (*batch layer)* управляє основним набором даних і попередньо обчислює пакетні подання, які постійно обчислюються. Рівень, який обслуговує *(serving layer)*, індексує пакетні подання, щоб їх можна було запитувати. Швидкісний рівень (*speed layer)* має справу лише з останніми даними, що компенсує затримку оновлень на рівні, який обслуговує дані. Два різних результати запиту можуть бути об'єднані, щоб сформувати нове представлення даних.

Інтернет речей (*IoT*) є ідеальною сферою для впровадження *лямбда-архітектури*. Дані створюються з надзвичайно високою швидкістю, набори даних можуть бути неймовірно великими, а запити можуть стосуватися як даних у стані спокою, так і даних у русі. *IoT* також виграє від комбінованого уявлення, яке може забезпечити *лямбда-архітектура*.

На рис. 2.32 під назвою «Лямбда-архітектура» показано, як нові дані обробляються за допомогою чотирьох рівнів лямбда-архітектури: рівня прийому, пакетного рівня, рівня обслуговування та швидкісного рівня.

Приклад використання лямбда-архітектури для отримання даних з автобусних маршрутів.

Це реальний приклад того, як лямбда-архітектура може бути використана, наприклад, для візуалізації 171 автобусного маршруту в Лос-Анджелесі, штат Каліфорнія. Структура рішення містить SACK (*Spark, Akka, Cassandra і Kafka*). Це дає змогу аналізувати дані майже в режимі реального часу. Одним із інструментів, що використовується тут, який не розглядався, є *Akka*. *Akka* – безкоштовний інструментарій з відкритим вихідним кодом для створення розподілених і стійких додатків, керованих повідомленнями. У цій структурі *Akka* використовується для отримання

метаданих інформації про маршрут і зберігання їх у *Cassandra* кожні 30 секунд.

У цьому прикладі *Akka* використовується для запиту даних через безкоштовний REST API. Потім дані зберігаються в *Cassandra*. Наступні запити надсилаються до *Kafka*. *Spark* використовується для зчитування інформації про транспортний засіб із *Kafka.* Коли всі ці дані доступні, інший API використовується для візуалізації положення автобусів за допомогою *OpenStreetMap*. Дивовижним у цьому процесі є те, що поточні позиції автобусів транслюються прямо з *Kafka* на карту за допомогою *вебсокетного зв'язку*.

Ця конкретна платформа не обмежується тільки даними IoT. Вона підходить для рішень, де є багато вхідних даних. Вона досить корисна, коли є кілька паралельних вхідних потоків даних. Це набагато краще рішення, ніж традиційна СУБД, яка не зможе надавати дані в режимі, близькому до реального часу.

Питання для самоконтролю

1. Чому *Spark* називають системою для *розподілених* обчислень? Поясніть, як він виконує завдання на кластері з багатьох вузлів.
2. Що таке обчислення *в пам'яті* (in-memory computing) і як ця концепція дає змогу *Spark* значно прискорювати обробку даних?
3. Назвіть і коротко опишіть призначення чотирьох основних компонентів екосистеми *Spark* (Spark Core, Spark SQL, Spark Streaming, MLlib).
4. Яка роль *Cluster Manager* (наприклад, YARN, Mesos або Standalone) в архітектурі *Spark*-додатка?
5. Назвіть щонайменше три ключові переваги *Spark* порівняно з класичним Hadoop MapReduce.
6. Поясніть, як використання RDD (Resilient Distributed Datasets) та DAG (Directed Acyclic Graph) дає змогу *Spark* бути ефективнішим за MapReduce, який постійно звертається до диска (HDFS).

7. Дайте визначення термінам: RDD, Transformation (трансформація) та Action (дія). У чому полягає їхня принципова відмінність?

8. Що таке *ліниві обчислення* (lazy evaluation) у *Spark* і як вони пов’язані з трансформаціями та діями?

9. Поясніть різницю між Driver Program, Executor та Worker Node у контексті виконання *Spark*-завдання.

10. Опишіть основну ідею лямбда-архітектури. Які три ключові шари вона охоплює і за що відповідає кожен з них?

11. Яку проблему вирішує лямбда-архітектура, поєднуючи пакетну (batch) та потокову (real-time) обробку даних?

12. Які компоненти екосистеми *Spark* можуть бути використані для реалізації *Batch Layer* та *Speed Layer* в рамках лямбда-архітектури?

2.4. Платформи та інструменти Big Data

2.4.1. Oracle Big Data SQL

Oracle BigData SQL підтримує запити до нереляційних даних, що зберігаються в кількох джерелах великих даних, включно із *Apache Hive, HDFS, Object Stores, Oracle NoSQL Database, Apache Kafka, Apache HBase* та іншими базами даних NoSQL.

Уніфіковані запити для розподілених даних підтримуються *Oracle Big Data SQL* (рис. 2.33) [9]. Отже, користувачі можуть легко переглядати й аналізувати дані з різних сховищ даних, ніби всі вони зберігаються в базі даних Oracle.

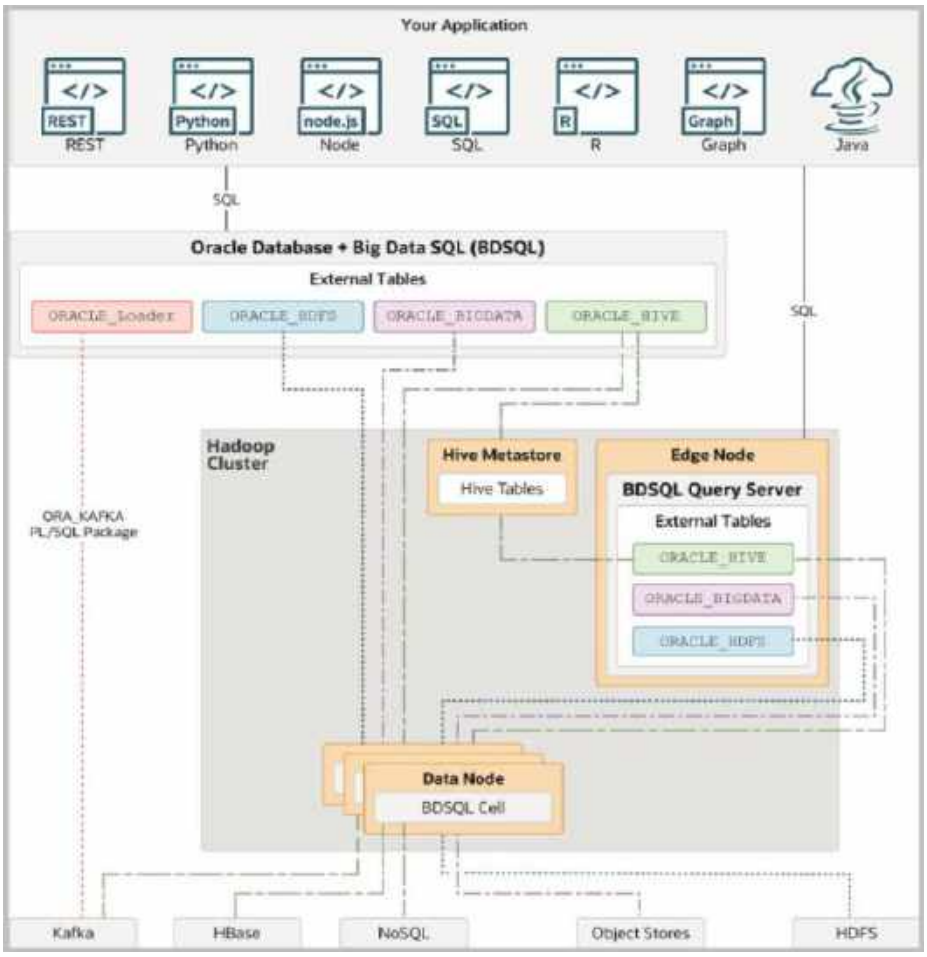


Рисунок 2.33 – Архітектура Oracle Big Data SQL

Oracle Big Data SQL дає змогу виконувати досить складні `SELECT` оператори SQL щодо даних в екосистемі *Hadoop* вручну або через існуючі програми. Наприклад, якщо ви є користувачем

Oracle Advanced Analytics, O*racle Big Data SQL* дає змогу розширити ваші моделі аналізу даних Oracle Database на великі дані в *Hadoop*.

На рис. 2.33 показана архітектурна схема, яка ілюструє потік даних запитів SQL, включно із компонентами, які беруть участь у запитах, наприклад драйвери, зовнішні таблиці та кластери й вузли Hadoop. На наведеній вище діаграмі показано потік даних для таких запитів:

– запит Kafka за допомогою Oracle SQL Access to Kafka (OSaK) – Oracle SQL Access to Kafka дає змогу Oracle SQL отримувати доступ до тем Kafka без використання Hive або Hadoop;

– пакет ORA_KAFKA PL / SQL містить функції та процедури, які дають змогу це зробити. Перш ніж надсилати запит Kafka, зареєструйте кластер Kafka в схемі вашої бази даних за допомогою `ORA_KAFKA.REGISTER_CLUSTER`, а потім створіть представлення за допомогою `ORA_KAFKA.CREATE_VIEWS` процедури.

Ця процедура створює зовнішні таблиці під обкладинками, вам не потрібно явно створювати зовнішні таблиці (Oracle SQL Access to Kafka) [13];

– запит Kafka за допомогою обробника сховища Hive – `ORACLE_HIVE` драйвер доступу може отримати доступ до тем Kafka за допомогою обробника сховища Hive, який є частиною інсталяції Oracle Big Data SQL. Спочатку створюєте зовнішню таблицю Hive, яка має доступ до тем Kafka, а потім над нею – таблицю Oracle Big Data SQL (розділ «Використання обробника сховищ Oracle Hive для Kafka для створення зовнішньої таблиці Hive для Kafka») [13; 14];

– запит Apache HBase – можете використовувати `ORACLE_HIVE` драйвер доступу для доступу до даних, що зберігаються в Apache HBase. Спочатку створюєте зовнішню таблицю Hive, яка отримує доступ до таблиці HBase, а потім над нею створюєте таблицю Oracle Big Data SQL («Створення зовнішньої таблиці Oracle для Apache HBase») [13; 14];

– Query NoSQL – `ORACLE_HIVE` драйвер доступу використовується для доступу до даних, що зберігаються в Oracle NoSQL Database. Спочатку створюєте зовнішню таблицю Hive, яка отримує

доступ до KVStore, а потім над нею – зовнішню таблицю в Oracle Big Data SQL («Створення зовнішньої таблиці для бази даних Oracle NoSQL»);

– сховища об'єктів запиту – `ORACLE_BIGDATA` драйвер доступу дає змогу створювати зовнішню таблицю над даними, що зберігаються в сховищах об'єктів. Oracle Big Data SQL підтримує доступ до сховищ об'єктів, як-от Oracle Object Store, Amazon S3 і Azure Blob Storage («Створення зовнішньої таблиці Oracle для доступу до сховища об'єктів»);

– запит HDFS – `ORACLE_HDFS` драйвер доступу дає змогу отримати доступ до багатьох типів даних, які зберігаються в HDFS, але які не мають метаданих Hive. Можете визначити формат запису текстових даних або вказати SerDe для певного формату даних («Створення зовнішньої таблиці Oracle для файлів HDFS») [13; 14].

2.4.2. Amazon Web Services

Amazon Web Services (AWS) надає широкий спектр сервісів для обробки та аналізу великих обсягів даних (Big Data). Ось деякі з основних можливостей AWS для Big Data [15]:

1. *Amazon S3 (Simple Storage Service)* – безпечне та масштабоване сховище даних, яке дає змогу зберігати будь-які обсяги даних, охоплюючи великі обсяги структурованих і неструктурованих даних.

2. *Amazon EMR (Elastic MapReduce)* – керована служба Hadoop, яка дає змогу легко обробляти й аналізувати великі обсяги даних за допомогою фреймворків, як-от Apache Hadoop, Apache Spark, Apache HBase, Apache Flink тощо.

3. *Amazon Redshift* – повністю керована хмарна data warehouse служба, яка дає змогу аналізувати великі обсяги даних і виконувати складні SQL-запити за допомогою значного масиву інструментів аналітики та бізнес-інтелекту.

4. *Amazon Athena* – інтерактивний сервіс для аналізу даних у S3 за допомогою стандартного SQL. Немає потреби вести складну інфраструктуру або завантажувати дані в спеціалізовані бази даних.

5. *Amazon Kinesis* – платформа для збору, обробки й аналізу стрімів даних у реальному часі. Вона дає змогу легко збирати та аналізувати дані зі значної кількості джерел, як-от вебсервери, девайси інтернету речей (IoT), системи моніторингу тощо.

6. *Amazon Quicksight* – сервіс для візуалізації даних, який дає змогу легко створювати інтерактивні та відчутні графіки й дашборди на основі даних, які розміщені в Amazon S3, Redshift, RDS та інших джерелах.

7. *AWS Glue* – повністю керована служба, яка дає змогу автоматизувати процеси підготовки та перетворення даних для аналізу. Вона використовує відомості про структуру й метадані даних для автоматичного створення ETL (Extract, Transform, Load) процесів.

Це лише кілька з можливостей, які надає AWS для роботи з великими обсягами даних. AWS також надає інші сервіси, які можуть бути використані для різноманітних завдань Big Data, як-от машинне навчання, обробка потокових даних, аналітика та інше.

2.4.3. Microsoft Azure

Microsoft надає широкий набір інструментів і сервісів для роботи з великими обсягами даних (Big Data). Ось деякі з основних можливостей Microsoft для Big Data [16].

1. *Microsoft Azure Data Lake Storage* (ADLS) – безпечне та масштабоване сховище даних, призначене спеціально для аналізу великих обсягів даних. ADLS дає змогу зберігати структуровані й неструктуровані дані та працювати з ними в реальному часі.

2. *Azure HDInsight* – керована служба Hadoop у хмарі, яка дає змогу обробляти й аналізувати великі обсяги даних за допомогою різних фреймворків, як-от Apache Hadoop, Apache Spark, Apache Hive, Apache HBase та інші.

3. *Azure Databricks* – манагована платформа для обробки й аналізу даних, розроблена спільно з Databricks, яка підтримує обробку потокових даних, аналітику машинного навчання та інші аналітичні роботи.

4. *Azure Synapse Analytics* – аналітична платформа, яка об'єднує в собі обробку даних (SQL Data Warehouse) та аналітичні засоби (Big Data Analytics) у єдиному сервісі. Вона дає змогу виконувати аналізи в реальному часі, інтегрувати дані з різних джерел і використовувати інструменти машинного навчання.

5. *Azure Stream Analytics* – служба для аналізу потокових даних у реальному часі. Дає змогу аналізувати та витягати цінні дані з потокових джерел, як-от девайси IoT, вебсервери, соціальні медіа тощо.

6. *Azure Machine Learning* – платформа для розробки, навчання та впровадження моделей машинного навчання в хмарі. Вона надає інструменти для створення, тренування й оцінювання моделей, а також для автоматизації процесів машинного навчання.

7. *Azure Data Factory* – служба для оркестрації та автоматизації потоків даних між різними джерелами даних і призначеннями. Дає змогу легко створювати, розгортати та управляти комплексними потоками даних у хмарному середовищі.

Це лише кілька з можливостей, які надає Microsoft Azure для роботи з великими обсягами даних. Microsoft також надає інші сервіси та інструменти для аналізу, обробки й візуалізації даних, які можуть бути використані для різних сценаріїв Big Data.

2.4.4. Google Cloud Platform

Google Cloud Platform (GCP) надає розширений набір інструментів і сервісів для роботи з великими обсягами даних (Big Data). Ось деякі з основних можливостей Google для Big Data [17].

1. *Google BigQuery* – повністю керована хмарна база даних для аналітики великих обсягів даних. BigQuery підходить для виконання аналітичних запитів через складання SQL-запитів до великих обсягів даних, збережених в обліковому записі Google Cloud.

2. *Google Cloud Dataflow* – послуга потокової обробки даних, яка дає змогу створювати, моніторити та запускати потокові програми для обробки великих обсягів даних у реальному часі.

3. *Google Cloud Dataproc* – керована служба Hadoop та Spark, яка дає змогу швидко створювати кластери обробки даних у хмарі для аналізу великих обсягів даних за допомогою Apache Hadoop, Apache Spark, Apache Pig та інших інструментів.

4. *Google Cloud Pub / Sub* – послуга потокового обміну повідомленнями, яка дає змогу надсилати й отримувати дані в реальному часі із масштабованими, надійними потоками.

5. *Google Cloud Storage* – безпечне та масштабоване сховище об'єктів, яке дає змогу зберігати та використовувати будь-які обсяги даних в обліковому записі Google Cloud.

6. *Google Cloud Bigtable* – повністю керована служба NoSQL бази даних, яка підтримує швидку обробку великих обсягів даних з низькою затримкою.

7. *Google Data Studio* – безкоштовний інструмент візуалізації даних, який дає змогу створювати красиві та інтерактивні звіти й дашборди на основі даних з Google Cloud та інших джерел.

Вказані лише деякі з можливостей, які надає Google Cloud Platform для роботи з великими обсягами даних. Google також надає інші сервіси та інструменти для аналізу, обробки й візуалізації даних, які можуть бути використані для різних сценаріїв Big Data.

Питання для самоконтролю

1. Поясніть сутність і особливості технології великих даних Hadoop.

2. Опишіть особливості Hadoop Distributed File System – розподіленої масштабованої файлової системи.

3. Опишіть принципи роботи та сутність MapReduce – моделі розподілених обчислювань у комп'ютерних кластерах.

4. Опишіть своїми словами особливість сховища даних, реалізованого для Hadoop на Hive.

5. Опишіть особливості Pig – скриптової мови програмування.

6. Опишіть особливості Sqoop – інструменту на базі командного рядка.

7. Опишіть особливості ZooKeeper – координатора для різних екземплярів Hadoop.

8. Опишіть особливості використання Hadoop.

9. Опишіть особливості Pig – високорівневої мови опису потоків даних.

10. Опишіть особливості Hive: SQL – подібна інфраструктура для організації сховищ даних.

11. Опишіть особливості розподіленої СУБД зі зберіганням по стовпцях HBase.

12. Опишіть особливості розподіленої системи обробки даних Spark.

ТЕМА 3
МЕТОДИ ТА ІНСТРУМЕНТАРІЙ МАШИННОГО НАВЧАННЯ

3.1. Data Science: наука про дані і машинне навчання

3.1.1. Штучний інтелект та інтелектуальний аналіз

У сучасному світі в усіх галузях потрібно вміти правильно й ефективно працювати з інформацією: збирати її, аналізувати, робити прогнози та використовувати для обґрунтованого прийняття рішень. Сукупність підходів, методів і технологій, призначених для виконання цих завдань, відома як «Наука про дані» (*Data Science, DS*). У різній літературі зміст складових та інструментарію DS подається по-різному. У цьому посібнику приймаються визначення термінів DS, розміщені в [18] (рис. 3.1).

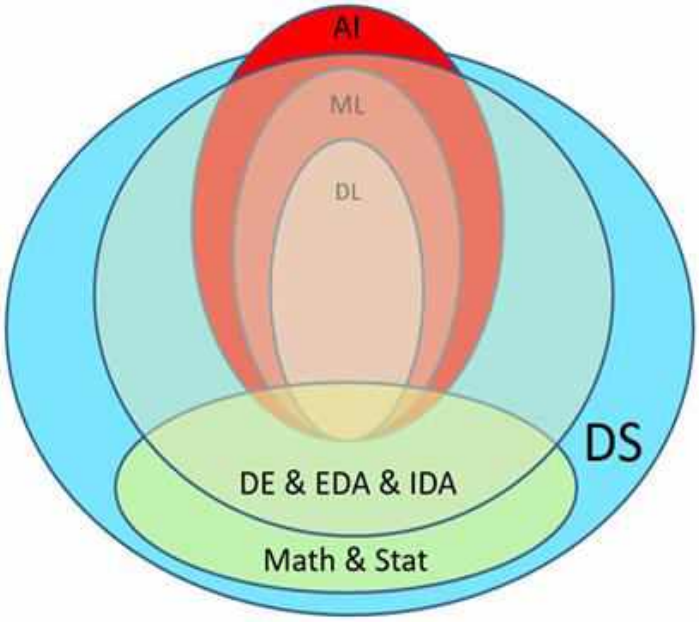


Рисунок 3.1 – Основні поняття Data Science та їх поєднання

Основні складові DS [17].

Інженерія даних (*Data Engineering, DE*) – методи й інструменти збирання, зберігання, попереднього аналізу й підготовки даних для подальшого використання в машинному навчанні.

Машинне навчання (*Machine Learning, ML*) – набір алгоритмів і технологій, які навчають комп'ютерні моделі на прикладах і застосовують їх для рішення прикладних завдань. На сьогодні прийнято вважати, що ML застосовується для навчання штучних нейронних мереж і класичних аналітичних методів класифікації та прогнозування. Якщо *DE* – це насамперед розробка, збір і керування датасетами, то ML фокусується на навчанні моделей на таких підготовлених даних. Проте в оголошеннях про вакансії дата-інженерів чи ML-розробників часто плутають обсяг обов'язків, додаючи до DE завдання ML, і навпаки.

Штучний інтелект (*Artificial Intelligence, AI*) – широка галузь, яка використовує прийоми машинного навчання, моделюючи такі аспекти людського пізнання, як навчання, логіка, розпізнавання образів і прийняття рішень саме на основі штучних нейронних мереж, нечіткої логіки, генетичних алгоритмів та іншого інструментарію, що імітує роботу людського пізнання.

Сфера застосування ML ширша: крім навчання моделей штучного інтелекту, він охоплює всі інші алгоритми автоматичного навчання моделей (класифікація, регресія, прогнозування тощо), тому ML є більш загальною категорією, ніж AI. Водночас AI також вивчає будову людського мозку та психічні процеси, що виходить за межі суто ML-методів.

Інтелектуальний аналіз даних (*Intelligent Data Analysis, IDA*) – підхід, що поєднує різноманітні методи й технології для глибокого аналізу та інтерпретації складних даних. IDA зазвичай полягає в застосуванні готових навчених інтелектуальних моделей і технологій для вирішення прикладних завдань – від програмування до творчих проєктів. Зокрема, на сьогодні є у доступі значна кількість великих мовних моделей і сервісів, як-от ChatGPT від OpenAI та ін. Часто ML / AI розглядають як етап підготовки та проєктування систем IDA, а інколи IDA вважають завершальною стадією побудови ML / AI-систем.

Окремо варто виділити **інтелектуальні моделі** – це аналітичні моделі, які здатні вчитися на прикладах і узагальнювати знання для роботи з новими даними та сценаріями. Вони відрізняються від простих систем розпізнавання тим, що можуть не

лише зіставляти вхідні дані з базою, а й прогнозувати їх зміну, наприклад передбачати, як зміниться обличчя людини за десять років.

У межах AI прийнято розрізняти:

вузький (слабкий) штучний інтелект (narrow / weak AI) – рішення конкретних завдань (розпізнавання мови, фото, машинний переклад, гра в шахи та ін.) без загальних здібностей до мислення;

загальний (сильний) штучний інтелект (general / strong AI) – теоретично здатний до універсального мислення, навчання й вирішення проблем на рівні людини; наразі прогнозується його з'явлення через 5–6 років.

3.1.2. Життєвий цикл машинного навчання й Data Science

У цьому посібнику пропонується таке подання ієрархії процесів DS:

1. *Data Engineering (DE):* збір, попередня обробка та організація даних (включно із *Big Data)* у зручні для аналізу датасети. DE можна розглядати як складову *Data Science* або як окрему дисципліну.

2. *Exploratory Data Analysis (EDA):* прикладний аналіз даних із використанням методів математики та статистики, а також швидке прототипування моделей (зокрема, нейромереж). Цей аналіз може передбачати пробну побудову моделей для аналізу певних закономірностей щодо ознак. Причому такими моделями можуть бути нейронні мережі глибокого навчання (*Deep Learning, DL*).

3. *Machine Learning (ML) / Deep Learning (DL) / Artificial Intelligence (AI):* за результатами EDA проводиться машинне навчання (ML) моделей і моделей глибокого навчання (DL) та штучного інтелекту (AI).

4. *Intelligent Data Analysis (IDA):* застосування навчених моделей для розв'язання реальних аналітичних завдань із різними типами даних (числа, текст, зображення, відео, аудіо).

Щодо взаємодії DS / ML / DL / AI, то в літературі є різні підходи. У цьому посібнику подається схема, наведена в [18], яка відповідає, на нашу думку, реальному стану речей.

У поєднанні *комп'ютерних наук* (*Computer Sciences*) зі знаннями та даними предметної сфери (*Domain & Data*), а також із бізнес-знаннями й відповідними даними про показники ефективності, вимоги користувачів, стратегії та обмеження (*Business Knowledge & Data*) формується процес *розробки програмного забезпечення* (*Software Development*).

Коли до цього додаються методи математики (зокрема, статистики) та кібернетики (теорії керування й оптимізації), отримуємо класичні підходи до математичного моделювання, прогнозування й оптимізації систем.

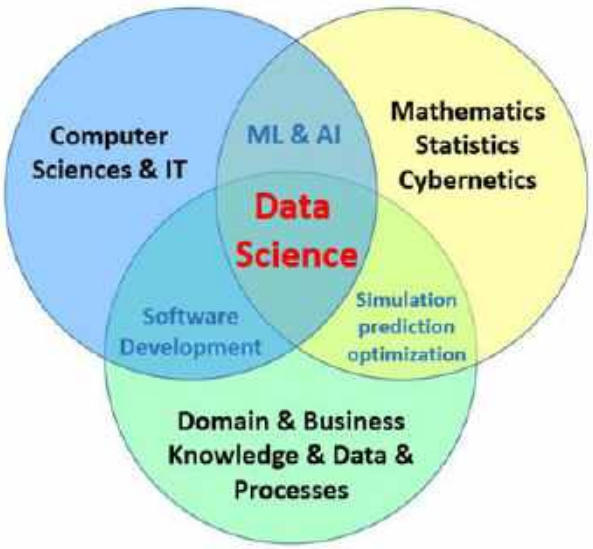


Рисунок 3.2 – Основні поняття та їхнє поєднання у сфері Data Science

А об'єднання інформаційних технологій і комп'ютерних наук із математикою та статистикою дало поштовх створенню *машинного навчання* та *штучного інтелекту* (ML & AI). Їхнє спільне поєднання (перетин) визначило фундамент для виникнення *Data Science* (DS) (рис. 3.2). Отже, щоб опанувати DS у конкретній галузі або для певного завдання, потрібно:

– володіти математикою, зокрема статистикою;

– уміти розробляти програмне забезпечення, насамперед на Python, R та ін.;

– мати ґрунтовні знання предметної сфери та чітко розуміти вимоги, обмеження й критерії ефективності обраних методів і технологій.

Методологія машинного навчання CRISP-DM

Життєвий цикл проєкту машинного навчання ділиться на шість етапів за методологією CRISP-DM. CRISP-DM – це абревіатура від *Cross-Industry Standard Process for Data Mining*, тобто міжгалузевий стандартний процес для добування даних. *Cross-Industry* – процес створення моделі МН повинен підходити компаніям із будь-якої галузі. *Standard Process* – описує узгоджену, повторювану послідовність кроків (методологію). *For Data Mining* – призначений для проєктів із добування даних і машинного навчання.

Процес починається з визначення бізнес-проблеми та переформулювання бізнес-проблеми з погляду цілі машинного навчання. Кінцевою метою процесу машинного навчання є отримання точних результатів для вирішення бізнес-проблеми. Кожен етап робочого процесу проілюстровано точками, які підсумовують ключові завдання.

На рис. 3.3 наведено фази процесу машинного навчання – методологія CRISP-DM, яка охоплює шість основних етапів:

1. Визначення бізнес-цілей (Define business goals).

Мета: перекласти бізнес-ціль на чітке аналітичне завдання.

Ключові кроки:

– визначити проблему, критерії успіху та обмеження (час, бюджет, нормативи);

– сформулювати аналітичні запитання й показники ефективності (KPIs, ROI);

– узгодити очікувані результати з усіма стейкголдерами, скласти план проєкту.

2. Дослідження й розуміння даних (Understand data).

Мета: сформувати інтуїцію щодо доступних даних та їхньої релевантності.

Ключові кроки:

– збір даних: підключення до БД, API-файлів відкритих даних;

– початковий огляд: статистика, розподіл, візуалізація, пошук аномалій, пропуски;

– оцінювання якості: повнота, точність, репрезентативність, упередження.

3. Підготовка даних (Prepare data).

Мета: перетворити «сирі» дані на чистий, узгоджений набір ознак.

Ключові кроки:

– очищення: усунення дублів, некоректних значень, обробка пропусків;

– інтеграція та збагачення: об’єднання різних джерел, feature engineering;

– трансформація: масштабування, кодування категорій, time-windowing тощо.

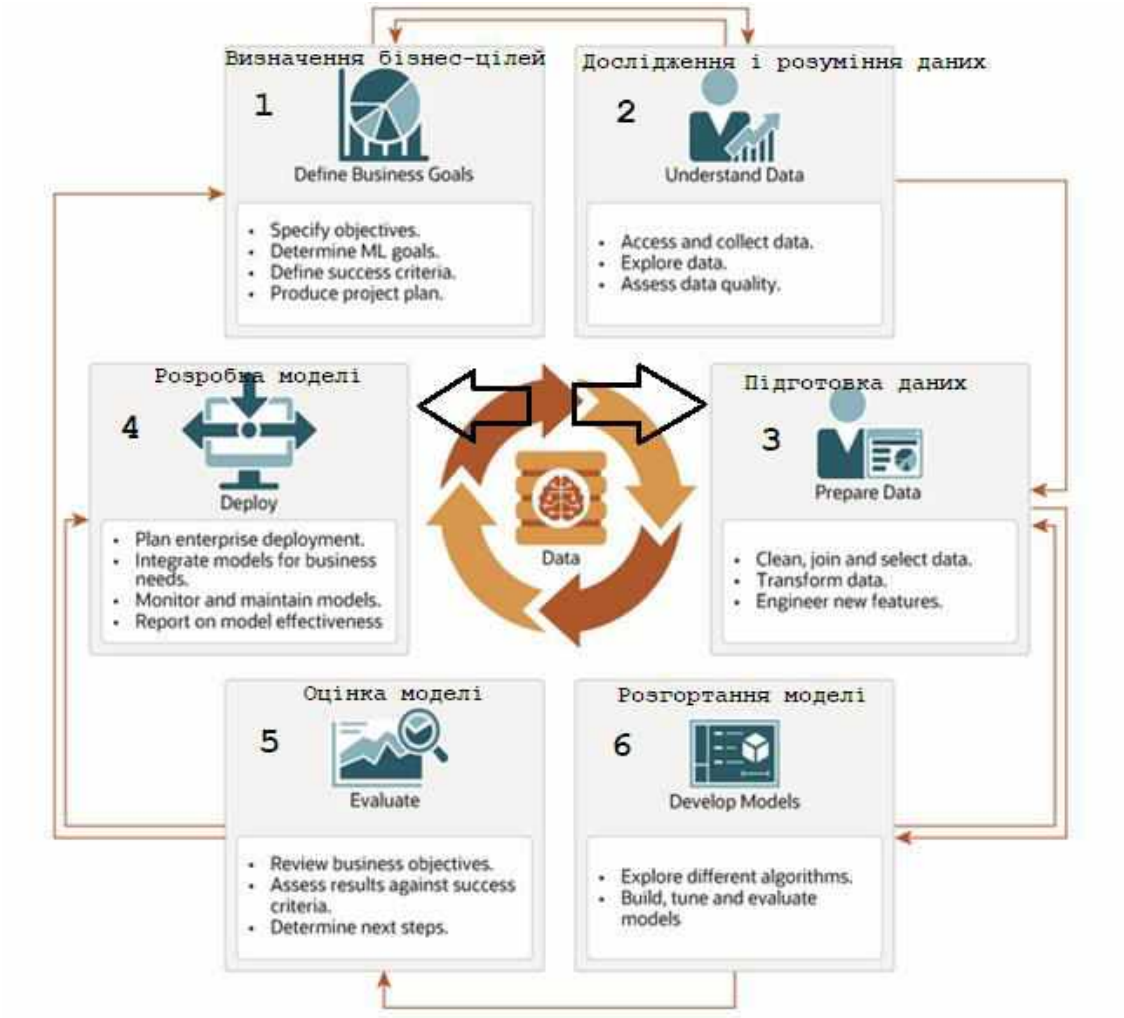


Рисунок 3.3 – Процес машинного навчання
за методологією CRISP-DM

4. Розробка моделі (Develop models).

Мета: підібрати та навчити моделі, здатні відповідати бізнес-цілям.

Ключові кроки:

– вибір алгоритмів (baseline → advanced): лінійні, дерева, ансамблі, CNN, RNN тощо;

– тренування з контрольованою валідацією (k-fold, time-series split та ін.);

– підбір гіперпараметрів (Grid / Random Search, Bayesian Opt тощо);

– фіксація експериментів (MLflow, Weights & Biases).

5. Оцінювання моделі (Evaluate).

Мета: визначити, чи відповідає модель бізнес-критеріям успіху.

Ключові кроки:

– порівняння з базовою лінією та цільовими KPI (MAE, F1, AUC, R^2 та ін.);

– перевірка стабільності (cross-validation) та узгодженості з доменною логікою;

– аналіз похибок, каузальний / сенситивний аналіз, explainability (SHAP).

6. Розгортання моделі (Deploy).

Мета: інтегрувати модель у реальний робочий процес і забезпечити її стійку роботу.

Ключові кроки:

– обрати формат розгортання: batch-pipeline, REST API, стримінговий сервіс;

– створити CI / CD-конвеєр (тести даних, тести моделі, automated rebuild);

– моніторинг у продакшені: дрифт даних і метрик, алерти, retraining triggers;

– документація й передача знань (Model Card, Data Sheet).

CRISP-DM надає спільну «мову» й структуру, яка допомагає командам розробників систематично проводити проєкти з аналітики даних і машинного навчання, забезпечуючи прозорість, повторюваність і орієнтацію на бізнес-результат.

3.1.3. Навчання «з учителем» і «без учителя»

Залежно від задачі, яку треба розв'язати, обирається відповідний алгоритм. Водночас для певного класу задач можна спробувати використати різні алгоритми для досягнення найбільшої точності прогнозування результату. У додатку А наведено методи, призначення використання й популярні алгоритми ML для розв'язання задач. Наявність чи відсутність умовного «учителя» (далі – *учителя*), тобто існуючої класифікаційної ознаки чи групи ознак МН, поділяється на такі види (рис. 3.4):

– навчання з учителем (*Supervised Learning*);

– навчання без учителя (*Unsupervised Learning*);

– навчання з підкріпленням (*Reinforcement Learning*).

У разі *навчання з учителем* (*Supervised Learning*) маємо вхідні дані, на підставі яких потрібно щось передбачити, і деякі гіпотези, а отже, готові правильні відповіді. У разі *навчання без учителя* маємо тільки дані, характерні властивості яких хочемо виділити, щоб розділити дані, наприклад, на групи.

Навчання з учителем застосовують, коли потрібно навчити машину розпізнавати об'єкти, явища, сигнали. Загальний принцип навчання з учителем такий: дивись на класифікаційну ознаку: набір вхідних даних № 1 дає значення класифікаційної ознаки «Так»; набір вхідних даних № 2 дає значення класифікаційної ознаки «Ні»; … і так перебрати всі комбінації входів і виходів (класифікаційної ознаки) для навчання, наприклад налаштування вагових коефіцієнтів конкретного методу (нейромережі, дерева рішень тощо) розпізнавати вибрану ознаку як на навчальній вибірці, так і на нових невідомих раніше даних.

Навчання без учителя (Unsupervised Learning) використовує принцип класифікації об'єктів на основі кількості інформації, значення етропії, результатом якого може бути висновок «ця річ схожа на іншу з такими значеннями ймовірностей». Алгоритми вивчають схожості й можуть виявляти відмінності та виконувати виявлення аномалій, розпізнаючи, що є незвичайним або несхожим.

Навчання з підкріпленням (Reinforcement Learning) використовують там, де потрібно правильно виконати поставлені завдання в зовнішньому середовищі, маючи безліч можливих варіантів дії. Наприклад, в комп'ютерних іграх, трейдингових операціях, для управління безпілотною технікою.

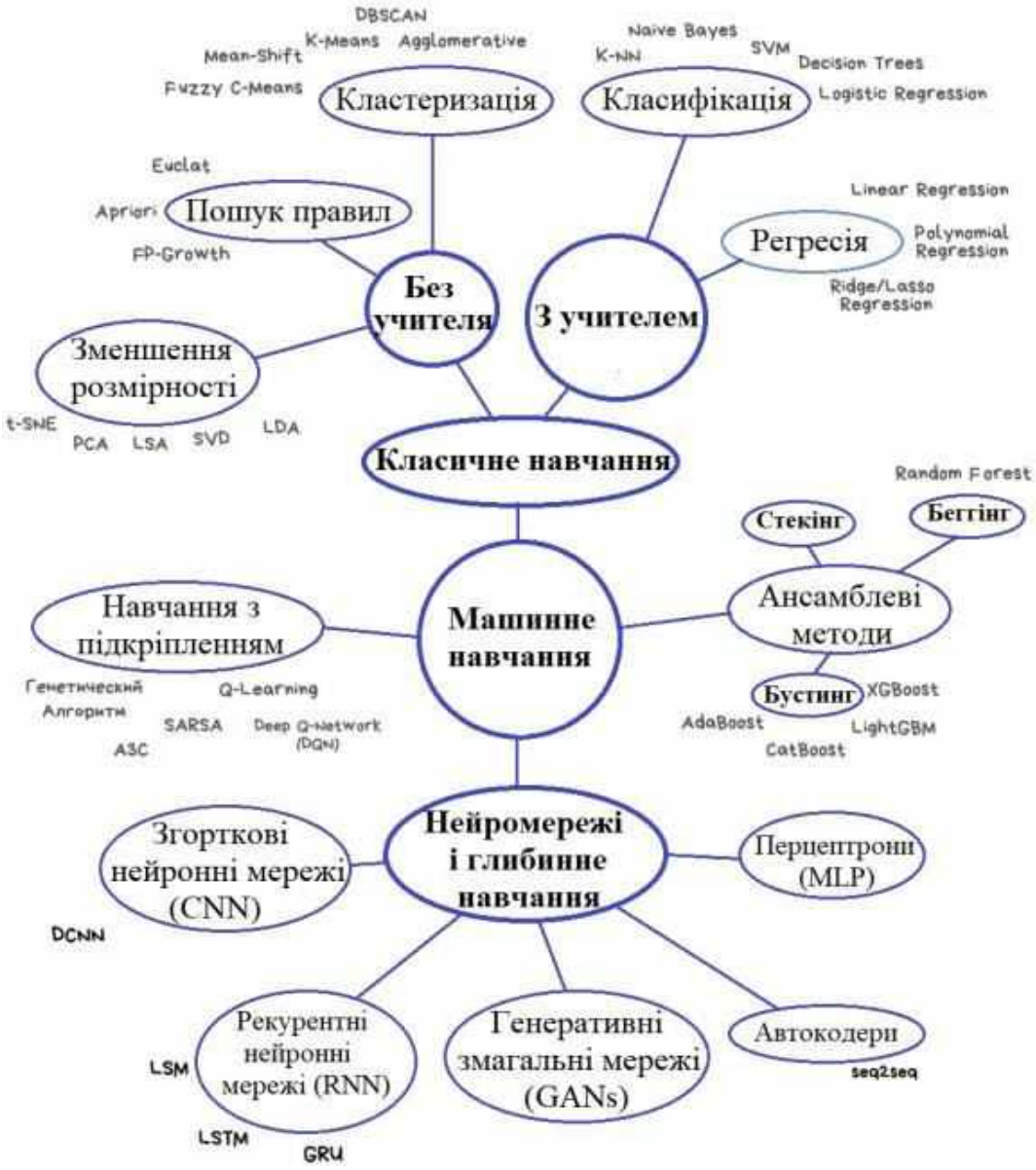


Рисунок 3.4 – Карта алгоритмів машинного навчання

За типом застосовуваних алгоритмів можна виділити два види навчання:

1. *Класичне навчання* – відомі й добре вивчені алгоритми навчання, розроблені загалом у 50-х роках XX ст., які використовувались для статистичних розрахунків у методах такого напряму, як *Data Mining*. Цей тип навчання підходить насамперед для

завдань класифікації, кластеризації, регресії тощо, які застосовуються для прогнозування, сегментації клієнтів та ін.

2. *Глибоке навчання і нейронні мережі – машинне навчання* нейронних мереж, які застосовуються там, де потрібні розпізнавання явищ, об'єктів, генерація зображень і відео, комп'ютерне бачення, складні алгоритми управління або прийняття рішень, машинний переклад та інші складні завдання (згорткові нейромережі, рекурентні найромежі, LSTM тощо).

Класичні задачі оптимізації в теорії систем зазвичай формулюються так: маємо вхідні дані, керовані змінні та контрольовані й неконтрольовані збурення й потрібно знайти екстремум (максимум чи мінімум) заданого критерію за певних обмежень.

Отже, типове формулювання задачі машинного навчання звучить так: маючи датасет (таблицю, текстовий файл, зображення, аудіо чи відео), потрібно побудувати й навчити інтелектуальну модель, яка оптимізує обрану метрику для заданої цільової ознаки. Додаткові обмеження можуть застосовуватися за потреби.

У машинному навчанні натомість виділяють лише набір вхідних даних і так звану «цільову ознаку» (англ. *target*). Цільових ознак може бути кілька, або ж за чергою кожен стовпець даних стає цільовим таргетом (наприклад, під час аналізу взаємозв'язку алергенів пацієнта).

3.1.4. Алгоритми машинного навчання

У рамках інтелектуального аналізу даних (IDA) також вирішуються завдання оптимального вибору моделей і технологій для виявлення складних закономірностей у різнорідних наборах даних (числових, текстових, графічних, динамічних) з метою отримання прогнозів чи інсайтів для ухвалення ефективних рішень у різних галузях.

У класичному машинному навчанні (*без нейронних мереж*) виділяють кілька основних груп алгоритмів. Ці методи широко застосовуються в різних задачах науки про дані та є базовими інструментами для студентів спеціальності «Комп'ютерні науки» (рис. 3.4). Нижче коротко розглянуто основні групи класичних

методів та їхні характеристики, а також наведено порівняльну таблицю ключових алгоритмів з їхнім принципом роботи, перевагами, недоліками та сферами застосування.

Лінійні методи. Алгоритми, що моделюють залежність між ознаками та цільовою змінною за допомогою лінійних функцій. Наприклад, лінійна регресія (для прогнозування числових значень) і логістична регресія (для класифікації). Лінійні моделі прості та інтерпретовані, проте обмежені здатністю представляти лише лінійні зв'язки між змінними.

Деревоподібні моделі. Методи, що використовують структуру дерева рішень для прогнозування. Класичним прикладом є дерево рішень, яке послідовно застосовує правила типу «if-then» на основі значень ознак для поділу даних на гілки. Дерева легко візуалізувати та пояснювати, вони здатні моделювати нелінійні залежності, але окремі дерева можуть перенавчатися. На основі дерев рішень побудовано й ансамблеві методи (випадковий ліс, бустинг), що поєднують багато дерев задля підвищення точності.

Методи найближчих сусідів (instance-based learning). Алгоритми, які зберігають навчальні приклади та приймають рішення на основі схожості нового прикладу з уже баченими. Найвідомішим методом є метод *k* найближчих сусідів (KNN), що класифікує об'єкт за допомогою «голосування» класів його *k* найближчих сусідів у просторі ознак. Такі методи прості в реалізації, проте повільні на великих масивах даних і чутливі до розмірності ознак.

Ансамблеві методи. Підходи, що поєднують кілька базових моделей для отримання кращого результату. Існують дві основні стратегії ансамблювання: бегінг (bagging) та бустинг.

Приклад бегінгу – випадковий ліс, який будує багато випадково різних дерев рішень і усереднює їхні відповіді.

Приклад бустингу – градієнтний бустинг, де прості моделі (зазвичай неглибокі дерева) додаються послідовно, кожна наступна намагається виправити помилки попередніх. Ансамблеві методи зазвичай дають вищу точність, зменшують перенавчання окремих моделей, але втрачають в інтерпретованості та потребують більше ресурсів.

Байєсівські методи. Алгоритми, які базуються на застосуванні теореми Байєса. Найпоширенішим є наївний байєсівський класифікатор, який припускає незалежність ознак і обчислює апостеріорні ймовірності класів для прийняття рішення. Байєсівські методи надзвичайно швидкі й ефективні на високовимірних даних (наприклад, у задачах класифікації тексту), хоча спрощене припущення незалежності ознак може не відповідати реальності.

Методи зменшення розмірності. Алгоритми, що слугують для скорочення кількості ознак під час збереження найбільш важливої інформації. Наприклад, метод основних компонент (PCA), який трансформує вихідні змінні в новий набір ортогональних компонент з максимальною дисперсією. Зменшення розмірності допомагає боротися з «прокляттям розмірності», пришвидшує навчання моделей та усуває шум, проте може ускладнювати інтерпретацію нових ознак і призводити до часткової втрати інформації.

У таблиці 3.1 приведено алгоритми МН / ML та їхнє призначення.

Таблиця 3.1 – Завдання, які вирішують алгоритми МН / ML, та їхнє призначення

Завдання	Призначення та використання	Популярні алгоритми
1	2	3
Класифікація	Спам-фільтри. Визначення мови. Пошук схожих документів. Аналіз тональності. Розпізнавання рукописних букв і цифр. Визначення підозрілих транзакцій	Наївний Байєс. Дерева розв'язку. Логістична регресія. K-найближчих сусідів. Машини опорних векторів
Регресія	Прогноз вартості цінних паперів. Аналіз попиту, обсягу продаж. Медичні діагнози. Будь-які залежності кількості від часу	Лінійна регресія. Поліноміальна регресія

Продовження таблиці 3.1

1	2	3
Кластеризація	Сегментація ринку (типів покупців, лояльності). Об'єднання близьких точок на карті. Стискання зображень. Аналіз і розмітки нових даних. Детектори аномальної поведінки	Метод K-середніх. Mean-Shift DBSCAN
Зменшення розмірності	Рекомендаційні системи. Красиві візуалізації. Визначення тематики й пошуку схожих документів. Аналіз фейкових зображень. Ризик-менеджмент	Метод основних компонент (PCA). Сингулярне розкладання (SVD). Латентне розміщення. Дирихле (LDA). Латентно-семантичний аналіз (LSA, pLSA, GLSA). t-SNE
Пошук правил	Прогнозування акцій і розпродаж. Аналіз товарів, які купуються разом. Розміщення товарів на полицях. Аналіз шаблонів поведінки на вебсайтах	Apriori Euclat. FP-growth
Навчання з підкріпленням	Самокеровані автомобілі. Роботи-пилососи. Ігри. Автоматична торгівля. Управління ресурсами підприємства	Q-Learning SARSA DQN. A3C. Генетичний алгоритм
Ансамблі	Там, де підходять усі класичні алгоритми (але працюють точніше). Пошукові системи. Комп'ютерний зір. Розпізнавання об'єктів	Random Forest Gradient Boosting
Нейронні мережі	Замість усіх вищенаведених алгоритмів. Визначення об'єктів на фото та відео. Розпізнавання й синтез мови. Опрацювання зображень, перенесення стилю. Машинний переклад	Перцептрон. Згорткові мережі (CNN). Рекурентні мережі (RNN). Дога коротка пам'ять (LSTM). Автоенкодери, LLM

Окрему групу складають: асоціативні правила, кластерний аналіз, дискримінантний аналіз, часові ряди (класичні моделі) (табл. 3.2):

Таблиця 3.2 – Класичні методи машинного навчання без учителя

Метод / сімейство	Принцип	Переваги	Недоліки	Типові приклади
1	2	3	4	5
Асоціативні правила: Apriori / FP-Growth (асоціативні правила)	Ітеративно знаходять часті набори ознак (itemsets) і виводять правила виду «якщо A, то B» з підтримкою *support* та довірою *confidence*	Інтуїтивні правила, легко пояснювати бізнесу. Працюють напряму з категоріальними даними (чеки, логи)	«Вибух» комбінацій → висока складність на великих наборах. Потрібен грамотний вибір порогів support / confidence; інакше – шум або занадто мало правил	*Market basket*-аналіз (розкладка товарів), перехресні продажі, рекомендації, виявлення шаблонів у мережевих журналах
K-means, ієрархічний, DBSCAN (кластеризація)	Максимізують внут-рішню схожість і мінімізують міжкластерну (K-means – середні; ієрархічний – дерево злиття; DBSCAN – щільні сфери)	Не потребує міток (label-free). Виявляє природну структуру даних; корисно для початкового дослідження	Чутливість до масштабу й вибору метрики. Потрібно задавати *k* (K-means) або ε/MinPts (DBSCAN)	Сегментація клієнтів / зображень, виділення тем у текстах (разом із LDA), групування генів / білків, anomaly detection (кластер-outliers)

Продовження таблиці 3.2

1	2	3	4	5
Дискримінантний аналіз. Лінійний / квадратичний дискримінантний аналіз (LDA / QDA)	Моделюють класові розподіли як (мульт) нормальні; обчислюють межу, що мінімізує ймовірність помилки: лінійну (LDA, спільна коваріація) або квадратичну (QDA, окремі коваріації)	Оптимальні в разі нормальності даних; швидкі й аналітичні рішення. Самі дають змогу зменшення розмірності (LDA-проєкції)	Припускають гаусові розподіли; нестійкі до відхилень і викидів. Погано працюють, якщо класи сильно перекриваються або коваріаційні матриці вироджені	Розпізнавання образів (faces, digits), біоінформатика, скоринг документів (topic vs. background), медична діагностика
Часові ряди (класичні моделі) ARIMA, SARIMA, Exponential Smoothing, Holt-Winters (класичні часові ряди)	Будують лінійну модель автокореляцій (AR), інтервального інтегрування (I), ковзного середнього (MA); розширення – сезонність (S) та комбіноване згладжування тренду / сезону	Добре теоретично обґрунтовані (стаціонар-ність, ACF / PACF). Порівняно інтерпретовані (тренд, сезонність, шум)	Лінійність і стаціонарність – обмеження на багато реальних процесів. Важке ручне підбирання (p, d, q) та сезонних параметрів; чутливі до структурних зламів	Прогноз продажів, фінансові та макроекономічні ряди, навантаження на сервери / мережі, попит на енергію

Питання для самоконтролю

1. Поясніть відмінності навчання з «учителем», «без учителя», «з підкріпленням».
2. Що входить у процедури дослідження та розуміння даних?
3. Наведіть математичні підходи до нормалізації даних.
4. Як розподіляється датасет на навчальну та тестову вибірки?

5. Наведіть приклади, інструментарій і форми візуалізації даних.

6. Опишіть на прикладі визначення бізнес-цілі.

7. Наведіть етапи розробки моделі.

8. Особливості вирішення проблем з оцінюванням аналітичних моделей.

9. Важливість достовірності та надійності отриманих моделей.

10. Важливість помилок аналізу в базовій статистиці.

11. Сутність оцінювання результатів аналізу.

3.2. Дослідження й розуміння даних на прикладах

3.2.1. Основні статистики розвідувального аналізу даних

Етап розуміння даних передбачає збір і дослідження даних, що охоплює завантаження даних і аналіз даних для вашої бізнес-проблеми. Важливо оцінити формати, розподіл даних, для чого їх можна завантажити у відповідні аналітичні прикладні системи (Statistica, Orange та ін.) або використати спеціальні бібліотеки, наприклад мов Python, R (matplptlib, seaborn та ін.), які мають широкий функціонал з дослідження даних.

Як зазначалось, розуміння даних здійснюється на основі так званого *розвідувального аналізу даних (РАД), під час якого досліджуються* основні властивості даних, наявності в них загальних закономірностей, розподілів і аномалій з використанням порівняно простих моделей.

Метою РАД є:

– виявлення основних структур і систематизація даних;

– виявлення відхилень та аномалій (викидів);

– перевірка основних статистичних гіпотез;

– побудова та дослідження даних з використанням порівняно простих моделей (регресій, дерев рішень).

Методи РАД застосовуються до усіх даних, а також до їхніх кластерів, тренувальних, валідаційних і тестових даних, за допомогою яких здійснюють:

– аналіз імовірнісних розподілів змінних;

– побудову й аналіз кореляційних матриць;

– факторний аналіз;

– дискримінантний аналіз;

– багатовимірне шкалювання та ін.

Зазвичай РАД може містити такі етапи:

1. Підрахунок кількісних показників у датасеті та за наявності:

– кількість рядків і стовпців усього та кількості пропущених значень у кожному стовпці;

– виявлення рядків, де є значний відсоток пустих, пропущених значень у різних ознаках, видалення таких рядків або заповнення їх середніми чи іншими значеннями;

– визначення форматів даних і приклади значень у кожному стовпці.

2. Побудова різних графіків для аналізу закономірностей щодо значень кожної ознаки (feature) та їхніх комбінацій тощо (бібліотеки matplotlib, seaborn та ін.).

Дослідивши дані, зможете визначити, наскільки ефективно їх можна використовувати для вирішення бізнес-проблем. Надалі можете за потреби видаляти деякі ознаки або додавати додаткові, спеціально обчислені або взяті із інших джерел. Тобто це етап для виявлення проблем щодо якості даних, як-от: чи повністю дані описують бізнес-проблему?; чи є в даних відсутні значення?; які типи помилок існують у даних і як їх можна виправити?

Найважливіші показники, що характеризують дані: дисперсія, середньоквадратичне відхилення; середнє; мода; медіана.

Дисперсія (*Variance*), σ^2

Дисперсія: чим більша дисперсія, тим «розкиданіші» дані навколо середнього, більша площина даних у двовимірному поданні; величина вимірюється в **квадраті одиниць** (якщо *x* вимірюється в гривнях, то дисперсія – у *гривнях*^{2})

Позначення	Формула	Що означає
Генеральна (теоретична) дисперсія σ^2	$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$	N – кількість усіх елементів у сукупності. μ – істинне середнє. Показує середній квадрат відхилень кожного значення від математичного сподівання
Вибіркова дисперсія s^2	$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$	*n* – обсяг вибірки, $\overline{\boldsymbol{x}}$. Ділення на *n–1* (коефіцієнт Бесселя) робить оцінку незміщеною

Середньоквадратичне відхилення (*Standard Deviation*), σ сігма

Середньоквадратичне відхилення (СКВ). **Інтерпретація:** повертає масштаб вимірювання до оригінальних одиниць (грн, см, °C тощо). Це «середнє» відхилення спостережень від середнього значення, але виміряне не абсолютною різницею, а коренем із середнього квадрата різниць. Часто показник СКВ використовують як міру ризику в ризик менеджменті

Позначення	Формула	Відношення до дисперсії
Генеральне СКВ, σ	$\sigma = \sqrt{\sigma^2} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$	Квадратний корінь із генеральної дисперсії
Вибіркове СКВ, s	$s = \sqrt{s^2} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$	Квадратний корінь із вибіркової дисперсії

Закон розподілу. Знання закону розподілу дає змогу правильно оцінити, наскільки рідкісними є певні спостереження. Багато статистичних моделей (лінійна регресія, класифікатори на основі гаусівських оцінок, аналіз часових рядів із гаусівським шумом) використовуються з припущенням про нормальність помилок або про нормальну форму базового розподілу. Нормальним називають розподіл імовірностей безперервної випадкової величини, яка має щільність (3.1), рис. 3.5:

$$f(x;\, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}}\, \exp\Big[-\frac{(x-\mu)^2}{2\sigma^2}\Big], \qquad (3.1)$$

де:

μ – середнє (location, *shift*). Зсуває криву ліворуч / праворуч без зміни форми;

σ – середньоквадратичне відхилення (СКВ). Збільшує або звужує вершину розподілу:

більше σ → ширша, нижча, «пласкіша» крива;

менше σ → вужча, вища, «загостреніша».

Якщо реальні дані значно відхиляються від нормального розподілу, оцінки моделі стають менш надійними. Довірчі інтервали для середнього, різниці середніх, коефіцієнтів регресії тощо будуються з використанням властивостей нормального закону.

Важливість відповідності даних нормальному закону розподілу пояснюється зручністю математичних властивостей. Нормальний розподіл повністю задається двома параметрами (μ, σ), з яких легко виводити всі статистики. Можна вирахувати моменти (центр, дисперсію, квантілі). Лінійні перетворення (зсув, масштаб) зберігають нормальність.

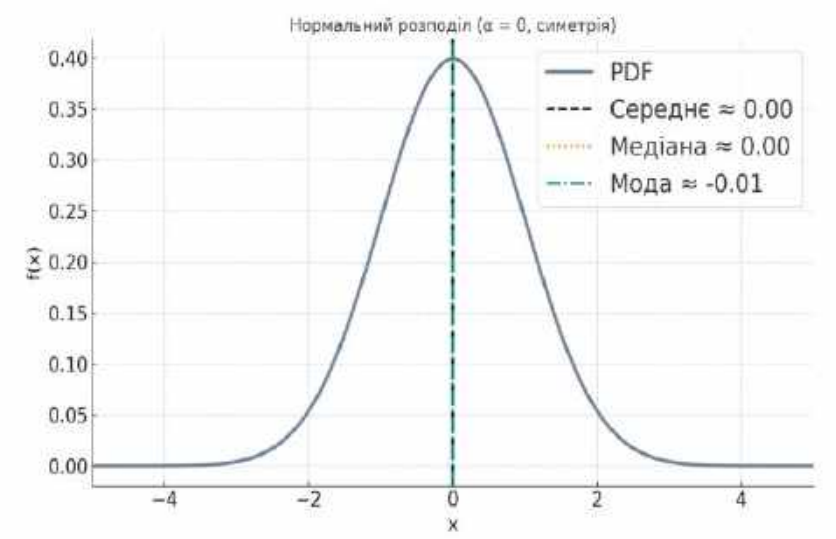


Рисунок 3.5 – Графік щільності ймовірності випадкової величини під час стандартного нормального розподілу *(Probability Density Function, PDF)*

Основні методи перевірки статистичних гіпотез. Більшість класичних методів (t-тест, F-тест, довірчі інтервали, аналіз дисперсії) було розроблено саме для нормально розподілених даних. Вони мають добре вивчені характеристики й за нормальності є оптимальними (наприклад, «найсильніші» тести з максимальною потужністю).

Універсальність у прикладних задачах. У багатьох природних, соціальних, технічних процесах шум або відхилення від ідеалу дійсно близькі до нормального: фізичні вимірювання, біологічні ознаки, фінансові дохідності за відповідних умов. Тому першим кроком у будь-якому аналізі має бути *оцінка форми розподілу* (гістограма, QQ-плот, тести нормальності: Шапіро-Уілк, Андерсона-Дарлінга). Це дасть основу для вибору коректних інструментів перевірки статистичних гіпотез.

Мода (Mode): це значення, яке з'являється найчастіше в наборі даних. Наприклад, якщо у нас є набір даних {1, 2, 2, 3, 4, 4, 4, 5}, то мода – 4. У стандартному нормальному розподілі мода також дорівнює 0. Це через те, що розподіл симетричний і його вершина розташована в точці нуль.

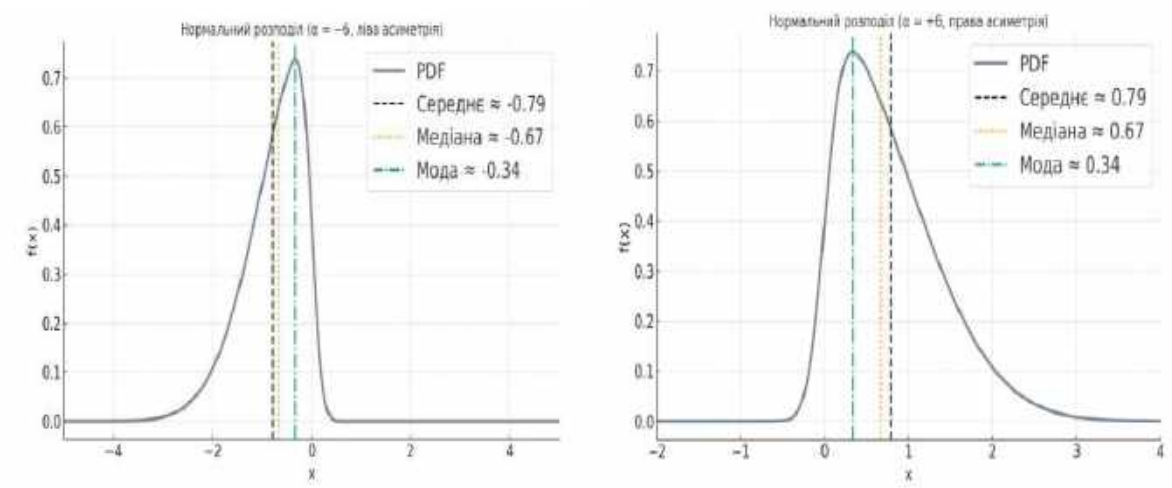
Медіана (Median): це значення, яке розділяє набір даних на дві рівні частини, коли вони упорядковані за зростанням. Наприклад, у наборі {1, 2, 3, 4, 5}, медіана – 3. Зокрема, під час стандартного нормального розподілу (розподіл з параметрами mean = 0 та standard deviation = 1) мода, медіана й середнє мають одне значення «0».

Медіана також дорівнює 0 в стандартному нормальному розподілі. Оскільки розподіл симетричний, 50 % значень розташовані ліворуч від нуля, а 50 % – праворуч, тому медіана дорівнює 0.

Середнє (Mean): середнє арифметичне в стандартному нормальному розподілі також дорівнює 0. Це особливість нормального розподілу: середнє, медіана й мода збігаються в нього завдяки симетрії. Отже, у стандартному нормальному розподілі мода, медіана й середнє мають однакові значення та дорівнюють нулю. Це свідчить про те, що розподіл симетричний навколо центральної точки.

На рис. 3.6 показано співвідношення середнього, медіани й моди за різних асиметрій у розподілі.

Параметри, що характеризують асиметрію та «хвости» розподілу, такі: γ_1 – коефіцієнт асиметрії (skewness). γ_2 – ексцес (kurtosis). Для прийняття рішення відповідності даних нормальному закону можна користуватись таким правилом: якщо параметри γ_1 та γ_2 лежать у межах $\pm$ 0.5, то можна сміливо застосовувати класичні параметричні процедури.



а) б)

Рисунок 3.6 – Закони розподілу із сильною асиметрією

а) ліва асиметрія, $\gamma_1 = -6$; б) права асиметрія, $\gamma_1 = +6$

1. *Відповідність нормальному розподілу.*

Якщо $|\gamma_1| < 0.2$ і $|\gamma_2| < 0.2$, розподіл майже не відрізняється від ідеальної форми нормального розподілу – «хвости» й центр практично збігаються.

2. *Добра відповідність нормальному розподілу.*

Межі $|\gamma_1| < 0.5$, $|\gamma_2| < 0.5$ вважаються достатньо жорсткими для більшості параметричних методів (t-тест, ANOVA, лінійна регресія). Помітні, але невеликі відхилення не спотворюють результати суттєво.

3. *Мінімальна відповідність нормальному розподілу.*

Деякі практики приймають навіть межі $|\gamma_1| < 1$ і $|\gamma_2| < 1$ як достатні, особливо для великих вибірок, де центральна гранична теорема «вирівнює» розподіл середнього.

Розподіл даних і гістограми

Для базового розуміння набору даних, наприклад даних у окремій колонці таблиці бази даних, зазвичай використовують *гістограми й бокс-плоти (бочкові діаграми)*, які є важливими інструментами для дослідження та візуалізації розподілу даних. Вони допомагають зрозуміти основні характеристики набору даних, виявляти викиди, аномалії та специфічні особливості. Ось кілька важливих аспектів їхнього використання.

Візуалізація розподілу даних: гістограми дають змогу швидко отримати уявлення про те, як розподілені дані.

Виявлення моди та основних піків: мода й піки допомагають ідентифікувати найбільш типові значення або діапазони в наборі даних.

Перевірка нормальності розподілу: гістограми дають змогу оцінити, чи розподіл даних наближається до нормального.

Виявлення викидів: вони можуть підказати, чи є в наборі даних викиди або аномалії.

Бокс-плоти (бочкові діаграми)

Візуалізація основних статистик: бокс-плоти демонструють основні характеристики даних, як-от медіана, квартилі, мінімум і максимум.

Виявлення викидів: «вуса» бокс-плоту (бочкової діаграми) розраховані так, щоб вони відображали границі нормального розподілу даних. Вони допомагають ідентифікувати викиди – точки даних, які виходять за межі цих границь.

Порівняння розподілу даних між групами: бокс-плоти дають змогу порівняти розподіл даних між різними категоріями або групами.

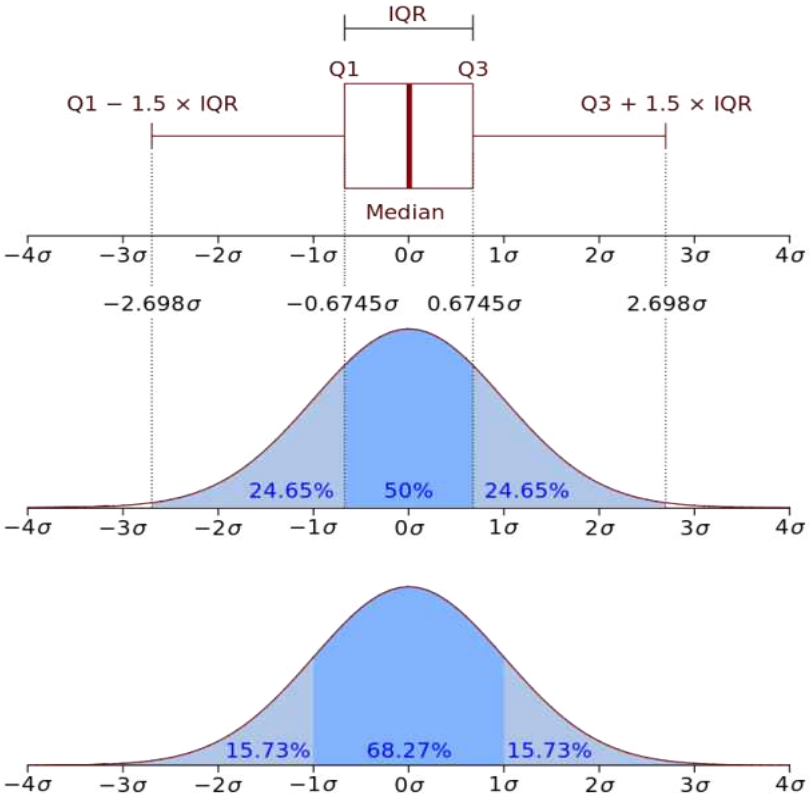


Рисунок 3.7 – Нормальний розподіл і бочкосхожий графік

Квартилі (Quartiles): квартилі розділяють набір даних на чотири рівні частини. Q1 (перший квартиль) – це значення, що розділяє першу чверть даних, Q2 (медіана) розділяє другу та третю чверті, Q3 (третій квартиль) розділяє останню чверть даних.

Міжквартильний розмах (IQR): це різниця між третім (Q3) та першим (Q1) квартилями. Він представляє ширину центральної частини розподілу даних, куди входить 50 % усіх значень (рис. 3.7).

Щодо розрахунку вусів бокс-плоту, вони зазвичай відображають відстань між квартилем Q1 (25-й персентиль) і квартилем Q3 (75-й персентиль). Тобто ця відстань є *міжквартильним розмахом* (*IQR* Q3 – Q1), в межах якого розміщено 50 % усіх значень. Ця відстань множиться на коефіцієнт (зазвичай 1,5 або 2), і вище та нижче цих значень встановлюються так звані «вуса» діаграми (полоски). Це дає змогу виявити викиди даних, які виходять за межі нормального розподілу. Бокс-плоти є ефективним інструментом для виявлення аномалій і специфічних особливостей у даних.

Приклад. Використання Python для дослідження даних.

Згенеруємо за допомогою NamPy масив для дослідження (лістинг 3.1).

Лістинг 3.1

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
# Генеруємо випадковий набір даних
np.random.seed(0)
data = np.random.normal(loc=100, scale=20, size=1000)
df = pd.DataFrame({"Value": data})
```

– *data = np.random.normal(loc = 100, scale = 20, size = 1 000):* створює масив даних `data`, який містить 1 000 випадкових значень з нормальним розподілом. Параметр `loc` визначає середнє значення розподілу, `scale` – стандартне відхилення, а `size` – кількість згенерованих значень;

– *np.random.seed(0):* встановлює початкове значення для генератора псевдовипадкових чисел. Встановлення одного й того

самого початкового значення забезпечує відтворюваність результатів під час використання генератора псевдовипадкових чисел. Тобто якщо запустите цей код з однаковим значенням `np.random.seed(0)`, отримаєте ті самі випадкові дані.

Найпростішим варіантом дослідження даних є використання функції *describe* бібліотеки *Pandas* для побудови описової статистики ***df.describe()***

На рис. 3.8 показано результати виконання цієї команди із обчисленими показниками. У межах інтерквартильного розмаху Q3 – Q1, який охоплює значення від 86,03 до 112,14, перебуває 50 % усіх значень масиву. У межах Q3 + 1,5IQR і Q1 – 1,5IQR – по 24,65 % значень. Викиди, які виходять за межі нормального розподілу, перебувають у межах > Q3 + 1,5 × IQR і < Q1 – 1,5 × IQR.

	Value
count	1000.000000
mean	99.094866
std	19.750541
min	39.077139
25%	86.031599
50%	98.839439
75%	112.139012
max	155.187102

Рисунок 3.8 – Обчислення статистичних показників функцією *df.describe()*

У лістингу 3.2 показано основну частину коду для розрахунку основних статистичних показників згенерованого масиву випадкових даних з використанням бібліотеки *NumPy*.

Лістинг 3.2

Розрахунок статистичних показників масиву даних NumPy

```
# Обчислимо медіану, квартилі, IQR, моду,
#середнє квадратичне відхилення, мінімум та максимум
median = np.median(data)
mean = np.mean(data)
q1 = np.percentile(data, 25)
q3 = np.percentile(data, 75)
iqr = q3 - q1
mode = stats.mode(data)[0][0]
std_dev = np.std(data)
minimum = np.min(data)
maximum = np.max(data)
```

На рис. 3.9 показано поєднання гістограми й бочкової діаграми для згенерованого масиву нормально розподіленого масиву випадкових чисел із середнім значенням 99,09, та їхні основні статистичні показники.

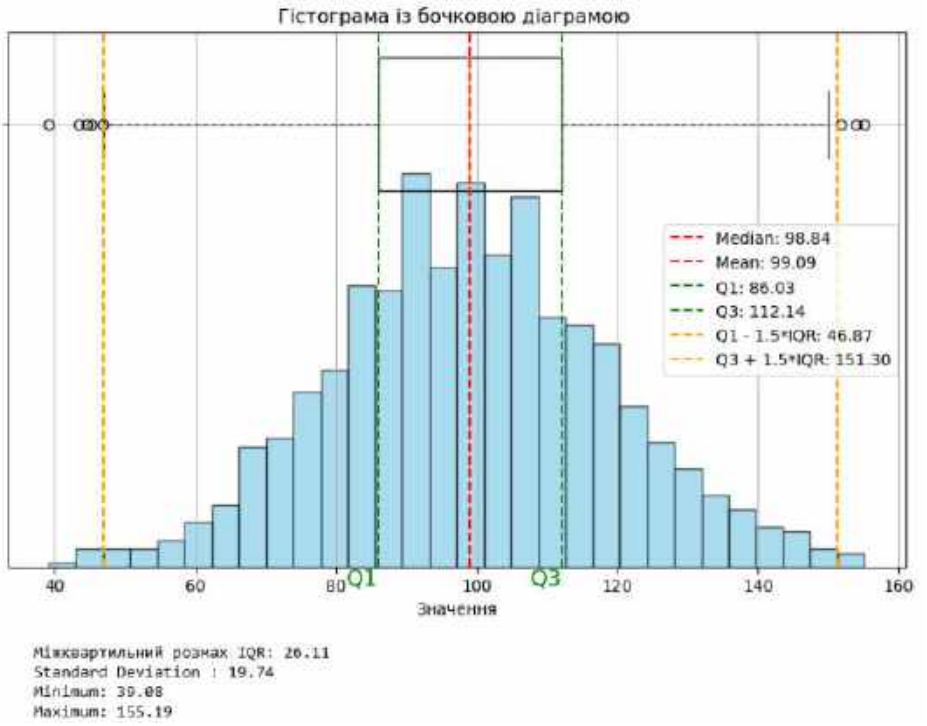


Рисунок 3.9 – Приклад поєднання гістограми й бочкової діаграми

median: медіана даних; *mean:* середнє значення даних; *q1:* перший квартиль; *q3:* третій квартиль; *iqr:* міжквартильний діапазон ниця між третім та першим квартилем; *mode:* мода даних; *std_dev:* стандартне відхилення даних (середньоквадратичне відхилення, СКВ); *minimum:* мінімальне значення в даних; *maximum:* максимальне значення в даних.

Для обчислення параметрів розподілу варто використати інструменти Pandas:

```
import pandas as pd
df = pd.Series(data)
μ    =  df.mean()        # Середнє
σ²   =  df.var(ddof=0)  # Генеральна дисперсія
s²   =  df.var(ddof=1)  # Вибіркова дисперсія
σ    =  df.std(ddof=0)  # СКВ генеральної сукупності
s    =  df.std(ddof=1)  # СКВ вибіркової сукупності
γ₁   =  df.skew()        # Асиметрія(Skewness)
γ₂   =  df.kurtosis()    # Ексцес   (Kurtosis)
```

Можна скористатись функціями з модуля *scipy.stats*, які за стандартним налаштуванням можуть повертати зміщені (biased) оцінки та повний момент четвертого порядку (raw kurtosis). Параметри *bias і fisher* контролюють саме ці моменти. *fisher = True:* приводить результат саме до *excess kurtosis* (віднімає 3), так що для нормального розподілу отримуємо 0. *bias = False:* аналогічно до *skew*, застосовує корекції, щоб оцінка стала *незміщеною* для вибірки.

У лістингу 3.3 показано обчислення параметрів розподілу для цього масиву:

Лістинг 3.3

```
# Обчислимо показники розподілу даних
n = len(data)                                #Розмір вибірки
""" ddof=0 – формула для всієї генеральної сукупності.
    ddof=1 – ділення на n-1  (вибіркова, «незміщена» оцінка)"""
mean = df["Value"].mean()
pop_variance = df["Value"].var(ddof=0)      # генеральна дисперсія
sample_variance = df["Value"].var(ddof=1)   # вибіркова дисперсія
pop_std = np.sqrt(pop_variance)             # СКВ генеральної сукупності
sample_std = df["Value"].std(ddof=1)        # СКВ вибіркової сукупності

# Асиметрія та ексцес розподілу
skewness          = stats.skew(data, bias=False)
excess_kurtosis   = stats.kurtosis(data, fisher=True, bias=False)

```

Отримані результати обчислення параметрів розподілу:

```
Розмір вибірки              ( n) : 1000.00
Середнє                       (μ)  : 99.09
Генеральна дисперсія         σ²  : 389.69
Вибіркова дисперсія          s²  : 390.08
СКВ генеральної сукупності σ  : 19.74
СКВ вибіркової сукупності  s   : 19.75
Асиметрія(Skewness)          (γ₁) : 0.034
Ексцес   (Kurtosis)           (γ₂) : -0.041
```

Значення параметрів Асиметрії та Ексцесу підтверджують нормальність розподілу даних. Рекомендуються також такі бібліотеки для дослідження даних:

PandasProfiling (PP) – бібліотека для візуалізації та аналізу даних за допомогою бібліотеки Pandas. Усе можна зробити однією командою ProfileReport, якій передається дата фрейм з даними. Результат можна зберегти в html-формат і потім переглядати в зручний спосіб у браузері:

```
import pandas as pd
from pandas_profiling import ProfileReport
report = ProfileReport(data)
report.to_file('report.html')
```

AutoViz – бібліотека для автоматичної візуалізації даних у Python. AutoViz визначає тип графіка для кожної змінної залежно від її характеристик. Числові змінні можуть бути подані як гістограми, лінійні графіки, діаграми розсіювання або як категоріальні – у формі різних діаграм, стовпчикових графіків, які можуть надати корисну візуальну інформацію.

SweetViz – бібліотека для візуалізації та аналізу даних у середовищі Python, яка автоматично будує гістограми для числових і категоріальних змінних. У цій бібліотеці є можливість аналізу зв'язку між числовою та категоріальною ознаками через побудову графіків відносності. Бібліотека дає змогу визначити зміну числових характеристик залежно від категоріальних ознак.

3.2.2. Підготовка даних: очищення, навчальна, тестова вибірки

Очищення даних

Очищення (*data cleaning / data wrangling*) – це перетворення «сирих» даних на послідовний, коректний і готовий до моделювання набір. Типові цілі:

надійність – прибираємо помилки, пропуски, дублікати;

узгодженість – вирівнюємо формати дат, категорій, одиниць виміру;

якість ознак – створюємо або перетворюємо колонки так, щоб модель «бачила» релевантну інформацію.

Підготовча фаза передбачає завершення обробки даних і охоплює всі завдання, пов'язані з переведенням даних у формат, який можна використовувати для створення моделі. Завдання з підготовки даних, імовірно, виконуватимуться кілька разів, ітеративно, а не в будь-якому встановленому порядку. Завдання можуть охоплювати вибір стовпців (атрибутів), а також вибір рядків у таблиці. Ви можете створювати подання для об'єднання даних або матеріалізації даних за потреби, особливо якщо дані збираються з різних джерел. Щоб очистити дані, знайдіть недійсні значення, значення зовнішнього ключа, яких немає в інших таблицях, а також відсутні та викидні значення. Щоб уточнити дані, можете застосувати такі перетворення, як агрегація, нормалізація, узагальнення та конструкції атрибутів, потрібні для вирішення проблеми машинного навчання. Наприклад, можете перетворити *Date_of_birth*-стовпець на *Age*; можете вставити середній дохід у разі, коли *Income*-стовпець нульовий; можете відфільтрувати рядки, що представляють викиди в даних, або відфільтрувати стовпці, які мають забагато відсутніх або ідентичних значень.

Крім того, потрібно додати нові обчислені атрибути, які краще описують проблему. Цей процес називається розробкою функцій. Наприклад, замість використання *суми покупки* можна створити новий атрибут: «*Кількість разів, коли сума покупки перевищує 500 доларів США за 12 місяців*». Клієнти, які часто роблять значні покупки, також можуть бути пов'язані з клієнтами, які відповідають або не відповідають на пропозицію.

Невеликі таблиці можна очищувати в Excel, проте для підвищення ефективності можна використати спеціалізовані інструменти або пакети мов програмування *R, Python* тощо. Наприклад, *OpenRefine* – безкоштовна програма з відкритим кодом (http://openrefine.org).

Унікальна функція *OpenRefine* – виявляти різні варіанти написання однієї назви (кластеризувати текстові дані) чи дати й зводити їх до одного вигляду. Досить корисно для виявлення друкарських помилок під час ручного введення даних. Крім того,

OpenRefine може виконувати функції *Excel* чи *LibreOffice* набагато більших даних, зокрема:

– повертати дані із рядків у стовпці, і навпаки (транспонувати таблиці);

– знаходити й виправляти помилки в наборі даних;

– розділяти дані з однієї колонки на декілька;

– здійснювати прості арифметичні операції;

– об’єднувати кілька наборів даних в один;

– конвертувати у формати відкритих даних (функція «Export»).

Також для очищення даних можна використовувати ресурс «Tidyr»: http://tidyr.tidyverse.org/

У таблиці 3.3 подані основні кроки та приклади з використанням пакетів: *pandas; numpy; sklean*. Під час завантаження зазначених пакетів *pandas, numpy*, відповідно, з іменами *pd, df*, показаними в лістингу 3.4, приклади надані в таблиці для датафрейму із іменем *df*.

Лістинг 3.4

```
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.impute import SimpleImputer
```

Таблиця 3.3 – Підготовка та очищення даних

Крок	Що робимо і як	Приклади
1	2	3

1. Первинний огляд даних	.info(), .describe(),
	# Прогляд даних
	print(df.info())
	print(df.describe(include="all").T.head())
		Зрозуміти типи, обсяг, статистику даних. Візуалізація пропусків

2. Усунення дублікатів	df.drop_duplicates()	Записи дублікати, дані завантажені двічі тощо

Продовження таблиці 3.3

1	2	3
3. Виправлення типів	pd.to_datetime, astype('category')	Дати, булеві поля, код країни
4. Робота з пропусками	# вилучити рядки / колонки # заповнити (mean/median/mode, KNN, # регресією). *numeric_pipe = Pipeline(steps=[* ("imputer", SimpleImputer(strategy="median")), ("scaler", StandardScaler())])	Заповнення відсутньої ціни товару середнім значенням
5. Стандартизація текстових значень	df['city_std'] = (df['city'] .str.replace(r'(?i)киев|київ', 'Київ', regex=True) .str.replace(r'(?i)львів|lviv', 'Львів', regex=True))	Заміна: «Киев », «київ» на «Київ». Заміна: «львів», «Lviv» на «Львів»
6. Видалення помилок і аномалій	# Видалення помилкових значень # (e.g., < 0 or > 100) df_clean_age = df[(df['age'] >= 0) & (df['age'] <= 100)].copy() # Видалення аномальних значень за # допомогою методу IQR Q1 = df_clean_age['transaction_usd'].quantile(0.25) Q3 = df_clean_age['transaction_usd'].quantile(0.75) IQR = Q3 – Q1 lower_bound = Q1 – 1.5 × IQR upper_bound = Q3 + 1.5 × IQR df_clean = df_clean_age[(df_clean_age['transaction_usd'] > = lower_bound) & (df_clean_age['transaction_usd'] < = upper_bound)].copy()	Методи: IQR-метод, z-score. Приклад помилок і аномалій: Вік = –5, Транзакція $10 000 000

Підбиваючи підсумок, на цьому етапі ви повинні:

– очистити, об’єднати та вибрати потрібні дані;

– перетворити за потреби дані в потрібний формат;

– визначити за потреби нові атрибути.

Як на вході, так і виході алгоритмів можуть бути абсолютно різнорідні величини. Очевидно, що результати моделювання не повинні залежати від одиниць вимірювання цих величин. Проте немає єдиного алгоритму, здатного охопити всі можливі варіанти постановки й розв’язання таких задач. У реальних задачах можуть бути такі варіанти:

Варіант 1. Існує готовий датасет (наприклад, результати медичних аналізів), а мета – підвищити ефективність лікування чи точніше оцінити поширення захворювання. Однак конкретні завдання та показники для вимірювання успіху ще треба самостійно визначити.

Варіант 2. Чітко сформульовано завдання (наприклад, покращити точність щоденного прогнозу кількості нових випадків COVID-19), але невідомо, які чинники потрібно врахувати. Отже, треба зібрати відповідний датасет, а потім уточнити постановку задачі, адже не всі відповідні дані можуть бути доступні.

Другий сценарій трапляється частіше та є більш загальним, тоді як перший – його спрощеним варіантом.

3.2.3. Формування наборів даних для навчання

Формування трьох наборів даних – навчального, валідаційного та тестового – є ключовим етапом у машинному навчанні.

Навчальний датасет (англ. *training set*) використовується для безпосереднього навчання моделей.

Валідаційний датасет (англ. *validation set*) слугує для вибору найкращої конфігурації чи гіперпараметрів серед кандидатів.

Тестовий датасет (англ. *test set*) застосовується лише до фінальної, оптимальної моделі, щоб оцінити її узагальнювальну здатність.

У змагальних задачах традиційно виділяють усі три набори, тоді як в індустріальних проєктах часто роблять поділ лише на *навчальний* і *валідаційний* (або на *навчальний* і *тестовий,* де тестовий виконує роль валідаційного).

Типові схеми розподілу даних на навчальний і тестовий датасет:

75 / 25 % або *80 / 20 %* (навчання / валідація) на практиці зустрічаються найчастіше.

70 / 30 % (іноді навіть *60 / 40 %)* – коли дані досить однорідні й модель надто легко «запам'ятовує» зразки.

90 / 10 % – використовується, якщо загалом мало даних і потрібно максимально зберегти обсяг для тренування.

Зазвичай вибір валідаційних зразків виконується рандомно, але для часових рядів (коли важлива послідовність) валідаційним вважають останній фрагмент ряду, щоб перевірити здатність моделі передбачати майбутні значення на основі попередніх.

Приклад коду для формування навчального, валідаційного та тестового наборів даних у пропорції: навчальні дані 60 %; валідаційні дані 20 %; тестові дані 20 % подано з поясненнями в лістингу 3.5.

Лістинг 3.5

```python
# Приклад коду для формування навчального, валідаційного
# та тестового наборів даних у пропорції :
# навчальні дані 60%; валідаційні  дані 20%; тестові дані 20%
# ВАРІАНТ 1
import pandas as pd
from sklearn.model_selection import train_test_split

#  DataFrame df із ознаками (features) та стовпцем 'target' із цільовою змінною
df = pd.read_csv('data.csv')  # Формуємо датафрейм Pandas

# Розділимо дані на X (ознаки) і y (цільовий стовпець)
X = df.drop(columns=['target'])
y = df['target']

# 1. Спочатку відокремимо тестовий набір (20%) від решти (80%)
X_temp, X_test, y_temp, y_test = train_test_split(
    X, y,
    test_size=0.20,       # 20% піде в тестовий набір
    random_state=42,      # для відтворюваності індексів
    stratify=y            # зберігаємо пропорції класів (якщо класифікація)
)

# 2. Далі поділимо залишок (80%) на навчальний (60%) та валідаційний (20%)
#    Щоб вийшло 60/20/20 від загального обсягу, беремо test_size=0.25 від тимчасового набору:
#    0.25 * 80% = 20%
X_train, X_val, y_train, y_val = train_test_split(
    X_temp, y_temp,
    test_size=0.25,  # 25% від X_temp → 0.25*80% = 20% від початкових даних
    random_state=42,
    stratify=y_temp
)
```

Варіант поділу 2 (лістинг 3.6) розбиває відразу датасет на навчальний (60 %), валідаційний (20 %) і тестовий (20 %).

Лістинг 3.6

```
    Варінт поділу 2 датасету.
    df = pd.read_csv('data.csv')
X, y = df.drop('target', axis=1), df['target']
    # Розбиваємо датасет відразу на навчальний (60%),
    # валідаційний (20%) і тестовий (20%)
    X_train, X_temp, y_train, y_temp =
train_test_split(
        X, y,
        test_size=0.40,   # 40% піде на валідацію + тест
        random_state=42,
        stratify=y # зберегти пропорції, якщо це
класифікація
    )
    X_val, X_test, y_val, y_test = train_test_split(
        X_temp, y_temp,
        test_size=0.50, # порівну розподілити тимчасовий
набір
        random_state=42,
        stratify=y_temp
        )
```

3.2.4. Нормалізація (стандартизація) даних

Як відомо, в масивах даних, які накопичуються, і в базах даних оперативні дані фіксуються необробленими, тобто в різних шкалах, типах і діапазонах. Для однакового трактування аналітичним методом значень даних усі вхідні та вихідні величини мають бути приведені до одного масштабу. Такий процес називається нормалізацією (ще зустрічаються терміни «нормування», «стандартизація»).

Цей процес полягає в зміні діапазону, у межах якого розподілені значення деякої ознаки, до іншого, зазвичай меншого діапазону, наприклад [–1; 1] або [0; 1]. Крім того, для підвищення швидкості та якості налаштування параметрів моделі машинного навчання корисно провести додаткове попереднє опрацювання даних, що вирівнює розподіл значень ще до початку етапу налаштування параметрів.

Ось декілька причин, чому нормалізація важлива для машинного навчання:

1. Забезпечення стабільності та швидкості навчання моделі – нормалізація даних допомагає уникнути чималих різниць між значеннями ознак, що може спричинити повільне навчання моделі або проблеми зі збіжністю.

2. Зменшення впливу шуму – нормалізація даних допомагає зменшити вплив великих значень або викидів на результат моделі.

3. Забезпечення правильного розподілу даних – деякі алгоритми машинного навчання (наприклад, нейронні мережі) працюють краще, коли вхідні дані мають нормальний розподіл.

Примітка! Досить не всі алгоритми потребують нормалізації. Наприклад, дерева рішень або лінійна регресія не потребують приведення входів і виходів до єдиного масштабу, а ось у нейронній мережі без цієї процедури не обійтися. Здебільшого нормалізація й кодування «зашиті» всередині алгоритмів, проте аналітик повинен розбиратися в їхніх різновидах, оскільки вибір схеми кодування й нормалізації істотно впливає на результат

Найпопулярнішою та найзручнішою бібліотекою для всіх вищезазначених процедур є *scikit-learn*, а саме модуль *sklearn.preprocessing.* Використання основних його класів подано в прикладах.

Отже, для нормалізації даних існують різні методи. Ось декілька з них.

Міні-Макс нормалізація. Цей метод перетворює значення ознак так, щоб вони перебували в заданому діапазоні (наприклад, від *a* до *b*, зазвичай від 0 до 1) (лістинг 3.7 (3.2; 3.3).

Формула:

$$x_{\text{norm}} = a + (b - a)\,\frac{x - x_{\min}}{x_{\max} - x_{\min}} \qquad (3.2)$$

Формула для *a = 0, b = 1:*

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \qquad (3.3)$$

```
                    # Лістинг 3.7
```
Приклад Міні-Макс нормалізації для sklearn.preprocessing

```
# X — масив shape (n_samples, n_features)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range=(0,1))
X_norm = scaler.fit_transform(X)
```

Стандартизація за допомогою стандартного відхилення (Z-score нормалізація). Цей метод перетворює дані так, щоб вони мали середнє значення 0 і стандартне відхилення 1 (σ = 1). Міні-Максна нормалізація оптимальна, коли значення ознаки *x* щільно заповнюють певний інтервал. Але такий підхід застосовний не завжди. Так, якщо в даних є порівняно рідкісні викиди, що набагато перевищують типовий розкид, саме вони визначать, згідно з попередньою формулою, масштаб нормалізації. Це призведе до того, що основна маса значень нормованої змінної *x'* зосередиться поблизу нуля. У цьому разі набагато надійніше орієнтуватися не на екстремальні значення, а на типові, тобто такі статистичні характеристики даних, як середнє і дисперсія. Наприклад, на всій множині *x* для деякої ознаки *V* обчислюють середнє значення та стандартне відхилення σ. Потім для кожного значення ознаки розраховується перетворення за формулою (3.4), лістинг 3.8:

$$x' = \frac{x - \text{mean}(x)}{\text{std}(x)} \tag{3.4}$$

де:

x – вибіркове середнє, s – вибіркове стандартне відхилення.

```
                    # Лістинг 3.8
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_std = scaler.fit_transform(X)
```

Нормалізація за сумою квадратів. Цей метод перетворює дані так, щоб сума квадратів значень ознак була рівна 1. Формула (3.5):

$$x' = \frac{x}{\sqrt{\sum_{i-1}^{n} x_i^2}} \tag{3.5}$$

Нормалізація за сумою квадратів (L_2-нормалізація, часто через *sklearn.preprocessing.Normalizer(norm='l2')* вирівнює кожен вектор-спостереження так, щоб його довжина (норма) дорівнювала 1.

```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer(norm='l2')
X_unit = normalizer.transform(X)
```

Логарифмічна нормалізація. Цей метод застосовується, коли дані мають значний діапазон, і їх потрібно змінити так, щоб вони стали більш нормально розподіленими. Формула (3.6):

$$x' = \log(x + c) \tag{3.6}$$

Значення *c* у формулі логарифмічної нормалізації використовується для уникнення взяття логарифму від нуля або негативного значення. Це допомагає уникнути виникнення помилок або нескінченності в обчисленнях, особливо коли діапазон вхідних даних містить нулі або від'ємні значення.

Зазвичай *c* обирають досить малим, наприклад 0.001 або 0.0001. Це значення обирається так, щоб воно було маленьким, щоб надто не впливати на фактичні значення (лістинг 3.8).

```
                    # Лістинг 3.9
```

Приклад логарифмічної нормалізація для sklearn

```
from sklearn.preprocessing import FunctionTransformer
# Створюємо трансформер для natural log
log_transformer = FunctionTransformer(func=lambda
x: np.log(x + 1e-6), validate=True)
X_log = log_transformer.transform(X)
```

Якщо після лог-трансформації потрібно ще привести дані до певного діапазону (наприклад, [0,1]), можна комбінувати *з MinMaxScaler* (лістинг 3.10)*:*

```
                    # Лістинг 3.10
from sklearn.preprocessing import
FunctionTransformer, MinMaxScaler
from sklearn.pipeline import make_pipeline
pipeline = make_pipeline(
    FunctionTransformer(lambda x: np.log(x + 1e-6),
validate=True),
    MinMaxScaler(feature_range=(0,1))
)
X_log_norm = pipeline.fit_transform(X)
```

Пояснення

Додаємо ε (наприклад, 1e-6 або 1), щоб уникнути проблем із нулем. Можна використовувати натуральний логарифм *(np.log)* або десятковий *(np.log10)*.

Для великих діапазонів значень логарифм «стискає» масштаб, зменшуючи вплив великих викидів. За потреби після логтрансформації застосовуйте додаткові скейлери *(MinMaxScaler, StandardScaler)*.

Робастна нормалізація (Robust Scaling)

Мета: зменшити вплив викидів через використання медіани та IQR. Формула (3.7):

$$x_{\text{robust}} = \frac{x - \text{median}(x)}{\text{IQR}(x)}, \quad \text{IQR} = Q_{75} - Q_{25} \tag{3.7}$$

```
# Лістинг 3.11
```

Приклад робастної нормалізації для sklearn

```
from sklearn.preprocessing import RobustScaler
scaler = RobustScaler()
X_rob = scaler.fit_transform(X)
```

Рекомендації щодо застосування

Min-Max: коли потрібно одразу отримати всі ознаки в єдиному діапазоні (наприклад, для нейромереж з активаціями [0,1][0,1][0,1]).

Z-score: коли важливо, щоб ознаки мали центр 0 і однаковий масштаб; потрібне для багатьох моделей (SVM, логістична регресія).

Robust: якщо в даних є викиди, які можуть «зламати» mean/σ, але не повинні надто впливати на масштабування.

Нормалізація за сумою квадратів

Текстові дані (TF-IDF, векторизація). Під час побудови векторних уявлень документів (bag-of-words, TF-IDF) рядки (документи) мають різну довжину і, відповідно, різну «вагу». L_2-нормалізація дає змогу порівнювати документи за кутом між векторами (косинусна схожість) без впливу абсолютної довжини тексту.

Кластеризація й пошук схожості

Якщо ви робите K-means або будь-який алгоритм, що базується на відстані (Euclidean, cosine), L_2-нормалізація гарантує, що всі

ознаки одного об’єкта складають вектор одиничної довжини. Це запобігає домінуванню «довгих» векторів над «короткими» в обчисленні відстаней.

Методи на основі dot-product (SVM, нейронні мережі)

У тих самих SVM (з ядром cosine) чи у шарах з активаціями, чутливими до величини входу, одиничні вектори допомагають стабілізувати навчання й уніфікувати масштаб різних зразків.

Компактне представлення

Коли важлива лише спрямованість вектора (взаємне співвідношення ознак), а не їхня абсолютна величина, L_2-нормалізація «знімає» вплив масштабу, лишаючи тільки «кут».

Приклади з практики

Рекомендаційні системи: уявлення користувача й товару як векторів у спільному просторі, нормалізовані вектори обчислюють cosine-схожість.

Комп’ютерний зір: дескриптори ключових точок (SIFT, SURF) після L_2-нормалізації стають інваріантними до освітленості.

Аналіз часових рядів: якщо щогодини Series-вектор нормалізувати, можна порівнювати форму сигналів незалежно від амплітуди.

Для навчання моделі також часто є корисним проведення *One-Hot кодування*. Потрібно взяти категоріальну змінну з *K* різними значеннями (містами, кольорами, класами тощо), створити *K* нових бінарних ознак, кожна з яких відповідає одній категорії. Для кожного спостереження треба змінити значення «1» у стовпчику тієї категорії, що відповідає його значенню, всі інші – «0» (лістинги 3.12; 3.13).

Приклад. Категоріальна ознака «Колір» зі значеннями {«червоний», «зелений», «синій»} перетворюється на три бінарні стовпці:

Колір	red	green	blue
зелений	0	1	0
синій	0	0	1
червоний	1	0	0

```
# Лістинг 3.12
```

Приклад One-Hot кодування для Pandas

```
import pandas as pd
df = pd.DataFrame({
    "Color": ["red", "green", "blue", "green", "red"]
})
df_ohe = pd.get_dummies(df, columns=["Color"])
print(df_ohe)
```

```
# Лістинг 3.13
```

Приклад One-Hot кодування для scikit-learn

```
from sklearn.preprocessing import OneHotEncoder
encoder = OneHotEncoder(sparse=False, dtype=int)
X_cat = [["red"], ["green"], ["blue"], ["green"],
["red"]]
X_ohe = encoder.fit_transform(X_cat)
print(encoder.get_feature_names_out(["Color"]))
print(X_ohe)
```

Питання для самоконтролю

1. Що таке центральна тенденція і які три основні міри ви знаєте для її опису? Поясніть поняття: середнє, медіана, мода.

2. Поясніть різницю між дисперсією та стандартним відхиленням. Чому стандартне відхилення часто є більш інтерпретованим показником розсіювання даних?

3. Що таке квантилі та квартилі? Як міжквартильний розмах (IQR) допомагає у виявленні викидів у даних?

4. Перелічіть щонайменше три поширені проблеми «брудних» даних, які потребують очищення, та запропонуйте методи для їхнього вирішення.

5. Навіщо потрібно розділяти вихідний набір даних на навчальну (train) та тестову (test) вибірки? Яка основна мета кожної з цих вибірок?

6. Що таке перенавчання (overfitting) моделі? Як використання валідаційної (validation) вибірки допомагає боротися з цією проблемою?

7. Навіщо потрібна нормалізація або стандартизація даних перед подачею в модель машинного навчання? Наведіть приклад алгоритму, чутливого до масштабу ознак.

8. У чому полягає різниця між нормалізацією (Min-Max Scaling) та стандартизацією (Z-score Standardization)?

9. Опишіть математичну формулу для стандартизації (Z-score). Якими будуть середнє значення та стандартне відхилення ознаки після застосування цієї процедури?

10. Чи потрібно застосовувати ту саму трансформацію (об’єкт Scaler, навчений на тренувальних даних) до тестової вибірки? Поясніть, чому.

3.3. Прикладні бібліотеки та хмарні платформи для МН

3.3.1. Бібліотеки Python

Важливою відмінністю МН є кількість вхідних параметрів (ознак), які модель здатна опрацювати. Для коректного прогнозу погоди в певній локації за теорією знадобиться ввести тисячі параметрів, які вплинуть на результат. Людина апріорі не може побудувати такий алгоритм, який буде використовуватися розумним способом. Для МН таких обмежень не існує, допоки вистачає потужності процесора й пам'яті.

Наведемо коротку характеристику основних фреймворків для МН.

Keras. Keras – це доволі міцна основа для старту. Якщо ви ознайомлені з Python і не займаєтесь високим рівнем досліджень, не розробляєте якусь особливу нейронну мережу, то Keras – це для вас. Основна увага зосереджена здебільшого на досягненні результатів, а не на нюансах моделі. Тож якщо розробляєте проєкт, пов'язаний, наприклад, з класифікацією зображень чи моделями послідовностей, почніть з Keras. Зможете отримати робочу модель досить швидко. Keras також інтегрований у TF (https://keras.io/) [21].

PyTorch. Порівняно з TF, PyTorch є більш інтуїтивно зрозумілим. Навіть якщо у вас немає досвіду машинного навчання, ви зрозумієте моделі PyTorch. У PyTorch немає такого інструменту візуалізації, як TensorBoard, але завжди можна використати бібліотеку на зразок matplotlib (https://pytorch.org) [22].

Caffe. Caffe працює добре, коли будують моделі глибокого навчання з даними про зображення. А коли йдеться про рекурентні нейронні мережі й мовні моделі, то Caffe відстає від інших фреймворків. Ключова перевага Caffe полягає в тому, що навіть якщо ви не володієте машинним навчанням, ви зможете будувати моделі глибокого навчання. Caffe використовується переважно для створення моделей глибокого навчання, для мобільних телефонів й інших обчислювальних платформ (https://caffe.berkeleyvision.org) [23].

Deeplearning4j. Deeplearning4j – це рай для програмістів Java. Він пропонує широку підтримку таких різних нейронних мереж, як CNN, RNN та LSTM. Фреймворк може опрацювати значну кількість даних і водночас не позначитися на швидкості (https://deeplearning4j.konduit.ai/) [24].

TensorFlow (TF). Відкрита та найпопулярніша бібліотека з глибокого машинного навчання від Google. TensorFlow призначена для досліджень і розроблення моделей. Google не просто має будь-які дані, а в них найбільш широко масштабований у світі комп'ютер, тому TF побудовано для масштабування. TF – це бібліотека, розроблена командою Google Brain для прискорення машинного навчання й досліджень нейронних мереж (https://www.tensorflow.org) [25].

Бібліотеку TF побудовано для роботи на декількох звичайних або графічних процесорах і навіть мобільних операційних системах, а також вона має кілька оболонок на різних мовах програмування: Python, C++, Java.

Нещодавно глибоке навчання почало перевершувати всі інші алгоритми машинного навчання під час отримання значної кількості даних, а отже, Google може використовувати глибокі нейронні мережі для покращення своїх послуг. Розроблено основу, яка називається TF, щоб дати змогу дослідникам і розробникам спільно працювати над моделлю ШІ. Тож розширена TF допомагає багатьом людям користуватися нею.

Вперше TF було оприлюднено наприкінці 2015 року, а перша стабільна версія з'явилася 2017 року (з відкритим кодом за ліцензією Apache Open Source). Отже, можна користуватися TF, змінювати та розповсюджувати модифіковану версію, отримуючи за це окрему плату, а нічого за це не платити Google. Назву TF вибрано тому, що вхідні дані приймаються як багатовимірний масив, відомий також як тензор. Можемо побудувати своєрідну блок-схему операцій (Graph), які потрібно виконати на конкретному вході. Вхідні дані надходять з одного кінця, а потім вони протікають через задану систему багатьох операцій і виходять з іншого кінця як вихід.

Фреймворк назвали *TensorFlow* тому, що тензор протікає в ньому через список операцій, а потім виходить з іншого кінця. TensorFlow добре працює із зображеннями й даними на основі послідовностей. Лабораторний практикум посібника базується на використанні саме функціоналу TF.

Фреймворк TensorFlow має спеціальний інструмент *TensorBoard* – інтерфейс, який використовується для візуалізації графа й інших інструментів для розуміння, налагодження й оптимізації моделі (рис. 3.10).



Рисунок 3.10 – Вкладки панелі TensorBoard

Джерело: https://www.guru99.com/tensorboard-tutorial.html

Основна ідея TensorBoard полягає в тому, що нейронна мережа може сприйматися як чорна скринька, а нам треба мати інструмент для перевірки того, що є всередині цієї скриньки. TensorBoard дає змогу почати глибоко досліджувати нейронну мережу. Його призначення – заглянути всередину моделі й побачити, як взаємодіють різні операції, як оновлюються ваги, змінюється значення функції втрат та інші важливі метрики. Об’єднавши ці дані, отримуємо потужний засіб для налагодження мережі й пошуку способів її вдосконалення.

3.3.2. Oracle Machine Learning

Oracle підтримує інтерфейси мов програмування для SQL, R і Python, а також безкодові інтерфейси користувача, як-от OML AutoML UI та Oracle Data Miner, а також керування моделлю REST і розгортання через служби OML.

Oracle Machine Learning Notebooks (OML Notebooks) базується на технології Apache Zeppelin, що дає змогу виконувати машинне навчання в Oracle Autonomous Database (Autonomous Data Warehouse (ADW), Autonomous Transactional Database (ATP) і Autonomous JSON Database (AJD). OML Notebooks допомагає користувачам досліджувати, візуалізувати та готувати дані, а також розробляти й документувати аналітичні методології (рис. 3.11). Важливим аспектом є можливість участі в цих програмах закладів освіти як інституціональних учасників.

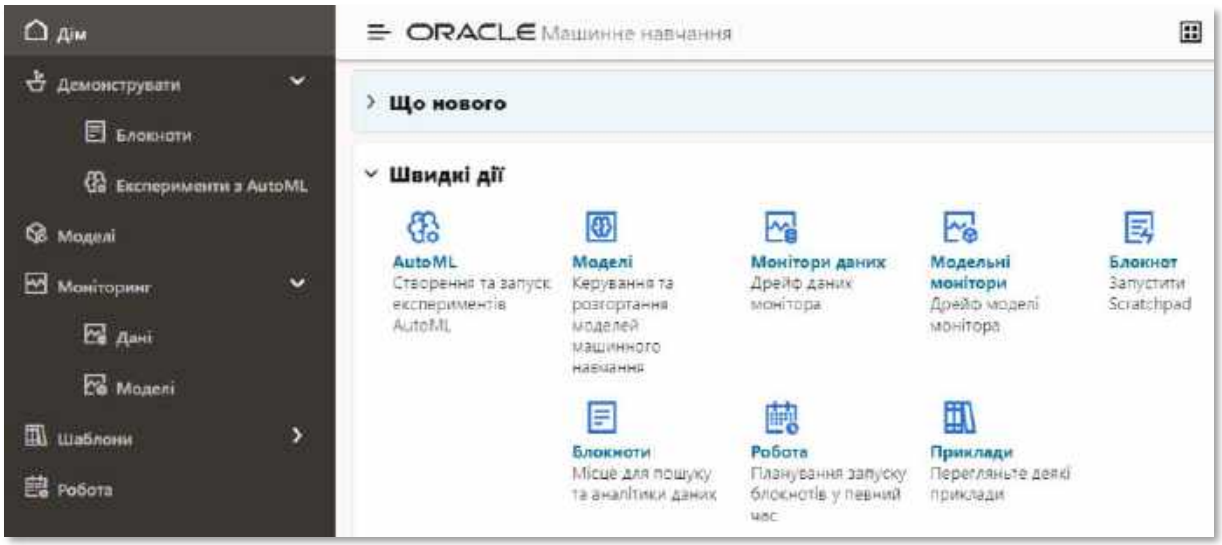


Рисунок 3.11 – Сторінка Oracle Cloud Machine Learning

Інтерфейс користувача AutoML (AutoML UI) – це інтерфейс машинного навчання Oracle, який забезпечує автоматичне машинне навчання без коду. Коли створюєте та запускаєте експеримент в інтерфейсі користувача AutoML, він автоматично виконує вибір алгоритму й функцій, а також налаштовує та вибирає модель, підвищуючи її продуктивність і точність. Бізнес-користувачі, які не мають досвіду роботи з даними, можуть використовувати AutoML UI для створення та розгортання моделей машинного навчання.

Сервіси Oracle Machine Learning Services (OML Services) розширюють функціональні можливості OML, щоб підтримувати розгортання та керування життєвим циклом моделі як для моделей OML у базі даних, так і для моделей машинного навчання у форматі Open Neural Networks Exchange (ONNX) сторонніх виробників через REST API. REST API для Oracle Machine Learning Services надає кінцеві точки REST API, розміщені в Oracle Autonomous Database. Ці кінцеві точки дають змогу зберігати моделі машинного навчання разом із метаданими та створювати оцінювальні кінцеві точки для моделі.

Oracle Machine Learning for Python (OML4Py) дає змогу запускати команди та сценарії Python для перетворення даних, а також для статистичного, машинного навчання й графічного аналізу даних, які зберігаються в службі Oracle Autonomous Database або доступні через неї за допомогою API Python. OML4Py – модуль Python, який дає нагоду користувачам Python маніпулювати даними в таблицях і представленнях бази даних за допомогою синтаксису Python. Функції та методи OML4Py прозоро перетворюють вибраний набір функцій Python у SQL для виконання в базі даних.



Рисунок 3.12 – Вивід результатів аналізу в Oracle Machine Learning for Python (OML4Py)

Користувачі OML4Py можуть використовувати автоматизоване машинне навчання (AutoML) для підвищення продуктивності користувачів і результатів машинного навчання за допомогою

автоматичного вибору алгоритмів і функцій, а також налаштування та вибору моделі. Користувачі можуть використовувати Embedded Python Execution для запуску визначених користувачем функцій Python у механізмах Python, створених середовищем автономної бази даних.

Oracle Machine Learning for R (OML4R) забезпечує орієнтоване на базу даних середовище для наскрізних аналітичних процесів у *R* із негайним розгортанням визначених користувачем функцій *R* у робочих середовищах. OML4R – це набір пакетів *R* і функцій Oracle Database, які дають змогу користувачеві *R* працювати з резидентними даними бази даних без використання SQL і запускати визначені користувачем функції *R*, які також називаються «сценаріями», в одній або кількох керованих базами даних Двигуни R. OML4R внесено до Oracle Database і Oracle Database Cloud Service.

Oracle Machine Learning for SQL (OML4SQL) надає доступ SQL до потужних алгоритмів машинного навчання в базі даних. Можна використовувати OML4SQL для створення та розгортання прогностичних і описових моделей машинного навчання, які можуть додати інтелектуальні можливості до додатків та інформаційних панелей. OML4SQL входить до *Oracle Database, Oracle Database Cloud Service і Oracle Autonomous Database.*

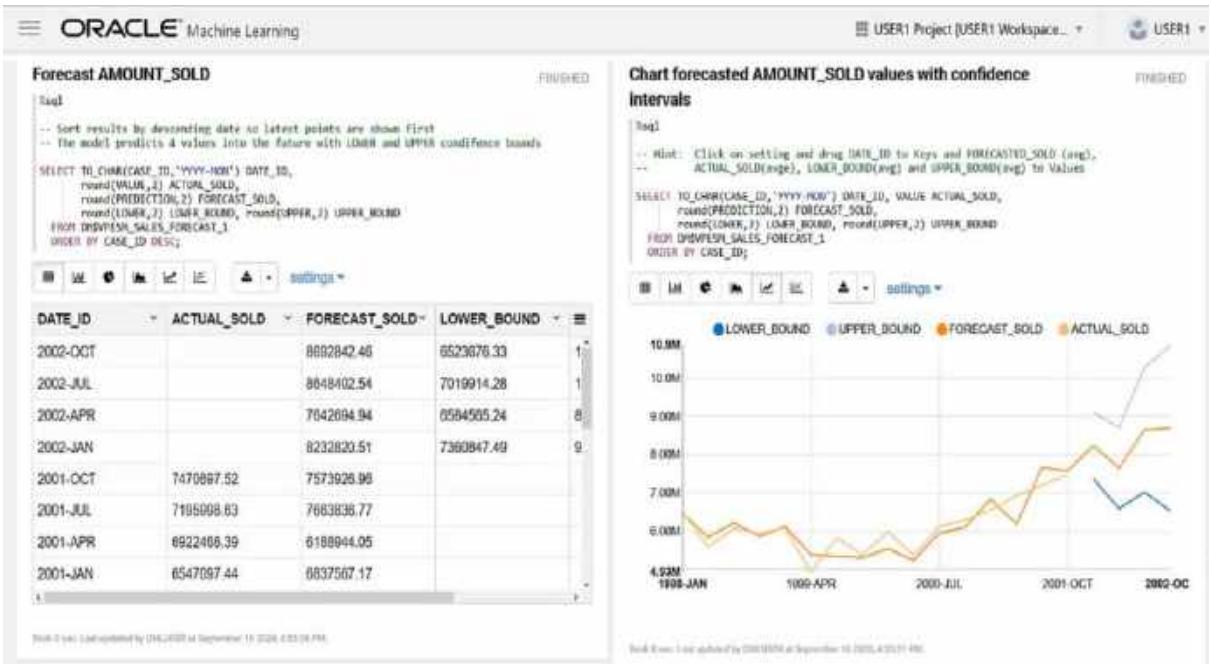


Рисунок 3.13 – Прогнозування часових рядів
в Oracle Machine Learning for SQL (OML4SQL)

Oracle Data Miner (ODMr) – це розширення Oracle SQL Developer, графічний інтерфейс користувача для виявлення прихованих закономірностей, взаємозв'язків і аналізу даних. ODMr надає редактор робочих процесів із функцією перетягування, щоб визначати та фіксувати кроки, які користувачі виконують для дослідження й підготовки даних і застосування технології машинного навчання.

Oracle Machine Learning for Spark (OML4Spark) надає масштабовані алгоритми машинного навчання через *R* API для середовищ Spark і Hadoop, щоб досліджувати й готувати дані, створювати та розгортати моделі машинного навчання. OML4Spark є компонентом Oracle Big Data Connectors і входить до Oracle Big Data Service.

Oracle Spatial Artifical Intelligence. Поточний випуск Oracle Spatial AI (2024 р.) надає алгоритми геопросторового машинного навчання для аналізу та моделювання геопросторових векторних даних і подій, пов'язаних із розташуванням. Він надає методи геопросторового машинного навчання, наскрізний робочий процес і відповідні API. Просторовий штучний інтелект інтегрований з Oracle Machine Learning (OML) і розгортається за допомогою OML4Py в безсерверній хмарній службі *Oracle Autonomous Database*. Це означає, що можна отримати доступ до Spatial AI через інтерфейси та служби OML на своєму примірнику автономної бази даних без сервера.

Таблиця 3.4 – Функції інтерфейсу користувача Oracle Machine Learning для підтримки мов Python та R

Підтримка сторонніх бібліотек Python та R	Сторонні бібліотеки для Python та R доступні в Oracle Machine Learning Notebooks. Інтерфейс користувача Oracle Machine Learning надає інтерпретатор Conda для встановлення сторонніх бібліотек Python та R у сеансі блокнота. Conda – це система керування пакетами та середовищем з відкритим кодом, яка дає змогу використовувати середовища, що містять сторонні бібліотеки Python та R

Продовження таблиці 3.4

Адміністративні ролі	Користувачі з *OML_SYS_ADMIN* роллю можуть встановлювати сторонні бібліотеки Python та R і завантажувати їх до сховища об'єктів для постійного зберігання. Користувач з *OML_SYS_ADMIN* роллю є адміністратором, також відомим як admin. Користувачі з *OML_DEVELOPER* роллю можуть використовувати інтерпретатор Conda для завантаження та активації сторонніх бібліотек за допомогою середовища Conda, наданого адміністратором. Користувач з OML_DEVELOPER роллю є звичайним користувачем Oracle Machine Learning
Підтримка R	Oracle Machine Learning for R підтримується в Oracle Machine Learning Notebooks. Використовуючи Oracle Machine Learning for R, можна виконувати дослідження даних і моделювання машинного навчання. OML4R доступний через Oracle Machine Learning Notebooks на Oracle Autonomous Database Serverless, включно із службами Autonomous Data Warehouse, Autonomous Transaction Processing та Oracle Autonomous JSON Database

3.3.3. Microsoft Azure Machine Learning

Azure Machine Learning – хмарний сервіс для прискорення життєвого циклу проєкту машинного навчання (ML) і керування ним. Фахівці з ML, спеціалісти з обробки даних та інженери можуть використовувати його у своїх повсякденних робочих процесах для навчання та розгортання моделей і керування операціями машинного навчання (MLOps).

Azure Machine Learning містить інструменти, які допоможуть створювати програми Generative AI на основі великих мовних моделей (LLM). Рішення охоплює каталог моделей, оперативний потік і набір інструментів для оптимізації циклу розробки додатків ШІ.

Каталог моделей у *Azure Machine Learning Studio* є центром для виявлення та використання широкого спектру моделей, які

дають змогу створювати програми Generative AI. У каталозі моделей представлені сотні моделей від таких постачальників моделей, як служба Azure OpenAI, Mistral, Meta, Cohere, NVIDIA, Hugging Face, зокрема моделі, навчені Microsoft.

Потік підказок *Azure Machine Learning* – це інструмент розробки, призначений для оптимізації всього циклу розробки програм штучного інтелекту на базі великих мовних моделей (LLM). Prompt flow надає комплексне рішення, яке спрощує процес створення прототипів, експериментування, ітерації та розгортання програм ШІ.

Фахівці з даних можуть використовувати моделі в машинному навчанні Azure, які вони створили в поширених фреймворках Python, наприклад PyTorch, TensorFlow, scikit-learn, XGBoost, LightGBM.

3.3.4. Amazon Web Servises Machine Learning

Станом на квітень 2025 року *Amazon Web Servises ML* підтримує такі методи:

Класичний ML – класифікація, регресія, кластеризація, пошук аномалій (вбудовані алгоритми *XGBoost, Linear Learner, Random Cut Forest, K-means, PCA*).

Глибоке навчання – CNN / ResNet / ViT для CV, LSTM / Transformer для NLP та TS forecasting; розподілене multi-node GPU / Trainium 2 training.

Reinforcement Learning – SageMaker RL (SimApp + Gym) для робототехніки, оптимізації.

AutoML – SageMaker Autopilot (табличні дані) та Bedrock Agents для генеративних сценаріїв.

Generative AI – текст, зображення, відео й код через Bedrock (сімейства Titan, Nova) та Amazon Q (LLM-помічник для девопс і аналітики), Amazon Web Services, Inc.US Press Center.

Збір / зберігання: S3, Redshift, RDS, OpenSearch.

Каталогізація / ETL: Glue Crawler + DataBrew / Lake Formation.

Анотація (якщо CV / NLP): Ground Truth (людські / роботизовані задачі).

EDA & Feature Eng: Data Wrangler (ноутбуки або інтерактивний UI).

Навчання: Training Jobs, Tuner, Debugger; фреймворки TensorFlow 2.x, PyTorch 2.x, MXNet – чи повністю код free Autopilot.

Оцінювання та реєстрація: Clarify, Pipelines, Model Registry (версифікація).

Продакшн-розгортання: Managed Endpoints, Lambda, EKS, Greengrass / Edge Manager.

Моніторинг: Model Monitor, CloudWatch, EventBridge; «drift alerts».

MLOps & DevTools

CI / CD; CodeCommit: CodeBuild, SageMaker Pipelines або Step Functions.

IaC: CloudFormation, CDK (python / TS).

Observability: CloudWatch Logs / Traces, AWS X-Ray.

ШІ-помічник: Amazon Q Developer для генерування коду, юніт-тестів і рецензій.

Новий функціонал, який з'явився у 2024–2025 рр.

Next-generation Amazon SageMaker – об'єднана Studio з Lakehouse-інтеграцією та підсистемою Data & AI Governance, доступна вже у Європі (Paris) та APAC (Mumbai) Amazon Web Services, Inc.Amazon Web Services, Inc.

Amazon Bedrock вийшов у GA із бібліотекою Foundation Models (Titan, Nova Micro / Lite / Pro) і модулем Guardrails для керування ризиками генеративного AI Amazon Web Services, Inc.The Verge.

AI – вбудований помічник-кодер для AWS Console та VS Code (бету представлено восени 2024).

Нові спеціалізації *AWS Certified Machine Learning – Specialty (2025 blueprint)* охоплюють Bedrock і Trainium 2 Amazon Web Services, Inc.

На рис. 3.14 зображено архітектуру безсерверного рішення на базі AWS для автоматичної обробки та аналізу зображень. Процес починається, коли фото завантажується до сховища Amazon S3. Ця подія створює повідомлення в черзі Amazon SQS, яке зі свого боку запускає функцію AWS Lambda.

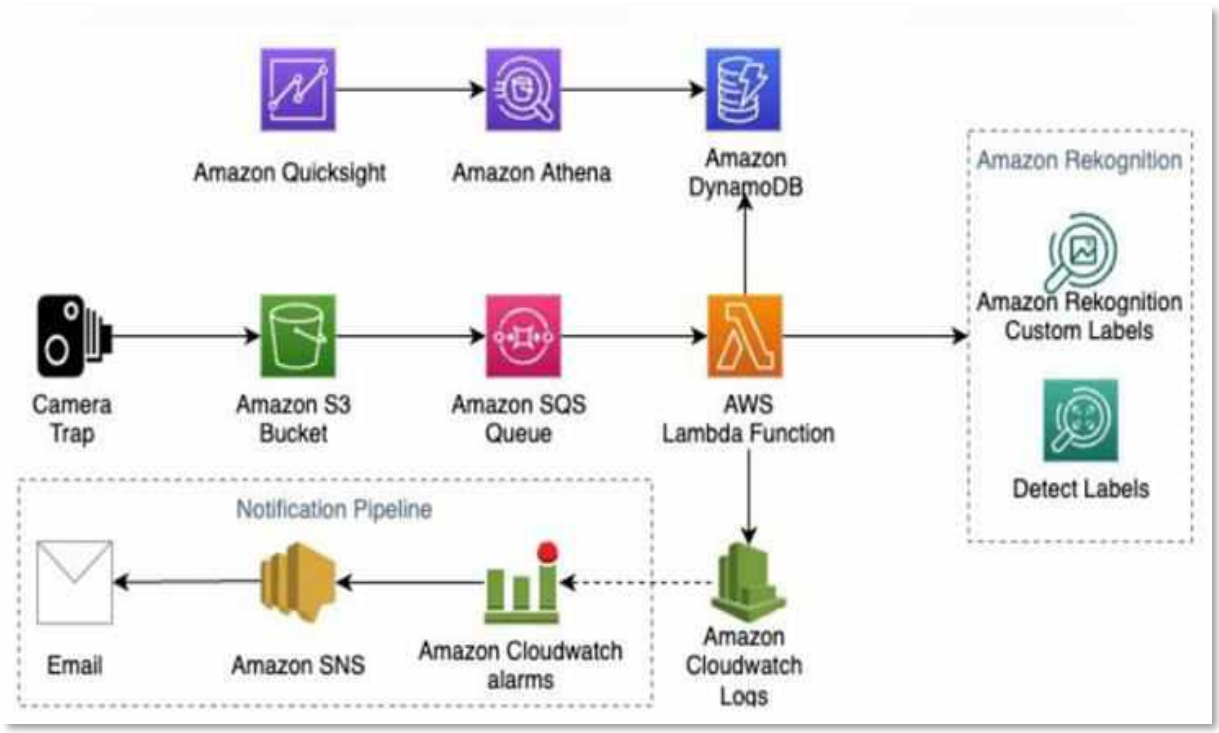


Рисунок 3.14 – Приклад архітектури рішення AWS

Функція Lambda виступає центральним компонентом, що оркеструє подальші дії: вона надсилає зображення до сервісу Amazon Rekognition для розпізнавання об’єктів (використовуючи як стандартні, так і кастомні мітки), зберігає результати аналізу в базу даних Amazon DynamoDB та записує логи своєї роботи в Amazon CloudWatch Logs. Паралельно існують два додаткові потоки: аналітичний, де дані з DynamoDB можна запитувати через Amazon Athena та візуалізувати в Amazon Quicksight, і система сповіщень, де тригери Amazon CloudWatch Alarms, налаштовані на певні події в логах, активують сервіс Amazon SNS для надсилання повідомлення на електронну пошту.

Таблиця 3.5 – Ключові сервіси AWS ML

Категорія	Основне призначення	Ключові сервіси / нові можливості
1	2	3
Джерела та сховище даних	Надійне зберігання та управління великими обсягами даних	Amazon S3, Redshift, RDS, OpenSearch, DynamoDB

Продовження таблиці 3.5

1	2	3
Підготовка та інтеграція	Екстракт-трансформ-завантаження (ETL), каталогізація метаданих, Lake House	AWS Glue, Lake Formation, EMR, DataZone
Платформа ML	Повний цикл ML: від збірки даних до продакшн-розгортання	Amazon SageMaker (next-gen Studio, Data Wrangler, Ground Truth, Pipelines, Model Registry, Clarify, Edge Manager) Amazon Web Services, Inc.Amazon Web Services, Inc.
AI-сервіси (готові моделі)	Швидке підключення CV / NLP / генеративних функцій через API	Amazon Bedrock (моделі Titan та Nova, Guardrails) Amazon Web Services, Inc.Amazon Web Services, Inc.; Rekognition, Textract, Comprehend, Transcribe, Translate, Polly, Lex, Kendra, Personalize, Forecast, Fraud Detector, Q
Інфраструктура й прискорювачі	Оптимізоване навчання та інференс	AWS Trainium 2, Inferentia 2, GPU EC2
Розгортання & моніторинг	Безпечне масштабування моделей, відстеження якості	SageMaker Endpoints & Model Monitor, Lambda, Fargate, API Gateway, CloudWatch

3.3.5. Google Colaboratory

Google Colaboratory (Coalab) – хмарне середовище Jupyter-ноутбуків, яке надає браузерний доступ до Python-експериментів без попередньої інсталяції ПЗ. Середовище орієнтоване на навчання, дослідження та швидкі ML-прототипи (табл. 3.6).

Google Colab залишається одним із найпростіших способів отримати GPU / TPU-потужності та повноцінне Python-середовище за хвилини. Free-версія чудово підходить для навчання й невеликих проєктів, а Pro / Enterprise-плани забезпечують довші сесії, новіші GPU, інтеграцію з Gemini LLM і керовані засоби MLOps.

Таблиця 3.6 – Групи користувачів Google Colaboratory

Група	Що дає користувачеві
Апаратні прискорювачі	Безкоштовний доступ до GPU (NVIDIA T4 / L4 24 GB VRAM) та TPU v5e; у платних планах – A100 80 GB і більші квоти X (formerly Twitter) X (formerly Twitter)
Попереднє середовище	Понад 300 бібліотек (TensorFlow 2.x, PyTorch 2.x, scikit-learn, pandas, OpenCV тощо) встановлені «з коробки»
Інтеграція з Google екосистемою	Зберігання ноутбуків у Google Drive, підключення BigQuery, Sheets API, GitHub-імпорт
Спільна робота	Шеринг посиланням (режими view / comment / edit), коментарі, історія версій
Gemini-асистент	У платформах *Colab Enterprise* та *Colab Pro* доступний LLM-помічник «Gemini in Colab» для автодоповнення коду й виправлення помилок (GA жовтень 2024) Google Cloud
Task Scheduler	Планові прогони ноутбуків, Terraform-керування середовищами (GA 2024, Terraform підтримка – лютий 2025)

Приклади доступних практичних *Google Colaboratory*, які надають розуміння й навички застосування нейромереж глибокого навчання [26]:

1. *Навчання класифікатора зображень*: створення моделі Keras на основі даних попередньо натренованого класифікатора для розпізнавання зображень.

2. *Класифікація тексту:* класифікації текстових відгуків про фільми на позитивні та негативні на основі датасету IMDB.

3. *Відповіді на запитання про багатомовний універсальний кодер речень:* відповіді на запитання з набору даних SQuAD за допомогою моделі машинного навчання.

4. *Інтерполяція відео:* прогнозування подій у відео між першим і останнім кадрами.

5. *Перенесення стилю:* перенесення стилів між зображеннями за допомогою глибинного навчання. У прикладі надано результати дослідження структури мережі довільної нейронної художньої стилізації в реальному часі (Голназ Гіасі, Хонглак Лі, Манджунат Кудлур, Вінсент Дюмулен, Джонатан Шленс, Матеріали Британської конференції з машинного зору (BMVC), 2017).

Приклад. Результат виконання скрипта практичної роботи «*Перенесення стилю»* з використанням модуля *TF Hub* (рис. 3.15):

```
    # Load TF Hub module.
    hub_handle =
'https://tfhub.dev/google/magenta/arbitrary-image-
stylization-v1-256/2'
    hub_module = hub.load(hub_handle)
```



а) б) в)

Рисунок 3.15 – Результат виконання скрипта
з використанням модуля TF Hub

а) оригінальне зображення; б) зображення, вибране для копіювання стилю (картина Мунка «Крик»); в) зображення з результатом перенесення стилю

3.3.6. Matlab Machine Learning

Ключові методи, якими Matlab ML «закриває» типові задачі:

Класичний ML: дерева, SVM, k-NN, ансамблі, автофічі (featureSelection, automl Live Task).

Глибоке навчання: CNN (ResNet, YOLOX, SAM), NN / LSTM / Transformer (зокрема, transformerEncoderLayer), Vision Transformer та BERT-приклади.

AutoML & Bayesian Optimization: з GUI або коду (bayesopt).

Reinforcement Learning: DQN, PPO, SAC; готові середовища Simulink чи Robot Gym.

Generative AI: текстові LLM (`llm`), diffusion-моделі для зображень, модулі «Gen AI for Robotics» (trajectory planning) MATLAB Central Blogs.

Робочий процес «від даних до продакшену»

Дані: datastore, tall або live-конектор до BigQuery/S3.

EDA & Прет-процесинг: Live Tasks, normalize, bagOfWords, movmean, signalSpectrogram.

Навчання: скрипти чи Learner-apps; у R2024b можна конвертувати *shallow net* із *fitcnet / fitrnet* у *dlnetwork* для подальшого донавчання MATLAB Central Blogs.

Оцінка: confusionchart, rocmetrics, Explainable AI (lime, shapley).

Розгортання:

MATLAB Compiler / MATLAB Production Server (REST-сервіси);

GPU Coder / HDL Coder – вбудовані системи;

Simulink Real-Time / ROS для робототехніки;

Операції (MLOps): Experiment Manager → автоматичне журналювання, порівняння та *Model Comparison Report*; інтеграція з Git, MLflow.

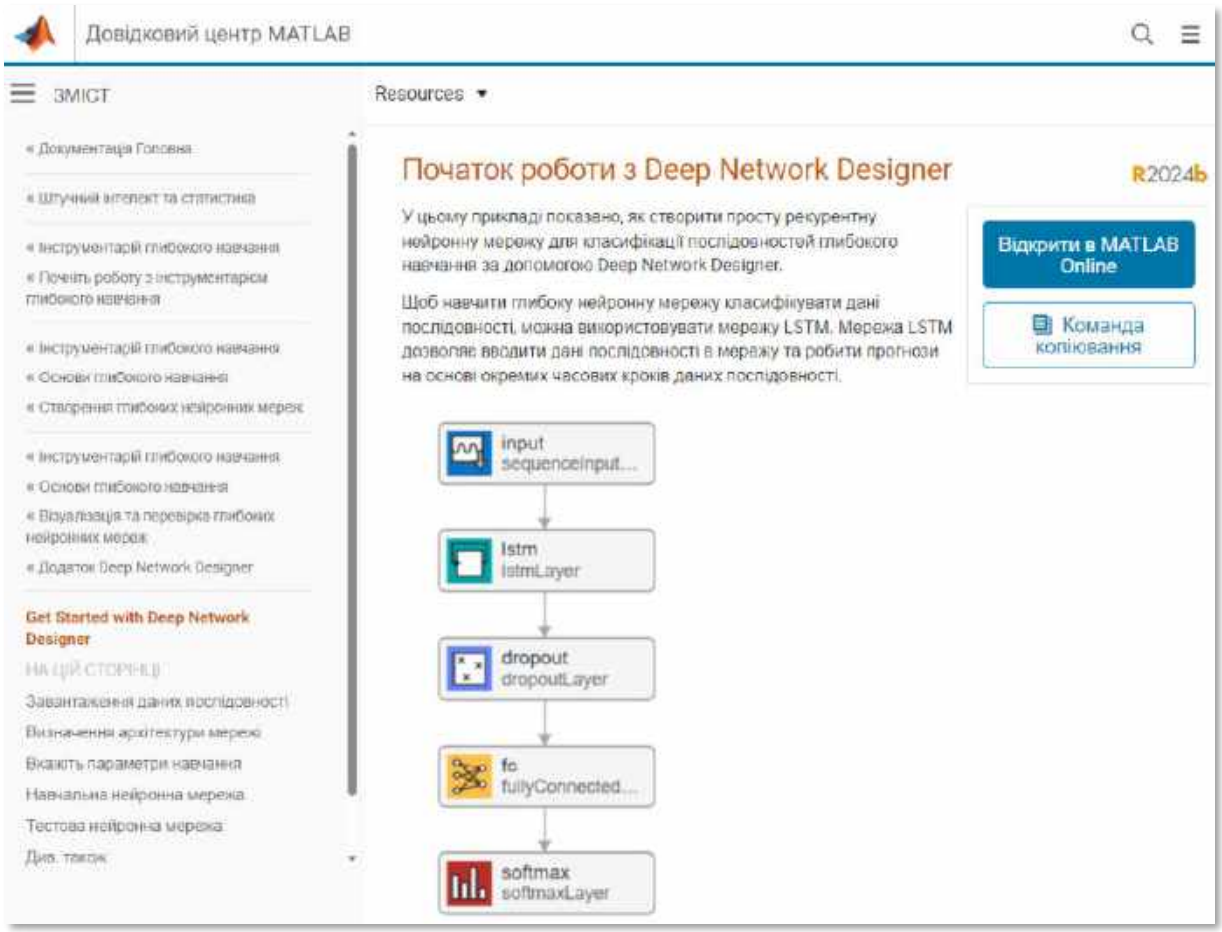


Рисунок 3.16 – Інтерфейс MATLAB ML

Чим MATLAB ML відрізняється від Python-стека?

Low-code Apps: дають змогу навчатись без коду та експортувати готові скрипти (рис. 3.16).

Єдина екосистема: сигнал-, зображення-, текст-, керування- та системне моделювання в одному середовищі.

Сильна інтеграція з Simulink для апаратно-орієнтованих систем.

Кодогенерація на GPU, FPGA та MCU без ручного портингу.

Прямий обмін із Python (*ру*.-інтерфейс, підтримка conda-env).

MATLAB сьогодні – це не лише «класичне» ML-середовище, а повноцінна платформа для глибокого, підкріплювального та генеративного навчання із розгортанням на CPU, GPU, FPGA і в роботизованих системах.

Корисні посилання:

Artificial Intelligence з MATLAB (головна сторінка). URL: MathWorks - Maker of MATLAB and Simulink;

Блог MathWorks «Apply AI with New R2024b Examples». URL: MATLAB Central Blogs;

Прев'ю Generative AI + Robotics (лют. 2025). URL: MATLAB Central Blogs.

Для навчання *matlabacademy.mathworks.com* надає такі ресурси (таблиця 3.7). Після пробного 30-денного періоду залишається 20 годин безкоштовного щомісячного доступу до хмарної платформи.

Таблиця 3.7 – Ресурси навчання й викладання в хмарному середовищі Matlab

	Що дає	Умови / обмеження
1	2	3
MATLAB Onramp	Інтерактивний 2-годинний курс «з нуля» у браузері	Завжди безкоштовно після створення MathWorks Account
Інші Onramp-курси	*Machine Learning, Deep Learning, Reinforcement Learning, Simulink* Onramp тощо (1–3 год кожен)	Також повністю безкоштовні й запускаються в хмарній сесії MATLAB Online

Продовження таблиці 3.7

1	2	3
MATLAB Online (оновлена IDE у браузері)	Не потребує установки; зручна для коротких лабораторних чи практикуму	30-денна trial-ліцензія або доступ через кампус-ліцензію. Для Onramp-курсів та MATLAB Grader – безстроково
MATLAB Mobile	Доступ до MATLAB Online зі смартфона, зчитування сенсорів (GPS, акселерометр) у хмару	Безкоштовно; потребує MATLAB Drive для зберігання даних
File Exchange / Community	35 000 + безкоштовних скриптів, функцій, моделей від спільноти	Відкритий доступ; реєстрація MathWorks Account

Питання для самоконтролю

1. Назвіть три основні бібліотеки Python для машинного навчання та коротко опишіть їхнє основне призначення (для загальних алгоритмів МН і глибокого навчання).

2. Поясніть основну різницю між *NumPy* та *Pandas*. В яких випадках ви обрали б одну бібліотеку замість іншої для підготовки даних?

3. Яка ключова перевага використання Oracle Machine Learning безпосередньо в базі даних Oracle? Як це впливає на переміщення даних і безпеку?

4. Опишіть загальний принцип роботи *OML AutoML UI*. Які кроки він автоматизує для користувача, що не є експертом у машинному навчанні?

5. Що таке *Azure Machine Learning Studio*? Поясніть різницю між Designer (конструктором) та *Notebooks* як двома середовищами для розробки моделей.

6. Назвіть та опишіть призначення двох ключових ресурсів в *Azure ML: Compute Instance* та *Compute Cluster*.

7. Що таке *Amazon SageMaker*? Опишіть його роль як комплексної платформи для побудови, тренування та розгортання моделей машинного навчання.

8. Поясніть різницю між керованими сервісами AWS вищого рівня (наприклад, Amazon Rekognition для аналізу зображень) і платформою SageMaker. В якому випадку що краще використовувати?

9. Які дві основні переваги надає *Google Colab* для студентів і дослідників, що працюють над завданнями машинного навчання?

10. Що таке *Classification Learner App* та *Regression Learner App у Matlab?* Як вони спрощують процес вибору та тренування моделей?

11. Порівняйте підхід до машинного навчання в Matlab та за допомогою бібліотек Python (наприклад, *Scikit-learn*). Назвіть одну перевагу та один недолік підходу Matlab.

ТЕМА 4
МАШИННЕ НАВЧАННЯ В КЛАСИЧНИХ МЕТОДАХ АНАЛІЗУ

4.1. Регресійно-кореляційний аналіз

4.1.1. Основи регресійно-кореляційного аналізу

У розділі 3 під час розгляду методів МН віднесли регресійно-кореляційний аналіз до методів *класичного навчання з учителем* (додаток А). Основна мета *регресійно-кореляційного аналізу* полягає в тому, щоб визначити, чи існує статистично значимий зв'язок між змінними, а також визначити характер цього зв'язку.

Регресійно-кореляційний аналіз – це процес побудови й аналізу математичної моделі як рівняння регресії (рівняння кореляційного зв'язку), що виражає залежність результативної ознаки від однієї або кількох ознак-факторів і дає оцінку міри щільності зв'язку.

То що ж виступає «учителем» у цьому методі з погляду машинного навчання? У регресійно-кореляційному методі «учителем» із погляду машинного навчання є *цільова ознака* (англ. *target, response variable*), тобто саме ті спостереження залежної змінної, на яких навчаємо метод і які намагаємося передбачити або апроксимувати.

У задачі регресії маємо набір ознак $X = \{x_1, x_2, \ldots, x_p\}$ і одну цільову змінну y. Значення y для кожного спостереження є «вчителем», який дає зворотний зв'язок під час навчання: модель підбирає такі параметри, щоб звести похибку між своїм прогнозом y і справжнім y до мінімуму.

У МН існує чималий клас задач, де потрібно встановити залежність між ознаками (атрибутами, показниками), які описують досліджуваний процес або об'єкт предметної сфери. Для цього будуються різні моделі, в яких дані ознаки існують як змінні.

Якщо модель буде коректно відображати залежність між вхідними та вихідними змінними, то за допомогою такої моделі можна буде передбачати значення вихідної змінної за заданим значенням вхідних.

Показники точності, оцінки перевірки гіпотез наявності регресійно-кореляційного аналізу часто зустрічаються під час оцінювання якості моделей інших методів.

Основні поняття та ідеї регресійно-кореляційного аналізу охоплюють:

– *кореляцію.* Кореляція визначає ступінь взаємозв'язку між двома змінними. Коефіцієнт кореляції може бути позитивним (якщо зміна в одній змінній супроводжується відповідною зміною в іншій) або негативним (якщо зміни в обох змінних рухаються в протилежних напрямах). Зазвичай використовуються такі коефіцієнти кореляції, як Пірсона, Спірмена або Кендалла;

– *лінійну регресію.* Це статистичний метод, що дає змогу моделювати зв'язок між двома змінними, припускаючи, що цей зв'язок можна виразити за допомогою прямої лінії. У лінійній регресії одна змінна використовується для прогнозування значень іншої змінної. Модель лінійної регресії шукає таку пряму лінію, яка найкраще відображає зв'язок між залежною та незалежною змінними;

– *лінія регресії.* Першим кроком аналізу є побудова лінії регресії так, щоб розсіювання точок уздовж неї було мінімальним. Тобто сума квадратів відхилень спостережуваних значень від оцінених за допомогою цієї лінійної залежності повинна бути мінімальною (4.1).

$$\sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \to \min , \qquad (4.1)$$

де:

n – число спостережень;

$\hat{y}_i$ – оцінка вихідного значення *i*-го спостереження, отримана за допомогою моделі;

y_i – реальні значення показника.

Цей метод відомий як метод найменших квадратів (МНК), а лінія, побудована з його допомогою, називається *лінією регресії.* Рівняння, що описує лінію регресії, має назву *рівняння регресії* (4.2).

$$\hat{y}_i = b_0 + b_1 \times x, \tag{4.2}$$

де:

$\hat{y}_i$ – оцінка значення вихідної змінної;

b_0 (intercept) – точка перетину лінії з віссю *y*, яку ще називають вільним членом;

b_0 – значення, яке приймає вихідна змінна *y*, при $x = 0$.

Коефіцієнт b_1(slope) визначає нахил лінії щодо осі *x* (іноді його називають кутовим коефіцієнтом). Тобто b_1 – це величина, на яку змінюється значення вихідної змінної *y* під час зміни вхідної змінної *x* на одиницю.

Лінія регресії – це пряма найкращого наближення для набору пар значень вхідних і вихідної змінної (x, y), що обирається так, щоб сума квадратів відстаней від точок (x_i, y_i) до цієї прямої, виміряних вертикально (тобто вздовж осі *y*), була мінімальна.

Дисперсія – це математична характеристика, яка показує, наскільки значення випадкової величини або елементів вибірки відхиляються від свого середнього значення (середнього арифметичного). Інакше кажучи, дисперсія вимірює розсіяння даних навколо середнього. Чим більше дисперсія, тим ширше розкидані значення навколо середнього. Для вибірки (на практиці): якщо маємо *n* спостережень $x_1, x_2, \ldots, x_n$, то *вибіркова дисперсія* обчислюється так (4.3):

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2 \tag{4.3}$$

де:

$\bar{x}$ – вибіркове середнє;

n – 1 використовується для незміщеної оцінки дисперсії.

Середньоквадратичне відхилення. Середньоквадратичне відхилення (СКВ, або англ. *Standard Deviation*) позначається зазвичай σ (сігма) – це величина, яка показує, наскільки в середньому елементи вибірки або значення випадкової величини відхиляються від свого середнього значення. Це корінь квадратний із дисперсії. Інакше кажучи, СКВ – це «усереднена» міра розкиду даних навколо середнього, виражена в тих самих одиницях, що й самі дані. Формула вибіркового СКВ (4.4.):

$$s = \sqrt{\frac{1}{n-1}\sum_{i-1}^{n}(x_i - \bar{x})^2} \qquad (4.4)$$

де:

s – вибіркове СКВ;

$\bar{x}$ – вибіркове середнє;

n – 1 використовується для незміщеної оцінки дисперсії.

Коефіцієнт детермінації (R^2) є мірою, яка вказує на те, наскільки добре лінійна регресійна модель підходить до даних. R-квадрат визначає відсоток дисперсії залежної змінної, який пояснюється незалежними змінними в моделі. Чим більше значення R^2, тим краще модель пояснює зміну залежної змінної (4.5).

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \qquad (4.5)$$

де:

- $SS_{\text{res}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ — сума квадратів залишків (помилок моделі),
- $SS_{\text{tot}} = \sum_{i=1}^{n}(y_i - \bar{y})^2$ — загальна сума квадратів (розкид реальних значень відносно середнього),
- y_i — фактичне значення,
- $\hat{y}_i$ — прогнозоване значення моделі,
- $\bar{y}$ — середнє значення залежної змінної.

Тестування гіпотез. Регресійно-кореляційний аналіз також може охоплювати тестування гіпотез про наявність зв’язку між змінними, значущість цього зв’язку, а також значимість коефіцієнтів регресії. Регресійно-кореляційний аналіз досить важливий для різних галузей науки й досліджень, включно із економікою,

психологією, соціологією, медициною та ін. Він допомагає виявляти та розуміти зв'язки між змінними, що може бути надто корисним під час ухвалення рішень і прогнозування.

Показники точності моделі. Якщо уявити, що y_i – спостережене значення, $\hat{y}_i$– прогноз моделі, n – кількість спостережень, то основні показники оцінювання якості й точності статистичних моделей можна викласти в таблиці 4.1.

Таблиця 4.1 – Основні показники точності статистичних моделей

Скорочення	Назва	Формула	Інтерпретація
MSE	Mean Squared Error Середньоквадратична помилка	$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	Середній квадрат відхилень; сильно збільшується щодо великих помилок
RMSE	Root MSE (Кв. корінь із MSE)	$\mathrm{RMSE} = \sqrt{\mathrm{MSE}}$	У тих самих одиницях, що й цільова ознака; «середня» помилка
MAE	Mean Absolute Error Середня абсолютна помилка	$\mathrm{MAE} = \frac{1}{n}\sum_{i-1}^{n}\lvert y_i - \hat{y}_i \rvert$	Показує, *наскільки* в середньому (у тих самих одиницях, що й сама змінна) прогнози моделі відхиляються від фактичних значень
MAPE	Mean Absolute Percentage Error Середня відсоткова помилка	$\mathrm{MAPE} = \frac{100}{n}\sum_{i-1}^{n}\left\lvert \frac{y_i - \hat{y}_i}{y_i} \right\rvert \%$	Переводить похибку у *відсотки* від фактичного значення
R^2	Coefficient of Determination Коефіцієнт детермінації	$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$	Частка дисперсії, пояснена моделлю (0 – 1, іноді <0)
R^2_{adj}	Adjusted R2 Скоригований R^2	$R^2_{adj} = 1 - (1 - R^2)\frac{n-1}{n-p-1}$	Коригує R^2 за число параметрів; корисний для порівняння моделей з різною *p*

4.1.2. Приклад регресійно-кореляційного аналізу

У цьому прикладі реалізуємо модель множинної лінійної регресії на Python за допомогою бібліотеки *Scikit Learn*. Для вивчання цієї моделі будемо використовувати набір даних про житло з метою прогнозування цін на житло на основі змінних у регіоні, в якому розташований будинок (*Блокнот перебуває в репозторії курсу на Мудлі)*. Набір даних *(USA_Housing.csv)*, який будемо досліджувати, складається із 5 000 рядків та з таких стовпців ознак (табл. 4.2):

Таблиця 4.2 – Показники таблиці цін на житло

Avg. Area Income:	Середній дохід мешканців регіону
Avg. Area House Age:	Середній вік будинків у регіоні
Avg. Area Number of Rooms:	Середня кількість кімнат на будинок у регіоні
Avg. Area Number of Bedrooms:	Середня кількість спалень на будинок у регіоні
Area Population:	Населення регіону, в якому розташований будинок
Price:	Ціна продажу будинку
Address:	Адреса будинку

Імпортування бібліотек

Перш ніж завантажувати дані, потрібно імпортувати всі відповідні бібліотеки, які потрібні для проєкту. Основною бібліотекою в цьому прикладі є *Scikit Learn*, із якої з відповідних підмодулів завантажуються функції (лістинг 4.1):

train_test_split – розбиває вибірку на навчальну й тестову частини випадково;

LinearRegression – модель звичайної лінійної регресії (метод найменших квадратів);

mean_absolute_error – середня абсолютна похибка між прогнозом і фактом;

mean_squared_error – середня квадратична похибка для регресійних моделей;

r2_score – коефіцієнт детермінації R^2, характеризує частку поясненої дисперсії.

Якщо бібліотеки невстановлені, їх можна інсталювати за допомогою команди !pip install.

```
                    # Листинг 4.1
    Імпорт актуальних бібліотек
    import numpy as np  # бібліотека для числових
обчислень
    import pandas as pd # завантаження, очищення й
аналіз даних
    import matplotlib.pyplot as plt # візуалізація,
графіки та діаграми
    import seaborn as sns # надбудова Matplotlib для
візуалізації
    from sklearn.model_selection import
train_test_split
    from sklearn.linear_model import LinearRegression
    from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
```

Імпорт даних

Щоб завантажити дані як фреймворк даних *pandas,* можемо використати метод *read_csv()* і передати місцезнаходження набору даних. У нашому випадку використовуємо URL-адресу до набору даних у репозиторії на GitHub (лістинг 4.2):

```
                    # Листинг 4.2
    Завантаження даних (датасету) і створення
    # датафрейму pandas
    df =
pd.read_csv("https://raw.githubusercontent.com/siglim
umuni/Datasets/master/USA_Housing.csv")
    # Прогляд перших п'яти рядків датафрейму df
    df.head()
```

Важливо пам'ятати! Якщо файл даних завантажений на локальний комп'ютер, потрібно вказати шлях до нього. Для завантаження файлів даних *.csv,. txt., json* у середовище Google Colab із локального середовища потрібно використовувати код (лістинг 4.3):

```
# Лістниг 4.3
```

Завантаження даних у Colab з локальної станції

```
# Під час виконання цього коду з'явиться запрошення вибрати файл локально.
from google.colab import files
uploaded = files.upload()
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

Результати виведення інформації методом *df.head()* (рис. 4.1).

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05

Рисунок 4.1 – Результати виведення інформації методом *df.head()*

У результатах для компактності виведення не показана колонка *Address*. Далі вона буде видалена й не буде використовуватися в аналізі, оскільки має нечисловий формат.

Дослідження даних

Як зазначалось, *розвідувальний аналіз даних (РАД)* є важливим кроком у процесах машинного навчання. Це спосіб зрозуміти дані, з якими працюємо, їхні характеристики, потреба в їхньому очищенні та взаємозв'язки між різними змінними. Використовуючи метод *info()*, можемо відобразити кількість рядків, стовпців та їхні типи даних, а також наявність у даних нульових значень (лістинг 4.4).

```
                    # Лістинг 4.4
Дослідження структури датафрейму df
df.info()
 <class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   Avg. Area Income              5000 non-null   float64
 1   Avg. Area House Age           5000 non-null   float64
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population               5000 non-null   float64
 5   Price                         5000 non-null   float64
 6   Address                       5000 non-null   object
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

Із результату видно, що датафрейм складається з 5 000 рядків і 7 стовпців, 6 з яких мають типи даних з плаваючою комою, а останній є рядковим типом даних. Крім того, наші дані не мають нульових значень.

Метод *describe()* швидкий і простий спосіб розпочати дослідження *фрейму даних*. Він повертає таблицю зі зведеною статистикою для всіх числових стовпців даних. У прикладі обгортаємо метод у функцію *round ()* і передаємо їй як аргумент 1, щоб повернути одне десяткове число для всіх значень.

Для виведення більш детальної статистичної інформації про датасет скористаємось методом *describe()* (рис. 4.2).

```
# Сумарна статистика числових стовпців датафрейму
round(df.describe(),1)
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.0	5000.0	5000.0	5000.0	5000.0	5000.0
mean	68583.1	6.0	7.0	4.0	36163.5	1232072.7
std	10658.0	1.0	1.0	1.2	9925.7	353117.6
min	17796.6	2.6	3.2	2.0	172.6	15938.7
25%	61480.6	5.3	6.3	3.1	29403.9	997577.1
50%	68804.3	6.0	7.0	4.0	36199.4	1232669.4
75%	75783.3	6.7	7.7	4.5	42861.3	1471210.2
max	107701.7	9.5	10.8	6.5	69621.7	2469065.6

Рисунок 4.2 – Детальна інформація про датасет, отримана методом *describe()*

Для виведення розширеної статистики щодо кожної числової колонки датасету можна виконати код (лістинг 4.5):

```
#Лістинг 4.5
Вибираємо тільки числові колонки
num = df.select_dtypes(include="number")
#1. Стандартні рядки describe()
base = num.describe()
#3. Додаємо метрики й оформлюємо їх РЯДКАМИ
extra = pd.DataFrame({
    "Ген. дисперсія":     num.var(ddof=0), # генеральна дисперсія
    "Виб. дисперсія ": num.var(ddof=1),    # вибіркова дисперсія
    "СКВ ген.":    num.std(ddof=0),        # СКВ генеральної сукупності
    "СКВ виб. ": num.std(ddof=1)           # СКВ вибіркове
}).T
#2. Склеюємо вертикально й округлюємо
full_table = pd.concat([base, extra]).round(1)
#3. Показуємо всю ширину таблиці
pd.set_option("display.expand_frame_repr", False)
print(full_table)
```

У результаті отримуємо таблицю із розширеною статистикою щодо кожного числового стовпця датафрейму. Для компактності на рис. 4.3 показані не всі стовпці.

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms
count	5000.0	5000.0	5000.0
mean	68583.1	6.0	7.0
std	10658.0	1.0	1.0
min	17796.6	2.6	3.2
25%	61480.6	5.3	6.3
50%	68804.3	6.0	7.0
75%	75783.3	6.7	7.7
max	107701.7	9.5	10.8
Ген. дисперсія	113570058.2	1.0	1.0
Виб. дисперсія	113592776.7	1.0	1.0
СКВ ген.	10656.9	1.0	1.0
СКВ виб.	10658.0	1.0	1.0

Рисунок 4.3 – Розширена статистика
щодо кожного числового стовпця датафрейму

Перевірка закону розподілу параметрів (змінних) датасету

Далі бажано перевірити закон розподілу параметрів (змінних) датасету (нормальний, рівномірний, логнормальний тощо) за допомогою гістограм. Можемо створити їх за допомогою методу *histplot()* із *Seaborn*, а також методу *subplots()* для створення декількох графіків. (Пояснення до коду лістингу 4.6 дано в блокноті до лабораторної роботи):

```
# Лістинг 4.6
```

```
    Створюємо об’єкт списку з іменами всіх числових
стовпців
    numeric_cols = df.columns[:-1]
    # Створюємо об’єкт subplots з 2 рядками та 3
стовпцями
    fig, axes =
plt.subplots(nrows=2,ncols=3,figsize=[12,6])
    # Перебираємо піддіаграми та будуємо гістограму
для кожного стовпця
    for col,ax in zip(numeric_cols,axes.ravel()):
      sns.histplot(df[col],ax=ax).set(title=col,ylabe
l="",xlabel="")
    # Відображаємо гістограми
    plt.tight_layout()
```

Результати виконання коду лістингу 4.6 (рис. 4.4).

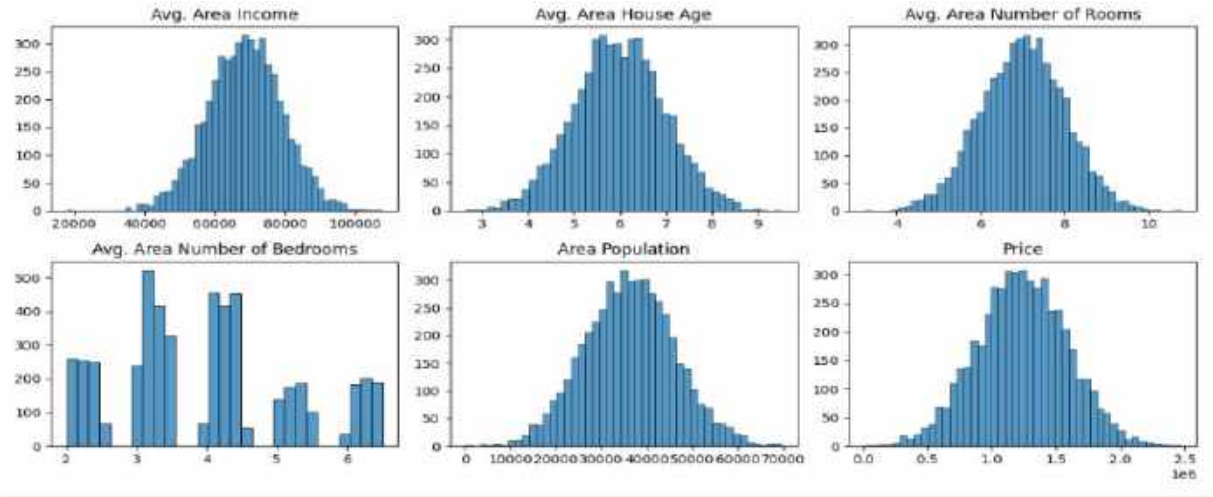


Рисунок 4.4 – Результати виконання коду лістингу

Усі змінні, крім середнього значення кількості спальних кімнат (*Avg. Area Number of Bedrooms)*, мають нормальний розподіл і більшість точок даних приблизні до середнього значення та медіани.

Як зазначалось, бочкоподібні графіки також є інформативним методом дослідження даних. Як видно із графіків (рис. 4.5), дані мають значення, які виходять за межі нормального розподілу, і їх можна вважати викидами. Проте, як видно із гістограм розподілів, дані загалом згруповані навколо середнього та явних відхилень від нормального розподілу немає.

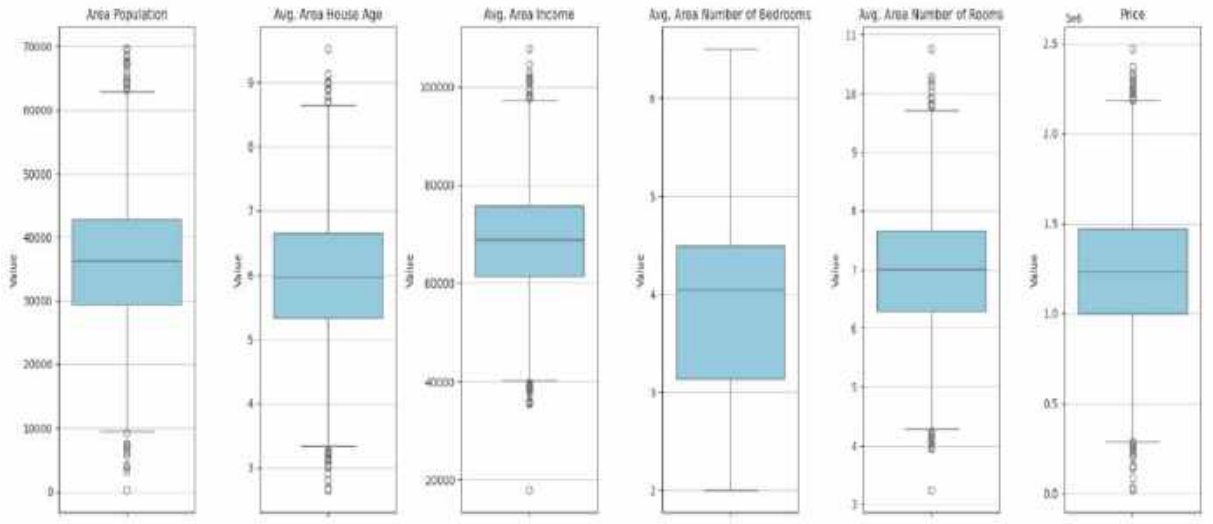


Рисунок 4.5 – Бочкоподібні графіки змінних датасету

Графіки отримано за допомогою бібліотеки *seaborn*. (Приклад коду наведений у блокноті до лабораторної роботи). Отже, бочкоподібні графіки дають змогу виявити викиди даних, які виходять за межі нормального розподілу, і є ефективним інструментом для виявлення аномалій та специфічних особливостей у даних.

Дослідження взаємокореляції між показниками

Наступним кроком бажано дослідити, як різні змінні пов'язані між собою, тобто наскільки вони взаємокорелюють. Для початку можемо створити кореляційну матрицю за допомогою методу *corr()*, а потім передати її в метод *Seaborn heatmap()* для створення так званої «теплової карти».

```
# Листинг 4.7
```

Створення матриці кореляції для всіх числових змінних

```
corr_matrix = round(df.corr(),2)

# Створюємо теплову карту та передаємо матрицю як аргумент даних
plt.figure(figsize=[12,6])
sns.heatmap(data = corr_matrix, cmap="coolwarm",annot=True)
plt.show()
```

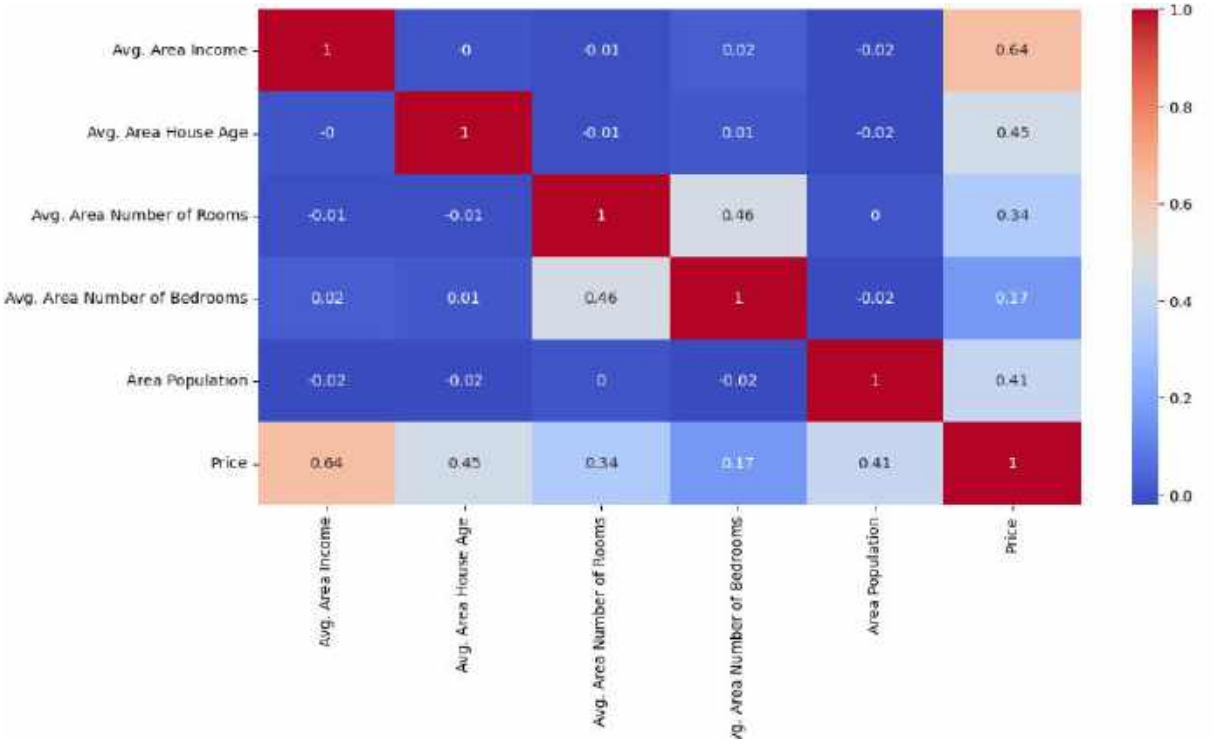


Рисунок 4.6 – Теплова карта взаємокореляції між показниками датасету

З теплової карти (рис. 4.6) бачимо, що існує певна позитивна кореляція, хоча різного ступеня між ціною та іншими змінними. Існує позитивна помірна кореляція між ціною та середнім доходом на душу населення в регіоні; позитивна слабка кореляція між ціною та віком будинку в цьому регіоні; позитивна слабка кореляція між ціною та всіма іншими змінними. Також інформативним є графічні зображення *корелограм* розсіювання, які можна отримати за допомогою бібліотеки *seaborn*, на яких можна також побачити незначні наявні кореляції між згаданими показниками (рис. 4.7).

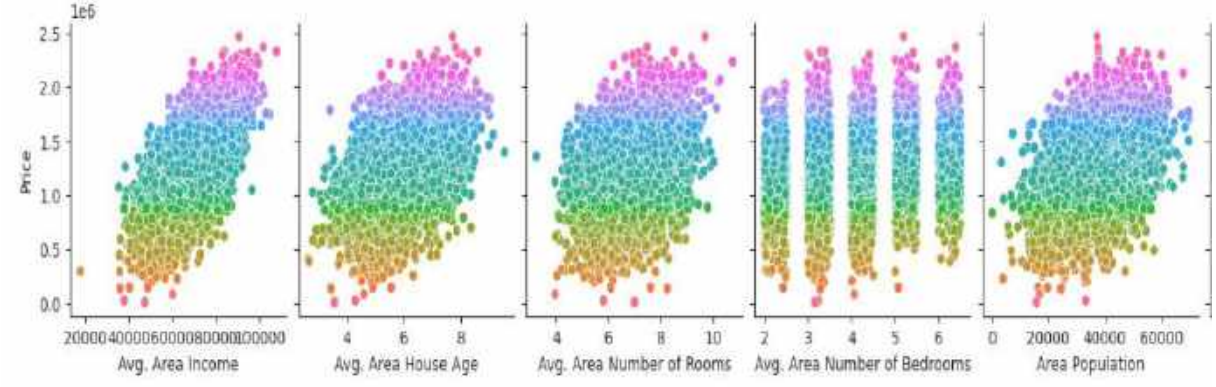


Рисунок 4.7 – Графічне зображення корелограм розсіювання

Побудова моделі

Перш ніж будувати модель, потрібно розділити дані на два масиви – *X* та *y*. *X* міститиме незалежні змінні, які будемо використовувати для прогнозування залежної або цільової змінної, представленої масивом *y*. Оскільки стовпець «Адреса» нечисловий, він не використовується в цьому регресійному аналізі (лістинг 4.8).

```
# Листниг 4.8
```

```
Створюємо підмножину фрейму даних, що містить незалежні змінні
    X = df[['Avg. Area Income', 'Avg. Area House Age',
'Avg. Area Number of Rooms',
           'Avg. Area Number of Bedrooms', 'Area
Population']]
    #Створюємо підмножину фрейму даних, що містить
залежну або цільову змінну  y
    y = df["Price"]
```

Далі нам потрібно знову розділити дані, але цього разу на *навчальну* та *тесто*ву множини. *Навчальний набір* слугуватиме навчальними даними для моделі, тоді як *тестовий набір* використовуватиметься для оцінювання моделі після того, як вона буде навчена. Можемо зробити це за допомогою функції *Scikit-Learn train_test_split()*, яку вже імпортували. Встановимо *test_size*, що дорівнює 0.33, яке означає, що 33 % даних будуть тестовими, а решта 67 % – навчальними. Можна встановити значення *random_state*, яке дорівнює 123, щоб отримати такий самий випадковий розподіл, як у прикладі (лістинг 4.9).

```
                    #Лістинг 4.9
    Розділюємо наші дані на навчальну та тестову множини
    X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.33, random_state=123)
```

Навчання моделі

Розділивши дані на навчальні та тестові набори, можемо почати навчання моделі. Нам потрібно створити об’єкт лінійної регресії, а потім подати на нього навчальні дані (лістинг 4.10).

```
                    #Лістинг 4.10
    Навчання моделі
    # Завантажуємо клас LinearRegression (лінійна
регресія)
    from sklearn.linear_model import LinearRegression
    # Створення об’єкта лінійної регресії
    model = LinearRegression()
    # Подання навчальних даних на модель (навчання
моделі)
    model.fit(X_train, y_train)
```

Пояснення рядків лістингу 4.10:

from sklearn.linear_model import LinearRegression: імпортуємо клас *LinearRegression* з модуля *linear_model*, що є частиною бібліотеки *scikit-learn* (часто скорочується до *sklearn)* у програмах Python.

model = LinearRegression(): цей рядок створює екземпляр (об’єкт) моделі лінійної регресії та зберігає його в змінній *model.*

model.fit(X_train, y_train): цей рядок «навчає» створену модель. Метод *.fit()* приймає два аргументи: навчальні ознаки *(X_train)* та відповідні їм правильні значення *(y_train),* на основі яких модель знаходить найкращі коефіцієнти для лінійного рівняння. Після виконання цих двох рядків ваша змінна ***model*** буде містити навчену модель, готову до створення прогнозів.

Далі можна вивести коефіцієнти регресії. Можемо отримати список, викликавши *coef_* в нашій моделі. А ще краще, створимо фрейм даних, щоб переглянути цю інформацію наочніше.

```
                    # Лістинг 4.11
    Створюємо фрейм даних з коефіцієнтами для кожної
    # змінної
    coefficients = pd.DataFrame(linear_model.coef_,
X.columns, columns=['Coeff'])
     # Для компактності виводимо перші п’ять рядків
    coefficients.head()
    # Значення постійного члену регресії
    print(linear_model.intercept_)

    -2630610.09
```

Отримуємо такі коефіцієнти моделі:

Avg. Area Income	x_1	21.573980
Avg. Area House Age	x_2	165392.852
Avg. Area Number of Rooms	x_3	118869.8975
Avg. Area Number of Bedrooms	x_4	3361.027931
Area Population	x_5	15.250554

Вільний член (постійний коефіцієнт *intercept)* у моделі дорівнює –2630610.09. У лінійній регресії *Вільний член* визначається як значення цільової або залежної змінної, якщо всі незалежні змінні дорівнюють 0 (економічного змісту зазвичай немає, хоча є і винятки).

Маючи значення постійно члена (*intercept*) –2630610.09, можемо записати формулу моделі множинної регресії:

$$y = -2630610.09 + 21.57 \times 1 + 165392.85 \times 2 + 118869.9 \times 3 + 3361.03 \times 4 + 15.25 \times 5$$

У лінійній регресії коефіцієнти показують, наскільки збільшиться цільова змінна, в нашому випадку ціна, на кожну одиницю збільшення незалежної змінної, що розглядається. Отже, із вищезазначеного видно збільшення середнього доходу *Avg. Area Income* (x_1) на одну одиницю (на 1$) за умови незмінності всіх інших змінних, що призведе до зростання ціни приблизно на 21,6$.

4.1.3. Оцінювання якості регресійної моделі

Для оцінювання моделі виконаємо деякі прогнози на основі тестового набору. Можемо зробити це за допомогою методу *predict()* в моделі й передати в нього тестовий набір даних (лістинг 4.12).

```
                    # Лістинг 4.12
    Робимо прогнози, використовуючи тестові дані
    predictions = linear_model.predict(X_test)
    print(predictions)
    # Створюємо діаграми розсіювання прогнозованих та
фактичних значень
    plt.figure(figsize=[8,6])
    plt.scatter(y_test, predictions)
    plt.title("Прогнозування ціни будинку")
    plt.ylabel("Прогнозна ціна")
    plt.xlabel("Фактична ціна")
    plt.show()
```

Результат виконання лістингу лістингу 4.12 показано на рис. 4.8.

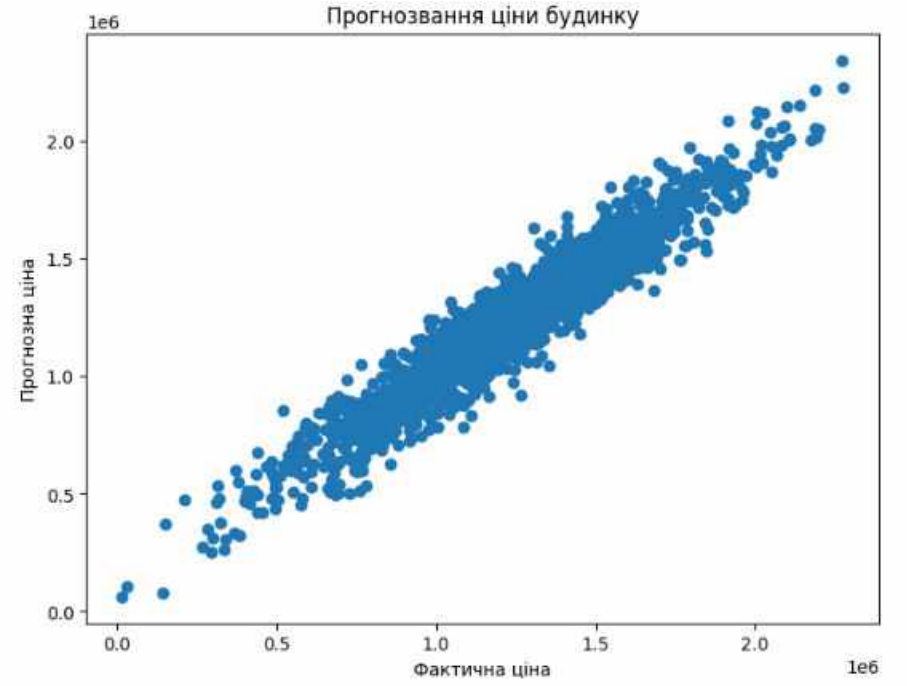


Рисунок 4.8 – Результати прогнозування ціни будинку

З графіка видно, що наші прогнози досить близькі до фактичних значень. Можемо рухатись далі, побудувавши гістограму залишків. У регресії залишки – це різниця між фактичними та прогнозованими значеннями (лістинг 4.13, рис. 4.9).

```
                    # Лістниг 4.13
    Побудова гістограми залишків
    plt.figure(figsize=[8,6])
    sns.histplot(y_test-
predictions).set(title="Distribution of Residuals")
    plt.show()
```

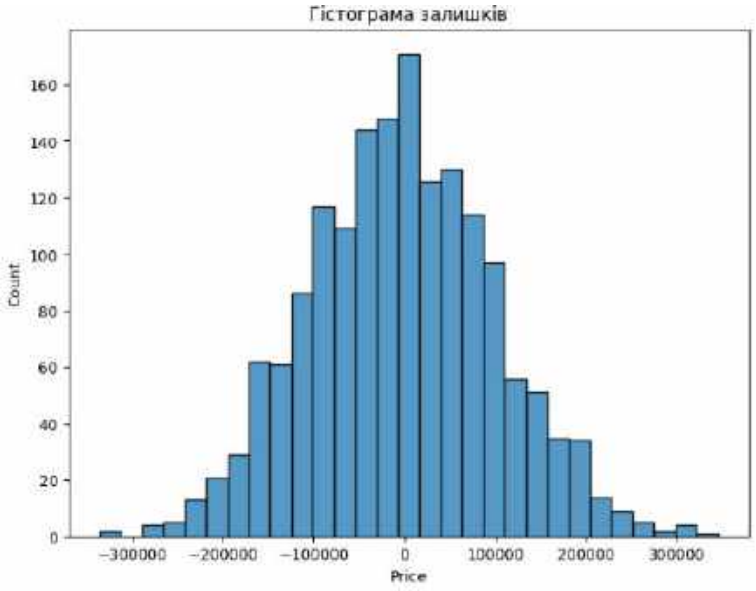


Рисунок 4.9 – Гістограма залишків моделі

Залишки (рис. 4.9) розподілені нормально, що є гарним способом визначити, наскільки модель відповідає даним. Останнім завданням буде оцінити модель за допомогою певних метрик. Для цього потрібно використовувати розглянуті в розділі показники *MAE, MSE, RMSE, R-квадрат Adjusted R-Squared.*:

– середня абсолютна похибка Mean Absolute Error (MAE);

– середня квадратична похибка Mean Squared Error (MSE);

– коренева середня квадратична похибка RootMeanSquaredError (RMSE);

– скоригований R-квадрат Adjusted R-Squared.

Нарешті, скоригований R-квадрат вимірює частку дисперсії цільової змінної, яка пояснюється незалежними змінними.

Оскільки вже імпортували функції для обчислення цих метрик зі *Scikit-Learn*, можемо відразу їх викликати та використати (лістинг 4.14):

```
#Лістинг 4.14
```

Обчислення метрик моделі

```
    print(f"The Mean Absolute Error is
{round(mean_absolute_error(y_test,predictions))}")
    print(f"The Mean Squared Error is
{round(mean_squared_error(y_test,predictions))}")
    print(f"The Root Mean Squared Error is
{round(np.sqrt(mean_squared_error(y_test,predictions)))}")
    print(f"The Adjusted R-Squared is {round(1 - (1 -
r2_score(y_test, predictions)) * (len(y) - 1) /
(len(y) - X.shape[1] - 1),2)}")
```

Отримані результати

```
The Mean Absolute Error is       (MAE)  81801
The Mean Squared Error is        (MSE)  10509340063
The Root Mean Squared Error is   (RMSE) 02515
The Adjusted R-Squared is                0.92
```

Завжди важливо мати відносні оцінки точності моделі, які дають змогу більш точно зрозуміти можливі похибки. Як оцінити якість моделі, якщо *MAE = 81 801* щодо середнього значення y = 1 232 073? Це мало чи багато? Це можна оцінити лише відносно тих чисел, які прогнозуємо чи аналізуємо. Тому для кращого розуміння бажано завжди обчислювати *відносні метрики* (лістинг 4.15).

```
                    # Лістинг 4.15
    Обчислення відносних помилок MAE, RMSE, MAPE
import numpy as np
    y_true  = y                          # ваш фактичний
вектор
    y_pred  = linear_model.predict(X)
    mean_y  = y_true.mean()
    rel_mae = 100 * 81801 / mean_y
    rel_rmse= 100 * 102515 / mean_y
    mape    = 100 * np.mean(np.abs((y_true - y_pred) /
y_true))
    print(f"Середнє значення y = {mean_y:,.0f}")
    print(f"Відносна MAE ≈ {rel_mae:.1f}%")
    print(f"Відносна RMSE ≈ {rel_rmse:.1f}%")
    print(f"MAPE ≈ {mape:.1f}%")

    Середнє значення y = 1,232,073
    Відносна MAE ≈ 6.6%
    Відносна RMSE ≈ 8.3%
    MAPE ≈ 7.6%
```

Як видно із результатів, відносні похибки < 10 % є непоганим результатом для цієї моделі. Також у регресійно-кореляційному аналізі завжди важливо оцінювати й розуміти довірчі інтервали моделі із заданими рівнями значущості.

Рівень значущості (α), рівень довіри (1 – α) та довірчі інтервали

У статистичних дослідженнях є поняття помилок першого та другого роду (розглядаються далі). Помилка першого (*Type I error*) означає *відхилення* нульової гіпотези про відсутність кореляції, коли насправді всі $b_i = 0$.

Якщо зазначаємо про *рівень значущості* $\alpha = 0.05$, проговорюємо «різниця є статистично значущою» саме за критерієм $\alpha = 0.05$ і погоджуємося, що в *5 %* випадків це рішення може бути хибним.

Рівень довіри $(1 - \alpha) = 95\ \%$ показує ймовірність того, що побудований довірчий інтервал містить справжнє значення оцінюваного параметра (наприклад, середнє μ). Тут під «справжнім» (іноді його називають «істинним» або «популяційним») мається

на увазі *реальне, але невідоме* значення параметра в генеральній сукупності, яке оцінюємо. Довірчий інтервал для середнього з відомого СКВ (σ) обчислюється за формулою (4.6):

$$\bar{x} \pm z_{1-\alpha/2} \frac{\sigma}{\sqrt{n}}, \tag{4.6}$$

де:

$\overline{x}$ – вибіркове середнє;

σ – відоме СКВ, стандартне відхилення генеральної сукупності;

n – обсяг вибірки;

$z_{1-\alpha/2}$ = (для 95 %) ≈ 1.96 – квантиль стандартного нормального розподілу.

Нижче надано результати розрахунку довірчих інтервалів для *рівня значущості* α = 0.05 і, відповідно, рівня довіри (1 – α) = 95 %.

У результатах виведення: *Coeff* – коефіцієнти моделі; *SE* – стандартна похибка; *CI_lower_95%* – нижня межа довірчого інтервалу; *CI_upper_95%* – верхня межа довірчого інтервалу.

Коефіцієнти моделі, стандартна похибка та довірчі інтервали для *α = 0.05*.

	Coeff	**SE**	**CI_lower_95%**	**CI_upper_95%**
Avg. Area Income	21.5740	0.1343	21.3106	21.8374
Avg. Area House Age	165392.8532	1443.8029	162562.3654	168223.3410
Avg. Area Number of Rooms	118869.8973	1605.5941	115722.2279	122017.5668
Avg. Area Number of Bedrooms	3361.0279	1309.0248	794.7646	5927.2913
Area Population	15.2506	0.1443	14.9677	15.5334

Оцінка значимості регресійної моделі

Для багатьох практичних застосувань знайти рівняння регресії та побудувати відповідну лінійну залежність недостатньо. Щоб вирішити задачу, потрібно побудувати регресійну модель. Відмінність *простої лінійної регресії* від *регресійної моделі* полягає в тому, що рівняння регресії здебільшого будується на основі вибіркових даних. Водночас із множини спостережень, що називається *генеральною сукупністю*, виділяється деяка вибірка, на

основі властивостей якої передбачається робити висновки про властивості всієї сукупності. Для цієї вибірки й будується рівняння регресії. Потім проводиться узагальнення отриманої лінійної залежності на всю генеральну сукупність.

Тобто *проста лінійна регресія* будується на основі вибірки спостережень, кожне з яких містить відповідність x → y, а *регресійна модель* передбачає, що виявлена лінійна залежність має місце для всіх можливих пар x → y, які містяться в генеральній сукупності. Однак під час побудови регресійної моделі виникає ряд проблем, що виходять за рамки побудови рівняння регресії.

По-перше, потрібно визначити, чи справедливо припущення про те, що *лінійна залежність*, виявлена для вибіркових спостережень, *буде істинною* для всієї сукупності.

По-друге, ситуація ускладнюється тим, що регресія взагалі не є детермінованою завданням, оскільки для реальних даних залежність зміни вихідної змінної від зміни вхідних носить випадковий характер. Отже, коефіцієнти регресії можуть розглядатися як статистика, що описує розподіл вихідної змінної.

Проста лінійна регресійна модель задається так. Нехай є вибірка даних, що містить n парних спостережень, у кожному з яких значенням незалежної змінної x_i відповідає значення залежної змінної y_i, пов'язаних через лінійне відношення (4.7):

$$y = \beta_0 + \beta_1 \times x + \varepsilon, \qquad (4.7)$$

де: ε – залишки моделі.

Для рівняння лінійної регресії в нашому прикладі коефіцієнт детермінації $R^2 = 0{,}92$, що вказує на досить суттєвий зв'язок залежної та незалежної змінних.

У багатьох задачах інтерес представляє не оцінка значення параметра моделі β_1 для генеральної сукупності, а деяке твердження про нього. У цьому разі завдання зводиться до перевірки гіпотези, за результатами якої гіпотеза буде або підтверджена, або відхилена

Але якщо для деякої вибірки визначимо коефіцієнт детермінації $R^2 = 0{,}2$, то це значення показує, що відповідна модель навряд чи є значущою, тобто лінійна залежність між вхідною та вихідною змінними відсутня. Але чи відсутня вона повністю? Можливо, лінійна залежність між змінними має місце навіть за досить малих значень коефіцієнта детермінації. Чи існує системний підхід, що дає змогу визначити наявність лінійної залежності між вхідною та вихідною змінними?

Методичну основу для оцінки значущості таких залежностей створюють *гіпотези лінійної регресії.*

Вводять у розгляд дві гіпотези – *нульова (основна) й альтернативна*. Нульова гіпотеза H_0 передбачає, що лінійні зв'язки між змінними відсутні, тоді як альтернативна гіпотеза H_a стверджує, що такі зв'язки мають місце.

Отже:

H_0: $\beta_i = 0$ – лінійний зв'язок між змінними відсутній;

Ha: $\beta_i \neq 0$ – лінійний зв'язок між змінними має місце.

Здійснюють *індивідуальні t-тести* для кожного коефіцієнта або *глобальний F-тест (тест на всю модель)*. Кожен *t-тест* оцінює, чи вносить відповідний предиктор x_i статистично значущий вклад у модель. *Глобальний F-тест* (тест на всю модель) перевіряє, чи загалом принаймні один предиктор пояснює варіацію y. F-тест – це загальна перевірка значущості всієї моделі.

Індивідуальні гіпотези дають змогу судити про кожен коефіцієнт окремо.

Глобальна гіпотеза оцінює одночасно всіх p-значення предикторів.

Щоб зробити остаточний висновок про придатність моделі до вирішення певної проблеми, потрібно визначити *ступінь її значимості*. Для цього можна використовувати *оцінку відхилення нульової гіпотези*. Для підтвердження або відхилення гіпотези використовується деякий статистичний показник, значення якого повинно бути обчислено на основі наявних спостережень

Якщо правильна гіпотеза H_0, то лінія регресії буде розташована горизонтально й побудована регресійна модель марна. З таким самим успіхом можна використовувати для оцінки просте середнє значення вихідної змінної, обчислене щодо всіх наявних спостереженнями. Навпаки, якщо вдасться відкинути нульову гіпотезу, підтвердиться значимість моделі, тобто наявність лінійної залежності між вхідною і вихідною змінними.

Наприклад, якщо значення, прийняте показником, досить малоймовірне в припущенні істинності гіпотези, то істинність гіпотези також малоймовірна і її варто відхилити. Водночас впевненість у відхиленні гіпотези тим сильніша, чим менше ймовірність появи цього значення показника. Таке перевірочне значення часто називають *p*-значенням. Чим менше *p*-значення, тим вище очікувана значимість моделі й сильніше лінійна залежність між вхідною та вихідною змінними.

Для перевірки істинності нульової та альтернативної гіпотез існують різні критерії.

Наприклад, гіпотеза H_0 може стверджувати, що вибірковий розподіл значень похибок змінної є нормальним, а альтернативна гіпотеза – що він таким не є. Тоді ймовірність значення перевірочної статистики або підтвердить, що дані мають нормальний розподіл, або покаже ступінь його відхилення від нормального. Одними з найбільш популярних критеріїв оцінки значущості регресійних моделей є *t-критерій*, побудований на основі *t-розподілу Стьюдента*, і *F-критерій*, побудований на основі *F-розподілу Фішера.*

У блокноті до лабораторної роботи надано лістинг, яким отримується комплексна оцінка нашої регресійної моделі:

1. Таблиця t-тестів для H_0: $\beta = 0$;
2. OLS Regression Results;
3. F-тест для загальної гіпотези H_0: ;
4. Детальна статистика одним блоком.

Нижче розглядаються найважливіші оцінки та їхнє пояснення в табл. 4.3.

Таблиця t-тестів для H_0: $\beta = 0$

	β̂	**SE**	**t**	**p_value**
const	–2.637299e+06	17157.8092	–153.7084	0.0000
Avg. Area Income	2.157800e+01	0.1343	160.6561	0.0000
Avg. Area House Age	1.656370e+05	1443.4130	114.7537	0.0000
Avg. Area Number of Rooms	1.206599e+05	1605.1604	75.1700	0.0000
Avg. Area Number of Bedrooms	1.651139e+03	1308.6712	1.2617	0.2071
Area Population	1.520070e+01	0.1442	105.3926	0.0000

Таблиця 4.3 – Пояснення до результатів оцінки моделі

Змінна	**Висновок (α = 0,05)**
const	Константа значуща. У моделі без змінних базова «точка відліку» була б на ≈ $ –2,6 млн (тлумачення нижче)
Avg. Area Income	**Дуже значуща.** На кожен додатковий $1 річного доходу в районі середня ціна будинку зростає ≈ $21,6
Avg. Area House Age	**Значуща.** Кожен додатковий рік середнього віку житла в районі – + $165 k
Avg. Area Number of Rooms	**Значуща.** Додаткова середня кімната в будинку району – + $121 k
Avg. Area Number of Bedrooms	**Незначуща** (p ≈ 0,21). Ефект спостерігається, але статистичних доказів недостатньо; ймовірно, ця ознака корелює з кількістю кімнат
Area Population	**Значуща.** Додатковий мешканець району додає ≈ $15 до середньої ціни будинку

F-тест для загальної гіпотези H_0:

F-тест (H_0: всі $\beta = 0$): F(5, 4994) = 11185,227, p-value = 0

F(5, 4994) = 11 185, p < 0,001

Відхиляємо H_0: разом п’ять предикторів істотно покращують модель порівняно з «порожньою» (тільки в разі наявності константи).

Детальна статистика одним блоком

Для компактності нижче подано результати комбінованого тесту на нормальність розподілу залишків (помилок) моделі, який поєднує в собі перевірку на асиметрію (skewness) і «товстощокість» (kurtosis) та їхнє пояснення в табл. 4.4.

```
==============================================================
Omnibus:                  5.580   Durbin-Watson:          2.005
Prob(Omnibus):            0.061   Jarque-Bera (JB):       4.959
Skew:                     0.011   Prob(JB):               0.0838
Kurtosis:                 2.847   Cond. No.               9.40e+05
==============================================================
```

Результати аналізу моделі та висновки

Досить високе $R^2 = 0.92$. Предиктори добре пояснюють ціль. Бажано перевірити, чи немає перенавчання на іншому (тестовому) наборі.

F-тест < 0.001. Сукупно змінні значущі: модель точно краща за константну.

Таблиця 4.4 – Пояснення результатів комбінованого тесту

Показник	Що перевіряє	Ваш результат
1	2	3
Omnibus = 5.580 Prob (Omnibus) = 0.061	Об'єднаний χ^2-тест Д'Агостіно – Пірсона (нормальність залишків: *skew* + *kurtosis*)	$p = 0.061 > 0.05 \Rightarrow$ нульова гіпотеза «залишки ~ N(0, σ^2)» не відхиляється. Нормальність прийнятна, але значення біля порогу варто перевірити на графіку QQ-plot
Jarque-Bera (JB) = 4.959 Prob(JB) = 0.084	Альтернативний тест нормальності (зважує відхилення за *skew* і *kurtosis*)	$p = 0.084 > 0.05 \Rightarrow$ також не виявлено суттєвих відхилень від нормального розподілу
Skew = 0.011	Асиметрія залишків (0 – ідеально симетрично)	$\approx 0 \Rightarrow$ симетрія майже ідеальна
Kurtosis = 2.847	«Загостреність» розподілу (3-гаусовий)	$2.85 < 3 \Rightarrow$ трохи «пласкуватий» (platykurtic), але близько до нормального

Продовження таблиці 4.4

1	2	3
Durbin-Watson = 2.005	Наявність автокореляції порядку 1 у залишках (0 … 4; 2 – відсутня)	~ 2.0 ⇒ серіальної кореляції не виявлено; припущення незалежності залишків виконується
Cond. No. = 9.40×10^5	Число обумовленості матриці X (оцінка мультиколінеарності / масштабів)	$> 10^4$ досить велике ⇒ ймовірна сильна мультиколінеарність або значні розбіжності в масштабах ознак. Може «роздувати» стандартні помилки й робити оцінки нестійкими. Бажано нормалізувати (scaling) ознаки

Розміри вибірки й $df : n \gg p$, тому класичні припущення *OLS* зазвичай тримаються.

Чотири з п’яти предикторів значущі. *Avg. Area Income* і *Area Population* мають найменші похибки → найбільш надійні.

Avg. Area Number of Bedrooms не пройшов тест – його вклад у присутності «Rooms» статистично не доведено (ймовірна мультиколінеарність). Можна спробувати вилучити або змінити представлення цієї ознаки.

Знак і масштаб β: усі значущі коефіцієнти додатні: збільшення значення ознаки збільшує ціну.

Avg. Area House Age: «старіння» житла в середньому на один рік у регіоні додає до ціни $165 тис. (більш вказує, що «старіший район» є елітним).

Avg. Area Number of Rooms: збільшення житла на одну кімнату додає до ціни $121 тис.

Питання для самоконтролю

1. У чому полягає основна мета регресійно-кореляційного аналізу?

2. Що виступає в ролі «учителя» для моделі лінійної регресії з погляду машинного навчання?

3. Що таке кореляція та якими бувають її коефіцієнти?

4. Який принцип лежить в основі методу найменших квадратів (МНК) під час побудови лінії регресії?

5. Що показує коефіцієнт детермінації (R^2) і як його інтерпретувати?

6. Що таке середньоквадратичне відхилення (СКВ) і як воно пов'язане із дисперсією?

7. У чому різниця між рівнянням регресії, побудованим для вибірки, та регресійною моделлю для генеральної сукупності?

8. Що таке нульова (H_0) та альтернативна (H_a) гіпотези під час перевірки значущості коефіцієнтів регресії?

9. Що таке p-значення (p-value) і як воно допомагає ухвалити рішення про відхилення нульової гіпотези?

10. Для чого використовують t-тест та F-тест під час оцінки регресійної моделі?

11. Які основні бібліотеки Python та їхні функції були використані в прикладі для побудови моделі лінійної регресії?

12. З якою метою в прикладі використовується метод *train_test_split*?

13. Для чого в розвідувальному аналізі даних (РАД) будували теплову карту кореляцій (heatmap)?

14. Які основні метрики точності (наприклад, MAE, MSE, RMSE) були розраховані для оцінки якості побудованої моделі?

15. Чому важливо розраховувати відносні показники помилок (наприклад, відносну MAE або MAPE)?

16. Яка змінна виявилася статистично незначущою за результатами t-тесту ($p > 0.05$) і яка ймовірна причина цього?

17. Що показує F-тест щодо загальної значущості побудованої моделі?

18. Для чого будується гістограма залишків моделі і який висновок було зроблено на її основі в прикладі?

4.2. Машинне навчання в класифікаційних задачах

4.2.1. Основні методи класифікації

Методи класифікації відіграють вирішальну роль у розв'язанні багатьох практичних задач машинного навчання: від прийняття кредитних рішень і виявлення шахрайства до аналізу настроїв у фінансових новинах і прогнозування ринкових сигналів. Завдяки класифікаційним алгоритмам перетворюємо складний багатовимірний простір ознак на зрозумілі категорії, що дає змогу як автоматизувати процеси прийняття рішень, так і забезпечувати високу точність там, де це критично (наприклад, у процесах оцінки кредитоспроможності чи виявлення аномальних транзакцій). Кожен метод має свої сильні та слабкі сторони: лінійні моделі добре підходять для швидких і прозорих рішень на невеликих даних, а ансамблеві методи й нейронні мережі демонструють високу продуктивність та здатність захоплювати складні відношення між ознаками.

Крім того, вибір оптимального алгоритму класифікації тісно пов'язаний із особливостями задачі та даних: розміром вибірки, балансом класів, вимогою до інтерпретації результатів і обчислювальними ресурсами. Наприклад, у фінансових системах, де важлива як швидкість прийняття рішення, так і прозорість моделі для аудиту, часто поєднують прості лінійні методи з більш складними ансамблевими та мережевими підходами в конвеєрі обробки даних. У результаті розуміння особливостей кожного методу класифікації дає змогу будувати надійні, інтерпретовані й ефективні системи для автоматизації бізнес-процесів і прийняття обґрунтованих рішень. У таблиці 4.5 подані основні методи класифікації з учителем (supervised), у таблиці 4.6 – без учителя (unsupervisedта), а також популярні Python-бібліотеки для їхньої реалізації.

Таблиця 4.5 – Методи класифікації з учителем (supervised) та Python-бібліотеки

Метод класифікації	Python-бібліотеки	Опис алгоритму	Застосування у фінансах
1	2	3	4
Логістична регресія	scikit-learn (LogisticRegression), statsmodels	Оцінює ймовірність належності до бінарного класу через сигмоїдну функцію від лінійної комбінації ознак	Кредитний скоринг (оцінка ймовірності дефолту), моделювання «прийняття / відмови» в кредитуванні
Метод опорних векторів (SVM)	scikit-learn (SVC, LinearSVC)	Шукає гіперплощину з максимальним відступом, що розділяє класи; ядра дають змогу працювати з нелінійними даними	Виявлення шахрайства (fraud detection), розпізнавання аномальних транзакцій
Дерева рішень	scikit-learn (DecisionTreeClassifier) decision-tree-id3 sklearn-c45	Рекурсивно розбиває простір ознак за порогами, формуючи набір інтерпретованих правил «якщо – то». ID3/C4.5	Оцінка кредитного ризику, клієнтський сегментинг за фінансовою поведінкою
k-найближчих сусідів (k-NN)	scikit-learn (KNeighborsClassifier)	Для кожного нового зразка шукає k найближчих сусідів у просторі ознак і класифікує за більшістю їх класів	Аномалії в торгівлі цінними паперами, первинна класифікація транзакцій як нормальних / аномальних
Наївний баєсівський (Naive Bayes)	scikit-learn (GaussianNB, MultinomialNB, BernoulliNB), nltk (для тексту)	Обчислює апостеріорні ймовірності класів за теоремою Байєса з припущенням умовної незалежності ознак	Сентимент-аналіз фінансових новин і повідомлень, класифікація тексту для алгоритмічної торгівлі

Продовження таблиці 4.5

1	2	3	4
Випадковий ліс (Random Forest)	scikit-learn (RandomForestClassifier)	Ансамбль дерев на випадкових підвибірках даних і ознак; прогноз – за голосуванням дерев	Прогнозування дефолтів, скоринг портфелів, оцінка вартості активів
Градієнтний бустінг	scikit-learn (HistGradientBoostingClassifier), xgboost (XGBClassifier), lightgbm (LGBMClassifier), catboost (CatBoostClassifier)	Послідовне навчання дерев, де кожне коригує помилки попереднього через градієнтний спуск у просторі функцій	Алгоритмічна торгівля, прогнозування цін акцій / валют, класифікація сигналів купівлі-продажу
Штучні нейронні мережі (NN)	scikit-learn (MLPClassifier), TensorFlow / Keras, PyTorch	Багатошарові персептрони або їх варіанти (CNN, RNN) моделюють складні нелінійні залежності через шари нейронів	Прогноз цінових часових рядів, алгоритмічна торгівля, кредитний скоринг із використанням нетипових ознак

Таблиця 4.6 – Методи класифікації без учителя (unsupervised) та Python-бібліотеки

Метод	Python-бібліотеки	Опис алгоритму	Застосування у фінансах
1	2	3	4
K-means	scikit-learn (KMeans), yellowbrick (для візуалізації)	Розбиває дані на kk груп за рахунок ітеративного перерахунку центрів кластерів і переназначення точок	Сегментування клієнтів за витратами, портфельне групування активів
Ієрархічна агломеративна кластеризація	scikit-learn (AgglomerativeClustering), scipy.cluster	Будує дерево кластерів (дендрограму) через послідовне об’єднання (або розбиття) груп	Аналіз кореляцій між активами, групування фінансових показників компаній

Продовження таблиці 4.6

1	2	3	4
DBSCAN	scikit-learn (DBSCAN)	Визначає кластери на основі щільності точок: кластери – сфери з високою концентрацією точок	Виявлення аномальних транзакцій (шахрайських операцій)
Gaussian Mixture Models (GMM)	scikit-learn (GaussianMixture)	Моделює дані як суму гаусіанських компонент; кожна точка належить компоненті з певною ймовірністю	Оцінка ризику через імовірнісне групування клієнтів, сегментація портфелів з урахуванням невизначеності
PCA + K-means	scikit-learn (PCA + KMeans)	Спочатку знижує розмірність за допомогою PCA, потім кластеризує в зменшеному просторі	Виділення ключових чинників ризику, оптимізація кластерного аналізу в разі значної кількості ознак

4.2.2. Основи застосування інструментарію дерев рішень

Дерева рішень – ієрархічні деревоподібні структури, що складаються з вирішальних правил виду «якщо... то...» і дають змогу виконувати класифікацію об’єктів. У дереві кожному об’єкту відповідає єдиний вузол, що дає рішення. Термін «дерево рішень» використовується й у суміжних сферах аналізу й обробки даних і штучного інтелекту. Наприклад, у завданнях алгоритмізації *дерево рішень* – це спосіб представлення процесу прийняття рішення, що має вигляд відповідей на ряд питань, які утворюють деревоподібну структуру, кожна кінцева вершина якої становить елементарне рішення. У менеджменті під деревом рішень розуміють графічне зображення альтернативних дій і їхніх наслідків. Однак між усіма цими тлумаченнями терміна багато спільного.

Дерева рішень (*decision trees*) належать до найбільш популярних і потужних інструментів МН, що дають змогу ефективно вирішувати завдання класифікації та регресії. На відміну від методів, які використовують статистичний підхід, як-от класифікатор Байєса, лінійна й логістична регресія, дерева рішень засновані на машинному навчанні й здебільшого не вимагають припущень про статистичний розподіл значень ознак.

Основні поняття методології дерев рішень

В основі дерев рішень лежать вирішальні правила виду «якщо... то...», які можуть бути сформульовані на природній мові. Тому дерева рішень є найбільш наочними й легко інтерпретуються моделями.

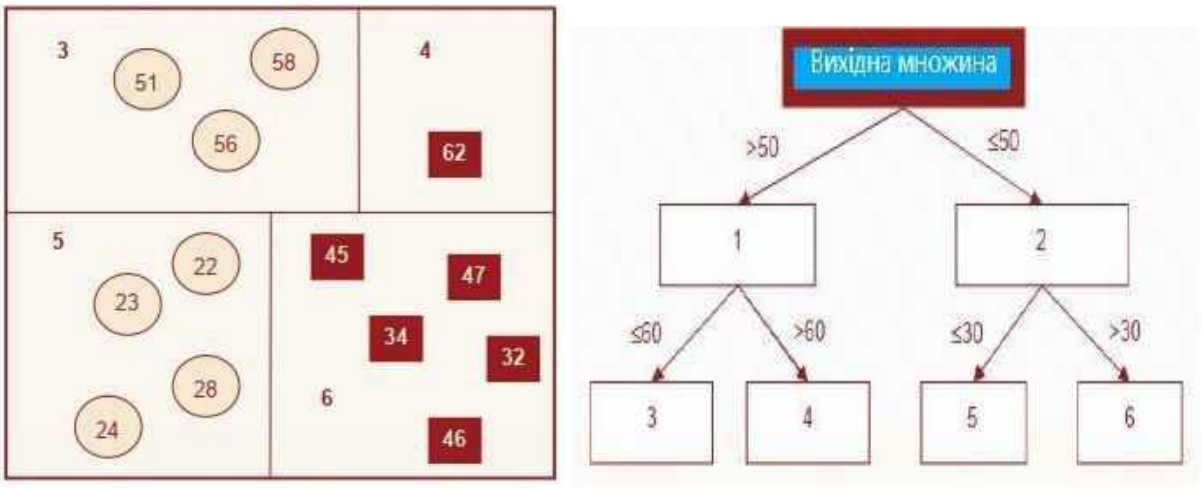
Для зручності наведемо базові поняття теорії дерев рішень у таблиці 4.7.

Таблиця 4.7 – Базові поняття теорії дерев рішень

Назва	Опис
Об’єкт	Приклад, шаблон, спостереження, запис
Атрибут	Ознака, незалежна змінна, властивість, вихідне поле
Мітка класу	Залежна змінна, цільова змінна, вихідне поле
Вузол	Внутрішній вузол дерева
Лист	Кінцевий вузол дерева, вузол рішення
Перевірка	Умова вузла

Розглянемо основну ідею алгоритмів побудови дерев рішень на прикладі.

Потрібно передбачити повернення або неповернення кредиту за допомогою набору вирішальних правил на основі єдиного атрибута «Вік клієнта». Для цього будемо використовувати множину спостережень, в кожному з яких вказується вік, а також факт повернення / неповернення позики. Графічно таку множину спостережень подано на рис. 4.10 (а).



а) б)

Рисунок 4.10 – Множина об’єктів і дерево рішень, що їх класифікує

Умовно приймемо, що об’єкт у формі кола вказує на повернення клієнтом займу, у формі прямокутника – на дефолт (неповернення) за займом, а всередині кожного об’єкта – зазначений вік. Кожен із клієнтів також може характеризуватись іншими показниками, наприклад сумою неповернення, та іншими чинниками, які позиціонують їх у різних зона з ризику, як це показано на рис. 4.10 (а). Потрібно розбити множину об’єктів (клієнтів) на підмножини так, щоб у кожну з них потрапили об’єкти тільки одного класу. Як видно на рис. 4.10 (б), структура дерев рішень проста й загалом аналогічна деревовидним ієрархічним структурам, що використовуються в інших сферах МН, наприклад деревах асоціативних правил.

До складу дерев рішень входять два види об’єктів – вузли (node) і листя (leaf). У вузлах містяться правила, за допомогою яких проводиться перевірка атрибутів, і множина об’єктів у цьому вузлі розбивається на підмножини. Листя – це кінцеві вузли дерева, в яких містяться підмножини, асоційовані з класами. У вузлах і листі вказуються показники ентропії, яка характеризує залишкову невизначеність розбиття (рис. 4.11).

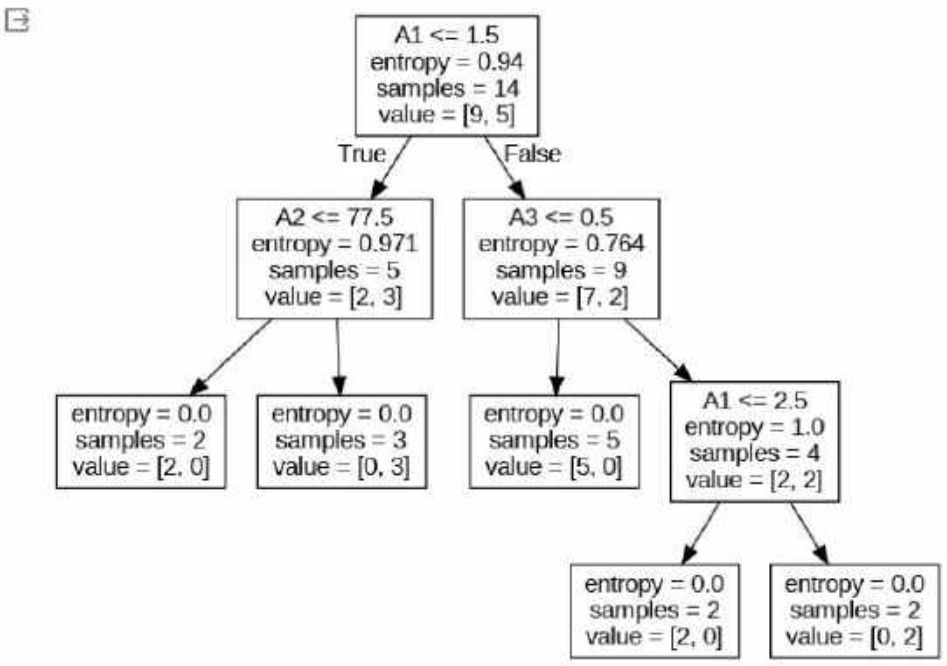


Рисунок 4.11 – Приклад кінцевих вузлів дерева

Основною відмінністю листа від вузла є те, що в листі не проводиться перевірка, яка розбиває асоційовану з ним підмножину і, відповідно, немає розгалуження. Листом може бути оголошений будь-який вузол, якщо прийнято рішення, що множина у вузлі досить однорідна в плані класової приналежності об’єктів і подальше розбиття не має сенсу, оскільки не приведе до істотного збільшення точності класифікації, а тільки ускладнить дерево. Дерево (рис. 4.10 (б) класифікує об’єкти, подані на рис. 4.10 (а), та об’єкти з номерами 1 і 2 – вузли, а з номерами 3, 4, 5 і 6 – листя. Звернемо увагу на те, що в дереві є по два листи, асоційованих з одним класом. У цьому немає будь-якого протиріччя, просто для класифікації об’єктів у них використовувалися різні способи перевірки, це може бути критерій обсягу неповернення тощо.

Для кожного аркуша в дереві є унікальний шлях. Початковий вузол дерева є вхідним: через нього проходять усі об’єкти. Зазвичай вхідний вузол називають кореневим вузлом (root node).

Отже, дерево росте зверху вниз. Вузли й листя, підлеглі вузлу вищого ієрархічного рівня, називаються *нащадками*, або *дочірніми вузлами*, а той вузол стосовно них – *предком,* або *батьківським вузлом*.

4.2.3. Алгоритми дерев рішень ID3, C4.5

Алгоритми дерев рішень ID3

Одними з найбільш популярних алгоритмів побудови дерев рішень є алгоритм ID3 і його модифікація C4.5. Алгоритм ID3 починає роботу з усіма навчальними прикладами в кореневому вузлі дерева. Для поділу масиву прикладів кореневого вузла вибирається один з атрибутів, і для кожного значення, прийнятого цим атрибутом, будується гілка та створюється дочірній вузол. Потім усі приклади розподіляються за дочірніми вузлами відповідно до значення атрибута.

Нехай атрибут X приймає три значення: A, B і C (рис. 4.12). Тоді під час розбиття вихідного масиву T по атрибуту X алгоритм створить три дочірніх вузли T_1(A), T_2(B) і T_3(C), у першому з яких будуть розміщені всі записи зі значенням A, в другому – зі значенням B, а в третьому – із значенням C.

Алгоритм повторюється рекурсивно, допоки у вузлах не залишаться тільки приклади одного класу, після чого вузли будуть оголошені листами й розбиття припиниться. Найбільш проблемним етапом алгоритму є вибір атрибута, за яким буде проводитися розбиття в кожному вузлі.

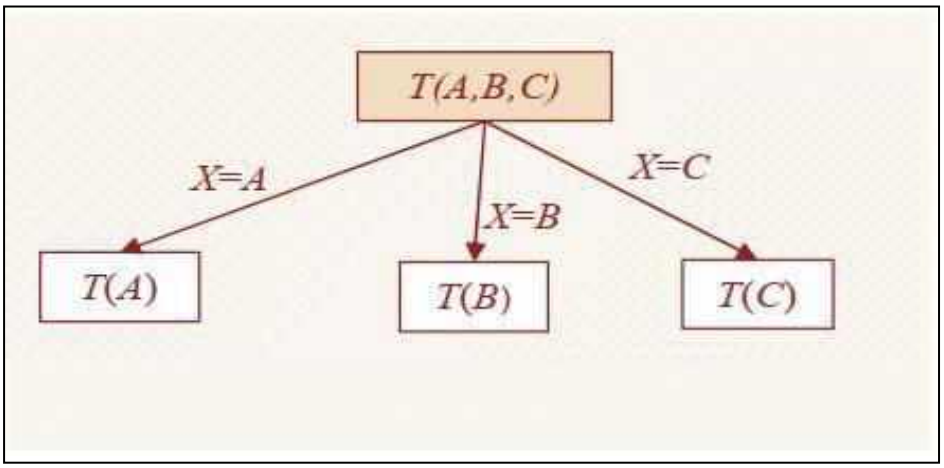


Рисунок 4.12 – Початкові вузли класифікаційного дерева

Для вибору атрибута розбиття метод ID3 використовує критерій, який називається приростом інформації (*information gain*) або зменшенням ентропії (*entropy reduction*).

Дамо основні поняття *інформації* та *ентропії* для розуміння алгоритму.

Інформація в теорії зв'язку та теорії ймовірностей розглядається як міра зменшення невизначеності про випадкову подію.

Невизначеність (ентропія). Для дискретного джерела з можливими повідомленнями (подіями) $x_1, x_2, \ldots, x_n$ з імовірностями $p_1, p_2, \ldots, p_n$ невизначеність (Shannon-ентропія) визначається як (4.8):

$$H = -\sum_{i=1}^{n} p_i \log_2 p_i. \qquad (4.8)$$

Чим вище значення H, тим більше середньої «непередбачуваності» містить джерело інформації. Також важливим є поняття біту інформації.

Приклад. У грі «орел чи решка» (дві рівноймовірні сторони монети): кидок монети дає 1 біт інформації. У грі із гральним кубиком (6 рівноймовірних результатів) вибір одного дорівнює $\log_2 6 \approx 2.5856$ біт.

Біт інформації. Біт (англ. *bit = binary digit*) – базова одиниця інформації в бінарній системі. Один біт відповідає тому обсягу інформації, який зменшує невизначеність у два рази (тобто пов'язаний із поділом простору варіантів навпіл). Математично, якщо $p_i = ½$, то

$$I(x_i) = -\log_2\left(\tfrac{1}{2}\right) = 1 \text{ біт}. \qquad (4.9)$$

1 біт – це інформація, потрібна для відповіді на питання «так / ні» з рівними шансами (50 % / 50 %).

Введемо у розгляд *міру приросту інформації*, яка обчислюється як (4.10):

$$\text{Gain (S)} = \text{Info (T)} - \text{InfoS (T)}, \qquad (4.10)$$

де:

Info (T) – ентропія множини *T* до розбиття;

InfoS (T) – ентропія після розбиття *S*.

Ця міра є приростом кількості інформації, отриманої в результаті поділу масиву T на підмножини T_1, T_2, ..., T_k за допомогою розбиття S. Щодо найкращого атрибута для використання у разі потреби розділення S, вибирається той атрибут, який забезпечує найбільший приріст інформації Gain (S).

Кількість інформації. Самоінформація – повідомлення x_i з імовірністю p_i – це міра зменшення невизначеності, коли дізнаємося, що відбулася саме ця подія. Вона визначається як:

$$I(x_i) = -\log_2 p_i. \qquad (4.11)$$

Розглянемо приклад використання алгоритму. Нехай дано набір даних T, який містить 14 записів і 3 вхідних атрибути A_1, A_2, A_3, а також вихідний атрибут (мітку класу) C, який приймає два значення – C_1 і C_2 (табл. 4.8).

Таблиця 4.8

№ з/п	A_1	A_2	A_3	C
1.	A	70	Так	C1
2.	A	90	Так	C2
3.	A	85	Ні	C2
4.	A	95	Ні	C2
5.	A	70	Ні	C1
6.	B	90	Так	C1
7.	B	78	Ні	C1
8.	B	65	Так	C1
9.	B	75	Ні	C1
10.	C	80	Так	C2
11.	C	70	Так	C2
12.	C	80	Ні	C1
13.	C	80	Ні	C1
14.	C	96	Ні	C1

У цьому прикладі 9 записів належить до класу C_1, а 5 – до класу C_2. Ентропія множини T буде:

$$\text{Info}(T) = -(9/14) \times \log_2(9/14) - (5/14) \times \log_2(5/14) = 0{,}94 \text{ біт}.$$

Розглянемо розбиття S_1 масиву на основі категоріальних атрибутів A_1, A_3.

Після використання атрибута A_1 для розбиття вихідної множини на три підмножини: $T_1(A)$, $T_2(B)$ і $T_3(C)$ загальна ентропія розбиття становитиме (умовно вважаємо, що вирази типу $0/4 \times \log_2(0/4) = 0 \times \infty = 0$):

```
InfoS1(T)=5/14(-2/5×log2(2/5)-3/5log2(3/5)) +
4/14(-4/4×log2(4/4)- 0/4×log2(0/4)) +
5/14(-3/5×log2(3/5)- 2/5×log2(2/5)) = 0,694 біт.
```

Отже, приріст інформації під час використання атрибута A_1 для розбиття вихідної підмножини становитиме:

```
Gain (S1) = 0,94 - 0,694 = 0,246 біт.
```

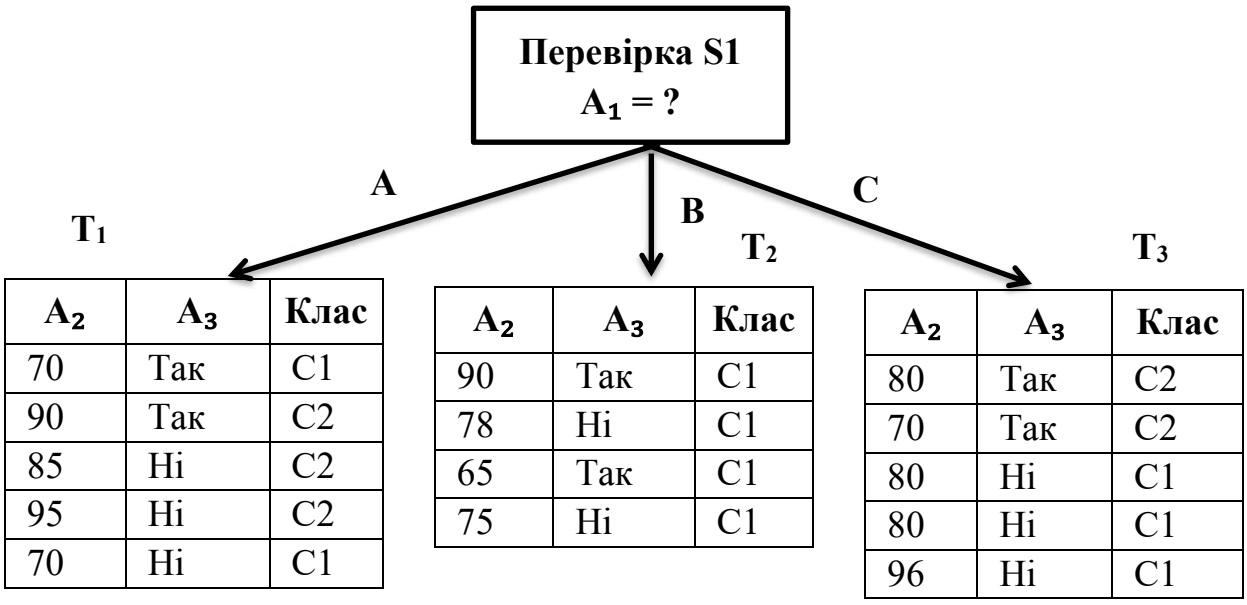
Під час використання розбиття на основі атрибута A_3, провівши аналогічні обчислення, отримаємо:

```
InfoS3(T) = 6/14 (- 3/6 × log2(3/6) - 3/6log2(3/6)) +
8/14 (-6/8× log2(6/8)- 2/8 × log2(2/8)) = 0,892 біт.
```

Тоді приріст інформації, який забезпечується розбивкою вихідної множини на основі атрибута A_3, буде:

```
Gain (S3) = 0,94 - 0,892 = 0,048 біт.
```

Отже, для початкового поділу множини T алгоритм вибере розбиття $\mathbf{S_1}$ з використанням атрибута A_1, що забезпечує найбільший приріст інформації (рис. 4.13).



A_2	A_3	Клас
70	Так	C1
90	Так	C2
85	Ні	C2
95	Ні	C2
70	Ні	C1

A_2	A_3	Клас
90	Так	C1
78	Ні	C1
65	Так	C1
75	Ні	C1

A_2	A_3	Клас
80	Так	C2
70	Так	C2
80	Ні	C1
80	Ні	C1
96	Ні	C1

Рисунок 4.13 – Три гілки класифікаційного дерева
(за даними табл. 4.8)

Для знаходження оптимального розбиття треба також розглянути числовий атрибут A_2. Перевірки за такими атрибутами складніші, оскільки потрібна установка порогів для розбиття, яка в загальному випадку може бути довільною. Існує алгоритм для визначення оптимального значення порога z. Спочатку навчальні приклади упорядковано в тому порядку, в якому будуть розглядатися значення атрибута: (v_1, v_2, ... v_m). Тоді існує тільки m – 1 порогів розбиття так, щоб кожне з них було перевірено на предмет максимального приросту інформації. Водночас як поріг можуть вибиратися серединні точки між сусідніми значеннями (v_i, v_{i+1})/2 або, як варіант, найменші значення з кожного інтервалу (v_i, v_{i+1}).

Для ілюстрації цього способу знаходження порога візьмемо числовий атрибут A_2. Після сортування прикладів за зростанням значень атрибута отримаємо такий порядок {65, 70, 75, 78, 80, 85, 90, 95, 96} і множину можливих порогових значень z = {65, 70, 75, 78, 80, 85, 90, 95}. З дев'яти значень оптимальним буде те, яке забезпечує найбільший приріст інформації. Якщо для кожного потенційного порога визначити приріст інформації подібно до того, як раніше це робилося для значень категоріальної змінної, то можна виявити, що кращим значенням порога буде z = 80. Обчислення приросту інформації для S_2 ($A_2 \leq 80$ або $A_2 \geq 80$) проводиться так:

$$\text{Info}_{\mathbf{S_2}}(T) = 9/14\ (-7/9 \times \log_2(7/9) - 2/9\log_2(2/9)) + 5/14$$
$$(-2/5 \times \log_2(2/5) - 3/5 \times \log_2(3/5)) = 0{,}837 \text{ біт};$$
$$\text{Gain}(\mathbf{S_2}) = 0{,}94 - 0{,}837 = 0{,}103 \text{ біт}.$$

Тепер можна порівняти приріст інформації для всіх трьох атрибутів ілюстративного прикладу. Бачимо, що атрибут A_1, як і раніше, забезпечує найбільший приріст інформації – 0.246, і отже, він повинен бути обраний для використання в першому розбитті дерева рішень. Кореневий вузол перевірить для атрибута A_1, в результаті чого будуть створені три гілки – по одній для кожного значення атрибута (рис. 4.13).

Після початкового розбиття всі вузли-нащадки міститимуть по кілька спостережень з вихідної множини, і для кожного вузла

потім буде повторений процес вибору атрибута розбиття й оптимізації порогу (якщо для розбиття використовується числовий атрибут).

Оскільки вузол-нащадок T_2, отриманий з гілки для значення B, містить 4 спостереження, які належать до одного класу, то ентропія дорівнює 0, вузол оголошується листом, і подальше розгалуження для нього не проводиться.

Для вузла T_1, що охоплює 5 спостережень, може бути зроблена перевірка за знайденими атрибутами. Опускаючи проміжні розрахунки, констатуємо, що оптимальне розбиття буде досягнуто за допомогою атрибута A_2 для альтернативних варіантів $A_2 \leq 70$ або $A_2 > 70$.

Оскільки множина T_1 містить 5 спостережень, з яких два належать до класу C_1, а три – до класу C_2, то, відповідно, до формули ентропії можна записати:

$$\text{Info}S_2(T_1) = -2/5 \times \log_2(2/5) - 3/5\log_2(3/5) = 0{,}97 \text{ біт.}$$

Під час використання атрибута A_2 для розбиття T_1 з порогом 70 отримаємо чисті вузли, тому $\text{Info}S_2(T_1) = 0$.

Приріст інформації, забезпечений цим розбиттям, становитиме:

$$\text{Gain}(S_2) = 0{,}97 - 0 = 0{,}97 \text{ біта}$$ й буде максимальним.

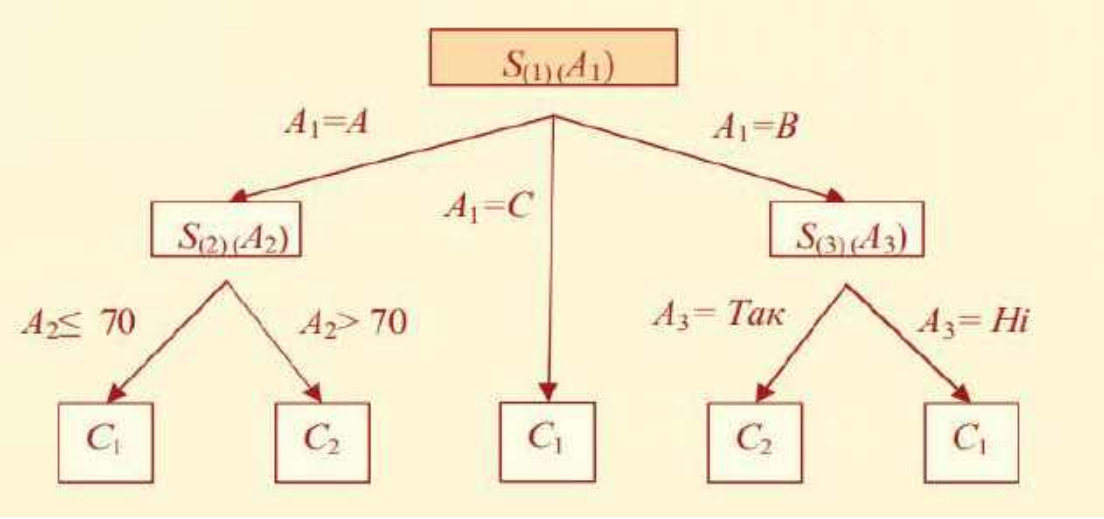


Рисунок 4.14 – Результуюче дерево класифікації

Аналогічно можуть бути виконані розрахунки й для підмножини T_3. У ньому оптимальна перевірка за значеннями атрибута A_3. Отже, будуть створені дві нові гілки, що відповідають значенням A_3 = Так і A_3 = Ні. У результаті отримаємо дві однорідних підмножини, що містять спостереження одного класу. Результуюче дерево подано на рис. 4.14.

Практичне застосування класичного алгоритму ID3 пов'язане з рядом проблем, характерних для моделей, заснованих на навчанні взагалі й дерев рішень зокрема. Основними з них є перенавчання та наявність пропусків у даних. Для їхнього ефективного подолання ID3 був доопрацьований, в результаті чого з'явилося його розширення, назване C4.5.

Алгоритми дерев рішень C4.5

Алгоритм – розвиток ID3, який уніфікує й доповнює його можливості:

– використовує так званий *Gain Ratio* (відношення *Information Gain* до ентропії ознаки), що усуває упередженість до ознак із багатьма категоріями;

– дає змогу працювати з *числовими ознаками* (шукає порогові значення й робить бінарні спліти);

– обробляє *пропущені значення* й нечіткі дані. Після побудови виконує *обрізку дерева* (post-pruning) для запобігання перенавчанню.

Алгоритми дерев рішень CART

Алгоритм CART (*Classification and Regression Trees*) використовується для оцінки так званої «чистоти» вузла під час побудови дерева рішень. У цьому алгоритмі використовується метрика *Джині (Gini Impurity).*

Формула: для вузла, в якому присутні K класів із частками p1, p2, …, p_K, *невизначеність* (*Gini Impurity*) обчислюється так:

$$G = 1 - \sum_{k=1}^{K} p_k^2 \tag{4.12}$$

$$p_k = \frac{\text{кількість зразків класу } k}{\text{загальна кількість зразків у вузлі}} \tag{4.13}$$

$G = 0 \rightarrow$ вузол *чистий*, тобто містить тільки один клас ($p_i = 1$ для деякого і всі інші $p_k = 0$).

$G \rightarrow 1 - 1 / K \rightarrow 1 \rightarrow$ вузол максимально «змішаний»: рівні частки всіх K класів ($p_k = 1$) дають найбільшу невизначеність.

Для *бінарної класифікації* (K = 2) спрощено:

$$G = 2p(1 - p),$$

де p – частка одного з класів (наприклад, «позитивних»).

Використання для виконання розбиття множини

На кожному вузлі алгоритм перебирає всі можливі ознаки й пороги, розбиває дані на дві групи.

Для кожного розбиття обчислює зважену суму Gini Impurity дочірніх вузлів:

$$G_{\text{split}} = \frac{N_L}{N} G_L + \frac{N_R}{N} G_R \qquad (4.14)$$

де:

N_L, G_L *і* N_R, G_R – кількість зразків і Gini в лівому та правому підвузлах;

N – загальна кількість у батьківському вузлі.

Вибирається спліт, що *мінімізує* G_{split} (тобто дає максимальне зменшення невизначеності).

Переваги Gini Impurity

Швидкий та інтуїтивний алгоритм: обчислюється за простими операціями над квадратами часток.

Чутливий до неоднорідності: добре відрізняє вузли з домінуванням одного класу від «змішаних».

Ефективний: зазвичай дає дерева схожої якості до Information Gain, але швидше рахується.

Gini Impurity – це *міра невизначеності* у вузлі дерева рішень, яку мінімізує CART під час кожного розбиття, обираючи те правило (ознака + поріг), що найкраще розділяє дані за класами.

Питання для самоконтролю

1. Поясніть терміни та наведіть формули: біт інформації, ентропія.

2. Поясніть алгоритми дерева рішень ID3 і C4.5.

3. Що означає помилка першого й другого типу під час класифікації?

4. Що означає термін Семплінг під час розбивання даних на навчальну й тестову вибірки?

5. Що означає бутстреп оцінка (переналагодження)?

6. Як за допомогою Lift-кривої можна визначити розмір вибірки?

7. Як визначається ефективність моделей, описуваних Lift-кривими?

8. Як характеризується якість моделей на основі площ ROC кривих?

9. Як Lift-діаграми використовують для порівняння моделей з погляду витрат класифікації?

4.3. Оцінювання якості класифікаційних моделей

4.3.1. Основні показники класифікаційних якостей моделей

На цій стадії проєкту оцінюється, наскільки модель задовольняє початково сформульованій бізнес-цілі й критеріям успіху. Якщо модель має передбачити клієнтів, які, ймовірно, придбають продукт, візьмуть / повернуть кредит тощо, то чи може деталь достатньо розрізнити два класи – клієнт є потенційним покупцем чи позичальником, який поверне кредит та ін.? Чи можна покращити модель, додавши інші дані? Чи варто вносити дані про транзакції, як-от покупки (дані про ринковий кошик)? Чи потрібно додавати в модель витрати, пов'язані з помилковими позитивними або помилковими негативними результатами?

Цей крок діє як перевірка якості, на основі якої можете визначити наступні кроки, як-от розгортання проєкту, або ініціювати подальші ітерації чи протестувати проєкт у передвиробничому середовищі, якщо це дозволяють обмеження. Отже, на цьому етапі потрібно:

– переглянути за потреби бізнес-цілі;

– оцінити результати за критеріями успіху;

– визначити наступні кроки;

– оцінити втрати в разі помилкової класифікації.

Помилки, яких припускається модель під час її практичного застосування на нових даних, можуть призводити до хибних висновків, що призводить до матеріальних, а отже, і фінансових витрат від неправильно ухвалених рішень. Такі витрати часто називають втратами помилки класифікації (*classification cost error*). Дійсно, під час розроблення моделі для кредитного скорингу можемо поставити за мету виявляти як сумлінних, так і недобросовісних позичальників. Закономірним є запитання: «Що краще – прийняти добросовісного клієнта за недобросовісного чи навпаки?» Відповідь на це очевидна навіть для неспеціаліста: у першому випадку втрачаємо тільки відсотки за кредитом, який не було видано, а в другому – всю суму, видану недобросовісному позичальнику. Тобто витрати помилок другого виду більші.

Під час побудови моделі потрібно мінімізувати ймовірність появи помилок, які спричиняють найбільші витрати класифікації. Така методика називається класифікацією з урахуванням витрат (*cost-sensitive classification*).

Типовими прикладами, коли витрати для будь-яких типів помилок різні, є:

– *кредитний скоринг* – витрати видачі кредиту несумлінним позичальникам істотно вищі, ніж втрати через відмову у видачі кредиту сумлінним клієнтам;

– *пошук середовища нафтового забруднення* – втрати від пропуску реальної нафтової плями істотно вищі, ніж втрати в разі помилкової тривоги;

– *технічна діагностика* – витрати неправильної ідентифікації проблеми істотно менші, ніж витрати пропуску проблеми;

– *директ-маркетинг* – витрати надсилання тематичної пошти клієнтам, які не відповідають на неї, істотно менші, ніж втрати через пропуск потенційних клієнтів.

Загалом важко знайти галузь, у якій витрати різних типів помилок були б однаковими. У бінарній класифікації кожний окремий результат класифікації, тобто віднесення спостереження до одного із класів, має чотири результати:

– істинно позитивний (true positive, TP);

– істинно негативний (true negative, TN);

– хибно позитивний (false positive, FP);

– хибно негативний (false negative, FN).

Нехай як позитивний результат передбачення обрано значення *Так*, а як негативний – *Ні.*

Істинно позитивним результат буде, коли фактичне цільове значення для цього прикладу *Так* і модель на виході видасть *Так*.

Істинно негативним результат буде, коли цільове значення для цього спостереження *Ні* й модель видасть *Ні.*

Хибно позитивне значення має місце, коли цільовий клас для спостереження *Ні*, а модель для нього сформує вихід *Так.*

За *хибно негативного* виходу цільова змінна набуває значення *Так*, а на виході модель видасть *Ні* (рис. 4.15).



Рисунок 4.15 – Матриця класифікації
(випадок із двома класами)

Уведемо в розгляд два основні показники, що дають змогу оцінити точність бінарної класифікаційної моделі.

Загальний показник успіху, або загальна точність (overall success rate, OSR, accuracy) – це число правильно класифікованих спостережень, віднесене до загальної кількості спостережень:

$$OSR = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.15}$$

Цю величину називають ще точністю класифікатора.

Загальний показник помилки (*overall error rate, OVR*) становитиме:

$$OVR = \frac{FP + FN}{TP + TN + FP + FN} \tag{4.16}$$

Для подальшого розгляду питань, пов’язаних з оцінкою ефективності та порівнянням моделей бінарної класифікації, введемо також такі показники:

1. *Позитивна прогностична цінність, точність* (precision). З усіх випадків, коли модель передбачила цей клас, яка частка виявилася правильною.

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{4.17}$$

2. *Чутливість, повнота* (sensitivity / recall, Se). Частка, яку модель змогла знайти з усіх реальних прикладів цього класу.

Визначається як відношення числа істинно позитивних спостережень до числа фактично позитивних спостережень (табл. 4.6):

$$Se = \frac{TP}{TP + FN} \tag{4.18}$$

3. *Специфічність* (specificity, Sp) визначається як відношення числа істинно негативних спостережень до числа фактично негативних спостережень:

$$S_P = \frac{TN}{TN + FP} \tag{4.19}$$

4. *Support (підтримка).* У різних контекстах машинного навчання й аналізу даних завжди пов'язане з частотою або ймовірністю появи певної події чи поєднання ознак у вибірці, але конкретне тлумачення й способи обчислення залежать від задачі. У звіті *classification_report* з *sklearn* параметр *support* означає *кількість* (або розмір підмножини) реальних прикладів кожного класу в тестовій вибірці. Але у контексті пошуку асоціативних правил, наприклад у транзакціях (методи Apriori, FP-Growth тощо), *support* має інший зміст і визначається як:

$$\text{support}(X) = \frac{\text{кількість транзакцій, у яких міститься набір } X}{\text{загальна кількість транзакцій}} \tag{4.20}$$

(*Асоціативні правила розглядаються в наступному розділі*).

Таблиця 4.9 – Пояснення до метрик Precision, Sensitivity

Метрика	Формула	Що вимірює
Precision, **Позитивна прогностична цінність**	TP / (TP + FP)	Наскільки «чисті» наші позитивні прогнози (мінімум хибних «позитивних»)
Sensitivity (Recall) **Чутливість, повнота**	TP / (TP + FN)	Наскільки «повно» виявляємо всі справжні позитивні випадки (мінімум пропущених «позитивних»)

Коли що важливіше?

Якщо **FP** досить дорого обходяться (наприклад, у спам-фільтрі: краще пропустити трохи спаму, ніж вилучити важливий лист) – оптимізуємо **Precision.**

Якщо **FN** неприйнятні (наприклад, у скринінгу онкології: краще помилково направити здорового на додаткові тести, ніж пропустити хворого) – оптимізуємо **Sensitivity.**

Приклад. Уявіть задачу виявлення шахрайських транзакцій.

High Precision: кожна транзакція, позначена як «fraud», дійсно шахрайська (мінімум false alarms).

High Sensitivity: зловмисні транзакції майже всі спіймані (мінімум пропущених).

Залежно від бізнес-контексту балансуємо між цими двома метриками або шукаємо компроміс через **F1-score**, яке є гармонійним середнім Precision і Recall:

Гармонійне середнє (precision, recall, F1-score):

$$F1 = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{4.21}$$

4.3.2. Класифікація з урахуванням втрат

Мінімізувати витрати помилок класифікації можна за допомогою вибору точки відсікання (cut-off point) – порогового значення ймовірності, що розділяє класи. Піднімаючи поріг (наближаючи його до 1), зменшуємо ймовірність *хибнопозитивних* результатів і збільшуємо ймовірність *хибнонегативних*. Знижуючи поріг, навпаки, збільшуємо ймовірність *хибнопозитивних* результатів і зменшуємо ймовірність *хибнонегативних*. Якщо відомо, які з цих помилок обходяться дорожче, то можна підібрати поріг так, щоб мінімізувати пов’язані з ними витрати.

За допомогою матриці витрат і матриці класифікації можна обчислити витрати навченої моделі на цій тестовій множині. Для цього достатньо перемножити відповідні кількості помилково класифікованих спостережень, узяті з матриці класифікації, на значення пов’язаних із ними витрат із матриці витрат (рис. 4.16).



Рисунок 4.16 – Обчислення повних витрат помилок класифікації

Одним із широко використовуваних методів оцінювання оптимального порога є правило Байєса. Нехай позитивним результатом класифікації є клас 1. Тоді повні витрати C_{total} будуть:

$$C_{total} = (1 - P) \cdot C_{FP},$$

де P – імовірність віднесення спостереження до класу 1.

Якщо позитивним результатом класифікації є клас 0, то:

$$C_{total} = P \cdot C_{FN}.$$

Отже, оптимальне правило віднесе спостереження до класу 1, якщо:

$$(1 - P) \cdot C_{FP} < P \cdot C_{FN}.$$

В іншому разі спостереження буде віднесено до класу 0. Імовірність P оцінюють як відношення числа прикладів відповідного класу до загального числа прикладів.

На практиці правило Байєса використовується так: якщо ймовірність, визначена моделлю для *i*-го спостереження:

$$P_i > \frac{1}{1 + (C_{FN} / C_{FP})}, \qquad (4.22)$$

то спостереження належить до класу 1, в іншому разі – до класу 0.

Звернемо увагу на те, що правило Байєса залежить тільки від відношення витрат, а не від їхніх фактичних значень. Отже, якщо витрати обох типів помилок рівні, то правило Байєса дасть поріг відсікання 0,5. Можна показати, що коли відношення витрат дорівнює 1, то очікувані витрати пропорційні показнику помилки.

Якщо подія, пов’язана з позитивним результатом класифікації, є рідкісною, то витрати *хибнонегативних* помилок зазвичай більші, ніж *хибнопозитивних*. Наприклад, витрати від пропусків потенційного клієнта в директ-маркетингу більші, ніж витрати від розсилки тим, хто не реагує на неї. Витрати від проведення шахрайської транзакції за кредитною карткою більші, ніж витрати від невиконання законної транзакції (хоча завжди є винятки). Для таких випадків поріг відсікання має бути заданий істотно нижчим, ніж 0,5.

4.3.3. Lift, Profit, ROC криві

Lift криві

Наявність витрат класифікації робить актуальним завдання не тільки оцінювання ефективності моделі, а й оптимізації її з погляду витрат. Досить часто витрати класифікації точно не задано, а відомо тільки, який тип помилок призводить до більших витрат з огляду на постановку завдання. Аналітику нічого не лишається, крім розглянути всі можливі варіанти й вибрати оптимальний.

Розглянемо приклад з директ-маркетингом. Нехай фірма здійснює розсилку 1 мільйону потенційних клієнтів. Припустимо, що зазвичай відгукується 0,1 %, тобто 1 000 осіб. Отже, відгук можна розглядати як рідкісну подію. Якщо позитивний результат є рідкісною подією, то *хибнонегативні помилки* пов’язані з більшими витратами, ніж *хибнопозитивні*. Інакше кажучи, пропуск імовірного клієнта (*хибнонегативний результат)* призводить до більших втрат, ніж розсилка респонденту, від якого не надійшло відгуку (хибнопозитивна помилка). Логіка очевидна: один респондент, що відгукнувся, здатний принести прибуток, який покриє витрати від тисячі «порожніх» відправлень.

З погляду оптимізації витрат має сенс відібрати підмножину зі 100 тис. респондентів, у якій кількість відгуків більша, ніж в усій множині, наприклад 0,4 %, або 400. Якщо обмежитися розсилкою тільки цим респондентам, то витрати на розсилку стануть у 10 разів меншими, тоді як кількість відгуків зменшиться лише у 2,5 разу. Збільшення кількості відгуків щодо кількості поштових

відправлень відоме як *ліфт-фактор*, або *коефіцієнт ліфта*, або просто *ліфт*. У цьому випадку коефіцієнт ліфта дорівнює 4. Якщо відомі витрати, то можна визначити прибуток, отриманий щодо цього значення ліфта L.

Позначимо вихідне число розсилок *N* (у нашому прикладі N = 1 000 000), а число відгуків – *n* (n = 1 000). Також позначимо розмір вибірки, для якого кількість відгуків очікується більш значною, *K* (тобто K = 400 000), а кількість відгуків у вибірці *k* (k = 800). Тоді приріст частки позитивних результатів, тобто коефіцієнт ліфта, становитиме:

$$L = \frac{k/K}{n/N} = \frac{800/400\,000}{1000/1\,000\,000} = \frac{0{,}002}{0{,}001} = 2\,.$$

Розглянемо варіант із моделлю, що враховує розсилку 400 тис. респондентів, для якої відсоток відгуку – 0,2 %, або 800 осіб. Це відповідає коефіцієнту ліфта, що дорівнює 2. Що краще: розіслати рекламу 100 тис. респондентів і отримати 400 відгуків чи зробити розсилку 400 тис. респондентів і отримати 800 відгуків? Підвищивши витрати в 4 рази, отримуємо збільшення кількості відгуків лише вдвічі.

Узагальнимо методику розрахунку ліфта на довільну модель бінарної класифікації. Нехай є множина з *N*-прикладів, з яких *n* позитивних. Для кожного прикладу модель видає ймовірність його належності до певного класу. Тоді завдання аналітика полягає у виборі підмножини прикладів, які мають найбільшу ймовірність позитивного результату.

Для цього відсортуємо приклади в порядку зменшення ймовірності позитивного результату й витягнемо деяке число спостережень *K* з отриманого списку, починаючи зверху. Тоді, щоб розрахувати коефіцієнт ліфта, достатньо обчислити відношення частки позитивних прикладів у вибірці *k / K* до частки позитивних прикладів у вихідній множині *n / N*.

№ п/п	Передбачена ймовірність	Фактичний клас
1	0,95	Так
2	0,93	Так
3	0,93	Ні
4	0,88	Так
5	0,86	Так
6	0,85	Так
7	0,82	Так
8	0,80	Так
9	0,80	Ні
10	0,79	Так
11	0,77	Ні
12	0,76	Так
13	0,73	Так
14	0,65	Ні
15	0,63	Так

Рисунок 4.17 – Фрагмент набору даних, відсортованих за зменшенням імовірності позитивної класифікації

У таблиці (рис. 4.17) подано фрагмент набору даних, що містить N = 150 спостережень, з яких n = 50 є позитивними (~0,33). Спостереження відсортовано в порядку зменшення ймовірності позитивної класифікації, тобто класу *Так*. Модель правильно класифікувала спостереження 1 і 2: вони дійсно позитивні. Для спостереження 3, найімовірніше, було допущено помилку, оскільки воно виявилося негативним. Тепер, якщо хочемо знайти підмножину з K = 10 спостережень, які з найбільшою вірогідністю є позитивними, потрібно взяти 10 перших спостережень з упорядкованого списку. Вісім із них позитивні (k = 8), тому їхня частка для вибірки дорівнює 0,8. Тоді коефіцієнт ліфта становитиме L = 0,8 / 0,33 = 2,42.

Отже, щоб мінімізувати витрати або максимізувати вигоду, достатньо, змінюючи обсяг вибірки, підібрати таку кількість спостережень з упорядкованого списку, що забезпечить оптимальне співвідношення кількості розсилок (а відповідно, і витрат на неї) та кількості відгуків. Для розв’язання цього завдання може бути побудовано кілька моделей, після чого потрібно перевірити, яка з

них ефективніша. Щоб обрати модель, яка дає змогу знайти оптимальне співвідношення між витратами й прибутком, для кожної моделі-кандидата будують графік, який називають Lift-кривою (Lift curve). Сімейство Lift-кривих утворює Lift-діаграму (Lift chart). Поведінка Lift-кривих і їхнє взаємне розташування на діаграмі дають змогу зробити висновок про те, яка з моделей працює краще.

За горизонтальною віссю Lift-діаграми відкладають розмір вибірки у відсотках від загальної кількості спостережень у впорядкованому списку (у нашому випадку це одержувачі розсилки), а на вертикальній – кількість отриманих відгуків. Кожна точка Lift-кривої показує очікуване число відгуків для вибірки заданого розміру. Точка з координатами (0; 0) відповідає нульовому числу розсилок і, отже, відгуків. Точка з координатами (100; 1 000) відповідає максимальному числу спостережень (розсилок) із відгуком від 1 000 респондентів (рис. 4.18).

За допомогою Lift-кривої можна визначити розмір вибірки, який дасть найкраще співвідношення витрат (кількості розсилок) і прибутку (кількості отриманих відгуків).

На рис. 4.18 позначено два окремі випадки, описані вище: 10 % розсилок, які дають 400 відгуків, і 40 % розсилок, які дають 800 відгуків. На практиці потрібно обмежитися розміром вибірки, у межах якої Lift-крива має досить крутий підйом (у нашому випадку приблизно до 40 %). Тоді збільшення кількості розсилок і, відповідно, витрат призводить до зростання кількості відгуків.

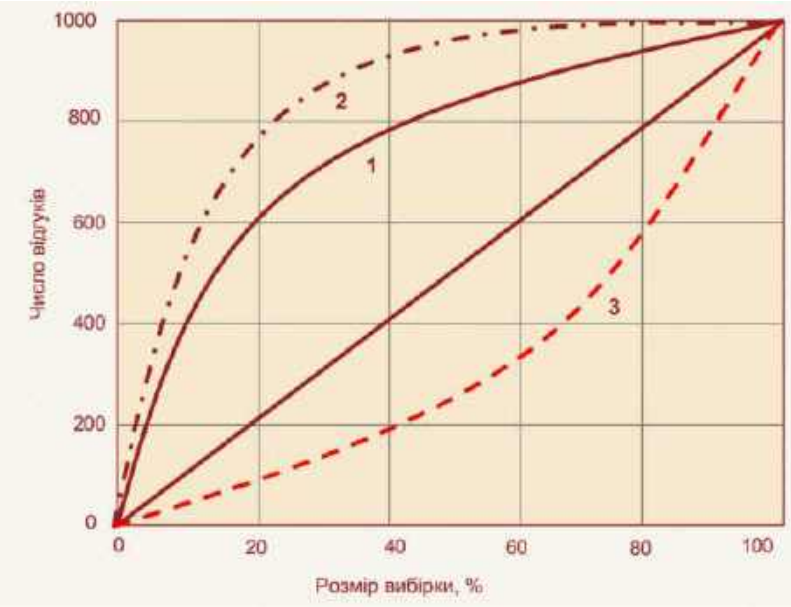


Рисунок 4.18 – Lift-криві

Так, зростання розсилок з 10 до 40 % дає змогу підняти відгуки вдвічі: з 400 до 800. Але подальше збільшення обсягу вибірки призводить до слабкого зростання кількості відгуків. Тому значення понад 40 % непрактичне: витрати зростають швидше, ніж можлива вигода. Зокрема, збільшивши кількість розсилок удвічі, з 40 до 80 %, отримаємо збільшення кількості відгуків приблизно на 10 %. Використання малих обсягів вибірки (менше 10 %) також не має сенсу. Хоча Lift-крива на цій ділянці має найбільшу крутизну, проте мале число розсилок навряд чи принесе хороший прибуток.

Діагональна лінія на Lift-діаграмі відображає роботу так званого довільного (випадкового) класифікатора, який нездатний виявити підмножини, де відсоток відгуків більший, ніж загалом щодо вибірки. Модель, якій відповідає діагональна лінія, марна.

Будь-яка модель, Lift-крива якої проходить вище за діагональну пряму, дає приріст числа відгуків стосовно випадкової моделі й забезпечує ліфт. Класифікатор, представлений на рисунку 4.18 кривою 3, що проходить нижче за діагональну пряму, дає від'ємний ліфт і буде гіршим за «марний», тому застосування таких моделей із погляду оптимізації витрат є безглуздим.

Якщо вище проходить Lift-крива, то ефективніша модель. За інших рівних умов перевагу варто віддати тій моделі, для якої Lift-крива проходить вище за інші або, що одне й те саме, максимально наближається до точки з координатами (0; 1 000).

Іноді за вертикальною віссю Lift-діаграми відкладають не кількість позитивних спостережень, а власне значення ліфта. Очевидно, що в цьому випадку Lift-крива спадатиме (рис. 4.19). Максимальний ліфт буде за обсягом вибірки, що дорівнює 1, оскільки для першого спостереження ймовірність позитивної класифікації найбільша.

Зі збільшенням обсягу вибірки в неї починає потрапляти дедалі більше негативних результатів, і ліфт зменшується. У межі, коли розмір вибірки становить 100 %, ліфт прагне до 1. Пряма L = 1, паралельна горизонтальній осі, аналогічна до діагональної лінії на рисунку 4.18 і демонструє «марну» модель.

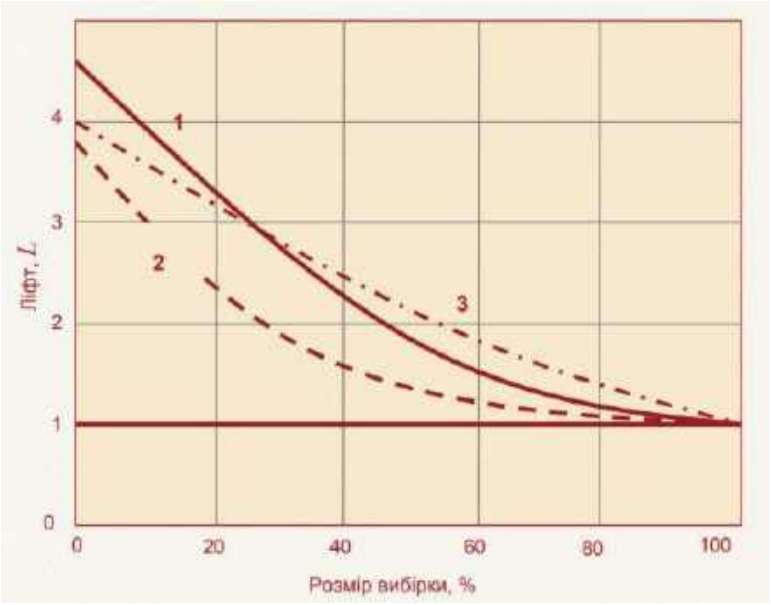


Рисунок 4.19 – Альтернативний спосіб побудови Lift-кривої

Різновид Lift діграм – Gain-діаграма

Ще один різновид *Lift-диграм* – Gain-діаграма (Gain chart), або кумулятивна Lift-діаграма (cumulative lift chart). За горизонтальною віссю Gain-діаграми також відкладається розмір вибірки, отриманої з упорядкованого списку. Але за вертикальною віссю фіксується не кількість (або відсоток) позитивних спостережень, що потрапили до вибірки, а відношення кількості істинно позитивних спостережень до числа всіх спостережень, класифікованих як позитивні, тобто TP / (TP + FN). Раніше зазначалося, що ця величина називається чутливістю (Se).

Зазначимо, що чутливість наближається до 0, якщо є досить значна кількість хибнонегативних спостережень. Верхня межа чутливості дорівнює 1 у разі, коли хибнонегативні спостереження відсутні (помилок немає). Методика використання Gain-діаграми загалом схожа до методики використання звичайної Lift-діаграми, але шукають оптимальне співвідношення між розміром вибірки та кількістю помилок класифікатора, тобто кількістю випадків хибнонегативної класифікації.

Збільшуючи розмір вибірки (зміщуючись по діаграмі праворуч), збільшуємо кількість хибнопозитивних спостережень, відповідно, ризик помилкової класифікації зростає. Зменшуючи вибірку (зміщуючись по діаграмі ліворуч), знижуємо кількість

помилок. Завдання аналітика полягає в підборі розміру вибірки так, щоб чутливість моделі була на досить високому рівні.

Якщо використовувати Gain-діаграму для порівняння моделей, то кращою також буде та з них, Gain-крива якої проходить вище. У діаграмі на рис. 4.20 перевагу потрібно віддати моделі 1, оскільки, наприклад, для вибірки розміром 20 % вона забезпечує чутливість приблизно 0,55, а модель 2 – лише 0,3.

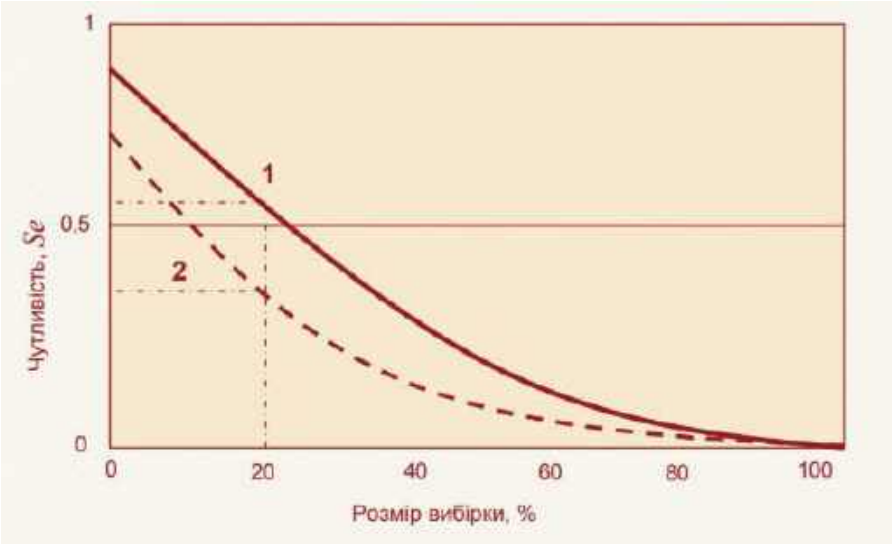


Рисунок 4.20 – Gain-крива

Якщо Gain-криві двох моделей перетинаються, це також означає, що для різних діапазонів ефективність моделей різна. Вибирати потрібно ту модель, крива якої проходить вище на більш актуальному діапазоні.

Взагалі використовувати Gain-криві для порівняння моделей зручніше, ніж звичайні Lift-криві, оскільки останні здебільшого застосовуються для оптимізації витрат помилок класифікації, а перші безпосередньо вказують на точність моделі.

Profit-криві

Lift-діаграми – досить зручний засіб порівняння моделей та їхньої оптимізації з погляду витрат класифікації. Однак вони не дають відповіді на важливе запитання: якими будуть витрати або вигоди від використання моделі? Зрозуміло, що для цього потрібні числові показники, пов’язані із розв’язуваним бізнес-завданням. Наприклад, у завданні кредитного скорингу варто знати, в які суми обійдуться необґрунтована відмова сумлінному позичальнику

й помилково виданий кредит неплатоспроможній особі. У завданні директ-маркетингу потрібна інформація про те, у скільки обходиться одне поштове відправлення з рекламою і який прибуток очікується від кожного потенційного клієнта.

Щоб точно дізнатися, прибуток чи збиток принесе використання певної моделі, недостатньо просто порівняти їх ефективність – треба використовувати числові значення витрат і прибутку. Для цих цілей використовується ще один графічний метод, званий Profit-кривою (Profit curve), або кривою доходу. Проілюструємо його на прикладі масової розсилки.

Уведемо такі позначення:

P – дохід;

R – виручка (від англ. *revenue* – прибуток, виручка);

C – витрати (від англ. *cost* – витрати);

NR – загальна кількість отриманих відгуків;

RR – виручка з одного відгуку (revenue per response);

CR – витрати одного відгуку (cost per response);

NM – число відправлень (number mailed);

CM – витрати на одне відправлення (cost per mailing);

PR – прибуток з кожного відгуку, що дорівнює різниці між витратами на продаж і прибутком від продажу, тобто PR = RR – CR.

Тоді дохід буде визначено так:

$$P = R - C = NR - RRR - (NR - CR + NM - CM).$$

Нехай рекламну розсилку здійснювали 10 тис. потенційних клієнтів (тобто NM = 10 000), з яких відгукнулися NR = 250. Припустимо, що виручка, отримана від кожного клієнта, який відгукнувся, становить RR = 100 од., витрати на обслуговування клієнта – CR = 60 од. а витрати на кожне відправлення – CM = 2 од.

Загальний виторг *R* визначається як добуток числа істинно позитивних результатів (TP = NR) і виторгу, отриманого від кожного клієнта, що відгукнувся, *RR*. Тоді повні витрати *C* дорівнюватимуть добутку витрат на одне відправлення й числа позитивних результатів, передбачених моделлю, тобто сумі істинно позитивних і хибнопозитивних результатів.

Отже, $P = R - C = (TP - 100) - (TP - 60 + (TP + FP) - 2) = (TP - 100) - TP - 60 + 2 - TP + 2 - FP = 42 - TP + 2 - FP$.

Уведемо в розгляд показник відгуку (response rate) як відсоток істинно позитивних результатів від загального числа позитивних результатів – (TP/(TP + FP)) – 100 %, інакше кажучи, частку відгуків від загального числа розсилок для кожної вибірки. Нехай перша вибірка становить 10 %, або 1 000 розсилок, а показник відгуку – 5 %. Це означає, що з 1 000 респондентів відгукнулися 50 (TP), а 950 розсилок (FP) залишилися без відповіді.

Тоді для першої вибірки прибуток становитиме 42 – 50 – 2 – 950 = 2 100 – 1 900 = 200. Отже, отримаємо від'ємний дохід. Проведемо аналогічні розрахунки для 10 вибірок зі зростанням кожної наступної на 10 % (табл. 4.10).

Таблиця 4.10 – Розрахунок прибутку

Обсяг вибірки, %	Обсяг вибірки, к-ть	Показник відгуку, %	Прибуток, од.
0	0	0	0
10	1000	5	42 - 50 - 2 - 950 = 200
20	2000	4,5	42 - 90 - 2 - 1100 = 1580
30	3000	4	42 - 120 - 2 - 2880 = -720
40	4000	3,5	42 - 140 - 2 - 3860 = -1840
50	5000	3	42 - 150 - 2 - 4850 = -3400
60	6000	2,5	42 - 150 - 2 - 5850 = -5400
70	7000	2	42 - 140 - 2 - 6860 = -7840
80	8000	1,5	42 - 120 - 2 - 7880 = -10 720
90	9000	1	42 - 90 - 2 - 8910 = -14 040
100	10 000	0,5	42 - 50 - 2 - 9500 = -16 900

Найбільший прибуток виходить з обсягу вибірки 20 %. Однак оскільки зі збільшенням кількості розсилок витрати зростають швидше, ніж виторг, під час подальшого збільшення розміру вибірки прибуток падає і навіть стає від'ємним. На основі цієї таблиці легко побудувати графік залежності прибутку від обсягу вибірки (рис. 4.21).

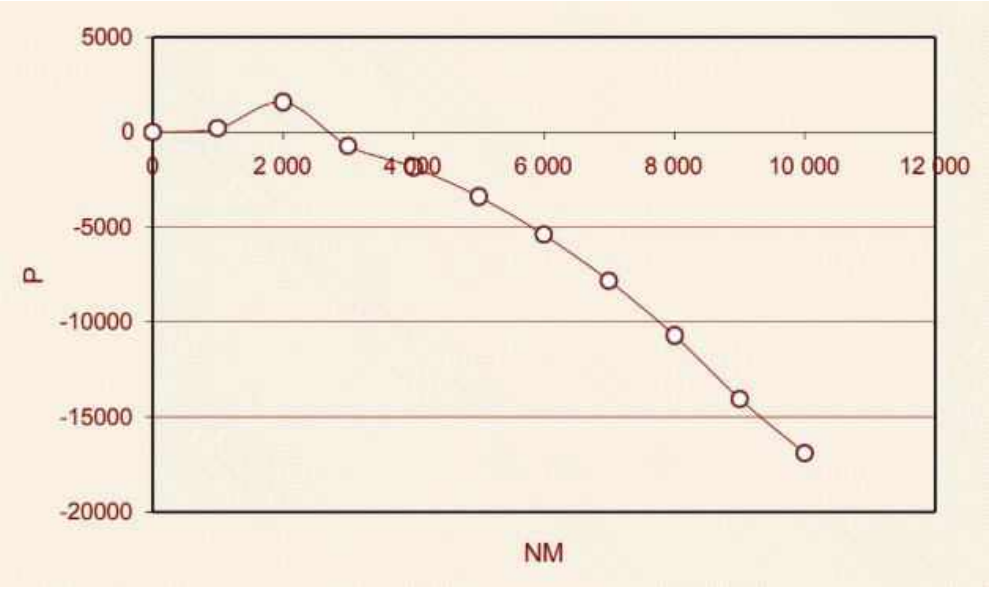


Рисунок 4.21 – Залежність маржі від обсягу вибірки

ROC аналіз

Крім Lift-кривих, які широко використовують у маркетингу, в кредитному скорингу та інших галузях, де застосовують бінарну ймовірнісну класифікацію, є ще одна графічна *методика оцінювання ефективності моделей*, яку називають *ROC-аналізом.*

Як зазначалося раніше, у задачах бінарної класифікації, коли модель передбачає ймовірність того, що спостереження належить до одного з двох класів, досить важливим є вибір точки відсікання, тобто порогу ймовірності, що розділяє два класи. Точка відсікання показує, після якого значення ймовірності на виході моделі один клас змінюється іншим. Наприклад, у *завданні кредитного скорингу* можемо встановити точку відсікання для позитивного результату 70 %. Якщо модель для деякого потенційного позичальника видає значення ймовірності, нижче за 0,7, то його визнають некредитоспроможним і відмовлять у видачі позики, а якщо вище за 0,7, то позичальника визнають кредитоспроможним і ухвалюватимуть позитивне рішення про видачу йому кредиту.

Обираючи точку відсікання, керуємо ймовірністю правильного розпізнавання позитивних і негативних прикладів. У разі зменшення порога відсікання збільшується ймовірність помилкового розпізнавання позитивних спостережень (хибнопозитивних результатів), а в разі збільшення зростає ймовірність неправильного розпізнавання негативних спостережень (хибнонегативних

результатів). Мета полягає в тому, щоб підібрати таке значення точки відсікання, яке дає найбільшу точність розпізнавання заданого класу, а якого саме – визначається постановкою задачі.

Саме для цього використовується ROC-аналіз. Методика була перенесена в Data Mining зі сфери обробки сигналів у радіолокації, а сама абревіатура ROC розшифровується як receiver operating characteristic, що в перекладі з англійської означає «робоча характеристика приймального пристрою». У радіотехнічному застосуванні завдання полягає у виявленні за допомогою приймального пристрою (радіолокатора) деякої цілі (морської, повітряної тощо). Водночас позитивний результат – ціль виявлено, негативний – ціль не виявлено. Нині ROC-аналіз широко застосовують в економіці та бізнесі, медицині та біології, наукових дослідженнях.

Перед розглядом методики введемо поняття *помилок I і II роду*. Якщо бінарну класифікаційну модель будують на основі навчальної вибірки, то всі приклади, які входять до неї, відповідають або позитивному, або негативному результатам. Тоді в процесі роботи моделі можуть виникнути такі помилки.

Помилки I роду

Приклад відповідає позитивному результату, але був розпізнаний як негативний: позичальник кредитоспроможний, але модель розпізнає його як некредитоспроможного; пацієнт хворий, але модель розпізнає його як здорового; радіолокаційна система не змогла виявити ціль за її сигналом, прийнятим локатором (пропуск цілі) тощо. Інакше кажучи, подію, що цікавить, помилково не виявлено.

Помилки II роду

Приклад відповідає негативному результату, але був розпізнаний як позитивний: позичальник некредитоспроможний, але модель визначає його як кредитоспроможного; пацієнт здоровий, але модель визначає його як хворого; радіолокаційна ціль відсутня, але система визначає її наявність (хибна тривога) тощо. Інакше кажучи, подія, що цікавить, присутня, але невиявлена.

Раніше зазначалося, що для оцінювання якості будь-якої бінарної класифікаційної моделі можна використовувати два показники – чутливість (Se) і специфічність (Sp).

Звернемо увагу: якщо відсутні хибнонегативні результати, чутливість дорівнює 1; якщо відсутні хибнопозитивні результати, специфічність дорівнює 1. Модель, яка здатна ідеально точно класифікувати як позитивні, так і негативні приклади, матиме 100-відсоткову чутливість і специфічність.

Щоб мінімізувати помилки I роду, потрібно використовувати модель з високою чутливістю. Щоб мінімізувати помилки II роду, треба використовувати модель із високою специфічністю.

Ідеальних моделей не буває, і на практиці доводиться шукати компроміс між чутливістю та специфічністю. Він досягається за рахунок вибору порогового значення ймовірності або точки відсікання. Як випливає з формули розрахунку чутливості, для її підвищення потрібно зменшити кількість хибнонегативних результатів, а це досягається через зниження порога.

Для збільшення специфічності потрібно знизити кількість хибнопозитивних результатів З цією метою збільшується поріг відсікання – так моделі буде складніше переплутати негативний результат із позитивним.

У загальному випадку завдання ROC-аналізу полягає в тому, щоб вибрати таку точку відсікання, яка забезпечить максимум чутливості та специфічності. Але чи завжди потрібна модель, яка однаково добре розпізнає як позитивні, так і негативні приклади? Якщо витрати обох типів помилок приблизно однакові, то така модель нас влаштує. Але на практиці схожих завдань мало. Частіше трапляються ситуації, в яких витрати одного типу помилок більші, ніж витрати іншого. Так, витрати помилкової тривоги набагато менші, ніж пропуску цілі. Тому поняття «оптимальний поріг» відносне: усе залежить від завдання.

Одним із перших практичних застосувань ROC-аналізу став *скринінг*, коли на основі деякого обмеженого набору медичних показників (вік, склад крові, рівень холестерину, температура, кров'яний тиск тощо) визначають наявність або відсутність захворювання в пацієнта. Завдання зводиться до бінарної класифікації, причому позитивним результатом найчастіше вибирається наявність захворювання, а негативним – його відсутність. Класифікаційна модель видає ймовірність віднесення кожного спостереження до одного з класів. Водночас вихід 1 вказує на 100-відсоткову

ймовірність наявності захворювання, а 0 – на достовірну його відсутність. У найпростішому випадку, якщо витрати помилок діагностики байдужі, потрібно вибрати таку точку відсікання, щоб максимізувати і чутливість, і специфічність. Але, на жаль, завдання медичної діагностики досить чутливі до витрат помилок класифікації.

Зробивши поріг відсікання занадто низьким, ризикуємо отримати так звану передіагностику, коли модель допускає багато хибнопозитивних помилок. Але ж перестраховка може виявитися небезпечною: лікування пацієнта від захворювання, якого не існує, буває пов'язане з необґрунтованим хірургічним втручанням, застосуванням болючих методів і ліків, що викликають побічні ефекти, не кажучи вже про фінансові витрати. Однак, зробивши поріг відсікання занадто високим, ризикуємо пропустити захворювання. Проте в скринінгу де-факто використовується правило, що зміщує діагностичне рішення в бік зниження ймовірності пропуску захворювання, тобто передіагностики. Зокрема, у багатьох діагностичних тестах потрібно, щоб чутливість була не менше 90 %: вчасно виявити хворобу важливіше за будь-які витрати

В основі ROC-аналізу лежать так звані ROC-криві. Щоб побудувати ROC-криву, потрібно змінювати поріг відсікання в інтервалі від 0 до 1 із заданим кроком Δx, наприклад 0,01. У результаті за кожного значення порога змінюватиметься кількість правильно й неправильно розпізнаних прикладів, відповідно, чутливість і специфічність моделі. Якщо розрахувати чутливість і специфічність для кожного значення порога, а потім побудувати графік, за вертикальною віссю якого відкладається чутливість, а за горизонтальною – величина $S = 100\ \% - Sp$, це й буде ROC-крива.

Типовий вигляд ROC-кривої наведено на рисунку 4.22. Збільшення значення по горизонтальній осі відповідає зменшенню специфічності моделі: точка $x = 0$ відповідає максимальній специфічності ($Sp = 100\ \%$), а точка $x = 100$ – мінімальній ($Sp = 0$).

Форми ROC- і Lift-кривих багато в чому схожі, так само як і методики їхнього аналізу. І це невипадково, оскільки принцип їхньої побудови приблизно один і той самий. Криві обох видів будують на основі відсікання: у разі Lift-кривих відсікання зазнає частина вибірки, оптимальна з позиції витрат помилкової класифікації, а у разі ROC-кривої відсікання зазнає ймовірність належності об'єкта до класу для забезпечення відповідного співвідношення Se і Sp.

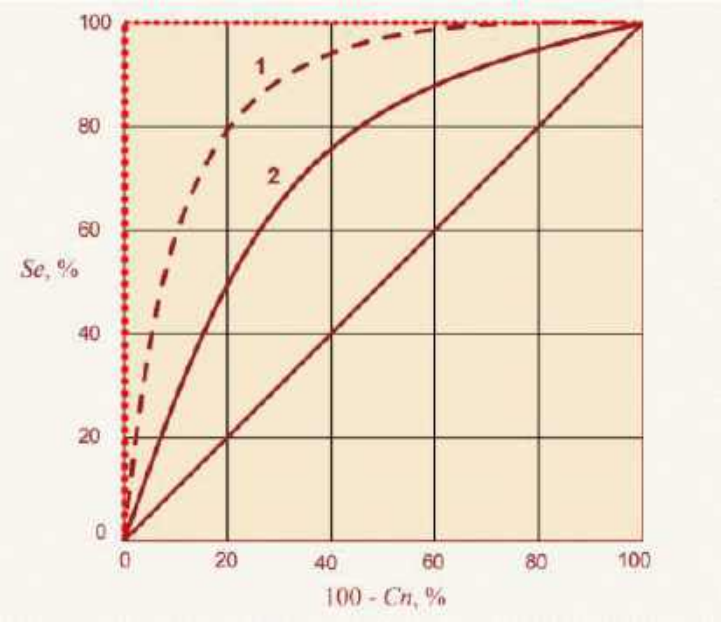


Рисунок 4.22 – ROC-криві до прикладу

Кожна точка ROC-кривої відповідає певному значенню порога відсікання. Водночас оптимальним буде значення, що відповідає точці ROC-кривої з координатами, максимально близькими до (0; 100), для якої і чутливість, і специфічність дорівнюють 100 %, тобто як позитивні, так і негативні приклади розпізнано правильно. Отже, якби вдалося побудувати модель, ROC-крива якої проходила б через точку з координатами (0; 100), отримали б ідеальний класифікатор (пунктирна лінія на рис. 4.22).

На практиці моделі припускаються помилок. Тому про якість бінарної класифікаційної моделі можна судити за ступенем кривизни ROC-кривої, тобто за тим, наскільки близько до точки, що відповідає ідеальному класифікатору, вона проходить. І тут, як і у випадку Lift-кривої, діагональна лінія відповідає «марному» класифікатору.

Площа під кривою (AUC) свідчить про прогностичну силу моделі, причому AUC = 1 відповідає ідеальному класифікатору. Що ж стосується реальних випадків, то, якщо площа під ROC-кривою більша за 0,7–0,8, відповідна модель демонструє досить високу точність.

За допомогою вибору порога відсікання домагаються оптимального співвідношення між чутливістю та специфічністю моделі. Для цього застосовують різні стратегії. Наприклад, часто поріг обирають так, щоб сума чутливості та специфічності була максимальною (тобто щоб забезпечувалася максимальна кількість як позитивних, так і негативних правильно розпізнаних прикладів).

Іноді поріг вибирають так, щоб дотримуватися балансу між чутливістю та специфічністю, тобто Se = Sp.

Чутливість і специфічність являють собою частки істинно позитивних і істинно негативних результатів класифікації. Зазвичай такі результати розподіляються нерівномірно.

У матриці класифікації на один позитивний результат може припадати кілька негативних, і навпаки. Тому в реальності лінія ROC-кривої буде не гладкою, а порізаною (рис. 4.23).

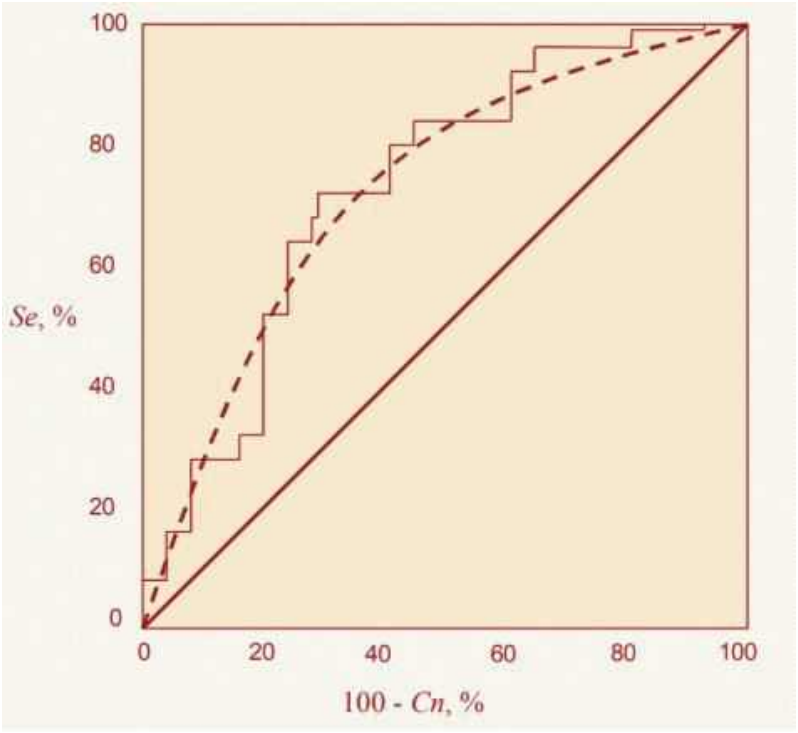


Рисунок 4.23 – Ламаний характер ROC-кривої

На рисунку видно, що з початку координат крива зростає на два позитивних спостереження й одне негативне, потім на п’ять позитивних і одне негативне тощо.

Викривленість ROC-кривої істотно залежить від особливостей вибірки даних. Цю залежність можна зменшити за рахунок перехресної перевірки через усереднення по блоках. Хоча зазвичай ідеального ступеня згладжування досягти не вдається.

4.3.4. Приклад побудови дерева рішень

Розглянемо приклад застосування методології дерев рішень з використанням бібліотек *sklearn, pandas, matplotlib.*

У прикладі будемо використовувати датасет *German Credit Data*, який сформовано одним із великих німецьких банків (дані деперсифіковані) й передано для відкритих досліджень професору Гансу Гофманну (Prof. Dr. Hans Hofmann) з Інституту статистики та економетрики Університету Гамбурга. Університет Стратклайда підготував числову версію (german.data-numeric) для алгоритмів, що не працюють з категоріальними даними.

Опис даних набору German Credit Data

Мета створення датасету: дослідження методів класифікації позичальників на «good» (надійні) та «bad» (ризикові) за рівнем кредитних ризиків із вбудованою матрицею вартостей помилок (Cost Matrix), де помилка «віднести поганого до хороших» дорожча, ніж навпаки (таблиця 0).

Матриця вартостей помилок датасету

	прогноз «good»	прогноз «bad»
реально good	0	1
реально bad	5	0

Неправильна позитивна помилка («bad» → «good») коштує в 5 разів дорожче, ніж неправильна негативна («good» → «bad»).

Розмір датасету: 1 000 спостережень, 20 ознак, без пропущених значень. Серед основних ознак такі: *Duration in month* (тривалість кредиту, місяці); *Credit history* – кредитна історія;

Purpose (мета кредиту: нове /б/в авто, меблі, радіо / телевізор, освіта тощо); *Credit amount* (сума кредиту); *Savings account / bonds* (баланс заощаджень) тощо (всього 20 ознак).

Усі атрибути закодовані як категоріальні або цілі числа відповідно до специфікацій StatLog. Для побудови моделей часто використовують обидва файли – оригінальний (*german.data*) та числовий (*german.data-numeric*) або завантажують через OpenML-функцію *fetch_openml('credit-g').*

Датасет береться із репозиторію OpenML.

У лістингу 4.16 показані потрібні компоненти, які завантажують для побудови й відображення дерева рішень.

```
                    # Лістинг 4.16
    import pandas as pd
    from sklearn.datasets import fetch_openml
    from sklearn.model_selection import
train_test_split
    from sklearn.tree import DecisionTreeClassifier,
plot_tree
    from sklearn.metrics import
classification_report, accuracy_score
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import
OneHotEncoder,StandardScaler
    from sklearn.compose import ColumnTransformer
    from sklearn.pipeline import Pipeline
```

Завантаження датасету, розподіл на *X* і *y* та виокремлення числових і категорійних ознак (лістинг 4.17):

```
                    # Лістинг 4.17
    Завантаження і обробка датасету
    #1. Завантажуємо дані з OpenML
    credit = fetch_openml('credit-g', version=1,
as_frame=True)
    df = credit.frame.copy()
    # У цьому датасеті цільова змінна 'class'
зі значеннями 'good'/'bad'
    df.rename(columns={'class': 'target'},
inplace=True)
```

```
    # 2. Розділяємо на X та y
    X = df.drop(columns='target')
    y = df['target']

    # 3. Виокремлюємо числові й категорійні ознаки
    num_feats = X.select_dtypes(include=['int64',
'float64']).columns.tolist()
    cat_feats = X.select_dtypes(include=['category',
'object']).columns.tolist()
```

Далі створюємо *препроцесінг-конвеєр* за допомогою *ColumnTransformer*-механізму, який дає змогу застосувати різні трансформації *паралельно*, до різних підмножин ознак в df, (лістинг 4.18).

```
                    # Лістинг 4.18
    Створюємо препроцесінг-конвеєр
    preprocessor = ColumnTransformer([
        ('num', StandardScaler(), num_feats),
        ('cat', OneHotEncoder(sparse_output=False,
handle_unknown='ignore'), cat_feats),
    ])
```

Пояснення до лістингу 4.18.

StandardScaler – трансформер для числових ознак *'num', OneHotEncoder* – для категоріальних ознак. *StandardScaler* обчислює середнє μ і стандартне відхилення (СКВ) σ по кожній числовій змінній на тренувальній вибірці, перетворює значення x у z = (x – μ) / σ. Це дає змогу зробити всі числові ознаки порівнянними за масштабом (*zero mean, unit variance*). *num_feats* список назв числових стовпців, до яких застосовуємо *StandardScaler* .

Далі створюємо *Pipeline* – структуру, що об'єднує послідовні кроки обробки й моделювання (лістинг 4.19).

```
                    # Лістинг 4.19
    Створюємо Pipeline
    pipeline = Pipeline([
        ('prep', preprocessor),
        ('clf', DecisionTreeClassifier(
            criterion='entropy',
            max_depth=5,
            random_state=42
        )),
    ])
```

Пояснення до лістингу 4.19.

'prep' застосовує preprocessor до вхідних даних. Під час виклику pipeline.fit(X_train, y_train) відбудеться:

X_prepared = preprocessor.fit_transform(X_train)

А *X_prepared* – це вже числовий масив, готовий для моделі.

'clf' – DecisionTreeClassifier(...), сама модель.

Параметри моделі дерева рішень з алгоритмом ID3 і розрахунком приросту інформації:

criterion='entropy': алгоритм обчислення приросту інформації як критерій розщеплення за методом ID3;

max_depth=5: не дає змогу дереву вирости глибшим за 5 рівнів (обмежує складність);

random_state=42: фіксує випадковість алгоритму (щоб результати були відтворювані).

Для застосування моделі викликається *pipeline* і обчислюється матриця втрат класифікації лістинг 4.20 (повна версія коду надана в блокноті ЛР у курсі на):

```
                    # Лістинг 4.20
    Навчання моделі
    pipeline.fit(X_train, y_train)
    # 8. Оцінка на тесті
    y_pred = pipeline.predict(X_test)
    print("Accuracy:", accuracy_score(y_test, y_pred))
    print(classification_report(y_test, y_pred,
target_names=['bad', 'good']))
```

Результат:

```
Accuracy: 0.7133333333333334
              precision    recall  f1-score   support

         bad       0.53      0.38      0.44        90
        good       0.76      0.86      0.81       210

    accuracy                           0.71       300
   macro avg       0.65      0.62      0.62       300
weighted avg       0.69      0.71      0.70       300
```

Пояснення результатів.

Accuracy 0.7133. Модель правильно класифікувала 71.33 % всіх випадків у тесті (з 300 спостережень правильними виявилися приблизно 214).

Support. Support у звіті *classification_report* з *sklearn* параметр *support* означає кількість (або розмір підмножини) реальних прикладів кожного класу в тестовій вибірці. У нашому випадку це означає, що в тесті з 300 записів:

90 записів належать до класу «bad» – ненадійні клієнти;

210 записів – до класу «good» – надійні клієнти.

Precision (точність, позитивна прогностична цінність):

bad (0.53). Модель передбачила «bad» певну кількість разів; лише 53 % з цих передбачень дійсно були «bad».

good (0.76). З усіх прогнозів «good» 76 % виявилися правильними.

Recall (чутливість, повнота):

bad (0.38). З 90 справжніх «bad» модель правильно ідентифікувала тільки 38 % (≈34 випадки), а решту віднесла помилково до «good».

good (0.86). З 210 справжніх «good» 86 % (≈181 випадок) модель класифікувала правильно, решту 14 % «good» вона пропустила (віднесла до «bad»).

F1-score (гармонійне середнє precision і recall):

bad (0.44). Баланс між 0.53 та 0.38 дає 0.44.

good (0.81). Для «good»: 0.76 та 0.86 → F1 ≈ 0.81.

Macro avg (макроусереднені метрики):

precision = (0.53 + 0.76) / 2 ≈ 0.65;

recall = (0.38 + 0.86) / 2 ≈ 0.62;

F1 = (0.44 + 0.81) / 2 ≈ 0.62.

Макроусереднення дає рівну вагу кожному класу незалежно від того, скільки прикладів у кожному.

Weighted avg (зважені метрики):

У цьому випадку кожен клас важить пропорційно до свого числа прикладів (support):

precision = (0.53 × 90 + 0.76 × 210) / 300 ≈ 0.69;

recall = (0.38 × 90 + 0.86 × 210) / 300 ≈ 0.71;

F1 = (0.44 × 90 + 0.81 × 210) / 300 ≈ 0.70.

Weighted avg відображає загальну якість моделі, враховуючи розподіл класів у тесті.

Що це означає для бізнесу?

Модель *краще виявляє «good»* – надійних клієнтів (високий *recall* для *good*), але часто пропускає «bad» – ненадійних клієнтів (низьке значення *recall* для *bad*).

Precision для *bad* (0.53) означає, що якщо модель попереджує, що клієнт «ненадійний», тільки в половині випадків це дійсно так.

Якщо для вас критично не пропустити «bad» клієнта («позичання коштів ризиковим»), варто покращити *recall* для *bad* (перемістити поріг класифікації чи використати інший алгоритм / зважування класів).

Загалом ці метрики допомагають зрозуміти, наскільки добре модель розрізняє «good» надійних та «bad» ненадійних позичальників, і де саме потрібні поліпшення залежно від бізнес-цілей.

Проаналізуємо два рівні отриманого дерева рішень (повна версія має 5 рівнів і подана в блокноті Colab до самостійної роботи) (рис. 4.24).

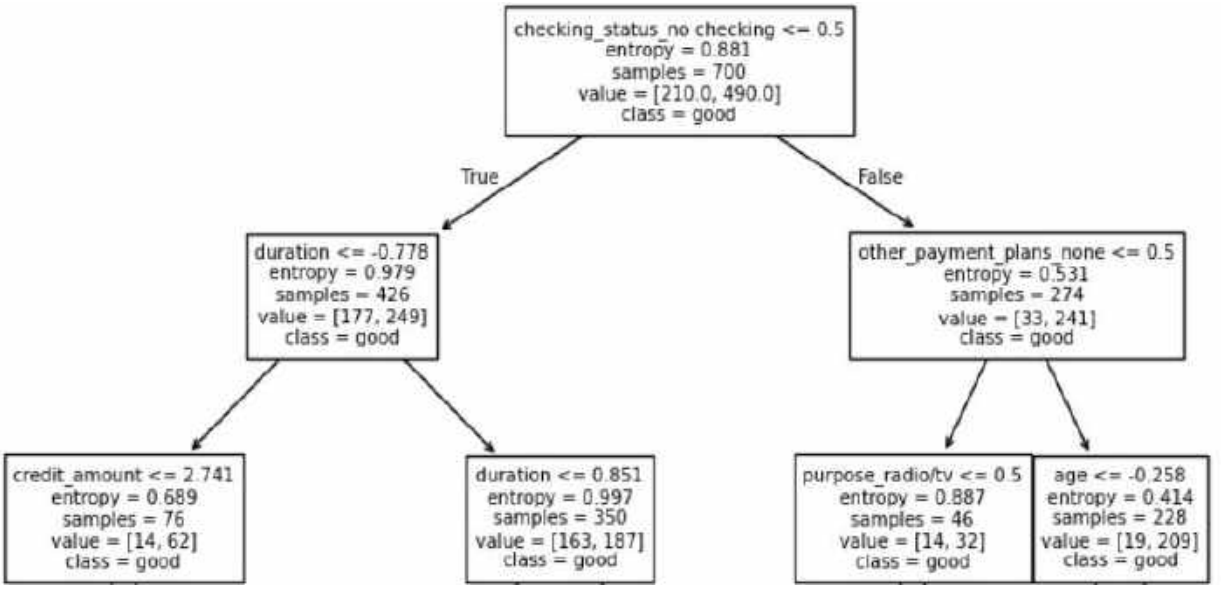


Рисунок 4.24 – Два рівні отриманого дерева рішень

Кореневий вузол (Root). Як бачимо, алгоритм на основі обчислення приросту інлормації (зменшення ентропії) вибрав для першого поділу *ознаку checking_status_no checking*, яка показує

наявність у клієнта поточного рахунку. *Умова: <= 0.5* означає «має поточний рахунок» (оскільки в бінарному one-hot кодуванні 1 = «no checking», 0 = «has checking»). У датасеті German Credit Data змінна *checking_status* описує *наявність і баланс поточного (checking) рахунку* клієнта в банку, на який можуть надходити зарплата або відкладатися гроші для щоденних операцій (це не кредитний чи депозитний рахунок). Навіть відсутність поточного (checking) рахунку в банку само собою не є «недоліком» даних, навпаки, це – досить інформативна ознака, оскільки:

нестабільність і непередбачуваність доходу

– банківський рахунок, особливо з регулярним надходженням (зарплатний проєкт), часто є ознакою стабільного працевлаштування та формальних доходів;

– якщо такого рахунку немає, не можемо бачити «пасивного тесту» на платоспроможність – типовий сигнальний маркер про ризиковість;

відсутність кредитної історії в цьому банку

– рахунок дає банку змогу відслідковувати рух коштів і своєчасність платежів (депозитні вклади, списання за кредитними та платіжними картами);

– без цієї інформації банк «сліпий» у частині поведінки клієнта, тож покладається на менш точні показники (соціальні дані, довідки тощо);

психологічний і бізнес-сигнал

– клієнти без рахунку можуть не бути лояльними до банку: вони не відкриті до крос-продажу (депозити, картки, онлайн-послуги);

– збільшується ймовірність, що під час виникнення прострочення вони з’являться в системі із запізненням або не зможуть обслуговувати борг через відсутність банківського інтерфейсу.

Отже, ознака *checking_status_no checking* підкреслює важливий ризиковий профіль клієнта:

а) відсутність формальної історії руху грошей;

б) можливі готівкові доходи;

в) низька фінансова інклюзія.

Саме тому для алгоритму моделі це виявилося одним із найсильніших предикторів: ті, хто *має* рахунок, зазвичай простіше обслуговують борг, а хто його *не має* – потрапляють у групу підвищеного кредитного ризику (тому дерево спочатку «розсікає» вибірку саме за цим критерієм).

entropy = 0.881: порівняно висока невизначеність даних датасету (якби пропорції класів були приблизно 50/50, то значення *entropy* наближалося б до 1. У цій вибірці «good» трохи більше).

samples = 700: у тренувальній вибірці 700 записів.

value = [210, 490]: із них 210 «bad» та 490 «good».

class = good: найбільша частка – «good».

Інтерпретація: алгоритм спочатку ділить дерево позичальників за наявністю поточного рахунку на дві групи і, відповідно, на дві гілки для подальшого аналізу:

ліва гілка (True): має рахунок *(checking_status_no checking <= 0.5)*;

права гілка (False): не має рахунку *(checking_status_no checking > 0.5)*.

Ліва гілка (клієнт має рахунок)

Ознака: *duration* (тривалість кредиту), але стандартизована, тому 0 відповідає середньому, а –0.778 означає коротші за середню тривалість.

Умова True: коротші кредити (≤ середнє – 0.778σ).

Тут співвідношення «good» / «bad» близьке до 50 / 50 → висока ентропія 0.979.

Підвузол «True → True»

credit_amount стандартизований; *<= 2.741* – це середньо малі та середні позики. Серед цих 76 клієнтів більшість (62) – «good» → ентропія знизилась, лідирує клас «good».

Підвузол «True → False»

Ці клієнти мають тривалість кредиту між двома межами (−0.778σ … +0.851σ). Значна невизначеність (entropy ≈1) – класи майже порівну, але «good» трохи переважають.

Права гілка (клієнт не має рахунку в банку)

other_payment_plans_none – чи немає інших планів погашення (0 = «none», 1 = «has one»).

True (≤ 0.5): не має планів → серед таких 241 «good» проти 33 «bad» → низька невизначеність.

Підвузол «False → True» (інші плани є)

Сюди потрапили клієнти, які мають інші плани і взяв кредит не на радіо / телевізор (≤ 0.5).

«good» усе ще переважають, але ентропія зросла.

Підвузол «False → False»

Це клієнти, які мають інші плани та кредити на радіо / ТВ, а також молодші за середній вік (–0.258σ). Ентропія 0.414 – сильний перекіс на користь «good» (209 проти 19).

Загальні висновки аналізу двох рівнів дерева рішень

Наявність рахунку – найважливіший предиктор, вибраний алгоритмом: клієнти, хто не має рахунку й використовує додаткові платіжні плани, досить рідко виявляються *«bad»*.

Тривалість і сума кредиту значущі для аналізу клієнтів, хто має рахунок: короткі й невеликі кредити – більш надійні клієнти.

Вік і мета кредиту (*радіо / ТВ*) – другорядні, але теж впливають у сегменті без рахунку та з іншими планами.

Ентропія в листкових вузлах показує ступінь чистоти:

Найнижча (≈0.414) – вузол із найнадійнішими позичальниками.

Найвища (≈0.997 – 0.979) – у вузлах, де класи майже порівну.

Отримані правила можна використати для швидкої сегментації клієнтів і оцінки ризику: дерево показує чітку ієрархію чинників від найбільш вагомих (рахунок у банку) до менш значущих (мета кредиту, вік).

Приклад аналізу частини нижній двох рівнів гілок і листків дерева

Із загального дерева візьмемо для прикладу останні два рівні, які завершуються листками дерева (рис. 4.25).

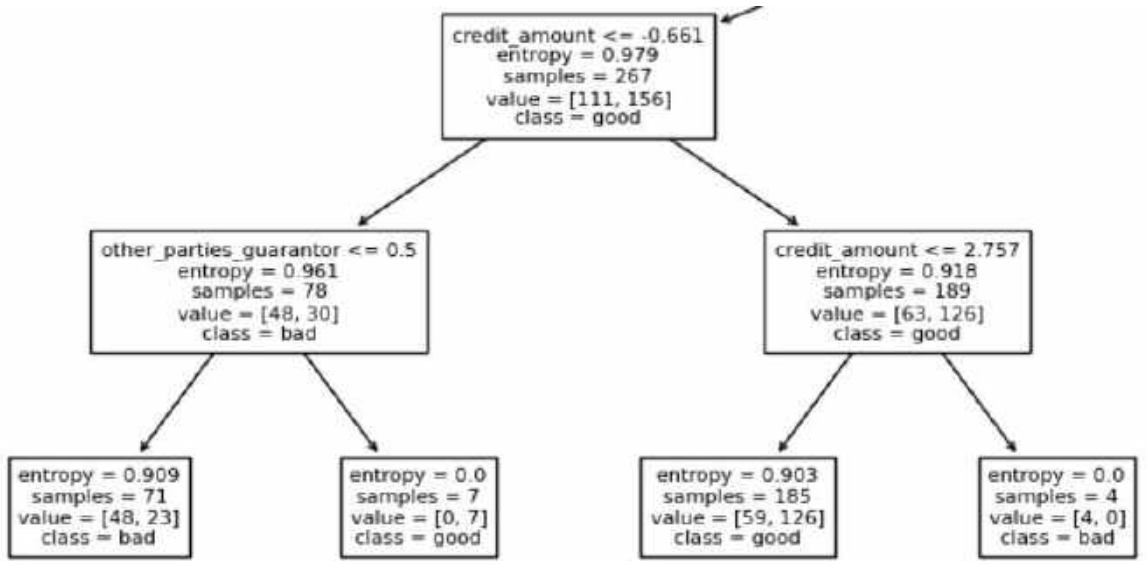


Рисунок 4.25 – Нижні два рівні гілок
і листків отриманого дерева рішень

Пояснення до дерева

Найважливіша ознака – сума кредиту (*credit_amount*). Тут значення *credit_amount* стандартизоване (0 = середнє, σ = 1), тому поріг 0.661 означає *порівняно невеликі позики.*

Ліва гілка – потрапляють клієнти з кредитами, меншими за поріг.

Права гілка – із середніми та великими сумами.

Ліва гілка (невеликі позики). Ознака *other_parties_guarantor* показує, чи є поручитель («1» = є, «0» = немає). Якщо поручителя *немає* (≤ 0.5), то серед невеликих кредитів значна частка – *bad* (48 із 78). Якщо поручитель є, навіть невеликий кредит стає «безпечним» (усі 7 записів тут – good).

Правило 1: якщо сума кредиту невелика **і** у клієнта *немає поручителя*, **то** високий ризик «bad».

Правило 2: якщо сума кредиту невелика **та** поручитель є, **то** майже гарантовано «good».

Права гілка (середні й великі позики). Щодо *середніх* (під поріг 2.757σ): переважна більшість клієнтів – *good* (126 із 189). Надто *великі* кредити (> 2.757σ) – усі 4 клієнти в тестовій частині виявилися «bad».

Правило 3: якщо кредит помірний (не надто великий), **то** домінує клас «good».

Правило 4: якщо кредит досить великий, то практично завжди «bad».

Загальні висновки й правила дерева рішень

Credit amount – найсильніший показник ризику.

Поручитель може «врятувати» клієнта з невеликою сумою, відносячи його до хороших – без поручителя навіть малі кредити ризикові.

Досить великі кредити без вагомих додаткових гарантій призводять майже гарантовано до дефолту.

Модель чітко розділяє клієнтів на три сегменти:

Якщо *невеликі кредити* + *поручитель* → надійні.

Якщо *невеликі кредити без поручителя* → ризикові, ненадійні.

Якщо *середні кредити* → більшість надійні.

Якщо *великі кредити* → завжди ризикові, ненадійні.

Нижче наведена ROC крива, яка показує, як змінюються дві ключові метрики класифікатора під час зміни порогу рішення. ROC крива отримана на основі результатів класифікації (лістинг отримання ROC кривої наведений у блокнотоі до ЛР).

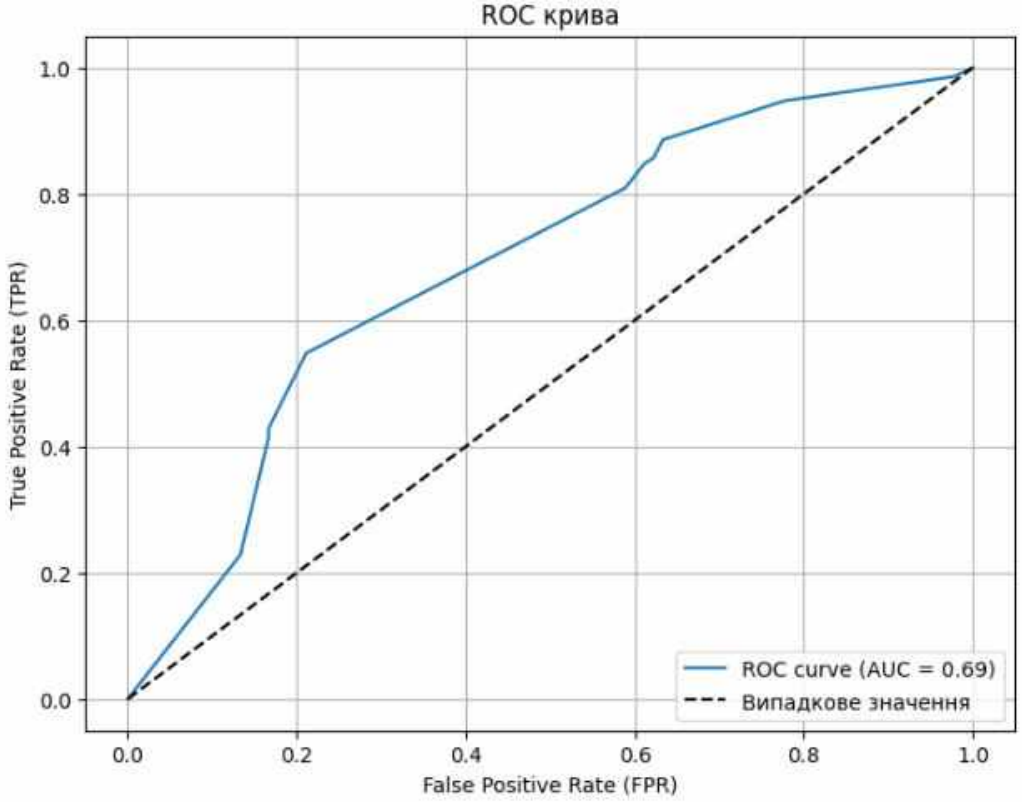


Рисунок 4.26 – ROC крива до прикладу аналізу дерева рішень

Пояснення ROC кривої

False Positive Rate (FPR) по осі X, тобто частка «помилкових тривог» серед усіх реальних негативних прикладів: *FPR = FP / (FP + TN)*.

True Positive Rate (TPR) по осі Y (те саме *Recall* або *Sensitivity*), тобто частка правильно виявлених позитивних прикладів серед усіх реальних позитивних: *TPR = TP / (TP + FN)*.

Кожна точка на синій лінії відповідає певному порогу θ для ймовірнісного прогнозу P(good) > θ:

низький поріг: модель класифікує багато прикладів як «позитивні»: і TPR, і FPR високі.

високий поріг: модель досить консервативна в позитивних прогнозах: і TPR, і FPR низькі.

Ідеальне дерево мало б ROC-криву, що йде по лівій грані графіка й зверху: низький FPR під час високого TPR (ловить майже всі «плюс»).

Діагональна штрихова лінія («Random») – це «крива випадкової моделі»: у разі випадкового передбачення точка лежить на TPR = FPR. Ваша модель повинна бути вищою за цю лінію, щоб показувати більше користі, ніж чиста випадковість.

AUC (Area Under the Curve). AUC = 0.69 на графіку означає, що в усіх можливих парах одного «good» і одного «bad» модель правильно розставить їх у правильному порядку з імовірністю 0.69 (69 %).

AUC = 0.5 відповідає випадковій моделі, AUC = 1.0 – ідеальній.

Основні висновки

AUC ≈ 0.69. Модель помітно краща за випадкову, але є простір для поліпшення (оптимальні значення – вище 0.8).

Форма кривої. Зліва внизу крива йде стрімко вгору (це добре: низький FPR у разі помірного TPR). Потім піднімається плавніше (відбувається «цінова» компромісна зона: щоб підняти TPR далі, доводиться відпускати більше FP).

Вибір порогу. Якщо важливі *мінімальні пропущені «good»* (максимальна чутливість), виберіть нижчий поріг у лівій частині кривої, навіть якщо FPR трохи зросте.

Якщо критично *уникати false alarms*, оберіть поріг біля «коліна» кривої – точка, де TPR починає зростати повільніше стосовно FPR.

Отже, ROC-крива дає візуальний і кількісний (AUC) інструмент для оцінювання як загальної роздільної здатності моделі, так і для вибору оптимального компромісу між чутливістю та специфічністю.

Питання для самоконтролю

1. Що таке матриця помилок (*confusion matrix*)? Назвіть і поясніть її чотири основні компоненти (TP, FP, TN, FN).

2. Поясніть різницю між точністю (*Accuracy)* та влучністю (*Precision*). В якій ситуації низька влучність може бути критичною проблемою, навіть у разі високої загальної точності?

3. Що таке повнота (*Recall* або *Sensitivity*)? Наведіть приклад задачі (наприклад, у медицині чи фінансах), де повнота є важливішою за влучність.

4. Дайте визначення F1-міри. Чому вона часто є кращим показником якості для незбалансованих наборів даних, ніж точність (*Accuracy*)?

5. Поясніть, що таке вартість помилок (*cost of misclassification*). Чому вартість *False Positive* та *False Negative* помилок рідко буває однаковою в реальних бізнес-завданнях?

6. Як матриця втрат (*cost matrix*) використовується для модифікації процесу прийняття рішень моделлю? Наведіть приклад.

7. Що показує ROC-крива? Поясніть, що означають осі цієї кривої (*True Positive Rate ma False Positive Rate*).

8. Що таке показник AUC (*Area Under the Curve*)? Яке значення AUC вказує на ідеальну модель, а яке – на модель, що не краща за випадкове вгадування?

9. Опишіть призначення *Lift Chart* (кривої підйому). Як вона допомагає оцінити, наскільки модель краща за відсутність будь-якої моделі (наприклад, під час вибору клієнтів для маркетингової кампанії)?

10. Як *Profit Curve* (крива прибутку) відрізняється від *Lift Chart*? Які додаткові дані потрібні для її побудови?

11. Уявіть, що ви порівнюєте дві моделі. Модель А має AUC = 0.85, а модель Б – AUC = 0.79. Чи завжди це означає, що модель А буде більш прибутковою? Поясніть свою відповідь.

4.4. Кластерний аналіз

4.4.1. Сутність методів кластерного аналізу

Кластерний аналіз (*cluster analysis*) – це групування об'єктів у кластери так, щоб об'єкти в межах одного кластеру були максимально схожими, а об'єкти з різних кластерів – якомога відмінними. Однією із широко використовуваних методик є розділова кластеризація (*partitioning clustering*), відповідно до якої для вибірки даних, що містить *n*-записів (об'єктів), задають число кластерів *k*, яке має бути сформовано. Потім алгоритм розбиває всі об'єкти вибірки на *k*-групи (k < n), які являють собою кластери.

Основні підходи

1. Ієрархічні методи (Hierarchical Clustering):

– агломеративні (знизу вверх): починають з кожного об'єкта як окремого кластера й поступово об'єднують їх за певним критерієм зв'язку;

– дивізійні (згори вниз): починають з єдиного великого кластера й ділять його.

2. Частотні методи (Partitioning):

– k-means: шукає k центрів, мінімізуючи суму квадратів відстаней внутрішньокластерних точок;

– k-medoids, PAM: схожі на k-means, але центром кластера є реальний об'єкт (медоїд).

3. Густинні методи (Density-based):

– DBSCAN: виділяє кластери як сфери з високою щільністю точок, викиди – точки низької щільності;

– OPTICS: розширює DBSCAN для різних щільностей.

Основні метрики відстані в кластерному аналізі

Правило, за яким обчислюється відстань у багатовимірному просторі ознак, називається метрикою. Найчастіше в практичних завданнях кластеризації використовують такі метрики.

Евклідова відстань

Евклідова відстань, або метрика L2 використовує для обчислення відстаней таке правило (4.23):

$$d_E(x,y)=\sqrt{\sum_i (x_i-y_i)^2}\quad , \qquad (4.23)$$

де:

$X = (x_1, x_2 ,..., x_m)$, $Y = (y_1, y_2 ,..., y_m)$ – вектори значень ознак двох записів.

Оскільки множина точок, рівновіддалених від деякого центру, щодо використання евклідової метрики утворюватиме сферу (або коло в двовимірному випадку), кластери, отримані з використанням евклідової відстані, також матимуть форму, близьку до сферичної.

Манхеттенська відстань

Манхеттенська відстань (Manhattan distance), або метрика L1, обчислюється за формулою (4.24):

$$d_M(\mathbf{x},\mathbf{y})=\sum_i |x_i-y_i|. \qquad (4.24)$$

Фактично це найкоротша відстань між двома точками, пройдена лініями, паралельними осям прямокутної системи координат (рис. 4.27).

Перевага метрики L1 полягає в тому, що її використання дає змогу знизити вплив аномальних значень на роботу алгоритмів. Кластери, побудовані на основі відстані Манхеттена, прагнуть до кубічної форми. Іноді відстань Манхеттена називають «відстанню міських кварталів» (city-block distance) через асоціації, що виникають із прямокутними формами забудови, яка характерна для сучасних міст.

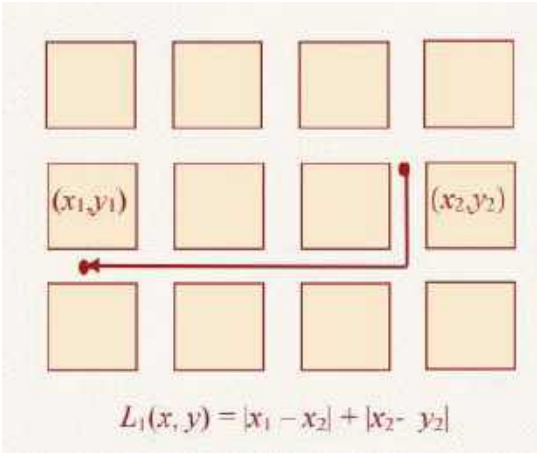


Рисунок 4.27 – Графічне подання Манхеттенської відстані (метрика L1)

Для категоріальних ознак як міру відстані можна використовувати *функцію відмінності (different function)*, яку задають так:

$$\boldsymbol{d(x_i) = f(x) = \begin{cases} 0, & \text{якщо } x_i = y \\ 1, & \text{в інших випадках} \end{cases}}, \qquad (4.25)$$

де: x_i і y_i – категоріальні значення.

4.4.2. Алгоритм кластеризації k-means

До найпростіших і найефективніших алгоритмів кластеризації належить k-means (Mac-Queen, 1967), або в українському варіанті *k-середніх* (від англ. *mean* – середнє значення). Ключовий момент *алгоритму k-means*: на кожній ітерації обчислюють відстань між об'єктами й центрами кластерів, щоб визначити, до якого з кластерів належить цей запис.

Він складається з чотирьох кроків:

1. Задано число кластерів *k*, яке має бути сформовано з об'єктів вихідної вибірки.
2. Випадково обирається *k*-записів, які слугуватимуть початковими центрами кластерів. Початкові точки, з яких потім виростає кластер, часто називають «насінням» (від англ. *seeds* – насіння, посіви). Кожен такий запис являє собою своєрідний «ембріон» кластера, що складається тільки з одного елемента.
3. Для кожного запису вихідної вибірки визначається найближчий до нього центр кластера.
4. Проводиться обчислення центроїдів – центрів ваги кластерів. Це робиться через визначення середнього для значень кожної ознаки всіх записів у кластері.

Для кращого розуміння методики скористаємося геометричною інтерпретацією визначення того, до якого з центрів кластерів найближче розташований певний запис. Межею між двома кластерами є точка, рівновіддалена від чотирьох центрів. З геометрії відомо, що множина точок, рівновіддалених від двох точок A і B, утворює пряму, яка перпендикулярна до відрізка AB і перетинає його середину. Принцип пошуку меж майбутніх кластерів пояснюється на рис. 4.28. Для простоти розглядається двовимірний

випадок, коли простір ознак містить лише два виміри (X_1, X_2). Нехай на першому кроці було відібрано три точки A, B і C. Тоді лінія 1, що проходить перпендикулярно відрізку AB через його середину, складатиметься з точок, рівновіддалених від точок A і B. Аналогічно будується пряма 2 для точок A і C, а також пряма 3 для точок B і C. За допомогою цих ліній можна визначити, до якого із центрів виявляється ближчою певна точка.

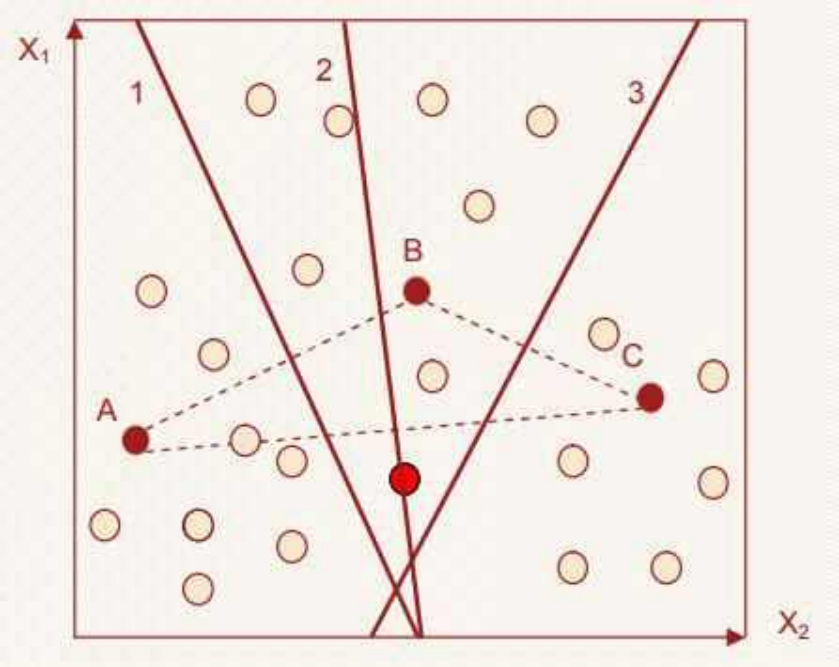


Рисунок 4.28 – Принцип пошуку меж майбутніх кластерів

Однак така геометрична інтерпретація корисна тільки для кращого розуміння. На практиці, щоб визначити, до «сфери впливу» якого центру кластера входить певний запис, обчислюють відстань від кожного запису до кожного центру в багатовимірному просторі ознак і обирають те «сім'я», для якого ця відстань мінімальна.

Розглянемо *обчислення позиції центроїдів*. Наприклад, якщо до кластера увійшли три записи з наборами ознак (x_1, y_1) , (x_2, y_2) , (x_3, y_3), то координати його центроїда розраховуватимуться так (4.26):

$$(x, y) = \left(\frac{(x_1 + x_2 + x_3)}{3}, \frac{(y_1 + y_2 + y_3)}{3} \right). \qquad (4.26)$$

Потім попередній центр кластера зміщується в його центроїд. На рис. 4.29 центри тяжіння кластерів представлені хрестиками.

Отже, центроїди стають новими центрами кластерів для наступної ітерації алгоритму. Кроки 3 і 4 повторюються, доки виконання алгоритму не буде перервано або доки не буде виконано умову відповідно до деякого критерію збіжності. Зупинка алгоритму відбувається, коли межі кластерів і розташування центроїдів перестають змінюватися, тобто на кожній ітерації в кожному кластері залишається один і той самий набір записів. Алгоритм *k-means* зазвичай знаходить набір стабільних кластерів за кілька десятків ітерацій.

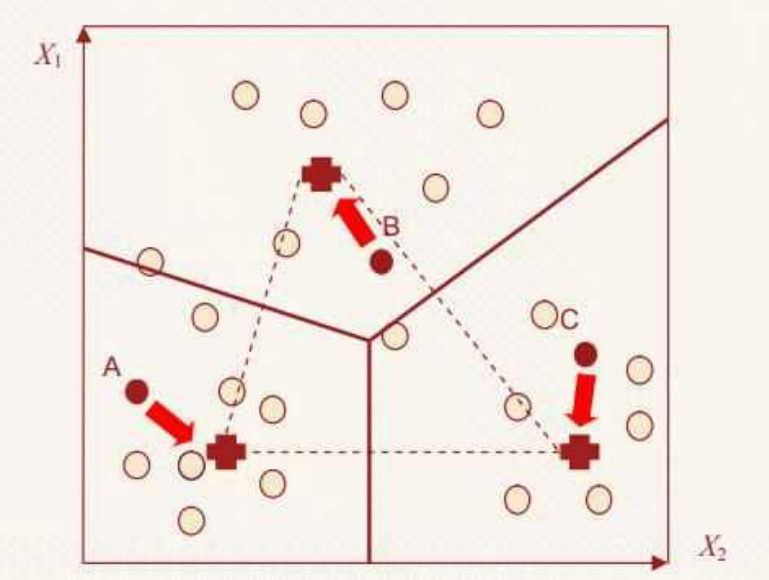


Рисунок 4.29 – Визначення центрів тяжіння кластерів (центроїдів) і нових меж кластерів

Що стосується критерію збіжності, то найчастіше використовується сума квадратів помилок між центроїдом кластера і всіма записами, що ввійшли до нього.

$$E = \sum_{i=1}^{k} \sum_{p \in C_i} (p - m_i)^2 , \qquad (4.27)$$

де:

$p \in C_i$ – довільна точка даних, що належить кластеру C_i;

m_i – центроїд цього кластера.

Інакше кажучи, алгоритм зупиниться тоді, коли помилка *E* сягне зазвичай досить малого значення.

Своєї популярності алгоритм *k-means* набув завдяки таким властивостям.

По-перше, це помірні обчислювальні витрати, які зростають лінійно зі збільшенням числа записів вихідної вибірки даних. Обчислювальна складність алгоритму визначається як $k \times n \times l$, де k – кількість кластерів, n – кількість записів і l – кількість ітерацій. Оскільки k і l задані, обчислювальні витрати зростають пропорційно числу записів вихідної множини.

По-друге, результати роботи *k-means* не залежать від порядку проходження записів у вихідній вибірці, а визначаються тільки вибором вихідних точок.

Одним із основних недоліків, притаманних алгоритму *k-means,* є відсутність чітких критеріїв вибору числа кластерів, цільової функції їхньої ініціалізації та модифікації. Крім того, він досить чутливий до шумів у даних і аномальних значеннях, оскільки вони здатні істотно вплинути на середнє значення, що використовується під час обчислення положень центроїдів. Щоб знизити вплив таких чинників, як шуми й аномальні значення, іноді на кожній ітерації використовують не середнє значення ознак, а їх *медіану*. Ця модифікація алгоритму називається *k-mediods* (k-медіан).

4.4.3. Математичне пояснення дендрограми в кластерному аналізі

Дендрограма – це дерево, що візуалізує процес ієрархічного кластерного аналізу. Вона ґрунтується на *ієрархічному агломеративному кластерному аналізі (HAC – Hierarchical Agglomerative Clustering)*, де об'єкти об'єднуються попарно на основі певної метрики відстані.

Основні математичні етапи побудови дендрограми

1. Початкове представлення даних. Маємо множину спостережень $X = \{x_1, x_2,\ldots,x_n\}$, кожне з яких є вектором у Rp.

2. Обчислення матриці відстаней. Наприклад, для Евклідової відстані.

3. Агломеративне об'єднання. Ітеративно об'єднуються найближчі кластери: спочатку кожна точка – окремий кластер, потім об'єднуються кластери на основі вибраного методу обчислення міжкластерної відстані:

– Single linkage *(мінімальна відстань)*: D (A, B) = min{d(a,b):a∈A,b∈B}.

Відстань між двома кластерами визначається *найближчою парою точок*. Добре для витягнутих (ланцюгових) кластерів, вразливо до шуму й «перетягування» кластерів;

– Complete linkage *(максимальна відстань)*: D (A, B) = max{d(a,b):a∈A,b∈B}.

Відстань між кластерами – це найдальша пара точок. Дає *компактні* й чітко розділені кластери, чутливий до крайніх значень;

– Average linkage *(середня відстань)*: D (A, B) = 1 / (|A||B|) $\sum_{a\in A,b\in B} d(a,b)$.

Відстань – це *середнє* всіх попарних відстаней між елементами кластерів A і B. Компроміс між Single і Complete. Стабільний метод, дає збалансовані кластери;

– Ward's method *(мінімізація внутрішньої дисперсії)*: мінімізує дисперсію в середині кластерів:

$$\Delta(A,B) = \frac{|A||B|}{|A|+|B|} \cdot \|\mu_A - \mu_B\|^2, \qquad (4.28)$$

де: μ_A, μ_B – центроїди кластерів.

У разі злиття двох кластерів оцінюється *збільшення сумарної квадратичної помилки* (SSE). Обирається пара кластерів, чия агрегація найменше збільшує внутрішню дисперсію. Дає *компактні* та *сферичні кластери*, передбачає, що кластери мають схожу форму / розмір;

– Ward's method – найпопулярніший для числових даних через його зв'язок із мінімізацією внутрішньогрупової варіації, аналогічно до K-Means.

4. Будується дендрограма. Візуальне представлення злиття кластерів. Висота вузла – значення відстані між кластерами під час злиття (рис. 4.30).

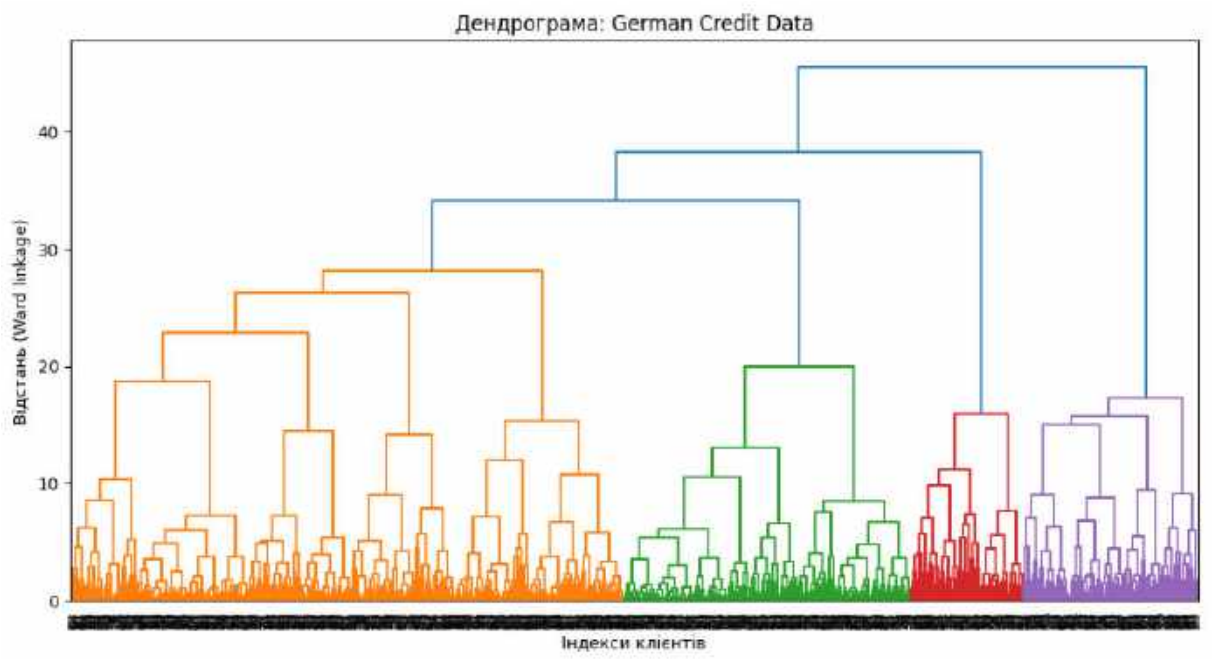


Рисунок 4.30 – Візуальне представлення злиття кластерів на дендрограмі

Для проведення кластерного аналізу потрібно використовувати функції *linkage, dendrogram:*

```
from scipy.cluster.hierarchy import linkage, dendrogram
```

Короткий опис функцій *linkage* та *dendrogram* з бібліотеки scipy.cluster.hierarchy:

```
scipy.cluster.hierarchy.linkage(y, method='single', metric='euclidean', optimal_ordering=False)
```

linkage: ця функція виконує *ієрархічне агломеративне кластерування*. Вона бере на вхід набір спостережень (як вектор відстаней між парами спостережень або масиву самих спостережень) і обчислює матрицю зв'язування. Ця матриця кодує ієрархічну структуру кластерів, які послідовно об'єднуються на основі обраного *методу зв'язування* (наприклад, *'single', 'complete', 'average', 'ward')* та *метрики відстані* (наприклад, *'euclidean', 'correlation'*).

Коротко кажучи, *linkage* визначає, як об'єднувати окремі точки даних у кластери, а потім кластери між собою, створюючи ієрархію кластерів [27].

dendrogram: ця функція приймає на вхід *матрицю зв'язування* `Z`, отриману від функції `linkage`, та візуалізує ієрархічно згруповані кластери як *дендрограму*. Дендрограма являє собою деревоподібну діаграму, де висота U-схожих гілок відображає відстань (або критерій зв'язування), на якій були об'єднані відповідні кластери. Листки дендрограми є окремі спостереження. Приклад використання функції *dendrogram:*

```
    scipy.cluster.hierarchy.dendrogram(Z, p=30,
truncate_mode=None, color_threshold=None,
labels=None, count_sort=False, distance_sort=False,
show_leaf_counts=True, no_plot=False,
no_labels=False, leaf_rotation=None,
leaf_font_size=None, leaf_label_func=None,
show_contracted=False, ax=None, orientation='top')
```

Пояснення деяких основних обов'язкових параметрів функції dendrogram

Z: обов'язковий параметр. Матриця зв'язування, отримана в результаті виконання функції linkage;

p: ціле число. Параметр, що використовується разом *з truncate_mode*. За замовчуванням 30.

truncate_mode: рядок, необов'язковий. Визначає, як обрізати дендрограму. Можливі значення:

– none (за стандартним налаштуванням): відображає повну дендрограму;

– lastp: відображає лише останні *p* об'єднаних кластерів;

– level: відображає всі кластери на рівні глибини *p*.

***labels*:** список (list), необов'язковий. Список рядків, що використовуються як мітки для листків дендрограми. Довжина списку повинна відповідати кількості початкових спостережень.

orientation: рядок (str), необов'язковий. Орієнтація дендрограми. Можливі значення:

– top (за стандартним налаштуванням): листки зверху;

– bottom: листки знизу;

– left: листки ліворуч;

– right: листки праворуч [28].

Отже, *dendrogram* візуально відображає результати ієрархічного кластерування, показуючи, як кластери формуються на різних рівнях схожості.

Так, *linkage* обчислює ієрархію кластерів, а *dendrogram* надає графічне представлення цієї ієрархії. Обидві функції є важливими інструментами для дослідження структури даних за допомогою ієрархічного кластерного аналізу.

Інформація, яка пояснює дендрограму

Кожна горизонтальна лінія = злиття кластерів.

Висота лінії = «вартість» злиття (міра відстані між кластерами).

Можна «обрізати» дендрограму на певній висоті, це дає бажану кількість кластерів.

Практичне використання дендрограм

Ціль дендрограми – побачити:

– коли відбувається злиття кластерів: оцінюється по осі Y (відстань);

– скільки кластерів логічно виділити: можна знайти великий розрив щодо висоти між злиттями. Якщо «обрізати» дендрограму горизонтально на певній висоті – отримаємо кількість кластерів на цьому рівні.

Приклад. Якщо бачимо великий розрив між відстанями на рівні ≈ 10, то оптимально обрізати трохи нижче – отримаємо, наприклад, 3–4 кластери.

Приклад побудови дендрограми буде розглянутий і на прикладі нижче.

4.4.4. Приклад кластерного аналізу: KMeans

Проведемо кластерний аналіз k-meen датасету German Credit Data (параграф 4.3.4).

1. Підготовка даних: числові ознаки стандартизуємо, категоріальні – one-hot (лістинг 4.21).

```
                    # Лістинг 4.21
    Підготовка даних
    # Імпорт необхідних компонентів
    import pandas as pd
    import matplotlib.pyplot as plt
    from sklearn.datasets import fetch_openml
    from sklearn.preprocessing import StandardScaler
    from scipy.cluster.hierarchy import linkage,
dendrogram, fcluster, inconsistent
    import numpy as np

    # Завантаження датсету German Credit Data
    data = fetch_openml('credit-g', version=1,
as_frame=True)
    df = data.frame.copy()

    # Вибір числових ознак
    numeric_cols =
df.select_dtypes(include='number').columns
    X = df[numeric_cols]

    # Масштабування ознак
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)
```

2. Кластеризація KMeans: шукаємо групи кластери з використанням метрики *Варда* (лістинг 4.22).

```
                    # Лістинг 4.22
    Ієрархічна кластеризація
    Z = linkage(X_scaled, method='ward')
    clusters = fcluster(Z, t=4,
criterion='maxclust')  # arbitrary t=4 clusters
    # 4. Attach cluster labels and target
    df['Cluster'] = clusters
    df['class'] = df['class'].astype(str)  # ensure
string
```

Будуємо дендрограми (лістинг 4.23).

```
                    # Лістинг 4.23
    Побудова дендрограми
    plt.figure(figsize=(14, 6))
    dendrogram(Z, truncate_mode='lastp', p=30,
leaf_rotation=90., leaf_font_size=12.,
show_contracted=True)
    plt.title("Дендрограма (скорочена)")
    plt.xlabel("Спостереження або кластери")
    plt.ylabel("Відстань")
    plt.grid(True)
    plt.show()
```

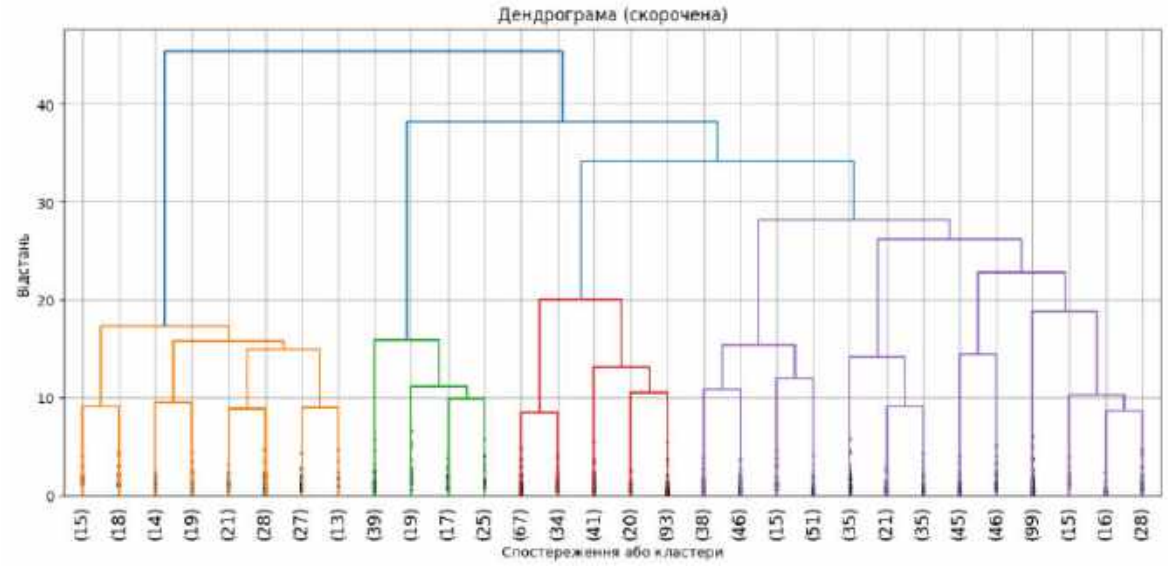


Рисунок 4.31 – Отримана дендрограма для 4-х класів (лістинг 4.23)

3. Обчислення BadShare та візуалізація результатів – частки ненадійних клієнтів Bad для кожного кластера.

```
                    # Лістинг 4.24
    Обчислення частки BadShare для кожного кластера
    cluster_summary = pd.crosstab(df['Cluster'],
df['class'])
    cluster_summary['Total'] =
cluster_summary.sum(axis=1)
    cluster_summary['BadShare'] =
cluster_summary['bad'] / cluster_summary['Total']
```

```
# Лістинг 4.25
```
Візуалізація частки BadShare для кожного кластера
```
plt.figure(figsize=(8, 5))
    sns.barplot(x=cluster_summary.index,
y=cluster_summary['BadShare'], palette='Reds')
    plt.title('BadShare vs Cluster')
    plt.xlabel('Cluster')
    plt.ylabel('Share of Bad Clients')
    plt.tight_layout()
    plt.grid(axis='y')
    plt.show()
```

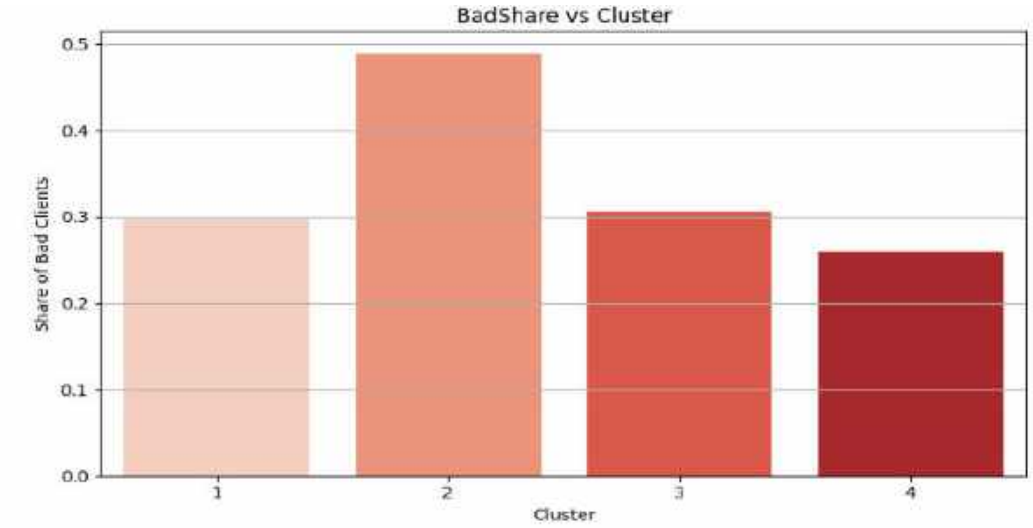


Рисунок 4.32 – Візуалізація частки ненадійних клієнтів банку (BadShare) для кожного кластера (лістинг 4.25)

На діаграмі рис. 4.32 можна оцінити кластери з найбільшою часткою ненадійних клієнтів. Таким є кластер 2.

4. Зниження розмірності через PCA (*Principal Component Analysis*), щоб на двох осях подивитися, як розділилися «надійні» й «ненадійні» клієнти.

У програмі будуть обчислені *Principal Component 1* та *Principal Component 2* – перші дві основні компоненти визначені в результаті аналізу головних компонент (PCA – Principal Component Analysis). Вони утворюють нову ортогональну систему координат, яка:

– максимально зберігає дисперсію даних;

– дає змогу зменшити кількість вимірів (наприклад, візуалізувати багатовимірні дані в 2D);

– є лінійною комбінацією вихідних ознак.

Приклад. Якщо у вас є 5 ознак: credit_amount, duration, age, installment_rate, number_existing_credits, то:

PC1 = 0.45 × credit_amount + 0.32 × duration – 0.12 × age + ...

PC2 = –0.11 × credit_amount + 0.51 × duration + ...

Коефіцієнти при змінних – це навантаження компонент (*loadings).* PCA – це про напрями та пояснення варіації, а перші 2 компоненти – просто найінформативніші напрями у ваших даних.

```
# Лістинг 4.26
```

Обчислення центроїдів візуалізація кластерів

```
    pca = PCA(n_components=2)
    X_pca = pca.fit_transform(X_scaled)
    df_pca = pd.DataFrame(X_pca, columns=['PC1',
'PC2'])
    df_pca['Cluster'] = df['Cluster'].astype(str)

    # Обчислення центроїдів у PCA-просторі
    centroids_pca = df_pca.groupby('Cluster')[['PC1',
'PC2']].mean()

    #  Візуалізація точок і центроїдів
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df_pca, x='PC1', y='PC2',
hue='Cluster', palette='tab10', alpha=0.6,
legend='full')
    plt.scatter(centroids_pca['PC1'],
centroids_pca['PC2'],
                color='black', s=200, marker='X',
label='Centroids')
    for i, (x, y) in centroids_pca.iterrows():
        plt.text(x + 0.2, y, f'C{i}', fontsize=12,
weight='bold')
    plt.title('PCA-кластери з центроїдами')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    print(centroids_pca)
```

Візуалізація точок і центроїдів за результатами виконання лістингу 4.26 показана на рис. 4.33. Нижче виведено числові значення центроїдів кластерів.

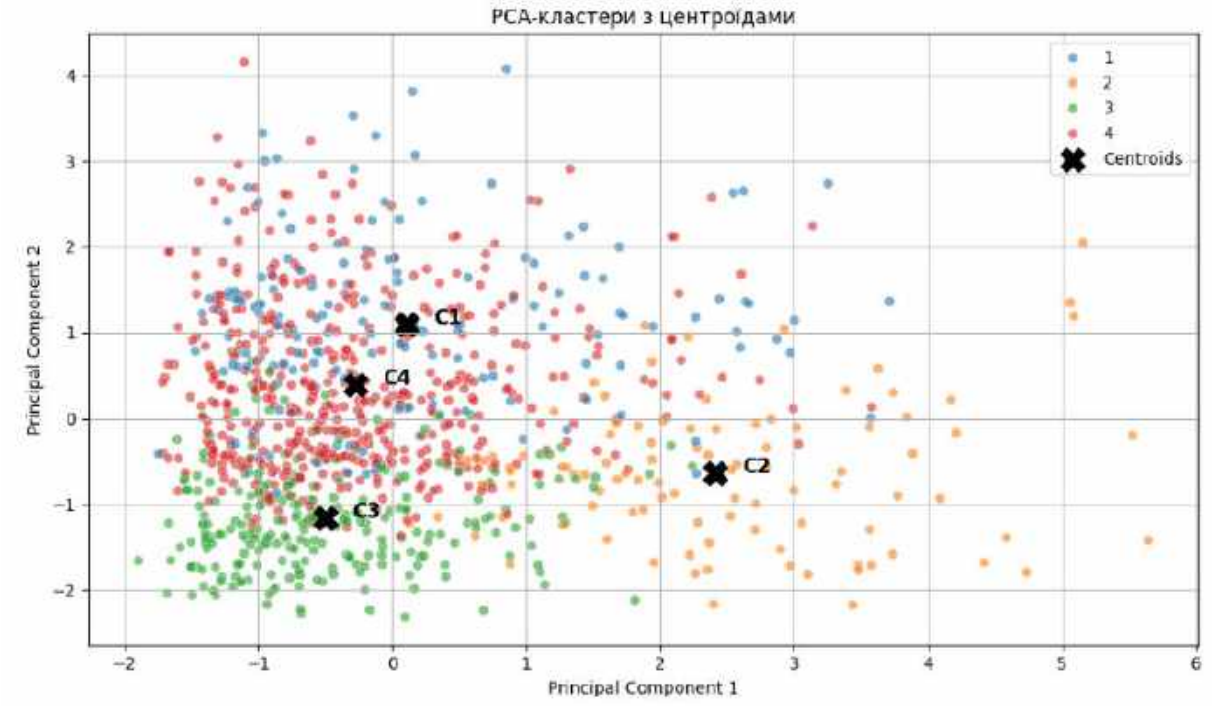


Рисунок 4.33 – Візуалізація точок і центроїдів за результатами виконання лістингу 4.26

Числові значення центроїдів кластерів (лістинг 4.26):

	PC1	PC2
Cluster		
1	0.109075	1.095017
2	2.412388	-0.642044
3	-0.494401	-1.164319
4	-0.269537	0.390567

5. Обчислення *Principal Component 1* (PC1) та *Principal Component 2* (PC2).

Аналіз компонентних навантажень PCA допомагає інтерпретувати значення основних компонент і зрозуміти, які вихідні ознаки найбільше впливають на розташування клієнтів у PCA-просторі (лістинг 4.27).

```
                    # Лістинг 4.27
Обчислення компонентних навантажень PCA
from sklearn.decomposition import PCA
import pandas as pd
# Припустимо, X_scaled — стандартизовані дані
pca = PCA(n_components=2)
pca.fit(X_scaled)

# Навантаження головних компонент:
loadings = pd.DataFrame(pca.components_.T,
                        columns=['PC1', 'PC2'],
                        index=numeric_cols)
print(loadings)
```

Результат виконання лістингу 4.27 дає числові значення компонентних навантажень.

Principal Component 1(PC1) та Principal Component 2 (PC2)

	PC1	PC2
duration	**0.657362**	-0.037356
credit_amount	**0.717989**	-0.011919
installment_commitment	-0.217797	0.119547
residence_since	0.059452	**0.542154**
age	0.014005	**0.622951**
existing_credits	0.020189	**0.438110**
num_dependents	0.028162	0.332012

Числові значення зі свого боку дають змогу краще охарактеризувати кожен кластер з погляду комбінації вихідних ознак та пов’язати ці характеристики з рівнем ризику («bad» часткою) у кожному кластері. Наприклад, кластер з великими кредитами та молодшими, менш «стабільними» клієнтами (кластер 2) виявився найбільш ризикованим. Нижче подана детальна статистика кластерів (код отримання статистики поданий у блокноті до самостійної роботи).

Статистика кластерів

Статистика BadShare vs Cluster:

class	bad	good	Total	BadShare
Cluster				
1	46	109	155	0.296774
2	49	51	100	0.490000
3	78	177	255	0.305882
4	127	363	490	0.259184

Статистика Кластерів:

	Count	duration_mean	duration_std	credit_amount_mean
Cluster				
1	155	20.232258	12.888039	3384.180645
2	100	37.710000	12.911740	8764.700000
3	255	17.949020	9.479015	2381.933333
4	490	19.222449	9.952521	2577.238776

	credit_amount_std	installment_commitment_mean
Cluster		
1	2596.252500	2.787097
2	3396.977746	2.270000
3	1521.171782	3.070588
4	1883.205618	3.124490

Приклад узагальненого аналізу результатів кластерного аналізу

Проведений кластерний аналіз розділив клієнтів на чотири *distinct* групи (кластери). Розглянемо характеристики кожного кластера на основі центроїдів у PCA-просторі та статистики ознак, а також розподіл «поганих» клієнтів.

Кластер 1 (n = 155)

PCA. Розташований у правому верхньому квадранті PCA-простору (позитивні значення обох основних компонентів, особливо PC2). Це може свідчити про клієнтів із певним поєднанням вихідних ознак, які призвели до такого розташування після зменшення розмірності.

Статистика. Характеризується середньою тривалістю кредиту, середньою сумою кредиту, помірними зобов’язаннями за

розстрочкою, досить високим середнім часом проживання та старшим середнім віком. Унікальною особливістю є те, що всі клієнти мають 2-х утриманців.

Ризик («bad» частка). Має порівняно низьку частку ненадійних («bad») клієнтів (приблизно 29.7 %).

Кластер 2 (n = 100)

PCA. Розміщений у правому нижньому квадранті PCA-простору (позитивний PC1 та негативний PC2, з найбільшим значенням PC1). Це вказує на клієнтів зі значно відмінними характеристиками, які сильно вплинули на першу головну компоненту.

Статистика. Виділяється найвищою середньою тривалістю та сумою кредиту, а також найнижчими зобов'язаннями за розстрочкою та молодшим середнім віком. Клієнти мають менше наявних кредитів і лише одного утриманця.

Ризик («bad» частка). Має найвищу частку ненадійних («bad») клієнтів серед усіх кластерів (49 %). Це робить цей кластер найбільш ризикованим.

Кластер 3 (n = 255)

PCA. Розташований у лівому нижньому квадранті PCA-простору (негативні значення обох основних компонентів). Це свідчить про клієнтів з протилежними характеристиками порівняно з кластером 2.

Статистика. Характеризується найнижчими середніми значеннями тривалості та суми кредиту, але вищими зобов'язаннями за розстрочкою. Клієнти цього кластера молодші, мають найменший середній час проживання та лише один існуючий кредит.

Ризик («bad» частка). Має помірну частку ненадійних («bad») клієнтів (приблизно 30.6 %).

Кластер 4 (n = 490)

PCA. Розташований у лівому верхньому квадранті PCA-простору (негативний PC1 та позитивний PC2). Це найбільший за кількістю клієнтів кластер.

Статистика. Має низькі середні значення тривалості та суми кредиту, високі зобов'язання за розстрочкою та найдовший середній час проживання. Клієнти цього кластера старші, але мають більший розкид у віці.

Ризик («bad» частка). Має найнижчу частку ненадійних («bad») клієнтів серед усіх кластерів (приблизно 25.9 %), що робить його найменш ризикованим.

Узагальнення та висновки

Кластери демонструють різні профілі клієнтів з погляду характеристик кредиту та демографічних показників.

Розмір та тривалість кредиту, а також зобов'язання за розстрочкою є важливими чинниками диференціації кластерів. Клієнти з більшими та довшими кредитами (кластер 2) у PCA-просторі значно відрізняються від клієнтів з меншими та коротшими кредитами (кластер 3 та 4).

Вік і час проживання також варіюються між кластерами, але їхній вплив на основні компоненти PCA може бути менш прямим.

Найбільш важливим є розподіл ненадійних («bad») клієнтів між кластерами.

Кластер 2 є найбільш проблемним, маючи найвищу частку «bad» клієнтів, незважаючи на порівняно низькі зобов'язання за розстрочкою. Можливо, значна сума та тривалість кредиту для молодших клієнтів з невеликою кількістю існуючих кредитів є чинником ризику.

Кластер 4 виглядає найменш ризикованим, попри високі зобов'язання за розстрочкою, що може компенсуватися довшим часом проживання та старшим віком.

Кластери 1 та 3 мають помірний рівень ризику, але відрізняються за своїми характеристиками. Кластер 1 має старших клієнтів з більшою кількістю утриманців, а кластер 3 – молодших клієнтів з меншими кредитами.

Приклад формування рекомендацій та подальшого аналізу

Результати кластерного аналізу можуть бути використані для сегментації клієнтів та розробки диференційованих стратегій управління ризиками.

Варто приділити особливу увагу клієнтам, що потрапляють до кластеру 2, та ретельно оцінювати їх кредитоспроможність. Можливо, потрібні додаткові критерії для схвалення кредитів для цієї групи.

Клієнти з кластеру 4 можуть вважатися менш ризикованими, що можна враховувати під час ухвалення рішень щодо кредитування.

Подальший аналіз може охоплювати вивчення інших ознак, які не були явно відображені в перших двох основних компонентах PCA, для кращого розуміння чинників ризику в кожному кластері.

Також корисно було б проаналізувати характеристики надійних («good») та ненадійних («bad») клієнтів безпосередньо в кожному кластері, щоб виявити специфічні відмінності.

Питання для самоконтролю

1. Поясніть своїми словами, яка основна мета кластерного аналізу. Чим він відрізняється від завдань класифікації?
2. Що означає принцип «високої внутрішньокластерної схожості та низької міжкластерної схожості»?
3. Опишіть покроково, як працює алгоритм k-середніх *(k-means),* починаючи від ініціалізації центроїдів.
4. Яка роль параметра *'k'* в цьому алгоритмі? Назвіть щонайменше один метод, який допомагає обрати оптимальне значення *'k'*.
5. Назвіть два основні недоліки алгоритму *k-means.*
6. У чому полягає ключова різниця між *Евклідовою* та *Манхеттенською відстаннями*? Наведіть приклад, де одна з них може бути доцільнішою за іншу.
7. Що таке *відстань Мінковського* і як вона узагальнює *Евклідову* та *Манхеттенську відстані*?
8. Чим принципово ієрархічна кластеризація відрізняється від методу k-середніх?
9. Поясніть різницю між агломеративним («висхідним») і дивізійним («низхідним») підходами в ієрархічній кластеризації.
10. Що з математичного погляду показує висота з’єднання гілок на дендрограмі?
11. Як за допомогою дендрограми можна визначити оптимальну кількість кластерів для набору даних?

4.5. Асоціативні правила

4.5.1. Основні відомості про асоціативні правила

Асоціативні правила (Association Rules) – метод некерованого (без учителя) навчання, що шукає інформативні взаємозв'язки («якщо → то») у великих наборах зазвичай платіжних транзакцій. Ще зустрічається назва цього підходу *аффінітивний аналіз* (affinity analysis) – один із поширених методів навчання *без вчителя*. Його назва походить від англійського слова *affinity*, яке в перекладі означає «близькість», «схожість».

Приклади застосування асоціативних правил:

– виявлення наборів товарів, які в супермаркетах часто купуються разом або ніколи не купуються разом;

– визначення частки клієнтів, що позитивно ставляться до нововведень у їхньому обслуговуванні;

– визначення профілю відвідувачів вебресурсу;

– визначення частки випадків, у яких нові ліки показують небезпечний побічний ефект.

Мета пошуку асоціативних правил – дослідження взаємного зв'язку між подіями, які відбуваються спільно. Різновидом аффінітівного аналізу є аналіз *ринкового кошика* (market basket analysis), метою якого є виявлення асоціації між різними подіями, тобто знайти правила для кількісного опису взаємного зв'язку між двома або більше подіями. Такі правила називаються *асоціативними правилами* (association rules)

Базовим поняттям у теорії асоціативних правил є транзакція – деяка множина подій, що відбуваються спільно.

Типова транзакція – придбання клієнтом товару в супермаркеті. У переважній більшості випадків клієнт купує не один товар, а набір товарів, який називається ринковим кошиком. Водночас виникає запитання: чи є купівля одного товару в кошику наслідком або причиною купівлі іншого товару, тобто чи пов'язані ці події? Цей зв'язок і встановлюють асоціативні правила,

наприклад, може бути виявлено асоціативне правило, яке стверджує, що клієнт, який купив молоко, з імовірністю 75 % купить і хліб.

Наступне важливе поняття – предметний набір або ринковий кошик.

Це непорожня множина предметів, що з'явилися в одній транзакції.

Аналіз ринкового кошика – це аналіз наборів даних для визначення комбінацій товарів, пов'язаних між собою. Інакше кажучи, проводиться пошук товарів, присутність яких у транзакції впливає на ймовірність наявності інших товарів або комбінацій товарів.

Основні алгоритми пошуку асоціативних правил у частих наборів

Apriori

Якщо набір елементів є частим, то всі його підмножини також є частими.

Якщо набір елементів не є частим, то жодна з його надмножин не може бути частою. Коротко про роботу алгоритму:

– знаходить часті одиничні елементи (itemsets of size 1): сканує базу даних і визначає елементи, що задовольняють заданий мінімальний поріг підтримки (minimum support);

– генерує кандидатські набори розміру k + 1: використовуючи знайдені часті набори розміру k, алгоритм генерує всі можливі кандидатські набори розміру k + 1, об'єднуючи часті набори розміру k, які відрізняються лише одним елементом;

– перевіряє підтримку кандидатських наборів: сканує базу даних для визначення фактичної підтримки кожного кандидатського набору розміру k + 1. Лише ті кандидатські набори, підтримка яких перевищує мінімальний поріг, вважаються частими наборами розміру k + 1;

– повторює кроки 2 та 3: процес триває, допоки не буде знайдено жодного нового частого набору елементів.

Основна проблема Apriori: генерування значної кількості кандидатських наборів, особливо для великих баз даних, що може бути обчислювально витратним.

Незважаючи на свою неефективність порівняно з пізнішими алгоритмами, як-от FP-Growth, Apriori є важливим фундаментом у сфері видобутку асоціативних правил.

FP-Growth

FP-дерево (FP це абревіатура *frequent itemsets, часті набори елементів*). Алгоритм швидший за Apriori для великих обсягів даних, але складніший у реалізації. FP-Growth не використовується безпосередньо для кластеризації даних на групи схожих об'єктів. Кластерний аналіз має на меті розділити дані на кластери на основі їхньої схожості за певними ознаками. Проте результати роботи алгоритму FP-Growth можуть бути використані як вхідні дані або як допоміжний крок для деяких підходів до кластерного аналізу, особливо коли дані мають транзакційний характер.

Основні характеристики асоціативних правил

Розглянемо основні характеристики асоціативних правил на прикладі 10 транзакцій у супермаркеті. Сучасні касові апарати в супермаркетах дають змогу збирати інформацію про покупки, яка може зберігатися в базі даних. Накопичені дані можуть використовуватися для побудови систем пошуку асоціативних правил.

У таблиці 4.11 наведено приклад, що містить дані транзакцій покупок у магазині ряду продуктів. Наш приклад для спрощення буде обмежений 10 транзакціями, що містять 13 видів продуктів.

Візуальний аналіз таблиці прикладу показує, що всі чотири транзакції, в яких фігурує *салат*, також містять *помідори*, і що чотири із семи покупок, що містять помідори, також містять *салат*. *Салат і помідори* здебільшого купуються разом. Асоціативні правила дають змогу виявляти й кількісно описувати такі збіги.

Таблиця 4.11. Приклад інформації про транзакції (чеки покупок)

№	Транзакції
1.	Сливи, салат, помідори
2.	Селера, цукерки
3.	Цукерки
4.	Яблука, морква, помідори, картопля, цукерки
5.	Яблука, апельсини, салат, цукерки, помідори
6.	Персики, апельсини, селера, помідори
7.	Квасоля, салат, помідори
8.	Апельсини, салат, морква, помідори, цукерки
9.	Яблука, банани, сливи, морква, помідори, цибуля, цукерки
10.	Яблука, картопля

Асоціативне правило складається із двох наборів предметів, що мають назву *умова* (*antecedent*) та *наслідок* (*consequent*) і записуються у вигляді: «*Якщо умова*, *то наслідок*». Умова може обмежуватися тільки одним предметом. Правила відображаються за допомогою стрілок, спрямованих від умови до наслідку, наприклад, *помідори → салат*.

Основними характеристиками, що описують асоціативне правило, є *підтримка* (*support*) і *достовірність (confidence*).

Якщо позначити базу даних транзакцій через *D*, а число транзакцій у цій базі *N*, то кожна транзакція d_i, … , *i* 1…*N* являє собою певний набір предметів. Позначимо підтримку правила через *S*, а вірогідність – через *C*.

Підтримка асоціативного правила – це число транзакцій, які містять як умову, так і наслідок. Наприклад, для асоціації *A* можна записати (4.29):

$$\text{Support}(X \to Y) = \frac{\text{Кількість транзакцій, що містять } X \cup Y}{\text{Загальна кількість транзакцій}} \qquad (4.29)$$

Достовірність асоціативного правила *A* являє собою міру точності правила й визначається як відношення кількості транзакцій, що містять умову й наслідок, до кількості транзакцій, що містять тільки умову (4.30).

$$\text{Confidence}(X \to Y) = \frac{\text{Кількість транзакцій, що містять } X \cup Y}{\text{Кількість транзакцій, що містять } X} \qquad (4.30)$$

Достовірність вимірює **надійність** правила. Висока достовірність означає, що якщо в транзакції є *X*, то досить імовірно, що там також буде *Y*. Однак висока достовірність не обов'язково означає значущість правила (наприклад, якщо *Y* є досить поширеним елементом).

Обидві метрики, підтримка та достовірність, є важливими для оцінювання якості та значущості асоціативних правил, виявлених за допомогою алгоритмів, як-от *Apriori* та *FP-Growth*. Зазвичай задаються мінімальні пороги для підтримки та достовірності, щоб відфільтрувати нецікаві або ненадійні правила.

Достовірність 0,67 здається не така вже й мала. Чому ж ми говоримо про «незначну достовірність» цієї асоціації? Справа в тому, що помідори зустрічаються в 7 чеках з 10 (P(B)= 0,7). Прийнято, що всі правила з *достовірністю* меншою, ніж проста ймовірність випадання наслідку не повинні розглядатися, адже вони, по суті, випадкові. Для прийняття асоціативного правила його *достовірність* повинна бути не меншою, ніж імовірність наслідку

Якщо *підтримка* й *достовірність* досить високі, можна з великою імовірністю стверджувати, що будь-яка майбутня транзакція, що містить умову, буде містити й наслідок. Для прикладу розглянемо правило *салат → помідори*, що напрошується з попередніх спостережень. Для нього:

$S(салат \rightarrow помідори) = 4/10 = 0{,}4;$

$C(салат \rightarrow помідори) = 4/4 = 1.$

Це правило, що зустрічається в 40 % транзакцій, є для вихідних даних абсолютно правильним – у всіх випадках, коли клієнт купує салат, він разом з тим купує й помідори. Це легко пояснити логічно – обидва продукти використовуються для готування овочевих блюд і дійсно часто купуються разом. Тепер розглянемо асоціацію *цукерки → помідори,* у якій містяться мало сумісні в гастрономічному смислі продукти.

Підтримка цієї асоціації $S = 4/10 = 0{,}4$ – та ж, що в попереднього правила, а вірогідність $C = 4/6 = 0{,}67$. Отже, порівняно невисока *достовірність* цієї асоціації дає привід засумніватися в тому, що вона є правилом.

Останнє зауваження при досить великій номенклатурі товарів призводить до потреби подвоювати кількість розрахунків. На практиці аналітики можуть віддавати перевагу правилам, які мають високу *підтримку* (вище певного рівня, наприклад 0,3) або високу достовірність (не менше 0,8–0,85).

Висування одночасних вимог щодо *підтримки й достовірності* дають змогу значно пом’якшити критерії (підтримку до 0,1–0,15, достовірність – до 0,67–0,75). Правила, для яких значення

підтримки й достовірності перевищують певні, задані користувачем пороги, називають *сильними правилами* (strong rules). Усі наведені вище числові значення – емпіричні.

Наприклад, у задачах виявлення шахрайських операцій значення підтримки може знижуватися й до 1 %, оскільки із шахрайством пов'язане порівняно невелике число транзакцій. Як видно з викладеного вище, знайти асоціації не важко. Але як визначити, чи є кожне окреме правило значимим, таким, що дає змогу не вгадувати, а прогнозувати наслідок залежно від умови?

Наприклад, у кожних 100 аваріях у росії одним з автомобілів є ВАЗ. Але правило *аварія ВАЗ*, як і зворотне йому, буде малозначимим, оскільки в деяких регіонах ВАЗ становить більше 80 % автомобільного парку. Аналогічно в жіночих бутиках, де, крім одягу й білизни, продаються шоколад і цукерки, вводити правило *білизна* → *шоколад* не раціонально, адже, за статистикою, шоколад і цукерки в них купують 87 % відвідувачок, які водночас можуть не купувати білизни. Крім об'єктивних оцінок (*підтримки* й *достовірності*) кожного зі згенерованих правил, різні джерела радять використовувати деякі суб'єктивні оцінки.

Усі вони так чи інакше базуються на об'єктивних оцінках.

Ліфт (від interest lift – підвищення інтересу) обчислюється в такий спосіб.

Для нашого прикладу в таблиці 4.11 візьмемо два правила з однаковою *вірогідністю*:

Ліфт – це відношення частоти появи умови в транзакціях, які містять й умову, і наслідок, до частоти появи наслідку загалом. Чим більше значення ліфта, тим частіше наслідок визначається умовою порівняно з випадками, коли умова відсутня. **Якщо ліфт дорівнює 1**, зв'язок відсутній, близькі ж до нуля значення свідчать про сильну зворотну залежність

P(*салат*) = 4/10 = 0,4. C(*помідори* → *салат*) = 4/7 = 0,57.

P(*цукерки* 6/10 = 0,6. C(*помідори* → *цукерки*) = 4/7 = 0,57.

Здавалося б, правила однаково достовірні. Після розрахунку *ліфта* все стає на свої місця:

L(*цукерки* $\rightarrow$ *салат*) = 0,57/0,4 = 1,425;

L(*помідори* $\rightarrow$ *цукерки*) = 0,57/0,6 = 0,95.

Проте не потрібно вважати ліфт універсальним мірилом адекватності.

Справа в тому, що правило з меншою підтримкою й більшим ліфтом може бути менш значимим, ніж альтернативне правило з більшою підтримкою й меншим ліфтом. Це пов'язане з тим, що останнє застосовується для більшої кількості покупців.

Левередж (*leverage* – важіль, плече) – це різниця між спостережуваною частотою, з якої умова й наслідок з'являються спільно, та добутком частот появи умови й наслідку окремо

$$T(A \rightarrow B) = S(A \rightarrow B) - P(A)\,P(B) \qquad (4.31)$$

Левередж дає змогу впоратися із ситуаціями, коли й *підтримка*, й *ліфт* у правил ідентичні, але їхня значимість відрізняється. Наприклад, у нашому овочевому магазині у правила *салат* $\rightarrow$ *помідори*

C =1; S = 0,7; L = 1/0,7 = 1,43.

Морква, як показує таблиця, також продається тільки з помідорами, і зустрічається чотири рази, тому й у правила *морква*$\rightarrow$*помідори*

C =1 ; S = 0,7; L = 1/0,7 = 1,43.

А левередж у цих правил відрізняється на 30 %:

T(*морква* $\rightarrow$*помідори*) = 0,3 – 0,3 × 0,7 = 0,09.

T(*салат* $\rightarrow$ *помідори*) = 0,4 – 0,4 × 0,7 = 0,12.

Отже, значимість другої асоціації більша ніж першої. Альтернативою *левередж* є *поліпшення*.

Поліпшення (*improvement*) – це відношення частоти спостережуваного виконання правила до добутку частот появи умови й наслідку окремо.

$$I(A \to B) = \frac{S(A \to B)}{P(A) \cdot P(B)} \qquad (4.32)$$

Фактично поліпшення показує, у скільки разів розглянуте правило забезпечує правильний прогноз краще, ніж випадкове вгадування.

$I(A \to B) \leq 1$ не є значимими

Такі міри, як ліфт, левередж і поліпшення, можуть використовуватися для обмеження набору розглянутих асоціацій через встановлення граничних значень значимості, нижче яких асоціації відкидаються.

4.5.2. Пошук асоціативних правил

У процесі пошуку асоціативних правил може виконуватися виявлення всіх асоціацій, підтримка й вірогідність для яких перевищують заданий мінімум. Найпростіший алгоритм пошуку асоціативних правил розглядає всі можливі комбінації умов і наслідків, оцінює для них підтримку й вірогідність, а потім вилучає всі асоціації, які не задовольняють заданим обмеженням. Число можливих асоціацій зі збільшенням предметів зростає експоненційно. Якщо в базі даних транзакції присутні k предметів і всі асоціації є бінарними (тобто містять по одному предмету в умові й наслідку), потрібно буде проаналізувати $k \cdot 2k$ 1 асоціацій. Оскільки реальні бази даних транзакцій, розглянуті під час аналізу ринкового кошика, звичайно, містять тисячі предметів, обчислювальні витрати в процесі пошуку асоціативних правил значні.

Наприклад, якщо аналізувати вибірку, що містить усього 100 предметів, то кількість асоціацій, утворених цими предметами, становитиме 100 299 6,4 1031. Природно, що побудова такої кількості правил і перевірка вірогідності кожного з них – завдання нелегке. Тому для її вирішення використовуються деякі алгоритмічні прийоми, що дають змогу скорочувати простір можливих

рішень. Однією з найпоширеніших є методика, заснована на *виявленні так званих частих наборів*, коли аналізуються тільки ті асоціації, які зустрічаються досить часто. На цій концепції заснований відомий алгоритм *пошуку асоціативних правил Apriori.*

Ієрархічні асоціативні правила

Якщо провадити пошук асоціативних правил серед окремих предметів (наприклад, товарів у магазині), то можна виявити, що в багатьох випадках асоціації з високою підтримкою для окремих товарів практично відсутні. Особливо це характерно для супермаркетів, де асортимент товарів кожного виду досить великий. Наприклад, у продажу можуть бути десятки різновидів таких товарів, як *кетчуп* і *макарони.* Тому, незважаючи на те, що загалом підтримка асоціації *макарони* → *кетчуп* може бути досить високою, підтримка асоціацій між окремими видами цих товарів, імовірно, буде низькою. Отже, такі асоціації хоча й можуть становити інтерес, будуть вилучені з розгляду, оскільки не задовольнятимуть деякому мінімальному порогу підтримки *S*min.

Для вирішення цієї проблеми під час пошуку асоціативних правил розглядають не окремі предмети, а їх ієрархію. Якщо на нижніх ієрархічних рівнях подібні цікаві асоціації відсутні, то на більш високих вони можуть мати місце. Інакше кажучи, підтримка окремого предмета завжди буде меншою, ніж підтримка групи, до якої він входить:

$$S(I) > S(i_j), \qquad (4.33)$$

де:

I – група в ієрархії;

$i_j - j$ – той предмет, що входить у цю групу.

Причини цього очевидні: загальна підтримка групи дорівнює сумі підтримки внесених у неї предметів.

В ієрархічній схемі кетчуп і макаронні вироби мають множину підвидів. Найчастіше під час обробки касових чеків може бути отримана інформація лише про конкретний товар. Якщо ж структура бази даних транзакцій дає змогу відображати ієрархію товарів, можна досліджувати всі утворені ними ієрархічні рівні. Асоціативні правила, виявлені для предметів, розташованих на

різних ієрархічних рівнях, отримали назву *ієрархічні асоціативні правила.*

У зарубіжній літературі вони також відомі як багаторівневі правила (*multilevel rules*) або узагальнені правила (*generalized rules*). Ієрархія предметів, пов'язаних з комп'ютерною технікою, одна з найбільш складних – вона містить 5 рівнів. Зазвичай ієрархічні рівні нумеруються згори донизу, починаючи з нульового. Вузол, що відповідає нульовому рівню, називають кореневим (*root node*). У комп'ютерній техніці перший рівень відповідає всім товарам, другий – категоріям комп'ютерного забезпечення, третій – конкретним предметам (товарам), четвертий – фірмі-виробникові, п'ятий – конкретній моделі певного товару.

Наприклад, товар (1) – принтер (2) лазерний (3) фірми Xerox (4) модель FX3100 (5). Шукати навіть у великій базі даних транзакцій його асоціацію з іншим товаром (1) – мишею (2) бездротовою (3) фірми Genius (4) модель KW-09 (5) марно. Підтримка такого правила навряд чи перевалить за 1 %, та й то лише з урахуванням того, що ця конкретна модель миші або принтера мають значну абсолютну популярність. Універсальним методом для схожих випадків є послідовний спуск за рівнями ієрархії від першого до останнього з виписуванням усіх асоціацій на кожному рівні. Логічно, що на вищих рівнях буде високою підтримка (наприклад, правил «*ноутбук → миша*») і вірогідність.

Звертаючи увагу на більш низькі рівні абстракції, аналізують нащадки лише тих категорій і підкатегорій, які є частими наборами, тобто зустрічаються не менше наперед заданої кількості разів, де k номер рівня. Як на недолік ієрархічного підходу до пошуку асоціативних правил, іноді вказують на те, що отримані правила в більшості випадків стосуються не окремих предметів, а їхніх груп, а це не завжди відповідає вимогам аналізу. Однак у деяких предметних сферах кількість окремих найменувань може бути настільки значною, що виявлення асоціативних правил навіть на високому рівні узагальнення – це вже успіх. Крім того, технологія бізнес-аналізу більшою мірою орієнтована на узагальнені даних, тому виробники й конкретні моделі рідко цікавлять дослідника під час побудови асоціацій. Існує кілька підходів до пошуку ієрархічних асоціативних правил.

Найчастіше використовуються спадні методи, коли часті предметні набори досліджуються на кожному ієрархічному рівні, починаючи з першого й закінчуючи рівнем з найбільшою деталізацією. Інакше кажучи, щойно виявляються всі часті предметні набори на першому рівні, починається пошук популярних предметних наборів на другому тощо. На кожному рівні для відкриття частих наборів може використовуватися будь-який алгоритм, наприклад Apriori та його модифікації.

4.5.3. Стратегії ведення пошуку правил

Відомі кілька стратегій ведення пошуку правил.

Використання однакового порога мінімальної підтримки Smin k const на всіх ієрархічних рівнях.

Під час пошуку правил одноразово задається певний поріг мінімальної підтримки (наприклад, 5 %), у разі досягнення якого набір вважається частим, у протилежному випадку – не вноситься до списку правил. Перевага підходу: висока швидкість аналізу предметної галузі, обумовлена відмовою від оцінки часткових наборів, отриманих з тих, які недостатньо часто зустрічаються. Недолік: ризик пропустити тонкі асоціації на нижніх рівнях ієрархії, а саме: перевагу користувачів одного виробника або зв'язку певних моделей товарів. Іноді цей недолік намагаються обійти за рахунок зниження мінімального рівня підтримки. Як наслідок, поява десятків і сотень незмістовних правил з низькою підтримкою й вірогідністю.

Зниження порога мінімальної підтримки під час переходу до нижніх рівнів ієрархії. Може здійснюватися індивідуально для кожної підгрупи (наприклад, S («*Комп'ютери*») = 0,1, тоді під час розподілу комп'ютерів на категорії «настільний» та «портативний» встановлюється S («*Настільний*») = 0,06 й S («*Портативний*») = 0,04) або функціонально.

Функціональний вид зниження порога зазвичай пов'язаний з кількістю підкатегорій або порядковим номером рівня. Наприклад, якщо категорія «Комп'ютери» з рівнем підтримки S^2 0,1 ділиться на дві підкатегорії: «настільний» та «портативний», то в обох під категорій:

$$S^3 = S^2/2 = 0{,}05.$$

Інший варіант функціонального підходу полягає у встановленні порогів мінімальної підтримки винятково залежно від рівня, скільки б там підкатегорій не було S^kmin S^1min / k. Щодо такого варіанта функціонального обмеження, якщо, наприклад, для всієї бази даних прийнятий мінімальний рівень підтримки S^1min0,2, для категорії «комп'ютери» (2 рівень) він буде 0,2 / 2 = 0,1; «настільні комп'ютери» – 0,2 / 3 = 0,067; «настільні комп'ютери Dell» 0,2 / 4 = 0,05 тощо.

Перевага цього підходу очевидна – він дає змогу піти набагато далі по ієрархії в пошуку асоціацій. Однак у випадку з індивідуальним завданням граничного рівня для кожної підкатегорії, левову частину часу займатиме саме ця процедура. У випадку ж встановлення порогу залежно від рівня чи кількості підкатегорій не враховуються індивідуальні переваги й недоліки окремих виробників і моделей.

Міжрівнева фільтрація заснована на так званому рівні проходу (level passage). Порівняно високий для верхніх рівнів ієрархії граничний рівень підтримки зберігається на нижніх у першому проході. Потім у кожному проході цей рівень знижується. Ті товари низького рівня, які мають асоціації, у такий спосіб ідентифікуються раніше, ніж їхні батьківські категорії, які, можливо, і не будуть мати відповідної підтримки. Рекомендується виконувати три-чотири проходи бази даних транзакцій для сполучення рівнів довіри представників різних рівнів.

4.5.4. Приклад побудови асоціативних правил: *apriori*

Розглянемо приклад побудови асоціативних правил з використанням бібліотеки *Python, mlxtend (machine learning extensions)*, що надає корисні інструменти для машинного навчання та аналізу даних. Її основна мета – розширити можливості основних бібліотек, як-от scikit-learn, надаючи додаткові функції для:

Виявлення частих наборів елементів та асоціативних правил: охоплює ефективні алгоритми, як-от Apriori та FP-Growth, для аналізу транзакційних даних.

Селекції ознак: пропонує різні методи для вибору найбільш важливих ознак для моделювання.

Оцінки моделей: містить функції для аналізу та порівняння продуктивності різних моделей машинного навчання.

Візуалізації: надає зручні засоби для візуалізації результатів аналізу та моделей.

Стекінгу та ансамблевих методів: дає змогу комбінувати кілька моделей для підвищення їхньої точності.

Обробки послідовностей: містить інструменти для аналізу послідовних даних.

Загальних утиліт машинного навчання: охоплює різні допоміжні функції для попередньої обробки даних та інших завдань.

Для побудови асоціативних правил використовуються функції ***apriori*** та ***association_rules,*** що надаються бібліотекою ***mlxtend.frequent_patterns.***

apriori(df, min_support=0.5, use_colnames=False, low_memory=False, max_len=None, verbose=0, store_transaction_id=False):

Призначення: знаходить часті набори елементів (frequent itemsets) у булевому DataFrame транзакцій. Вхід: DataFrame, де кожен рядок представляє транзакцію, а стовпці – елементи (товари). Значення є булевими (True, якщо елемент присутній у транзакції, False – якщо ні).

Параметри:

min_support: мінімальна підтримка (частка транзакцій, що містять набір елементів) для того, щоб набір вважався частим (значення від 0 до 1).

use_colnames: якщо True, використовує назви стовпців як назви елементів у наборах.

low_memory: оптимізація для роботи з великими датасетами, може бути повільнішою.

max_len: максимальна кількість елементів у наборі.

verbose: рівень виведення інформації про процес.

store_transaction_id: чи зберігати ідентифікатори транзакцій для кожного набору. Вихід: DataFrame з двома стовпцями: support (підтримка набору) та itemsets (сам набір елементів у вигляді frozenset).

association_rules (df, metric='confidence', min_threshold=0.8, support_only=False, include_if_contains=None, exclude_if_contains=None):

Призначення: генерує асоціативні правила на основі DataFrame частих наборів елементів, отриманого від функції apriori. Вхід: DataFrame частих наборів елементів (з підтримкою та наборами).

Параметри:

metric: метрика для оцінки сили правила (наприклад, 'confidence', 'lift', 'support', 'leverage', 'conviction').

min_threshold: мінімальне значення обраної метрики для того, щоб правило було виведено.

support_only: якщо True, виводить лише підтримку антецедента та консеквента.

include_if_contains: фільтрує правила, які містять в антецеденті або консеквенті вказані елементи.

exclude_if_contains: фільтрує правила, які не повинні містити в антецеденті або консеквенті вказані елементи. Вихід: DataFrame з різними стовпцями, що описують згенеровані правила, включаючи antecedents (антецедент – ліва частина правила), consequents (консеквент – права частина правила) та різні метрики (support, confidence, lift тощо).

Отже, основні завдання функцій apriori apriori і association_rules такі:

apriori – це функція для знаходження, що часто купують разом (часті набори елементів).

association_rules – це функція для знаходження правил «якщо... то... » на основі того, що часто купують разом. Вона визначає ймовірність того, що певні товари будуть куплені, якщо вже куплені інші.

Нижче наведено приклад побудови асоціативних правил на типових *«market-basket»* даних на прикладі відкритого датасету Groceries (від Kaggle).

Опис датасету Groceries (від Kaggle)

Датасет ***Groceries*** один із найпопулярніших наборів для аналізу типу «Market Basket» на Kaggle: *38765 рядків* із деталями покупок; кожен рядок – одна транзакція (список товарів у кошику); *169*

унікальних товарів (від хліба й молока до овочів). Посилання на датасет: https://www.kaggle.com/datasets/heeraldedhia/groceries-dataset

Нижче надано лістинг 4.28, який виконує такі процедури:

1. **Імпортує класи, функції** і завантажує датасет Groceries Dataset.

```
                    # Лістинг 4.28-1
    Імпорт бібліотек і завантаження датасету
# Groceries Dataset
    import pandas as pd
    from mlxtend.frequent_patterns import apriori,
association_rules
    # Завантаження датасету
    try:
        df =
pd.read_csv('https://www.kaggle.com/datasets/heeralde
dhia/groceries-dataset/download')
    except:
        df = pd.read_csv('Groceries_dataset.csv')
```

2. Підготовлюємо дані у формат, придатний для алгоритму Apriori. Кожна транзакція буде представлена як рядок, а кожен товар – як стовпець з булевим значенням (True/False).

```
                    # Лістинг 4.28-2
    Підготовка даних
    # Перетворення даних у формат, придатний для mlxtend
    # Видаляємо пробіли на початку та в кінці назв
товарів
    df['itemDescription'] =
df['itemDescription'].str.strip()

    # Групуємо дані за номером учасника
(Member_number) та назвою товару
#(itemDescription) та підраховуємо кількість кожного
товару в кожній транзакції.
    # Потім перетворюємо це у формат, де кожен рядок
- транзакція, а стовпці - товари.
# Заповнюємо відсутні значення (товари, яких не було
в транзакції) нулями.
    basket = (df.groupby(['Member_number',
'itemDescription'])['itemDescription']
              .count().unstack().reset_index().fillna(0)
               .set_index('Member_number'))
```

```
    # Перетворюємо кількості товарів на булеві
значення (True, якщо товар був у транзакції, False -
якщо ні).
    def encode_units(x):
        if x <= 0:
            return False
        if x >= 1:
            return True

    basket_sets = basket.applymap(encode_units)
```

3. Застосовуємо алгоритм Apriori для знаходження частих наборів елементів (наборів товарів, які часто купують разом). Мінімальна підтримка встановлена на 1 % (тобто набір товарів має зустрічатися щонайменше в 1 % транзакцій).

Лістинг 4.28-3

```
    Застосування алгоритму Apriori
    # Знаходимо часті набори елементів (товари, які
часто купують разом) з мінімальною підтримкою 1%
(0.01).
    # `use_colnames=True` використовує назви товарів
у вихідних наборах елементів.
    frequent_itemsets = apriori(basket_sets,
min_support=0.01, use_colnames=True)
```

4. Генеруємо асоціативні правила на основі знайдених частих наборів елементів. Використовується метрика «ліфт» для оцінки сили правил, і виводяться лише правила з ліфтом >= 1 (позитивна асоціація).

Лістинг 4.28-4

```
    Генерування асоціативних правил
    # Генеруємо правила на основі знайдених частих
наборів елементів.
    # `metric="lift"` використовуємо метрику "ліфт"
для оцінки сили # # правил.`min_threshold=1` виводимо
лише правила з ліфтом >= 1 (позитивна асоціація).
    rules = association_rules(frequent_itemsets,
metric="lift", min_threshold=1)
```

5. Виводимо згенеровані правила та перші 10 правил, відсортовані за ліфтом

```
                # Лістинг 4.28-5
    Виведення правил
    print(rules)
    # Додатково: Виведення перших 10 правил,
відсортованих за ліфтом
    print("\nПерші 10 правил, відсортовані за ліфтом:")
    print(rules.sort_values('lift',
ascending=False).head(10))
```

Результат виконання лістингів 4.28:

```
Перші 10 правил, відсортовані за ліфтом:
                                      antecedents  \
15180  (sausage, whole milk, other vegetables)
15189                    (yogurt, rolls/buns)
15181   (yogurt, other vegetables, rolls/buns)
15188                    (sausage, whole milk)
12800                           (curd, yogurt)
12797                    (sausage, whole milk)
15179        (sausage, whole milk, rolls/buns)
15190              (yogurt, other vegetables)
15193                    (sausage, rolls/buns)
15176   (whole milk, other vegetables, yogurt)

Перші 10 правил, відсортовані за ліфтом (продовження):
                consequents  antecedent support  \
15180                    (yogurt, rolls/buns)   0.050282
15189 (sausage, whole milk, other vegetables)   0.111339
15181                   (sausage, whole milk)   0.052335
15188  (yogurt, other vegetables, rolls/buns)   0.106978
12800                   (sausage, whole milk)   0.040277
12797                          (curd, yogurt)   0.106978
15179              (yogurt, other vegetables)   0.048743
15190        (sausage, whole milk, rolls/buns)   0.120318
15193   (whole milk, other vegetables, yogurt)   0.082350
15176                   (sausage, rolls/buns)   0.071832
```

Узагальнення й аналіз результатів асоціативних правил (найкращі 10 за ліфтом)

Наведена таблиця демонструє 10 найкращих асоціативних правил, виявлених у датасеті транзакцій продуктового магазину, відсортованих за метрикою ліфт (*lift*) у спадному порядку. Ліфт є мірою того, наскільки частіше ***антецедент*** (ліва частина правила) та ***консеквент*** (права частина правила) з'являються разом, ніж можна було б очікувати, якби вони були незалежними. Значення ліфта > 1 вказує на позитивну асоціацію між товарами. Нижче подана розрахована розширена статистика найкращих 10 правил за ліфтом *(повна версія блокноту розміщена у курсі на СДН).*

	consequent	**support**	**support**	**confidence**	**lift**	**representativity**\
15180		0.111339	0.013597	0.270408	2.428689	1.0
15189		0.050282	0.013597	0.122120	2.428689	1.0
15181		0.106978	0.013597	0.259804	2.428575	1.0
15188		0.052335	0.013597	0.127098	2.428575	1.0
12800		0.106978	0.010005	0.248408	2.322046	1.0
12797		0.040277	0.010005	0.093525	2.322046	1.0
15179		0.120318	0.013597	0.278947	2.318415	1.0
15190		0.048743	0.013597	0.113006	2.318415	1.0
15193		0.071832	0.013597	0.165109	2.298554	1.0
15176		0.082350	0.013597	0.189286	2.298554	1.0

	leverage	**conviction**	**zhangs_metric**	**jaccard**	**certainty**	**kulczynski**
15180	0.007998	1.218025	0.619400	0.091854	0.178999	0.196264
15189	0.007998	1.081831	0.661957	0.091854	0.075641	0.196264
15181	0.007998	1.206467	0.620721	0.093310	0.171133	0.193451
15188	0.007998	1.085650	0.658702	0.093310	0.078893	0.193451
12800	0.005696	1.188173	0.593239	0.072897	0.158372	0.170966
12797	0.005696	1.058742	0.637549	0.072897	0.055483	0.170966
15179	0.007732	1.219997	0.597810	0.087459	0.180326	0.195977
15190	0.007732	1.072451	0.646451	0.087459	0.067556	0.195977
15193	0.007681	1.111724	0.615642	0.096715	0.100496	0.177197
15176	0.007681	1.131903	0.608665	0.096715	0.116532	0.177197

Приклад аналізу асоціативних правил. Загальні спостереження

Комбінації продуктів: більшість найкращих правил містять комбінації базових продуктів харчування, як-от ковбаса (sausage), цільне молоко (whole milk), інші овочі (other vegetables), йогурт (yogurt), булочки (rolls / buns), сир (curd). Це свідчить про те, що існують певні звичні покупки, коли ці товари часто купуються разом.

Високий ліфт: значення ліфта у цих правилах варіюються від приблизно 2.3 до 2.43, що вказує на досить сильну позитивну асоціацію між антецедентом та консеквентом. Це означає, що наявність антецедента значно підвищує ймовірність придбання консеквента.

Низька підтримка: значення підтримки (support) для цих правил відносно низькі (приблизно 0.013–0.014 для більшості). Це означає, що хоча асоціація між цими комбінаціями продуктів є сильною, самі ці комбінації не є надзвичайно поширеними у всіх транзакціях.

Помірна достовірність: значення достовірності (confidence) варіюються приблизно від 0.11 до 0.28. Це означає, що від 11 до 28 % транзакцій, що містять антецедент, також містять консеквент. Хоча асоціація є сильною (високий ліфт), сама по собі ймовірність купівлі консеквента після купівлі антецедента не є надто високою.

Аналіз окремих правил (з акцентом на ліфт)

{ковбаса, цільне молоко, інші овочі} -> {йогурт, булочки} (ліфт ≈ 2.43): клієнти, які купують ковбасу, цільне молоко та інші овочі, значно частіше купують йогурт та булочки разом, ніж можна було б очікувати випадково.

{йогурт, булочки} -> {ковбаса, цільне молоко, інші овочі} (ліфт ≈ 2.43): аналогічно клієнти, які купують йогурт та булочки, значно частіше купують ковбасу, цільне молоко та інші овочі разом.

{йогурт, інші овочі, булочки} -> {ковбаса, цільне молоко} (ліфт ≈ 2.43): клієнти, які купують йогурт, інші овочі та булочки, значно частіше купують ковбасу та цільне молоко разом.

{ковбаса, цільне молоко} -> {йогурт, інші овочі, булочки} (ліфт ≈ 2.43): клієнти, які купують ковбасу та цільне молоко, значно частіше купують йогурт, інші овочі та булочки разом.

{сир, йогурт} -> {ковбаса, цільне молоко} (ліфт ≈ 2.32): клієнти, які купують сир та йогурт, частіше купують ковбасу та цільне молоко разом.

{ковбаса, цільне молоко} -> {сир, йогурт} (ліфт ≈ 2.32): клієнти, які купують ковбасу та цільне молоко, частіше купують сир та йогурт разом.

{ковбаса, цільне молоко, булочки} -> {йогурт, інші овочі} (ліфт ≈ 2.32): клієнти, які купують ковбасу, цільне молоко та булочки, частіше купують йогурт та інші овочі разом.

{йогурт, інші овочі} -> {ковбаса, цільне молоко, булочки} (ліфт ≈ 2.32): клієнти, які купують йогурт та інші овочі, частіше купують ковбасу, цільне молоко та булочки разом.

{ковбаса, булочки} -> {цільне молоко, інші овочі, йогурт} (ліфт ≈ 2.30): клієнти, які купують ковбасу та булочки, частіше купують цільне молоко, інші овочі та йогурт разом.

{цільне молоко, інші овочі, йогурт} -> {ковбаса, булочки} (ліфт ≈ 2.30): клієнти, які купують цільне молоко, інші овочі та йогурт, частіше купують ковбасу та булочки разом.

Узагальнення

Результати асоціативних правил свідчать про існування певних усталених моделей покупок у цьому продуктовому магазині. Найсильніші асоціації спостерігаються між базовими продуктами, що часто використовуються в повсякденному харчуванні. Хоча ці комбінації не є надзвичайно поширеними у всіх транзакціях (низька підтримка), коли вони трапляються, ймовірність одночасної купівлі всіх включених товарів значно вища, ніж якщо б ці покупки були випадковими.

Приклади практичного застосування

Ці правила можуть бути використані для:

– розміщення товарів у магазині. Розміщення товарів з високим ліфтом поруч може стимулювати імпульсні покупки;

– рекомендаційних систем. Пропонування клієнтам товарів на основі їхнього кошика;

– маркетингових кампаній. Створення цільових акцій та пропозицій на комбінації товарів із сильною асоціацією;

– аналізу кошика покупця. Краще розуміння звичок та потреб клієнтів.

Важливо зазначити, що ці результати є лише найкращими 10 правилами, відсортованих за ліфтом. Для більш повного аналізу потрібно розглянути всі згенеровані правила з різними порогами підтримки та достовірності. Також корисно дослідити правила з нижчим ліфтом, але вищою підтримкою, оскільки вони можуть представляти більш поширені, хоча й менш неочікувані асоціації.

Питання для самоконтролю

1. Опишіть основні характеристики, що описують асоціативне правило: підтримка (support) і вірогідність (confidence).

2. Дайте визначення показника «ліфт», що застосовується під час побудови асоціативних правил.

3. Дайте визначення показника «левередж», що застосовується під час побудови асоціативних правил.

4. Поясніть метод пошуку асоціативних правил Apriory.

5. Опишіть стратегії пошуку правил: використання однакового порогу мінімальної підтримки; зниження порога мінімальної підтримки.

6. Вкажіть правильний варіант. Різниця між спостережуваною частотою, з якою умова й наслідок з'являються спільно, та добутком частот появи умови й наслідку окремо є: левередж; ліфт; поліпшення.

7. Вкажіть правильний варіант. Відношення частоти спостережуваного виконання правила до добутку частот появи умови й наслідку окремо є: левередж; ліфт; поліпшення.

8. Вкажіть правильний варіант. Відношення частоти появи умови в транзакціях, які містять й умову, і наслідок, до частоти появи наслідку загалом є: левередж; ліфт; поліпшення.

9. Що означає термін «аффінітивний аналіз» (affinity analysis)?

10. Що означає типова транзакція щодо асоціативного аналізу взаємодії з торговельного підприємства із клієнтом?

4.6. Прогнозування часових рядів

4.6.1. Методи та інструментарій прогнозування часових рядів

Задачі прогнозування часових рядів

Прогнозування часових рядів вирішують завдання передбачення майбутніх значень на підставі історичних спостережень, де спостереження впорядковані в часі. Основні підходи та задачі:

Короткострокове та довгострокове прогнозування:

– *короткострокове* (хвилини, години, дні) важливе для управління запасами, енергопостачання та трейдингу;

– *довгострокове* (місяці, роки) – для стратегічного планування, капіталовкладень, демографічних прогнозів.

Оцінка компонент часового ряду:

– розклад часового ряду на *тренд, сезонність* та *шуми*, щоб окремо моделювати кожну із компонент (алгоритми ETS, STL);

– ідентифікація *автокореляцій* (алгоритми ARIMA, SARIMA).

Аномалії та втручання:

– виявлення та моделювання *різких змін* (outliers, intervention analysis);

– прогноз з урахуванням *подій* (свята, маркетингові кампанії) за допомогою регресорів у моделях.

Мультирядові прогнози:

– багатовимірні *часові ряди* (Multivariate, VAR), коли декілька показників взаємопов’язані.

Інтеграція зовнішніх чинників:

– використання *регресорів* (погода, ціни сировини) в моделях (SARIMAX, Prophet).

Інструментарій для прогнозування часових рядів поданий у табл. 4.12.

Таблиця 4.12 – Інструментарій для прогнозування часових рядів

Метод / Бібліотека	Опис	Популярні імплементації за допомогою бібліотек
Exponential Smoothing (ETS)	Автоматичне згладжування рівня, тренда і сезонності	statsmodels.tsa.holtwinters
ARIMA / SARIMA	Авторегресія + ковзне середнє + диференціювання	statsmodels.tsa.arima.model
Prophet	Компонентний підхід з простим API для бізнес-даних	fbprophet / prophet
State-space / Kalman Filter	Оцінка прихованих змінних та адаптивне оновлення	statsmodels.tsa.statespace
VAR (Vector AR)	Мультирядові AR-моделі	statsmodels.tsa.vector_ar
LSTM / RNN	Нелінійні моделі глибокого навчання для TS	tensorflow.keras, pytorch
XGBoost / LightGBM	Гібридні моделі із лагованими ознаками	xgboost, lightgbm
scikit-learn	Загальні ML-моделі на лагованих даних	LinearRegression, RandomForest тощо

Рекомендації щодо застосування інструментарію для часових рядів подано у таблиці 4.13.

Таблиця 4.13 – Рекомендації щодо застосування інструментарію для часових рядів

Метод	Рекомендоване призначення
1	2
Exponential Smoothing (ETS)	Просте та швидке прогнозування з трендом і сезонністю, коли потрібно автоматично адаптувати рівень зміни
ARIMA / SARIMA	Аналітичне моделювання автокореляцій та сезонності в однорідних рядах; добре працює на середніх обсягах даних
Prophet	Прогнози для бізнес-даних з нерегулярними подіями (свята, акції), коли потрібен простий інтерфейс налаштування

Продовження таблиці 4.13

1	2
State-space / Kalman Filter	Динамічна оцінка прихованих станів (рівня, тренду) з можливістю адаптації в реальному часі
VAR (Vector Autoregression)	Мультирядове прогнозування, коли потрібно моделювати взаємозалежність між декількома часовими рядами
LSTM / RNN	Застосовується за наявності великих обсягів даних і складних нелінійних залежностей у серії
XGBoost / LightGBM (лаговані ознаки)	Гібридний підхід для врахування нетривіальних взаємодій ознак і екзогенних чинників
ML з лагами (RandomForest, LinearRegression тощо)	Просте тестування кореляцій минулих значень із майбутніми, корисно як baseline або для feature engineering

У цьому розділі розглядаються найбільш популярні алгоритми аналізу та прогнозування часових рядів: авторегресійні ARIMA, SARIMA та моделі експоненційного згладжування Холта-Вінтерса.

4.6.2. ARIMA. Авторегресійна інтегрована модель ковзного середнього

Моделі ARIMA (Autoregressive Integrated Moving Average) є потужними інструментами для прогнозування часових рядів.

Вони мають три основні параметри:

p (порядок авторегресії, AR): кількість лагових значень ряду, що використовуються для прогнозування;

d (порядок диференціювання): кількість разів, скільки потрібно диференціювати ряд, щоб зробити його стаціонарним;

q (порядок ковзного середнього, MA): кількість лагових похибок прогнозування, що використовуються для прогнозування.

Одним із ключових кроків під час побудови моделей прогнозування часових рядів є дослідження автокореляційних функцій (АКФ, Autocorrelation Functions ACF) і часткових автокореляційних функцій (ЧАКФ, Partial Autocorrelation Function PACF).

Роль АКФ у виборі параметрів ARIMA

АКФ може допомогти визначити початкові значення для параметрів *p* та *q*.

Порядок MA (q): якщо АКФ «відрізає» після лага q (тобто кореляція стає незначною після цього), це може свідчити про те, що порядок MA дорівнює q.

Порядок AR (p): хоча часткова автокореляційна функція (ЧАКФ) є більш корисною для визначення p, АКФ також може надати інформацію. Повільне спадання АКФ часто вказує на нестаціонарність, що потребує диференціювання (параметр d).

У моделі **ARIMA(p, d, q)** параметри p (авторегресія) і q (ковзне середнє) часто вибирають, дивлячись на ЧАКФ і АКФ, відповідно:

– ЧАКФ «обрізається» після лагу p (тобто значущі лише перші p лагів), це підказує, який порядок AR слід обрати;

– АКФ «обрізається» після лагу q, це орієнтир для вибору MA.

Виявлення сезонних патернів

Якщо в АКФ спостерігаються піки на регулярних лагах (s, 2s, 3s …), це свідчить про наявність *сезонності* з періодом s (наприклад, щомісячний чи щоденний цикл). Це допомагає обрати модель *SARIMA* або внести сезонні компоненти в ETS.

ETS розшифровується як *Error–Trend–Seasonal:*

E (Error) – тип помилки (Additive або Multiplicative).

T (Trend) – тип тренду (None, Additive, Additive damped або Multiplicative).

S (Seasonal) – тип сезонної компоненти (None, Additive або Multiplicative).

Наприклад, модель *ETS(A, A, A)* означає: **A**dditive Error, **A**dditive Trend, **A**dditive Seasonality.

Розуміння структури часового ряду

Сезонність: АКФ може допомогти виявити сезонність у часовому ряді. Якщо в АКФ спостерігаються значні піки на лагах, що відповідають сезонному періоду (наприклад, кожні 12 місяців), це свідчить про наявність сезонності.

Тренди: повільне спадання АКФ часто вказує на наявність тренду в даних.

Перевірка припущень моделі.

Діагностика моделі

Після підгонки ARIMA / ETS моделі залишки повинні вести себе як «білий шум» – їхня АКФ не повинна показувати значущих автокореляцій:

– якщо в залишках є значна автокореляція, це означає, що модель не повністю врахувала залежності в даних, і її потрібно переглянути;

– якщо в залишках видно структуру (сезонність, автокореляцію), потрібно модифікувати модель (додати лаґи, сезонні компоненти, змінити порядок).

4.6.3. SARIMA. Сезонна авторегресійна інтегрована модель ковзного середнього

Модель SARIMA (Seasonal Autoregressive Integrated Moving Average) – це потужний статистичний метод для аналізу та прогнозування часових рядів, який враховує як *несезонні,* так і *сезонні* залежності, а також тренди. Вона є розширенням моделі ARIMA і додає компоненти для моделювання сезонності [28]. Повна нотація моделі SARIMA виглядає так: **SARIMA (p, d, q) × (P, D, Q)m**

Вибір параметрів для кожної складової

Несезонна складова (p, d, q):

p (порядок авторегресії – AR): кількість минулих (agged) значень часового ряду, які використовуються для прогнозування поточного значення. Наприклад, якщо p = 1, то поточне значення залежить від попереднього;

d (порядок інтегрування / різницювання – I): кількість разів, яку потрібно взяти різниці між послідовними значеннями ряду, щоб зробити його стаціонарним (тобто щоб середнє значення та дисперсія не залежали від часу). Це допомагає усунути тренди;

q (порядок ковзного середнього – MA): кількість минулих помилок прогнозу (різниць між фактичними та прогнозованими значеннями), які використовуються для прогнозування поточного значення.

Сезонна складова (P, D, Q)m:

P (Сезонний порядок авторегресії – Seasonal AR): аналогічно до несезонного p, але для сезонних лагів. Наприклад, якщо дані місячні (m = 12) і P = 1, то поточне значення залежить від значення 12 місяців тому (Y_{t-12}).

D (Сезонний порядок інтегрування / різницювання – Seasonal I): кількість разів, яку потрібно взяти сезонні різниці (наприклад, $Y_t - Y_{t-m}$), щоб усунути сезонну нестаціонарність або сезонний тренд.

Q (Сезонний порядок ковзного середнього – Seasonal MA): аналогічно до несезонного q, але для помилок прогнозу на сезонних лагах.

m (Період сезонності): кількість періодів в одному повному сезонному циклі. Наприклад:

m = 4 для квартальних даних;

m = 12 для місячних даних;

m = 7 для тижневих даних з денною сезонністю;

m = 24 для денних даних з годинною сезонністю (якщо такий цикл є).

Рекомендації щодо використання SARIMA

Модель SARIMA є доцільною, коли часовий ряд демонструє:

Тренд: зростання або спадання з часом (враховується параметром d).

Сезонність: регулярні, передбачувані коливання, що повторюються впродовж фіксованого періоду m (враховується параметрами P, D, Q, m).

Автокореляцію: залежність поточних значень від минулих значень або минулих помилок (враховується параметрами p, q, P, Q).

Як це працює (спрощено):

Стаціонаризація: спочатку до ряду застосовуються сезонні (D) та несезонні (d) операції різницювання, щоб усунути тренди та сезонну нестаціонарність.

Моделювання залежностей: на отриманому стаціонарному ряді будується модель, що враховує:

– несезонні авторегресійні (p) та ковзні середні (q) ефекти;

– сезонні авторегресійні (P) та ковзні середні (Q) ефекти на лагах, кратних періоду m.

Переваги SARIMA:

– здатність моделювати складні часові ряди з трендами та сезонністю;

– добре розроблена теоретична база.

Недоліки SARIMA:

– вимагає, щоб ряд був стаціонарним або міг бути зроблений таким шляхом диференціювання;

– ідентифікація правильних порядків (p, d, q) (P, D, Q)m може бути складним та ітеративним процесом.

Процес побудови моделі на прикладі SARIMA

1. Візуалізація даних: побудова графіка часового ряду для виявлення тренду, сезонності та інших особливостей.

2. Стаціонаризація: перевірка ряду на стаціонарність (наприклад, за допомогою тесту Дікі-Фуллера). Якщо ряд нестаціонарний, застосовується несезонне (d) та/або сезонне (D) диференціювання. Період m визначається з графіка або знань про дані.

3. Ідентифікація порядків (p, q) та (P, Q): аналіз автокореляційної функції (ACF) та часткової автокореляційної функції (PACF) для диференційованого ряду:

– несезонні порядки p, q визначаються за значущими лагами на початку АКФ та ЧАКФ;

– сезонні порядки P, Q визначаються за значущими лагами, кратними періоду m (тобто m, 2m, 3m,…);

4. Оцінка параметрів моделі: використання статистичних методів (наприклад, метод максимальної правдоподібності) для оцінки коефіцієнтів моделі.

5. Діагностика моделі: перевірка залишків (різниць між фактичними значеннями та прогнозами моделі). Залишки повинні бути схожими на білий шум (випадкові, без автокореляції).

6. Прогнозування: використання підібраної моделі для прогнозування майбутніх значень часового ряду.

4.6.4. Моделі експоненційного згладжування

Моделі експоненційного згладжування – це клас методів прогнозування часових рядів, які генерують прогнози на основі зважених середніх минулих спостережень, причому ваги спадають

експоненційно для старіших даних. Це означає, що останні спостереження мають більший вплив на прогноз.

Загальні моделі експоненційного згладжування: (ES) Smoothing

Основна ідея експоненційного згладжування полягає у «згладжуванні» випадкових коливань у часовому ряді для виявлення основної структури (рівень, тренд, сезонність).

Ключові характеристики ES моделей

Згладжувальні параметри (константи): це значення (зазвичай між 0 та 1), які визначають, наскільки швидко спадають ваги для минулих спостережень. Чим ближче параметр до 1, тим більша вага надається останнім спостереженням, і тим швидше модель реагує на зміни.

Компоненти ряду: різні ES моделі розроблені для роботи з різними компонентами часового ряду:

Рівень (Level): середнє значення ряду після усунення тренду та сезонності.

Тренд (Trend): довгостроковий напрямок руху ряду (зростання, спадання або стабільність).

Сезонність (Seasonality): регулярні, повторювані коливання в ряді, пов’язані з певними періодами (наприклад, день тижня, місяць, квартал).

Основні типи моделей експоненційного згладжування

1. Просте експоненційне згладжування *(Simple Exponential Smoothing – SES):*

– використовується для рядів *без тренду та сезонності*;

– моделює лише *рівень* ряду;

– має один згладжувальний параметр α (альфа) для рівня;

– прогноз на майбутнє є просто останнім оціненим рівнем;

2. Подвійне експоненційне згладжування *(Метод Холта / Holt's Linear Trend Method):*

– використовується для рядів **з** *трендом, але без сезонності*;

– моделює *рівень та тренд* (зазвичай лінійний);

– має два згладжувальні параметри: α (альфа) для рівня та β (бета) для тренду;

– прогноз враховує як поточний рівень, так і напрямок тренду.

3. Потрійне експоненційне експоненційне згладжування або Модель Холта-Вінтерса (*Holt-Winters Exponential Smoothing).*

Модель Холта-Вінтерса є розширенням методів експоненційного згладжування, призначеним для часових рядів, які мають *як тренд, так і сезонність*.

Основні компоненти моделі Холта-Вінтерса:

Рівень (l_t): Оцінка середнього значення ряду в момент часу t.

Тренд (b_t): Оцінка нахилу (зростання або спадання) ряду в момент часу t.

Сезонний компонент (s_t): Оцінка сезонного фактора для моменту часу t (або для відповідного періоду в сезоні).

Модель Холта-Вінтерса має *три згладжувальні параметри:*

α (альфа): для рівня;

β (бета): для тренду;

γ (гамма): для сезонного компонента.

Існують два основні варіанти методу Холта-Вінтерса, залежно від характеру сезонності:

Адитивний метод Холта-Вінтерса (Additive Holt-Winters).

Використовується, коли амплітуда сезонних коливань є *приблизно постійною* незалежно від загального рівня часового ряду.

Сезонний компонент *додається* до рівня та тренду для отримання прогнозу.

Рівняння (спрощено):

Рівень: $lt = \alpha(y_t - s_t - m) + (1 - \alpha)(l_{t-1} + b_{t-1})$.

Тренд: $b_t = \beta(l_t - l_{t-1}) + (1 - \beta)b_{t-1}$.

Сезонність: $s_t = \gamma(y_t - l_t) + (1 - \gamma)s_t - m$.

Прогноз на h кроків вперед: $y(t) + h_t = l_t + h \cdot b_t + s_t + h - m$ (або відповідний сезонний індекс).

Тут y_t – фактичне значення, m – довжина сезонного періоду.

Мультиплікативний метод Холта-Вінтерса (Multiplicative Holt-Winters).

Використовується, коли амплітуда сезонних коливань *змінюється пропорційно* до загального рівня часового ряду (тобто сезонні коливання більші, коли рівень ряду вищий, і менші, коли рівень нижчий).

Сезонний компонент *множиться* на суму рівня та тренду.

Рівняння (спрощено):

Рівень: $l_t = \alpha(y_t s_t - m) + (1 - \alpha)(l_t - 1 + b_t - 1)$.

Тренд: $b_t = \beta(l_t - l_t - 1) + (1 - \beta)b_t - 1$.

Сезонність: $s_t = \gamma(y_t / l_t) + (1 - \gamma)s_t - m$.

Прогноз на h кроків вперед: $y^t + h \mid t = (l_t + h \cdot b_t) \cdot s_t + h - m$ (або відповідний сезонний індекс).

Вибір між адитивною та мультиплікативною моделями зазвичай базується на візуальному аналізі графіка часового ряду: якщо сезонні «гойдалки» стабільні за амплітудою, використовується адитивна модель; якщо вони розширюються або звужуються зі зміною рівня ряду, краще підходить мультиплікативна.

Моделі Холта-Вінтерса є популярними та ефективними для прогнозування сезонних даних завдяки своїй відносній простоті та здатності адаптуватися до змін у рівні, тренді та сезонності [28–30].

4.6.5. Приклади аналізу функцій автокореляції (АКФ, ЧАКФ)

Аналіз даних, вибір моделі та параметрів

Підготовка часових динамічних даних є критично важливим етапом перед будь-яким аналізом або прогнозуванням часових рядів. Цей процес охоплює кілька кроків, спрямованих на очищення, трансформацію та структурування даних у формат, придатний для моделювання. Розглянемо основні етапи підготовки на прикладах з використанням бібліотеки *pandas* у Python та візуалізацією за допомогою *matplotlib.*

Підготовка часових рядів до аналізу й моделювання зазвичай складається з таких етапів:

1. Завантаження даних.
2. Дискриптивний огляд та дослідження даних.
3. Попередня обробка даних і коректне представлення міток часу.
4. Обробка пропусків і аномалій.
5. Дослідження автокореляційних функцій АКФ, ЧАКФ.
6. Перевірка стаціонарності та диференціювання.
7. Підбір параметрів моделей та прогнозування.
8. Виведення показників точності та оцінювання якості моделі на тестових даних.

Розглянемо основні етапи використання моделі на прикладі датасету про надходження податку на доходи фізичних осіб (ПДФО) одного із регіонів України 2015–2023 рр.

1. Завантаження даних

У лабораторному практикумі можна використовувати як локальне середовище Python, так і *Google Colab,* у якому файл даних завантажується за допомогою пропрієтарного коду Colab (лістинг 4.29).

```
# Лістинг 4.29
```

Завантаження даних із локального середовища

```
# У запиті вказати локальний шлях до вашого файл даних
.csv
    from google.colab import files
    uploaded = files.upload()
    for fn in uploaded.keys():
      print('User uploaded file "{name}" with length
{length} bytes'.format(
          name=fn, length=len(uploaded[fn])))
```

files.upload повертає словник завантажених файлів. Словник упорядковано за назвами файлів, значення в ньому – це завантажені дані. (Інструментарій завантаження й зберігання даних із зовнішніх джерел у середовищі Colab) [32].

2. Дискриптивний аналіз даних часового ряду

Завантажуємо відповідні компоненти бібліотек для часових рядів і досліджуємо дані створеного датафрейму (*dataframe*) *df_arima* із завантаженого файлу даних (dataset) *pdfo.csv* (лістинг 4.30):

```
# Лістинг 4.30
```

Імпорт відповідні компоненти бібліотек

```
    import pandas as pd
    import numpy as np
    import matplotlib.pyplot as plt
    from statsmodels.tsa.arima.model import ARIMA
    from sklearn.metrics import mean_squared_error,
mean_absolute_error
    from statsmodels.tsa.stattools import adfuller
    from statsmodels.graphics.tsaplots import
plot_acf, plot_pacf
    # Створення датафрейму з іменем  df_arima
    df_arima = pd.read_csv('pdfo.csv')
```

Датафрейм формату pandas із іменем *df_arima* містить надходження податку на доходи фізичних осіб одного із регіонів України із січня 2015 р. по травень 2023 р., усього 101 значення. Усю цю інформацію, а також статистику отримаємо після виконання методів: *df_arima.info()* – стислий звіт про датафрейм; *print(df_arima.head())* – вивід інформації на екран (лістинг 4.31).

df_arima.info() – це метод об'єкта *df_arima*, який надає стислий звіт про датафрейм.

```
                    # # Лістинг 4.31
    Вивід інформації про датасет
    print(df_arima.info()) # стислий звіт про датафрейму
df_arima
    print(df_arima.head())# виводить перші n рядків
датафрейму df_arima
    print(df_arima.describe())# генерує описову
статистику датафрейму df_arima
```

Результат виведення *print(df_arima.info())* дає опис колонок датафрейму, зокрема бачимо, що він вміщує 101 значення *(Non-Null Count)*, має 3 колонки*: id* і *m11010100* – числові і колонка *date* – типу *object.*

```
    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 101 entries, 0 to 100
    Data columns (total 4 columns):
     #   Column     Non-Null Count  Dtype
    ---  ------     --------------  -----
     0   id          101 non-null    int64
     1   date        101 non-null    object
     2   m11010100  101 non-null    float64
    dtypes: float64(1), int64(2), object(1)
```

Власне, числові дані надходжень перебувають у колонці m11010100, а дати надходжень у колонці *date*, які під час завантаження отримали тип *object,* і їх потрібно буде привести до типу дати. Результат виведення *print(df_arima.describe())* дає описову статистику числових колонок датасету:

```
              id          m11010100
count    101.000000   1.010000e+02
mean     51.000000    3.349106e+08
std      29.300171    1.427655e+08
min      1.000000     8.139973e+07
25%      26.000000    2.230621e+08
50%      51.000000    3.506495e+08
75%      76.000000    4.519791e+08
max      101.000000   6.354853e+08
```

Завжди важливим є графічний аналіз даних для розуміння їх розподілу, динаміки тощо. Графічний аналіз автокореляційних функцій (АКФ, ЧАКФ) важливий для розуміння, який метод буде більш ефективним для прогнозування і підбору коефіцієнтів моделей. Для цього потрібно виконати попередню обробку даних. Для відображення часового ряду виконаємо стандартні процедури *plt* (лістинг 4.32):

```
                    # Лістинг 4.32
Візуалізація часового ряду
plt.figure(figsize=(12, 6))
plt.plot(df_arima['m11010100'])
plt.title('Часовий ряд ПДФО')
plt.xlabel('Дата')
plt.ylabel('Значення')
plt.show()
```

Результат виконання вказано на рис. 4.34, щодо якого досить чітко варто зазначити про наявність тренду, а піки, періодичні піки можуть вказувати на сезонні явища. Наступним завданням є дослідження цих показників і врахування їх у прогнозній моделі.

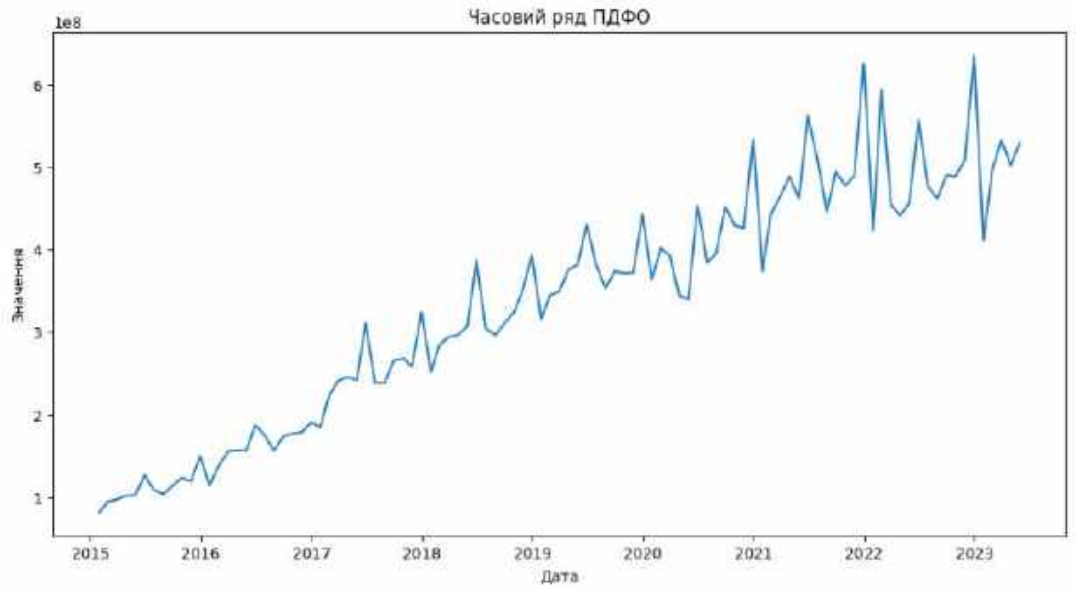


Рисунок 4.34 – Графік числового ряду надходжень ПДФО (лістинг 4.32)

3. Попередня обробка даних і коректне представлення міток часу

Для коректного моделювання потрібна попередня обробка даних.

Рядок 1. Цей рядок коду перетворює стовпець під назвою *'date'* у датафреймі *df_arima* на тип даних *datetime64[ns]*. Це тип даних у pandas, призначений для представлення дат та часу.

```
                    # Лістинг 4.33
    Попередня обробка даних
    df_arima['date'] = pd.to_datetime(df_arima['date'])
# рядок 1
    df_arima.set_index('date', inplace=True)  # рядок 2
    df_arima.sort_index(inplace=True)         # рядок 3
    df_arima = df_arima[['m11010100']]        # рядок 4
    df_arima = df_arima.resample('M').mean()  # рядок 5
```

Для правильної обробки часових рядів потрібно, щоб дати були представлені як об’єкти *datetime.* Це дає змогу *pandas* виконувати операції, специфічні для часу, як-от сортування за часом, вибірка за діапазоном дат, передискретизація (resampling) тощо. Якщо дати залишаються у вигляді рядків, їх сортування може бути неправильним (наприклад, «2024-01-02» може бути відсортовано перед «2024-01-10», оскільки «1» менше «10»).

Приклад. У нашому датафреймі є стовпець «*date*», у якому дати подані у форматі «2024-01-01», «2024-01-02», «2024-01-03». Цей рядок коду перетворить їх на об'єкти *datetime,* які pandas може правильно інтерпретувати як дати.

Рядок 2. Цей рядок коду встановлює стовпець *'date'* як індекс датфрейму *df_arima.* Індекс у *pandas* – це мітка або мітки, які використовуються для ідентифікації рядків:

– для часових рядів часто корисно мати дату як індекс. Це робить вибірку, фільтрацію та інші операції з часом більш зручними та ефективними;

– багато функцій для аналізу часових рядів у pandas та інших бібліотеках (наприклад, statsmodels) очікують, що часовий ряд буде індексований за датою.

Рядок 3. Цей рядок рядок коду сортує датафрейм *df_arima* за його індексом, яким тепер є стовпець 'date'. Для аналізу часових рядів критично важливо, щоб дані були впорядковані за часом. Моделі часових рядів покладаються на часову послідовність даних.

Рядок 4. Цей рядок коду вибирає лише стовпець «m11010100» з даними про податкові надходження (ПДФО) створює новий датафрейм, що містить лише цей стовпець. Цей рядок коду дає змогу виділити цю змінну, що спрощує подальший аналіз.

Рядок 5. Цей рядок коду передискретизує (resample) часовий ряд до місячної частоти. Передискретизація означає зміну частоти часових рядів (наприклад, з щоденної на місячну, з місячної на річну). У цьому випадку «M» вказує на місячну частоту, а *.mean()* означає, що для кожної групи (місяця) обчислюється середнє значення.

4. Обробка пропусків і аномалій

У нашому датасеті пропуски й аномалії відсутні, але зазвичай пропуски при прогнозуванні часових рядів заповнюють лінійною інтерполяцію. Тобто пропуски заповнюються значеннями, що лежать на прямій лінії між двома найближчими дійсними точками. Інтерполяція добре підходять для даних, що змінюються відносно плавно, наприклад:

– лінійна інтерполяція:

```
df_arima.interpolate(method='linear')
```

– поліноміальна інтерполяція заданого порядку:

```
df_arima.interpolate(method='polynomial',
order=...)
```

У нашому прикладі відсутні пропуски і аномалії, тому ця процедура не виконується.

5. Дослідження автокореляційних функцій АКФ, ЧАКФ

Одним із ключових кроків під час побудови моделей прогнозування часових рядів є дослідження автокореляційних функцій (АКФ, Autocorrelation Functions ACF) і часткових автокореляційних функцій (ЧАКФ, Partial Autocorrelation Function PACF). Багато методів прогнозування часових рядів (наприклад, ARIMA) покладаються на наявність автокореляції. Якщо дані мають значну автокореляцію, це означає, що попередні значення можуть бути корисними для прогнозування майбутніх. Якщо ж автокореляція незначна, інші методи (наприклад, методи, що враховують зовнішні фактори) можуть бути більш доречними.

Візуалізація АКФ, ЧАКФ

Візуалізацію АКФ (ACF), ЧАКФ (PACF) дають функції *plot_acf, plot_pacf* бібліотеки *statsmodels.graphics.tsaplots* (лістинг 4.34).

```
                    # Лістинг 4.34
    Візуалізація АКФ (ACF), ЧАКФ (PACF)
    from statsmodels.graphics.tsaplots import
plot_acf, plot_pacf
    # Автокореляційна функція (ACF)
    plot_acf(df_arima['m11010100'], lags=20)
    plt.title('Автокореляційна функція (ACF)')
    plt.show()
    # Часткова автокореляційна функція (PACF)
    plot_pacf(df_arima['m11010100'], lags=20)
    plt.title('Часткова автокореляційна функція (PACF)')
    plt.show()
```

df_arima['m11010100']: це перший аргумент функції *plot_acf*. Він вказує, який часовий ряд потрібно проаналізувати.

lags = 20: це другий аргумент функції *plot_acf*. Він визначає кількість лагів (зсувів у часі), для яких буде обчислено та відображено автокореляцію. Лаг 1 означає кореляцію між значенням

ряду в момент часу t і значенням в момент часу t-1, лаг 2 – між t і t-2 тощо. *lags = 20* означає, що графік покаже кореляцію для перших 20 лагів (рис. 4.35).

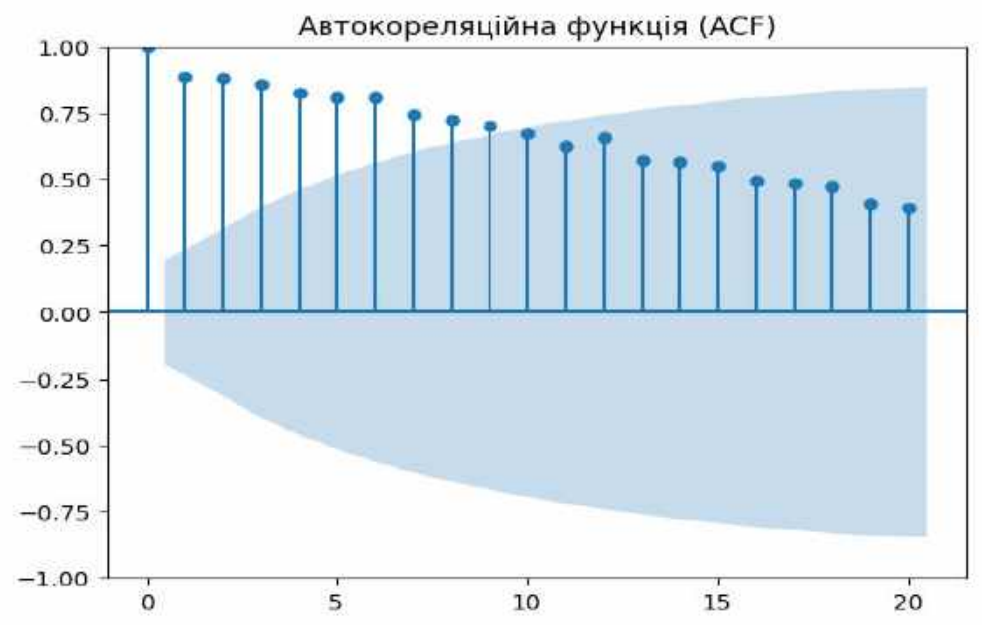


Рисунок 4.35 – Графік автокореляційнох функції (ACF) (лістинг 4.34)

На графіку (рис. 4.35) показано результат виведення автокореляційної функції (АКФ) датфрейму *df_arima,* на якому можна відмітити:

Аномально високі значення на малих лагах:

– при лагу 0 автокореляція завжди = 1.0 (ряд ідеально корелює з самим собою);

– на лагах 1–3 вона ≈ 0.9–0.8 – це надзвичайно високі кореляції.

Повільне спадання («довгий хвіст»):

– замість швидкого «обрізання» до нуля, коефіцієнти кореляції з лагами зменшуються поступово (0.9 → 0.4 за 20 кроків);

– це класична ознака нестаціонарного ряду з трендом: значення сильно корелюють із досить віддаленим минулим.

Відсутність явно виражених сезонних піків:

– якщо б був виразний сезонний цикл (наприклад, період 12), ми б побачили піки на лагах ≈ 12, 24 тощо. Тут же спад є монотонним.

Висновки для моделювання

Ряд нестаціонарний: повільний спад АКФ свідчить про наявність тренду або змінного рівня. Перш ніж застосовувати ARMA / ARIMA, потрібно *продиференціювати* ряд (однократне різницеве перетворення), щоб ліквідувати тренд і отримати стаціонарний ряд.

Після диференціювання (перетворення $y_t' = y_t - y_t - 1)$ АКФ має показати швидше падіння (хвіст), а ЧАКФ допоможе обрати порядок AR-компоненти.

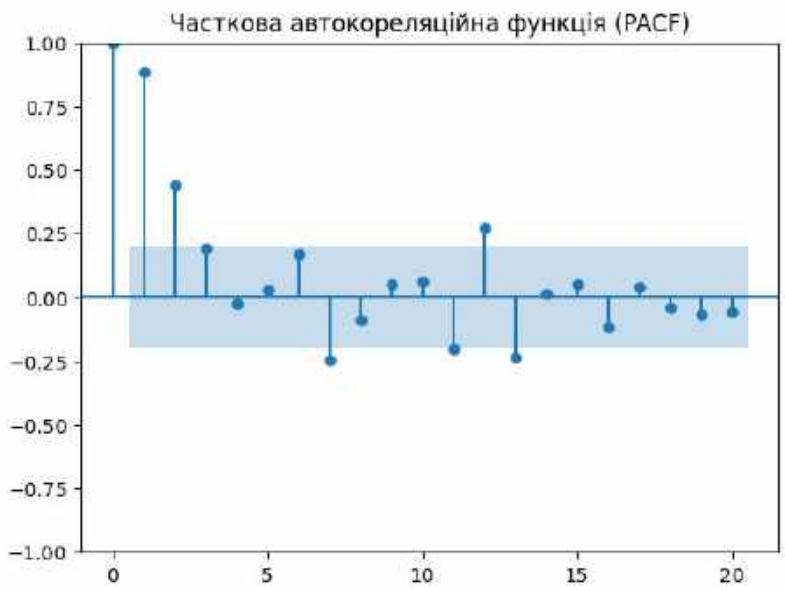


Рисунок 4.36 – Графік частинної автокореляційнох функції (PACF) (лістинг 4.34)

На графіку часткової автокореляційної функції (ЧАКФ, PACF) (рис. 4.36) можна виділити такі особливості:

Великий піковий лаг-1 (~0.9):

– часткова автокореляція на лагу 1 досить висока (≈0.9), що вказує на сильний прямий зв’язок y_t із попереднім значенням y_{t-1};

– це сигнал, що в моделі ARIMA варто розглянути компонент $p \geq 1$.

Значущий піковий лаг-2 (~0.45):

– другий лаг також має помітну часткову автокореляцію (~0.45), отже після першого відрізка автокореляції ще залишається «хвіст»;

– це може вказувати на доцільність AR(2) або ARIMA(p, 1,…) з $p = 2$.

Лаг-3 і далі …:

– ЧАКФ після лагу 2 спадає майже до нуля (за межами довірчого інтервалу) й флуктує близько нуля;

– вибір моделей ARIMA без великої кількості AR-компонентів буде обґрунтованим (достатньо p = 1 або p = 2).

Відсутність явно виражених сезонних піків:

– лаги, кратні сезонному періоду (наприклад, 12), не демонструють значущих часткових автокореляцій;

– можна спробувати модель SARIMA, яка може бути точнішою, але спробуємо для початку модель ARIMA.

Рекомендація для моделі ARIMA:

*Параметр **d:*** ряд нестаціонарний, d = 1.

*Параметр **p:*** початково p = 1 або 2, враховуючи значущі лаги в ЧАКФ.

*Параметр **q:*** визначається за ACF залишків після диференціювання (швидке падіння в АКФ).

Тобто для першого прогнозування можна спробувати ARIMA(1, 1, 0) або ARIMA(1, 2, 1), оцінити інформаційні критерії (AIC / BIC) та залишкові автокореляції, а далі коригувати модель.

6. Перевірка стаціонарності та диференціювання

Тест на стаціонарність Дікі-Фуллера

Одним із найпоширеніших способів перевірити, чи є часовий ряд *стаціонарним* (тобто чи не змінюються з часом статистичні властивості часового ряду: середнє, дисперсія, автокореляція) є *Тест Дікі-Фуллера (Augmented Dickey-Fuller, ADF*), і який можна використати із бібліотек *statsmodels.*

```
from statsmodels.tsa.stattools import adfuller
```

Сутність і важливість тесту

Нульова гіпотеза (H_0): ряд має одиничний корінь, ряд *нестаціонарний* (містить тренд або змінний рівень).

Альтернативна гіпотеза (H_1): ряд *стаціонарний.*

Стаціонарність важлива, тому що більшість класичних моделей (ARIMA, SARIMA, ETS) вимагають, щоб ряд був стаціонарним; інакше їхні оцінки можуть бути упередженими або непридатними для прогнозування. У лістингу 4.35 наведено код застосування методу тесту Дікі-Фуллера та оцінок ряду щодо стаціонарності:

```
# Лістинг 4.35
```

Перевірка стаціонарності за допомогою тесту Дікі-Фуллера

```
result_adf = adfuller(df_arima['m11010100'])
print('Результати тесту Дікі-Фуллера:')
print(f'ADF Statistic: {result_adf[0]}')
print(f'p-value: {result_adf[1]}')
print('Critical Values:')
for key, value in result_adf[4].items():
    print(f'\t{key}: {value}')
# Прийняття рішення щодо стаціонарності
if result_adf[1] <= 0.05:
    print("Часовий ряд є стаціонарним (відхиляємо нульову гіпотезу).")
else:
    print("Часовий ряд не є стаціонарним (не можемо відхилити нульову гіпотезу). Потрібне диференціювання.")
```

Результати тесту Дікі-Фуллера:

```
ADF Statistic: -1.7441734604943453
p-value: 0.40850859853778665
Critical Values:
    1%: -3.506944401824286
    5%: -2.894989819214876
    10%: -2.584614550619835
Часовий ряд не є стаціонарним (не можемо відхилити нульову гіпотезу). Потрібне диференціювання.
```

Пояснення результату:

ADF Statistic = –1.7442

Critical Values показують пороги для відмови від H_0 на різних рівнях значущості:

на рівні 1 % потрібна ADF ≤ –3.5069;

на рівні 5 % потрібна ADF ≤ –2.8949;

на рівні 10 % потрібна ADF ≤ –2.5846.

p-value = 0.4085 (≈ 40.9 %) значно вище звичних порогів 0.01, 0.05 або 0.10.

Оскільки ADF Statistic (–1.7442) **не** є меншою (більш від'ємною) за жодну з critical values і p-value значно перевищує 0.05, **не** можемо відкинути нульову гіпотезу.

Висновок

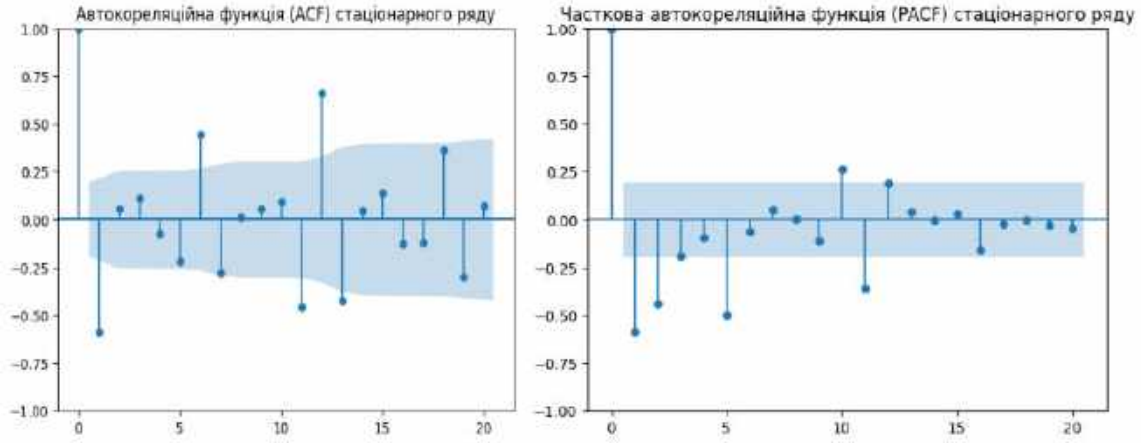
Ряд нестаціонарний. Його поведінка (наприклад, тренд або змінна варіативність) потребує усунення, перш ніж застосовувати ARIMA / SARIMA, ETS чи інші лінійні моделі.

Потрібне диференціювання (наприклад, перший порядок: $y_t' = y_t - y_{t-1}$), щоб зробити ряд стаціонарним. Після диференціювання тест АКФ має повторити з меншим p-value та ADF Statistic, який лежатиме лівіше від критичних значень.

Стаціонарний ряд дає змогу будувати коректні моделі прогнозування з надійними інтерпретаціями й точними довірчими інтервалами.

У блокноті до ЛР наведено код для дифенціювання ряду. Стаціонарний ряд отриманий після двох процедур диференціації, тобто вибраний порядок диференціювання (d) для ARIMA: 2.

На рис. 4.37 показані АКФ і ЧАРФ після другого етапу диференціювання.



а) б)

Рисунок 4.37 – Графіки автокореляційних функцій після диференціювання

а) АКФ; б) ЧАКФ

Інтерпретація результатів графіка АКФ (рис. 4.37, а)

Лаг 1: спостерігається значуща негативна автокореляція (приблизно 0.55). Це означає, що якщо поточне значення високе, попереднє значення, ймовірно, було низьким, і навпаки.

Лаги 2, 3, 4, 5: є менші, але все ще статистично значущі автокореляції. Наприклад, на лагу 3 є позитивна кореляція (приблизно +0.1), а на лагу 5 – негативна (майже –0.2).

Лаг 6: спостерігається досить сильна позитивна автокореляція (приблизно +0.45). Це може вказувати на певну циклічність або сезонність з періодом 6.

Лаг 12: найбільш виражена позитивна автокореляція після лагу 0 (майже +0.7). Це досить сильний показник сезонності з періодом 12. Якщо дані, наприклад, місячні, це вказують на річну сезонність, значення поточного місяця сильно корелюють зі значеннями того самого місяця минулого року.

Інші значущі лаги: є й інші статистично значущі автокореляції на лагах 7 (негативна), 13 (негативна), 14 (позитивна), 15 (позитивна), 17 (позитивна) та 19 (позитивна). Те, що значущі кореляції не згасають швидко, а продовжуються на великих лагах і мають певну структуру (особливо піки на лагах 6 та 12), є важливим.

Синя затінена область (довірчий інтервал): ця сфера представляє довірчий інтервал (зазвичай 95 %). Якщо стовпець автокореляції виходить за межі цієї сфери, це означає, що кореляція на цьому лагу є статистично значущою (тобто малоймовірно, що вона виникла випадково, і вона, ймовірно, не дорівнює нулю).

Загальні висновки з графіка АКФ

Наявність автокореляції: Часовий ряд є автокорельованим, оскільки багато коефіцієнтів автокореляції є статистично значущими. Це означає, що минулі значення ряду містять інформацію, яка може бути використана для прогнозування майбутніх значень.

Можлива сезонність: Високі значущі піки на лагах 6 та особливо 12 вказують на наявність сезонних компонентів у даних. Якщо період вимірювання – місяць, то це піврічна та річна сезонність.

Структура залежності: АКФ не обривається різко після кількох лагів (як це було б для чистого процесу ковзного середнього MA(q). Натомість вона повільно згасає з вираженими сезонними піками. Це може вказувати на те, що для моделювання цього ряду можуть підійти моделі типу AR (авторегресійні), ARMA (авторегресійні ковзного середнього) або, враховуючи сезонність, SARIMA (сезонні авторегресійні інтегровані ковзного середнього).

Стаціонарність: АКФ для стаціонарного ряду зазвичай швидко згасає до нуля (або коливається навколо нуля в межах

довірчого інтервалу) для несезонних рядів. У вашому випадку, хоча його приймаємо як стаціонарний, наявність сильних сезонних піків є домінуючою характеристикою. Згасання несезонних кореляцій відбувається, але сезонні компоненти залишаються сильними.

Цей графік АКФ надає важливу інформацію для ідентифікації відповідної моделі часового ряду (наприклад, для прогнозування). Наявність значущих автокореляцій, особливо сезонних, є ключовим моментом.

Інтерпретація результатів графіка ЧАКФ (рис. 4.37, б)

Лаг 1: Досить значуща негативна часткова автокореляція (приблизно 0.58). Це вказує на сильну пряму негативну залежність між поточним значенням та значенням попереднього періоду.

Лаг 2: Значуща негативна часткова автокореляція (приблизно –0.45). Це прямий зв'язок між Yt та Yt – 2 після усунення впливу Yt – 1.

Лаг 3: Значуща негативна часткова автокореляція (майже – 0.18).

Лаг 4: Значуща негативна часткова автокореляція (майже – 0.1).

Лаг 5: Досить значуща негативна часткова автокореляція (приблизно –0.5). Цей пік є надто великим і може вказувати на важливий авторегресійний вплив на цьому лагу.

Лаги 6, 7, 8: Менші, але все ще статистично значущі часткові автокореляції (негативна на лагу 6 та 8, позитивна на лагу 7).

Лаги 10, 11, 12: Спостерігаються значущі часткові автокореляції: позитивна на лагу 10 (майже +0.27), негативна на лагу 11 (майже –0.35) та позитивна на лагу 12 (майже +0.2). Наявність значущих піків на цих вищих лагах може вказувати на сезонні авторегресійні ефекти, якщо, наприклад, період сезонності становить приблизно 12.

Подальші лаги: Після лагу 12 більшість значень часткової автокореляції перебувають близько до меж довірчого інтервалу або всередині нього, що вказує на їх меншу статистичну значущість.

Загальні висновки з графіка ЧАКФ

Ідентифікація моделі AR(p): ЧАКФ зазвичай «обривається» (значення стають статистично незначущими) після лагу p для чистого процесу AR(p). У вашому випадку графік не показує чіткого обриву після невеликої кількості лагів. Є значущі значення на лагах 1, 2, 3, 4, 5, а потім знову на лагах 10, 11, 12.

Складний авторегресійний процес: Наявність кількох значущих піків на різних лагах (особливо великий пік на лагу 5 та група піків навколо лагу 12) свідчить про те, що часовий ряд, ймовірно, має складну авторегресійну структуру. Це може бути авторегресійний процес вищого порядку або охоплювати сезонні авторегресійні компоненти.

Можлива сезонність: Значущі піки на лагах, близьких до типових сезонних періодів (наприклад, лаг 12 для річної сезонності за місячними даними), можуть вказувати на необхідність включення сезонних AR-членів у модель (наприклад, у моделі SARIMA).

Рекомендації щодо використання інформації ЧАКФ разом з АКФ

– якщо ЧАКФ обривається після лагу p, а АКФ згасає повільно, це типово для моделі AR(p);

– якщо АКФобривається після лагу q, а ЧАКФ згасає повільно, це типово для моделі MA(q);

– якщо і АКФ, і ЧАКФ згасають повільно (не обриваються), це може вказувати на модель ARMA(p, q);

– наявність значущих піків на сезонних лагах в АКФ та/або ЧАКФ вказує на потребу в сезонній моделі (SARIMA).

7. Підбір параметрів моделей та прогнозування

У нашому випадку, оскільки ЧАКФ має кілька значущих піків і не обривається чітко, це свідчить про наявність авторегресійних компонентів, можливо, охоплюючи сезонні. Для точного визначення моделі потрібно аналізувати цей графік ЧАКФ разом із графіком АКФ.

4.6.6. Приклади використання моделей ARIMA, SARIMA

Використання моделі ARIMA

Отже, за результатами аналізу можемо спробувати модель ARIMA(1, 2, 1), але враховуючи наявність тренду й сезонних компонентів, можемо очікувати, що модель не зможе врахувати ці важливі компоненти.

Відповідно до результатів аналізу АКФ, ЧАКФ вибрані такі параметри моделі ARIMA:

p (порядок авторегресії, AR): кількість лагових значень ряду, що використовуються для прогнозування: **значення 1**;

d (порядок диференціювання): кількість разів було продиференційовано ряд, щоб зробити його стаціонарним: **значення 2;**

q (порядок ковзного середнього, MA): кількість лагових похибок прогнозування, що використовуються для прогнозування: **значення 1.**

Базовий код для прогнозування ARIMA показано в лістингу 4.36. Параметри моделі задаються в *order_arima = (1, 2, 1).*

```
                    # Лістинг 4.36
    Базовий код для прогнозуання ARIMA (1, 2, 1)
order_arima = (1, 2, 1)
    # Перенавчання моделі ARIMA з визначеним порядком
    print("\nНавчання моделі ARIMA з новим
порядком...")
    try:
        model_arima = ARIMA(df_arima['m11010100'],
order=order_arima)
        model_arima_fit = model_arima.fit()
        print("Модель ARIMA успішно навчена.")
```

Результат прогнозування за моделлю ARIMA(1, 2, 1) показаний на рис. 4.38.

Не будемо оцінювати якість, точність та основні меторики моделі ARIMA(1, 2, 1), тому що графічний аналіз показує, що модель не враховує сезонні явища, і, очевидно, для такого типу задач, як прогнозування податкових надходжень із явно вираженими сезонними явищами, більше підійдуть моделі SARIMA або моделі експоненційного згладжування.

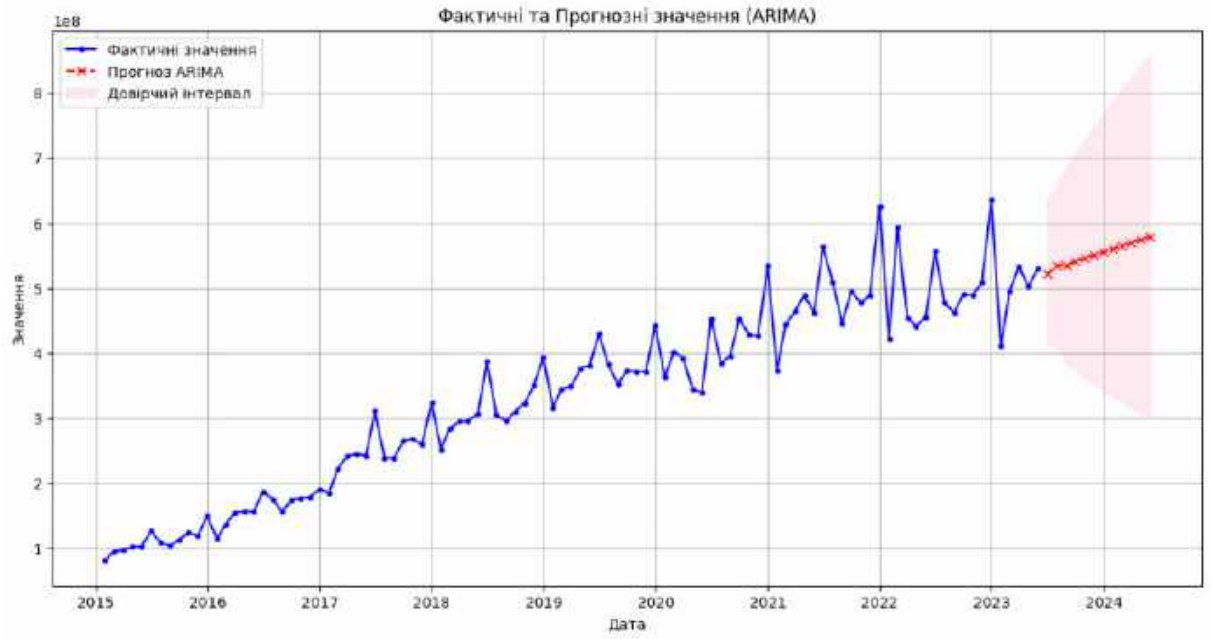


Рисунок 4.38 – Графічний результат прогнозування
за моделлю ARIMA(1, 2, 1)

Використання моделі SARIMA

На основі аналізу АКФ, ЧАКФ після *другого диференціювання* можна зробити такі висновки:

– обидва графіки показують залишкову структуру після другого диференціювання, що вказує на наявність як несезонних, так і сезонних компонентів, які потрібно змоделювати;

– диференціювання допомогло досягти стаціонарності, принаймні частково.

Визначення сезонного періоду (m):

АКФ: спостерігаються дуже значні позитивні сплески на лагах 6 (майже +0.45) та 12 (майже +0.7), а також менший, але значущий позитивний сплеск на лагу 18. Найсильніший сплеск на лагу **12.**

ЧАКФ: є значущий позитивний сплеск на лагу 12 (приблизно +0.2) та менші значущі сплески навколо нього (лаг 10 позитивний, лаг 11 негативний).

Висновок по **m**: Найбільш виражена періодичність спостерігається на лагу 12. Тому рекомендується сезонний період m = 12. Сплеск на лагу 6 може бути гармонікою або вказувати на півріч-ну сезонність, але зазвичай починають з домінуючого періоду.

Визначення несезонних параметрів (p, q) та сезонних (P, D, Q)

Несезонне диференціювання ***(d)***

За умовою, дані аналізуються *після другого диференціювання*, отже, **d = 2**, проте можна починати дослідження прогнозну модель із значенням **1**.

Сезонне диференціювання ***(D)***

Навіть після d=2, ACF демонструє досить сильний сплеск на лагу 12 (сезонний лаг). Це вказує на те, що сезонна нестаціонарність, ймовірно, все ще присутня. Тому рекомендується *одне сезонне диференціювання:* D = 1.

Сезонний порядок ***MA (Q)***

АКФ: На лагу m = 12 є досить значний позитивний сплеск. Якщо розглядати сезонні лаги (12, 24,…), цей сплеск на лагу 12 є домінуючим і не схоже, щоб ACF швидко затухала на подальших сезонних лагах (хоча їх не так багато на графіку). Це типово для сезонного MA компонента.

Рекомендується Q = 1.

Сезонний порядок ***AR (P)***

ЧАКФ: На лагу m = 12 є значний позитивний сплеск. Після цього лагу для наступних сезонних лагів (наприклад, 24, якого немає на графіку), очікується «обрив» ЧАКФ для моделі сезонного AR.

Рекомендується P = 1.

Несезонний порядок ***MA (q)***

АКФ: Спостерігається значний негативний сплеск на лагу 1 (приблизно –0.55). Також є значущі сплески на лагах 2, 3, 4, 5. ACF не обривається чітко після одного-двох лагів, а скоріше повільно затухає або має складну структуру для несезонних лагів. Наявність значного сплеску на лагу 1 часто вказує на $q \geq 1$.

Несезонний порядок ***AR (p)***

ЧАКФ: Спостерігаються значні негативні сплески на лагах 1 (приблизно –0.58), 2 (приблизно –0.45) та 5 (приблизно –0.5). Також значущі менші сплески на лагах 3, 4. PACF також не обривається чітко після одного-двох лагів для несезонних компонентів. Наявність значного сплеску на лагу 1 часто вказує на $p \geq 1$.

Коли і АКФ, і ЧАКФ повільно затухають або мають складну структуру для несезонних лагів, це часто вказує на змішану модель ARMA(p, q). Поширеним початковим вибором є $p = 1$ та $q = 1$.

На основі вищенаведеного аналізу, рекомендовані початкові параметри для моделі SARIMA (p, d, q)(P, D, Q)m.

Несезонні параметри:

p = 1 (порядок AR);

d = 2 (порядок несезонного диференціювання, задано);

q = 1 (порядок MA).

Сезонні параметри:

P = 1 (порядок сезонного AR);

D = 1 (порядок сезонного диференціювання);

Q = 1 (порядок сезонного MA);

m = 12 (сезонний період).

Отже, рекомендована початкова модель:

SARIMA (1, 2, 1)(1, 1, 1)12, проте доцільним буде почати досліджувати простішу модель SARIMA (1, 1, 1)(1, 1, 1)12.

Базовий код для моделі SARIMA (1, 1, 1)(1, 1, 1)12, яка показала кращу точність на навчальні й вибірці, приведено в лістингу 4.37.

```
                    # Лістинг 4.37
    Модель SARIMA (1,1,1)(1,1,1)12
    order_sarima = (1, 1, 1)  # (p, d, q)
    seasonal_order_sarima = (1, 1, 1, 12) # (P, D, Q, s)
    print(f"\nВизначений порядок SARIMA:
{order_sarima}")
    print(f"Визначений сезонний порядок SARIMA:
{seasonal_order_sarima}")
    # Навчання моделі SARIMA
    print("\nНавчання моделі SARIMA...")
    try:
        model_sarima = SARIMAX(df_arima['m11010100'],
                       order=order_sarima,
                       seasonal_order=seasonal_order_
sarima,
                       enforce_stationarity=False, #
Може бути True
                       enforce_invertibility=False) #
Може бути True
```

```
        model_sarima_fit = model_sarima.fit()
        print("Модель SARIMA успішно навчена.")
        print(model_sarima_fit.summary()) # Виведення
зведення моделі
```

Метрики на навчальних даних:

```
MSE (навчальні): 796684019398403.75
MAE (навчальні): 21490104.52
RMSE (навчальні): 28225591.57
MAPE (%) (навчальні): 7.88%
```

На рис. 4.39 показано графік моделі SARIMA (1, 1, 1) (1 ,1, 1)12 із прогнозними значеннями та довірчим інтервалом на 12 місяців вперед.

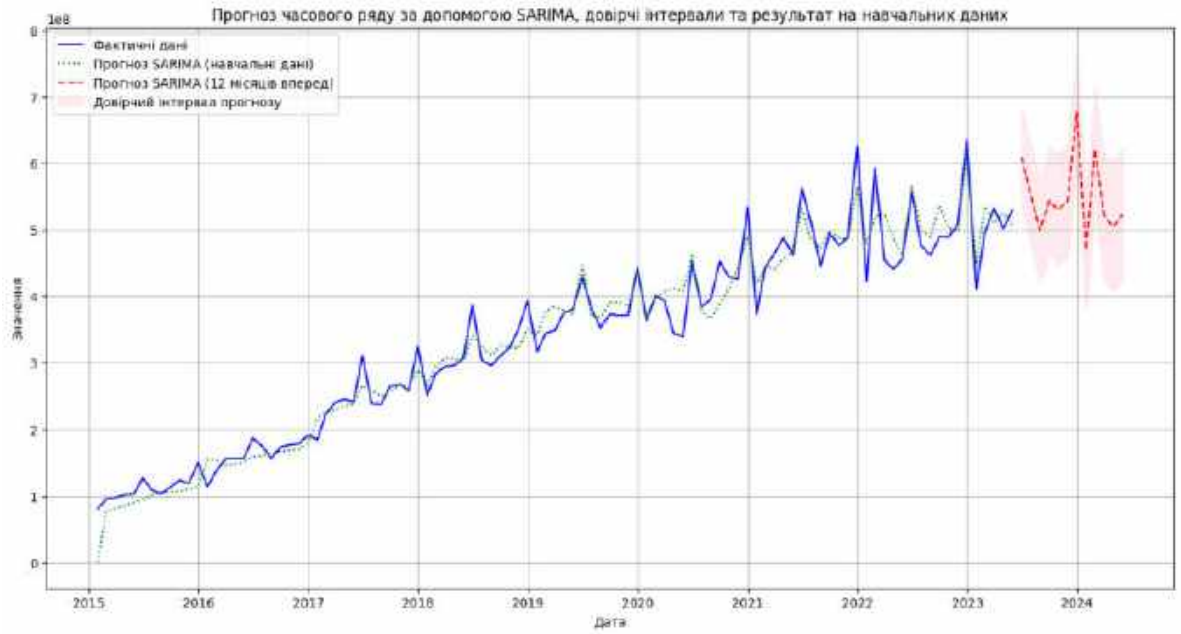


Рисунок 4.39 – Графік, отриманий на основі моделі SARIMA (1 ,1, 1)(1, 1, 1)12

Як видно із результатів оцінки відносної середньої помилки, MAPE (%) на навчальних даних показала 7.88 %, це непоганий результат.

Підбір кращих параметрів для моделі SARIMA також можна виконувати за допомогою спеціалізованих бібліотек, які автоматично перебирають різні комбінації параметрів (p, d, q)(P, D, Q)m та обирають найкращу модель на основі інформаційного критерію

(наприклад, AIC – інформаційний критерій Акаіке або BIC – баєсівський інформаційний критерій).

Найпопулярнішою бібліотекою для цього в Python є *pmdarima* (раніше відома як *pyramid-arima*). Вона надає функцію *auto_arima,* яка автоматизує цей процес.

4.6.7. Приклад використання моделі Холта-Вінтерса

Як уже зазначалось, моделі Холта-Вінтерса належать до *сімейства потрійного експоненційного згладжування.* Попередній аналіз АКФ та ЧАКФ дав змогу визначити в нашому файлі ПДФО (pdfo.scv) наявність сезонних явищ (рис. 4.34).

Тренд: на графіку (рис. 4.34) чітко видно висхідний тренд. Загальний рівень значень ПДФО зростає з часом (з 2015 по 2023 рр.).

Сезонність: також спостерігається сезонний компонент. Є регулярні коливання всередині кожного року. Наприклад, схоже, що є піки та спади, які повторюються щорічно.

Характер сезонності: важливо звернути увагу, як змінюється амплітуда сезонних коливань. На нашому графіку видно, що розмах сезонних коливань збільшується разом зі зростанням загального рівня ряду. Тобто коли середні значення ПДФО вищі, то й різниця між піками та спадами сезону стає більшою в абсолютному вираженні. Це вказує на *мультиплікативну сезонність*. Якби сезонні коливання залишалися приблизно однаковими за величиною незалежно від рівня тренду, це була б *адитивна сезонність.*

Вибір моделі Холта-Вінтерса

Модель Холта-Вінтерса призначена для даних з трендом та сезонністю. Існує два основних типи цієї моделі, що відрізняються способом врахування сезонності.

Адитивна модель Холта-Вінтерса: підходить, коли амплітуда сезонних коливань є сталою і не залежить від рівня часового ряду (4.34):

$$Y_t = Рівень_t + Тренд_t + Сезонність_t + Помилка_t\,. \qquad (4.34)$$

Мультиплікативна модель Холта-Вінтерса: підходить, коли амплітуда сезонних коливань змінюється пропорційно до рівня часового ряду (тобто сезонність є відсотком від тренду) (4.35):

$$Yt = (Рівень_t + Тренд_t) \times Сезонність_t + Помилка_t . \quad (4.35)$$

Потрібно зазначити, що залежно від виду часового ряду модель може бути не суто *адитивною* чи *тультиплікативною,* а *комбінованою:* адитивний тренд, адитивна сезонність; адитивний тренд, мультиплікативна сезонність; мультиплікативний тренд, адитивна сезонність; мультиплікативний тренд, мультиплікативна сезонність.

Враховуючи наявність чіткого майже тренду та вираженої сезонності, амплітуда якої зростає разом із рівнем ряду, найбільш схожою моделлю для нашого часового ряду ПДФО буде *модель Холта-Вінтерса (Адитивний тренд, Мультиплікативна сезонність)*.

Однією із великих перерв алгоритму *Холта-Вінтерса* є автоматичне оцінювання й підгонка параметрів моделі при мінімізації похибок*:* α (альфа): для рівня; β (бета): для тренду; γ (гамма): для сезонного компонента.

Базовий код для застосування моделі *Холта-Вінтерса* показаний лістингу 4.38.

```
                    # Лістинг 4.38
    Холта-Вінтерса:
    #(Адитивний тренд, Мультиплікативна сезонність)
    # код для алгоритму Холта Вінтерса
    from statsmodels.tsa.holtwinters import
ExponentialSmoothing
    # Виберіть ряд даних для Холта-Вінтерса
    time_series_hw = df_arima['m11010100']
    trend_hw = 'add' # Адитивний тренд
    seasonal_hw = 'mul' # Мультиплікативна сезонність
    seasonal_periods_hw = 12 # 12 місяців у році
    # Навчання моделі Холта-Вінтерса
    print("\nНавчання моделі Холта-Вінтерса...")
    try:
        model_hw = ExponentialSmoothing(
            time_series_hw,
            trend=trend_hw,
```

```
            seasonal=seasonal_hw,
            seasonal_periods=seasonal_periods_hw
        )
        model_hw_fit = model_hw.fit()
        print("Модель Холта-Вінтерса успішно навчена.")
        print(model_hw_fit.summary()) # Виведення
зведення моделі
```

Метрики й оцінені параметри моделі *Холта-Вінтерса (адитивний тренд, мультиплікативна сезонність)*, отримані кодом лістингу 4.38, наведені нижче. Графік прогнозних значень, отриманий на основі моделі, подано на рис. 4.40.

```
    Параметри Холта-Вінтерса: Тренд='add',
Сезонність='mul', Сезонний період=12
                ExponentialSmoothing Model Results
    ================================================
    Dep. Variable:   m11010100   No. Observations:   101
    Model: ExponentialSmoothing SSE 667655637051633 20.000
    Optimized:          True       AIC           3478.609
    Trend:              Additive   BIC           3520.451
    Seasonal:     Multiplicative   AICC          3486.951
    Seasonal Periods:   12      Date:   Fri, 23 May 2025
    Box-Cox:            False   Time:    21:02:31
    Box-Cox Coeff.:               None
    ================================================
                      coeff      code       optimized
    ------------------------------------------------
    smoothing_level        0.3585714     alpha       True
    smoothing_trend        0.0001        beta        True
    smoothing_seasonal     0.2509938     gamma       True
```

Оцінка моделі Холта-Вінтерса

Адитивний тренд, Мультиплікативна сезонність

```
Метрики на навчальних даних (In-sample):
MSE (навчальні): 661045185199636.88
MAE (навчальні): 17348681.14
RMSE (навчальні): 25710799.00
MAPE (%) (навчальні): 5.20%
```

Отже, як видно із метрик результатів і на навчальних даних, модель *Холта-Вінтерса (Адитивний тренд, Мультиплікативна сезонність)* показала вищу точність із розглянутих алгоритмів, зокрема *MAPE 5,20 %.*

На графіку можна виокремити два діапазони, де модель не змогла передбачити дані, це початок періоду Covid і початок вторгнення росії в Україну. Модель може дати змогу оцінити орієнтовні втрати податкових надходжень, спричинених цими форсмажорними обставинами.

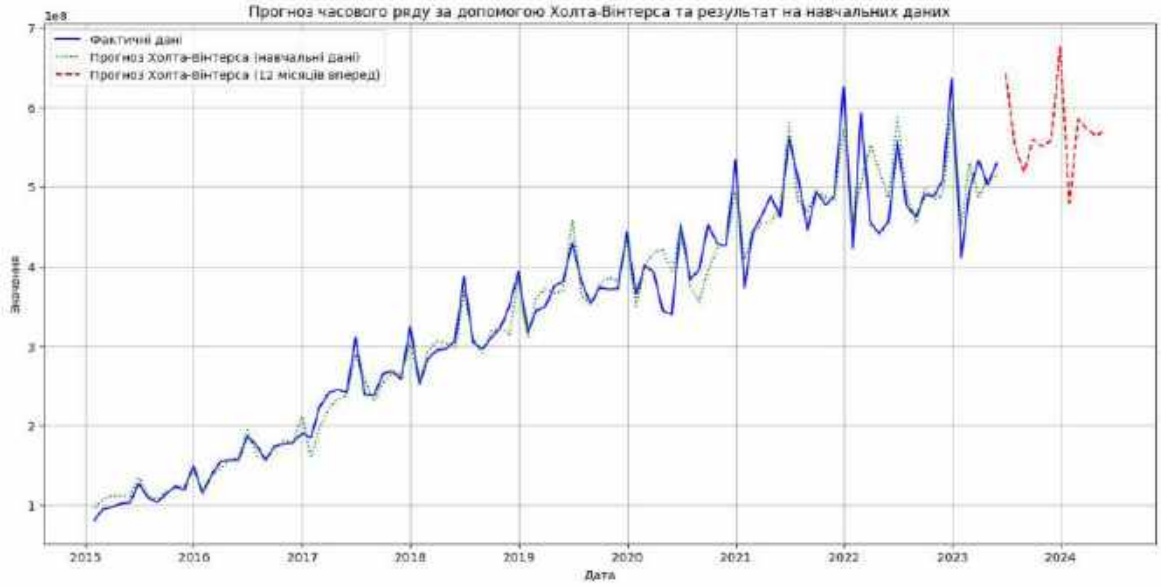


Рисунок 4.40 – Графік отриманий на основі моделі Холта-Вінтерса (адитивний тренд, мультиплікативна сезонність)

Як зазначалось, лабораторним практикумом передбачається проведення досліджень моделей машинного навчання на платформі Oracle Machinre Learning. Платформа дає змогу використовувати моделі, зокрема модель *Холта-Вінтерса*, яка показує ще більшу точність (*MAPE* у межах *4–4,5 %*) [30].

Питання для самоконтролю

1. Що містить у собі розвідувальний аналіз даних часових рядів?

2. Які аспекти аналізу аномалій у часових рядах важливі для виявлення непередбачуваних відхилень?

3. Які методи використовуються для аналізу закону розподілу даних у часових рядах?

4. Що таке стаціонарність у контексті аналізу часових рядів і чому вона важлива?

5. Як визначається сезонність у часових рядах і як вона впливає на моделювання?

6. Які методи використовуються для інженерії ознак часових рядів?

7. Що містить у собі формування тренувального й валідаційного датасетів у задачах прогнозування часових рядів?

8. Які основні характеристики моделей ARIMA і як вони використовуються для прогнозування часових рядів?

9. Які основні характеристики моделей SARIMA і як вони використовуються для прогнозування часових рядів?

10. Які основні характеристики моделей експоненційного згладжування і як вони використовуються для прогнозування часових рядів?

11. Які особливості має модель Facebook Prophet і в чому її переваги порівняно з іншими методами?

12. Які особливості має модель BERTA і в чому її переваги порівняно з іншими методами?

13. Які моделі нейромереж використовуються для прогнозування часових рядів і як вони працюють?

14. Як можна використати моделі переднавчених нейромереж для прогнозування часових рядів і як вони працюють?

4.7. Алгоритми пошуку аномалій в даних

4.7.1. Огляд задач пошуку аномалій у різних сферах

Фінансова сфера

Виявлення шахрайства з кредитними картками

Приклад. Клієнт зазвичай робить покупки в Україні на невеликі суми. Раптом його карткою оплачується велика покупка електроніки в іншій країні, де він ніколи не був, ще й серед ночі за його місцевим часом. Системи виявлення аномалій помітять цю транзакцію як нетипову за місцем, часом, сумою та категорією товару. Банк може тимчасово заблокувати картку та зв'язатися з клієнтом для підтвердження операції.

Аномалія. Нехарактерна географія, час, сума, тип покупки.

Маніпуляції на фондовому ринку

Приклад. Раптове аномальне зростання обсягів торгів щодо акцій маловідомої компанії без видимих новинних причин, або скоординовані малі угоди, спрямовані на штучну зміну ціни. Алгоритми можуть виявити такі патерни, що відхиляються від звичайної ринкової активності, вказуючи на можливі інсайдерські торги або схеми «pump and dump».

Аномалія. Незвичайний обсяг торгів, нетипова цінова динаміка, підозрілі послідовності угод.

Відмивання грошей

Приклад. Клієнт раптово починає отримувати численні дрібні перекази з різних незв'язаних джерел, після чого швидко переводить всю суму на закордонний рахунок. Така поведінка, що відрізняється від його попередньої фінансової активності, може бути позначена як підозріла.

Аномалія. Зміна патернів надходжень і виведення коштів, незвичні джерела та отримувачі.

Пошук аномалій у кібербезпеці

Виявлення вторгнень в мережу (IDS / IPS)

Приклад. Сервер компанії зазвичай має стабільний обсяг вхідного та вихідного трафіку з певними типами запитів. Раптом фіксується масовий потік незвичних запитів з однієї IP-адреси на

рідко використовуваний порт, або спроби підключення з використанням відомих облікових даних, які довго не використовувалися. Це може сигналізувати про DDoS-атаку, сканування вразливостей або спробу несанкціонованого доступу.

Аномалія. Нетиповий обсяг трафіку, незвичні протоколи або порти, спроби доступу з підозрілих IP, нехарактерні логіни.

Виявлення шкідливого програмного забезпечення (Malware)

Приклад. Програма на комп'ютері користувача починає нетипово активно використовувати мережу, звертатися до невідомих доменів, модифікувати системні файли або шифрувати дані користувача. Така поведінка, що відрізняється від звичайної роботи легітимних програм, може вказувати на зараження вірусом-шифрувальником або шпигунським ПЗ.

Аномалія. Незвична активність процесів, несанкціонований доступ до файлів, нетипова мережева активність.

Пошук аномалій у промисловості й медицині

Прогнозування відмов обладнання

Приклад. Датчики на промисловому верстаті (наприклад, двигуні) фіксують температуру, вібрацію, рівень шуму. Упродовж тривалого часу ці показники коливаються в межах норми. Раптом датчик вібрації починає показувати значно вищі значення, ніж зазвичай, або температура поступово, але неухильно зростає поза межі очікуваного діапазону. Це може свідчити про знос підшипника або іншу проблему, що насувається, дозволяючи провести технічне обслуговування до повної відмови обладнання.

Аномалія. Відхилення показників датчиків (температура, вібрація, тиск) від нормальних робочих діапазонів.

Контроль якості продукції

Приклад. На лінії розливу напоїв камера фіксує рівень рідини в кожній пляшці. Більшість пляшок наповнюються до стандартного рівня. Однак іноді трапляються пляшки зі значно нижчим або вищим рівнем рідини. Система розпізнає їх як аномалії (брак) і відбраковує.

Аномалія. Відхилення параметрів продукту (вага, розмір, колір, склад) від встановлених стандартів.

Виявлення аномалій на медичних зображеннях

Приклад. Під час аналізу МРТ-знімку мозку пацієнта система, навчена на тисячах «здорових» знімків, виявляє ділянку з нетиповою текстурою, яскравістю або формою, яка не схожа на нормальні структури мозку. Це може бути ранньою ознакою пухлини або іншого патологічного процесу, на яку лікарю варто звернути особливу увагу.

Аномалія. Нетипові патерни, текстури, форми на рентгенівських знімках, МРТ, КТ.

Моніторинг стану пацієнтів

Приклад. Пристрій пацієнта з хронічним захворюванням серця постійно відстежує його пульс, артеріальний тиск та рівень активності. Раптове значне падіння або стрибок тиску, не пов'язаний із фізичним навантаженням, або нерегулярний серцевий ритм, що триває певний час, можуть бути зафіксовані як аномалії, сповіщаючи медичний персонал або родичів.

Аномалія. Несподівані зміни життєво важливих показників (пульс, тиск, насичення киснем).

Електронна комерція та маркетинг

Виявлення нетипової поведінки користувачів

Приклад. Користувач, який зазвичай переглядає кілька сторінок товарів перед покупкою, раптом заходить на сайт і впродовж хвилини додає до кошика десятки дорогих товарів з різних категорій і намагається оформити замовлення з новою, раніше не використовуваною кредитною карткою. Це може бути ознакою зламу акаунту або шахрайської активності.

Аномалія. Нехарактерна навігація щодо сайту, надзвичайно велика кількість замовлень, підозрілі спроби оплати.

Моніторинг ефективності рекламних кампаній

Приклад. Після запуску нової рекламної кампанії кількість переходів на сайт різко зростає, але при цьому показник конверсії (відсоток відвідувачів, що здійснили покупку) аномально низький, а показник відмов (bounce rate) надзвичайно високий. Це може свідчити про нерелевантний трафік або проблеми на цільовій сторінці.

Аномалія. Несподівані зміни в ключових метриках (CTR, конверсія, bounce rate).

4.7.2. Основні методи пошуку аномалій в даних

1. Метод опорних векторів (Support Vector Machine – SVM)

Принцип. SVM – це зазвичай алгоритм керованого навчання, який використовується для класифікації. Однак його можна адаптувати для виявлення аномалій за допомогою підходу, відомого як *One-Class SVM.*

One-Class SVM навчається на «нормальних» даних. Він намагається знайти гіперплощину, яка найкраще охоплює більшість даних у просторі ознак. Будь-яка точка даних, що потрапляє далеко за межі цієї гіперплощини, вважається аномалією.

Інструментарій: Python: *scikit-learn (клас OneClassSVM).*

2. Методи, засновані на відстані (охоплюючи евклідові аномалії)

Принцип. Ці методи базуються на припущенні, що нормальні точки даних розташовані близько одна до одної, тоді як аномалії знаходяться далеко від своїх сусідів. Евклідова відстань є найпоширенішою метрикою, але можуть використовуватися й інші (наприклад, манхеттенська відстань, відстань Махаланобіса).

Сутність Евклідових аномалій: K-найближчих сусідів (k-NN): для кожної точки даних розраховується середня відстань до її k найближчих сусідів. Точки з найбільшими середніми відстанями вважаються аномаліями (підрозділ 4.4).

Кластеризація: методи, як-от DBSCAN, можуть ідентифікувати точки, які не належать до жодного щільного кластера, як аномалії (шум). Точки в розріджених областях, далеко від центрів кластерів (якщо використовується, наприклад, k-середніх), також можуть бути кандидатами на аномалії. Інструментарій: Python: *scikit-learn (класи NearestNeighbors, DBSCAN, KMeans), NumPy та SciPy* для обчислення відстаней.

3. Статистичні методи (на основі розподілу)

Принцип. Ці методи припускають, що нормальні дані відповідають певному статистичному розподілу (наприклад, гауссовому). Точки даних, які мають низьку ймовірність виникнення згідно з цим розподілом, вважаються аномаліями (підрозділ 3.2). Один із найпопулярніших методів:

Z-оцінка (Standard Score): обчислюється, на скільки стандартних відхилень точка даних віддалена від середнього значення. Точки з Z-оцінкою, що перевищує певний поріг (наприклад, 3), вважаються аномаліями.

Міжквартильний розмах (IQR): точки, що виходять за межі Q1 – 1.5 × IQR або Q3 + 1.5 × IQR, вважаються аномаліями.

Інструментарій: Python: *SciPy (модуль stats), NumPy, Pandas*.

4. Методи на основі дерев рішень (ізоляційний ліс)

Принцип. Ізоляційний ліс (*Isolation Forest)* є ефективним методом, особливо для багатовимірних наборів даних. Він працює на основі принципу, що аномалії легше «ізолювати» (відокремити від решти даних), ніж нормальні точки.

Алгоритм: будується ансамбль випадкових дерев рішень. У кожному дереві дані рекурсивно розділяються шляхом випадкового вибору ознаки та випадкового значення поділу в діапазоні цієї ознаки. Аномальні точки, як правило, вимагають менше таких поділів, щоб бути ізольованими, і тому матимуть коротші середні шляхи в деревах (розділ 4.2). Інструментарій: Python: *scikit-learn (клас IsolationForest)*

5. Методи на основі щільності (Local Outlier Factor – LOF) densité

Принцип. LOF оцінює локальну щільність навколо кожної точки даних порівняно з локальною щільністю її сусідів. Аномалії матимуть значно нижчу локальну щільність, ніж їхні сусіди.

Алгоритм: для кожної точки розраховується «коефіцієнт локального викиду». Цей коефіцієнт відображає ступінь ізольованості точки відносно її оточення. Значення LOF, значно більше за 1, вказує на аномалію.

Інструментарій: Python: *scikit-learn (клас LocalOutlierFactor)*

Вибір найкращого методу та інструменту залежить від характеристик ваших даних (розмірність, обсяг, тип), наявності міток (кероване чи некероване навчання) та конкретних вимог вашого завдання. Часто ефективним підходом є комбінування декількох методів.

4.7.3. Приклад використання методу Евклідової відстані

Розглянемо пошук аномалій на прикладі аналізу трафіку інтернету.

Величини, що характеризують трафік, які можна зібрати різними засобами та інструментрацієм (сніфери, аналізатори пакетів мереж тощо) також відомі як ознаки, можуть потребувати певної підготовки та трансформації перед аналізом. Наприклад, величина, яка отримується у результаті виконання команди «ping», дає змогу оцінити швидкості інтернету. Ця функція описується проміжками часу.

Трансформація даних. Величини, що контролюються, а саме швидкість завантаження та вивантаження, мають розміри *зворотного* часу. Через це «ping» не є оптимальним вибором для статистичного аналізу. Кращі результати досягаються за допомогою пов'язаної функції, яку будемо називати «швидкість пінгу», яка отримується через застосування простого перетворення (4.36):

$$\text{ping rate} = \frac{1}{\text{ping time}} \tag{4.36}$$

Цей процес «модифікації» величин з огляду на аналіз називається «проектуванням ознак» і, як правило, є важливою частиною робочого процесу машинного навчання (повний лістинг прикладу наданий у блокноті до лабораторного практикуму).

Дослідження даних. Ознайомимося з даними датфрейму *df* після трансформації даних *rpi.csv*

	upload_rate	**download_rate**	**ping_rate**
count	673.000000	673.000000	673.000000
mean	13.697608	90.449212	48.657348
std	1.080471	3.465647	6.030512
min	6.210000	58.480000	13.067113
25%	13.640000	90.220000	45.781257
50%	13.980000	91.560000	49.207755
75%	14.180000	92.070000	52.314936
max	14.550000	92.870000	61.854395

Дані колонок *upload_rate, download_rate, ping_rate* продаються в міл. с.

Інтерактивні візуалізації залишаються важливим способом дослідження проблеми. Завантажимо Клас *Axes3D* з *mpl_toolkits.mplot3d*, який дає змогу створювати тривимірні сюжети, розширюючи функціональні можливості знайомого модуля *matplotlib.pyplot.*

Метод *interact, який* відіграє важливу роль в анімації візуалізації, завантажуємо із бібліотеки *ipywidgets.* Цей модуль зі свого боку є частиною середовища *Ipython* для інтерактивного запуску Python (лістинг).

```
                    # Лістинг
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from ipywidgets import interact
```

Виявлення Евклідових аномалій. Обчислення відхилення від середнього (*Calculate the means*).

За допомогою відповідної функції Pandas обчислимо список *mu,* елементами якого є середня швидкість завантаження, вивантаження та пінгування. Як показано на тривимірному графіку рис. 4.41, аномалії перебувають далеко від «середньої точки». Точки, що розміщені за певною відстанню від середньої, вважатимемо аномальними, тоді як усі інші вважатимуться нормальними. Уявімо координати *[швидкість завантаження, швидкість вивантаження, швидкість пінгу]* як точки даних {x,y,z}. Виміряємо відстань від середнього місця розташування$\{\mu_x,\mu_y,\mu_z\}$ за формулою (4.37):

$$d = \sqrt{(x-\mu_x)^2 + (y-\mu_y)^2 + (z-\mu_z)^2}. \qquad (4.37)$$

Це основний інгредієнт системи виявлення аномалій, яку побудуємо. Крім того, це звичайне поняття відстані називається евклідовим, щоб відрізнити його від можливих узагальнень. Як було зазначено, точки x, y або z, які значно відрізняються від середніх показників μ_x, μ_y, та μ_z, і будуть вважатись аномальними. Поняття «значно» будемо визначати і досліджувати.

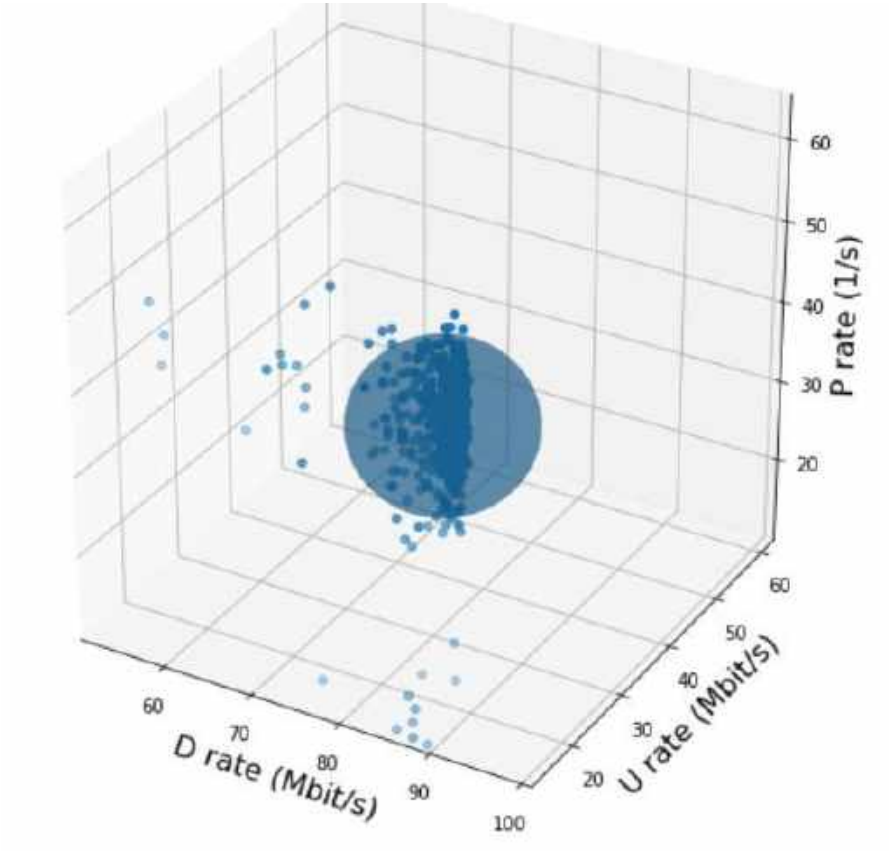
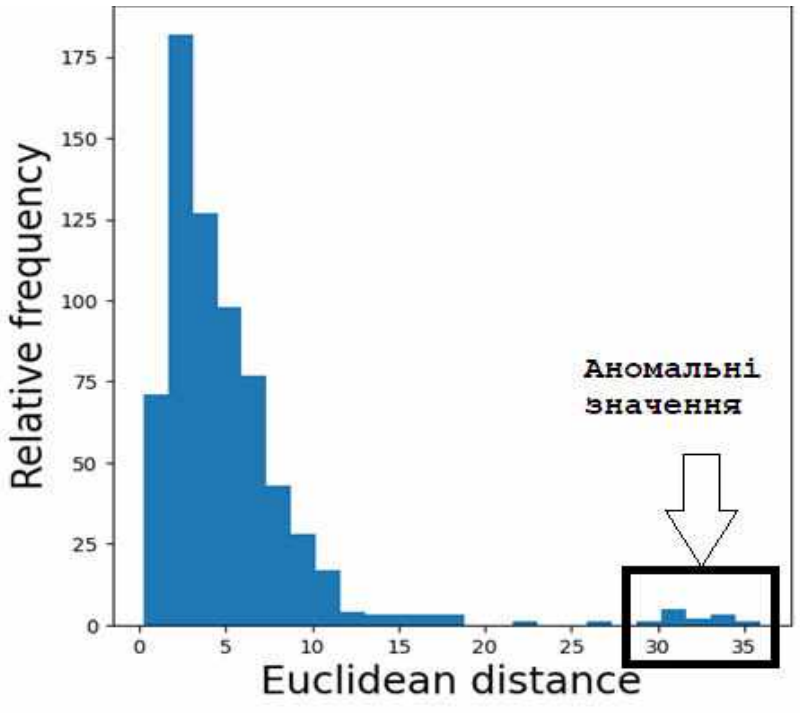


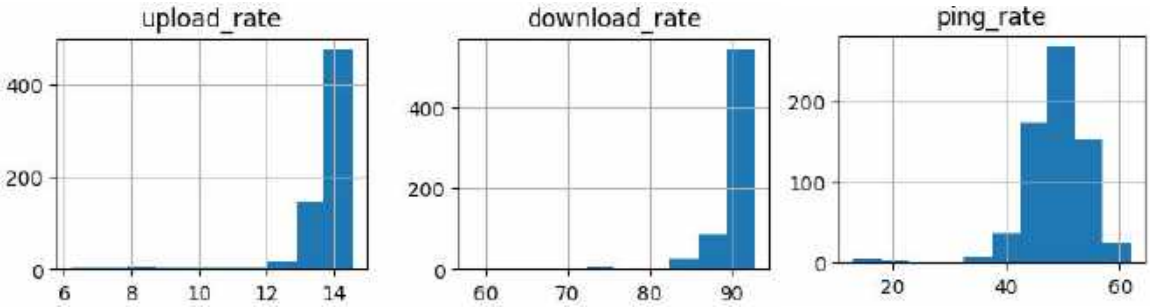
Рисунок 4.41 – Розсіювання значень навколо середнього

Обчислення Евклідової відстані і побудова гістограм. Для обчислення Евклідової відстані між кожною точкою та середнім розташуванням можемо використати функції Numpy *np.square manp.sqrt.* Вони обчислюють за елементами квадратний ступінь і квадратний корінь вхідного масиву. Використання поелементних функцій дає змогу обчислити відстані без циклів «for», які повторюються по точках даних.

Також для візуалізації корисно вивести гістограму відстані, що відокремлює кожну точку даних від середнього розташування. Тоді аномалії легко розпізнаються в «хвості» гістограми як для відносної частоти (рис. 4.42, а), так і для кожного показника трафіка (рис. 4.42, б).



а)



б)

Рисунок 4.42 – Гістограми відстаней, що відокремлюють кожну точку даних від середнього розташування

Обчислення нормованої відстані. Відношення кожної відстані до максимального значення d є важливо ознакою в оцінюванні відхилень. Найдальші аномальні точки мають нормовану відстань 1. Обчислимо нормовану Евклідову відстань для кожної точки в наборі даних і виведемо у вигляді гістограми, на якій також можна графічно оцінити точки із анормальними відхиленнями від середнього (рис. 4.43)

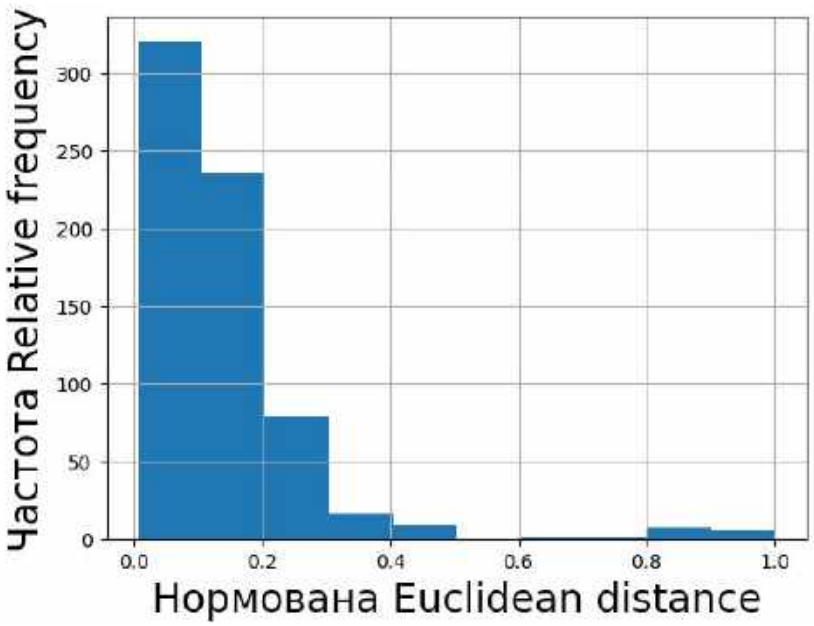


Рисунок 4.43 – Оцінка точок із аномальними відхиленням від середнього

Візуалізація частоти сигналів тривоги. Щоб позначити точки як нормальні та аномальні, можна визначити *межу прийняття рішення*. Потрібно вибрати відстань, у разі виходу точки за яку можна вважати нетиповою. Хоча положення межі можна зафіксувати на основі оціночних міркувань, кількісний підхід все ж є кращим. Можна поступово збільшувати відстань прийняття рішень і стежити за тим, як змінюється частота тривожних значень. Зокрема, частота тривожних значень – це кількість аномалій, поділена на загальну кількість точок даних. Ця процедура «випробовує» всі можливі межі прийняття рішень, дозволяючи нам вибрати кращу.

Можна змінювати нормалізовану відстань від 0 до 1 і записувати частоту тривожних значень на кожному кроці. Можна зберегти значення нормалізованої відстані та швидкості зміни тривожних значень у списках *nmd_range* та *ecl_alarm_rate,* відповідно. Ці результати можна візуалізувати як графік (рис. 4.43).

Вибір і візуалізація межі прийняття рішення. Наведений вище підхід перетворює проблему фіксації межі прийняття рішення в проблему вибору частоти тривог. Це є правильним, оскільки область застосування зазвичай визначає розумний вибір частоти тривожних значень. Наприклад, перевіряючи безпеку двигунів

літаків, можемо вибрати таке значення, щоб частота тривоги була відносно високою, щоб виявляти як великі, так і незначні аномалії, а рівень ризику залишався низьким.

Припустимо, що для цього конкретного випадку оптимальна частота тривоги становить 0,1. Можна знайти найменшу нормовану відстань прийняття рішення *(ecl_decision),* щоб частота тривоги впала строго нижче порогового значення. Оскільки частота сигналізації на вибраній межі рішення не буде рівною точно 0,1, будемо зберігати точну швидкість у змінній *ecl_threshold* (рис. 4.44).

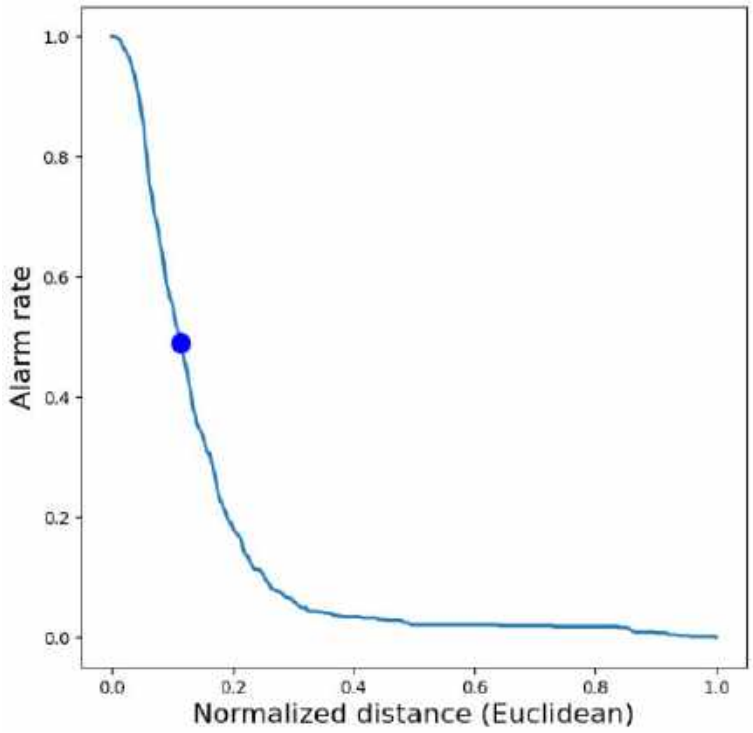


Рисунок 4.44 – Вибір і візуалізація межі прийняття рішення

Пояснення до графіка

Вісь X: «Normalized distance (Euclidean)» – нормалізована відстань (Евклідова) показує відстань від певної точки даних до деякого нормованого центру або до найближчих сусідів.

Вісь Y: «Alarm rate» (рівень спрацьовування сигналізації / тривог) показує частку (від 0 до 1, або від 0 до 100 %) точок даних, які будуть класифіковані як аномалії (або «тривоги»), якщо встановити певний поріг відстані. Високий «Alarm rate» означає, що система досить чутлива і позначає багато точок як аномалії. Низький «Alarm rate» означає, що система менш чутлива і позначає лише найбільш віддалені точки як аномалії.

Коли обираємо *досить малий поріг відстані* (значення на осі X близькі до 0), навіть невеликі відхилення вважаються аномаліями. Це призводить до *високого рівня спрацьовування сигналізації* (значення на осі Y близькі до 1.0). Система досить «прискіплива», «чутлива».

Зі *збільшенням порогу відстані* (рухаючись вправо по осі X), стаємо більш «терпимими» до відхилень. Тільки точки, що розташовані дійсно на великій відстані, будуть позначені як аномалії. Внаслідок цього рівень спрацьовування сигналізації зменшується.

Можливі компроміси. *Синя точка* є точкою компромісу. Адміністратор, що налаштовує систему, аналізує графік і вирішує, який рівень хибних тривог він готовий прийняти, щоб досягти бажаного рівня виявлення справжніх аномалій, або навпаки. Вибір залежить від конкретного завдання: наприклад, у медицині краще мати більше хибних тривог, ніж пропустити захворювання, тоді як у системі рекомендацій товарів кілька пропущених «аномальних» (рідкісних) вподобань можуть бути менш критичними, ніж постійні нерелевантні «тривоги»-рекомендації.

Отже, графік візуалізує, як зміна «суворості» правила (порогової відстані) впливає на те, скільки об’єктів буде позначено як «тривога», допомагаючи знайти оптимальний баланс.

Питання для самоконтролю

1. У чому полягає ключова ідея методу One-Class SVM для виявлення аномалій і на яких даних він зазвичай навчається?

2. Поясніть, як методи, засновані на відстані (наприклад, з використанням Евклідової відстані або k-найближчих сусідів), ідентифікують аномалії.

3. Яке фундаментальне припущення роблять статистичні методи (наприклад, Z-оцінка, IQR) щодо «нормальних» даних?

4. Що характеризує «локальний фактор викиду» (LOF) і чому він може бути ефективнішим за глобальні методи на даних зі змінною щільністю?

5. Наведіть приклад ситуації, коли використання методу на основі кластеризації (наприклад, DBSCAN) буде доцільним для пошуку аномалій.

6. Які основні бібліотеки Python ви б використали для реалізації завдань пошуку аномалій, і яка роль кожної з них (наприклад, для обробки даних, реалізації алгоритмів, візуалізації)?

7. Яке призначення функції *interact* з бібліотеки *ipywidgets* у контексті візуалізації даних, як це було показано в прикладі коду для 3D-графіка?

8. Опишіть два різних приклади застосування пошуку аномалій у фінансовій сфері. Які типи даних там можуть аналізуватися?

9. Як виявлення аномалій у показниках датчиків може допомогти в обслуговуванні промислового обладнання?

10. Поясніть своїми словами, що відображає графік «Вибір і візуалізація межі прийняття рішення» (графік залежності «Alarm rate» від «Normalized distance»).

11. Який компроміс потрібно враховувати під час вибору точки на кривій (межі прийняття рішення) на згаданому графіку? Що означає вибір точки з високим «Alarm rate»?

12. Які аргументи функції *scatter_view* у наданому раніше прикладі коду для побудови 3D-графіка керують кутом огляду візуалізації?

13. Якщо інтерактивний 3D-графік не відображається в Jupyter Notebook після виконання коду з *ipywidgets* та *matplotlib,* які три першочергові кроки для діагностики проблеми ви б зробили?

14. Яку роль відіграє магічна команда *%matplotlib inline* у середовищі Jupyter Notebook під час роботи з графіками?

ТЕМА 5
МЕТОДИ ТА ІНСТРУМЕНТАРІЙ ШТУЧНОГО ІНТЕЛЕКТУ

5.1. Штучні нейронні мережі

5.1.1. Сучасний стан та актуальність штучних нейронних мереж

Теорія штучних нейронних мереж почалася з роботи Воррена Маккаллока й Волтера Піттса 1943 р., яка вперше показала, як «нейрони» можна описати логічним численням. Саме тоді зародилася ідея, що сукупність простих елементів здатна відтворити складну обчислювальну поведінку, – фундамент, на якому базуються всі сучасні глибокі мережі.

Ідея штучних нейронних мереж (ШНМ) зародилася ще в середині XX століття, вченими, натхненими дослідженнями структурою та функціонуванням людського мозку. Перші концепції, як-от перцептрон Френка Розенблатта (1957), продемонстрували здатність до навчання на простих задачах. Однак через обмежені обчислювальні потужності та теоретичні труднощі (наприклад, нездатність перцептрона вирішувати нелінійно роздільні задачі), інтерес до ШНМ значно знизився, що призвело до так званої «зими штучного інтелекту». Новий сплеск інтересу відбувся у 1980-х роках з розробкою алгоритму зворотного поширення помилки, що дозволило навчати багатошарові мережі (Хопфілд).

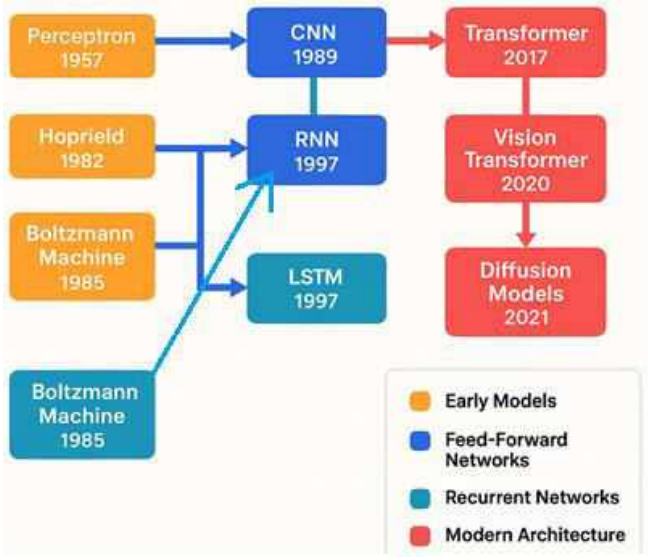


Рисунок 5.1 – Історія розвитку штучних нейронних мереж

Проте справжній прорив стався на початку 2010-х років завдяки поєднанню трьох ключових факторів: доступності великих обсягів даних (BigData), значного зростання обчислювальних потужностей (зокрема, завдяки графічним процесорам – GPU) та вдосконаленню алгоритмів глибокого навчання (Deep Learning). Поворотним моментом став перехід до трансформерів (2017 р.), які навчилися працювати з довгими контекстами й різними модальностями (рис. 5.1)

Perceptron (1957). Найперший формалізований «штучний нейрон»: лінійний комбінаційний вузол із пороговою (step) активацією та правилом оновлення ваг через помилки. Показав, що машина може навчатися з даних, але також виявив межі лінійного розділювача (нездатність опрацювати XOR). Саме ці обмеження стимулювали розвиток багатошарових мереж із back-propagation.

Hopfield Network (1982). Поява рекурентної мережі з симетричними вагами, яка мінімізує енергетичну функцію й поводиться як асоціативна пам’ять: за «зашумленим» входом вона сходиться до найближчого збереженого патерна. Концепція енергетичного ландшафту стала фундаментом для сучасних енергетичних та attention-моделей (Modern Hopfield = математика self-attention).

Boltzmann Machine (1985). Стохастичне розширення Хопфілда, де вузли вибирають стани з імовірністю, пропорційною енергії. Може моделювати складні розподіли даних і генерувати нові приклади; дало початок обмеженим Болцман-машинам (RBM) і глибоким вірувальним мережам, які вперше «розігріли» хвилю deep learning у 2000-х.

CNN (Convolutional Neural Network, 1989). Започатковуються локальні фільтри з розділенням ваг (shared weights) та підвибірковий пулінг, тож мережа вчиться просторово-інваріантних ознак. Від *LeNet через AlexNet і ResNet* до *EfficientNet CNN* стали стандартом для комп’ютерного зору, аналізу медичних знімків та інших задач розпізнавання образів.

RNN (Recurrent Neural Network, 1997). Додається петля з прихованим станом, щоб обробляти послідовності довільної довжини. Зберігається контекст, але існує недолік від затухання / вибуху градієнтів, що обмежує довгі залежності. Незважаючи на це, алгоритм заклав основу для мовних і музичних моделей.

LSTM (Long Short-Term Memory, 1997). Удосконалюється RNN із коміркою пам'яті та трьома «воротами» (forget, input, output), які керують потоком інформації й знімають проблему затухання градієнтів. Завдяки цьому LSTM стали рушієм мовного моделювання, розпізнавання мовлення та прогнозів часових рядів до приходу Transformer.

Transformer (2017). Архітектура, що повністю покладається на багатоголовий self-attention без згорток і рекуренції. Паралельність і масштабованість дали змогу тренувати гігантські мовні моделі (GPT, Gemini), які оперують контекстами тисяч токенів і виконують генерацію тексту, коду, музики.

Vision Transformer (2020). Ідея Transformer переноситься на зображення: картинку ділять на патчі, які трактуються як «токени». Попри відсутність індуктивних упереджень CNN, ViT досягає конкурентної точності на великих наборах даних і легко інтегрується у мультимодальні системи.

Diffusion Models (2021). Генеративний підхід, що навчається поетапно додавати й зворотно прибирати ґаусівський шум, перетворюючи його на семантично змістовне зображення чи аудіо. Демонструє наразі найкращу фотореалістичність (Stable Diffusion, DALL-E 3) і стає основою творчих інструментів нового покоління.

Сьогодні штучні нейронні мережі, особливо їх глибокі архітектури, є домінуючою силою в багатьох галузях штучного інтелекту. Архітектури, як-от згорткові нейронні мережі (CNN), рекурентні нейронні мережі (RNN), зокрема LSTM та GRU, і особливо трансформери, продемонстрували революційні результати. CNN домінують у комп'ютерному зорі (розпізнавання об'єктів, сегментація зображень). RNN та трансформери кардинально змінили обробку природної мови (NLP), забезпечуючи якісний машинний переклад, аналіз тональності, генерацію тексту та розуміння мови. Трансформери, зокрема, лягли в основу великих мовних моделей (LLM), як-от GPT, BERT, PaLM та інші, які здатні генерувати зв'язний, осмислений текст, відповідати на питання, писати код та виконувати багато інших завдань на рівні, що раніше вважався недосяжним. Розвиток генеративно-змагальних мереж (GAN) відкрив нові можливості у генерації реалістичних

зображень, відео та інших даних. Сучасні ШНМ – це складні системи з мільярдами, а іноді й трильйонами параметрів, навчання яких вимагає значних ресурсів.

Класифікація сучасних структур нейромереж

Класифікація структур сучасних ШНМ подана в таблиці 5.1.

Отже, сьогодні живемо в еру генеративних LLM та мультимодальності. 2023–2025 рр. позначилися стрімким розвитком великих мовних і мультимодальних моделей (LLM). OpenAI випустила GPT-4o, що вивело діалогові системи на новий рівень швидкості та універсальності, тоді як Google представила Gemini 1.5 Pro з контекстним вікном до 1 млн токенів, відкриваючи шлях до «пам'яті» довжиною в проєкти. Паралельно модель Sora навчилася генерувати реалістичне відео з тексту, демонструючи, що нейромережі можуть симулювати фізичний світ, а не лише описувати його.

Актуальність штучних нейронних мереж сьогодні надзвичайно висока і продовжує зростати. Вони є рушійною силою сучасних застосунків на базі ШІ. Яскравим прикладом є *чат-боти* та *віртуальні асистенти*. Якщо раніше чат-боти переважно працювали за жорсткими скриптами, то сучасні системи, побудовані на великих мовних моделях (LLM), здатні вести напрочуд природні діалоги, розуміти контекст, надавати розгорнуті та релевантні відповіді, адаптуватися до стилю співрозмовника і навіть проявляти елементи *«творчості»*. Це значно розширило їхнє застосування від клієнтської підтримки до персоналізованого навчання та розваг.

Таблиця 5.1 – Класифікація сучасних структур нейромереж

Клас сімей	Базова ідея	Ключові різновиди / віхи	Типові задачі
1	2	3	4
1. Пряме поширення (Feed-Forward)	Шари повністю з'єднаних нейронів, інформація рухається тільки вперед	MLP, Deep Fully-Connected Nets	Табличні дані, базова регресія / класифікація
2. Згорткові мережі (CNN)	Локальні фільтри + пулінг для виділення інваріантних просторових ознак	LeNet → AlexNet → VGG → Inception → ResNet → EfficientNet	Комп'ютерний зір, об'єктна детекція, медичні зображення

Продовження таблиці 5.1

1	2	3	4
3. Рекурентні мережі (RNN)	Стан, що передається в часі; пам'ять про попередні кроки	Vanilla-RNN, LSTM, GRU, Bidirectional/Stacked варіанти	Тимчасові ряди, мова, аудіо
4. Механізм уваги й Transformers	Повністю на self-attention, жодних згорток / рекуренцій; висока паралельність	Encoder-Decoder (T5), Decoder-only (GPT), Vision Transformer, Performer	NLP, мультимодальність, код, відео
5. Генеративні змагальні мережі (GAN)	Генератор vs дискримінатор у мінімакс-грі	DCGAN, StyleGAN, CycleGAN, Diffusion-GAN гібриди	Фотореалізм, deep-fake, реставрація зображень
6. Безперервні / диференціальні (Neural ODE, SDE-Net)	Динаміка описується системами ODE / SDE; «нескінченно глибокі» мережі	Neural ODE, Latent ODE	Моделювання фізики, імітація процесів
7. Спайкові та нейроморфні	Біологічно-правдоподібні імпульсні сигнали; подія-кероване обчислення	SNN, Loihi-Net, LIF-Net	Робототехніка низького енергоспоживання
8. Модульні / Mixture-of-Experts	Великий «роутер» вмикає лише релевантних експертів ➔ масштаб із розрідженістю	Switch-Transformer, GLaM, DeepMoE	Великі LLM із зменшеними FLOPs

Іншим важливим напрямом є *інтелектуальні агенти* – системи, що можуть автономно сприймати оточення, приймати рішення та виконувати дії для досягнення поставлених цілей. Нейронні мережі забезпечують цих агентів можливостями сприйняття (через комп'ютерний зір та розпізнавання мови), навчання (наприклад, за допомогою навчання з підкріпленням для оптимізації стратегій) і прийняття рішень у складних, динамічних середовищах. Це знаходить застосування в робототехніці, автономному транспорті, ігрових ботах, системах управління ресурсами та багатьох інших сферах. Розвиток ШНМ також стимулює прогрес у

медицині (діагностика захворювань за знімками, розробка ліків), фінансах (виявлення шахрайства, алгоритмічний трейдинг), наукових дослідженнях (аналіз великих масивів даних, моделювання складних систем) та мистецтві (генерація музики, візуальних образів). Оскільки здатність ШНМ навчатися на даних і виявляти складні закономірності продовжує вдосконалюватися, їх роль у створенні все більш інтелектуальних та автономних систем буде лише зростати.

Масове впровадження ШІ вже зафіксували аналітики McKinsey: 2024 р. 65 % компаній використовували ШНМ принаймні в одній бізнес-функції, а найдинамічніше зростання спостерігається у маркетингу, ланцюгах постачання та розробці продуктів. У медицині глибокі моделі підсилюють радіологів: національне дослідження скринінгу раку молочної залози в Німеччині показало підвищення виявлення пухлин на 17 % без зростання хибнопозитивних результатів.

З огляду на вибухове зростання генеративних можливостей ключовими темами стають якісні дані, етичні бар'єри, енергоспоживання й довіра до рішень, які пропонують моделі. Проте навіть попри ці виклики, нейромережі залишаються основною рушійною силою цифрової трансформації, формуючи екосистему інструментів і сервісів, що змінюють майже кожну галузь – від аграрного аналізу до кінематографу. Разом узяті ці тенденції роблять штучні нейромережі не просто технологією, а ключовою інфраструктурою цифрової економіки, що визначатиме конкурентоспроможність галузей у найближче десятиліття.

5.1.2. Біологічний нейрон і його штучна модель

Бологічний нейрон і сучасний ШІ

Людський мозок складається з приблизно 10^{11} найпростіших клітин, які називаються *нейронами,* які здатні обробляти інформацію. Нейрони пов'язані між собою, і кількість зв'язків оцінюється як 10^{15}. Кожен нейрон окремо виконує тільки найпростіші перетворення, але паралельна робота великого числа нейронів і величезна кількість зв'язків між ними призводять до того, що

нейронна мережа людського мозку загалом може виконувати досить складні перетворення сигналів в реальному часі. Крім того, значна кількість зв'язків робить нейронну мережу стійкою до помилок, які виникають в окремих зв'язках, – робота всієї мережі не зазнає істотних змін.

Тобто людський мозок можна представити розгалуженою мережею біологічних нейронів. Коли наголошуємо про поняття *штучної нейронної мережі*, будемо мати на увазі, що йдеться про обчислювальну модель і інструментарій, що реалізує окремі функції людського мозку, наприклад: зір – розпізнавання образів; розуміння тексту; розуміння мови, звуків; перекладання тексту тощо. Зрозуміло, що зі штучних нейромереж різного призначення можна компонувати *комп'ютеризовані андроїди* різної складності, які значно швидше виконують пошукові, обчислювальні та інші задачі. Але на сьогодні ШНМ ще не реалізують когнітивні функції людського мозку у всій його повноті, глибині та взаємозв'язку. Сучасний ШІ, який переважно є *«вузьким» ШІ (Artificial Narrow Intelligence – ANI),* який здатен виконувати завдання, пов'язані з окремими когнітивними функціями, іноді навіть перевершуючи людину у вузько спеціалізованих сферах. Він є потужним інструментом для автоматизації та допомоги людині. Але на сьогодні йому досі бракує, і ще невідомо погано це чи добре для людства:

загального інтелекту (Artificial General Intelligence – AGI): здатності розуміти, навчатися та застосовувати знання у широкому спектрі завдань так, як це робить людина;

свідомості та підсвідомості: суб'єктивного досвіду, відчуття «я»;

справжнього розуміння: глибокого осмислення інформації, а не простого розпізнавання патернів;

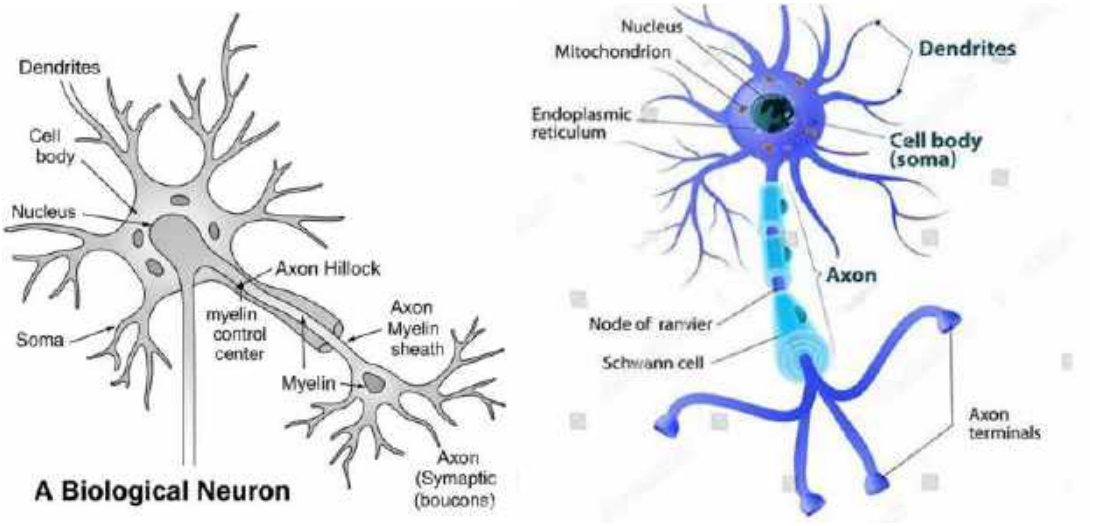
здорового глузду: базових знань про світ та інтуїтивного розуміння ситуацій;

внутрішньої мотивації та цілепокладання;

емоцій та емпатії.

Дослідження в галузі ШІ продовжуються, і в майбутньому його можливості, ймовірно, розширяться. Однак відтворення повноцінного людського пізнання залишається надзвичайно складним завданням.

У спеціальні й літературі біологічний нейрон подається спеціальними нейробіологічними термінами. У технічній літературі вони подаються спрощено для загального розуміння зв'язку біологічних процесів та їх штучним відтворенням. Біологічний нейрон – це спеціалізована клітина нервової системи, яка відповідає за прийом, обробку та передачу інформації у вигляді електричних та хімічних сигналів (рис. 5.2).



а) б)

Рисунок 5.2 – Спрощені зображення біологічних нейронів

Основні компоненти біологічного нейрона (рис. 5.2)

Дендрити (Dendrites). Короткі, розгалужені відростки, що відходять від тіла клітини. Їхня основна функція – отримувати сигнали від інших нейронів (або сенсорних рецепторів) через спеціальні контакти, які називаються синапсами.

Сома (Soma) або Тіло клітини (Cell Body). Центральна частина нейрона, що містить ядро та інші органели. Тут відбувається інтеграція (підсумовування) сигналів, отриманих від дендритів. Якщо сумарний сигнал досягає певного порогу, генерується електричний імпульс.

Аксонний горбик (Axon Hillock). Ділянка соми, де відбувається генерація потенціалу дії (електричного імпульсу), якщо сумарна деполяризація досягає порогового рівня.

Аксон (Axon). Довгий відросток, що передає електричний імпульс (потенціал дії) від тіла клітини до інших нейронів, м'язів або залоз. Аксони можуть бути вкриті мієліновою оболонкою, яка прискорює передачу сигналу.

Аксонні терміналі (Axon Terminals) або Синаптичні закінчення (Synaptic Boutons). Кінцеві розгалуження аксона, які утворюють синапси з дендритами або сомами інших нейронів. Під час надходження потенціалу дії сюди вони вивільняють хімічні речовини – нейромедіатори – у синаптичну щілину, які впливають на наступний нейрон.

Синапс (Synapse). Місце контакту між двома нейронами (або нейроном та ефекторною клітиною), де відбувається передача сигналу. Ця передача може бути збуджуючою або гальмівною, залежно від типу нейромедіатора та рецептора. «Сила» синапсу (його здатність впливати на постсинаптичний нейрон) може змінюватися з часом, що є основою навчання та пам'яті.

Модель штучного нейрону

Штучний нейрон, як зазначаємо, є математичною функцією, що моделює деякі аспекти роботи біологічного нейрона і може бути реалізований програмно чи апаратно (рис. 5.3).

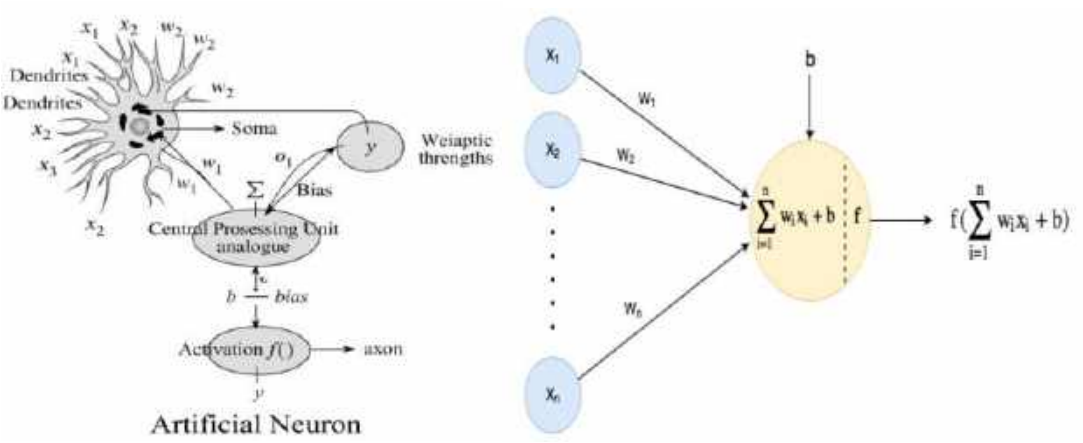


Рисунок 5.3 – Модель штучного нейрона

Входи (Inputs x_i). Числові значення, що надходять до нейрона.

Ваги (Weights w_i). Числові коефіцієнти, що присвоюються кожному входу, визначаючи його важливість.

Суматор (Summation Function). Обчислює зважену суму входів $\Sigma(x_i \times w_i)$.

Зміщення (Bias b). Додатковий параметр, що додається до зваженої суми, даючи змогу зсувати функцію активації. $z = \Sigma(x_i \times w_i) + b$.

Функція активації (Activation Function f(z)). Перетворює результат суматора z на вихідний сигнал нейрона y. Вводить нелінійність: *y = f(z).*

Вихід (Output y). Кінцевий сигнал, що видається нейроном.

Біологічний нейрон – складна клітина, а штучний – досить спрощена математична абстракція, втім основні елементи й принцип роботи збігаються. Взаємовідповідність між біологічним та штучним нейроном подана в таблиці 5.2.

Таблиця 5.2 – Взаємовідповідність між біологічним та штучним нейроном

Біологічний нейрон	Штучний нейрон	Опис Аналогії
Дендрити – гілочки, що збирають сигнали від інших нейронів	**Входи (x_i)**	Дендрити отримують сигнали від інших нейронів; входи штучного нейрона отримують числові значення (пікселі, слова, сенсорні показники)
Синапси – контакти, сила яких залежить від частоти активації	**Ваги (w_i)**	Синаптична сила (ефективність передачі сигналу через синапс) може змінюватися; ваги в штучному нейроні визначають силу впливу кожного входу і можуть зміню'ватися під час навчання
Сома – інтегрує збудження й гальмування	**Суматор та зміщення ($\Sigma(x_i * w_i) + b$)**	Сома інтегрує (підсумовує) всі отримані сигнали; суматор штучного нейрона обчислює зважену суму входів та додає зміщення
Аксонний горбик та поріг активації – «рішення» запустити потенціал дії	**Функція активації (f(z))**	Якщо сумарний сигнал у сомі перевищує поріг, генерується потенціал дії; функція активації визначає, чи буде нейрон «активований» і яким буде його вихід на основі зваженої суми
Аксон – передає імпульс далі	**Вихідний зв'язок $f(\sum w_i \times x_i + b)$**	Аксон передає сигнал (потенціал дії) від тіла клітини; вихідний зв'язок передає обчислене значення y
Пластичність синапсів – зміна сили зв'язків під дією досвіду	Оновлення ваг ***w_i*** **(back-prop, SGD)**	Процес навчання, що мінімізує помилку й «пам'ятає» новий досвід

Штучний нейрон взяв за основу роботу біологічного нерона: збирання інформації, зважування, порогову реакцію, і оброблює її як спрощений математичний блок. Масово з'єднані у шари, ці блоки утворюють багатошарові, глибокі нейромережі, що сьогодні розпізнають мову, керують роботами і пишуть наукові тексти

Ключові паралелі й відмінності

Накопичення сигналів. Обидві системи інтегрують численні вхідні імпульси. У біології це електрохімічні потенціали; у моделі – арифметичні суми чисел.

Пороговість / активація. Якщо комбінація сигналів перевищує певний поріг, реальний нейрон генерує «залп» (спайк). У штучному нейроні роль порогу виконує нелінійна функція: ReLU, sigmoid, tanh тощо.

Навчання та пам'ять. Нейро-пластичність реалізується через зміну числа рецепторів чи концентрації нейромедіаторів. У ШІ еквівалентом є коригування ваг під час градієнтного спуску, коли модель «запам'ятовує» корисні патерни даних.

Часові масштаби. Потенціал дії триває мілісекунди, тоді як «імпульс» у нейромережі – миттєвий розрахунок. Для моделювання біологічних процесів застосовують спеціальні спайкові мережі (SNN), які наближують часову динаміку.

Енергетика. Мозок споживає ~20 Вт і керує 86 млрд нейронів, а великі мовні моделі (LLM) потребують мегават-годин на тренування.

5.1.3. Функції активації та їх використання в ШНМ

Функції активації визначають вихідний сигнал нейрона на основі суми його зважених входів та зміщення. Основна їхня роль – *введення нелінійності* в модель. Без нелінійних функцій активації багатошарова нейронна мережа, незалежно від кількості шарів, поводилася б як один лінійний перцептрон і не змогла б навчатися складним залежностям у даних (наприклад, розпізнавати образи чи розуміти мову).

Наприклад нейрон отримує вхідні сигнали ($x_1, x_2,...,x_n$), кожен з яких має свою вагу ($w_1, w_2,...,wn$). Також є зміщення (b). Спочатку обчислюється *зважена сума* (5.1):

$$Z = (w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_n \cdot x_n) + b = \sum_{i=1}(w_i \cdot x_i) + b. \quad (5.1)$$

Потім ця сума z передається *функції активації* f, яка і генерує кінцевий вихід нейрона y: y = f(z)

Нижче подані деякі популярні функції активації, з їхніми властивостями кодом їх використання бібліотеки *keras*. Більше функцій активації подано у [33].

Усічена одиниця (ReLU) (у Вікіпедії: «випрямлена одиниця», що змістовно – менш коректно) (рис. 5.4).

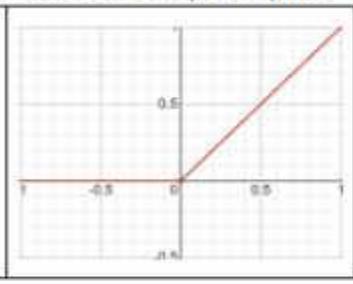
keras.activations.relu(x, alpha=0.0, max_value=None, threshold=0.0)				
Випрямлена лінійна		$f(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \text{ for } x < 0 \\ x \text{ for } x \geq 0 \end{cases}$	$[0, \infty)$

Рисунок 5.4 – Функція активації ReLU «усічена одиниця»

Має ряд підвидів: Leaky ReLU & Parametric ReLU (PReLU); ReLU-6; Exponential Linear (ELU, SELU).

Сігмоїда (логістична функція) «sigmoid» (рис. 5.5):

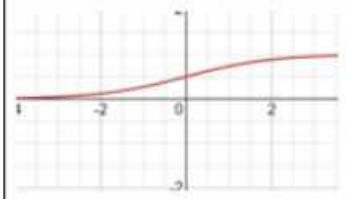
keras.activations.sigmoid(x)				
Логістична або Сігмоїдна		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	(0,1)

Рисунок 5.5 – Функція активації Сігмоїда (логістична функція) «sigmoid»

Гіперболічний тангенс «tanh» (рис. 5.6):

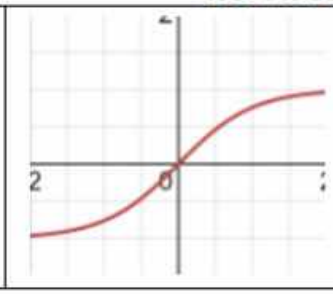
keras.activations.tanh(x)				
Гіперболічний тангенс		$f(x) = tanh(x) = \frac{(e^x - e^{-x})}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1,1)$

Рисунок 5.6 – Функція активації Гіперболічний тангенс «tanh»

Історично першою функцією активації була сігмоїда, але через багато помножень на досить малі значення сигнал часто «губився» між шарами і занулювався. Це було однією з причин повільного розвитку нейронних мереж. Але згодом було винайдено функцію ReLU, яка мала потрібні нелінійні властивості, легко автоматизувалась та була позбавлена такого недоліку. Наразі в усіх прихованих шарах найбільш популярною є саме ReLU, а у вихідному шарі, особливо для задач класифікації, використовують сигмоїду 150 (якщо усі значення є позитивними) чи – гіперболічний тангенс (якщо є від'ємні значення).

Загальні рекомендації щодо функції активації

Для прихованих шарів зазвичай починають з ReLU. Якщо виникають проблеми з «мертвими нейронами», можна спробувати Leaky ReLU, PReLU або ELU. Для вихідного шару вибір залежить від типу задачі:

Бінарна класифікація: Сигмоїда.

Багатокласова класифікація: Softmax.

Регресія: Лінійна функція.

Сигмоїда та Tanh рідше використовуються в глибоких прихованих шарах через проблему згасаючого градієнта, але Tanh може бути корисною завдяки своїй нуль-центрованості. Вибір оптимальної функції активації часто є емпіричним процесом і може залежати від конкретної архітектури мережі, даних та задачі. Експериментування з різними функціями є важливою частиною побудови ефективних моделей ШНМ.

Для початку досліджень можна починати з ReLU / GELU. ReLU краще підходить для класичних CNN, GELU – для будь-якої архітектури з LayerNorm (Transformer, MLP-Mixer).

Глибокі RNN часто замінюють Tanh на ReLU-подібні (ReLU, Leaky), але LSTM / GRU самі містять сигмоїд-вікна; залишайте дефолт.

Self-normalizing мережі (SELU + AlphaDropout) зменшують потребу в BatchNorm, але чутливі до ініціалізації LeCun-Normal.

Edge / мобільні застосунки беруть Hard-Swish або ReLU6 через прості integer-арифметику в NPU / TPU-lite.

Вихідний шар: Sigmoid для «one-vs-all», Softmax для «many-vs-many», Tanh для відновлення зображень, Linear для MSE-регресії.

5.1.4. Навчання ШНМ як задача мінімізації помилки

Процес навчання нейронної мережі полягає в підстроювання ваг її нейронів. Метою навчання є пошук стану ваг, яке мінімізує вихідну помилку мережі навчальному та тестовому наборі.

Завдання пошуку оптимального стану систем за мінімумом або максимумом деякої функції утворюють окремий клас і називаються завданнями оптимізації.

У разі штучної нейронної мережі оптимізація ведеться по функції помилки, залежної від стану ваг нейронів. Стан системи, при якому забезпечується мінімум цільової функції (в цьому випадку вихідний помилки мережі), називається оптимальним.

Наприклад, мережа, яка містить 28 зв’язків, за умови, що вага кожної зв’язку є бінарним, матиме 228, або приблизно 250 млрд комбінацій ваг, серед яких потрібно найти оптимальну, або наближену до оптимальної.

Нейронні мережі, які містять більш нейрони в прихованому шарі, мають не одне, а кілька оптимальних станів, обумовлених наявністю у функції помилки декількох мінімумів. Однак істинно оптимальним станом є тільки одне, де забезпечується «найменший», або, як кажуть, глобальний мінімум.

Решта мінімуми функції помилки називаються локальними. наявність множини мінімумів істотно ускладнює пошук глобального

оптимуму. На рис. 5.7. представлена залежність помилки мережі E від деякого узагальненого стану вектора ваг мережі W, яка має три локальних мінімуму.

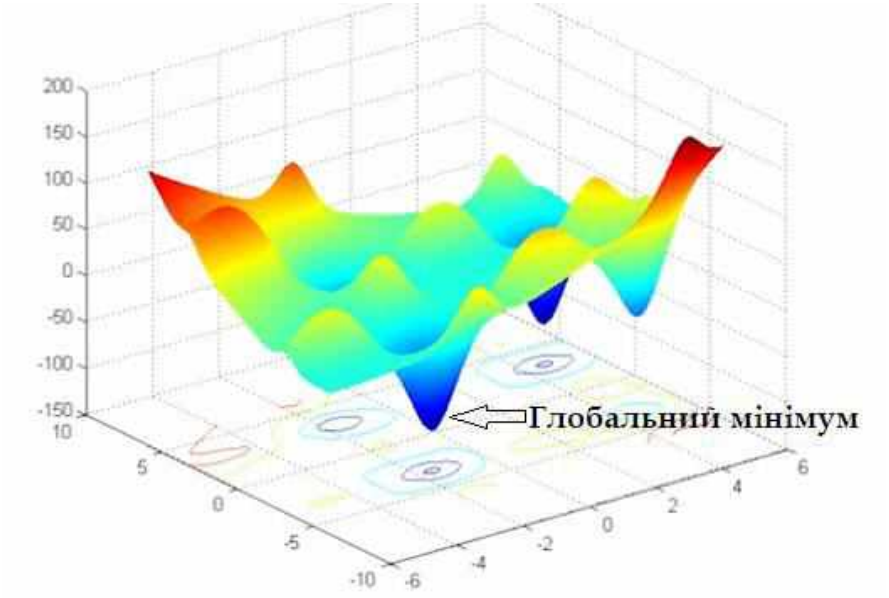
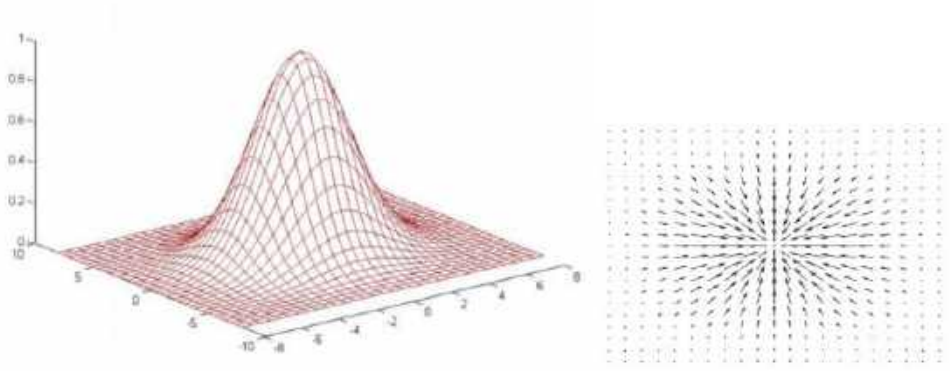


Рисунок 5.7 – Залежність помилки мережі E
від деякого узагальненого стану вектора ваг мережі W

Основою алгоритмів мінімізації помилки під час навчання ШНМ є градієнтні методи.

Розглянемо поняття – градієнт (від лат. *gradientis* – крокуючий). Це характеристика, що показує напрямок найшвидшого зростання деякої величини, значення якої змінюється від однієї точки простору до іншої. Наприклад, якщо взяти висоту поверхні Землі над рівнем моря (двовимірне простір), то її градієнт в кожній точці поверхні буде показувати вгору (рис. 5.8, а)



а) б)

Рисунок 5.8 – Графічне пояснення градієнта
а) тривимірне зображення висоти на поверхні; б) градієнт висоти та його поле векторів

Операція градієнта перетворює «пагорб», в поле векторів, напрямок яких в кожній точці збігаються з напрямом найбільш крутого підйому, а довжина пропорційна крутизні підйому. Видно, що всі вектори спрямовані до вершини. На пологій ділянці у «підніжжя пагорба» довжина векторів невелика. На самому крутому ділянці вектори мають максимальну довжину. І нарешті, там, де вершина «пагорба» утворює купол, довжина векторів зменшується (рис. 5.8, б).

Більшість методів навчання нейронних мереж є локальними, тобто виконують пошук локальних мінімумів функції помилки. Навчання проводиться, допоки алгоритм не досягне мінімуму, значення помилки в якому виявиться менше деякого заданого порогу ε. Взагалі кажучи, якщо значення глобального мінімуму заздалегідь невідомо, то неможливо сказати, наскільки близько до нього розміщений поточний стан ваг мережі.

Для пошуку глобального мінімуму використовуються різні методи:

Метод перебору всіх можливих значень цільової функції гарантовано дозволить виявити глобальний мінімум, але в більшості випадків цей спосіб не застосовується на практиці через величезну кількість станів ваг, який буде потрібно розглянути;

Можна кілька разів послідовно провести процес оптимізації, щоразу змінюючи початковий стан моделі в надії, що якесь із них виявиться в «Області тяжіння» глобального мінімуму.

Найбільшу популярність під час навчанні ШНМ отримали градієнтні алгоритми. В їх основі лежать метод градієнтного (найшвидшого) спуску, метод сполучених градієнтів та ін.

Причини поширеності градієнтних алгоритмів полягають в тому, що вони вимагають відносно невеликого числа ітерацій для навчання мережі, забезпечують досить низький рівень помилки і стійкі до аномальних значень вхідних ознак. **Недоліком** градієнтних методів є те, що вони можуть зупинятись у локальних мінімумах цільової функції.

Метод найшвидшого спуску – це метод знаходження локального мінімуму функції за допомогою руху вздовж *антиградієнта.* У разі навчання нейронних мереж цільовою функцією є *функція*

вихідної помилки мережі E (W). Нагадаємо, що градієнт є сумою похідних функції по трьох координатах. Графік цільової функції в тривимірних координатах (x, y, z) нагадує рельєф земної поверхні. Мінімуми функції утворюють западини, а максимуми – пагорби (рис. 5.9).

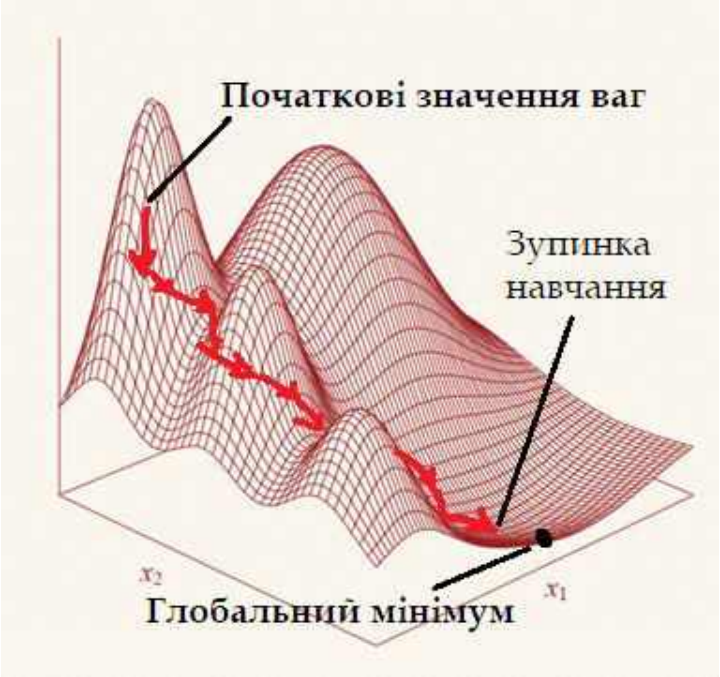


Рисунок 5.9 – Графік цільової функції в тривимірних координатах (x, y, z)

Процес оптимізації, тобто пошук оптимального стану ваг ШНМ, починається з випадкового вибору точки на рельєфі, яка вибирається в результаті початкової ініціалізації ваг вхідних нейронів невеликими випадковими значеннями.

Потім у цій точці обчислюється градієнт функції, і на його основі визначається точка, відповідна нового стану ваг. Водночас, згідно з розглянутого вище, властивості градієнта нова точка формується з певним кроком у напрямку максимального спадання функції. Далі обчислюється наступна точка, і процес повторюється, доки не буде досягнутий найближчий мінімум.

Напрямок корекції ваг визначили, але залишається відкритим питання про величину цієї корекції, або, як кажуть, про крок корекції. Чи впливає величина корекції ваг на результати навчання і якщо так, то як її вибрати? Вплив кроку корекції на кожній

ітерації очевидно. Якщо крок корекції виявиться занадто великим, то алгоритм може «переступити» через впадину на рельєфі і не виявити відповідного мінімуму. Якщо крок корекції буде занадто малим, то для досягнення мінімуму буде потрібно велика кількість ітерацій, що призведе до дуже довгої процедури навчання (рис. 5.10).

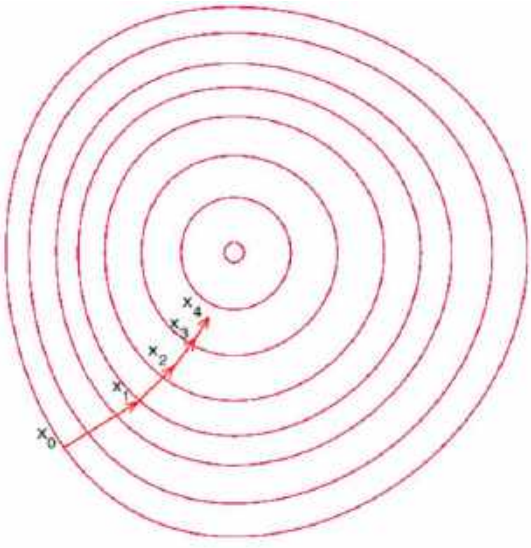


Рисунок 5.10 – Умовне зображення градієнтного спуску

На рис. 5.10 представлена «впадина» градієнтного рельєфу. При градієнтному спуску здійснюється перехід з точки в точку в напрямку антіградієнта. Водночас відстань між двома сусідніми точками x_i і $x_i + 1$ змінюється залежно від номера кроку. Початкові кроки найбільші, а наступні зменшуються в міру наближення до мінімуму.

Припустимо, що є вектор ваг $\mathbf{W} = (w_0, w_1, w_2 \ldots w_m)^T$, які потрібно налаштувати таким чином, щоб мінімізувати вихідну помилку мережі. Для цього можна використовувати метод градієнтного спуску, який визначить напрямок корекції ваг. Градієнт функції помилки щодо вектора ваг запишеться як вектор *часткових похідних* (5.2):

$$\nabla E(\mathbf{W}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots, \frac{\partial E}{\partial w_m} \right], \qquad (5.2)$$

Величина кроку корекції ваг

$$\Delta w = -\eta \frac{\partial E}{\partial w}, \tag{5.3}$$

де:

w – поточний стан ваги;

∂w – крок корекції стану ваги;

E (w) – поточне значення помилки;

η – константа, яка називається коефіцієнтом швидкості навчання.

Методика використання похідної функції помилки для управління кроком корекції ваг нейронів ілюструється на рис. 5.11.

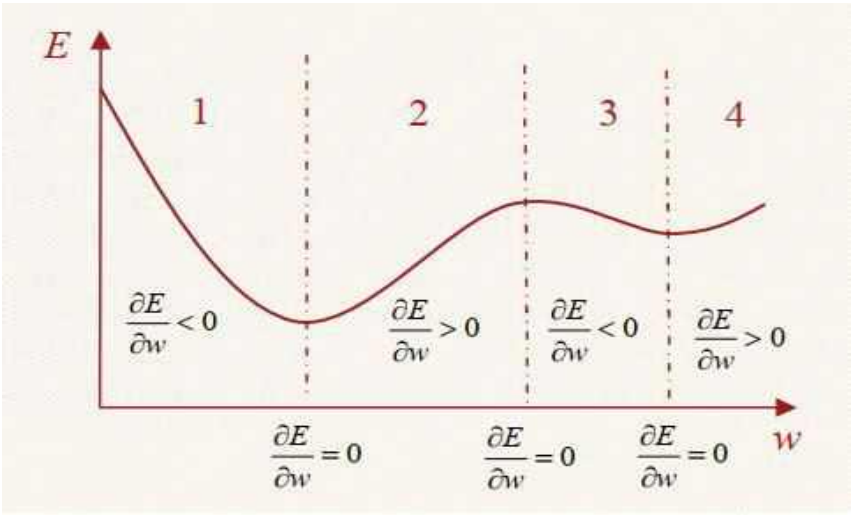


Рисунок 5.11 – Управління кроком корекції ваг

Величину кроку корекції ваги на кожній ітерації навчання часто називають швидкістю навчання. Відповідно до формули (5.2) вона визначається значенням похідної функції помилки. Однак, крім похідної, на швидкість навчання впливає додатковий параметр коефіцієнт швидкості навчання η (0; 1). Він підбирається експериментально. Якщо коефіцієнт швидкості навчання вели *(η ~1),* то швидкість навчання визначається тільки похідною функції помилки.

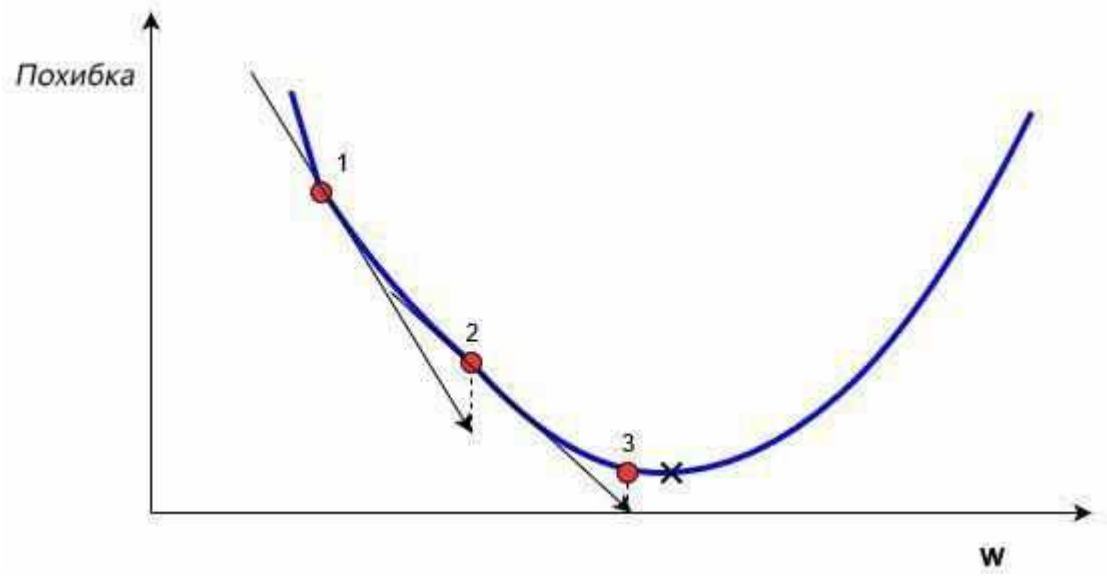


Рисунок 5.12 – Похибка градієнтного спуску

У разі зменшення коефіцієнта виникає ефект «гальмування» процесу: крок корекції зменшується у відповідності зі значенням коефіцієнта. Наприклад, при $\eta = 0,5$ кожен крок корекції зменшується в два рази.

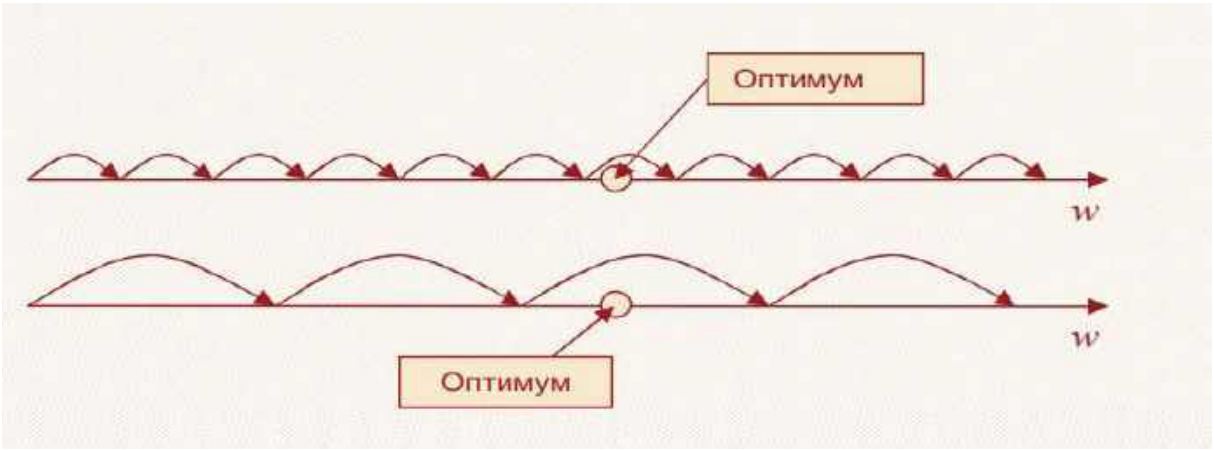


Рисунок 5.13 – Підбір коефіцієнта швидкості навчання

При підборі η керуються такою логікою:

– якщо задати його значення великим, наприклад 0,8 або 0,9, то такий крок корекції ваг призведе до прискорення процесу навчання для досягнення оптимуму потрібно менше число ітерацій, але водночас дещо знизиться точність пошуку оптимуму;

– якщо задати малий коефіцієнт швидкості навчання (0,1 – 0,2), то час, потрібний для навчання, зросте, але збільшиться і очікувана точність налаштування на оптимум (рис. 5.13).

5.1.5. Градієнтний спуск і алгоритм зворотного поширення помилки

Розглянемо приклад типової тришарової ШНМ з одним прихованим шаром і одним виходом (рис. 5.14).

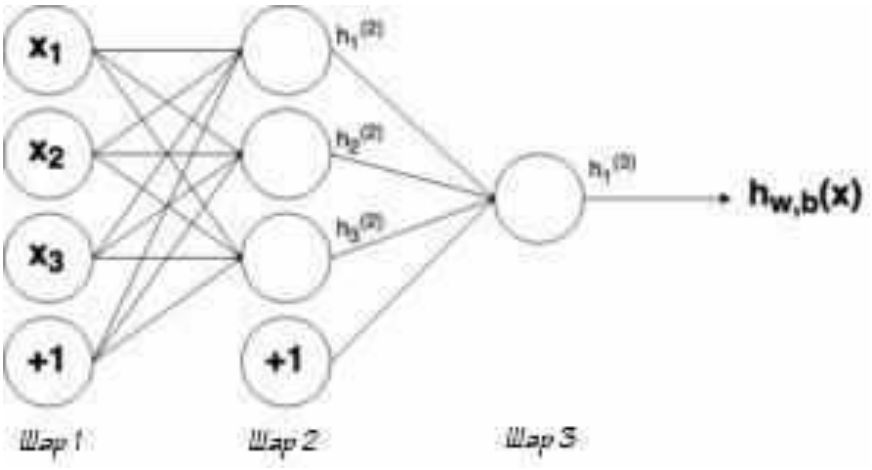


Рисунок 5.14 – Типова тришарова ШНМ

Базові рівняння для нейронної мережі з трьома шарами.

Вихід цієї нейронної мережі знаходиться за формулою (5.4):

$$h_{W,b}(x) = h_1^{(3)} = f(w_{11}^{(2)} h_1^{(2)} + w_{12}^{(2)} h_2^{(2)} + w_{13}^{(2)} h_3^{(2)} + b_1^{(2)}) \quad (5.4\ )$$

Можна спростити це рівняння до $h_1(3)=f(z_1^{(2)})$, додавши нове значення $z_1^{(2)}$, яке означає (5.5):

$$z_1^{(2)} = w_{11}^{(2)} h_1^{(2)} + w_{12}^{(2)} h_2^{(2)} + w_{13}^{(2)} h_3^{(2)} + b_1^{(2)} \quad (5.5)$$

Хочемо дізнатися, як впливає зміна у вазі $w_{12}^{(2)}$ на функцію оцінки. Це означає, що нам потрібно обчислити $\partial J/\partial w_{12}^{(2)}$. Щоб зробити це, потрібно використовувати правило диференціювання складної функції (5.6):

$$\frac{\partial J}{\partial w_{12}^{(2)}} = \frac{\partial J}{\partial h_1^{(3)}} \frac{\partial h_1^{(3)}}{\partial z_1^{(2)}} \frac{\partial z_1^{(2)}}{\partial w_{12}^{(2)}} \quad (5.6)$$

Якщо придивитися, то права частина повністю скорочується (за принципом (2/5) × (5/2) = 2/2 = 1). $\boldsymbol{\partial J/\partial w_{12}^{(2)}}$ було розбито на три множники, два з яких можна замінити. Почнемо з $\boldsymbol{\partial z_1^{(2)}/\partial w_{12}^{(2)}}$ (5.7):

$$\begin{aligned}\frac{\partial z_1^{(2)}}{\partial w_{12}^{(2)}} &= \frac{\partial}{\partial w_{12}^{(2)}}\left(w_{11}^{(1)}h_1^{(2)} + w_{12}^{(1)}h_2^{(2)} + w_{13}^{(1)}h_3^{(2)} + b_1^{(1)}\right) \\ &= \frac{\partial}{\partial w_{12}^{(2)}}\left(w_{12}^{(1)}h_2^{(2)}\right) = h_2^{(2)}\end{aligned} \tag{5.7}$$

Часткова похідна $z_1^{(2)}$ по $w_{12}^{(2)}$ залежить тільки від одного добутку в дужках, $w_{12}^{(1)}h_2^{(2)}$, оскільки всі елементи в дужках, крім $w_{12}^{(2)}$, не змінюються. Похідна від константи завжди дорівнює 1, а $\partial/\partial w_{12}^{(2)}$ скорочується до просто $h_2^{(2)}$, що є звичайним виходом другого вузла з шару 2.

Наступна часткова похідна складної функції $\partial h_1^{(3)}/\partial z_1^{(2)}$ є частковою похідною активаційної функції вихідного вузла $h^{(3)}$. Через те, що нам потрібно брати похідні активаційної функції, випливає умова її включення до нейронних мереж – функція має бути диференційованою. Для сигмоїдальної активаційної функції похідна матиме такий вигляд:

$$\frac{\partial h}{\partial z} = f'(z) = f(z)\big(1 - f(z)\big), \tag{5.8}$$

де: f(z) є самою активаційною функцією.

Тепер нам потрібно розібратися, що робити з $\partial J/\partial h_1^{(3)}$. Згадаємо, що J(w, b, x, y) є функція квадрата похибки, має такий вигляд (5.8.1):

$$J(w, b, x, y) = \frac{1}{2}\left\|y_1 - h_1^{(3)}\left(z_1^{(2)}\right)\right\|^{(2)}, \tag{5.8.1}$$

тут y_1 є очікуваним виходом для вихідного вузла. Знову використовуємо правило диференціювання складної функції (5.9) :

$$\begin{aligned} &Let\ u = \| y_1 - h_1^{(3)}(z_1^{(2)}) \| \text{ и } J = \frac{1}{2}u^2 \\ &\frac{\partial J}{\partial h} = \frac{\partial J}{\partial u}\frac{\partial u}{\partial h} : \Leftarrow \text{Використовуємо для спрощення} \\ &\frac{\partial J}{\partial h} = -(y_1 - h_1^{(3)}) \end{aligned} \tag{5.9}$$

З'ясували, як знаходити $\partial J/\partial w_{12}^{(2)}$ принаймні для ваг зв'язків із вихідним шаром. Перед тим, як перейти до одного з прихованих шарів, введемо деякі нові значення δ, щоб трохи скоротити наші вирази (5.10):

$$\delta_i^{(n_l)} = -(y_i - h_i^{(n_l)}) \cdot f'(z_i^{(n_l)}), \tag{5.10}$$

де: *i* є номером вузла у вихідному шарі.

У нашому прикладі є тільки один вузол, тому *i* = 1. Повний вигляд похідної функції оцінювання запишеться так (5.11):

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b, x, y) = h_j^{(l)} \delta_i^{(l+1)}, \tag{5.11}$$

де вихідний шар, у нашому випадку l = 2, а *i* відповідає номеру вузла.

Поширення в прихованих шарах

З вагами в прихованих шарах (у нашому випадку в шарі 2) виконуються такі дії.

Для ваг, які з'єднані з вихідним шаром, похідна $\partial J/\partial h = -(y_i - h_i^{(nl)})$ мала сенс, тому що функцію оцінювання можна одразу знайти через порівняння вихідного шару з наявними даними. Але виходи прихованих вузлів не мають подібних уже наявних даних для перевірки, вони пов'язані з функцією оцінки тільки через інші шари вузлів. Для визначення зміни в функції оцінки через зміни ваг використовуємо *метод зворотного поширення.*

Розглянемо детально ці дії на графічному прикладі (рис. 5.15). Значення, яке буде зворотно поширюватися, – $\delta i^{(nl)}$, тому що воно в найближчому зв'язку з функцією оцінки. А що з вузлом j у другому шарі (прихованому шарі)? Як він впливає на $\delta i^{(nl)}$ у нашій мережі? Він змінює інші значення через вагу $w^{(2)}$ (рис. 5.15 нижче, де j = 1 і = 1).

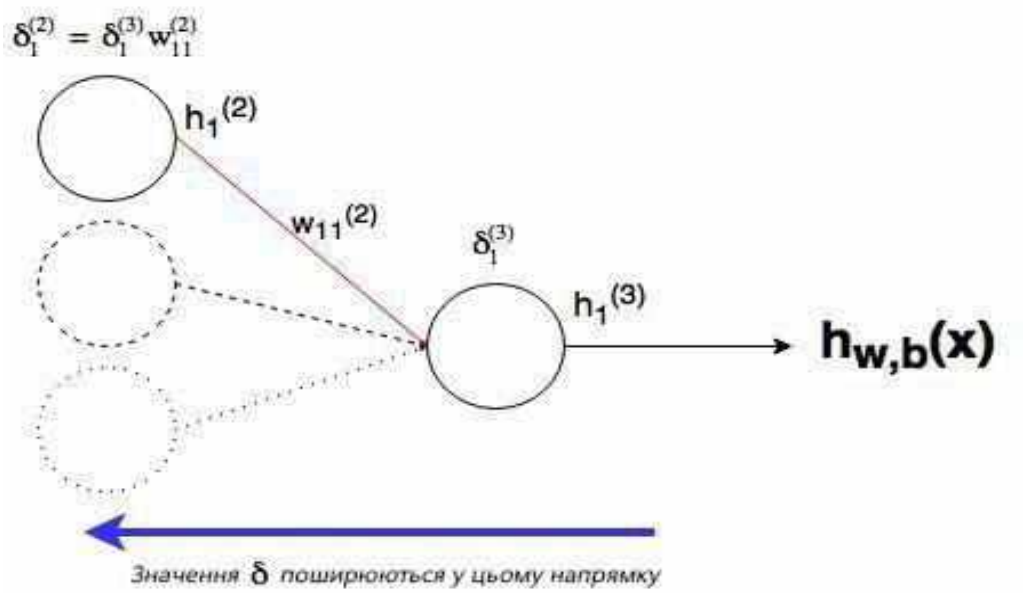


Рисунок 5.15 – Графічне пояснення
зворотного поширення – $\delta i^{(nl)}$

Як можна зрозуміти з рис. 5.15, вихідний шар з'єднується з прихованим вузлом через вагу. У випадку, коли у вихідному шарі є тільки один вузол, загальний вираз прихованого шару буде виглядати так (5.12):

$$\delta_j^{(l)} = \delta_1^{(l+1)} w_{1j}^{(l)} f'(z_j)^{(l)}, \qquad (5.12)$$

де j номер вузла в шарі l.

Але що буде, якщо у вихідному шарі є багато вихідних вузлів? У цьому випадку $\delta j^{(l)}$ перебуває за зваженою сумою всіх пов'язаних між собою похибок, як показано на рис. 5.16:

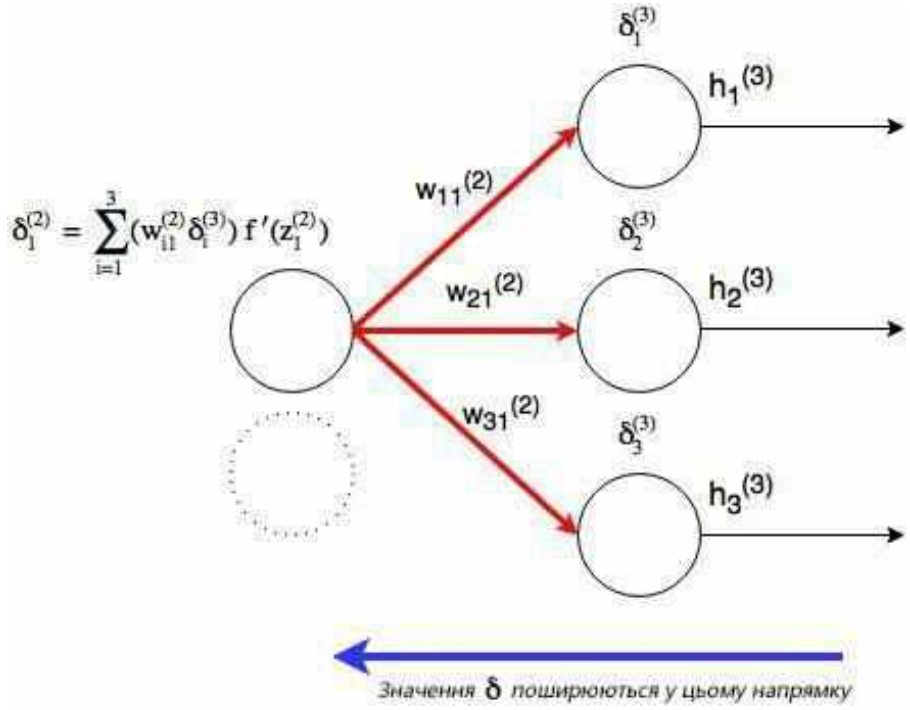


Рисунок 5.16 – Зважена сума всіх пов’язаних між собою похибок

На рис. 5.16 показано, що кожне значення δ з вихідного шару підсумовується для знаходження $\delta_1^{(2)}$, але кожний вихід δ має бути зваженим відповідними значенням $wi_1^{(2)}$. Інакше кажучи, вузол 1 у шарі 2 сприяє змінам похибок у трьох вихідних вузлах, водночас отримана похибка (або значення функції оцінювання) у кожному з цих вузлів має бути «передана назад» значенню δ цього вузла.

Сформуємо загальний вираз значення δ для вузлів у прихованому шарі (5.13):

$$\delta_j^{(l)} = \left(\sum_{i=1}^{s_{(l+1)}} w_{ij}^{(l)} \delta_i^{(l+1)}\right) f'(z_j^{(l)}), \qquad (5.13)$$

де:

j – номер вузла в шарі l;

i – номер вузла в шарі l + 1 (що аналогічно позначенням, яке використовували раніше);

s(l+1) – кількість вузлів у шарі l+1.

Тепер можемо знайти вираз (5.14):

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b, x, y) = h_j^{(l)} \delta_i^{(l+1)} \qquad (5.14)$$

Принцип роботи з вагами зміщення аналогічний звичайним вагам, використовуючи правила диференціювання складних функцій (5.15):

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b, x, y) = \delta_i^{(l+1)} \tag{5.15}$$

Отже, реалізувати градієнтний спуск у нейронних мережах можна на основі виразів (5.16):

$$\begin{aligned} w_{ij}^{(l)} &= w_{ij}^{(l)} - \alpha \frac{\partial}{\partial w_{ij}^{(l)}} J(w, b) \\ b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(w, b) \end{aligned} \tag{5.16}$$

Для такої реалізації нам потрібно буде застосувати цикли. Як відомо, цикли в мові програмування Python працюють досить повільно. У реальних алгоритмах застосується векторизація та інші підходи для таких розрахунків.

5.1.6. Методи та інструменти оптимізації функції помилки

Основні бібліотеки для глибокого навчання в Python, як-от *TensorFlow (з Keras API)* та *PyTorch*, мають вбудовану підтримку для зворотного поширення помилки (через автоматичне диференціювання) та надають широкий набір готових до використання градієнтних оптимізаторів.

У таблиці 5.3 подано декілька основних найбільш вживаних алгоритмів оптимізації *TensorFlow, PyTorch* із зазначенням їх використання.

Загальні рекомендації щодо використання алгоритмів

Рекомендується починати з Adam або AdamW. Ці оптимізатори часто дають хороші результати «з коробки» для багатьох задач і є гарною відправною точкою. *AdamW* є кращим вибором, якщо ви плануєте використовувати затухання ваг (*weight decay).*

*SGD з Моментом / Nesterov з*алишається сильним конкурентом і може навіть перевершити Adam на деяких задачах, особливо якщо гіперпараметри (швидкість навчання, момент) ретельно налаштовані. Часто вимагає більш уважного підбору розкладу зміни швидкості навчання (*learning rate schedule*).

Таблиця 5.3 – Основні алгоритми оптимізації

Метод, концепція оптимізації	Бібліотеки та реалізація	Рекомендації щодо використання / примітки
SGD (Стохастичний градієнтний спуск)	**TensorFlow**: `tf.keras.optimizers.SGD(...)` **PyTorch**: `torch.optim.SGD(...)`	Базовий оптимізатор. У TensorFlow / PyTorch, під час використання з міні-пакетами, це реалізація **міні-пакетного SGD**. Може вимагати ретельного налаштування швидкості навчання (`learning_rate` або `lr`). Часто використовується з **моментом** для кращої збіжності
Adam (Adaptive Moment Estimation)	**TensorFlow**: `tf.keras.optimizers.Adam(...)` **PyTorch**: `torch.optim.Adam(...)`	Досить популярний і часто ефективний оптимізатор, який добре працює «за стандартним налаштуванням» для широкого спектра задач. Поєднує ідеї моменту (зберігає ковзне середнє градієнтів) та RMSProp (зберігає ковзне середнє квадратів градієнтів). **Часто є першим вибором для експериментів**
AdaGrad (Adaptive Gradient)	**TensorFlow:** `tf.keras.optimizers.Adagrad(...)` **PyTorch:** `torch.optim.Adagrad(...)`	Адаптує швидкість навчання для кожного параметра окремо – більші оновлення для рідкісних параметрів, менші – для частих. Добре підходить для **розріджених даних** (наприклад, в NLP, де деякі слова зустрічаються рідко). Недолік: швидкість навчання може з часом стати занадто малою, зупиняючи навчання
RMSProp (Root Mean Square Propagation)	**TensorFlow:** `tf.keras.optimizers.RMSprop(...)` **PyTorch:** `torch.optim.RMSprop(...)`	Вирішує проблему AdaGrad із занадто швидким згасанням швидкості навчання, використовуючи ковзне середнє квадратів градієнтів. Часто є хорошим вибором для **рекурентних нейронних мереж (RNN)**

Як уже зазначалось, швидкість навчання *(Learning Rate)* – найважливіший гіперпараметр для налаштування. Занадто мала швидкість призведе до повільного навчання, занадто велика – до розбіжності. Використовуйте техніки пошуку оптимальної швидкості навчання (learning rate finders) або розклади зміни швидкості навчання (learning rate schedulers), такі як зменшення при плато (reduce on plateau) або косинусоїдальне затухання (cosine annealing).

Розмір міні-пакету (Batch Size): впливає на стабільність градієнтів та швидкість навчання. Типові значення: 32, 64, 128, 256. Більші розміри пакетів можуть прискорити навчання за рахунок кращого використання GPU, але можуть призвести до гіршої генералізації.

Специфіка задачі: для досить розріджених даних (наприклад, у деяких задачах NLP) AdaGrad або FTRL можуть бути корисними. Для RNN іноді добре себе показує RMSProp.

Експериментуйте: немає єдиного «найкращого» оптимізатора для всіх задач. Найкращий спосіб знайти оптимальний варіант – це експериментувати з різними оптимізаторами та їхніми гіперпараметрами на вашій конкретній задачі та даних.

Одна й та сама реалізація може працювати інакше в різних фреймворках через дефолтні значення ε, β та порядок bias-корекції. Завжди перевіряйте офіційну документацію або *help(optimizer)* перед запуском експерименту.

Пам'ятайте, що ефективне навчання нейронної мережі – це не лише вибір оптимізатора, а й правильна архітектура мережі, підготовка даних, ініціалізація ваг, регуляризація та налаштування гіперпараметрів.

Питання для самоконтролю

1. Пояснити сутність ШНМ.
2. Приклади застосування ШНМ в економічних системах.
3. Поясніть сутність персептрона Розенблата.
4. Для чого потрібні функції активації?
5. Наведіть класифікацію ШНМ.

6. Для чого використовуються передавальні функції нейрона?
7. Приклад математичної моделі штучного нейрона.
8. Узагальнена математична модель нейронної мережі.
9. Наведіть приклад передавальних функцій активації нейрона.
10. Наведіть приклад бібліотек Python для програмування ШНМ.
11. Особливості роботи складної структури ШНМ.
12. Поясніть сутність методу локальної оптимізації.
13. Поясніть сутність методу глобальної оптимізації.
14. Поясніть сутність градієнтного алгоритму навчання нейронних мереж.
15. Поясніть сутність процесу оптимізації за градієнтним спуском.
16. Опишіть модель ШНМ як модель шарів нейронів і функцій активації.
17. Поясніть навчання нейромережі як задачу оптимізації.
18. Поясніть на прикладі градієнтний метод оптимізації (наприклад, метод найшвидшого спуску).
19. Поясніть на прикладі розрахунок прямого та зворотного поширення помилки.

5.2. Згорткові нейронні мережі (CNN)

5.2.1. Структура згорткових нейронних мереж

Призначення. Згорткові нейронні мережі (*Convolutional Neuronal Network* (CNN)) загалом використовуються для обробки візуальних даних, як-от зображення та відео. Також успішно застосовуються в обробці природної мови (NLP) та інших задачах, де важлива локальна структура даних.

Ключові особливості

Згорткові (ковуляційні) шари (Convolutional Layers): застосовують фільтри (ядра) до вхідних даних для виявлення локальних ознак, таких як краї, текстури, форми;

Шари підвибірки (Pooling Layers): зменшують розмірність карт ознак, зберігаючи найважливішу інформацію (наприклад, максимальне (MaxPooling) або середнє значення в області);

Повнозв'язні шари (Fully Connected Layers): зазвичай розташовуються в кінці мережі для класифікації або регресії на основі вивчених ознак.

Типова архітектура конвенту викладена на рис. 5.17. Принцип роботи мережі продемонстровано на прикладі розпізнавання рукописної цифри 7. Зображення цієї цифри подається на вхід мережі. Далі воно проходить через згорткову частину, де до нього застосовуються конволюційні операції за допомогою фільтрів (ядер). У результаті нейронна мережа здатна визначити, яка цифра зображена [35].

Найважливішим компонентом моделі є *згортковий шар.* Його головна мета – *оптимізувати зображення для аналізу*: зменшити його розмір для швидшого обчислення ваг та покращити здатність мережі узагальнювати інформацію.

У процесі згортки мережа *виокремлює та зберігає ключові характеристики зображення***,** ігноруючи при цьому нерелевантний шум. Наприклад, якщо модель вчиться розпізнавати слона на фоні гори, традиційна нейронна мережа дасть вагу всім пікселям, включно з пікселями гори. Це може бути зайвим і навіть ввести мережу в оману.

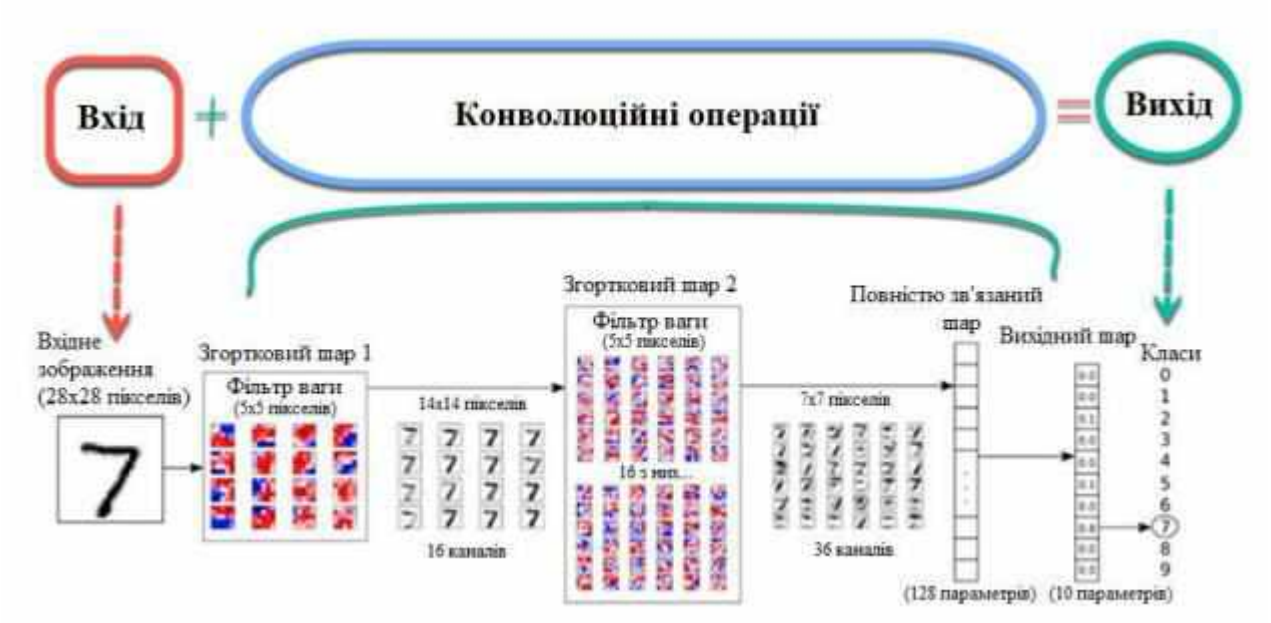


Рис. 5.17 – Типова архітектура згорткової мережі

Згорткова нейронна мережа, навпаки, використовує математичний метод (власне, ***згортку***) для *вибіркового аналізу пікселів, концентруючись лише на тих, що є найбільш значущими для розпізнавання об'єкта*. Цей підхід дозволяє мережі пошарово вивчати все складніші ознаки. По суті, згортка розглядає зображення невеликими фрагментами, щоб виявити найважливіші елементи в кожному з них.

Є чотири компоненти конвуляційних операцій: 1. Згортання (Convolution); 2. Нелінійність, використання функції активації (наприклад ReLU); 3. Об'єднання або макс-пул операція (MaxPuling); 4. Класифікація (повністю пов'язаний шар, FullConnection).

5.2.2. Процедури згортання (convolution)

Основне завдання згортки – виявляти локальні особливості об'єктів на зображенні. Це дає змогу нейронній мережі навчатися розпізнавати конкретні візуальні елементи (наприклад, краї, кути, текстури) і знаходити їх у будь-якій частині зображення.

По суті, *згортка – це операція, під час якої фрагмент зображення поелементно множиться на спеціальний шаблон, який називається фільтром (ядром).* Уявити це можна так: програма

бере невелику ділянку зображення (зазвичай розміром 3 × 3 пікселі) і для кожного пікселя цієї ділянки виконує множення на відповідне значення з комірки фільтра (рис. 5.18). Після цього всі отримані добутки додаються, і їхня сума записується як один новий елемент. Результатом такої обробки всього зображення є так звана *карта ознак*. Цей процес (сканування, множення та сумування) повторюється крок за кроком, доки фільтр не пройде по всьому зображенню. Важливо, що після операції згортки розмір отриманої карти ознак зазвичай *стає меншим* за розмір початкового зображення.

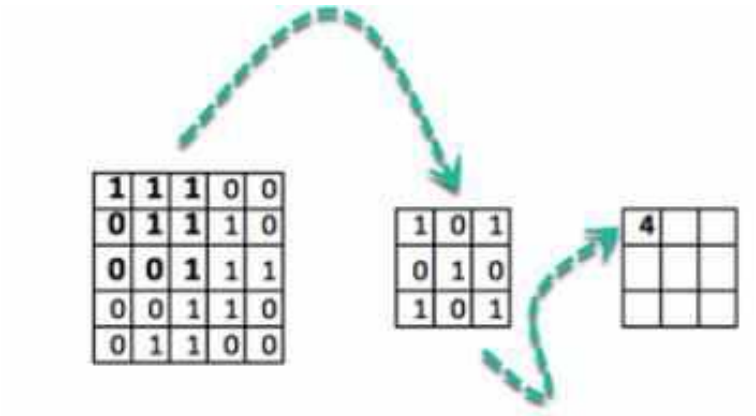


Рис. 5.18 – Операція згортки (конволюції)

На рис. 5.19 наведено деякі із доступних багатьох фільтрів. Кожен фільтр має певне призначення (поняття *ядро* (kernel) є синонімом фільтра).

Обчислення під час згортки

У процесі згортки фільтр (невелика матриця пікселів) накладається на фрагмент вхідного зображення. Цей фільтр, який зазвичай має розмір 3 × 3 або 5 × 5, рухається по всьому зображенню. Інакше кажучи, мережа пересуває це «вікно» фільтра по пікселях вхідного зображення, на кожному кроці обчислюючи результат згортки.

Наприклад, якщо розмір вікна фільтра 3 × 3, то для обчислення одного елемента вихідної матриці відбувається поелементне множення цієї 3 × 3 ділянки зображення на 3 × 3 матрицю фільтра, а потім результати цього множення підсумовуються.

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	

Рис. 5.19 – Деякі із доступних фільтрів для операції згортки

Варто зазначити, що ширина і висота на виході можуть відрізнятися від ширини і висоти входу. Це відбувається через ***ефект межі.*** Наприклад зображення має карту параметрів (розмір) 5 × 5 і фільтр розміром 3 × 3. У центрі є лише одне вікно, де фільтр може відображати сітку 3 × 3 (рис. 5.20). Карта функції на виході зменшиться на дві клітини разом з розміром 3 × 3.

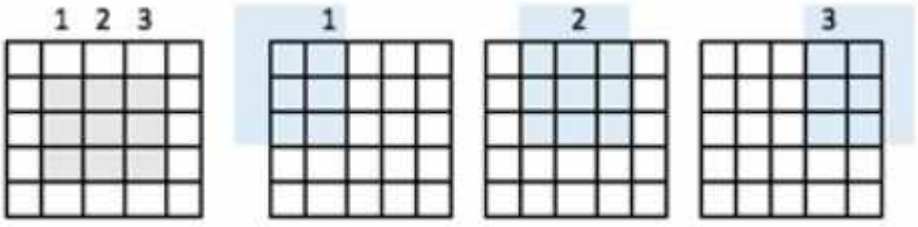


Рисунок 5.20 – Ефект межі в процесі згортки

Підкладка (Padding) для збереження вихідного розміру

Для того щоб розмір зображення на виході згорткового шару збігався з розміром на вході, використовується підкладка (padding). Це техніка додавання відповідної кількості рядків і стовпців по краях вхідної матриці. Підкладка допомагає фільтру

згортки обробляти крайні пікселі, дозволяючи йому «центруватися» на кожному вхідному елементі. Отже, вхідна та вихідна матриці можуть мати однаковий розмір, наприклад 5 × 5.

Параметри налаштування згорткових функцій

Робота згорткових шарів визначається трьома ключовими параметрами, серед яких:

1. Глибина (Depth). Визначає кількість фільтрів, що застосовуються на етапі згортки. Якщо раніше розглядалася глибина 1 (один фільтр), то в реальних задачах часто використовується більше фільтрів. Наприклад, під час використання трьох фільтрів кожен з них виконує свою операцію згортки, створюючи відповідну карту ознак. На рис. 5.21 показано виконання операції згортки із трьома фільтрами.

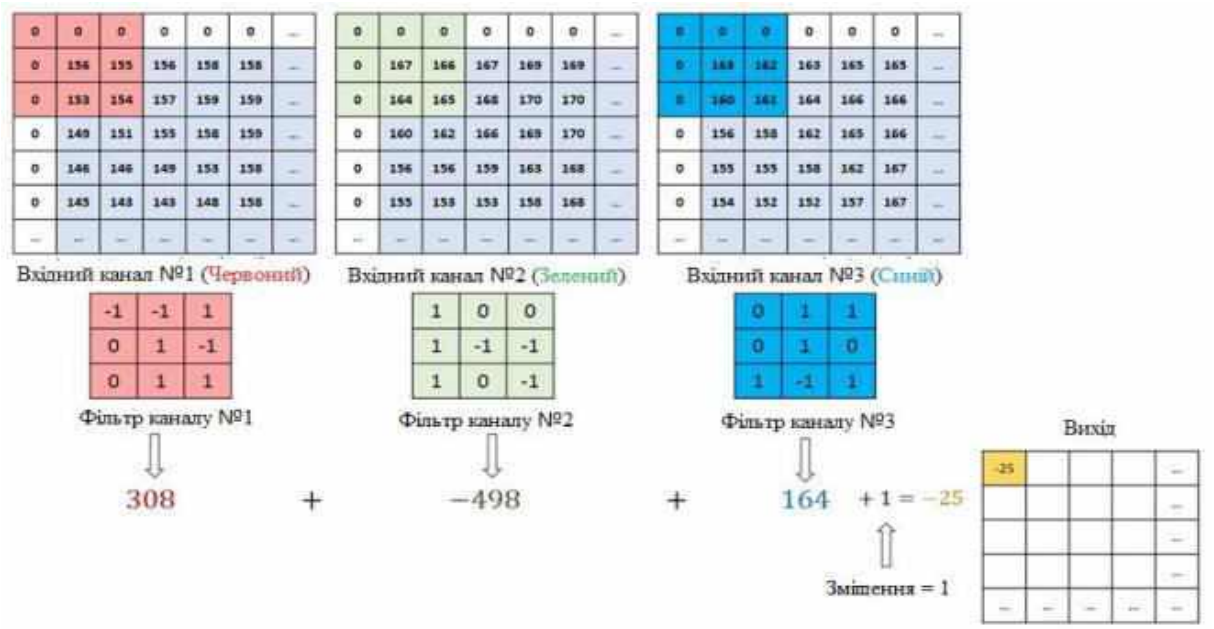


Рисунок 5.21 – Операція згортки із трьома фільтрами

2. Крок (Stride). Регулювання «стрибків» фільтра, параметр, який визначає, на яку відстань (в пікселях) фільтр згортки, зсувається щодо зображення після кожного застосування. Якщо крок дорівнює 1, фільтр переміщується на один піксель за раз, аналізуючи сусідні, частково перекриваючі ділянки. Якщо крок дорівнює 2, фільтр «перестрибує» через один піксель, тобто зсувається одразу на два пікселі. Збільшення значення кроку призводить до того, що фільтр робить менше «зупинок» на зображенні. Як наслідок, це зменшує розмір вихідної карти ознак і може прискорити обчислення.

3. Нульова підкладка (Zero-padding). Вирівнювання розмірів: техніка, при якій по краях вхідної карти ознак (або вихідного зображення) додаються додаткові рядки та стовпці, заповнені нулями. Основна мета такої операції – зробити так, щоб розмір карти ознак на виході згорткового шару збігався з розміром карти на вході, а також для більш ретельної обробки країв зображення.

Нелінійність (ReLU): додавання гнучкості моделі

Після того, як операція згортки обчислила вихідні значення, їх зазвичай пропускають через функцію активації. Основне призначення цього кроку – внести нелінійність у модель. Це критично важливо, оскільки більшість реальних залежностей у даних є нелінійними, і без цього мережа не змогла б їх ефективно вивчати. Однією з найпоширеніших функцій активації в згорткових нейронних мережах є *ReLU (Rectified Linear Unit).* Принцип її роботи дуже простий: усі пікселі на карті ознак, що мають від'ємне значення, замінюються на нуль, тоді як усі додатні значення залишаються без змін.

Операція Макс-пулінгу (Max-Pooling):

Операція Макс-пулінгу (Max-Pooling): зменшення розмірності та узагальнення, поширений крок, який зазвичай виконується після шару активації. Його ключова мета – зменшити просторову розмірність карти ознак (тобто її висоту та ширину), зберігаючи водночас найважливішу інформацію. Це дає кілька суттєвих переваг:

– зменшує обчислювальну складність для наступних шарів мережі, оскільки обробляється менший обсяг даних;

– зменшує кількість параметрів (вагових коефіцієнтів), які мережі потрібно навчити, що зі свого боку допомагає запобігти перенавчанню (ситуації, коли модель надто добре адаптується до тренувальних даних і погано працює на нових, невідомих даних).

Для виконання макс-пулінгу визначається розмір «вікна» (наприклад, 2 × 2 пікселі) та крок, з яким це вікно буде пересуватися по карті ознак. Стандартний підхід полягає у виборі максимального значення з усіх пікселів, що потрапляють у це вікно на кожному його положенні. Наприклад, уявіть, що вікно пулінгу розміром 2 × 2 рухається по карті ознак. Якщо воно накриває

ділянку з пікселями, то операція макс-пулінгу візьме найбільше значення з цих чотирьох (тобто 3), і це число стане єдиним представником цієї ділянки в новій, зменшеній карті ознак. Потім вікно зсувається (наприклад, на 2 пікселі, якщо крок пулінгу дорівнює 2), і процес повторюється для всієї карти ознак (рис. 5.22).

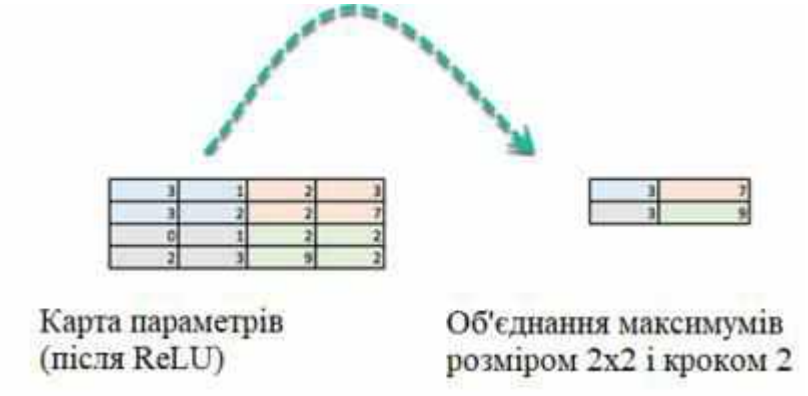


Рисунок 5.22 – Операція Макс-пулінгу, об’єднання через визначення максимума

Також відома операція об’єднання, як середнє, яка більше зменшує розмір карти об’єктів.

Dropout: техніка боротьби з перенавчанням моделі

Dropout (проріджування) – це техніка регуляризації, яка використовується в нейронних мережах для *зменшення перенавчання (overfitting)*. Перенавчання виникає, коли модель надто добре «підлаштовується» під тренувальні дані, запам’ятовуючи навіть шум та випадкові особливості, і в результаті погано узагальнює на нових, раніше не бачених даних. Узагальнена й розширена структура CNN показана на рис. 5.23.

На кожному кроці тренування (для кожної ітерації або мінібатчу даних) шар Dropout випадково «вимикає» (ігнорує) певну частку нейронів у попередньому шарі. Це означає, що ці тимчасово вимкнені нейрони не передають сигнал далі (під час прямого поширення) і не оновлюють свої ваги (під час зворотного поширення помилки). Ймовірність «вимкнення» кожного нейрона задається параметром, який називається *коефіцієнтом проріджування (dropout rate)*. Наприклад, якщо коефіцієнт дорівнює 0.5, то в середньому 50 % нейронів будуть тимчасово деактивовані.

Набір нейронів, що «вимикаються», змінюється випадковим чином на кожній ітерації. Під час тестування (або використання моделі для передбачень) усі нейрони в шарі активні – жоден нейрон не «вимикається».

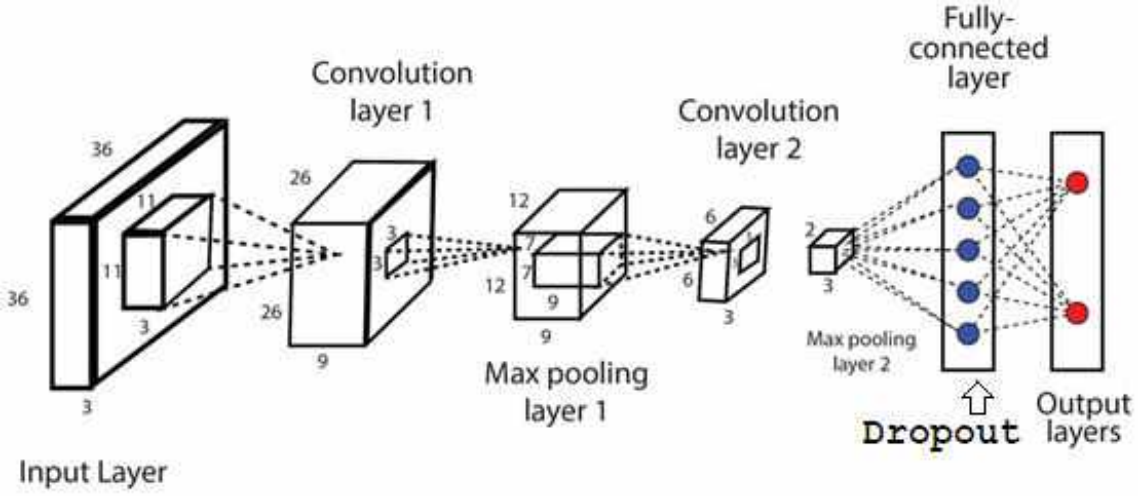


Рисунок 5.23 – Узагальнена й розширена структура CNN

Однак, щоб компенсувати той факт, що під час тренування активними була лише частина нейронів, виходи нейронів цього шару зазвичай *масштабуються*. Їхні активації множаться на ймовірність того, що нейрон *не* був вимкнений під час тренування (тобто *1 – dropout_rate)*. Це гарантує, що очікувана сума вхідних сигналів для наступного шару залишається приблизно однаковою як під час тренування, так і під час тестування.

Існує також популярна техніка «інвертований Dropout» (inverted dropout), де масштабування (ділення на *1 – dropout_rate*) відбувається ще на етапі тренування. У такому випадку на етапі тестування жодних додаткових дій з масштабування не потрібно.

Ефективність Dropout пояснюється тим, що ця процедура:

– зменшує складну коадаптацію нейронів. Нейрони стають менш залежними один від одного. Оскільки будь-який нейрон може бути «вимкнений», інші нейрони змушені навчатися виявляти корисні ознаки самостійно або в комбінації з різними випадковими підмножинами інших нейронів;

– спонукає до вивчення більш робастних (стійких) ознак. Мережа стає менш чутливою до специфічних ваг окремих нейронів;

– діє як тренування ансамблю моделей. Кожна ітерація тренування з Dropout фактично тренує дещо іншу «проріджену» архітектуру мережі. Підсумкова поведінка мережі нагадує усереднення результатів великої кількості різних мереж, що зазвичай покращує узагальнення.

Dropout найчастіше застосовують до повнозв'язних шарів нейронної мережі, хоча існують і варіанти для згорткових шарів (наприклад, Spatial Dropout).

Процедура Softmax

На виході CNN-класифікаторів, також, застосовують спеціальну функцію Softmax, яка є стандартним «останнім кроком», оскільки трансформує вихідні логіти на зрозумілі, порівнювані ймовірності, що спрощує як навчання (через крос-ентропію), так і інтерпретацію результатів. Варіанти застосування Softmax у CNN показано в таблиці 5.4.

Таблиця 5.4 – Застосування Softmax у зорткових нейромережах

Етап	Роль softmax
Фінальний шар класифікатора	Після останнього *Dense* (fully-connected) шару CNN маємо KK логітів. Softmax перетворює їх на ймовірності класів, що зручно для оцінки впевненості, візуалізації та метрик (accuracy, precision, ROC)
Втрати під час навчання	У більшості фреймворків CrossEntropyLoss (PyTorch) / categorical_crossentropy (Keras) поєднує softmax + log-loss в одну стабільну операцію, але концептуально softmax створює розподіл, а log-loss міряє різницю з one-hot міткою
Топ-k / threshold inference	Оскільки вихід нормований, можна брати top-k класів або відсічні пороги (напр., «вивести "невпевнено", якщо pmax<0.6p_{\max}<0.6»)
Пояснюваність (Class Activation Maps, Grad-CAM)	Для Heat-map'ів важливо знати, з якою саме ймовірністю модель обрала клас – це саме softmax-вихід
Мультимодальні або ансамблеві системи	Ймовірності, зведені через softmax, легко усереднювати чи комбінувати з іншими моделями / датчиками

Функція *softmax* є активаційною функцією, але з особливою роллю: її застосовують майже виключно в останньому шарі, щоб отримати нормовані ймовірності класів, а не для створення внутрішньої нелінійності як у звичних прихованих hidden-активаціях.

5.2.3. Приклад використання згорткових нейромереж

Розглянемо побудову, навчання та оцінювання згорткової нейромережі (далі – CNN) на відомому наборі рукописних цифр MNIST.

Опис набору MNIST_784 для класифікаційних задач

MNIST (Modified National Institute of Standards and Technology) – це набір рукописних цифр від 0 до 9, який використовується для задач класифікації, тобто перевірки різних класифікаційних моделей. Він є частиною більшого датасету NIST і містить 70 000 зразків рукописних цифр. Оригінальний набір даних NIST був створений Yann LeCun, Corinna Cortes та Christopher J. C. Burges 1998 року. Набір MNIST_784 є похідним від більш раннього набору NIST, розробленого Національним інститутом стандартів і технологій США, який був створений як зручний і стандартизований набір даних для тестування алгоритмів машинного навчання, особливо у сфері класифікації зображень [34].

Кількість зразків: 70 000.

Кількість класів: 10 (цифри від 0 до 9). Білі цифри на темному фоні.

Формат зображень: 28 × 28 пікселів, чорно-білі (градації сірого).

Розмірність ознак: 784 (28 × 28 = 784 пікселів).

Діапазон значень пікселів: 0–255 (можна нормалізувати до 0–1).

Кожен зразок у MNIST представлений як одномірний вектор розмірністю 784, де кожен елемент відповідає інтенсивності пікселя (від 0 до 255).

Діапазон пікселів: від 0 до 255: 0 – чорний колір. 255 – білий колір. Фон – чорний (пікселі мають значення 0). Цифри – білі або сірі (пікселі зі значеннями від 1 до 255). Цифри написані вручну та відцентровані на зображенні.

Наприклад: порожні ділянки (чорний фон) = 0. Контури цифри = 255 (білий фон). Сірі пікселі (напівтони) = значення між 0 і 255.

Приклад структури (рис. 5.24):

pixel1	pixel2	...	pixel784	label
0	0	...	128	5
34	0	...	255	3
...	...	...	...	...



Рисунок 5.24 – Структура й формат чисел набору MNIST_784

Тобто кожен рядок містить вектор із 784 значень + мітку класу (яке число зображене). MNIST_784 у бібліотеці *scikit-learn* є більш пізньою адаптацією оригінального NIST у форматі, зручному для використання у Python і входить до *sklearn.datasets* і має форму *(n_samples, n_features),* де *n_features = 784.* Цей датасет популярний для навчання моделей класифікації зображень.

У наборі білі цифри на чорному фоні, тому що більшість алгоритмів початку 90-х (SVM, K-NN, багатошарові перцептрони) працювали із вхідними векторами, де відсутність сигналу позначалася 0 (нульове тло = «порожній» піксель). Тобто зображення стають розрідженими, отже, швидше множаться матриці й економиться пам’ять. На той момент більшість бібліотек комп’ютерного зору (*LeImage, Imlib, пізніше OpenCV*) теж вважали 0 = чорний, 255 = білий. Тож інверсія одразу давала сигнал «цифру видно» під час показу через *imshow(..., cmap='gray').* Для розпізнавання фото рукописної цифри (чорне на білому) рекомендується *інвертувати* (наприклад, операцією, *img = 255 - img*) перед подачею в модель, навчену на MNIST, інакше точність буде значно меншою.

Розробка структури нейромережі

Отже, згорткова (конволюційна) нейронна мережа (CNN) зазвичай компонується із таких шарів:

1) згортковий шар (Convolution);

2) функцію активації (ReLU);

3) шар об'єднання (Max-Pooling);

4) повністю зв'язаний шар (Dense) + поцедура Dropuot;

5) функція прогнозування (Softmax).

Для покращення класифікаційних і прогнозних якостей моделі розширена структура CNN може мати декілька різних згорткових шарів, шарів Max-Pooling, застосовувати процедуру *Dropuot* до повно зв'язаного вихідного шару *(Dense).*

До того ж згорткові шари застосовують різні фільтри для субрегіонів зображення чи даних. Функція активації ReLU додає нелінійність, а шари об'єднання (*Max-Pooling*) зменшують розмірність карт функцій. Усі ці шари витягують із зображення важливу інформацію, а карта функцій подається на вихідний повністю зв'язаний шар з функцією *Softmax* для прогнозування. Всі ці параметри повинні задаватись при використанні відповідних функцій і методів.

Створимо версію коду у середвищі Colab, базуючись на поточному стані бібліотека TensorFlow, та використовуючи новий *API* для побудови моделей *Keras. (Повна версія блокноту розміщена у темі курсі на Moodle):*

Крок 1. Імпорт необхідних компонентів бібліотек і визначення версії TF

```
# Лістинг 5.1
Імпорт необхідних компонентів бібліотек
import numpy as np
import tensorflow as tf
from sklearn.datasets import fetch_openml
from sklearn.model_selection import
train_test_split
from sklearn.preprocessing import MinMaxScaler

print("TensorFlow version: ", tf.__version__)
```

```
>> TensorFlow version:  2.18.0
```

Крок 2. Завантаження набору даних

Завантажуємо набір даних MNIST_784 і розділяємо його на навчальну та тестову вибірки у пропорції 80 % – навчальна і 20 % – тестова вибірки). Параметр *random_state=42* буде гарантувати, що поділ буде однаковим під час повторного виконання коду. Це важливо для відтворюваності (лістинг 5.2):

```
                    # Лістинг 5.2
    mnist = fetch_openml("mnist_784")
    print(mnist.data.shape)
    print(mnist.target.shape)
    X_train, X_test, y_train, y_test =
train_test_split(mnist.data, mnist.target, test_size=0.2,
random_state=42)
```

Можемо проглянути перші рядки ознак, виконавши: *print(X_train)* і міток *print(y_train)* датасету. Непотрібно дивуватись, що значення ознак видяться «0», адже це може бути темна частина зображення (*pixel1 – pixel9).* А один рядок, це одномірний вектор розмірністю 784. Рядки не упорядковані, що можна побачити по виведених мітках :

```
Перші 5 рядків датасету (ознаки):

       pixel1 pixel2 pixel3 pixel4 pixel5 pixel6 pixel7 pixel8 pixel9 \
47339   0      0      0      0      0      0      0      0      0
67456   0      0      0      0      0      0      0      0      0
12308   0      0      0      0      0      0      0      0      0
32557   0      0      0      0      0      0      0      0      0
664     0      0      0      0      0      0      0      0      0
Перші 5 рядків датасету (мітки):
47339    5
67456    4
12308    8
32557    0
 664     2
```

Перед моделюванням важливо знати типи даних, якими представлені ознаки й мітки. Виконавши *print(X_train.dtypes), y_train.dtype* можемо переконатись, що ознаки мають тип *int64,* а мітки мають тип *categorial,* тобто мітки потрібно перетворити на числовий тип. Це важливо, оскільки цільові значення представляють класи цифр (від 0 до 9), які зазвичай обробляються як цілочисельні типи.

Виконуємо : *y_test = y_test.astype(int)*

Крок 3. Нормалізація (масштабування) даних для подальшого використання

Цей розділ підготовлює дані для використання в моделі машинного навчання. Він включає масштабування та зміну форми даних зображення. Для нормалізації, у даному прикладі, будемо використовувати мінімаксний масштабувальник *MinMaxScaler.* Цей масштабувальник (*scaler)* трансформує ознаки, масштабуючи кожну ознаку до заданого діапазону, який зазвичай знаходиться між 0 та 1 (лістинг 5.3).

```
                    # Лістинг 5.3
    Масштабування (ноирмалізація)
scaler = MinMaxScaler()
    X_train_scaled =
scaler.fit_transform(X_train.astype(np.float64))
    X_test_scaled =
scaler.transform(X_test.astype(np.float64))
```

X_train.astype(np.float64): перетворює дані навчального зображення *X_train* на 64-бітний тип з плаваючою комою перед масштабуванням. Це часто робиться для запобігання потенційним проблемам з типами даних під час процесу масштабування.

scaler.fit_transform(): викликається на навчальних даних.

fit(): обчислює мінімальне та максимальне значення для кожної ознаки в навчальних даних, а transform() потім застосовує масштабування на основі цих обчислених значень. Результат зберігається в X_train_scaled.

scaler.transform(X_test.astype(np.float64)): масштабує дані тестового зображення *X_test,* використовуючи мінімальні та максимальні значення, отримані з навчальних даних. Важливо використовувати саме *transform()* метод на тестових даних (а не на *fit_transform()),* щоб запобігти витоку даних з тестового набору в процес навчання. Масштабовані тестові дані зберігаються в *X_test_scaled*.

Нормалізація (масштабування) допомагає покращити продуктивність багатьох алгоритмів машинного навчання, особливо нейронних мереж, гарантуючи, що всі вхідні ознаки мають подібний діапазон значень.

Крок 4. Зміна форми масштабованих даних зображення 1D → 2D

Вихідні дані MNIST часто представляють як плоский масив із 784 пікселів на зображення (28 × 28). Vетод *reshape()* використовується для перетворення 1D-масивів значень пікселів назад у 2D-формат зображення, але з додатковим виміром для колірного каналу (лістинг 5.4):

```
# Лістинг 5.4
    Перетворення 1D-масивів у 2D-формат
X_train_scaled = X_train_scaled.reshape(-1, 28, 28, 1)
    X_test_scaled = X_test_scaled.reshape(-1, 28, 28, 1)
```

(–1, 28, 28, 1) визначає нову форму:

–1: означає, що розмір цього виміру визначається, і він буде кількістю зразків (зображень) у наборі даних.

28 та *28*: представляють висоту та ширину зображень (28 × 28 пікселів).

1: представляє кількість кольорових каналів. Оскільки зображення MNIST мають градації сірого, вони мають один канал.

Це переформатування необхідне для підготовки даних до використання зі згортковими нейронними мережами (ЗНМ), які зазвичай очікують вхідних даних у цьому багатовимірному форматі (вибірки, висота, ширина, канали).

Крок 5. Програмування згорткової нейромережі

Код показаний у лістингу 5.5 представляє структуру моделі згорткової нейронної мережі (CNN) із вказаними параметрами за допомогою *API Keras*, який є частиною бібліотеки *TensorFlow*. Лістинг 5.5 визначає структуру нейронної мережі:

model = tf.keras.Sequential (): створюється модель CNN з іменем *model* з лінійним розташуванням;

```
                    # Лістингу 5.5
    Створення моделі CNN
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(28, 28, 1)),
        tf.keras.layers.Conv2D(32, (5, 5),
padding='same', activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Conv2D(36, (5, 5),
padding='same', activation='relu'),
        tf.keras.layers.MaxPooling2D((2, 2)),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(7 * 7 * 36,
activation='relu'),
        tf.keras.layers.Dropout(0.4),
        tf.keras.layers.Dense(10,
activation='softmax')
    ])
```

tf.keras.layers.Input(shape=(28, 28, 1)): вхідний шар. Він визначає форму вхідних даних, які очікує модель. У цьому випадку очікуються зображення розміром 28 × 28 пікселів з 1 каналом (градації сірого);

tf.keras.layers.Conv2D(32, (5, 5), padding='same', activation='relu'): 2D згортковий шар. Він застосовує 32 фільтри розміром 5 × 5 до вхідного зображення. padding='same'Гарантує, що вихідне зображення має ті самі просторові розміри, що й вхідне. reluФункція активації вводить нелінійність;

tf.keras.layers.MaxPooling2D((2, 2)): об’єднання 2D max. Він зменшує кількість семплів вихідного сигналу згорткового шару, беручи максимальне значення в кожному вікні 2 × 2. Це допомагає зменшити просторові розміри та обчислювальну складність;

tf.keras.layers.Conv2D(36, (5, 5), padding='same', activation='relu'): ще один 2D згортковий шар, подібний до попереднього, але з 36 фільтрами;

tf.keras.layers.MaxPooling2D((2, 2)): ще один шар максимального об’єднання для подальшого зниження частоти дискретизації;

tf.keras.layers.Flatten(): шар згладжує багатовимірний вихід згорткового та об’єднуючого шарів в одновимірний вектор. Це потрібно перед передачею даних у повністю зв’язані шари;

tf.keras.layers.Dense(7 × 7 × 36, activation='relu'): щільнозв'язаний (або повністю зв'язаний) шар з 7 × 7 * 36 нейронами. Це число визначається з вихідної форми після об'єднання шарів. Relu – використовується активація;

tf.keras.layers.Dropout(0.4): шар відсіву. Він випадковим чином встановлює 40 % вхідних одиниць на 0 під час навчання. Цей метод допомагає запобігти перенавчанню, зменшуючи залежність від певних нейронів;

tf.keras.layers.Dense(10, activation='softmax'): вихідний шар. Він має 10 нейронів, що відповідають 10 можливим класам (цифри 0–9) у наборі даних MNIST. softmaxФункція активації видає розподіл ймовірностей за класами, де сума ймовірностей для всіх класів дорівнює 1.

Крок 6. Компіляція моделі CNN

Перш ніж модель можна буде навчити, її потрібно скомпілювати. Це включає налаштування процесу навчання (лістинг 5.6):

```
                    # Лістингу 5.6
    Компіляція моделі
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

optimizer='adam': визначає алгоритм оптимізації, який використовується для оновлення ваг моделі під час навчання. Adam – популярний та ефективний оптимізатор.

loss='sparse_categorical_crossentropy': функція втрат, яка використовується для вимірювання ефективності.

моделі. sparse_categorical_crossentropy: зазвичай використовується для задач класифікації з кількома класами, де цільовими мітками є числа (на відміну від векторів, закодованих одним клацанням). Метою навчання є мінімізація цих втрат.

metrics=['accuracy']: визначає метрики, що використовуються для оцінювання продуктивності моделі під час навчання та тестування. accuracyВимірює частку правильно класифікованих екземплярів.

Крок 7. Модельне навчання, тренування моделі CNN

Нижче показано рядок, яким модель навчається на навчальних даних:

```
# Тренування моделі
model.fit(X_train_scaled, y_train, epochs=15, batch_
size=100)
```

model.fit(X_train_scaled, y_train, ...): метод навчає модель, використовуючи надані навчальні дані (`X_train_scaled` для ознак та `y_train` для міток);

epochs=15: задано 15 епох навчання. Епоха – це один повний прохід через весь навчальний набір даних. Тут модель буде навчатися лише протягом 15 епох;

batch_size=100: batch_size визначає кількість зразків на оновлення градієнта. Навчальні дані поділяються на пакети по 100 зразків, а ваги моделі оновлюються після обробки кожного пакета. Навчання пакетами є більш ефективним з точки зору використання пам'яті та також може призвести до швидшого навчання.

Нижче показано дві останні із 15 епох навчання і результати останньої 15-ї епохи, *accuracy: 0.9970 – loss: 0.0102*, показують гарні результати навчання на навчальній вибірці.

```
    Epoch 14/15
    560/560 ━━━━━━━━━━━━━━━━━━━━ 141s 251ms/step -
accuracy: 0.9985 - loss: 0.0048
    Epoch 15/15
    560/560 ━━━━━━━━━━━━━━━━━━━━ 143s 253ms/step -
accuracy: 0.9970 - loss: 0.0102
    <keras.src.callbacks.history.History at
0x785c5d2fa210>
```

Крок 8. Оцінка моделі з використанням тестової вибірки

Оцінювання на тестовому наборі допомагає нам зрозуміти, наскільки модель має властивості узагальнення на нових, невідомих даних, що є вирішальним показником її ефективності в реальному світі (лістинг 5.7):

```
                    # Лістинг 5.7
    Оцінка моделі
    test_loss, test_acc = model.evaluate(X_test_scaled,
y_test, verbose=2)
    print('\nВтрати на тестових даних:', test_loss)
    print('\nТочність на тестових даних:', test_acc)

    >>
438/438 - 9 с - 20 мс/крок - точність: 0,9830 - втрати:
0,0550
    Втрати на тестових даних:   0.054955523461103 44
        Точність на тестових даних: 0.9829999804496765
```

model.evaluate(X_test_scaled, y_test, verbose=2): Цей рядок виконує оцінку.

model – це модель нейронної мережі, яка була попередньо навчена;

X_test_scaled: масштабовані ознаки з тестового набору даних. Це зображення, які модель намагатиметься класифікувати;

y_test: це справжні мітки для тестового набору даних. Це правильні класифікації для кожного зображення в X_test_scaled;

verbose=2: контролює, скільки інформації відображається під час процесу оцінювання. Значення 2 означає, що індикатор виконання не відображатиметься, але інформація про втрати та будь-які показники буде виведена в кінці кожної епохи (хоча на *epochs=1* попередньому кроці навчання оцінювання виконуватиметься лише протягом однієї «епохи»);

Метод *evaluate* повертає список значень, де перше значення – це втрати, а наступні значення – це метрики, які були вказані під час компіляції моделі. У цьому випадку ми вказали лише accuracy як метрику;

test_loss, test_acc = ...: розпаковуються значення, повернуті *model.evaluate();*

test_loss: зберігатиме розраховані втрати в тестовому наборі даних. Втрати – міра того, наскільки «хибними» є прогнози моделі. Менші втрати як зазвичай кращі;

test_acc: зберігатиме розраховану точність у тестовому наборі даних. Точність – це частка правильно класифікованих зображень. Вища точність як зазвичай краща;

print('\nВтрати на тестових даних:', test_loss): рядок виводить обчислене значення *test_loss* в консоль. \nНа початку додано новий рядок для кращої читабельності;

print('\nТочність на тестових даних:', test_acc): рядок виводить обчислене значення *test_acc* в консоль. Знову ж таки, *\n* додає новий рядок.

Оцінюючи ці значення, ми можемо побачити, наскільки добре модель працювала на тестових даних з точки зору втрат та точності. Ця інформація є важливою для розуміння продуктивності моделі та для прийняття рішень про те, чи потрібне подальше навчання або коригування.

Якщо втрати дорівнюють 0, це означає, що результати класифікації моделі ідеально відповідають справжнім міткам, що є ідеальним сценарієм. Вищі значення втрат вказують на гіршу продуктивність.

Значення втрат не обмежене безпосередньо між 0 та 1, як це відбувається з точністю. Його масштаб залежить від конкретної функції втрат та характеру задачі.

Як видно із результатів цього блоку: втрати на тестових даних: 0.055. Втрати зазвичай є невід'ємним числом.

Точність на тестових даних, частка правильно класифікованих зображень: 98 %.

Втрати моделі – міра того, наскільки *неправильними* є прогнози моделі. Вона кількісно визначає штраф за помилки. Конкретна функція втрат, що використовується тут, *sparse_categorical_crossentropy.* Ця функція втрат зазвичай використовується для задач класифікації з кількома класами, подібних до тієї, з якою ви працюєте (класифікація цифр від 0 до 9). Вона вимірює різницю між прогнозованим розподілом ймовірностей за класами та істинним класом

Точність моделі – відображає частку правильно класифікованих зображень. Показник розраховується як частка правильно класифікованих зображень у загальній кількість зображень датасету. Точність завжди має значення від 0 до 1 (або від 0 до 100 %). Точність 1 означає, що модель класифікувала кожен екземпляр правильно, тоді як точність 0 означає, що вона класифікувала кожен екземпляр неправильно

Крок 9. Збереження моделі у форматі Keras і подальше її використання

Час навчання в Colab і на локальній ПЕОМ може бути значним залежно від кількості епох. При кількості епох 15 на локальній ПЕОМ: 16 МГб ОЗУ AMD Ryzen 5 1600X Six-Core Processor, 3.60 GHz – навчання відбувалось 16 хвилин.

Після навчання зберігаємо модель у форматі Keras:

```
# Збереження моделі
model.save('my_mnist_model.keras')  # Збереження моделі
```

Формат *Keras* – означає високорівневий API для створення та тренування нейронних мереж, який є частиною *TensorFlow*. Він надає простий та інтуїтивно зрозумілий спосіб визначення складних моделей. Особливості моделей у форматі *Keras* наявність простих та гнучких API для створення моделей. Keras пропонує кілька способів створення моделей, які відповідають різним потребам:

Sequential API: найпростіший спосіб створення моделей, який підходить для моделей, що складаються з лінійного стеку шарів. Це саме той підхід, що використовується у наданому коді. Шар за шаром ви додаєте компоненти до вашої моделі, наприклад, згорткові шари *(Conv2D),* шари пулінгу *(MaxPooling2D),* шари вирівнювання *(Flatten)* та щільні (повністю зв'язані) шари *(Dense).*

Functional API: цей API дає змогу створювати моделі з більш гнучкою топологією, включаючи моделі з кількома входами, кількома виходами або зі спільними шарами.

Model Subclassing: цей підхід дає змогу створювати моделі через успадкування від класу *tf.keras.Model.* Це дає найбільшу гнучкість і контроль над моделлю, але вимагає глибшого розуміння того, як працює TensorFlow.

Крок 10. Тестування моделі на нових зображеннях

Завантажимо файл архіву з новими зображеннями рукописних цифр *images_archive.zip* у середовище Colab або інше Pytyhon середовище. В архіві розміщені 10 цифр від 0 до 9. Перевіримо, як справиться навчена модель із розпізнаванням нових зображень.

Завантаження архіву нових зображень

Розархівуємо й створимо директорію із зображеннями *path='extracted_files'* (лістинг 5.8):

```
                    # Лістинг 5.8
from zipfile import ZipFile
# Ім'я архіву
zip_filename = 'images_archive.zip'
# Відкриття ZIP-архіву
with ZipFile(zip_filename, 'r') as zipf:
    # Витягування всіх файлів у вказану директорію
    zipf.extractall(path='extracted_files')
```

Завантажимо із директорії для прикладу одне зображення із цифрою «4» і на його прикладі покажемо необхідні процедури. У нашому тестовому архіві цифри написані зазвичай чорним на білому фоні й можуть мати будь-який розмір, але в наборі для дослідження методів класифікації MNIST цифри у форматі – біле на чорному розміром 28 × 28 пікселів. Тому нам потрібно спочатку нормалізувати (рис. 5.25, а), привести зображення до розміру 28 × 28 (рис. 5.25, б), і потім інвертувати для отримання білої цифри на чорному (рис. 5.25, в):



а б в

Рисунок 5.25 – Етапи оброблення числа перед розпізнаванням

Попередньо ми імпортували *matplotlib.pyplot* як *plt.* Тут імпортуємо модуль *image* для роботи із зображеннями із бібліотеки *PIL. У* лістингу 5.9 показано завантаження однієї цифри із архіву та збереження її в змінній *img.*

```
# Лістинг 5.9
```
Завантаження цифри «4» із архіву
```
from PIL import Image
    # Шлях до каталогу із зображенням
    file_path = '/content/extracted_files/number-4.jpg'
    # Використання функції open() для завантаження
зображення
    img = Image.open(file_path)
    # Відображення зображення
    plt.imshow(img)
    plt.axis('off')  # Вимкнення осей для кращого
візуального вигляду
    plt.show()
```

img = Image.open(file_path): тут використовується функція open() з модуля *PIL.Image* для завантаження файлу зображення, вказаного в *file_path*. Завантажене зображення зберігається у змінній *img*.

Нормалізація

Нормалізація в цьому випадку – це такі кроки:

1. Приведення зображення до квадратного розміру. Це дає змогу зменшити вірогідність неправильної класифікації через деформацію зображення перед передачею в модель для аналізу.

```
# Лістинг 5.10
```
Приведення розмірів зображення до квадратної форми
```
    def squarification(img):
        # Отримання розмірів оригінального зображення
        width, height = img.size
        # Визначення більшої сторони
        max_side = max(width, height)
        # Створення нового білого зображення з розмірами
більшої сторони
        new_img = Image.new('RGB', (max_side,
max_side), (255, 255, 255))
        # Визначення координат для вставки оригінального
зображення в центр нового зображення
        x = (max_side - width) // 2
        y = (max_side - height) // 2
        # Вставка оригінального зображення в нове
зображення
        new_img.paste(img, (x, y))
        return new_img
```

2. Приведення до розміру 28 × 28, як у наборі MNIST (лістинг 5.11).

```
                    # Лістинг 5.11
    Приводимо зображення до квадратної форми
    img = squarification(img)
    # Перетворення зображення до потрібного для аналізу
розміру та режиму
    img = img.resize((28, 28)).convert('L')  # 'L'
означає чорно-білий режим
```

3. Вхідні зображення в тренувальній моделі були білими цифрами на чорному фоні, тому додаємо функцію інверсії кольору (лістинг 5.12):

```
                    # Лістинг 5.12
    Функція інвертації зображення
    def invert_image(img):
        img_array = np.array(img)
        # Інверсія зображення
        inverted_img_array = 255 - img_array
        inverted_img_array =
inverted_img_array.astype('uint8')
        if inverted_img_array.ndim > 2:
            inverted_img_array =
inverted_img_array[:, :, 0]
        # Конвертація назад в об'єкт зображення PIL
        inverted_img =
Image.fromarray(inverted_img_array)
        return inverted_img
```

4. Застосуємо інверсію та переглянемо отримане зображення (лістинг 5.13).

```
                    # Лістинг 5.13
    Інвертуємо зображення
    inverted_img = invert_image(img)
    # Відображення зображення
    plt.imshow(inverted_img, cmap='gray')
    plt.axis('off')  # Вимкнення осей для кращого
візуального вигляду
    plt.show()
```



Розпізнавання одного об'єкта. Конвертація отриманого масиву в тензор

Наступним кроком перетворимо картинку в масив кольорів. Нормалізуємо кольори в масиві за рахунок ділення на 255, що перетворить усі кольори на 0 або 1. А також конвертуємо отриманий масив у тензор. Отриманий тензор можна передати моделі для прогнозування класу, до якого належить зображення. Модель у наступному прикладі коду завантажена із попередньо збереженої на ранніх етапах цієї лабораторної роботи.

```
# Перетворення зображення в масив
img_array = np.array(inverted_img)
```

Отриманий тензор можна передати моделі для прогнозування класу, до якого належить зображення (лістинг 5.14).

```
                    # Лістинг 5.14
Розпізнавання (викликаємо збережену модель)
    model =
tf.keras.models.load_model('my_mnist_model.keras')
    predictions = model.predict(img_array)
```

Виводимо результат прогнозування класу об'єкта (класифікації) (лістинг 5.15).

```
                    # Лістинг 5.15
Виведення результату
predicted_class = np.argmax(predictions, axis=1)
print('Predicted class:', predicted_class)

>>
1/1 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
Predicted class: [4]
```

Отже, навчена модель правильно розпізнала цифру «4».

Розпізнавання масиву об'єктів (цифр)

Тепер спробуємо класифікувати кожне із зображень з архіву. Для класифікації інших зображень, що були отримані з архіву, створимо функцію, яка перетворюватиме зображення на тензор, виконуючи всі описані вище процедури нормалізації та модифікації зображення (лістинг 5.16).

```
                    # Лістинг 5.16
    # Функція для конвертації зображення у масив чисел
для аналізу моделлю
    def image_to_array(img):
        # Приводимо зображення до квадратної форми
        squar_img = squarification(img)
        # Перетворення до потрібного для аналізу розміру
та режиму
        resized_img = squar_img.resize((28,
28)).convert('L')
        # Інверсія зображення оскільки
        inverted_img = invert_image(resized_img)
        # Перетворення зображення в масив
        img_array = np.array(inverted_img)
        # Нормалізація зображення
        img_array = img_array / 255.0
        # Розширення розмірності для створення пакету,
що містить
    # лише одне зображення
        img_array = img_array.reshape(-1, 28, 28, 1)
        return img_array
```

Подаємо масив цифр для розпізнавання на вхід навчальної та збереженої моделі. *'my_mnist_model.keras'* із іменем *model* (лістинг 5.17).

```
                    # Лістинг 5.17
```

Підготовка масиву зображень

```
    # Шлях до каталогу, що містить витягнуті зображення
    extracted_dir = 'extracted_files'
    # Отримати список усіх файлів у каталозі
    image_files = [f for f in
os.listdir(extracted_dir)
      if os.path.isfile(os.path.join(extracted_dir, f))]
    # Завантажуємо навчену модель
    model =
tf.keras.models.load_model('my_mnist_model.keras')
    results = []
    # Оброблюємо почергово кожне зображення
    for image_file in image_files:
      file_path = os.path.join(extracted_dir,
image_file)
```

```
        try:
          # завантаження зображення
          img = Image.open(file_path)
          # Попередня обробка зображення за допомогою
раніше визначеної функції
          img_tensor = image_to_array(img)
```

Здійснюємо класифікацію і виводимо результат із значеннями *Confidence.*

```
                    # Лістинг 5.18
    Прогнозування (класифікація)
        predictions = model.predict(img_tensor)
    # Отримати передбачуваний клас
        predicted_class = np.argmax(predictions,
axis=1)
        confidence = np.max(predictions) # Отримати
показник достовірності
        results.append({'Image File': image_file,
'Predicted Class': predicted_class[0], 'Confidence':
confidence})
    # Створити pandas датафрейм з результатів
    results_df = pd.DataFrame(results)
     # Відобразити результати у таблиці
    print("\nClassification Results:")
    results_df
```

Результат класифікації

```
Classification Results:
Image File    Predicted Class  Confidence
number-3.jpg  3                1.00000
number-1.jpg  1                1.00000
number-2.jpg  2                1.00000
number-5.jpg  5                1.00000
number-8.jpg  8                1.00000
number-4.jpg  4                1.00000
number-9.jpg  9                0.99585
number-7.jpg  7                1.00000
number-0.jpg  0                1.00000
number-6.jpg  6                1.00000
```

Результат показав, що згорткова нейромережа успішно із максимальним ступенем впевненості класифікувала всі зображення рукописних цифр.

Питання для самоконтролю

1. Поясніть сутність нейромереж зі згорткою (CNN).
2. Поясніть метод обробки фільтрів у конволютивних ШНМ на основі *max-пулінгу*.
3. Відмінність повнозв'язних моделей лінійних ШНМ і моделей зі згорткою.
4. Наведіть приклади роботи нейромережі зі згорткою.
5. Поясніть архітектуру TensorFlow.
6. Наведіть простий приклад TensorFlow.
7. Опишіть особливості користування TensorBoard.
8. Запишіть приклад нейронної мережі у форматі Keras.
9. Запишіть команду імпорту даних CSV за допомогою Pandas.
10. Поясніть особливості тренування моделі лінійної регресії за допомогою TensorFlow.
11. Завантаження економічних даних (датасет, наприклад MNIST) у середовище машинного навчання.
12. Створення наборів тренування / тестування для конвулятивних ШНМ.
13. Вибір архітектури CNN. Формування вхідного шару CNN.
14. Формування шарів згорткового та об'єднання, повністю пов'язаного шару та шару Logit.
15. Визначення оцінювача з моделлю CNN. Оцінювання прогнозної точності моделі CCN.
16. Підготувати звіт про виконання роботи й розмістити на СДН Moodle.

5.3. Рекурентні штучні нейромережі (RNN) та мережі LSTM

5.3.1. Основні відомості про RNN

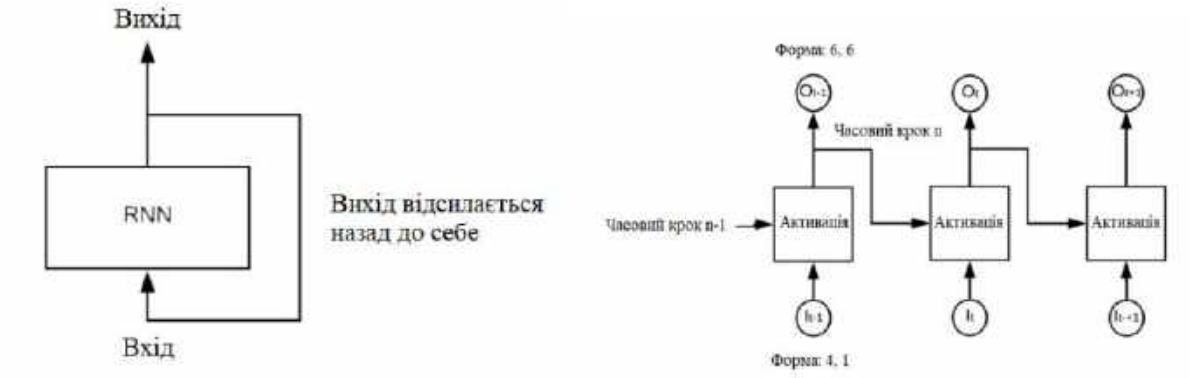
Архітектура рекурентної нейронної мережі (*Recurrent Neural Network – RNN*) є досить нескладною і значною мірою покладається на матричні обчислення. Процес навчання починається з того, що вхідні дані обробляються через множення на початкові, випадково обрані вагові коефіцієнти та додавання зміщення. Далі застосовується функція активації, а отримані результати використовуються для формування прогнозу. Розбіжність між прогнозом та фактичними даними вимірюється за допомогою показника втрат: велике значення втрат свідчить про низьку ефективність моделі.

Щоб «навчити» мережу, її параметри (вагові коефіцієнти) оптимізують, наприклад, за допомогою стохастичного градієнтного спуску, який коригує ваги у напрямку зменшення помилки. Після кожного такого коригування мережа перевіряє свої оновлені «знання» на наступній порції даних. Цей цикл оптимізації повторюється багаторазово, доки помилка не стане мінімально можливою, вказуючи на те, що модель вичерпала потенціал для подальшого навчання на наявних даних. Ключовим обмеженням цього типу мереж є їхня нездатність зберігати інформацію про попередні стани. Це означає, що кожен вхід обробляється ізольовано, що становить проблему при роботі з даними, де послідовність має значення, наприклад, під час прогнозування часових рядів або аналізі тексту, де розуміння контексту залежить від попередніх слів.

Для вирішення проблеми відсутності пам’яті в традиційних нейронних мережах була розроблена нова архітектура – рекурентна нейронна мережа (RNN). Хоча RNN загалом схожа на звичайну нейронну мережу, ключовою відмінністю є наявність стану пам’яті у її нейронах. Математичні операції для інтеграції цієї пам’яті є відносно простими.

Розглянемо для прикладу просту модель з одним нейроном, що отримує послідовність даних. У звичайній нейронній мережі вихід генерується множенням вхідного сигналу на вагові коефіцієнти з подальшим застосуванням функції активації. В RNN цей вихідний сигнал використовується повторно: він повертається на вхід нейрона для обробки на наступних часових кроках. Часовий крок визначає, через який інтервал вихід попереднього обчислення стає вхідними даними для наступного множення матриць.

Так, наприклад (як показано на рис. 5.26, а), мережа з одним нейроном спочатку обчислює добуток матриць вхідних даних та вагових коефіцієнтів, додаючи нелінійність за допомогою функції активації. Отриманий результат є виходом на часовому кроці t-1. Цей самий вихід потім слугує вхідними даними для наступного матричного множення на часовому кроці t.



а б

Рисунок 5.26 – Структура RNN з одним нейроном, а; структура мережі RNN, б.

Отже, контекст попередніх прикладів частково буде зберігатися (рис. 5.26, б).

Але такий підхід безпосереднього змішування сигналів призводив до того, що практична цінність таких мереж була невисока, оскільки сигнали змішувалися і навчити систему було важко. Досить часто вона не досягала конвергенції. І часто зв’язки в системі представляли собою фактично шум.

Ще у 80-х рр. була запропонована архітектура *рекурентної нейронної мережі Елмана.* Взагалі всі типи НМ, які використовують зворотні зв’язки, називаються рекурентними НМ, оскільки фактично це рекурентна функція, яка враховує попередні ітерації.

Архітектура рекурентної НМ Елмана, по суті, подібна до попередніх запропонованих варіантів. Єдина відмінність полягала в тому, що виходи прихованого шару з попередньої ітерації додаємо, змішуємо чи множимо на наступні входи не напряму, а направляємо на додатковий вхідний шар – *шар контексту.*

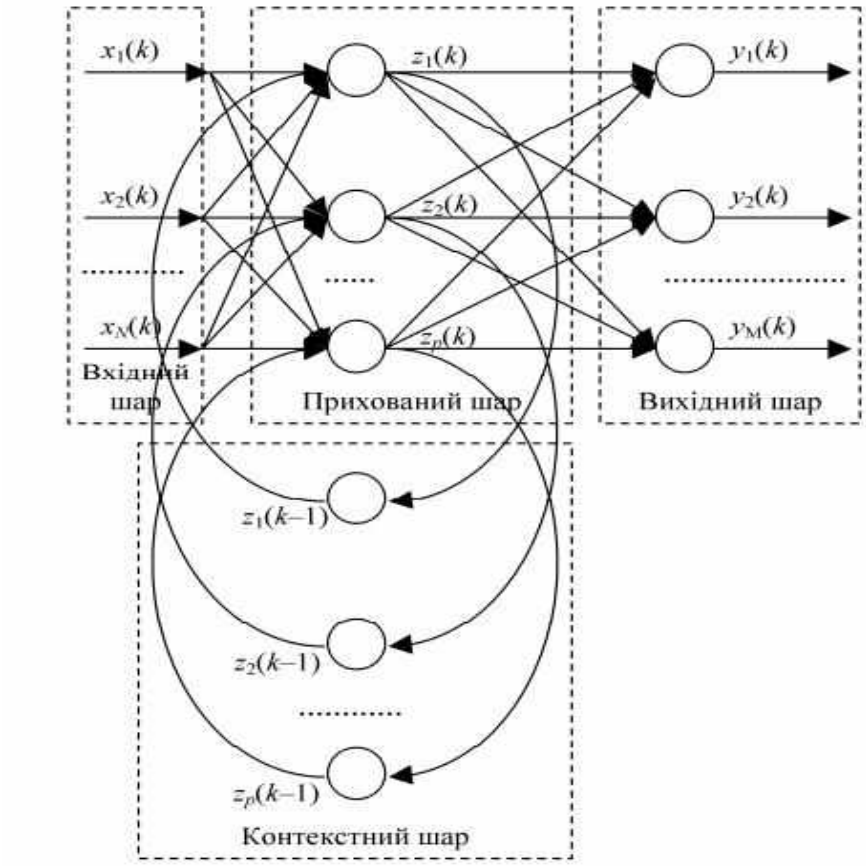


Рисунок 5.27 – Рекурентна мережа Елмана із шаром контексту

Зворотні зв’язки містять елементи затримки сигналу й утворюють контекстний шар нейронів. У мережі Елмана кількість нейронів контекстного і прихованого шарів збігається. Крім того, тут немає власних зворотних зв’язків у нейронів контекстного шару.

5.3.2. Використання, обмеження і покращення RNN

Рекурентні нейронні мережі (RNN) ефективно застосовуються для *аналізу текстових даних,* створення *субтитрів до зображень, визначення тональності висловлювань* (аналізу настроїв) та у *машинному перекладі*. Наприклад, аналізуючи відгуки на

фільм, можна автоматично визначити емоційне забарвлення вражень глядача. Така автоматизація є надзвичайно корисною, скажімо, для кінокомпаній, які не мають ресурсів для ручного оброблення великої кількості відгуків, причому машина часто справляється з цим завданням точніше.

Попри теоретичну здатність RNN обробляти інформацію в часі, на практиці виникають труднощі з *ефективним поширенням інформації через довгі послідовності* (значну кількість часових кроків). Якщо мережа має багато шарів, її навчання стає проблематичним через так звану *проблему зникаючого градієнта*. Сутність її в тому, що під час оновлення вагових коефіцієнтів за допомогою градієнтного спуску, значення градієнтів мають тенденцію зменшуватися при проходженні через глибші шари мережі.

Зрештою, градієнти можуть стати настільки малими, що вагові коефіцієнти перестають оновлюватися, і *модель припиняє навчатися.* Оскільки навчання відбувається саме завдяки змінам градієнта, що впливають на вихід мережі, їх відсутність або незначність означає, що мережа не може досягти прийнятного рівня точності.

RNN у часових рядах

Давайте розглянемо застосування рекурентних нейронних мереж (RNN) для аналізу даних часових рядів. Ключовою особливістю часових рядів є їхня залежність від попередніх значень; іншими словами, минулі дані містять важливу інформацію, яку мережа може виявити та використати. Основна мета прогнозування часових рядів – це оцінка майбутніх значень послідовності (наприклад, цін на акції, показників температури, обсягу ВВП тощо).

Підготовка даних для роботи з RNN та часовими рядами має свої специфічні аспекти. По-перше, завдання полягає в прогнозуванні наступного значення ряду. Тобто використовуємо інформацію з минулого для оцінки значення в момент часу *t+1*. Відповідно, цільова змінна (мітка) для навчального прикладу є вхідною послідовністю, зміщеною на один часовий крок уперед. По-друге, зазвичай на кожному кроці обробки в мережу подається одне спостереження (одне значення ряду). Отже, визначений «часовий

крок» (кількість попередніх значень, що враховуються) визначає довжину послідовності, яка рекурентно обробляється мережею. Наприклад, якщо встановити часовий крок рівним 10, це означає, що вхідна послідовність з десяти послідовних спостережень буде оброблена мережею ітеративно.

5.3.3. LSTM архітектура нейронних мереж

Основні відомості про мережі LSTM

Класична ШНМ Елмана, яку щойно розглянули, досить ефективна з одношаровим чи багатошаровим перцептроном, але й вона має недоліки.

Основний недолік – згасання впливу прикладів по віддаленню. Зазвичай максимальний вплив на відповідь нашої НМ на певній ітерації мають приклади, які були на попередній ітерації, на двох ітераціях назад тощо, і чим далі, тим цей вплив зменшується.

Відповідно, наша НМ має досить коротку пам’ять. На відміну від одношарового перцептрону вона її хоча б має, але згасання є серйозним недоліком. І досить часто бувають ситуації, коли інформація, важлива для правильного прогнозування, знаходиться не на найближчих прикладах, а на 10-20-30 ітерацій назад. Як вирішити проблему забування і одночасно не перетворити НМ в надгігантську архітектуру, яку буде важко натренувати?

Для розв’язання проблеми зникаючого градієнта в RNN дослідники *Хохрейтер, Шмідхубер та Бенджіо* запропонували вдосконалену архітектуру – *Long Short-Term Memory (**LSTM**),* або «довга короткострокова пам’ять». LSTM дозволяє мережі ефективніше *зберігати та передавати релевантну інформацію з минулого* на більш пізні етапи обробки. Так, LSTM використовує досконалішу архітектуру для вибіркового збереження та перенесення інформації в часі.

Архітектура LSTM реалізована у бібліотеці TensorFlow (наприклад, *tf.contrib.rnn.LSTMCell*). Більш детальне обговорення LSTM виходить за рамки цього тексту; зацікавленим рекомендується звернутися до офіційної документації.

Ідея полягає в тому, що, оскільки різні приклади в послідовності мають різний вплив на те, яку відповідь ми повинні видати на поточній ітерації, пропонується розробити метод, який буде надавати певним елементам у контексті в попередніх ітераціях більшу вагу, більший вплив на результат, а певним – менший.

За рахунок чого це можливо?

Якщо візьмемо будь-яку рекурентну НМ і розгорнемо її в часі, то помітимо, що її можна розглядати, як послідовність великої кількості однакових НМ, кожна з яких передає свій результат на вхід наступній. І фактично економимо місце, стискаючи їх всі в одну цілісну структуру.

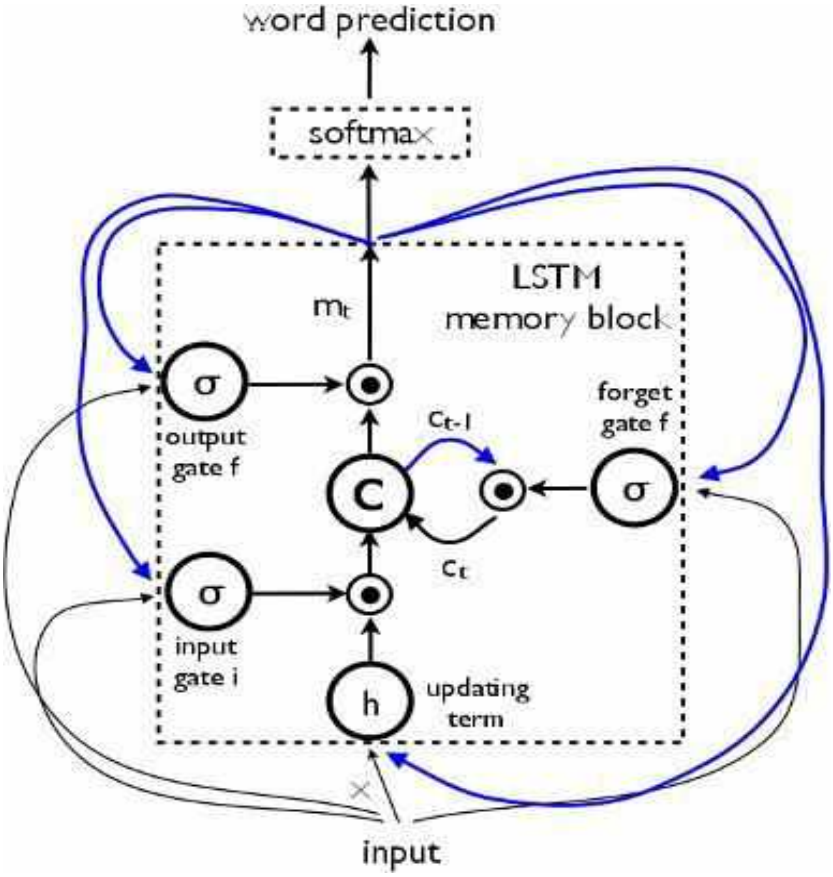


Рисунок 5.28 – Нейромережа LSTM
(довга короткострокова пам’ять)

У нейромережах LSTM пропонується розширити класичну схему рекурентної НМ таким поняттям, як ***gate*** – елемент всередині НМ, який буває ***memory gate*** і ***forget gate***, і який визначає, з якою ***імовірністю*** маємо цей приклад забути чи запам’ятати для наступних ітерацій.

На збереження чи видалення прикладу впливають попередні приклади. Якщо вони свідчать про те, що це слово в реченні відіграє важливу роль для майбутньої класифікації наступних слів, відповідно, надамо вагам на цій ітерації більше значення. Якщо поточне слово відіграє досить слабку роль як прогнозування на подальших ітераціях, зменшуємо відсоток впливу цього прикладу на наступні ітерації.

Наприклад, у розмовних реченнях досить часто певні займенники (слова типу АЛЕ, АБО, ТОМУ) відіграють менш важливу роль для розуміння того, про що взагалі йдеться, ніж такі слова, як ФРАНЦІЯ, ГУЛЯТИ, КОХАТИ, які досить сильно звужують контекст. Тому перші матимуть менший вплив, ніж другі.

LSTM-архітектура в генеруванні тексту

За допомогою LSTM НМ можна вирішити задачу генерування тексту. Уже загальновідомі приклади, коли на вхід ШНМ передається велика кількість тексту – набір книжок (розглядаємо книжку як послідовність індексів слів), і дані послідовності ітерація за ітерацією передаються LSTM-нейромережі, і їй формують задачу прогнозування індексу наступного слова.

У процесі НМ вивчає досить складні закономірності і, відтворивши процес генерування тексту, тобто даючи на вхід певне слово, а не тренувальну вибірку, беремо очікування, що видала мережа, і передаємо його на її вхід. Відповідь змінюється, й ітерація за ітерацією отримуємо послідовність слів. У разі достатньо великої кількості тренувальних прикладів і їх різноманітності, результати тренування LSTM нейромереж вражають. Сьогодні вже нікого не дивують згенеровані ШНМ тексти, схожі до віршів Шекспіра (звісно, на поезії Шекспіра заздалегідь ШНМ повинна була б бути навчена).

Ще одна ефективна властивість LSTM ШНМ та взагалі рекурентних ШНМ це те, що в процесі ітеративної обробки прикладів, мережа постійно на виході видає певний вектор ознак. Тобто, отримавши на вхід певне слово, маємо вектор контексту, отримавши на вхід наступне слово – цей вектор контексту змінюється.

Відповідно, якщо натренували нашу рекурентну ШНМ на достатній кількості прикладів, можемо отримати певний енкодер.

Припустимо, маємо словник на 100 000 слів і НМ, що натренована на задачі прогнозування наступного слова. Надаючи на вхід такої НМ, наприклад слова в закодованому вигляді *one hot*-кодування і розмірності векторів 100 000, в якому тільки один елемент активний, будемо отримувати після кожної ітерації вектор контексту, який буде відмінний від попередніх.

Фактично перекодували наш текст в послідовність стиснених векторів вже не розмірністю 100 000, а, наприклад, розмірністю 100 (якщо розмірність нашої LSTM НМ була 100). Відповідно перекодовуємо кожне слово в певний стовимірний вектор. А надалі послідовність таких векторів можна аналізувати конволютивними ШНМ, оскільки можемо аналізувати біграми, триграми, чотириграми вже на більш високоабстрактному рівні репрезентації, аналізувати вже не просто індекси слів в словнику, а аналізувати їхній контекст.

Тому зазвичай у задачах аналізу емоційної оцінки коментарів на певному ресурсі або статті, робимо тренувальну вибірку, де є декілька тисяч точно відомих позитивних та негативних коментарів.

Створюємо словник, в якому кодуємо кожне слово індексом. Далі, використовуючи рекурентну НМ, можемо створити енкодер, який перекодує наші індекси слів у щось, що має більше сенсу для НМ: не просто номер у словнику, а вектор, в якому відстань між двома близькими векторами буде відповідати відстані між двома досить подібними словами.

Цей процес називається embedding – створення простору, в якому приклади будуть розміщені відповідно до їхньої пов'язаності (рис. 5.29). У випадку мови одним із перших етапів обробки тексту є створення правильних багатовимірних репрезентацій слів. *One hot* репрезентація є хорошою для початкового навчання певної системи, але аналізувати таку репрезентацію досить важко, оскільки нам нічого не говорить інформація, що дане слово у словнику із індексом 1000, а інше – 10000, хоча, можливо, вони синоніми.

Натомість, якщо тренуємо систему, в якій схожі слова будуть відповідати векторам, які розміщені поряд, то, аналізуючи відстань між векторами, можемо аналізувати контекст, який є в реченні.

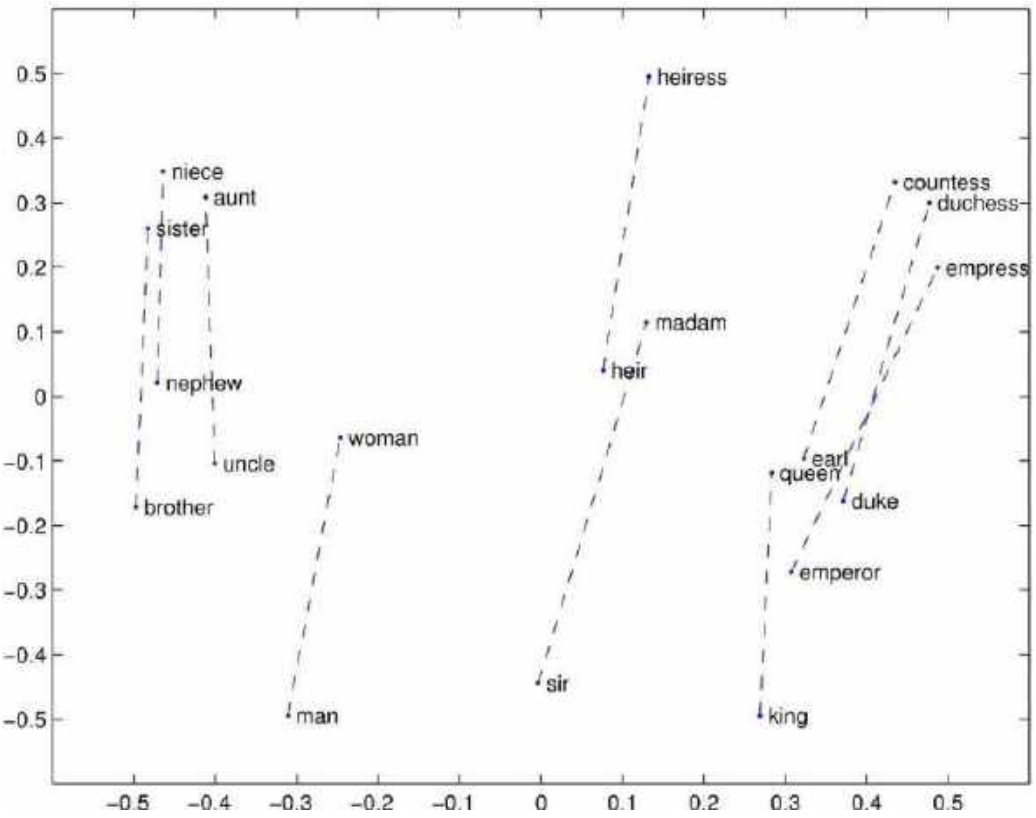


Рисунок 5.29 – Пояснення принципу створення простору *embedding*

Отже, один із перших етапів розуміння тексту – це створення багатовимірної векторної репрезентації слів, а потім – конструювання НМ, які ітерація за ітерацією аналізують послідовності цих багатовимірних векторів слів і знаходять там закономірності.

Далі тренуємо рекурентну НМ класифікатора або над нею можемо сконструювати конволютивну НМ, оскільки фактично, якщо маємо речення із 20 слів і кожному слову, припустимо, відповідає 200-вимірний вектор, відповідно, речення можемо репрезентувати як певну картинку розмірністю 200 на 20 і проходитися, як пам’ятаємо з випадку НМ зі згорткою, певним віконцем по цій картинці й вивчати фільтри певних типових зворотів, мовленнєвих конструкцій, закономірностей. І це робимо вже не вручну, а НМ буде самостійно все ефективніше підбирати фільтри, що корелюють із нашою задачею, методом регресії або методом зворотного поширення помилки. Цей підхід можна використовувати для задач аналізу емоційного забарвлення в тексті.

5.3.4. Приклад у прогнозування часових рядів: RNN+LSTM

Зазвичай під час вибору методу та інструментарію для прогнозування складних часових рядів, прикладом яких є ціни на акції, золото тощо, роблять порівняльний аналіз моделей:

SARIMA (Seasonal Autoregressive Integrated Moving Average)

Переваги. Чудово підходить для рядів із чіткою сезонністю та трендом. Модель є статистичною, а її компоненти (p, d, q) легко інтерпретувати.

Недоліки. Основний недолік – припущення про лінійність залежностей у даних. Фінансові ринки рідко бувають лінійними. Модель погано адаптується до раптових змін волатильності та складних нелінійних патернів.

CNN (Convolutional Neural Network / Згорткова нейронна мережа)

Переваги. Ефективно виявляє короткострокові локальні патерни (наприклад, формації «свічок» за останні кілька днів). Може використовуватися як частина гібридної моделі (наприклад, CNN-LSTM).

Недоліки. CNN не має вбудованої пам'яті для обробки часових послідовностей. Вона аналізує «зріз» даних, але не розуміє довгострокового контексту, що є критичним для ринків.

RNN (Recurrent Neural Network / Рекурентна нейронна мережа) та LSTM

Переваги. Створені спеціально для роботи з послідовними даними. **LSTM** є вдосконаленим типом RNN, який вирішує проблему «зникаючого градієнта», даючи змогу мережі **вивчати та запам'ятовувати довгострокові залежності**. Це ідеально для фінансових даних, де на ціну впливають як вчорашні новини, так і глобальні економічні тренди, що тривають місяцями.

Недоліки. Моделі є більш складними та вимагають більших обчислювальних ресурсів порівняно з SARIMA.

Отже, за результатами такого аналізу LSTM є найкращим вибором для складних часових рядів. Основні аргументи:

довгострокова пам'ять: LSTM за допомогою своїх «вентилів» (gate) може зберігати інформацію впродовж тривалих періодів, що дає змогу враховувати довгострокові тренди та цикли.

обробка нелінійності: на відміну від SARIMA, LSTM чудово вивчає складні та нелінійні взаємозв'язки між минулими та майбутніми цінами.

гнучкість: LSTM може легко працювати з багатовимірними даними. Наприклад, для прогнозування ціни закриття (Close) можна подати на вхід не тільки її минулі значення, а й ціни відкриття (Open), максимуму (High), мінімуму (Low) та обсяги торгів (Volume), що значно збагачує модель.

Розробимо структуру моделі RNN-LSTM, код для її навчання та прогнозування цін на золото на відомому навчальному датасеті *Gold Stock Price Data (goldstockv1.csv)* [35].

Цей набір даних зазвичай містить щоденні ціни на акції, пов'язані з ринком золота. Хоча існують різні варіації, стандартна структура включає історичні дані з біржових торгів. Датасет ідеально підходить для завдань навчання моделей для прогнозування часових рядів, фінансового аналізу та тестування торгових стратегій.

Структура та основні стовпці датасету *goldstockv1.csv*:

Зазвичай, датасет містить такі поля для кожного торгового дня:

Date (Дата): Торговий день, за який наведено дані (наприклад, 2008-01-02).

Open (Ціна відкриття): Ціна акції на момент початку торгової сесії.

High (Максимум): Найвища ціна, якої досягла акція протягом торгового дня.

Low (Мінімум): Найнижча ціна, до якої опускалася акція протягом торгового дня.

Close (Ціна закриття): Ціна акції на момент завершення торгової сесії. Це значення найчастіше використовується для прогнозування.

Volume (Обсяг): Кількість акцій, які були продані та куплені за торговий день. Високий обсяг може свідчити про значний інтерес до активу.

Часовий діапазон у цьому конкретному наборі Gold Stock Prices (Kaggle) *(gold_stock_price.csv)* охоплюють період з 2014 до

кінця 2024 року. Тому важливо розуміти, що модель, навчена на цих даних, «не знає» про ринкові події, що відбулися уже 2025 р. і приводиться як навчальний приклад.

Запрограмуємо таку структуру моделі:

Вхідний шар : Input(shape=(X_train.shape[1], X_train.shape[2])),
Шар LSTM : (128, return_sequences=True),
Шар Dropout : (0.2),
Шар LSTM: (64, return_sequences=True),
Шар Dropout: (0.2),
Шар LSTM: (32),
Шар Dropout: (0.2),
Шар повнозв'язний : Dense(1)

```
# 3.1.Імпортуємо необхідні бібліотеки
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense,
Dropout
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error,
mean_absolute_error

file_path = 'goldstockv1.csv'
data = pd.read_csv(file_path)

# 3.3.Перетворення та сортування за датою
data['Date'] = pd.to_datetime(data['Date'])
data.sort_values('Date', inplace=True)

# 3.4.Використовуємо лише ціну закриття для прогнозу
prices = data[['Close']]

# 3.7.Параметри послідовності
sequence_length = 60
X, y = create_sequences(prices_scaled, sequence_length)

train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

```
    # 3.8.Створення моделі LSTM

    from keras import Input

    model = Sequential([
        # додавання вхідного шару для уникнення
повідомлення UserWarning
        Input(shape=(X_train.shape[1],
X_train.shape[2])),
        LSTM(128, return_sequences=True),
        Dropout(0.2),
        LSTM(64, return_sequences=True),
        Dropout(0.2),
        LSTM(32),
        Dropout(0.2),
        Dense(1)
    ])

    model.compile(optimizer='adam', loss='mse')

    # 3.9.Навчання моделі
    history = model.fit(X_train, y_train, epochs=10,
batch_size=32, validation_data=(X_test, y_test),
verbose=1)
```

```
62/62 ━━━━━━━━━━━━━━━━━━━━ 21s 181ms/step - loss: 0.0028 - val_loss: 0.0019
Epoch 7/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 20s 178ms/step - loss: 0.0032 - val_loss: 0.0023
Epoch 8/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - loss: 0.0023 - val_loss: 0.0072
Epoch 9/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 10s 160ms/step - loss: 0.0027 - val_loss: 0.0043
Epoch 10/10
62/62 ━━━━━━━━━━━━━━━━━━━━ 11s 185ms/step - loss: 0.0027 - val_loss: 0.0014
```

```
    # 3.10. Прогнозування
    predictions = model.predict(X_test)
    predictions =
scaler.inverse_transform(predictions)
    # 3.11. Оцінка точності
    actual_prices = scaler.inverse_transform(y_test)
    mse = mean_squared_error(actual_prices,
predictions)
    mae = mean_absolute_error(actual_prices,
predictions)
    print(f'MSE: {mse}, MAE: {mae}')
```

```
    # 3.12. Візуалізація результатів
    plt.figure(figsize=(12, 6))
    plt.plot(actual_prices, label='Actual Prices')
    plt.plot(predictions, label='Predicted Prices')
    plt.legend()
    plt.title('Stock Price Prediction using LSTM')
    plt.show()

    # Прогнозування даних на 30 днів вперед
    k = 30  # кількість кроків для прогнозу
    input_data = X_test[-1]
    future_predictions = []

    for _ in range(k):
        pred = model.predict(input_data.reshape(1,
input_data.shape[0], input_data.shape[1]))
        future_predictions.append(pred[0][0])
        input_data = np.append(input_data[1:],
[[pred[0][0]]], axis=0)

    future_predictions =
scaler.inverse_transform(np.array(future_predictions)
.reshape(-1, 1))
```

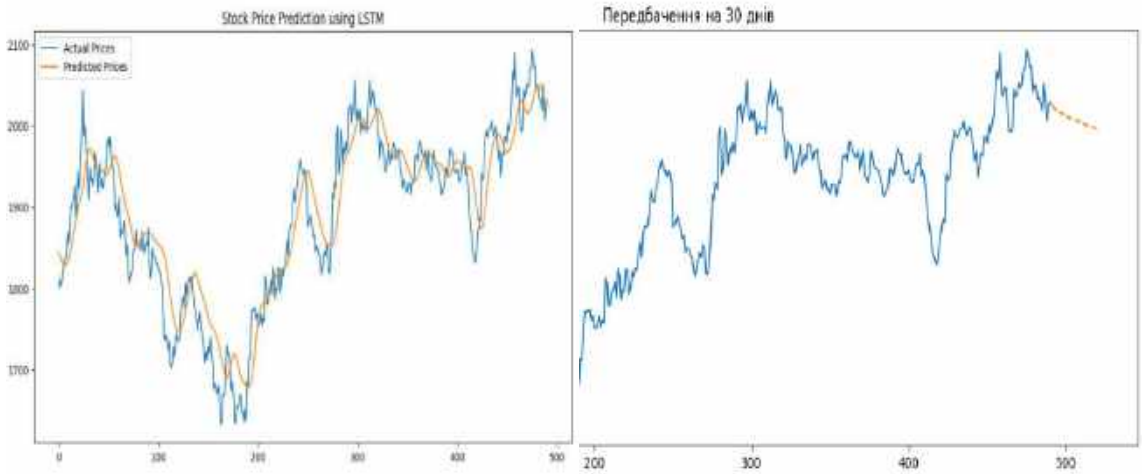


Рисунок 5.30 – Результати навчання моделі

```
    plt.figure(figsize=(15, 5))
    plt.plot(np.arange(len(actual_prices)),
actual_prices, label='Actual Prices')
    plt.plot(np.arange(len(actual_prices),
len(actual_prices) + k), future_predictions,
label=f'Передбачення на {k} днів', linestyle='--')
    plt.legend()
    plt.title(f'Передбачення на {k} днів')
    plt.show()
```

Ця LSTM модель демонструє середню точність у прогнозуванні цін акцій. Візуально спостерігається загальна відповідність основному тренду та вловлювання деяких середньострокових коливань, однак модель має тенденцію до згладжування різких змін, запізнення в прогнозах та неточного передбачення екстремальних значень.

У результаті побудови моделі LSTM для прогнозування значень часового ряду було здійснено прогнозування майбутніх цін на основі останніх доступних даних. Однак згенерований графік прогнозів є рівним та лінійним, без виражених коливань. Це може бути результатом недостатньої складності моделі, що не змогла захопити сезонні чи трендові коливання, або через те, що модель просто «усереднила» прогнозовані значення. Як наслідок, модель демонструє плавний, лінійний графік без значних змін.

Питання для самоконтролю

1. Пояснити сутність LSTM-архітектури нейронних мереж.
2. Пояснити призначення бібліотеки машинного навчання Pandas.
3. Пояснити задачу та сутність врахування контексту в рекурентних ШНМ.
4. Пояснити переваги й недоліки VDG-архітектури ШНМ.
5. Які архітектури (зокрема, LSTM та GRU) були розроблені для вирішення проблеми зникаючого градієнта?
6. Описати основні компоненти LSTM-комірки: вентиль забування, вентиль входу та вентиль виходу (`forget`, `input`, `output gates`). Яка функція кожного з них?
7. Порівняння LSTM і GRU. Чим архітектура GRU (`Gated Recurrent Unit`) відрізняється від LSTM? Які її основні переваги та недоліки порівняно з LSTM?
8. У чому полягає основна відмінність рекурентної нейронної мережі (RNN) від звичайної мережі прямого поширення (Feed-Forward Network)?
9. Для обробки яких типів даних призначені RNN і чому традиційні мережі погано справляються з такими даними?

10. Що таке прихований стан (`hidden state`) в RNN і яка його роль у передачі інформації в часі?

11. Що таке проблеми зникаючого (`vanishing gradient`) та вибухаючого (`exploding gradient`) градієнтів? Чому вони є особливо гострими для RNN?

12. Чому прості RNN (Simple RNN) неефективні для вивчення довгострокових залежностей у послідовностях?

13. Які існують основні типи архітектур RNN залежно від співвідношення вхідної та вихідної послідовностей (one-to-many, many-to-one, many-to-many)? Наведіть приклади завдань для кожної.

14. Наведіть 3–4 конкретні приклади завдань, де успішно застосовуються RNN (наприклад, у NLP, аналізі часових рядів).

15. Як механізм уваги (`Attention Mechanism`) доповнює RNN, особливо в задачах типу `seq2seq` (наприклад, у машинному перекладі)?

16. Які фундаментальні обмеження RNN (включно із LSTM / GRU) привели до появи нових архітектур, як-от Трансформери (Transformers)?

5.4. Інструментарій AI для аналізу геопросторових даних

5.4.1. Актуальність просторового аналізу

Геопросторові дані є всюди, і більшість подій і бізнес-даних пов'язані з місцем розташування. Розташування відіграє вирішальну роль у впливі на навколишнє середовище, події та бізнес. Отже, аналіз і застосування інформації про місцезнаходження та пов'язаної інформації для збору корисної інформації для різних програм є важливим.

Нижче наведено кілька випадків використання, у яких просторовий аналіз допомагає організаціям краще зрозуміти бізнес-можливості, вплив на навколишнє середовище або операційні ризики та отримати цінну інформацію для прийняття обґрунтованих рішень:

Аналіз моделей раку та епідемічних захворювань, як-от холера, SARS та Covid-19.

Пошук гарячих точок злочинності, щоб допомогти поліції прийняти рішення щодо кадрового забезпечення та патрулювання.

Виявлення моделей автомобільних аварій або загибелі пішоходів, щоб допомогти оптимізувати розташування червоних світлофорів і дорожню мережу.

Прогнозування цін на житло на основі даних перепису та інформації про місцезнаходження та допомога у виборі житла з огляду на його близькість до економічних можливостей, шкіл, медичних закладів і доріг для поїздок на роботу.

Корпоративні задачі в геопросторовому аналізі містять компоненти HCM, ERP, CRM та промислові рішення для комунального господарства, оборони та державного сектору. Більшість корпоративних бізнес-даних мають просторовий компонент, тому можливості просторового аналізу та моделювання додають значну цінність цим програмам. Наприклад, системи управління персоналом можуть використовувати кластеризацію для аналізу скарг, порушень техніки безпеки та дисциплінарних гарячих точок. Службові програми можуть навчити моделі виявляти місця

заряджання електромобілів залежно від їхнього великого навантаження.

З 2024 року функціонал Oracle DB поповнився функціоналом Oracle Spatial Artificial Intelligence (Oracle Spatial AI) для задач геопросторового аналізу. Загалом Geospatial Artificial Intelligence (GeoAI) або Spatial AI відноситься до можливостей геопросторового машинного та глибокого навчання, які збирають корисну інформацію та знання, що дозволяє користувачам виявляти, відстежувати, відкривати, класифікувати, прогнозувати та аналізувати ділові події, геопросторові об'єкти та наземні особливості на Землі, пов'язані з місцем розташування [37].

Науковці геопросторових даних і користувачі додатків можуть використовувати Oracle Spatial AI для підготовки й аналізу даних, що зберігаються в Oracle Spatial і OCI-сховищі об'єктів, тренування просторових моделей машинного навчання та застосування цих моделей у різноманітних програмах. Наприклад, користувачі можуть використовувати методи кластеризації, регіоналізації та виявлення аномалій для визначення просторових моделей і аномалій. Користувачі також можуть застосовувати методи просторової регресії та класифікації для прогнозного аналізу.

Oracle Oracle Spatial AI можна використовувати для підвищення цінності в багатьох інших сферах застосування.

5.4.2. Oracle Spatial AI для задач геопросторового аналізу

Oracle Spatial Artifitial Intlligence і OML

Поточний випуск Oracle Spatial AI (2024 р.) надає алгоритми геопросторового машинного навчання для аналізу та моделювання геопросторових векторних даних і подій, пов'язаних із розташуванням. Він надає методи геопросторового машинного навчання, наскрізний робочий процес і відповідні API.

Просторовий штучний інтелект інтегрований з Oracle Machine Learning (OML) і розгортається за допомогою OML4Py в безсерверній хмарній службі Oracle Autonomous Database. Це означає, що ви можете отримати доступ до Spatial AI через інтерфейси та служби OML на своєму примірнику автономної бази даних без сервера.

Oracle Spatial AI постачається як частина Oracle Machine Learning (OML). Він повністю використовує функції OML і доповнює OML, додаючи можливості просторового моделювання. Просторовий штучний інтелект у поєднанні з OML покращує користувацький досвід для науковців із обробки даних і, зокрема, для користувачів геопросторових даних.

Spatial Machine Learning

Spatial Machine Learning – це машинне навчання просторових даних і даних про місцезнаходження для збору корисної інформації та знань для різних програм.

Більшість даних пов'язано з місцезнаходженням. Розташування та взаємовідносини між ними впливають на бізнес і результати подій. Такий вид впливу розташування називається просторовим ефектом.

Існує два типи просторових ефектів, а саме:

Просторова залежність і Просторова неоднорідність.

Просторове машинне навчання включає ці просторові ефекти, беручи до уваги дані про місцезнаходження (на додаток до даних про бізнес або атрибути) для покращення аналітичних і прогнозних моделей.

Одним із підходів просторового машинного навчання є використання традиційних моделей машинного навчання шляхом інтеграції просторово явних незалежних змінних, згенерованих за допомогою операцій розробки просторових ознак (табл. 5.5). Для цієї мети Oracle Spatial AI надає деякі алгоритми Spatial Feature Engineering. Однак цей підхід може бути обмежений тим, які просторово-інженерні функції використовуються, оскільки вони можуть бути не в змозі повністю охопити просторові ефекти в більшості сценаріїв застосування.

Крім використання традиційних моделей машинного навчання, які включають розробку просторових функцій, Spatial AI дає змогу створювати спеціалізовані моделі машинного навчання, призначені для повного включення просторової інформації та просторових зв'язків або просторових ефектів (як-от просторова залежність і просторова неоднорідність).

Таблиця 5.5

Техніки	Алгоритми
Інженерія просторових об'єктів	Трансформер просторового відставання Категоріальний трансформер відставання Трансформер просторових координат Просторовий комп'ютер
Просторова кластеризація	LISA Hotspot DBSCAN з регіоналізацією Агломерація з районуванням
Виявлення просторової аномалії	Фактор локального викиду (LOF)
Просторова регресія	Просторова крос-регресивна модель (SLX) Модель просторового відставання (або модель просторової авторегресії (SAR)) Модель просторової помилки (SEM) Географічний регресор (GR) Географічно зважена регресія (GWR) Просторові режими (OLS_Regimes) Просторові фіксовані ефекти (SFE)
Просторова класифікація	Класифікатор SLX Класифікатор GWR Географічний класифікатор

У таблиці наведено методи просторового машинного навчання та відповідні алгоритми, які підтримує Spatial AI. Використовуючи ці методи, можемо створювати просторові моделі машинного навчання або генерувати просторово спроектовані функції для використання в традиційних моделях машинного навчання для виявлення закономірностей і прогнозування.

5.4.3. Приклади блокнотів моделей Spatial AI на Oracle ML

Робочий процес просторового моделювання на основі ШІ

Основні процедури

Дізнайтеся про робочий процес моделювання просторового штучного інтелекту для керованого та неконтрольованого машинного навчання.

На рис. 5.31 показано блок-схему типового робочого процесу просторового моделювання та аналізу на основі Spatial AI.

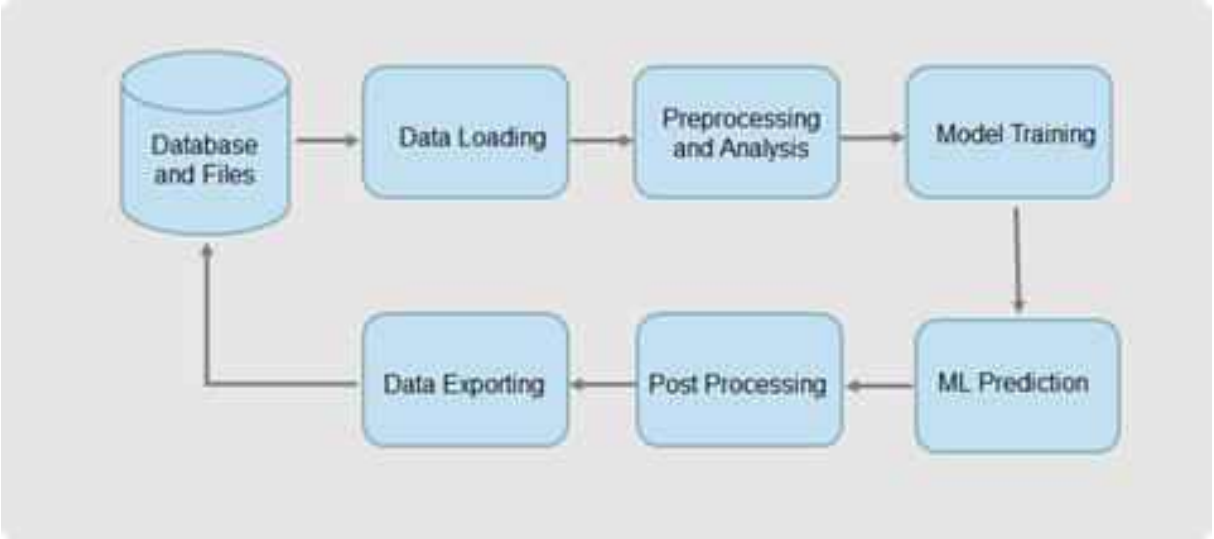


Рисунок 5.31 – Типовий робочий процес просторового моделювання та аналізу на основі Spatial AI

Як показано на попередньому рисунку, робочий процес складається з таких етапів: завантаження даних, попередня обробка, навчання та оцінювання моделей ML, прогнозування за допомогою навченої моделі, постобробка результатів і експорт результатів назад у базу даних або файли в іншому сховищі.

Робочий процес застосовується як до керованого (регресія, класифікація та виявлення аномалій), так і до неконтрольованого (кластеризація та регіоналізація) машинного навчання. Однак в останньому випадку дії з навчання та передбачення об’єднуються в один крок.

Дії робочого процесу, показані на попередньому рисунку, підтримуються такими компонентами:

Доступ до даних: цей компонент допомагає читати та записувати в одне з таких джерел даних – безсерверну автономну базу даних, сховище об’єктів OCI або локальні файли. Однак зауважте, що основною системою зберігання є база даних Autonomous Database Serverless. Один проксі-кадр просторових даних використовується як для завантаження, так і для експорту даних.

Попередня обробка: цей компонент готує, обробляє та доповнює вихідні дані. Він включає заповнення відсутніх значень, масштабування даних, розробку нових функцій і поділ вихідного

набору даних на навчальний набір даних, набір даних перевірки та тестовий набір даних.

Просторовий аналіз: цей компонент забезпечує базову просторову аналітику. Потім вони використовуються разом із або для полегшення моделювання машинного навчання для досягнення кращих результатів або отримання кращої інформації.

Алгоритми машинного навчання штучного інтелекту: цей компонент є основою просторового штучного інтелекту та надає алгоритми та методи для описового та прогнозного аналізу.

Кожен алгоритм надає методи, які дозволяють створювати або навчати моделі та робити прогнози.

Алгоритми регресії, класифікації та виявлення аномалій: це контрольовані алгоритми навчання, у яких кожен алгоритм має відповідний метод для навчання, метод балів для оцінювання та метод прогнозування для прогнозування.

Алгоритми кластеризації та регіоналізації: це алгоритми неконтрольованого навчання, де кожен алгоритм має відповідний метод як для навчання, так і для кластеризації.

Крім навчання та застосування кожної моделі окремо, цей компонент також забезпечує адаптивну просторову регресію. Цей інструмент дає змогу автоматично шукати та оцінювати різні алгоритми регресії та знаходити найкращий алгоритм для конкретної програми.

Постобробка: цей компонент зазвичай охоплює очищення результатів, видалення надмірностей, геоприв'язку просторових даних, перетворення форматів даних, а також створення та зберігання корисних просторових шарів.

Оскільки машинне навчання Spatial AI застосовується лише з просторовими векторними даними, результати є відносно простими без особливої потреби в постобробці. Отримані дані можна безпосередньо зберегти в базі даних за допомогою компонента доступу до даних. Потім ви можете далі обробляти дані за допомогою функцій Oracle Spatial.

Просторовий конвеєр: конвеєр надає можливості для організації та спрощення робочого процесу просторового машинного навчання. Просторовий конвеєр розширює існуючий *scikit-learn*-конвеєр для отримання просторової інформації, як-от геометричні дані та просторові ваги.

5.4.4. Інтеграція Spatial AI з OML

Oracle Spatial AI – це бібліотека Python, розгорнута за допомогою Oracle Machine Learning (OML) і працює на серверах OML4Py, породжених Oracle Database, у середовищі Autonomous Database Serverless.

Просторовий штучний інтелект інтегрований з OML, переважно OML Notebooks і OML4Py. Тому він розглядається як розширення продукту OML з додатковими можливостями просторового моделювання.

На рис. 5.32 показано архітектуру просторового штучного інтелекту Oracle, інтегрованого з OML на безсерверній автономній базі даних.

Можна взаємодіяти з базою даних за допомогою безпечного з'єднання OML. Крім автономної бази даних, Oracle Spatial AI також може отримувати доступ до просторових даних із OCI Object Storage або локальної файлової системи.

Можна отримати доступ до Oracle Spatial AI через інтерфейс користувача OML.

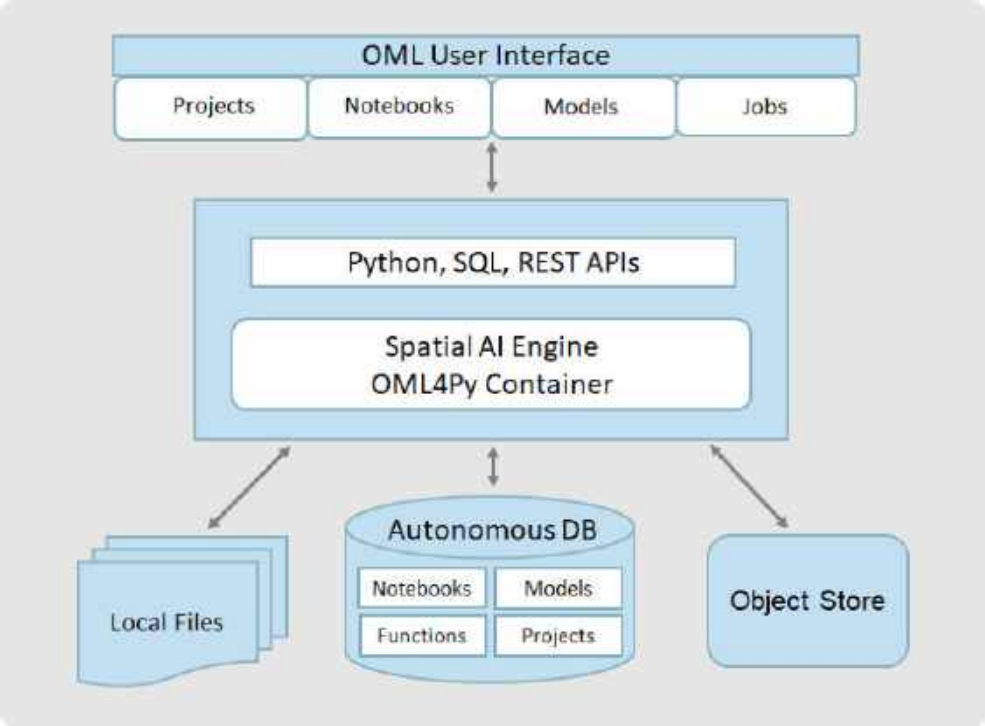


Рисунок 5.32 – Архітектура просторового штучного інтелекту Oracle, інтегрованого з OML

Використовуючи цей інтерфейс, ви можете виконувати такі дії:

– створювати та керуйте проєктами та блокнотами;

– планувати та контролювати завдання з підготовки та прогнозування моделі;

– публікувати функції та моделі Python через OML4Py Embedded Execution REST і SQL API, щоб використовувати масштабованість і високу продуктивність автономних баз даних (Посібник користувача Oracle Machine Learning для Python).

5.4.5. API Python для просторового штучного інтелекту

Можна виконати робочий процес просторового машинного навчання за допомогою API Python, наданого Oracle Spatial AI. API Python містить такі основні пакети:

– *oraclesai.data*: забезпечує уніфіковану структуру для доступу до даних (читання та запис) до бази даних і сховища об'єктів, а також обробки даних у робочому процесі машинного навчання;

– *oraclesai.preprocessing*: надає функції для попередньої обробки даних, розробки функцій і підготовки етикеток;

– *oraclesai.analysis*: забезпечує базовий аналіз даних, зокрема дозволяє користувачам використовувати функціональні можливості Oracle Spatial через Python;

– *oraclesai.clustering*: надає методи й алгоритми просторової кластеризації;

– *oraclesai.outliers*: надає методи та алгоритми виявлення просторових аномалій;

– *oraclesai.regression*: надає методи та алгоритми просторової регресії;

– *oraclesai.pipeline*: надає просторовий конвеєр для об'єднання кроків обробки в робочому процесі машинного навчання;

– *oraclesai.classification*: надає методи та алгоритми просторової класифікації.

(Додаткова інформація про пакети в довіднику Python API для Oracle Spatial AI).

5.4.6. Приклад використання Oracle Spatial AI

Spatial AI надає понад 20 зразків блокнотів, які демонструють, як використовувати методи просторового машинного навчання в різних сценаріях використання (табл. 5.6). Ці блокноти автономні та розміщені в інтерфейсі користувача (UI) OML. У лабораторному практикумі студенти, що отримали облікові дані для доступу до Oracle Machine Learning, отримують прямий доступ до прикладів блокнотів із можливістю їхньої модифікації, доповнення та запуску за допомогою інтерфейсу користувача OML на своїй схемі екземпляра автономної хмарної бази даних.

Таблиця 5.6 – Шаблони блокнотів OML4Py
з використанням Oracle Spatial AI

Завдання	Блокноти
1	2
Доступ до даних	OML4Py Spatial AI PAR Object Store
Інженерія просторових об’єктів	Категоріальний трансформер відставання просторового ШІ OML4Py OML4Py Spatial AI Scoord Transformer OML4Py Spatial AI Spatial Imputer OML4Py Spatial AI Spatial Lag Transformer
Просторова кластеризація	OML4Py Spatial AI Agglomerative Clustering and Regionalization OML4Py Spatial AI DBSCAN Аварії OML4Py Spatial AI Hotspot Clustering
Виявлення просторової аномалії	OML4Py Spatial AI LOF Аварії
Просторова регресія	Географічний регресор просторового ШІ OML4Py OML4Py Spatial AI GWR Regressor OML4Py Spatial AI OLS Regressor OML4Py Spatial AI Spatial Lag and Error Regressors OML4Py Spatial AI Spatial Fixed Effects Regressor

Продовження таблиці 5.6

1	2
Просторова класифікація	Просторовий географічний класифікатор OML4Py AI **OML4Py просторовий класифікатор AI SLX**
Просторовий аналіз	OML4Py Spatial AI Exploratory Analysis OML4Py Spatial AI Spatial Operations
OML4Py	OML4Py Spatial AI Embedded Execution for SQL OML4Py Spatial AI SpatialDataFrame до OML DataFrame OML4Py Spatial AI Save Load Run

У цьому прикладі буде показаний приклад використання блокноту «OML4Py Spatial AI SLX Classifier», Класифікатор SLX (Spatial Cross-Regressive) – просторовий штучний інтелект в машинному навчанні Oracle для Python (OML4Py) [36].

Цей алгоритм виконує логістичну регресію, яка охоплює процес інженерії ознак під час навчання, який обчислює просторове запізнення (середні значення з сусідніх вибірок) певних стовпців і додає їх до набору даних як нові ознаки для використання в навчанні моделі.

Алгоритм додає один або кілька стовпців із просторовим відставанням певних ознак, що представляють середнє значення із сусідніх спостережень. Використання класу SLXClassifier вимагає визначення просторових ваг за допомогою параметра spatial_weights_definition, який встановлює взаємодію між сусідніми спостереженнями.

Для задачі класифікації з кількома класами алгоритм використовує стратегію «один проти решти», навчаючи модель для кожного класу. Одна з поширених проблем цієї стратегії полягає в тому, що вона може призвести до незбалансованого набору даних, де частка елементів одного класу набагато більша, ніж іншого. Для обробки цього сценарію клас SLX Classifier надає такі два методи передискретизації:

– випадковий. Цей метод створює дублікати випадкових вибірок (із заміною) з класу меншості;

– метод синтетичної передискретизації меншості (SMOTE, Synthetic Minority Oversampling Technique).

Цей алгоритм вибирає випадкову вибірку класу меншості a, і з її k найближчих сусідів вибирає випадкового сусіда b. Вектор ab множиться на випадкове число в діапазоні [0, 1], і результат додається до вибірки a, генеруючи новий синтетичний екземпляр [37].

Для використання Oracle Spacial AI в інтерфейсі блокноту Oracle ML завантажуються відповідні компоненти:

Лістинг 5.19

```
%python

import geopandas
import matplotlib.pyplot as plt
import contextily as cx
import sklearn.metrics as metrics
from sklearn.preprocessing import RobustScaler
from sklearn.linear_model import LogisticRegression
from libpysal import weights
import pandas as pd

import oml

# Spatial AI imports
from oraclesai import SpatialDataFrame,
DBSpatialDataset, GeoDataFrameDataset
from oraclesai.pipeline import SpatialPipeline
from oraclesai.classification import SLXClassifier
from oraclesai.weights import SpatialWeights,
KNNWeightsDefinition, DistanceBandWeightsDefinition
from oraclesai import enable_geodataframes
enable_geodataframes(z)
```

У блокнотах на платформі Oracle ML дані зазвичай завантажуються до таблиць автономної бази даних Oracleю Відповідно, і результати аналізу, і моделювання можна також зберігати у відповідних таблицях, які доступні для візуалізації функціоналом Oracle Apex, тощо. На рис. 5.33 показано дані прикладу, виведені в інстефесі приклад блокноту «OML4Py Spatial AI SLX Classifier»:

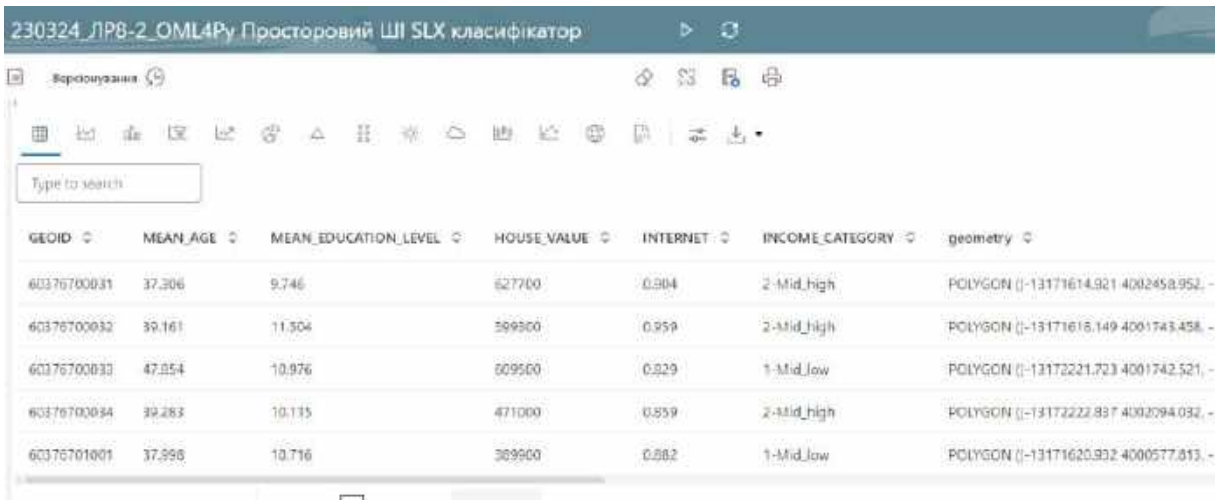


Рисунок 5.33 – Дані прикладу виведені в інстефесі «OML4Py Spatial AI SLX Classifier»

Тренування моделі

```
%python

    classifier_pipeline.fit(X_train.drop(["GEOID"]),
"INCOME_CATEGORY",
slxclassifier__column_ids=["MEAN_AGE","MEAN_EDUCATION
_LEVEL","HOUSE_VALUE","INTERNET"])


    SpatialPipeline(steps=[('scale', RobustScaler()),
('slxclassifier',
      SLXClassifier(balance_method=('smote',),
     random_state=(456,),

patial_weights_definition=<oraclesai.weights.weights_
definit
     ion.KNNWeightsDefinition object at
0x7f28f1790e60>))])
```

Отримання прогнозних значень на тестовій вибірці

```
    array(['3-High', '3-High', '3-High', '3-High',
'3-High', '3-High', '3-High', '3-High', '2-
Mid_high'], dtype='<U10')
```

Карти графіків забарвлюються фактичними та прогнозованими категоріальними значеннями

```
%python

fig, ax = plt.subplots(1, 2, figsize=(15,15))

# Set plot's labels and titles
ax[0].set_title('Test Data');
ax[1].set_title('Prediction - SLX Classifier');

# Plot the choropleth map
X_test.plot(ax=ax[0],
column=X_test["INCOME_CATEGORY"].values,
cmap=plt.get_cmap("Paired"), legend=True,
edgecolor='black', alpha=0.75, linewidth=0.1,
with_basemap=True )
X_test.plot(ax=ax[1], column=predictions_test,
cmap=plt.get_cmap("Paired"), legend=True,
edgecolor='black', linewidth=0.1, alpha=0.75,
with_basemap=True )
```

Вивід статистики моделі (рис. 5.34)

```
%python

print('Score: ',
classifier_pipeline.score(X_train.drop(["GEOID"]),
"INCOME_CATEGORY"))
print('\nSummary:',
classifier_pipeline.named_steps['slxclassifier'].summary)
```

Score: 0.5774545454545454

Summary: Multi-Class Logistic Model Results

label	deviance	llf	aic	bic	D2	adj_D2
2-Mid_high	2799.201489	-1399.600744	2817.201489	-18907.753830	0.100334	0.097709
1-Mid_low	2865.859053	-1432.929527	2883.859053	-18841.096265	0.078911	0.076222
3-High	1525.459910	-762.729955	1543.459910	-20181.495409	0.504476	0.503030
0-Low	1908.180333	-954.090167	1926.180333	-19798.774985	0.378353	0.376539

Parameters (Average Results)

Variable	Est.	STD	Min	Median	Max
constant	-1.637	0.799	-2.626	-1.582	-0.756
MEAN_AGE	0.036	0.382	-0.568	0.126	0.460
MEAN_EDUCATION_LEVEL	0.324	1.879	-1.890	0.003	3.178
HOUSE_VALUE	-0.281	0.736	-0.773	-0.671	0.992
INTERNET	0.238	1.102	-1.498	0.506	1.437
SLX-MEAN_AGE	0.026	0.029	-0.013	0.024	0.068
SLX-MEAN_EDUCATION_LEVEL	-0.037	0.106	-0.207	-0.007	0.075
SLX-HOUSE_VALUE	-0.002	0.035	-0.041	-0.010	0.055
SLX-INTERNET	0.042	0.096	-0.076	0.030	0.182

Рисунок 5.34 – Статистика просторової мультикласової логістичної моделі

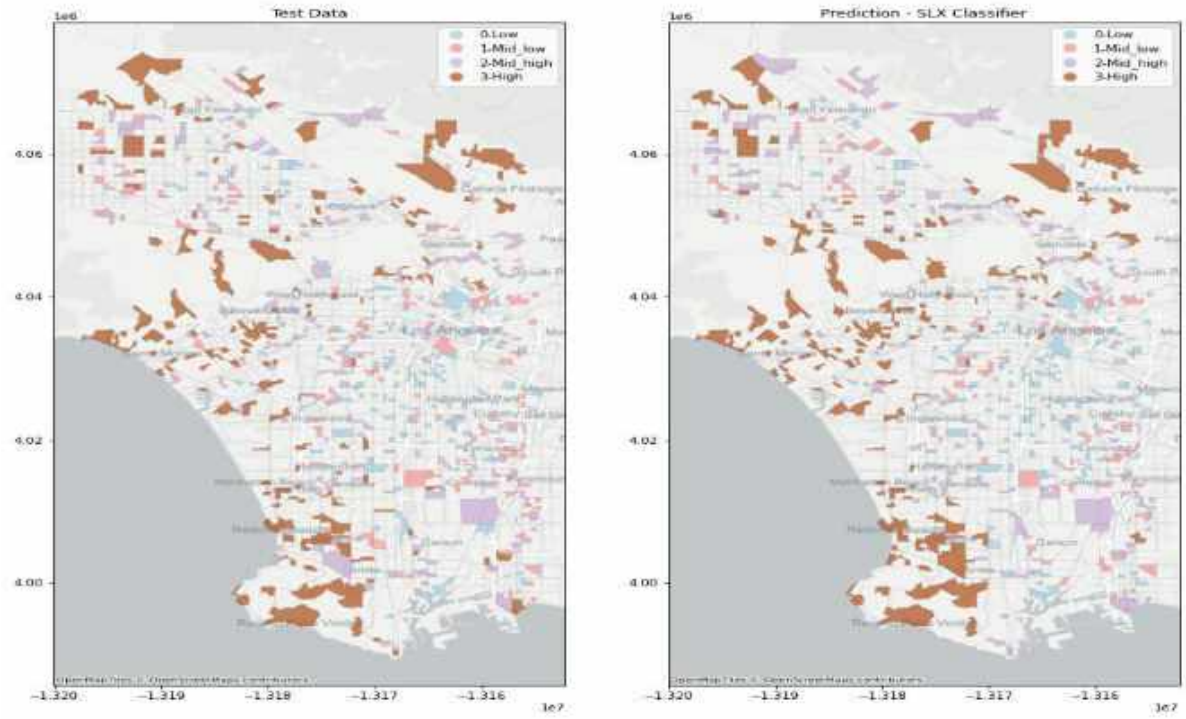


Рисунок 5.35 – Результати роботи алгоритму SLX

Виведення показників точності моделі

```
%python

    print("Test accuracy score:
",metrics.accuracy_score(X_test['INCOME_CATEGORY'].
values, predictions_test))
```

Test accuracy score: 0.5836972343522562

Базовим алгоритмом для SLX Classifier є sklearn LogisticRegression. Отже, щоб порівняти результати прогнозів просторової та непросторової моделі, можна перевірити точність прогнозу на тестовому наборі за допомогою непросторової логістичної регресії.

```
%python
    logreg_classifier =
LogisticRegression(random_state=random_state)
    log_reg_classifier_pipeline =
SpatialPipeline([('scale', RobustScaler()),
('logregclassifier', logreg_classifier)])
    log_reg_classifier_pipeline.fit(X_train.drop
("GEOID"), "INCOME_CATEGORY")
    log_reg_predictions_test =
log_reg_classifier_pipeline.predict(X_test.drop
(["GEOID","INCOME_CATEGORY"]))
```

Non-spatial accuracy score: 0.5604075691411936

```
    print("Non-spatial accuracy score:
",metrics.accuracy_score(X_test['INCOME_CATEGORY'].va
lues, log_reg_predictions_test))
```

Аналіз отриманих результатів просторової мультикласової логістичної моделі (рис. 5.34, 5.35)

Модель намагається не просто спрогнозувати число, а класифікувати об’єкти за чотирма категоріями *(Low, Mid-low, Mid_high, High).*

Проведемо аналіз отриманої статистики.

1. Загальна ефективність моделі

Точність (Score): загальний показник точності моделі становить *57.7 %.* Це означає, що модель правильно класифікує

майже 6 з 10 об'єктів. Це значно краще, ніж випадкове вгадування (25 % для 4 класів), але вказує на те, що модель має значний простір для вдосконалення.

Ефективність по класах (adj_D2): таблиця «Summary» показує найважливішу деталь – модель працює досить нерівномірно для різних класів:

Добре розпізнає клас ***3-High*** *(50.3 %):* модель найкраще розуміє, які чинники визначають «високий» клас.

Добре розпізнає клас 0-Low (37.7 %): вона також непогано справляється з визначенням «низького» класу.

Досить погано розпізнає середні класи: модель пояснює *лише 9.8 %* варіації для класу *2-Mid_high* і всього 7.6% для *1-Mid_low.*

Простими словами: модель може відрізнити «бідні» райони від «багатих», але *плутається і не може чітко розмежувати райони «трохи нижче» та «трохи вище» середнього*. Вони для неї надто схожі.

2. Аналіз впливу чинників (параметрів)

Ця модель є *просторовою (SLX)*, що означає, що вона враховує не тільки характеристики самого об'єкта, але й характеристики його *сусідів*.

MEAN_AGE – середній вік у цій локації.

SLX-MEAN_AGE – середній вік у *сусідніх* локаціях.

Ключові спостереження:

Освіта та інтернет є важливими:

MEAN_EDUCATION_LEVEL (0.324) та INTERNET (0.238): вищий рівень освіти та доступ до інтернету в самій локації значно підвищують ймовірність потрапляння до вищої категорії (наприклад, `High`). Це логічно.

Існує просторовий ефект («ефект сусідства»):

SLX-INTERNET (0.042): наявність інтернету в сусідів також має невеликий, але позитивний вплив. Це класичний приклад просторового «перетікання» (spillover).

Неочевидні результати:

HOUSE_VALUE (–0.281): коефіцієнт для вартості житла *від'ємний*. Це контрінтуїтивно та потребує дослідження. Можливо,

змінна виміряна в зворотному масштабі або є сильна взаємодія з іншими чинниками. Наприклад, у районах з досить високою вартістю житла можуть мешкати пенсіонери (MEAN_AGE), що зміщує їх у нижчу категорію за іншими показниками.

SLX-MEAN_EDUCATION_LEVEL (–0.037): вищий рівень освіти в сусідів має невеликий *негативний* вплив. Це може вказувати на ефект «просторової конкуренції» або сегрегації.

Ефекти неоднорідні: стовпці Min та Max показують, наскільки сильно змінюється вплив змінної для різних класів. Наприклад, коефіцієнт для MEAN_EDUCATION_LEVEL коливається від –1.89 до +3.17, що підтверджує, що освіта по-різному впливає на ймовірність потрапляння до класів «Low» та «High».

3. Висновки та рекомендації

Основна проблема: головна слабкість моделі – нездатність розрізняти два середні класи (*Mid-low* та *Mid-high)*. Вони «розмивають» загальну точність.

Просторові ефекти підтверджено: модель доводить, що характеристики сусідніх територій впливають на локацію, а отже, використання просторових моделей є виправданим.

Неоднозначні предиктори: змінні *HOUSE_VALUE та SLX-MEAN_EDUCATION_LEVEL* мають неочікуваний вплив і потребують глибшого аналізу.

Рекомендації

Спростити класифікацію: спробуйте об’єднати два середні класи (*1-Mid_low та 2-Mid_high*) в один загальний клас «Middle». Модель на 3 класи (*Low, Middle, High*), ймовірно, покаже набагато вищу точність, оскільки їй не доведеться розрізняти досить схожі категорії.

Перевірити дані: уважно дослідіть змінну *HOUSE_VALUE*. Можливо, її потрібно трансформувати (наприклад, взяти логарифм) або перевірити на наявність помилок.

Спробувати іншу модель: оскільки просторові ефекти значущі, можна спробувати більш складні моделі, наприклад *Spatial Durbin Model (SDM)*, яка враховує не тільки вплив сусідніх характеристик (SLX), але й просторову залежність у самій цільовій змінній.

Питання для самоконтролю

1. Чому аналіз просторових даних є актуальним сьогодні? Наведіть приклад бізнес-завдання (наприклад, з рітейлу, логістики чи страхування).

2. Які унікальні інсайти може надати просторовий аналіз порівняно з традиційним аналізом табличних даних?

3. Що таке Oracle Spatial AI? Назвіть щонайменше три типи завдань, які можна вирішувати за допомогою цього інструменту.

4. Яка ключова перевага виконання геопросторового аналізу безпосередньо в базі даних Oracle (in-database)?

5. Опишіть, як відбувається інтеграція між *Oracle Spatial AI* та *Oracle Machine Learning (OML).* Яку роль відіграє кожен компонент у побудові моделі, що враховує просторові ознаки?

6. Уявіть, що ви працюєте з блокнотом *OML (OML Notebook)*. Опишіть загальні кроки для реалізації проєкту: від завантаження даних до створення прогнозної моделі вартості нерухомості на основі її розташування.

7. Наведіть приклад, як можна використати результати просторової кластеризації (наприклад, визначення «гарячих точок» ДТП) як нову ознаку (feature) для подальшого аналізу в моделі машинного навчання OML.

8. Яке основне призначення *Python API* для Oracle *Spatial AI*? Навіщо він потрібен, якщо багато операцій можна виконати через SQL?

9. Які базові просторові об’єкти (наприклад, точка, лінія, полігон) можна створювати та аналізувати за допомогою цього API?

10. Як за допомогою Python API можна виконати таку операцію, як перебування всіх клієнтів у радіусі 5 км від нового магазину? Опишіть логіку, не обов’язково точний код.

ТЕРМІНОЛОГІЧНИЙ ДОВІДНИК (ГЛОСАРІЙ)

Скорочення	Зміст, пояснення
AI	Artificial Intelligence. Штучний інтелект – широка галузь, яка використовує прийоми машинного навчання, моделюючи аспекти й процеси людського пізнання.
Apache Spark	Проєкт Apache, який позиціонується як інструмент для «блискавичних кластерних обчислень» на основі Spark – базового елементу для. великомасштабної паралельної та розподіленої обробки даних
ARIMA	Autoregressive Integrated Moving Average. Авторегресійна інтегрована модель ковзного середнього для прогнозування часових рядів.
Avro	Система серіалізації для виконаних міжмовних викликів та довгострокового зберігання даних.
Big Data	Великі дані – набори структурованої та неструктурованої інформації настільки великих розмірів, що традиційні способи та підходи, засновані на рішеннях класу бізнесової аналітики та системах управління реляційними базами даних.
CAGR	Compound Anual Growth Rate – сукупний середньорічний темп зростання. Виражається у відсотках і показує, на скільки відсотків за рік приростає досліджуваний параметр.
CNN	Convolutional Neural Network – згорткові нейронні мережі.
Colab	Google Colaboratory – платформа машинного навчання.
Data Science	Наука про дані – наука про оброблення й аналіз даних за допомогою методів і інструментарію машинного навчання.

DE Data Engineering – інженерія даних, наука про технології накопичення, збереження, оброблення та підготовку даних для використання.

Flume Розподілений сервіс для колекціонування, збирання та переміщення великих масивів даних.

GraphX Бібліотека для маніпуляцій над графами та виконання з ними паралельних операцій.

Hadoop Набір утиліт, бібліотек і фреймворків, що вільно розповсюджуються для розробки й виконання розподілених програм, які працюють на кластерах із сотень і тисяч вузлів.

HBase Нереляційна розподілена база даних, що виконується HDFS.

HDFS Hadoop Distributed File System. Розподілена файлова система на основі Hadoop.

Hive Розподілене сховище даних, що управляє даними, що зберігаються в HDFS, і надає мову запитів на базі SQL для роботи з цими даними.

ImageNet Великий набір даних для навчання та оцінювання моделей глибокого навчання, що використовується переважно для завдань класифікації зображень та обробки комп'ютерного зору.

IoE Internet of Everything. Більш широке поняття, яке охоплює не лише взаємодію машини з машиною (Machine-to-Machine, M2M), але й взаємодію людей з машинами (Peole-to-Machine, P2M) та взаємодію людей з людьми (People-to-People, P2P).

IoT Internet of Things. Концепція мережі взаємозв'язаних фізичних пристроїв, які мають вбудовані сенсори й програмне забезпечення, що дає змогу здійснювати передачу й обмін даними між фізичним світом і комп'ютерними системами.

JSON	JavaScript Object Notation. Текстовий відкритий формат, оснований на Javascript представленні та призначений для обміну даними в інтернет між сервером та клієнтом або сервером і сервером.
LSTM	Long Short-Term Memory. Довга короткострокова пам'ять – шар нейромережі.
Mahout	Набір бібліотек для машинного навчання в середовищі Big Data.
Map Reduce	Модель розподілених обчислень, призначена для паралельних обчислень над великими (до кількох петабайт) обсягами даних.
Max-Pooling	Макс-пулінг – зменшення розмірності та узагальнення інформації про зображення.
ML	Machine Learning. Машинне навчання – набір алгоритмів і технологій, що навчають комп'ютерні моделі на прикладах і застосовують їх для рішення прикладних задач штучного інтелекту.
MNIST	Modified National Institute of Standards and Technology. Набір рукописних цифр від 0 до 9, який використовується для задач класифікації.
NoSQL DB	NoSQL баз даних, які відрізняються від реляційних моделей даних, підходом до розподіленості та реплікації, завдяки чому можуть у різній мірі підходити під певні види конкретних завдань Big Data.
OML	Oracle Machine Learning – платформа машинного навчання Oracle.
Oozie	Сервіс для запису та планування завдань Hadoop.
Open Data	Відкриті дані – концепція, яка відображує ідею, що дані визначені законодавством мають бути доступні для легкої обробки програмними засобами, тобто бути доступними для читання комп'ютером.

Pig	Мова управління потоком даних та виконавче середовище для аналізу великих обсягів даних.
ResNet	Нейромережа Residual Network (2015 р). Використовує residual blocks – прямі з'єднання (skip connections), які дають змогу уникати затухання градієнта (vanishing gradient).
RNN	Recurrent Neural Network – рекурентна нейронна мережа.
SARIMA	Сезонна авторегресійна інтегрована модель ковзного середнього.
Spark	Базовий елемент для великомасштабної паралельної та розподіленої обробки даних.
SparkSQL	Компонент Spark, який підтримує запити даних за допомогою SQL або за допомогою Hive Query Language.
Spatial AI	Функціонал штучного інтелекту для аналізу геопросторових даних на платформі Oracle ML.
Sqoop	Інструмент для пересилання даних між структурованими сховищами та HDFS.
VGG16	Згорткова нейронна мережа (CNN), розроблена дослідниками з Visual Geometry Group (VGG) Оксфордського університету.
YARN	Yet Another Resource Negotiator. Модуль, що з'явився в Hadoop 2.0, який відповідає за управління ресурсами кластерів та планування завдань.
ZooKeeper	Розподілений координаційний сервіс, що надає примітиви для побудови розподілених програм.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Економічна стратегія України 2030. Український інститут майбутнього. URL : https://strategy.uifuture.org/index.html (дата звернення: 07.11.2024).

2. План «Диджиталізація». URL : https://forbes.ua/innovations/plan-didzhitalizatsiya-na-vidnovlennya-ta-tsifroviy-rozvitok-do-2025-roku-khochut-zaluchiti-i-vitratiti-692-mlrd-grn-kudi-pidut-groshi-05072022-7000 (дата звернення: 07.05.2024).

3. IDC (International Data Corporation). URL : https://www.idc.com/ (дата звернення: 07.05.2025).

4. Класифікація відкритих даних «5 зірок Open Data». URL : 5-star Open Data (5stardata.info) (Дата звернення: 07.05.2025).

5. SpeedTest. URL : https://www.speedtest.net (дата звернення: 07.05.2025).

6. Cisco Annual Internet Report (2018–2023). White Paper. URL : https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (дата звернення: 07.05.2025).

7. Директива (ЄС) 2019/1024 Європейського Парламенту і Ради від 20 червня 2019 року про відкриті дані та повторне використання інформації. URL : https://zakon.rada.gov.ua/go/984_036-19 (дата звернення: 04.04.2024).

8. Рейтинг Відкритих даних (Open Data Index). URL : https://index.okfn.org/ (дата звернення: 13.01.2025).

9. 11 типів сучасних баз даних: короткий опис, схеми і приклади БД. URL :_https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikladi-bd (дата звернення: 10.05.2025).

10. Архітектура Hadoop. URL : https://hadoop.apache.org/docs/ r1.2.1/hdfs_design.html#Large+Data+Sets (дата звернення: 04.04.2025).

11. Ланде Д. В., Субач І. Ю., Гладун А. Я. Оброблення надвеликих масивів даних (Big Data) : навчальний посібник. Київ, 2021. 168 с.

12. Олещенко Л. М. Технології оброблення великих даних: конспект лекцій з дисципліни «Технології оброблення великих даних» [Електронний ресурс] : навч. посіб. для студ. спеціальності 121 «Інженерія програмного забезпечення» (освітня програма «Інженерія програмного забезпечення мультимедійних та інформаційно-пошукових систем») / КПІ імені Ігоря Сікорського. Електронні текстові дані (1 файл: 5,55 Мбайт). Київ : КПІ імені Ігоря Сікорського, 2021. 227 с.

13. Oracle Big Data Lite Virtual Machine. URL : https://www.oracle.com/database/technologies/bigdatalite-v411.html (дата звернення: 13.01.2025).

14. Use Oracle Big Data SQL to Access Data. URL : https://docs.oracle.com/en/bigdata/big-data-sql/4.1.2/bdsug/bigsql.html#GUID-60F61D46-EAF7-463E-A62C-C1E08EA58726 (дата звернення: 29.05.2025).

15. AWS. How has the Hadoop ecosystem evolved? URL : https://aws.amazon.com/what-is/hadoop/ (дата звернення: 29.05.2025).

16. Microsoft Azure. URL : https://learn.microsoft.com/en-us/azure/storage/blobs/data-lake-storage-introduction (дата звернення: 29.05.2025).

17. Google. Build next generation agentic applications. URL : https://cloud.google.com/ (дата звернення: 29.05.2025).

18. Мокін В. Б., Дратований М. В. Наука про дані: машинне навчання та інтелектуальний аналіз даних [Електронне видання] : навчальний посібник. Вінниця : ВНТУ, 2024. 263 с.

19. Machine Learning Tutorial. URL : https://www./machine-learning/ (дата звернення: 29.05.2025).

20. Штовба С. Д., Козачко О. М. Machine learning: стартовий курс [Електронне видання] : навчальний посібник. Вінниця : ВНТУ, 2020. 81 с.

21. Keras. A superpower for ML developers. URL : https://keras.io/ (дата звернення: 29.05.2025).

22. PyTorch. Get Started. URL : https://pytorch.org/ (дата звернення: 29.05.2025).

23. Caffe. Deep learning framework. URL : https://caffe.berkeleyvision.org/ (дата звернення: 29.05.2025).

24. Deeplearning4j Suite Overview. URL : https://deeplearning4j.konduit.ai/ (дата звернення: 29.05.2025).

25. TensorFlow.An end-to-end platform for machine learning. URL : https://www.tensorflow.org/ (дата звернення: 29.05.2025).

26. Google Colab. Ласкаво просимо до Colab! URL : https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=Wf5KrEb6vrkR (дата звернення: 29.05.2025).

27. Linkage. Scipy.cluster.hierarchy. URL : https://docs.scipy.org/doc/scipy-1.15.2/reference/generated/scipy.cluster.hierarchy.linkage.html#linkage (дата звернення: 29.05.2025).

28. Dendrogram. Scipy.cluster.hierarchy. URL : https://docs.scipy.org/doc/scipy/reference/generated/scipy.cluster.hierarchy.dendrogram.html (дата звернення: 29.05.2025).

29. Hyndman Rob, Koehler Anne B., Ord J. Keith, Snyder Ralph D. Forecasting with Exponential Smoothing: The State Space Approach. Springer-Verlag Berlin Heidelberg, 2008. P. 125.

30. JOrd, J. K., et al, Time Series Forecasting: The Case for the Single Source of Error State Space Approach, Working Paper, Department of Econometrics and Business Statistics, Monash University, VIC 3800, Australia, April 2, 2005. URL : https://www.google.com.ua/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwitqpaX4dz_AhWZdcAKHdJQARwQFnoECA0QAQ&url=https%3A%2F%2Frobjhyndman.com%2Fpapers%2FSSOE.pdf&usg=AOvVaw1jaZ6W7nz69dCQNABFDvas&opi=89978449 (дата звернення: 29.12.2024).

31. Redych, O. and Boichuk, R. Comparative Analysis of Holt-Winters Algorithms on the Oracle Machine Learning Platform. *Proceedings of the 12th International Conference on Applied Innovations in IT,* Koethen, Germany, 19 March 2024. Volume 12, issue 1, pp. 71–77. DOI 10.25673/115644 ; PPN 1884680062 (дата звернення: 29.04.2025).

32. Інструментарій завантаження й зберігання даних із зовнішніх джерел у середовищі Colab. URL : https://colab.research.google.com/notebooks/io.ipynb#scrollTo=7Z2jcRKwUHqV

33. Передавальна функція штучного нейрона. URL : https://surli.cc/gswxsw (дата звернення: 25.03.2025).

34. Набір MNIST (Modified National Institute of Standards and Technology). URL : https://www.kaggle.com/datasets/aadeshkoirala/mnist-784/data (дата звернення: 25.03.2025).

35. Харченко В. О. Моделювання нейронних мереж : навч. посіб. Суми : СДУ, 2024.

36. Набір даних (Gold Stock Prices) для аналізу та прогнозування цін на акції, пов'язані із золотом. URL : https://www.kaggle.com/datasets/sahilwagh/gold-stock-prices (дата звернення: 25.03.2025).

37. Using Oracle Spatial AI on Autonomous Database Serverless. SLX Classifier. URL : https://docs.oracle.com/en/cloud/paas/autonomous-database/serverless/cspai/slx-classifier.html (дата звернення: 25.03.2025).

38. Walker, B. (2019). Synthetic Minority Over-sampling Technique (SMOTE) from Scratch. Medium.

39. Феняк Р., Виклюк Я. Підходи до автоматизації проектування архітектури програмного забезпечення з використанням штучного інтелекту. *Вісник Хмельницького національного університету. Серія: Технічні науки.* 2024. Том 333, № 2. URL : https://doi.org/10.31891/2307-5732-2024-333-2-5

40. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition by Aurélien Géron Released October 2022. Publisher(s): O'Reilly Media, Inc. URL : https://www.oreilly.com/library/view/hands-on-machine-learning/9781098125967/ (дата звернення: 29.05.2025).

41. An Introduction to Machine Learning with Python (O'Reilly) by Andreas C. Mueller and Sarah Guido. Copyright 2017 Sarah Guido and Andreas Mueller, 978-1-449-36941-5.

42. CSVkit-бібліотека для Python за численними маніпуляціям з CSV файлами і безліччю інструментів для командного рядка. URL : https://github.com/wireservice/csvkit (дата звернення: 29.05.2025).

43. Textql інструмент із запуску SQL запитів на CSV / TSV файлах. URL : https://github.com/dinedal/textql (дата звернення: 29.05.2025).

44. PapaParse парсер дуже великих CSV файлів. URL : https://github.com/mholt/PapaParse (дата звернення: 29.05.2025).

45. Tablib-бібліотека для роботи з будь-якими табличними даними, включаючи CSV. URL : https://github.com/kennethreitz/tablib (дата звернення: 29.05.2025).

ДОДАТКИ

Додаток А

Порівняльна таблиця класичних методів машинного навчання

Метод	Принцип роботи	Основні переваги	Основні недоліки	Типові сфери застосування
1	2	3	4	5
Лінійна регресія	Моделює лінійну залежність між вхідними ознаками та неперервною цільовою змінною, підбираючи рівняння прямої (гіперплощини), що найкраще відповідає даним	- Проста в реалізації та легка для інтерпретації результатів (коефіцієнти моделі). - Швидке навчання навіть на великих наборах даних. - Ефективна, якщо справжня залежність близька до лінійної	- Припускає лінійну форму зв'язку між ознаками і виходом (не враховує складних нелінійних взаємодій). - Чутлива до викидів у даних (поодинокі аномальні точки сильно впливають на модель). - Може недонавчатися, якщо реальна залежність нелінійна або модель надто спрощена	- Прогнозування кількісних показників: ціни на нерухомість, обсяги продажів, попит тощо. - Виявлення й аналіз залежностей між змінними у прикладних дослідженнях
Логістична регресія	Лінійний метод класифікації, що оцінює ймовірність належності об'єкта до класу за допомогою логістичної сигмоїдальної функції; використовується переважно для бінарної класифікації (може бути розширений до мультикласового випадку)	- Інтерпретовані параметри моделі (ваги показують вплив ознак). - Менш схильна до перенавчання під час використання регуляризації (L1 / L2). - Дає виходи як ймовірності, що зручно для прийняття рішень	- Вимагає приблизної лінійної відокремлюваності класів у просторі ознак (нелінійні межі не моделює без перетворення ознак). - Може перенавчатися на малих вибірках з великою кількістю ознак без належної регуляризації. - Чутлива до аномальних спостережень і багатоколінеарності між ознаками	- Класифікація з двома класами: спам / не спам, наявність захворювання (діагностика), відтік клієнта тощо. - Скорингові моделі у фінансах (кредитний скоринг), коли потрібна ймовірність події

1	2	3	4	5
Метод опорних векторів (SVM)	Класифікаційний алгоритм, що шукає оптимальну гіперплощину, яка максимально розділяє класи, збільшуючи відстань (маржу) між найближчими точками різних класів. Для нелінійних випадків використовується «трюк з ядром» – відображення даних у простір більшої розмірності, де їх можна розділити лінійно	- Ефективний на високовимірних даних (багато ознак). - Працює навіть за відносно малого обсягу навчальної вибірки (потребує лише опорних векторів). - Може моделювати нелінійні межі між класами через ядра (RBF, поліноміальні тощо)	- Обчислювально затратний на великих наборах даних (повільне навчання при великій кількості об’єктів). - Вимагає ретельного налаштування ядра та параметрів (C, параметри ядра) для досягнення високої якості. - Не надає безпосередньо ймовірності класифікації (потрібні додаткові методи для оцінки довіри в прогноз)	- Високовимірні задачі класифікації: розпізнавання текстів (класифікація документів), зображень (розпізнавання образів), біоінформатика (класифікація генів / білків). - Сценарії, де кількість ознак велика відносно кількості спостережень
K-ближчих сусідів (KNN)	Непараметричний метод «лінивого» навчання, що для нового прикладу знаходить *k* найближчих (за визначеною метрикою відстані) сусідніх точок навчального набору та прогнозує клас шляхом голосування більшості їхніх міток (або усереднює значення сусідів для регресії).	- Вкрай простий у реалізації та розумінні алгоритм. - Не потребує явної фази навчання: нові дані просто додаються до бази і враховуються при прогнозі. - Мінімум гіперпараметрів (лише *k* та метрика відстані); універсальний для класифікації і регресії	- Погано масштабується на великих даних: щодо класифікації нового об’єкта потрібно обчислювати відстань до кожної точки навчання (повільний прогноз). - Чутливий до «прокляття розмірності»: з ростом числа ознак метрика відстані втрачає значимість, точність падає. - Може працювати гірше за наявності шуму або нерелевантних ознак, оскільки враховує всі ознаки без ваг	- Рекомендаційні системи (пошук схожих користувачів або продуктів на основі метрики схожості). - Фільтрація спаму та класифікація текстів за схожістю до прикладів у навчанні. - Базові задачі розпізнавання образів, де потрібен простий базовий підхід для порівняння

1	2	3	4	5
Наївний байєсівський класифікатор	Ймовірнісний метод класифікації, що застосовує теорему Байеса для обчислення *P (клас / ознаки)* – апостеріорної ймовірності класу за даними ознаками. Називається «наївним», бо передбачає статистичну незалежність усіх ознак між собою (зафіксувавши клас)	- Досить швидкий та простий у тренуванні і застосуванні (обчислює лише частоти або ймовірності). - Ефективний на даних з великою кількістю ознак (напр. текстових), навіть якщо навчальна вибірка невелика. - Несхильний до перенавчання: мінімум параметрів, є вбудовані методи згладжування для рідкісних випадків	- Припускає незалежність ознак, що майже ніколи повністю не виконується в реальних даних (ігнорує взаємодії між ознаками). - Може присвоювати нульові ймовірності комбінаціям «ознаки-клас», які не траплялися в навчальних даних (проблема вирішується згладжуванням). - Для безперервних ознак вимагає припущення про розподіл (наприклад, нормальність для Gaussian Naive Bayes)	- Класифікація текстів: спам-фільтри, аналіз тональності відгуків, тематична категоризація документів. - Медичні діагностичні системи (оцінка ймовірності хвороби за симптомами). - Системи рекомендацій та рейтингові оцінки, де важлива швидкість обчислень
Дерева рішень	Структурована модель у вигляді дерева, де кожне внутрішнє рішення перевіряє значення певної ознаки, розгалужуючи дані на підмножини, а листові вузли містять підсумковий прогноз (клас або числове значення). Модель будується жадібно, рекурсивно розбиваючи дані за критерієм найкращого поділу (наприклад, критерій Джіні або інформаційний приріст)	- Зрозумілі та легкі для візуалізації моделі: дерево можна представити як набір правил, зрозумілих людині. - Універсальні: застосовуються і для класифікації, і для регресії. - Не потребують попереднього масштабування ознак чи складної підготовки даних (інваріантні до монотонних перетворень ознак). - Можуть моделювати нелінійні залежності та враховувати взаємодії між ознаками	- Схильні до перенавчання без спеціальних заходів (наприклад, обрізки дерева або встановлення мінімального числа зразків на лист). - Нестабільні: невеликі зміни в даних можуть суттєво змінити структуру дерева (висока дисперсія оцінок). - Можуть бути упередженими щодо ознак з великою кількістю унікальних значень (такі ознаки частіше обираються для розгалужень). - За наявності глибокого дерева інтерпретація стає складнішою	- Прийняття рішень у бізнесі та фінансах: напр. рішення про видачу кредиту на основі характеристик клієнта (доходи, кредитна історія тощо). - Медичні експертні системи: діагностика захворювань за результатами аналізів (дерево рішень визначає діагноз). - Як підмодулі в ансамблевих методах (випадковий ліс, бустинг) для покращення точності моделей

1	2	3	4	5
Випадковий ліс	Ансамблевий метод бегінгу, що комбінує безліч дерев рішень. Кожне дерево тренується на випадковій вибірці даних та підмножині ознак, а фінальний прогноз отримується шляхом голосування (для класифікації) або усереднення (для регресії) результатів всіх дерев	- Зменшує перенавчання порівняно з одиничним деревом шляхом усереднення помилок різних моделей. - Добре працює на великих наборах даних і з великою кількістю ознак (вбудована випадкова вибірка ознак зменшує вплив «вибуху розмірності»). - Часто забезпечує вищу точність прогнозу, ніж окреме дерево, завдяки комбінуванню багатьох моделей. - Стійкий до шуму та викидів: окремі помилкові розгалуження згладжуються при голосуванні	- Втрачається інтерпретова-ність: складно пояснити підсумкове рішення, адже воно отримане з сотень дерев (можна оцінити значущість ознак, але не простий правило). - Більше ресурсів та часу для навчання: тренування багатьох дерев є обчислювально складнішим, ніж однієї моделі. - Повільніше отримання прогнозу для нових даних (потрібно пропустити приклад через усі дерева)	- Задачі класифікації та регресії, де важлива висока точність і надійність моделі більше, ніж її прозорість: кредитний скоринг, прогноз цін на товари чи нерухомість, визначення шахрайських транзакциз. - Аналіз даних з великою кількістю ознак або об’єктів, де одиночне дерево перенавчується або дає нестабільні результати
Градієнтний бустинг	Ансамблевий метод, що належить до бустингу: будує серію простих моделей послідовно. Кожна нова модель навчається на помилках попередніх, намагаючись зменшити градієнт (похибку) поточної ансамблевої моделі. Як слабкі моделі часто використовують дрібні дерева; підсумкова модель є їх комбінованим прогнозом	- Досить висока точність прогнозування завдяки послідовному виправленню похибок (нерідко перевершує інші алгоритми на табличних даних). - Може виявляти складні нелінійні взаємозв’язки між ознаками та цільовою змінною (враховує взаємодії ознак через деревоподібні моделі). - Порівняно стійкий до мультиколінеарності: якщо ознаки корельовані, модель їх відповідним чином зважить через внесок у помилку	- Чутливий до викидів: аномальні дані можуть сильно вплинути на послідовність навчання і спричинити перенавчання. - Висока обчислювальна складність: навчання багатьох послідовних моделей потребує багато часу і ресурсів, особливо на великих даних. - Багато гіперпараметрів для налаштування (кількість дерев, глибина, швидкість навчання тощо), що ускладнює вибір оптимальної моделі	- Задачі, де критично отримати найкращу точність: виявлення шахрайства, прогнозування доходів, моделі оцінки ризиків у фінансах, рейтингові задачі. - Змагання з аналізу даних (Kaggle тощо), де градієнтний бустинг і його реалізації (XGBoost, LightGBM, CatBoost) часто дають найкращі результати

1	2	3	4	5
Метод головних компонент (PCA)	Статистичний метод зменшення розмірності, який трансформує вихідний набір ознак у менший набір нових змінних – головних компонент. Кожна компонента є ортогональною (некорельованою) лінійною комбінацією початкових ознак і пояснює максимально можливу варіацію (дисперсію) у даних. Відбираючи перші кілька компонент, що пояснюють основну частку дисперсії, можна значно скоротити кількість вимірів.	- Усуває мультиколінеар-ність: замінює корельовані ознаки новими незалежними компонентами. - Зменшує шум у даних, відкидаючи компоненти з низькою дисперсією (які вважаються переважно шумовими). - Стискає дані: менше вимірів означає менші вимоги до пам’яті і швидше навчання моделей, зменшує ризик перенавчання на багатовимірних даних	- Втрата інтерпретованос-ті ознак: головні компоненти є комбінаціями початкових змінних і часто не мають зрозумілого фізичного змісту. Можлива втрата частини інформації: якщо відкинути компоненти, що сумарно мають помітну варіацію, модель може втратити важливі фактори. - Припускає лінійність у даних: ефективний, коли основні структури в даних є лінійними; може погано працювати, якщо справжні залежності нелінійні (для цього потрібні нелінійні методи на кшталт kernel PCA або t-SNE)	- Візуалізація даних: скорочення багатовимірних даних до 2–3 головних компонент для побудови графіків і виявлення кластерів чи аномалій. - Попередня обробка даних: зменшення розмірності перед навчанням моделей (наприклад, в задачах розпізнавання облич або рукописного тексту) для пришвидшення роботи алгоритмів і зниження ризику перенавчання. Стиск сигналів та зображень: зменшення розміру зображень, звуку тощо, залишаючи основні компоненти (відомий приклад – метод Eigenfaces для розпізнавання облич)

Додаток Б

Методи класифікації

Група	Тип навчання	Ключові задачі
Асоціативні правила	*Непідконтрольне* / rule-learning	Виявлення закономірних спільних появ ознак (market-basket, web-log)
Кластерний аналіз	Непідконтрольне	Групування подібних об’єктів без заздалегідь заданих міток
Дискримінантний аналіз	*Підконтрольне* (генеративна класифікація)	Розділення класів через моделювання їх розподілів
Часові ряди (класичні моделі)	Підконтрольне (прогноз)	Прогнозування послідовностей із часовою структурою

Метод / сімейство	Принцип	Переваги	Недоліки	Типові приклади
1	2	3	4	5
Apriori / FP-Growth (асоціативні правила)	Ітеративно знаходять часті набори ознак (itemsets) і виводять правила виду «якщо A, то B» з підтримкою *support* та довірою *confidence*	Інтуїтивні правила, легко пояснювати бізнесу. Працюють напряму з категоріальними даними (чеки, логи)	«Вибух» комбінацій → висока складність на великих наборах. Потрібен грамотний вибір порогів support / confidence; інакше – шум або занадто мало правил	*Market basket*-аналіз (розкладка товарів), перехресні продажі, рекомендації, виявлення шаблонів у мережевих журналах
K-means, ієрархічний, DBSCAN (кластеризація)	Максимізують внутрішню схожість та мінімізують міжкластерну (K-means – середні; ієрархічний – дерево злиття; DBSCAN – щільні області)	Не потребує міток (label-free). Виявляє природну структуру даних; корисно для початкового дослідження	Чутливість до масштабу й вибору метрики. Потрібно задавати *k* (K-means) або ε / MinPts (DBSCAN)	Сегментація клієнтів / зображень, виділення тем у текстах (разом із LDA), групування генів / білків, anomaly detection (кластер-outliers)

1	2	3	4	5
Лінійний / квадратичний дискримінантний аналіз (LDA / QDA)	Моделюють класові розподіли як (мульт)нормальні ; обчислюють межу, що мінімізує ймовірність помилки: лінійну (LDA, спільна коваріація) або квадратичну (QDA, окремі коваріації)	Оптимальні за нормальності даних; швидкі й аналітичні рішення. Самі дають можливість зменшення розмірності (LDA-проекції)	Припускають гаусові розподіли; нестійкі до відхилень і викидів. Погано працюють, якщо класи сильно перекриваються-ся або коваріаційні матриці вироджені	Розпізнавання образів (faces, digits), біоінформатика, скоринг документів (topic vs. background), медична діагностика
ARIMA, SARIMA, Exponential Smoothing, Holt-Winters (класичні часові ряди)	Будують лінійну модель автокореляцій (AR), інтервального інтегрування (I), ковзного середнього (MA); розширення – сезонність (S) та комбіноване згладжування тренду / сезону	Добре теоретично обґрунтовані (стаціонарність, ACF/PACF). Порівняно інтерпретовані (тренд, сезонність, шум)	Лінійність і стаціонарність – обмеження на багато реальних процесів. Важке ручне підбирання (p, d, q) та сезонних параметрів; чутливі до структурних зламів	Прогноз продажів, фінансові та макроекономічні ряди, навантаження на сервери / мережі, попит на енергію

Додаток В

Довідник переднавчених моделей штучних нейронних мереж

1. VGG16

VGG16 – це глибока згорткова нейронна мережа (CNN), розроблена дослідниками з *Visual Geometry Group (VGG)* Оксфордського університету. Модель отримала популярність після участі в змаганні *ILSVRC (ImageNet Large Scale Visual Recognition Challenge)* 2014 року, де вона показала високі результати в класифікації зображень. Нижче дається детальний опис цієї моделі.

2. ResNet (Residual Network)

Рік створення: 2015 (Microsoft Research).

Особливість. Використовує *residual blocks* – прямі з'єднання (skip connections), які дають змогу уникати **затухання градієнта (vanishing gradient)**.

Варіанти: ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-152 (число – кількість шарів).

Переваги

Ефективна для дуже глибоких мереж (100 + шарів).

Висока точність на ImageNet.

Підходить для складних завдань (медична діагностика, автономне керування).

Використання. Класифікація, виявлення об'єктів (YOLO, Faster R-CNN).

3. Inception (GoogleNet)

Рік створення: 2014 (Google Brain)

Особливість. Використовує **Inception blocks** – одночасно застосовує згортки різного розміру (1 × 1, 3 × 3, 5 × 5) для захоплення багатомасштабних ознак.

Варіанти: InceptionV1 (GoogLeNet), InceptionV2, InceptionV3, InceptionV4, Inception-ResNet.

Переваги

Менше параметрів, ніж у VGG16 (≈6.6 млн vs. 138 млн).

Ефективна для мобільних пристроїв та вбудованих систем.

Висока точність на ImageNet.

Використання. Класифікація, розпізнавання сцен, медичні зображення.

4. EfficientNet

Рік створення: 2019 (Google Brain).

Особливість. Використовує **компаунд-скейлінг** – збільшує глибину, ширину та роздільну здатність одночасно.

Варіанти: EfficientNet-B0 до EfficientNet-B7.

Переваги

Висока ефективність – кращі результати щодо меншої кількості параметрів.

Легко адаптується для різних завдань (класифікація, сегментація, детекція).

Відмінний вибір для розгортання в продакшн.

Використання. Мобільні пристрої, розпізнавання облич, медичне зображення.

5. DenseNet (Densely Connected Network)

Рік створення: 2017 (Cornell University).

Особливість. Використовує щільні зв'язки (dense connections) – кожен шар отримує вхід від усіх попередніх шарів.

Варіанти: DenseNet-121, DenseNet-169, DenseNet-201.

Переваги

Менше параметрів при глибокій архітектурі.

Краще збереження градієнтів під час навчання.

Підвищена узагальнююча здатність.

Використання. Класифікація медичних зображень, розпізнавання тексту.

6. Vision Transformer (ViT)

Рік створення: 2020 (Google Research).

Особливість. Використовує **трансформери** (аналогічно до BERT та GPT) для обробки зображень, представляючи зображення як патчі.

Переваги

Вища точність на великих наборах даних.

Легше масштабувати до надвеликих моделей.

Кращі результати в порівнянні з CNN у багатьох задачах.

Використання. Класифікація, сегментація, генерація зображень.

7. MobileNet

Рік створення: 2017 (Google).

Особливість. Використовує **глибинні згортки (depthwise separable convolutions)**, що зменшують обчислювальну складність.

Варіанти: MobileNetV1, MobileNetV2, MobileNetV3.

Переваги

Легка та швидка – підходить для мобільних пристроїв та IoT.

Менша точність, але значно швидше.

Використовується для задач реального часу.

Використання. Розпізнавання облич, детекція об'єктів (на телефонах).

8. Swin Transformer

Рік створення: 2021 (Microsoft Research Asia).

Особливість. Гібридна модель – поєднує переваги **трансформерів** із локальними вікнами (Sliding Window).

Переваги

Відмінна продуктивність у класифікації та сегментації.

Масштабованість для високоякісних зображень.

Використання. Класифікація, сегментація (наприклад, у медичній візуалізації).

Опис набору ImageNet для комп'ютерного зору

Цікаві факти про ImageNet

Для створення датасету було використано *Amazon Mechanical Turk***,** де люди вручну маркували зображення. ImageNet став *золотим стандартом* для перевірки якості моделей комп'ютерного зору. Модель *AlexNet (2012)* навчена на наборі скоротила помилку класифікації на 10 %, що дало старт буму глибокого навчання.

ImageNet – це великий набір даних для навчання та оцінювання моделей глибокого навчання, що використовується переважно для завдань *класифікації зображень* та *обробки комп'ютерного зору***.**

Основні характеристики ImageNet

Характеристика	Значення
Рік створення	2009
Автори	Стенфордський університет (Fei-Fei Li)
Кількість зображень	≈ 14 мільйонів
Кількість класів	21 841 класів (WordNet категорії)
Роздільна здатність	Різна (від 256×256 пікселів і вище)
Мова анотацій	Англійська
Ліцензія	Не вільна (обмежене використання)

Структура ImageNet

ImageNet базується на **ієрархії WordNet** – структурі, яка групує слова за змістом у категорії (синсети). Кожен клас у ImageNet відповідає певному **синсету** – групі схожих понять.

Класи (synnsets): 21 841 клас (у змаганні ILSVRC – 1 000 класів).

Типи зображень:

Тварини (собаки, коти, птахи тощо).

Транспорт (автомобілі, літаки, поїзди).

Побутові предмети (стільці, чашки, телевізори).

Люди (різні пози, діяльності).

Природні об'єкти (гори, річки, рослини).

ImageNet-1k. Підмножина з *1000 класів і 1.2 млн зображень*, яку використовують у більшості досліджень.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

З 2010 до 2017 року проводилось щорічне змагання з розпізнавання зображень на основі ImageNet.

Завдання в змаганні

Класифікація: Визначити, до якого класу належить зображення.

Локалізація: Визначити координати об’єкта на зображенні.

Детекція: Знайти кілька об'єктів одночасно.

Ключові досягнення ILSVRC:

2012: *AlexNet* (перший прорив глибокого навчання).

2015: *ResNet* (зменшення помилки до рівня людини).

2017: Останній рік офіційного проведення.

Використання ImageNet у сучасних моделях

Навчання з перенесенням (Transfer Learning). Використання попередньо навчених моделей на ImageNet для інших завдань (медична діагностика, автономні автомобілі).

Бенчмарк (Benchmark). Вимірювання продуктивності нових архітектур (ResNet, EfficientNet, Vision Transformers).

Популярні моделі, навчені на ImageNet. VGG16 / VGG19 (2014); ResNet (2015); Inception (2015); EfficientNet (2019) ;Vision Transformer (ViT) (2020)

Додаток Д

Опис та архітектура моделі VGG16

Основні характеристики VGG16

1. *Кількість шарів:* 16 (13 згорткових шари + 3 повнозв'язні шари).
2. *Вхідний розмір зображення:* 224 × 224 × 3 (RGB-канали).
3. *Кількість параметрів:* ≈138 мільйонів.
4. *Функція активації:* ReLU (Rectified Linear Unit).
5. *Призначення:* класифікація зображень, виявлення об'єктів, витягування ознак.

Архітектура VGG16

1. Згорткові шари (Convolutional Layers)

Використовуються 3 × 3 фільтри (ядра) з кроком 1.

Паддинг (padding="same") зберігає розмір зображення.

2. Шари підвибірки (Max Pooling)

Після кожного блоку згортувань додається шар 2 × 2 MaxPooling з кроком 2 для зменшення розміру карти ознак.

3. Повнозв'язні шари (Fully Connected Layers)

Три щільних (Dense) шари: перші два з 4 096 нейронами, останній – вихідний із 1 000 класами (у випадку ImageNet).

4. Функція Softmax

Для розрахунку ймовірностей приналежності до певного класу.

Структура моделі VGG16:

Блок	Тип шару	Розмір виходу
Вхід	RGB-зображення (224 × 224 × 3)	(224, 224, 3)
Блок 1	2 × Conv (64 фільтри)	(224, 224, 64)
	MaxPooling (2 × 2)	(112, 112, 64)
Блок 2	2 × Conv (128 фільтрів)	(112, 112, 128)
	MaxPooling (2 × 2)	(56, 56, 128)
Блок 3	3 × Conv (256 фільтрів)	(56, 56, 256)
	MaxPooling (2 × 2)	(28, 28, 256)
Блок 4	3 × Conv (512 фільтрів)	(28, 28, 512)
	MaxPooling (2 × 2)	(14, 14, 512)
Блок 5	3 × Conv (512 фільтрів)	(14, 14, 512)
	MaxPooling (2 × 2)	(7, 7, 512)
FC	2 × Dense (4 096 нейронів)	(4 096)
Вихід	Dense + Softmax (1 000)	(1 000)

Переваги VGG16

Простота архітектури: використовує лише 3 × 3 згортки та 2 × 2 підвибірку.

Універсальність: використовується для класифікації, сегментації та детекції.

Переносне навчання: модель доступна у вигляді попередньо навченої на ImageNet ваговій матриці.

Недоліки VGG16

Великий розмір. 138 мільйонів параметрів, що вимагає значних обчислювальних ресурсів.

Повільність. Більший час навчання порівняно з легшими моделями (MobileNet, ResNet).

Високі вимоги до пам'яті. Підходить для роботи на потужних GPU.

Застосування VGG16

Класифікація зображень (ImageNet, CIFAR-10, MNIST).

Витягнення ознак (feature extraction) для інших моделей.

Розпізнавання об'єктів у відео та зображеннях.

Навчальне видання

Редич Олександр Володимирович

Великі дані та машинне навчання

Навчальний посібник

Відповідальний за випуск *Н. Б. Добрянська*

Редактори *М. П. Клименко,*
М. М. Грабарчук

Форматування та
комп’ютерна верстка *Д. П. Завальніцька*

Здано до друку 22.12.2025. Формат 29,7 × 41,99
Папір офсетний № 1. Гарнітура «Times New Roman»
Ум. друк. арк. 13.5
Наклад 300 прим. Замовлення № 1151

Підготовлено до друку науково-організаційним відділом
Державного податкового університету
08205, вул. Університетська, 31, м. Ірпінь, Київська область,
Україна

Свідоцтво про внесення суб’єкта видавничої справи
до державного реєстру видавців, виготовлювачів і
розповсюджувачів видавничої продукції
Серія ДК № 7669 від 20.09.2022



Редич Олександр Володимирович

Кандидат економічних наук, доцент кафедри комп'ютерних та інформаційних технологій і систем. Керівник та акредитований інструктор міжнародних освітніх програм, до яких долучив Державний податковий університет у різні роки: Cisco Academy (2007), Oracle Academy (2017), Amazon WS Academy (2024). Автор більше 100 наукових і науково-методичних робіт. Сфера наукових інтересів: комп'ютерні інтелектуальні технології, наука про дані (Data Science), великі дані (Big Data) методи та інструментарій машинного навчання (ML), штучний інтелект (AI).