МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ДНІПРОВСЬКА ПОЛІТЕХНІКА»

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра системного аналізу та управління

К.С. Хабарлак, Т.В. Хом’як

АНАЛІЗ ТА ОБРОБКА ВЕЛИКИХ ДАНИХ

Конспект лекцій
для здобувачів ступеня магістра
освітньо-професійної програми «Системний аналіз»
зі спеціальності 124 Системний аналіз

Дніпро
НТУ «ДП»
2024

Хабарлак К.С.

Аналіз та обробка великих даних [Електронний ресурс] : конспект лекцій для здобувачів ступеня магістра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / К.С. Хабарлак, Т.В. Хом'як ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 111 с.

Автори:

Хабарлак К.С., доктор філософії;

Хом'як Т.В., канд. фіз.-мат. наук, доц.

Затверджено науково-методичною комісією зі спеціальності 124 Системний аналіз (протокол № 3 від 10.05.2024) за поданням кафедри системного аналізу та управління (протокол № 5 від 10.05.2024).

У конспекті лекцій з курсу «Аналіз та обробка великих даних» подано теоретичні основи аналізу великих даних та нереляційних баз даних. Здобувач ознайомиться з архітектурами розподілених систем, бібліотекою Apache Spark та її інтерфейсом до мови програмування Python, поняттям кадру даних, особливостями роботи та операціям з ним, методами розподіленого машинного навчання, створенням та роботою з нереляційними базами даних, зокрема MongoDB.

Відповідальний за випуск завідувач кафедри системного аналізу та управління Т.А. Желдак, канд. техн. наук, доц.

Зміст

Вступ 6
Лекція 1 – Обробка великих даних в Python 8
1.1. Термінологія та концепції розподіленого машинного навчання 8
1.2. Етапи робочого процесу машинного навчання 11
1.3. Різні способи масштабування 13
1.4. Моделі розподілених обчислень 14
1.5. Архітектура розподілених систем 15
1.6. Моделі взаємодії розподілених систем 16
1.7. Зв'язок у розподіленому середовищі 18
1.8. Контрольні питання 19
Лекція 2 – Apache Spark для обробки та аналізу великих даних в Python 20
2.1. Загальна структура Spark 20
2.2. Ключові терміни Apache Spark 21
2.4. Загальна ідея архітектури Spark 22
2.5. Spark DataFrame 24
2.6. Налаштування локального середовища PySpark 25
2.7. Контрольні питання 37
Лекція 3 – Операції над кадрами даних. Інструменти візуалізації 38
3.1. Що таке кадр даних (DataFrame) в Spark? 38
3.2. Побудова Spark DataFrame 39
3.3. Візуалізація кадру даних 41
3.4. Читання даних з файлу 42
3.5. Операції над кадрами даних PySpark 43
3.6. Контрольні питання 46
Лекція 4 – Інструменти машинного навчання в PySpark. Регресія 47
4.1. Проста лінійна регресія 47
4.2. Рівняння регресії 47
4.3. Найменші квадрати 48
4.4. Множинна лінійна регресія 50
4.5. Оцінювання результативності моделі 51
4.6. Регресія у PySpark 52

4.7. Навчання регресії у PySpark 53
4.8. Контрольні питання 56
Лекція 5 – Машинне навчання в PySpark. Класифікація 57
5.1. Що таке класифікація в машинному навчанні? 57
5.2. Ліниві учні та учні, які прагнуть вчитися 58
5.3. Класифікація в машинному навчанні проти регресії 58
5.4. Приклади класифікації машинного навчання в реальному житті 59
5.5. Різні типи задач класифікації в машинному навчанні 59
5.6. Дерева рішень 64
5.7. Випадковий ліс 66
5.8. Контрольні питання 68
Лекція 6 – Нереляційні БД. Переваги та недоліки над реляційними БД 69
6.1. Мова структурованих запитів (SQL) 69
6.2. Не тільки мова структурованих запитів (NoSQL) 71
6.3. Типи баз даних NoSQL 72
6.4. Ключові відмінності між базами даних SQL та NoSQL 73
6.4.1. Масштабованість 73
6.5. Варіанти використання технологій баз даних SQL та NoSQL 74
6.5.1. Варіанти використання баз даних SQL 75
6.5.2. Випадки використання баз даних NoSQL 75
6.6. Контрольні питання 76
Лекція 7 – MongoDB – відкрита документна нереляційна БД 77
7.1. Основні концепції MongoDB 79
7.2. Налаштування MongoDB на локальному комп’ютері 83
7.3. MongoDB Compass – візуальний інструмент роботи з MongoDB 84
7.4. Контрольні питання 87
Лекція 8 – Засоби роботи з MongoDB в Python 88
8.1. Створення баз даних та колекцій 88
8.2. Додавання даних 90
8.3. Операції пошуку даних 92
8.4. Запити до даних 93
8.5. Сортування результатів 95
8.6. Видалення даних 96

8.7. Контрольні питання 98
Лекція 9 – Ланка між PySpark та MongoDB 99
9.1. Витягнути, перетворити, завантажити, або ETL 100
9.2. Об’єднання та уніфікація даних з багатьох джерел засобами PySpark 102
9.3. Контрольні питання 109
Рекомендовані джерела інформації 110

Вступ

Аналіз великих даних є надзвичайно важливою навичкою в сучасному світі, де люди та пристрої Інтернету речей безперервно створюють величезні обсяги інформації. У сирому вигляді ці дані мають обмежену корисність, оскільки їх важко організувати, зберегти та проаналізувати. Саме тому технології обробки та аналізу великих даних швидко розвиваються, пропонуючи нові способи структурування та отримання цінних висновків з цих масивів інформації.

У даному конспекті лекції з курсу «Аналіз та обробка великих даних» подано теоретичні основи аналізу великих даних та нереляційних баз даних. У межах курсу будуть розглянуті наступні теми:

1. Обробка великих даних в Python.
2. Apache Spark для обробки та аналізу великих даних в Python.
3. Операції над кадрами даних. Інструменти візуалізації.
4. Інструменти машинного навчання в PySpark. Регресія.
5. Машинне навчання в PySpark. Класифікація.
6. Нереляційні БД. Переваги та недоліки над реляційними БД.
7. MongoDB – відкрита документна нереляційна БД.
8. Засоби роботи з MongoDB в Python.
9. Ланка між PySpark та MongoDB.

Завдяки цим завданням здобувач ознайомиться з інструментарієм обробки та аналізу великих даних: бібліотеками Apache Spark і PySpark, нереляційною базою даних MongoDB, мовою програмування Python для аналізу великих даних. Здобувач навчиться збирати, зберігати, оброблювати та аналізувати великі масиви даних; будувати моделі регресії та класифікації, використовуючи великі набори даних, та робити передбачення на нових, невідомих вхідних значеннях; опанує нереляційні бази даних для зберігання та обробки великих даних.

Мета дисципліни – сформувати у здобувачів вищої освіти навички збору, обробки та аналізу великих із використанням сучасних бібліотек Apache Spark і MongoDB та мови програмування Python. Знання та навички, отримані в курсі, будуть корисними для подальшого працевлаштування здобувача.

Завдання курсу:

- навчитися збирати, зберігати, оброблювати та аналізувати великі масиви даних;
- навчитися будувати моделі регресії та класифікації, використовуючи великі набори даних, та робити передбачення на нових, невідомих вхідних значеннях;
- опанувати нереляційні бази даних для зберігання та обробки великих даних;

– отримати практичні навички роботи з бібліотеками Apache Spark, PySpark, базою даних MongoDB, мовою програмування Python для обробки та аналізу великих даних.

Дисциплінарні результати навчання:

1. Здійснювати пошук, агрегацію та візуалізацію великих даних для аналізу поведінки досліджуваної системи.
2. Застосовувати методи побудови регресії на великих даних. Проводити аналіз та прогнозування.
3. Застосовувати методи класифікації (лінійних, метричних моделей та на основі дерев рішень) із використанням інструментарію обробки великих даних мови програмування Python.
4. Розуміти способи підгонки параметрів моделей машинного навчання на великих даних.
5. Проводити первинну обробку великих масивів даних різної природи, виявляти закономірності та візуалізувати дані.
6. Розуміти підходи щодо обробки та аналізу слабко структурованих даних для зберігання в нереляційних базах даних, вміти проводити їх подальший аналіз.

Лекція 1 – Обробка великих даних в Python

Напевно, не потрібно говорити, що машинне навчання стало однією з найцікавіших технологій нашого часу та епохи. Великі компанії, такі як Google, Facebook, Apple, Amazon, IBM та багато інших, інвестують значні кошти в дослідження та застосування машинного навчання з поважних причин. Хоча може здатися, що машинне навчання стало модним словом нашого часу, це зовсім не ажіотаж. Ця захоплююча сфера відкриває шлях до нових можливостей і стала незамінною в нашому повсякденному житті. Розмова з голосовим помічником на наших смартфонах, рекомендація правильного продукту для наших клієнтів, припинення шахрайства з кредитними картками, фільтрація спаму з наших електронних поштових скриньок, виявлення та діагностика медичних захворювань – цей список можна продовжувати і продовжувати.

Якщо ви хочете стати фахівцем з машинного навчання, краще вирішувати проблеми або, можливо, навіть подумати про кар'єру в галузі досліджень машинного навчання, тоді цей курс для вас! Однак для новачка теоретичні концепції, що лежать в основі машинного навчання, можуть бути досить складними. Проте, багато практичних книг, які були опубліковані в останні роки, допоможуть вам почати займатися машинним навчанням, впроваджуючи потужні алгоритми навчання. На мою думку, використання прикладів коду слугує важливій меті. Вони ілюструють концепції, застосовуючи вивчений матеріал безпосередньо в дії. В даному конспекті лекцій ми розглянемо не лише необхідні деталі концепцій машинного навчання, пояснення того, як працюють алгоритми машинного навчання, але і як їх використовувати і, що найважливіше, як уникнути найпоширеніших підводних каменів.

Перш ніж зануримося в сферу машинного навчання, дозвольте мені відповісти на ваше найважливіше питання: «Чому саме Python?». Відповідь проста: вона потужна і водночас дуже доступна. Python стала найпопулярнішою мовою програмування для науки про дані, тому що вона дозволяє забути про нудні частини програмування і пропонує нам середовище, де ми можемо швидко записувати свої ідеї та втілювати концепції безпосередньо в життя.

1.1. Термінологія та концепції розподіленого машинного навчання

Пам'ятаєте, як науковці з даних запускали свої алгоритми машинного навчання на наборах даних, що вміщалися в пам'яті ноутбука? Або збирали власні дані? Це не було пов'язано з нестачею даних у світі; ми вже вступили в епоху зеттабайтів. Для багатьох задач дані були, але вони були заблоковані у виробничих системах, які створювали, фіксували, копіювали та обробляли дані у величезних масштабах. Дослідники даних знали, що отримання доступу до них дозволить їм створювати кращі та глибші моделі машинного навчання. Але це була не єдина проблема – як щодо обчислень? У багатьох випадках дослідники даних не мали доступу до достатніх обчислювальних потужностей або

інструментів для підтримки запуску алгоритмів машинного навчання на великих наборах даних. Через це їм доводилося робити вибірки даних і працювати з CSV або текстовими файлами.

Коли у 2016-2017 роках відбулася революція публічних хмар, ми нарешті змогли отримати бажану обчислювальну потужність. Все, що нам було потрібно – кредитна картка в одній руці та комп'ютерна миша в іншій. Одне натискання кнопки – і бум, сотні машин стали доступними для нас! Але нам все ще бракувало належних інструментів з відкритим вихідним кодом для обробки величезних обсягів даних. Існувала потреба в розподілених обчисленнях та автоматизованих інструментах зі здоровою екосистемою.

Зростання діджиталізації, коли компанії використовують цифрові технології для зміни своєї бізнес-моделі та створення нових потоків доходів і можливостей для створення цінності, посилило розчарування науковців з даних. Діджиталізація призвела до збільшення обсягів даних, але аналітики даних не могли працювати з ними достатньо швидко, оскільки не мали необхідних інструментів. Виснажливий процес багатоденного очікування, щоб випробувати один алгоритм машинного навчання або отримати вибірку виробничих даних, заважав багатьом розкрити весь свій потенціал. Потреба в удосконаленні та автоматизації зростала.

Малі компанії бачили, як більші демонстрували позитивний вплив на їхній бізнес, надаючи автоматизовані, персоналізовані рішення для своїх клієнтів, покращуючи їхні настрої та збільшуючи прибутки. З фантастики машинне навчання перетворилося на гарячий товар. Компанії зрозуміли, що для того, щоб скористатися його перевагами, їм знадобиться більше інструментів, а також спеціалізовані команди для їх розробки, що, в свою чергу, збільшило попит на інженерів, які створюють надійні, масштабовані, прискорені та високопродуктивні інструменти для підтримки робочих навантажень, пов'язаних з машинним навчанням.

Netflix, провідна світова мережа інтернет-телебачення, яка щодня транслює сотні мільйонів годин контенту, заявила, що використовує машинне навчання в усіх аспектах бізнесу. Це включає рекомендації персоналізованого контенту для клієнтів, оптимізацію процесів виробництва фільмів і шоу в студіях Netflix, оптимізацію кодування відео і аудіо, а також поліпшення рекламних витрат і рекламного креативу для охоплення нових потенційних клієнтів.

Машинне навчання знайшло застосування в широкому спектрі галузей, але не лише в технологічно орієнтованих компаніях. Фахівці з аналізу даних та аналітики багатонаціональної нафтогазової компанії Shell використовують машинне навчання на великих масивах даних, щоб допомогти бізнесу отримати уявлення про можливості продукту та оптимізацію процесів, а також перевірити ефективність різних способів дій. Одним із прикладів є їхня модель прогнозування запасів, яка виконує понад 10 000 симуляцій на всіх ділянках і об'єктах, прогнозуючи попит і покращуючи запаси. Shell також використовує

машинне навчання для створення рекомендаційної системи для своєї програми лояльності Go+, яка пропонує персоналізовані пропозиції та винагороди окремим клієнтам. Такий підхід забезпечує Shell вдосконалену модель взаємодії, яка допомагає утримувати клієнтів, задовольняючи їхні конкретні потреби.

Інші галузі використовують машинне навчання для виявлення шахрайства, рекомендаційних систем, діагностики пацієнтів тощо. Погляньте на рис. 1.1, щоб отримати уявлення про те, як ви можете стимулювати інновації за допомогою машинного навчання у вашій галузі. Як показують ці приклади, можливість використовувати великі масиви даних для створення рішень з доведеним впливом на бізнес відкрила очі багатьом компаніям, які прагнуть розвивати свій бізнес і збільшувати прибутки.

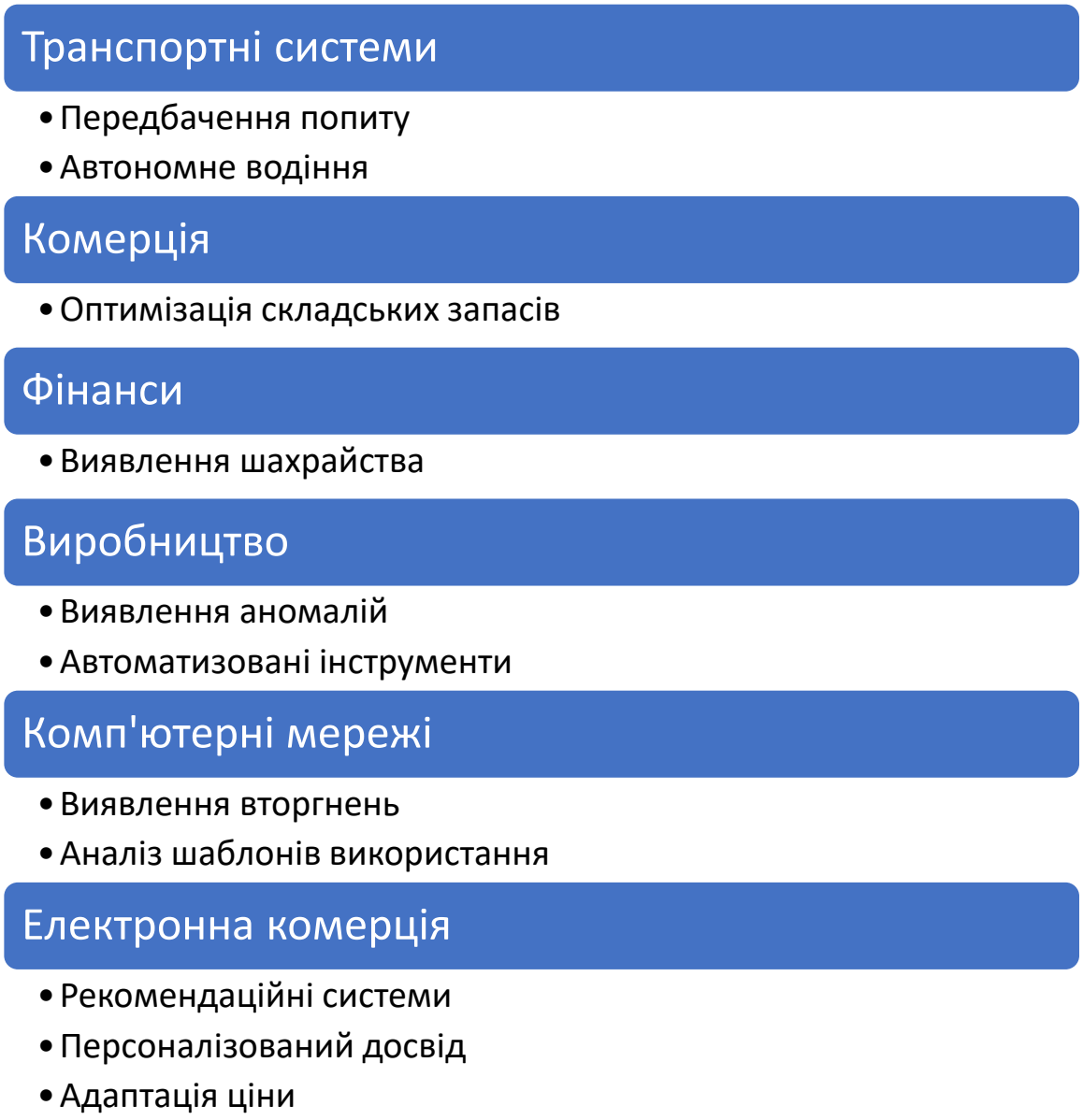

Рис. 1.1. Сфери застосування розподіленого машинного навчання.

Дослідницькі спільноти в галузі комп'ютерних наук та інженерії суттєво допомогли у створенні масштабованого машинного навчання. За останні роки науковці провели сотні, якщо не тисячі досліджень з використання машинного навчання, розподілених обчислень і баз даних, а також побудови більш розумних і ефективних алгоритмів для підтримки розподіленого машинного навчання. В результаті з'явилися розподілені платформи загального призначення, такі як надзвичайно популярна Apache Spark. Apache Spark – це масштабований універсальний движок для аналітики та машинного навчання. У той же час, команди розробників різних бібліотек машинного навчання, створених для підтримки робочих навантажень на одній машині, невтомно додають внутрішню підтримку для виконання в розподіленому середовищі.

1.2. Етапи робочого процесу машинного навчання

Сьогодні багато додатків керуються машинним навчанням, використовуючи моделі машинного навчання для відповіді на такі питання, як «Як мій додаток може автоматично адаптуватися до потреб клієнта?», «Як я можу автоматизувати цей виснажливий процес, щоб мої співробітники могли робити більше зі своїм часом?», «Як я можу знайти сенс у моїй купі даних, не витрачаючи на це цілий рік?». Однак, як фахівці з обробки даних, ми маємо відповісти лише на одне питання: «Як ми можемо зробити так, щоб процес відповідав на ці питання?».

Коротка відповідь – машинне навчання. Більш повна відповідь – робочий процес машинного навчання.

Робочий процес машинного навчання складається з набору етапів, які допомагають нам досягти мети – мати модель машинного навчання, яка працює у виробництві та вирішує бізнес-задачі. Що таке модель машинного навчання? Гарне запитання! Модель машинного навчання – це результат роботи алгоритму машинного навчання. Відтепер ми будемо називати її просто моделлю. Автоматизація цього робочого процесу називається конвеєром машинного навчання. Щоб підвищити точність моделі, робочий процес є ітеративним. Це дозволяє нам здійснювати повний контроль над моделлю, включаючи автоматизацію, моніторинг і розгортання, а також над її результатами.

Робочі процеси машинного навчання складаються з декількох етапів, деякі з яких можна пропустити, а деякі – повторити:

1. **Збереження та завантаження даних.** Перший етап – зібрати дані, необхідні для процесу, і завантажити їх у середовище, де ви будете проводити експеримент з машинного навчання.
2. **Аналіз та перевірка даних.** Далі проаналізувайте зібрані дані та оцініть їхню якість. Цей етап часто включає статистичне тестування того, наскільки добре навчальні дані відображають реальні події, їх розподіл і

різноманітність у наборі даних. Це також називається дослідницьким аналізом даних (EDA).

3. **Очистка/попередня обробка даних.** Після етапу 2 ви можете дійти висновку, що дані зашумлені. Зашумлений набір даних – це набір даних зі стовпчиками, які взагалі не сприяють навчанню, наприклад, рядки з нульовими або некоректними значеннями. Вони вимагають більшої обчислювальної потужності, але не покращують точність моделі. На цьому етапі фахівці з даних проведуть статистичні тести на даних, щоб перевірити кореляцію між ознаками і проаналізувати, які ознаки дають цінність як є, які потребують більшої попередньої обробки або інжинірингу, а які є надлишковими.
4. **Збагачення ознак.** На попередньому етапі стовпці даних виводяться як ознаки. Це дескриптори даних, які використовуються як вхідні дані для моделі машинного навчання. Ознаки в машинному навчанні часто є зовнішніми по відношенню до вихідних даних, а це означає, що нам потрібно збагатити наявні дані даними з інших джерел. Це вимагає від нас розробки коду для обчислення та створення цих ознак і збагачення ними набору даних перед навчанням моделі. Існує багато способів зробити це, і часто для цього потрібні знання предметної області. Крім того, ознаки вже можуть бути присутніми в іншому наборі даних, і в цьому випадку все, що нам потрібно зробити, це об'єднати два набори даних в один перед навчанням моделі.
5. **Розділення даних на навчальну та перевірочну частини**. Навчальний набір використовується для навчання моделі машинного навчання, а валідаційний (перевірочний) набір – для оцінки роботи моделі на невідомих даних.
6. **Навчання та налаштування моделі**. Подайте навчальні дані алгоритму машинного навчання та налаштуйте параметри для покращення продуктивності. Перевірте результати за допомогою спеціального набору даних для перевірки. Процес валідації відбувається в середовищі розробки, або локально на вашому комп'ютері, або в середовищі розробки/експериментів у хмарі. Результатом цього етапу є модель.
7. **Оцінка моделі за допомогою тестових даних.** Це останній етап тестування перед запуском моделі у виробництво. На цьому етапі ви знову вимірюєте продуктивність моделі на раніше не бачених даних, цього разу тестуючи її в умовах, подібних до виробничих. Після цього етапу ви можете повернутися до етапу 6.
8. **Розгортання моделі.** На цьому етапі аналітики даних разом з інженерами з машинного навчання та виробництва пакують модель і розгортають її на виробництві з усіма вимогами.

9. **Моніторинг моделі**. Під час виробництва модель повинна постійно контролюватися на предмет дрейфу. Дуже важливо постійно оцінювати цінність моделі для бізнесу і знати, коли її потрібно замінити.

Кожен з цих етапів повторюється сам по собі, і може статися так, що, отримавши певний результат, ви захочете повторити весь процес заново. Наприклад, у випадку дрейфу моделі, дані та модель не є репрезентативними для бізнес-проблеми, і вам потрібно буде розпочати процес з самого початку.

Кожен етап унікальний і сильно залежить від даних, системних вимог, ваших знань, використовуваних алгоритмів, наявної інфраструктури та бажаного результату.

Етапи з 3 по 6 часто вважають експериментальною фазою машинного навчання. Вам потрібно буде повторювати ітерації і створювати кілька версій даних і моделі, поки ви не знайдете найкращу версію моделі.

1.3. Різні способи масштабування

Вам, напевно, цікаво, як виконати робочий процес машинного навчання на великому наборі даних. Для цього слід знати два терміни: масштабоване машинне навчання та розподілене машинне навчання. При розподіленому машинному навчанні ми використовуємо багатовузлову систему машинного навчання для виконання робочого процесу. За допомогою масштабованого машинного навчання ми оперативно обслуговуємо модель, щоб вона могла масштабуватися відповідно до вимог додатків, які її споживають.

Існує два практичних, взаємодоповнюючих способи масштабування обчислювальних потужностей (рис. 1.2):

- **Вертикальне масштабування** (також відоме як масштабування вгору) – передбачає додавання спеціального обладнання з більшою обчислювальною потужністю до існуючої машини (машин).
- **Горизонтальне масштабування** (також відоме як масштабування назовні) – передбачає додавання більшої кількості вузлів/машин і створення розподіленої топології вузлів, які спільно обчислюють робочі навантаження.

Хоча масштабоване машинне навчання може бути досягнуто будь-яким методом масштабування, розподілене машинне навчання за допомогою Spark в основі своїй базується на масштабуванні з використанням розподілених топологій системи. Можливо, ви думаєте: «Я можу навчати і робити висновки з більш потужним процесором і масштабувати вертикально». Хоча часто під час навчання ми маємо доступ до потужного процесора, під час роботи з моделлю на виробництві та використання її для виводу, ми часто не маємо його не маємо.

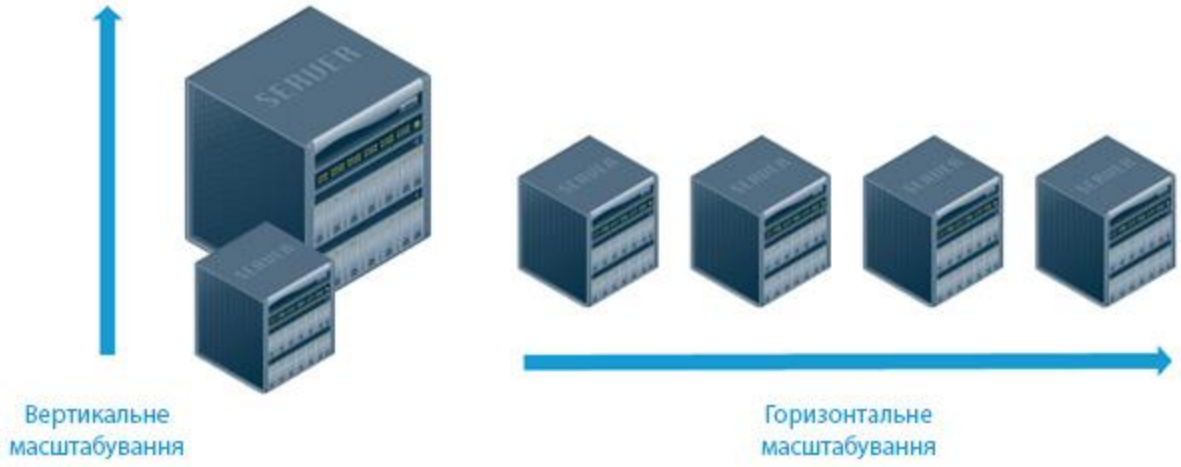

Рис. 1.2. Способи масштабування машинного навчання.

1.4. Моделі розподілених обчислень

Розподілені обчислення – це використання розподілених систем, де кілька машин працюють разом як єдине ціле для вирішення обчислювальної задачі. Програма, яка виконується всередині такої системи, називається розподіленою програмою, а процес написання такої програми – розподіленим програмуванням. Наша мета – знайти найкращий спосіб розбити проблему на окремі задачі, які декілька машин можуть вирішувати паралельно за допомогою обміну повідомленнями. Існують різні моделі розподілених обчислень для машинного навчання, і ми можемо розділити їх на дві групи: моделі загального призначення, які ми можемо налаштувати для підтримки розподілених додатків машинного навчання, і спеціалізовані обчислювальні моделі, спеціально розроблені для запуску алгоритмів машинного навчання.

- **Моделі загального призначення.** Моделі розподілених обчислень загального призначення дозволяють користувачам написати власний потік обробки даних, використовуючи певну абстракцію. Apache Spark – це рушій розподілених обчислень загального призначення, який в своїй основі реалізує модель програмування MapReduce.
- **MapReduce.** Компанія Google представила алгоритм MapReduce у 2004 році в дослідницькій роботі, в якій обговорювалося, як її пошукова система обробляє великі обсяги даних. Як розробники або фахівці з науки про дані, ми визначаємо функцію відображення, яка обробляє пару ключ/значення для створення набору проміжних пар ключ/значення, і функцію редукції, яка об'єднує всі проміжні значення, пов'язані з одним і тим же проміжним ключем. Цей підхід є розширенням стратегії аналізу даних «розбити-застосувати-об'єднати».

На практиці кожне завдання розбивається на декілька функцій відображення та редукції. Дані розбиваються на частини і розподіляються по різних вузлах/машинах, і кожен фрагмент даних обробляється на виділеному вузлі. Багато рішень спрямовані на максимальне збереження локальності даних, коли розбиті дані є локальними для вузла, що їх обробляє.

За необхідності, на виході редуктора можна виконати ще один раунд розділення-застосування-об'єднання. Прикладами рішень з відкритим вихідним кодом, які так чи інакше реалізують ці концепції, є Apache Spark та Hadoop MapReduce.

1.5. Архітектура розподілених систем

Почнемо з короткого обговорення мережевих топологій. Топології – це те, як ми організовуємо комп'ютери для формування розподілених систем. Ми можемо розділити їх на два типи: фізичні топології, які описують, як розташовані і з'єднані комп'ютери, і логічні топології, які описують, як в системі проходять потоки даних і як комп'ютери обмінюються інформацією через мережу. Багатовузлові комп'ютерні топології зазвичай масштабуються фізично шляхом додавання нових комп'ютерів (так званих вузлів).

Інженери часто обговорюють топології у фінальній розмові про архітектуру. Архітектурні вимоги випливають з цілей проекту, даних, поведінки системи та існуючих програмних інструментів, які використовуються. Для науковців про дані головною метою є визначення методів навчання розподілених моделей, а також того, як моделі будуть розгортатися і обслуговуватися.

Вузли, які формують топологію розподіленої системи, з'єднані через мережу за певною архітектурною схемою, розробленою для покращення здатності обробляти навантаження та оптимізації швидкості і використання ресурсів. Вибір архітектури, зроблений на етапі проектування, впливає на роль кожного вузла в топології, спосіб їхньої взаємодії та загальну стійкість системи до збоїв.

Окрім розуміння фізичної топології розподіленої системи, ви повинні знати про відмінності між централізованими та децентралізованими системами, про те, як взаємодіють комп'ютери, які способи зв'язку підтримуються, як система справляється з безпекою та збоями. Ви можете думати про це як про будівельні блоки, з яких ви будете будувати свою систему.

Централізовані та децентралізовані системи. У централізованій системі всі вузли залежать від одного вузла, який приймає рішення. Така система виграє від більшого контролю над рішеннями, але є більш схильною до збоїв, оскільки вузол, що приймає рішення, стає єдиною точкою відмови, яка може вивести з ладу всю систему.

У децентралізованій топології системи вузли є незалежними і приймають власні рішення. Кожен вузол зберігає і оперує власними даними, тому немає

єдиної точки відмови. Це означає, що система більш толерантна до збоїв, але це також означає, що рішення, прийняті окремими вузлами, повинні бути скоординовані та узгоджені.

Децентралізовані системи можуть отримати вигоду від мультихмарної або гібридної хмарної архітектури, де машинні вузли знаходяться в різних регіонах і у різних хмарних провайдерів. Прикладом може слугувати мережа підключених пристроїв Інтернету речей (IoT): кожен пристрій є незалежним, але обмінюється даними через мережу з іншими пристроями та/або хмарою, залежно від його підключення до Інтернету. Топологія, яку ви оберете, впливатиме на методи зв'язку, які можуть використовувати пристрої, та їхні можливі ролі в мережі. Коли йдеться про навчання моделей, децентралізований підхід означає, що кожна модель навчається самостійно.

1.6. Моделі взаємодії розподілених систем

Архітектура моделі взаємодії визначає, як вузли в системі взаємодіють через мережу, які їхні ролі в системі та які обов'язки вони несуть разом з цими ролями. У цьому розділі ми розглянемо три можливі архітектури: клієнт/сервер, однорангову (рис. 1.3) та георозподілену. Існують й інші архітектури, що використовуються та розробляються, які тут не розглядаються, але це ті, з якими ви, найімовірніше, зіткнетеся.

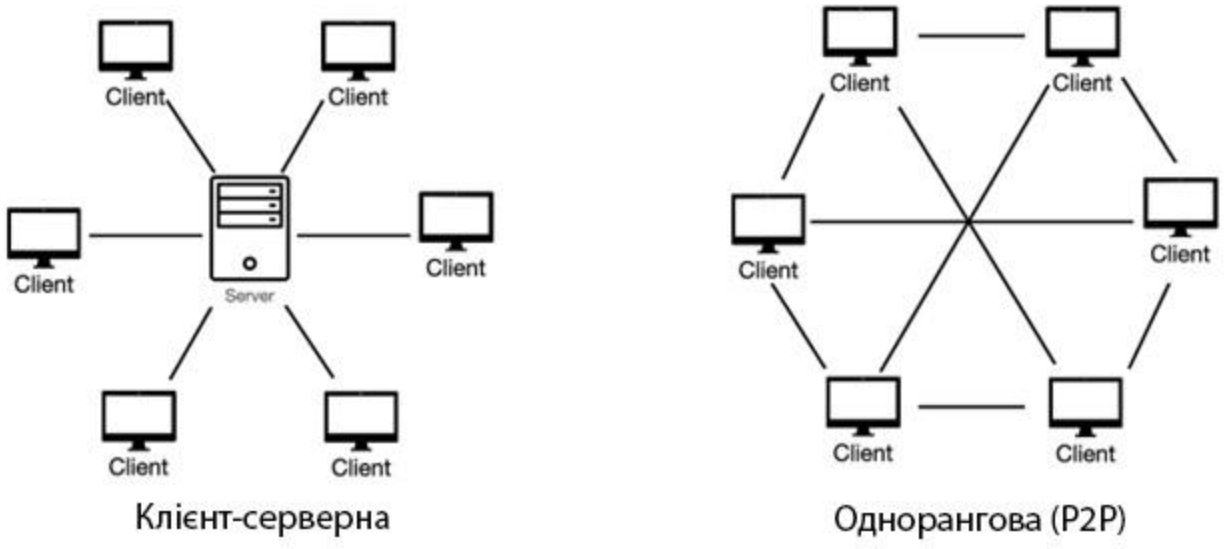

Рис. 1.3. Клієнт-серверні системи та однорангові (P2P).

- **Клієнт-серверна.** У моделі взаємодії клієнт/сервер існує чіткий розподіл обов'язків. Завдання розподіляються між клієнтами, які надсилають запити, і серверами, які надають відповіді на ці запити. Роль вузла може змінюватися залежно від структури та потреб системи.

- **Одноранговa мережа.** У моделі однорангової взаємодії (P2P) робоче навантаження розподіляється між вузлами, або пірами (peer-to-peer). Всі вузли мають рівні привілеї і можуть обмінюватися інформацією безпосередньо, не покладаючись на виділений центральний сервер. Кожен одноранговий вузол може бути як клієнтом, так і сервером. Ця топологія є більш гнучкою і дешевшою у реалізації, оскільки немає необхідності прив'язувати машину до конкретної відповідальності. Однак вона має деякі недоліки: кожен вузол повинен мати повну копію даних, а оскільки всі дані обмінюються через мережу без виділеного координатора, до одного вузла може потрапити кілька копій.
- **Георозподілена.** Георозподілена модель взаємодії (рис. 1.4) найчастіше зустрічається в георозподілених хмарних дата-центрах. Вона спрямована на вирішення проблем, пов'язаних з такими питаннями, як конфіденційність даних і розподіл ресурсів. Одна з проблем полягає в тому, що затримка одноранговоі взаємодії в георозподіленій моделі може бути високою, залежно від відстані між вузлами. Тому при розробці розподілених робочих навантажень машинного навчання на основі цієї моделі взаємодії необхідно чітко визначити, як і в яких випадках вузли взаємодіють між собою. Прикладом, коли георозподілена модель взаємодії є гарним вибором, є забезпечення навчання з IoT пристроями, де дані не можуть бути централізовані в одному центрі обробки даних. Розробка системи для навчання моделі на кожному пристрої в декількох децентралізованих вузлах і збір вихідних даних для створення однієї цілісної моделі дозволяє нам отримати вигоду з даних всіх пристроїв, не обмінюючись приватною інформацією.

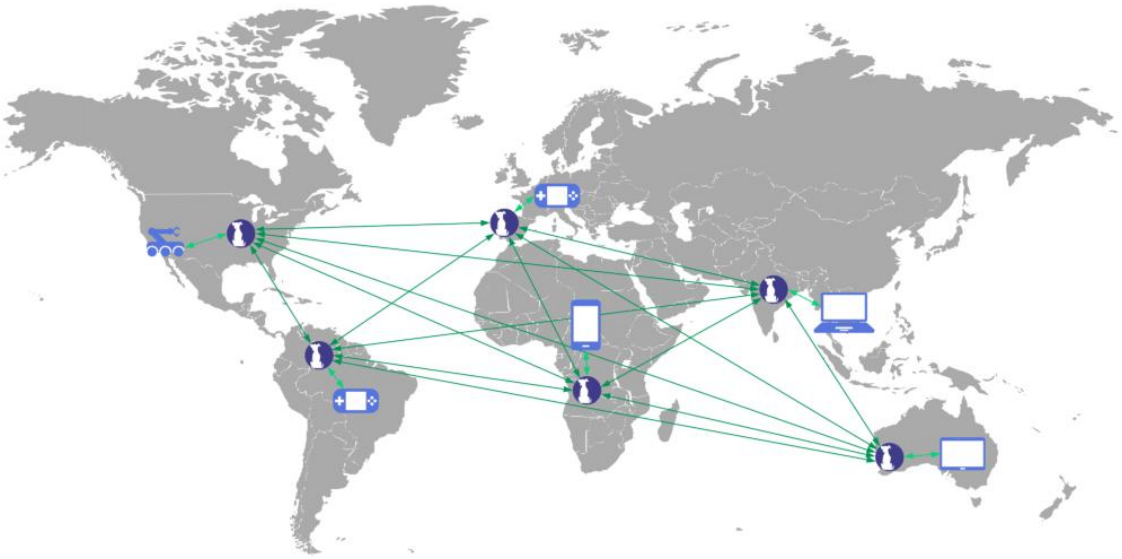

Рис. 1.4. Приклад георозподіленої системи.

1.7. Зв'язок у розподіленому середовищі

Те, як наші вузли взаємодіють у розподіленому середовищі, має значний вплив на механізми відмов, безпеку та пропускну здатність. Зв'язок може бути синхронним або асинхронним, залежно від потреб моделі розподілених обчислень.

Розподіл робочого навантаження машинного навчання між декількома машинами вимагає розділення даних та/або самої програми, щоб рівномірно розподілити навантаження між усіма машинами. Рішення про використання асинхронного або синхронного зв'язку між машинами впливає на час обчислень і може призвести до виникнення вузьких місць.

- **Асинхронний.** Основним механізмом асинхронного зв'язку є черга. Запити до певного вузла ставляться в чергу на виконання і в кінцевому підсумку можуть повернути результат або ні. Цей механізм корисний в системах, де обмін інформацією не залежить від часу, оскільки немає необхідності отримувати відповідь негайно. Можна уявити це як текстове повідомлення: ви надсилаєте повідомлення своєму другові з питанням про плани на вечерю в наступну суботу, знаючи, що, швидше за все, отримаєте відповідь, але вона не потрібна вам негайно. Асинхронна комунікація дозволяє обмінюватися повідомленнями в розподіленій системі, не блокуючи жодних процесів під час очікування відповіді; зазвичай їй надають перевагу, коли це можливо.
- **Синхронний.** Вимога синхронного зв'язку випливає зі стеку функцій комп'ютерних наук, де функції повинні виконуватися в певному порядку – це означає, що якщо вузол надсилає запит до іншого вузла, він не може продовжувати обробляти подальшу логіку функції, чекаючи на відповідь. Ви можете використовувати синхронний зв'язок для конкретних випадків розподіленого машинного навчання і, за необхідності, використовувати виділене обладнання, наприклад, спеціальні мережеві кабелі. Припустимо, ви хочете обговорити з другом плани на вечерю на сьогоднішній вечір. Ваші дії залежатимуть від уподобань вашого друга щодо їжі та наявності вільних місць у обраному вами ресторані. Ви знаєте, що якщо ви виберете популярний ресторан і не забронюєте столик зараз, то там не залишиться вільних місць. Що ви робите? Замість того, щоб надсилати смс, ви берете телефон і телефонуєте своєму другу, щоб отримати інформацію синхронно; тепер ви обидва заблоковані під час телефонної розмови. Ви отримуєте необхідну інформацію і переходите до наступного етапу – контакту з рестораном.

1.8. Контрольні питання

1. Наведіть відмінності між різними способами масштабування. Який найчастіше використовується із Apache Spark?
2. Якими є основні етапи робочого процесу машинного навчання? Які інструменти ви знаєте для вирішення кожного з етапів?
3. В чому полягають переваги та недоліки централізованих та децентралізованих систем обробки?
4. Що таке клієнт-серверна та однорангова системи?
5. Погляньте на рис. 1.1. Проведіть дослідження в яких ще системах використовуються розподілені обчислення, зокрема із використанням Spark?
6. Навіщо потрібна розподілена обробка даних? Невже не можна аналіз даних виконувати на власному ноутбуці?
7. Що таке синхронний та асинхронний зв’язок? Наведіть приклади власних задач та подумайте, який спосіб зв’язку було б доречніше використати?
8. Що таке георозподілена система взаємодії? Коли її використовують?

Лекція 2 – Apache Spark для обробки та аналізу великих даних в Python

Що таке Spark? Spark – це потужний аналітичний механізм для обробки великих даних, спрямований на швидкість, простоту використання та розширюваність для програм великих даних. Сьогодні Spark став де-факто стандартом для структурування та керування додатками великих даних. Це перевірена та широко поширена технологія, яку використовують багато компаній, які щодня обробляють великі дані.

Spark надає API високого рівня в Java, Python, Scala і R. У даному посібнику ми будемо користуватись Python через PySpark, API, який надає модель програмування Spark для Python. Оскільки Python є найдоступнішою мовою програмування для потужного і експресивного API Spark, простота PySpark робить його найкращим вибором для нас. PySpark надає такі дві важливі функції:

- він дозволяє нам писати програми Spark за допомогою API Python;
- він забезпечує оболонку PySpark для інтерактивного аналізу даних у розподіленому середовищі.

Apache Spark має низку функцій, яких не було у його попередників, наприклад здатність до обробки в пам’яті та потокової обробки, але найважливішою особливістю є те, що Spark є єдиною платформою для обробки великих даних.

Це означає, що Spark поставляється з кількома вбудованими бібліотеками та інструментами, які стосуються різних аспектів роботи з великими даними. Він має вбудований механізм роботи з SQL для виконання обробки великих даних, бібліотеку для масштабованого машинного навчання, механізм потокової обробки для потокової аналітики та багато іншого.

Великі компанії мають багато різноманітних потреб у даних, і, як наслідок, інженерам і аналітикам, доводиться об’єднувати та інтегрувати багато інструментів і методів разом, щоб вони могли створювати різні конвеєри даних для задоволення цих потреб. Але такий підхід може створити дуже серйозну проблему залежності, що створить велику перешкоду для підтримки цього робочого процесу. Це одна з головних причин, чому Spark досяг такого успіху – він вже містить майже все, що вам може знадобитися.

2.1. Загальна структура Spark

Якщо у вас є невеликі дані, їх можна проаналізувати за допомогою одного комп’ютера за розумний проміжок часу. Якщо у вас є великі обсяги даних, використання одного комп’ютера для аналізу й обробки даних (і їх зберігання) може бути надто повільним або навіть неможливим. Ось чому ми хочемо використовувати Spark.

Spark має основну бібліотеку та набір вбудованих бібліотек (SQL, GraphX, Streaming, MLlib), як показано на рисунку 2.1. Як можна побачити, за допомогою

DataSource API Spark може взаємодіяти з багатьма джерелами даних, такими як MongoDB, Hadoop, HBase, Amazon S3, Elasticsearch, MySQL та інші.

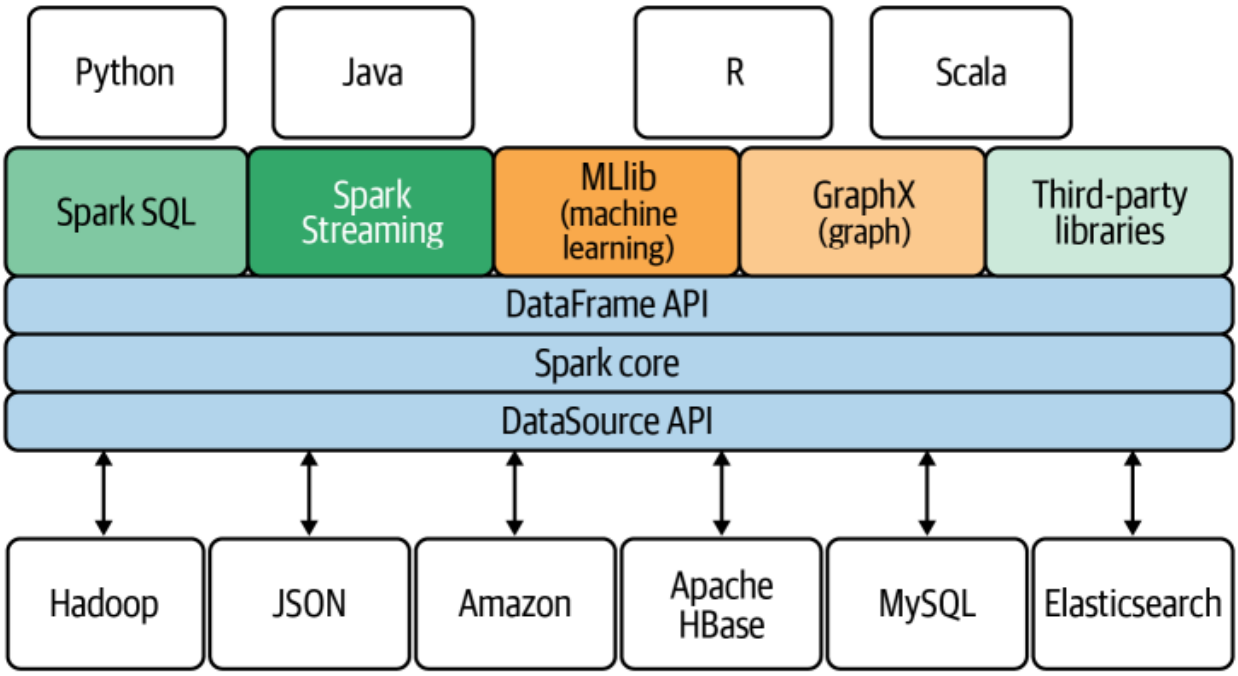

Рисунок 2.1. Бібліотеки Apache Spark.

Цей рисунок показує справжню потужність Spark: можливо використати кілька різних мов для написання програм Spark, а потім застосувати багатий набір бібліотек для вирішення різноманітних проблем з великими даними. При цьому читання та запис можна здійснювати з різних джерел даних.

2.2. Ключові терміни Apache Spark

Щоб зрозуміти архітектуру Spark, необхідно зрозуміти кілька ключових термінів:

- *SparkSession* (сесія Spark). Клас SparkSession, визначений у пакеті pyspark.sql, є точкою входу для програмування Spark за допомогою API наборів даних (Dataset) і кадрів даних (DataFrame). Щоб зробити щось корисне з кластером Spark, вам спочатку потрібно створити екземпляр цього класу, який надає вам доступ до екземпляра SparkContext.
- *SparkContext* (контекст Spark). Клас SparkContext, визначений у пакеті pyspark, є основною точкою входу для функціональності Spark. SparkContext підтримує з'єднання з менеджером кластера Spark і може використовуватися для створення так званих стійких розподілених наборів даних (Resilient Distributed Dataset, RDD) і широкомовних змінних у кластері. Коли ви створюєте екземпляр SparkSession,

SparkContext стає доступним у вашому сеансі як атрибут SparkSession.sparkContext.

- *Driver* (драйвер). Усі програми Spark (включаючи ті, що використовують оболонку PySpark та автономні програми Python) працюють як незалежні набори процесів. Ці процеси координуються контекстом Spark у програмі драйвера. Щоб надіслати автономну програму Python до Spark, необхідно написати програму драйвера із використанням PySpark API (або Java, або Scala). Ця програма відповідає за процес запуску функції main() програми та створення SparkContext. Його також можна використовувати для створення RDD і DataFrame.
- *Worker* (робітник). У кластерному середовищі Spark є два типи вузлів: один (або два, для високої доступності) головні *(master)* і набір робочих. Робочий – будь-який вузол, який може запускати програми в кластері. Якщо для програми запущено процес, ця програма отримує виконавців на робочих вузлах, які відповідають за виконання завдань Spark.
- *Cluster manager* (менеджер кластера). Master вузол також відомий як менеджер кластера. Основною функцією цього вузла є керування середовищем кластера та серверами, які Spark використовуватиме для виконання завдань. Менеджер кластера розподіляє ресурси для кожної програми. Spark підтримує п’ять типів менеджерів кластерів, залежно від того, де він працює:
  1. *Standalone* – автономний – вбудоване кластерне середовище Spark.
  2. *Mesos* – ядро розподілених систем.
  3. *Hadoop YARN*.
  4. *Kubernetes*.
  5. *Amazon EC2*.

2.4. Загальна ідея архітектури Spark

Загальний вигляд архітектури Spark представлено на рис. 2.2. Неформально кластер Spark складається з головного вузла («менеджера кластера»), який відповідає за керування програмами Spark, і набору «робочих» (виконавчих) вузлів, які відповідають за виконання завдань, наданих програми Spark (ваших програми, які ви хочете запускати в кластері Spark).

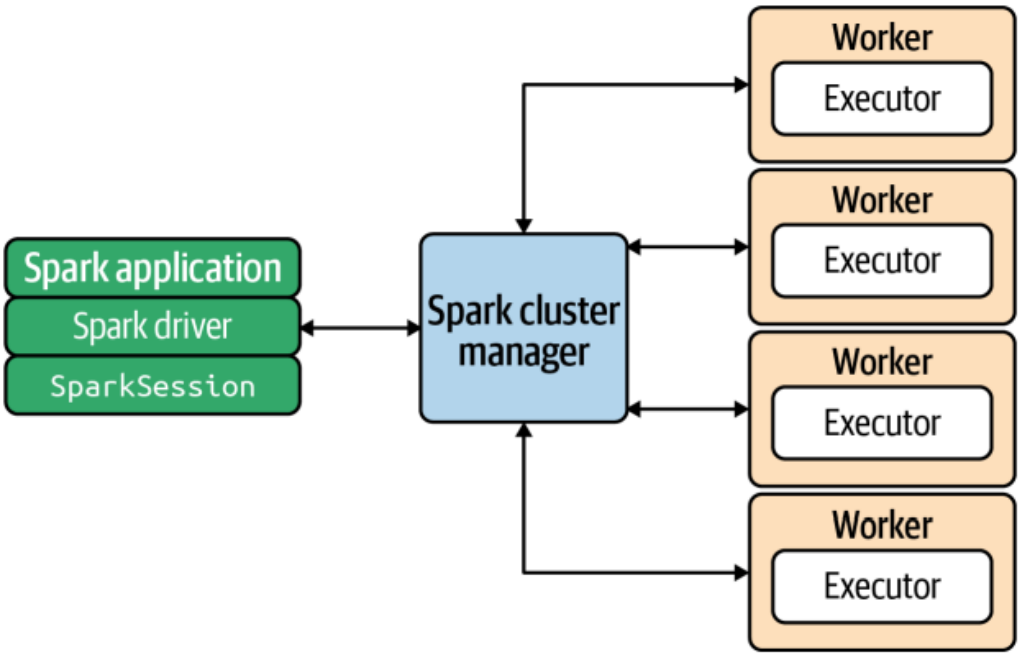

Рис. 1.2. Архітектура Spark.

Залежно від середовища, у якому працює Spark, менеджером кластера, який керуватиме цим кластером серверів, буде автономний менеджер кластера Spark, Kubernetes, Hadoop YARN або Mesos. Коли кластер Spark працює, ви можете надіслати програми Spark менеджеру кластера, який надасть вашій програмі ресурси, щоб ви могли завершити аналіз даних.

Ваш кластер може мати один, десятки, сотні або навіть тисячі робочих вузлів, залежно від потреб вашого бізнесу та вимог вашого проекту. Ви можете запустити Spark на автономному сервері, наприклад MacBook, Linux або Windows, але зазвичай для робочих середовищ Spark запускається на кластері серверів Linux. Щоб запустити програму Spark, вам потрібно мати доступ до кластера Spark і мати програму-драйвер, яка задає необхідні перетворення та дії на RDD даних і надсилає такі запити менеджеру кластера.

Однак, API роботи з даними здебільшого не відрізняється працюєте ви з кластером або з одним вузлом на власному комп'ютері. У цьому курсі всі програми драйверів будуть написані із використанням PySpark та виконуватимуться на одному вузлі.

Які компанії використовують Spark?

- Facebook щодня обробляє 60 ТБ даних. Spark і MapReduce є основою алгоритмів, які використовуються для обробки виробничих даних.
- Viacom зі своїми 170 кабельними, телевізійними та онлайн-мережами приблизно в 160 країнах перетворюється на підприємство, що керується даними, збираючи та аналізуючи петабайти мережевих даних, щоб збільшити лояльність глядачів і дохід.

- Illumina опрацьовує тисячі геномів (це великі дані, які не можуть поміститися на одному сервері або бути обробленими ним) за допомогою Spark, PySpark, MapReduce та розподілених алгоритмів.
- IBM щодня використовує Spark, MapReduce і розподілені алгоритми для масштабування своїх обчислень і операцій.

2.5. Spark DataFrame

Spark має два поняття структурованих даних: DataFrame і Dataset. Spark Dataset – це розподілена колекція даних, натомість, Spark DataFrame – це набір даних Spark, організований у стовпці з іменами.

Це означає, що Spark DataFrame є дуже схожою на таблиці, які ми знаємо в реляційних базах даних або в електронних таблицях (як Excel). Отже, у Spark DataFrame кожен стовпець має назву, і всі стовбці мають однакову кількість рядків. Крім того, усі рядки всередині стовпця повинні зберігати дані одного типу, але кожен стовпець може зберігати різні типи даних.

З іншого боку, набори даних Spark вважаються сукупністю будь-яких типів даних. Таким чином, набір даних також може бути набором неструктурованих даних, як-от файли журналу, JSON і XML тощо. Але для більшості програм більш зручно використовувати DataFrame, а не Dataset, для представлення даних.

Основною відмінністю DataFrame від кадрів даних в інших бібліотеках, наприклад Pandas, є розподіленість. Spark DataFrame базується на Spark Dataset. Ці набори даних є колекціями даних, які розподіляються в кластері. Розглянемо дані, представлені в табл. 2.1.

Табл. 2.1. Приклад кадру даних Apache Spark.

ID	**Name**	**Value**
1	Anne	502
2	Carls	432
3	Stoll	444
4	Percy	963
5	Martha	123
6	Sigrid	621

Якщо ви використовуєте Spark у кластері з 4 вузлами (один вузол драйвера, а інші три вузли робочі). Кожен робочий вузол кластера зберігатиме частину цих даних. Таким чином, ви, як програміст, бачитимете, керуватимете та трансформуватимете цю таблицю так, ніби це єдина та об’єднана таблиця. Але під капотом Spark розділить ці дані та збереже їх у вигляді декількох фрагментів у кластері Spark. На рис. 2.3 показано ілюстрацію зберігання цих даних на кластері Spark.

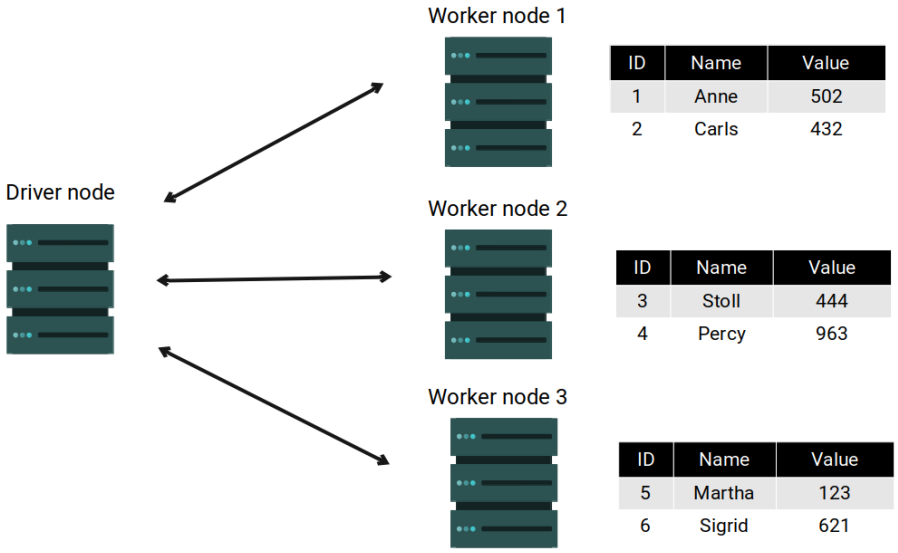

Рис. 2.3. Зберігання єдиного кадру даних на кластері Spark з 4-х вузлів.

Spark DataFrame завжди розбивається на багато дрібних частин, і ці частини завжди розподіляються по кластеру машин. Кожен із цих невеликих фрагментів загальних даних вважається секцією (partition) загального кадру даних.

2.6. Налаштування локального середовища PySpark

Проведемо налаштування одного вузла Apache Spark на локальному комп'ютері. Для цього необхідно встановити декілька компонентів:

- Python 3.11
- Java Development Kit 17
- Apache Spark 3.5.2
- Apache Hadoop
- PySpark

1. Встановимо Python. Зауважимо, що необхідно встановити саме версію 3.11. Дану версію можна завантажити за посиланням. На початку установки необхідно обов'язково проставити галочку «Add python.exe to PATH», як показано на рис. 1.4.

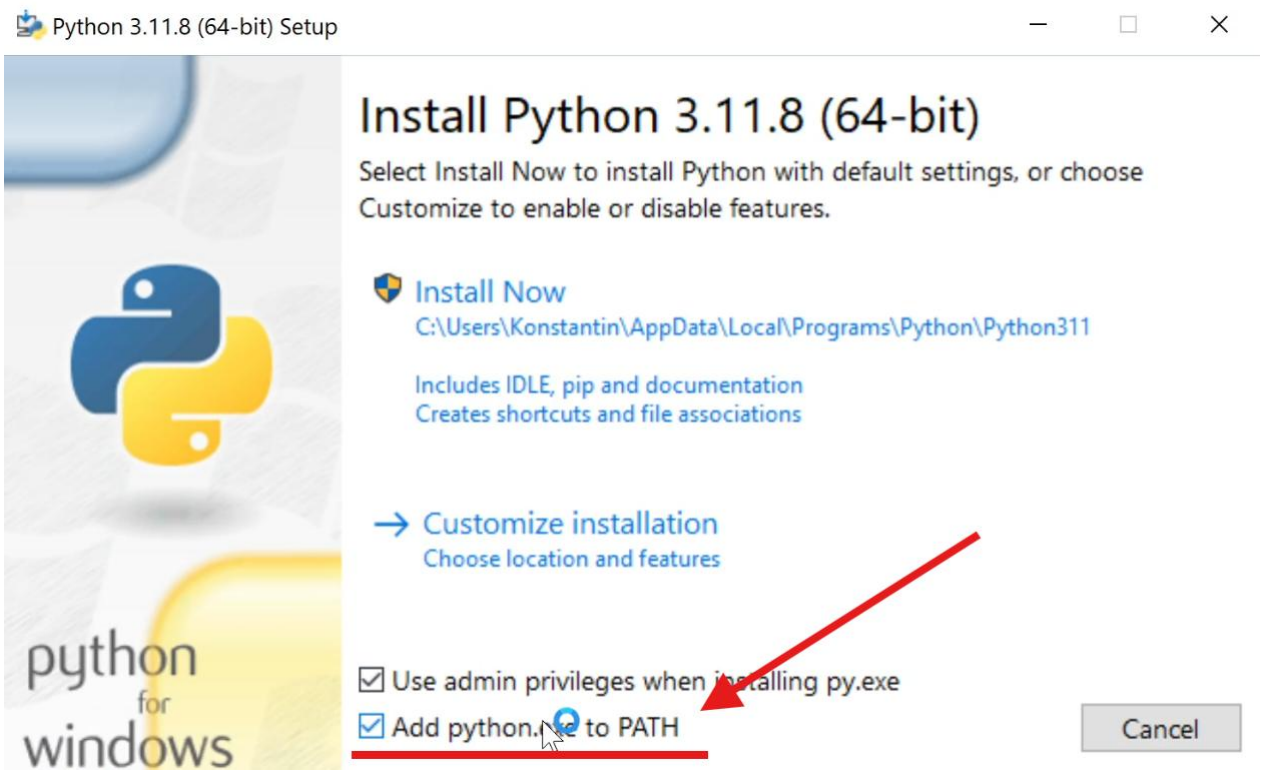

Рис. 1.4. Встановлення Python 3.11.

2. Тепер встановимо Java Development Kit, доступний за посиланням. Перед завантаженням необхідно обрати версію JDK 17 та ОС Windows, як показано на рис. 1.5.

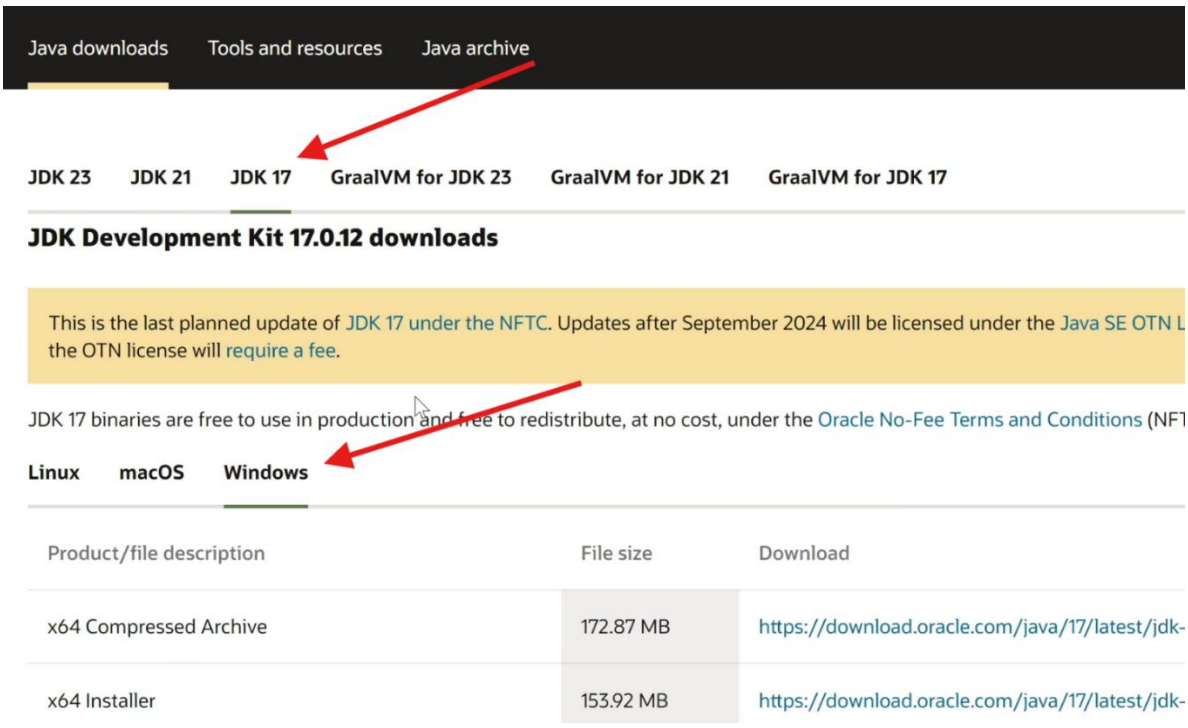

Рис. 1.5. Завантаження Java Development Kit 17.

3. Наступним кроком є завантаження Apache Spark, доступного за посиланням. Завантажуємо версію 3.5.2, як показано на рис. 1.6. Для зручності можна скористатись прямим посиланням на дану версію.

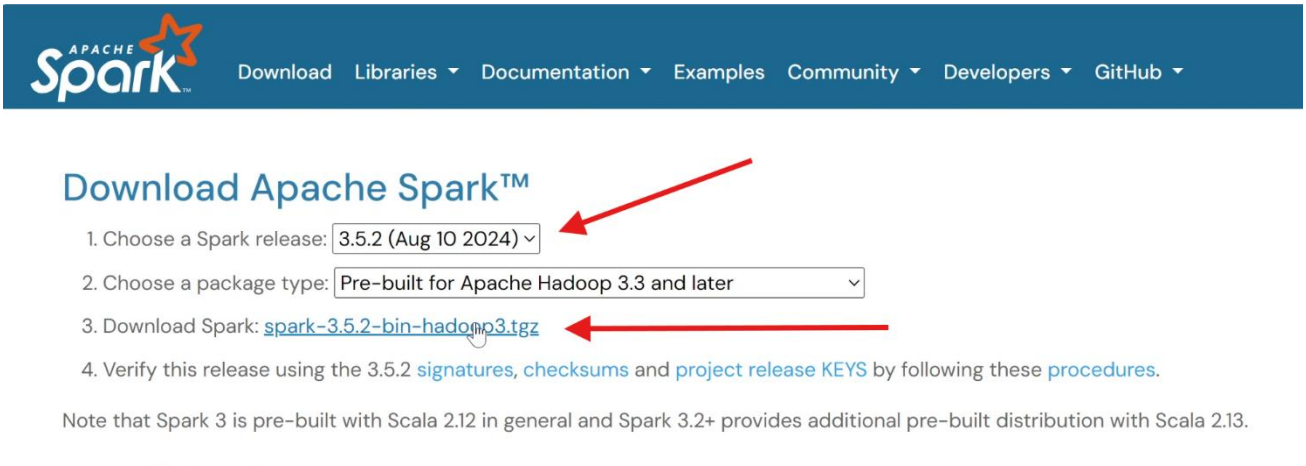

Рис. 1.6. Версія Apache Spark для завантаження.

4. Для встановлення Apache Spark переходимо в корінь диску C:\ та створюємо папку «spark» (рис. 1.7).

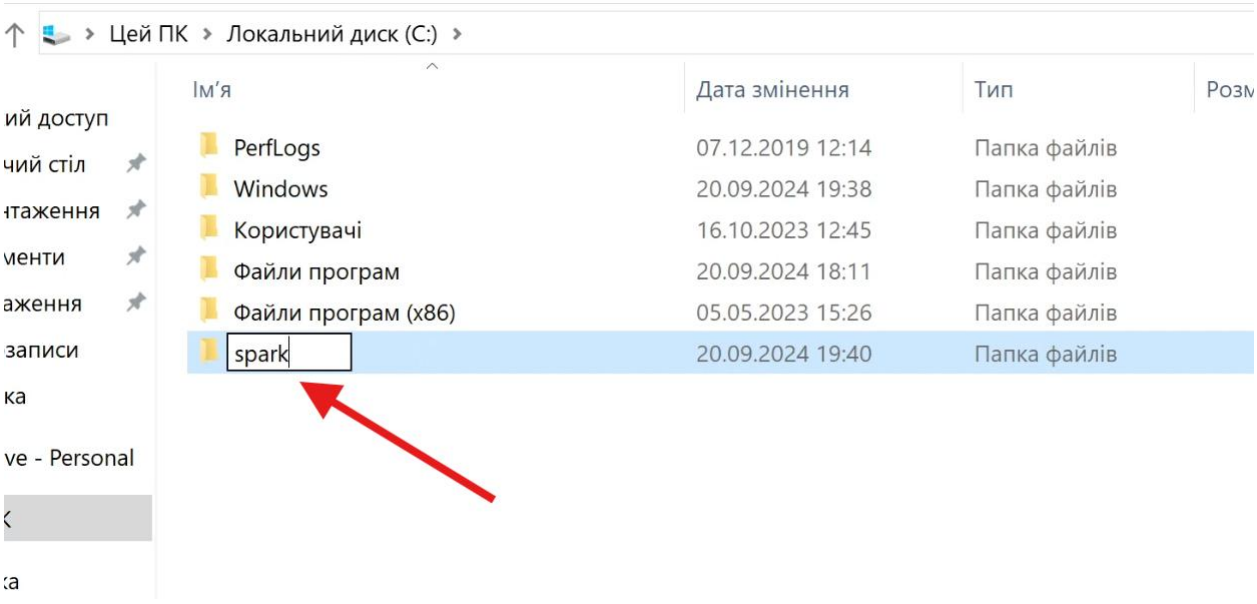

Рис. 1.7. Папка «spark».

5. Копіюємо всі файлі з архіву, завантаженого на кроці 3, в створену папку spark так, щоб зміст папки C:\spark відповідав рис. 1.8.

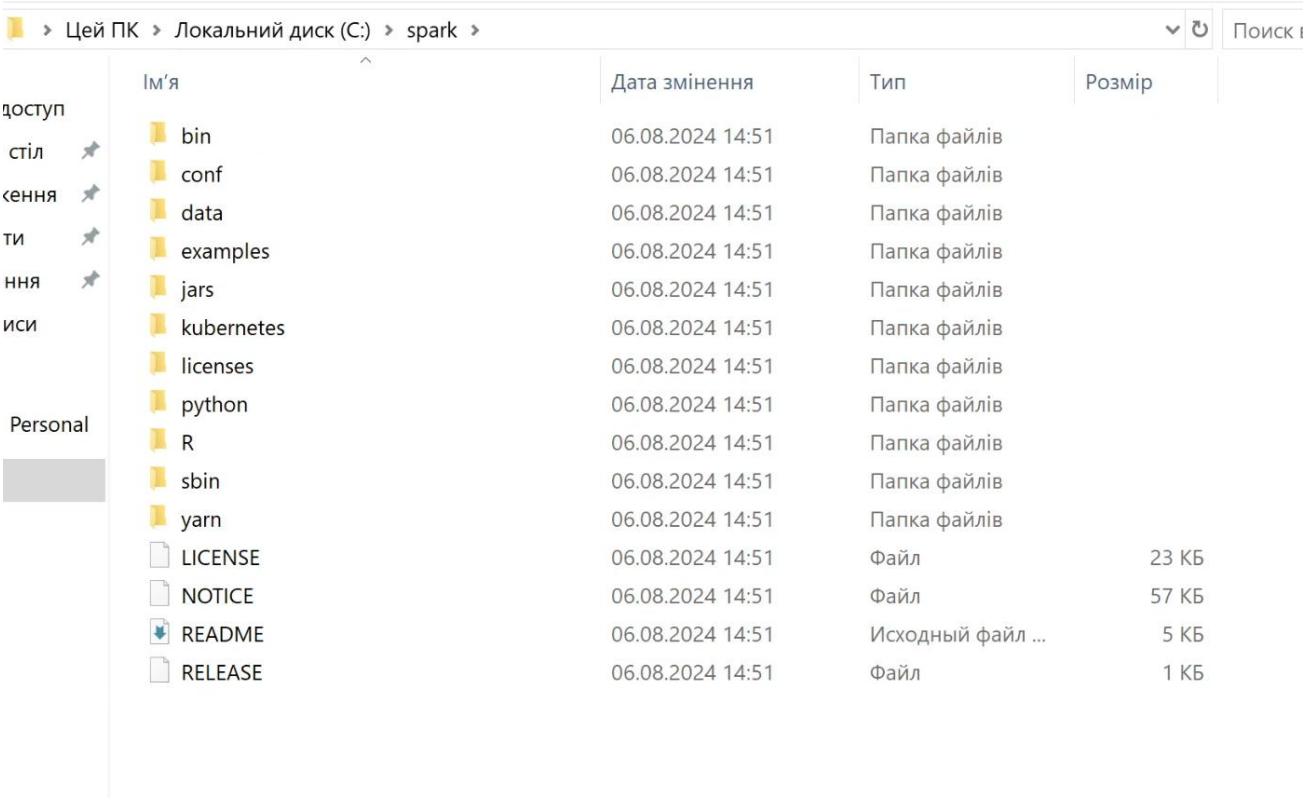

Рис. 1.8. Зміст папки C:\spark

6. Завантажуємо Hadoop 3.3.6. Для цього переходимо за [посиланням], натискаємо «…», далі «Download» (рис. 1.9). Завантажився 1 файл winutils.exe.

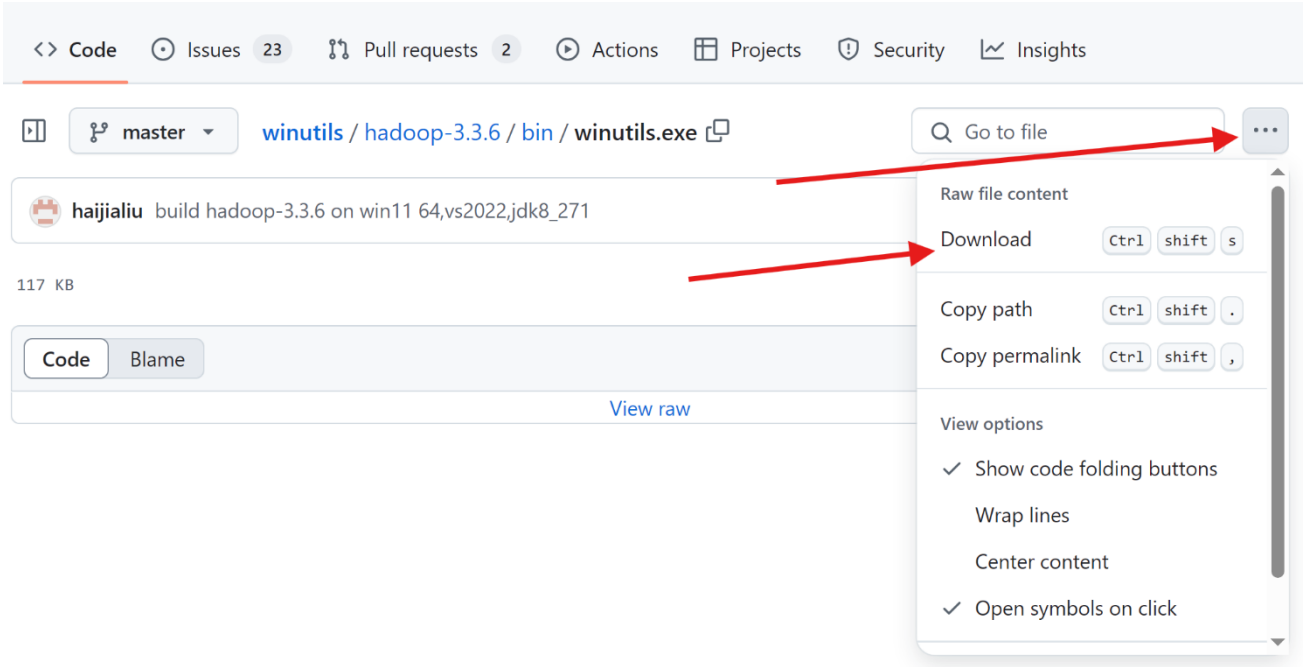

Рис. 1.9. Завантаження Apache Hadoop.

7. Створюємо папку C:\hadoop (рис. 1.10), і в ній папку bin (рис. 1.11).

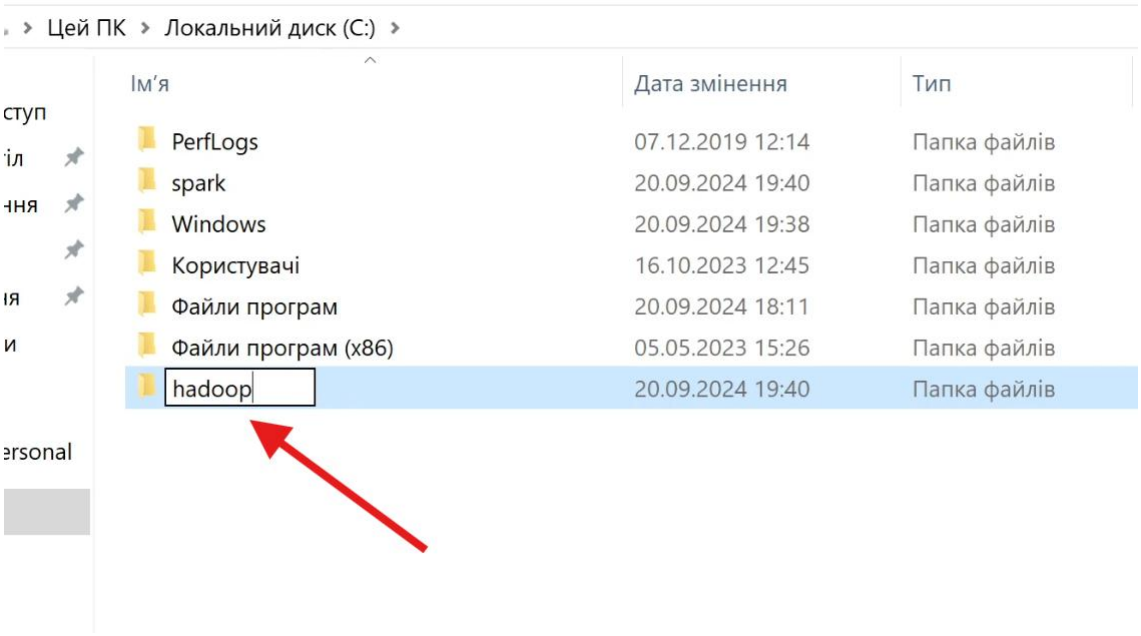

Рис. 1.10. Створення папки hadoop.

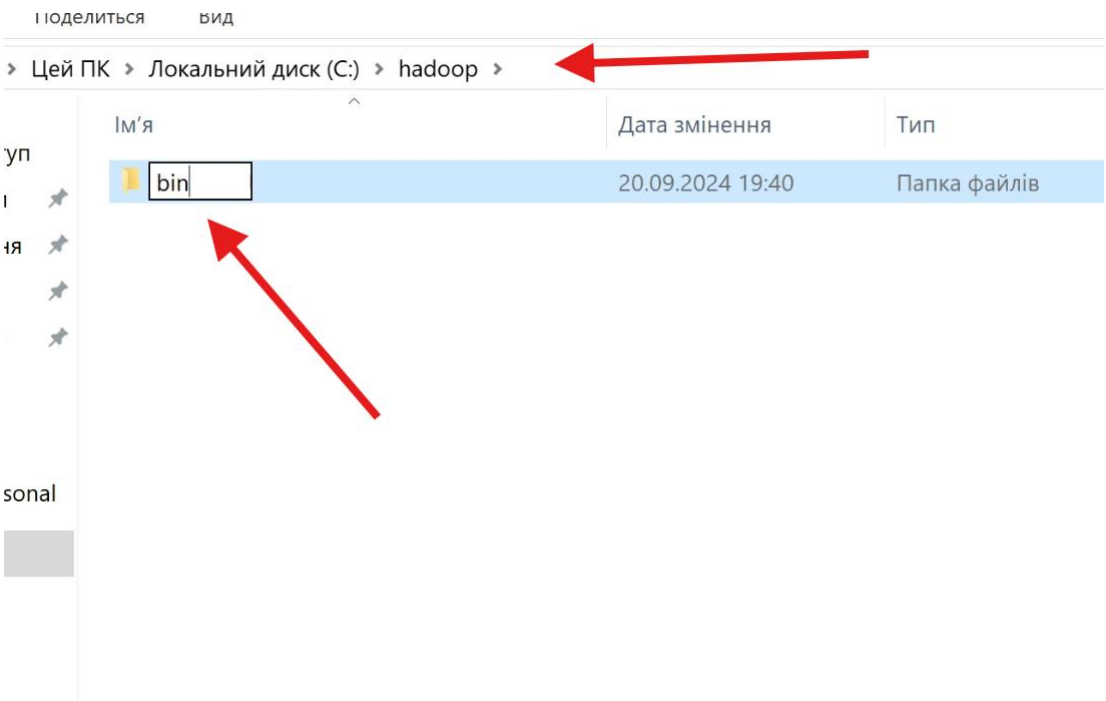

Рис. 1.11. Створення теки bin.

8. Копіюємо файл winutils.exe, завантажений на кроці 6 в папку C:\hadoop\bin (рис. 1.12).

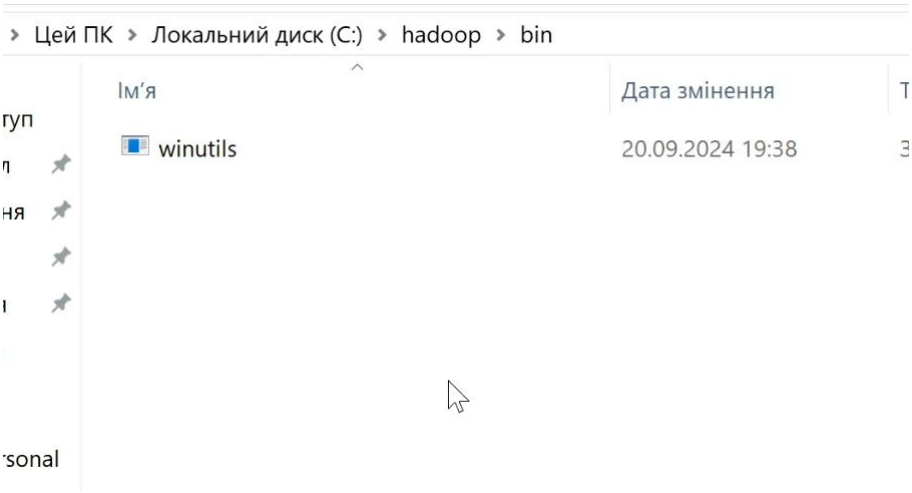

Рис. 1.12. Завершене встановлення Hadoop.

9. Переходимо до створення необхідних змінних оточення системи. Для цього в пошуку вводимо та натискаємо «Редагування змінних оточення системи» (рис. 1.13).

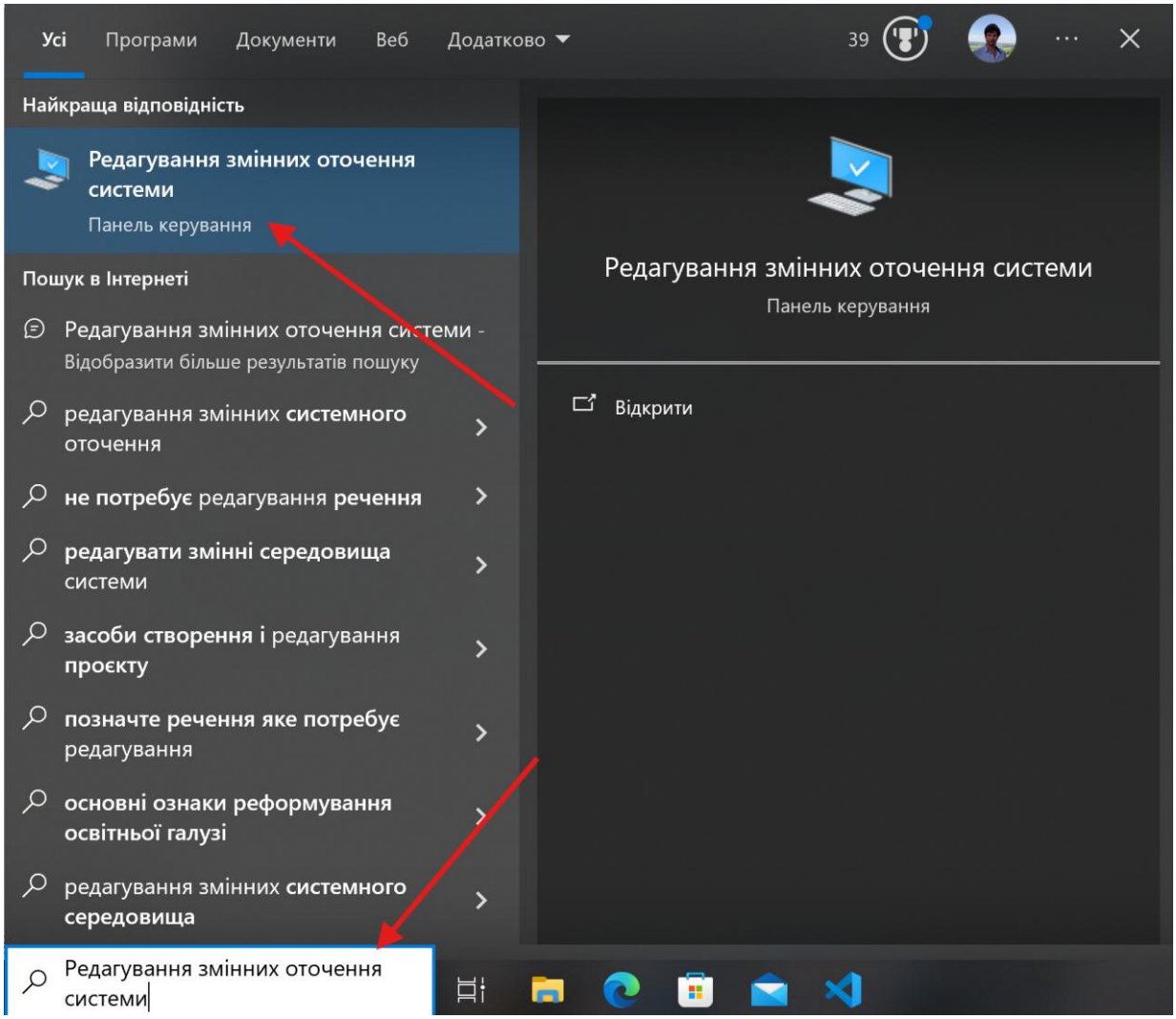

Рис. 1.13. Пошук «Редагування змінних оточення системи».

10. У вікні, що відкрилось, обираємо «Змінні оточення…» (рис. 1.14).

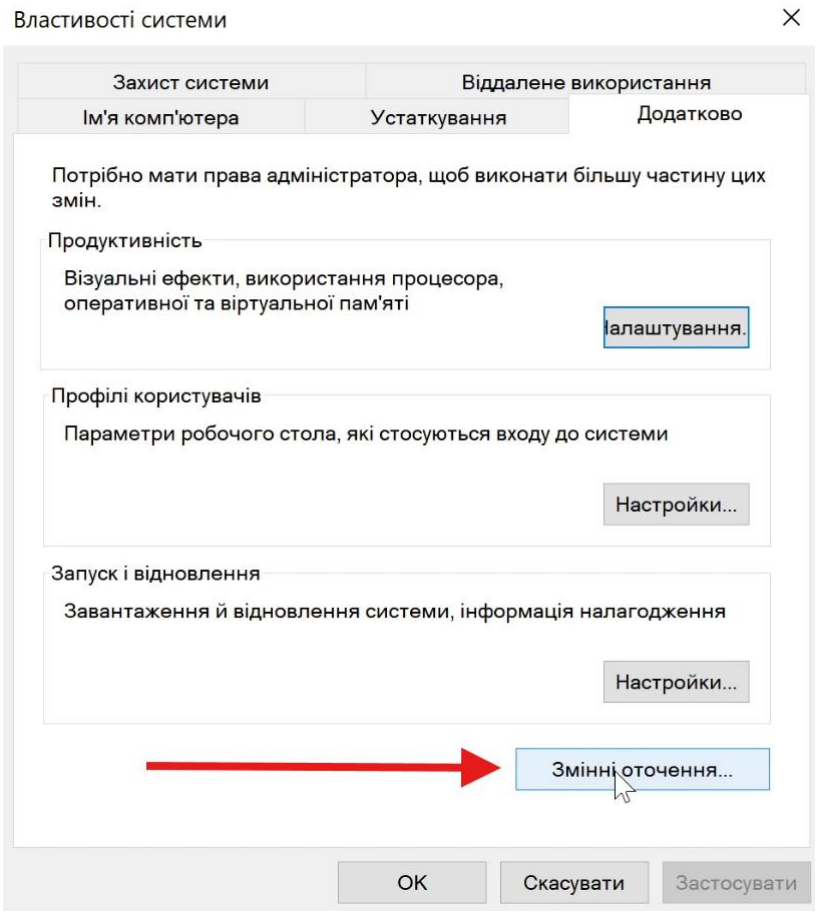

Рис. 1.14. Змінні оточення.

11.Далі створюємо змінні (рис. 1.15):

- Ім’я змінної: JAVA_HOME
  Значення змінної: C:\Program Files\Java\jdk-17
  Як показано на рис. 1.16.
- Ім’я змінної: HADOOP_HOME
  Значення змінної: C:\hadoop
  Як показано на рис. 1.17.
- Ім’я змінної: SPARK_HOME
  Значення змінної: C:\spark
  Як показано на рис. 1.18.
- Ім’я змінної: PYSPARK_PYTHON
  Значення змінної: python
  Як показано на рис. 1.19.

Остаточний результат створення змінних показано на рис. 1.20.

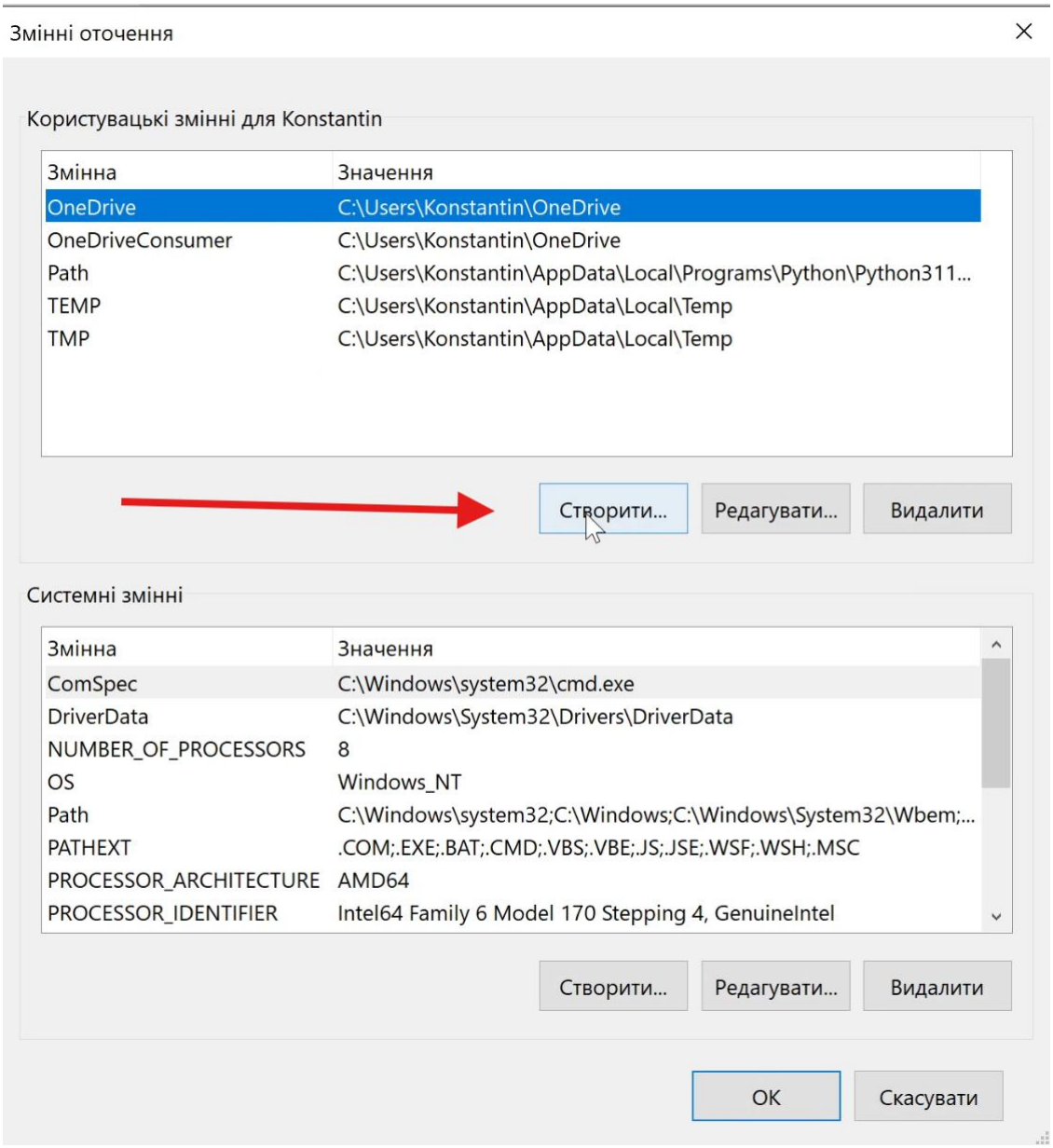

Рис. 1.15.Створення змінних оточення.

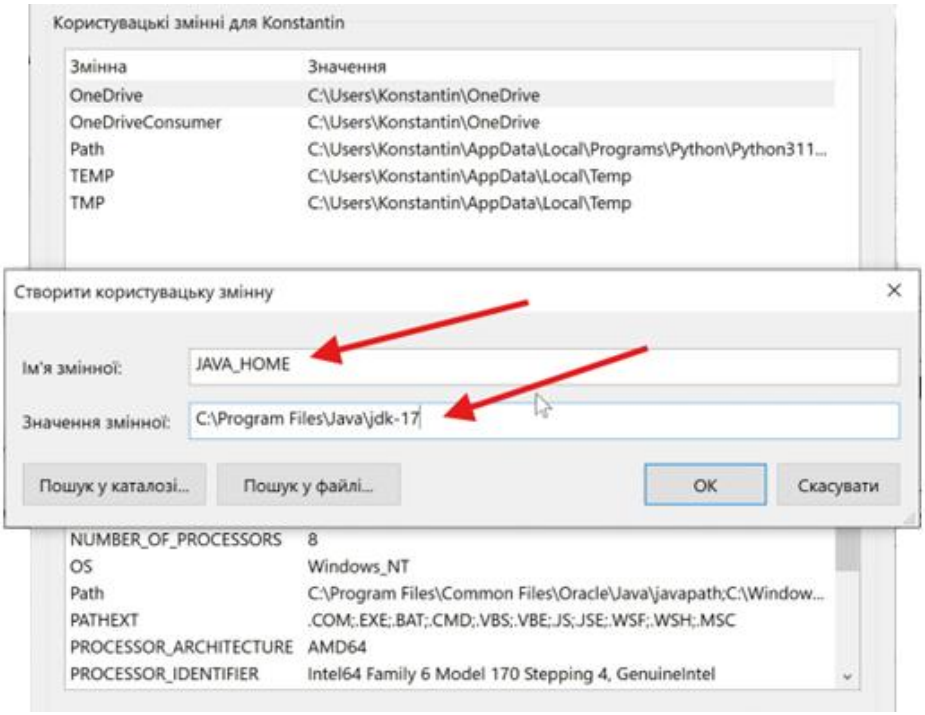

Рис. 1.16. Створення змінної JAVA_HOME.

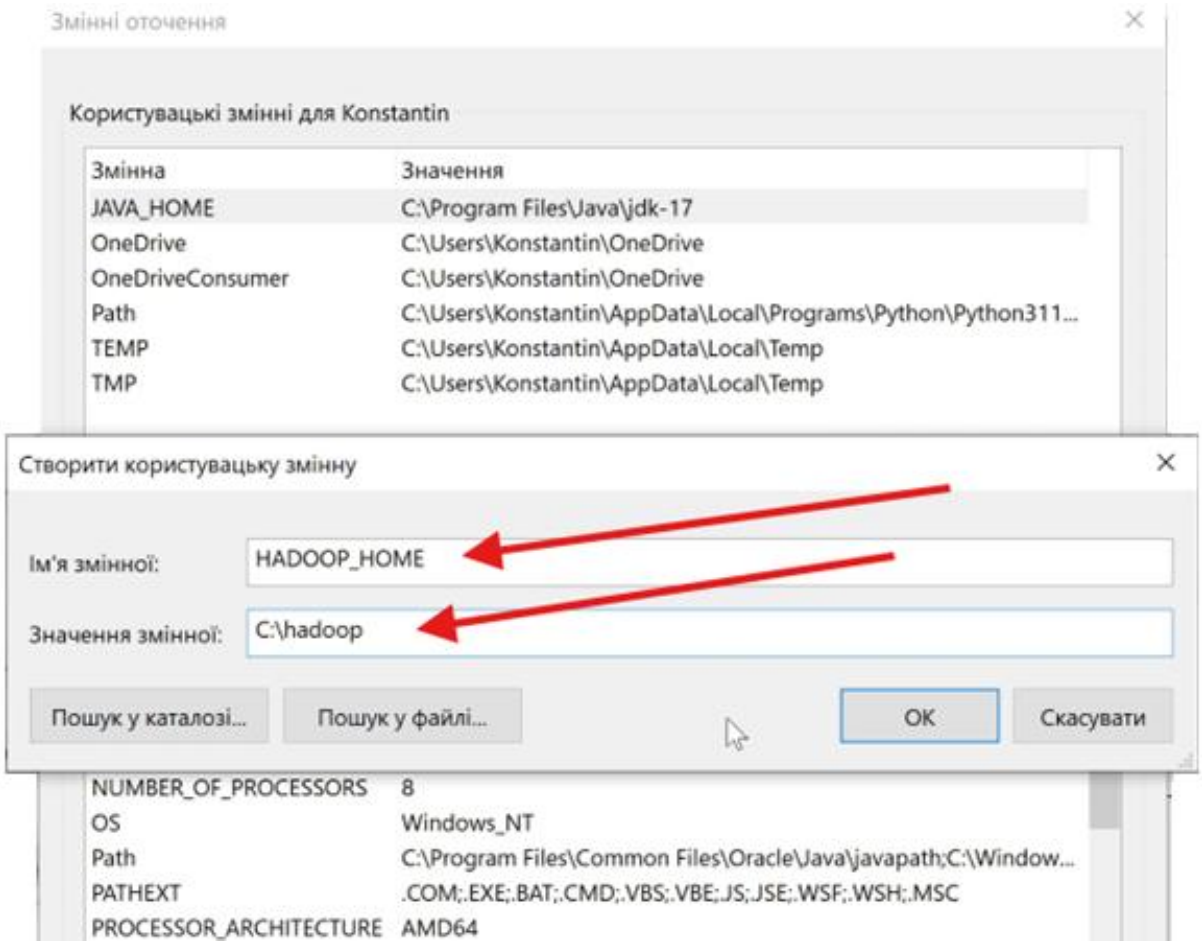

Рис. 1.17. Створення змінної HADOOP_HOME.

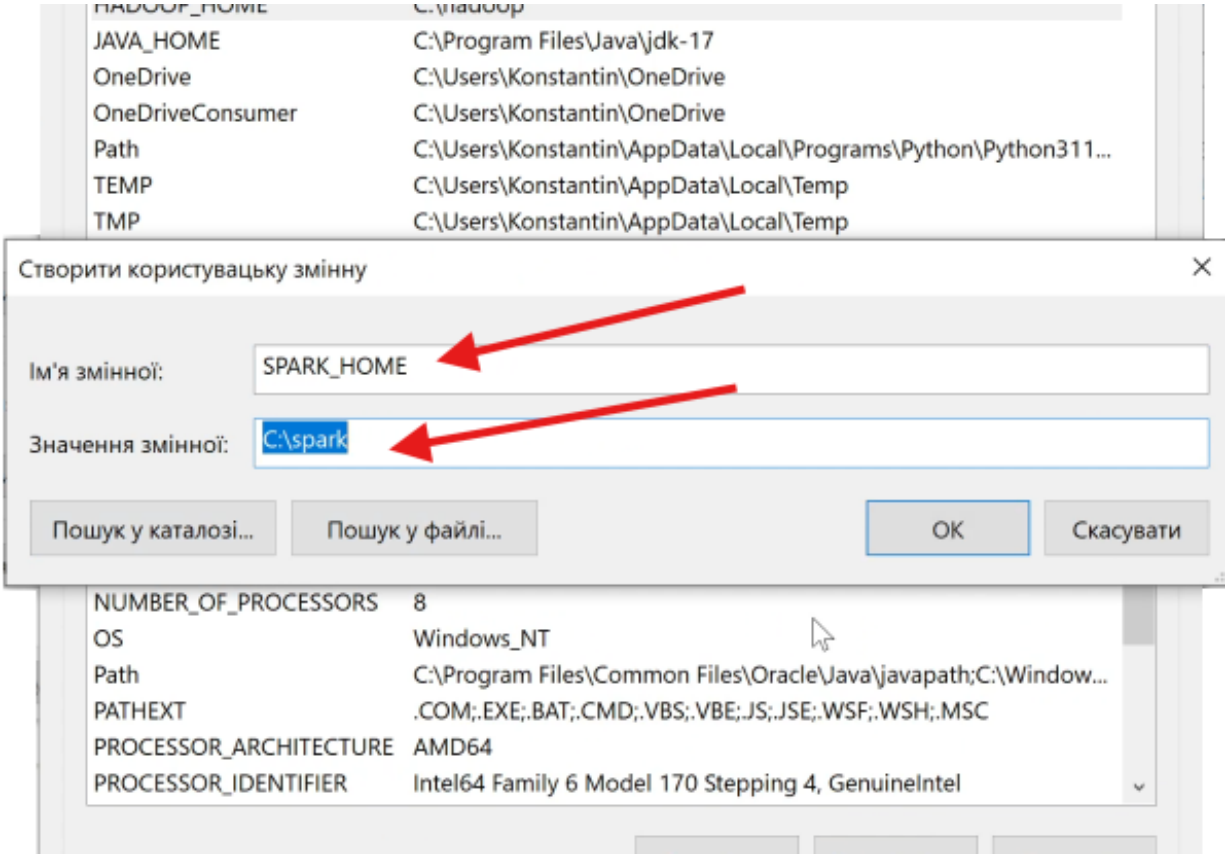

Рис. 1.18. Створення змінної SPARK_HOME.

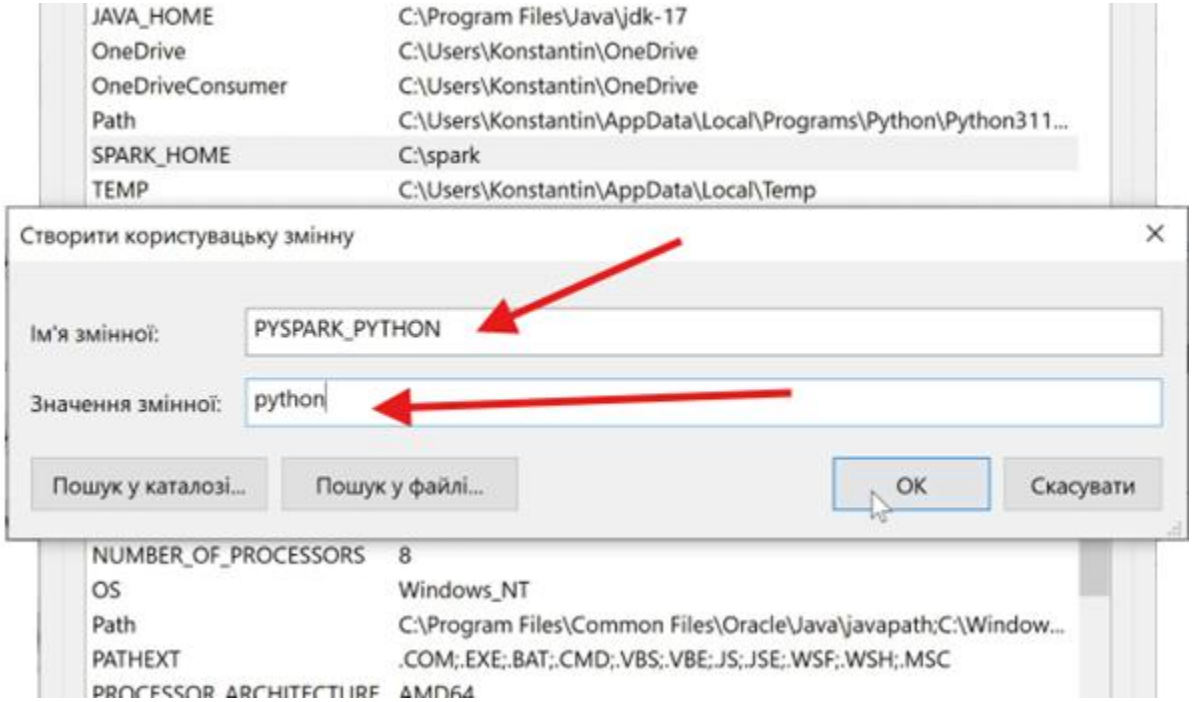

Рис. 1.19. Створення змінної PYSPARK_PYTHON.

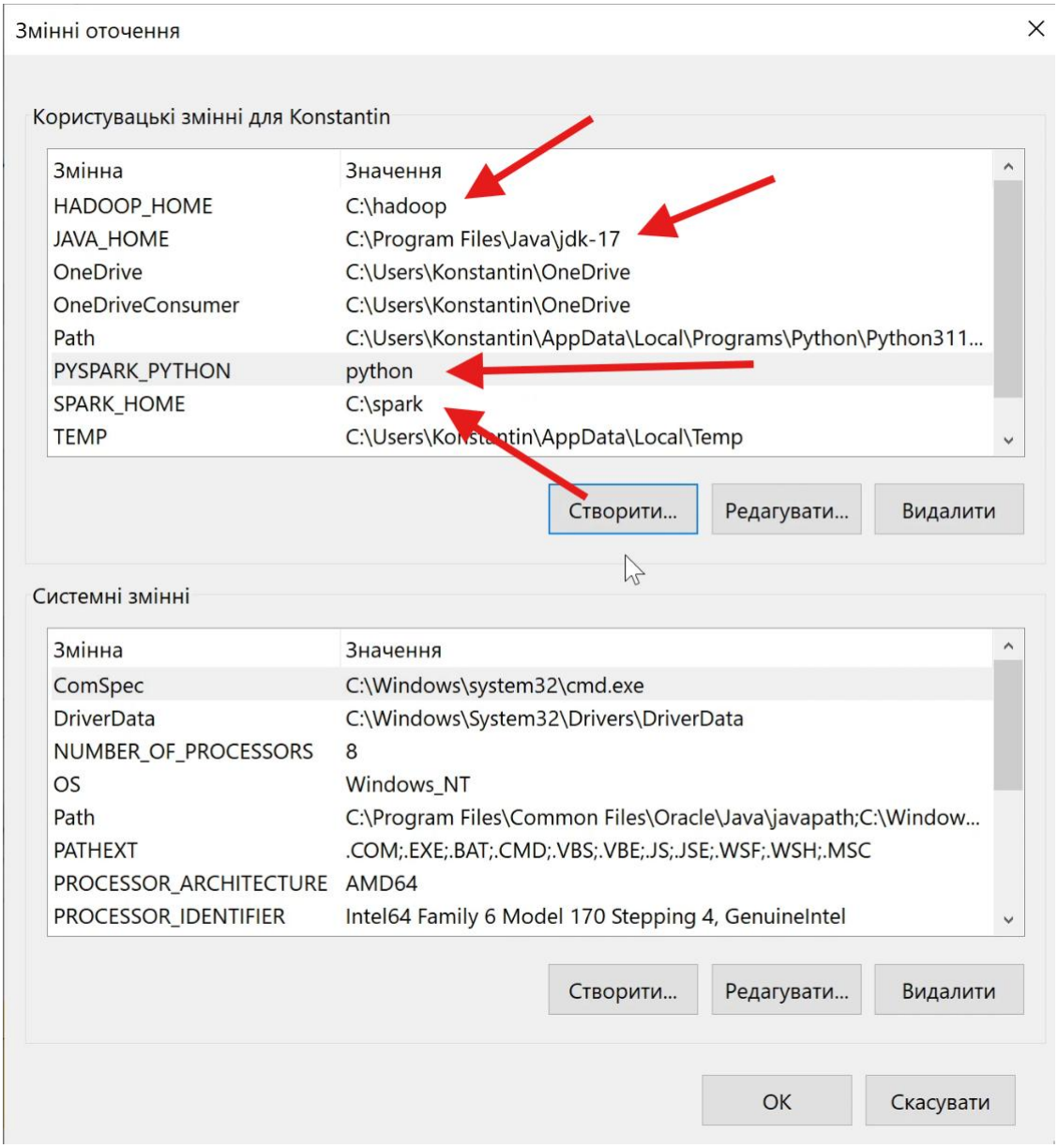

Рис. 1.20. Остаточний результат створення змінних середовища.

12. Редагуємо змінну середовища Path. Для цього обираємо «Path» та натискаємо «Редагувати» (рис. 1.21). У вікні, що відкрилось, натискаємо «Створити» та додаємо «C:\Program Files\Java\jdk-17\bin» (без лапок), аналогічним чином додаємо «C:\spark\bin» та «C:\hadoop\bin». Результат додавання змінних показано рис. 1.22. Для збереження результатів натискаємо «ОК».

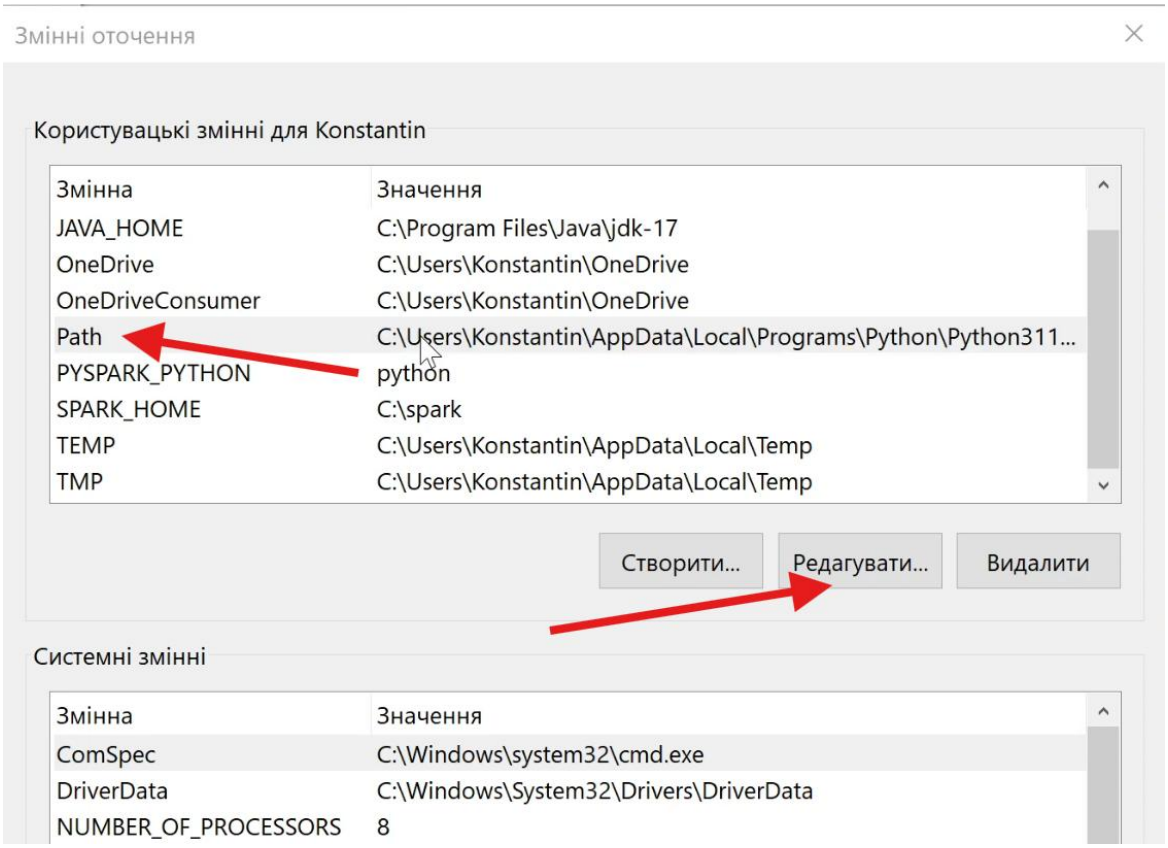

Рис. 1.21. Редагування змінної «Path».

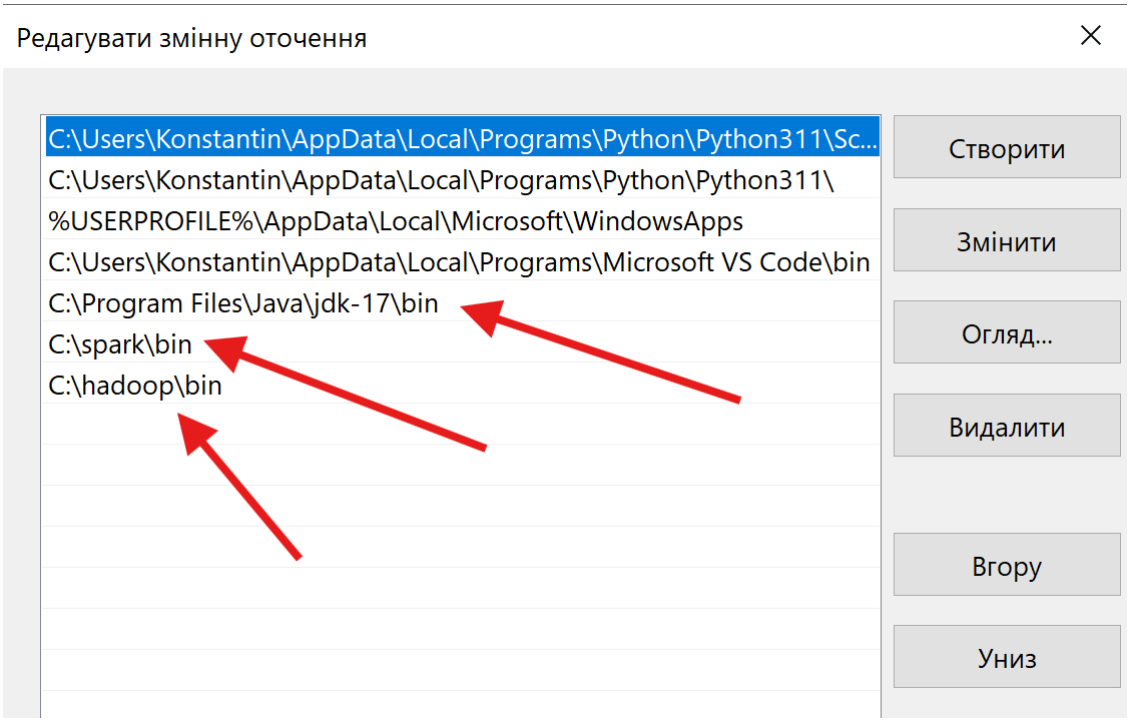

Рис. 1.22. Остаточний результат редагування змінної «Path».

13. Останнім компонентом, що необхідно встановити, є PySpark. Це робиться відкриттям командного рядка та виконанням команди «pip install pyspark» (рис. 1.23). Після цього ви можете запускати Spark програми на власному комп'ютері.

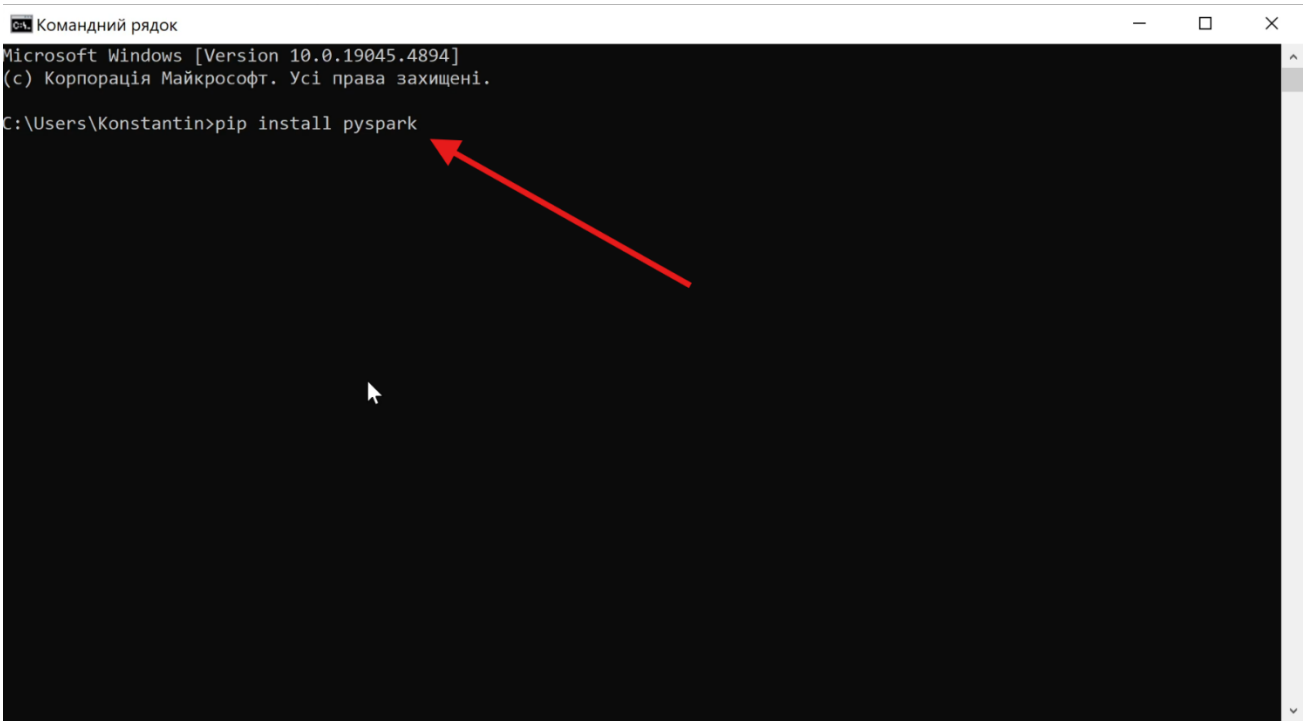

Рис. 1.23. Встановлення PySpark.

2.7. Контрольні питання

1. З якими технологіями та мовами програмування може взаємодіяти Spark?
2. Чому Spark набув такої популярності для обробки великих даних?
3. Без створення екземпляру якого класу неможлива жодна обробка даних в Spark?
4. Як пов'язані між собою поняття драйверу, робітника та менеджера кластера (в термінології Spark)? За якими компонентами закріплено які ролі?
5. Яка кількість робітників може бути в одному кластері Spark?
6. Поясніть поняття кадру даних. Які аналогічні структури даних ви знаєте в інших програмних пакетах?
7. Яку роль відіграє сесії Spark? Чи можлива робота із кадром даних без створення сесії?
8. Здійсніть встановлення та налаштування PySpark на власному комп'ютері. Які додаткові програмні пакети необхідно було для цього встановити?

Лекція 3 – Операції над кадрами даних. Інструменти візуалізації

В епоху великих даних ефективна та масштабована обробка даних є надзвичайно важливою для організацій у різних галузях. Apache Spark з його надійною екосистемою став провідним фреймворком для розподілених обчислень. Серед багатьох його компонентів PySpark виділяється як потужний інструмент, що поєднує дружнє середовище Python з можливостями розподілених обчислень Spark. В основі PySpark лежить DataFrame (кадр даних), універсальна та ефективна структура даних, призначена для легкої обробки великих наборів даних.

Цей розділ заглиблюється у світ кадрів даних PySpark, досліджуючи їх фундаментальні концепції, ключові операції та практичне застосування. Незалежно від того, чи є ви аналітиком даних, інженером-програмістом або дослідником, який працює з великими даними, розуміння того, як маніпулювати та аналізувати дані за допомогою кадрів даних PySpark є важливим для використання повного потенціалу Spark у ваших проектах.

3.1. Що таке кадр даних (DataFrame) в Spark?

DataFrame – це розподілена колекція даних, організована в іменовані стовпці. Він схожий на таблицю в реляційній базі даних або кадр даних у бібліотеці pandas в R/Python, але оптимізований для великомасштабної обробки даних. PySpark DataFrames надають високорівневий API, який абстрагується від складнощів розподілених обчислень, дозволяючи користувачам писати стислий і читабельний код.

У цій лекції буде розглянуто ряд загальних операцій, які можна виконувати з фреймами даних PySpark:

1. Побудова кадру даних.
2. Ліниве виконання Spark та візуалізація кадру даних.
3. Читання даних з файлу.
4. Фільтрування даних.
5. Групування та агрегація.
6. Об’єднання (операція Join).
7. Сортування та впорядкування.

У цій главі будуть надані практичні приклади, щоб проілюструвати, як кадри даних PySpark можуть бути використані в реальних сценаріях. Від аналізу поведінки клієнтів в електронній комерції до оптимізації логістики ланцюгів поставок, кадри даних PySpark є універсальним інструментом, який може покращити ваші робочі процеси обробки даних. Наприкінці цього розділу ви матимете ґрунтовне розуміння фреймів даних PySpark і навички їх ефективного застосування у ваших проектах з великими даними.

3.2. Побудова Spark DataFrame

Існує кілька різних методів створення Spark DataFrame. Наприклад, оскільки DataFrame – це в основному набір даних із рядків, ми можемо створити DataFrame із колекції рядків за допомогою методу createDataFrame() із сеансу Spark:

Код

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
from datetime import date
from pyspark.sql import Row

data = [
  Row(id = 1, value = 28.3, date = date(2021,1,1)),
  Row(id = 2, value = 15.8, date = date(2021,1,1)),
  Row(id = 3, value = 20.1, date = date(2021,1,2)),
  Row(id = 4, value = 12.6, date = date(2021,1,3))
]

df = spark.createDataFrame(data)
```

Spark DataFrame у Python є об’єктом класу pyspark.sql.dataframe.DataFrame, як ви бачите нижче:

Код

```
print(type(df))
```

Вивід

```
<class 'pyspark.sql.dataframe.DataFrame'>
```

Якщо ви спробуєте побачити, що знаходиться всередині такого об’єкта, ви отримаєте невеликий опис стовпців, присутніх у DataFrame:

Код

```
print(df)
```

Вивід

```
DataFrame[id: bigint, value: double, date: date]
```

Отже, у наведеному вище прикладі ми використовуємо конструктор Row() (з модуля pyspark.sql) для створення 4 рядків. Метод createDataFrame() поєднує ці

4 рядки разом, щоб сформувати наш новий DataFrame df. Результатом є Spark DataFrame із 4 рядками та 3 стовпцями (ідентифікатор, значення та дата).

Але ви можете використовувати різні методи для створення того самого Spark DataFrame. За допомогою наведеного нижче коду ми створюємо DataFrame під назвою students з двох різних списків Python (даних і стовпців).

Перший список (дані) – це список рядків. Кожен рядок представлено кортежем Python, який містить значення в кожному стовпці. А другий список (стовпці) містить імена для кожного стовпця в DataFrame.

Щоб створити DataFrame студентів, використаємо ці два списки в методі createDataFrame():

Код

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.getOrCreate()
from datetime import date
from pyspark.sql import Row

data = [
  (12114, 'Anne', 21, 1.56, 8, 9, 10, 9, 'Economics', 'SC'),
  (13007, 'Adrian', 23, 1.82, 6, 6, 8, 7, 'Economics', 'SC'),
  (10045, 'George', 29, 1.77, 10, 9, 10, 7, 'Law', 'SC'),
  (12459, 'Adeline', 26, 1.61, 8, 6, 7, 7, 'Law', 'SC'),
  (10190, 'Mayla', 22, 1.67, 7, 7, 7, 9, 'Design', 'AR'),
  (11552, 'Daniel', 24, 1.75, 9, 9, 10, 9, 'Design', 'AR')
]

columns = [
  'StudentID', 'Name', 'Age', 'Height', 'Score1',
  'Score2', 'Score3', 'Score4', 'Course', 'Department'
]

students = spark.createDataFrame(data, columns)
print(students)
```

Вивід

```
DataFrame[StudentID: bigint, Name: string, Age: bigint, Height: double,
Score1: bigint, Score2: bigint, Score3: bigint, Score4: bigint,
Course: string, Department: string]
```

Примітка. Насправді Spark виводить деяку системну інформації під час роботи. В прикладах виводу її не наведено. Читач, запускаючи код на своєму комп’ютері, також може ігнорувати її.

3.3. Візуалізація кадру даних

Ключовим аспектом Spark є ліниве виконання. Іншими словами, для більшості операцій Spark лише перевірятиме, чи правильний ваш код і чи має він сенс. Spark фактично не запускатиме та не виконуватиме операції, які ви описуєте у своєму коді, якщо ви явно не попросите це за допомогою операції тригера.

Ви можете помітити цю «лінь» у вихідних даних нижче:

Код

```
print(students)
```

Вивід

```
DataFrame[StudentID: bigint, Name: string, Age: bigint, Height: double,
Score1: bigint, Score2: bigint, Score3: bigint, Score4: bigint, Course:
string, Department: string]
```

Зверніть увагу що, коли ми друкуємо об’єкт, який зберігає Spark DataFrame (наприклад, df і students), Spark обчислить і надрукує лише загальну інформацію про структуру вашого Spark DataFrame, а не сам DataFrame.

Отже, як ми можемо побачити наш кадр даних? Як ми можемо візуалізувати рядки та значення, які зберігаються в ньому? Для цього використаємо метод show(). За допомогою якого Spark надрукує таблицю в текстовому форматі, як видно нижче:

Код

```
students.show()
```

Вивід

```
+---------+-------+---+------+------+------+------+------+---------+----------+
|StudentID|   Name|Age|Height|Score1|Score2|Score3|Score4|   Course|Department|
+---------+-------+---+------+------+------+------+------+---------+----------+
|    12114|   Anne| 21|  1.56|     8|     9|    10|     9|Economics|        SC|
|    13007| Adrian| 23|  1.82|     6|     6|     8|     7|Economics|        SC|
|    10045| George| 29|  1.77|    10|     9|    10|     7|      Law|        SC|
|    12459|Adeline| 26|  1.61|     8|     6|     7|     7|      Law|        SC|
|    10190|  Mayla| 22|  1.67|     7|     7|     7|     9|   Design|        AR|
|    11552| Daniel| 24|  1.75|     9|     9|    10|     9|   Design|        AR|
+---------+-------+---+------+------+------+------+------+---------+----------+
```

За замовчуванням цей метод показує лише верхні рядки вашого DataFrame, але ви можете вказати, скільки саме рядків ви хочете бачити, використовуючи show(n), де n – це кількість рядків. Наприклад, можна візуалізувати лише перші 2 рядки df таким чином:

Код

```
students.show(2)
```

Вивід

```
+---------+------+---+------+------+------+------+------+---------+----------+
|StudentID|  Name|Age|Height|Score1|Score2|Score3|Score4|   Course|Department|
+---------+------+---+------+------+------+------+------+---------+----------+
|    12114|  Anne| 21|  1.56|     8|     9|    10|     9|Economics|        SC|
|    13007|Adrian| 23|  1.82|     6|     6|     8|     7|Economics|        SC|
+---------+------+---+------+------+------+------+------+---------+----------+
only showing top 2 rows
```

Отримання назв стовпців. Якщо вам потрібно, ви можете легко зібрати список Python із назвами стовпців, наявними у вашому DataFrame, так само, як ви це зробили б у pandas DataFrame – за допомогою методу columns вашого DataFrame, наприклад:

Код

```
print(students.columns)
```

Вивід

```
['StudentID', 'Name', 'Age', 'Height', 'Score1', 'Score2', 'Score3', 'Score4',
'Course', 'Department']
```

Отримання кількості рядків. Якщо ви хочете дізнатися кількість рядків у Spark DataFrame, просто скористайтеся методом count() цього DataFrame. У результаті Spark створить цей DataFrame і підрахує кількість наявних у ньому рядків.

Код

```
print(students.count())
```

Вивід

```
6
```

3.4. Читання даних з файлу

Найчастіше ви будете не створювати кадри даних з нуля, а вивантажувати їх з іншого джерела, будь-то база даних або файл. Розглянемо читання даних з файлу CSV – текстового файлу, де колонки розділені комами.

Створимо файл data.csv в тій же директорії, що і файл Python, із наступним змістом:

data.csv

```
id,"name","occupation","salary"
1,"John Doe","Software Engineer",80000
2,"Jane Smith","Data Scientist",95000
3,"Bob Brown","DevOps Engineer",70000
4,"Alice Johnson","Product Manager",100000
5,"Mike Davis","Cloud Architect",110000
```

Для читання файлу із Spark, використаємо команду `spark.read.csv('data.csv', header=True, inferSchema=True)`.

- `header=True` повідомляє PySpark, що перший рядок у CSV містить назви стовпців.
- `inferSchema=True` дозволяє PySpark автоматично визначати типи даних стовпців на основі їх вмісту.

Код

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("Load CSV").getOrCreate()

df = spark.read.csv('data.csv', header=True, inferSchema=True)

df.show()
```

Вивід

```
+---+-------------+-----------------+------+
| id|         name|       occupation|salary|
+---+-------------+-----------------+------+
|  1|     John Doe|Software Engineer| 80000|
|  2|   Jane Smith|   Data Scientist| 95000|
|  3|    Bob Brown|  DevOps Engineer| 70000|
|  4|Alice Johnson|  Product Manager|100000|
|  5|   Mike Davis|  Cloud Architect|110000|
+---+-------------+-----------------+------+
```

3.5. Операції над кадрами даних PySpark

Кадри даних PySpark надають різні операції для обробки та аналізу даних. Розглянемо на прикладах деякі з найбільш часто використовуваних операцій.

1. Фільтрування даних

Фільтрування дозволяє вибирати рядки, які відповідають певній умові.

Код

```
from pyspark.sql import SparkSession

# Створюємо простий кадр даних
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [("Alice", 25), ("Bob", 30), ("Charlie", 35)]
df = spark.createDataFrame(data).toDF("name", "age")

# Фільтруємо рядки із віком більше 30
filtered_df = df.filter(df.age > 30)
filtered_df.show()
```

Вивід

```
+-------+---+
|   name|age|
+-------+---+
|Charlie| 35|
+-------+---+
```

2. Групування та агрегація

Групування дозволяє групувати рядки за одним або декількома стовпцями, тоді як агрегування дозволяє виконувати обчислення для кожної групи.

Код

```
from pyspark.sql import SparkSession

# Створюємо кадр даних
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [("Alice", 25000), ("Bob", 30000), ("Charlie", 35000), ("Alice", 14000)]
df = spark.createDataFrame(data).toDF("name", "salary")

# Групуємо за колонкою 'name' та рахуємо середнє за колонкою 'salary'
grouped_df = df.groupBy("name").mean()
grouped_df.show()
```

Вивід

```
+-------+-----------+
|   name|avg(salary)|
+-------+-----------+
|  Alice|    19500.0|
|    Bob|    30000.0|
|Charlie|    35000.0|
+-------+-----------+
```

3. Об’єднання

Об’єднання дозволяє об’єднати два кадра даних на основі спільного стовпця, функція є подібною до знайомої JOIN з SQL.

Код

```
from pyspark.sql import SparkSession

# Створюємо кадр даних
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [("Alice", 25000), ("Bob", 30000), ("Charlie", 35000), ("Alice", 14000)]
df = spark.createDataFrame(data).toDF("name", "salary")

# Створюємо ще одним кадр даних з колонками 'name' та 'occupation'
data2 = [("Alice", "Engineer"), ("Bob", "Doctor"), ("Charlie", "Teacher")]
columns2 = ["name", "occupation"]
df2 = spark.createDataFrame(data2).toDF("name", "occupation")

# Об’єднуємо df та df2 за колонкою 'name'
joined_df = df.join(df2, "name")
joined_df.show()
```

Вивід

```
+-------+------+----------+
|   name|salary|occupation|
+-------+------+----------+
|  Alice| 25000|  Engineer|
|  Alice| 14000|  Engineer|
|    Bob| 30000|    Doctor|
|Charlie| 35000|   Teacher|
+-------+------+----------+
```

4. Сортування та впорядкування

Сортування дозволяє впорядковувати рядки за одним або кількома стовпцями.

Код

```
from pyspark.sql import SparkSession

# Створюємо кадр даних
spark = SparkSession.builder.appName("Example").getOrCreate()
data = [("Alice", 25000), ("Bob", 30000), ("Charlie", 35000), ("Alice", 14000)]
df = spark.createDataFrame(data).toDF("name", "salary")
```

```
# Сортуємо за спаданням колонки 'salary'
sorted_df = df.sort(df.salary.desc())
sorted_df.show()
```

Вивід

```
+-------+------+
|   name|salary|
+-------+------+
|Charlie| 35000|
|    Bob| 30000|
|  Alice| 25000|
|  Alice| 14000|
+-------+------+
```

3.6. Контрольні питання

1. Які способи створення кадру даних існують? Як зчитати дані в кадр даних?
2. В чому відмінність кадру даних PySpark від аналогічної структури в інших прикладних пакетах?
3. Чи отримаємо ми вміст кадру даних PySpark якщо запустимо команду print(df)? Що необхідно зробити, щоб візуалізувати кадр даних на консоль?
4. Як здійснити фільтрування даних в DataFrame?
5. Наведіть декілька прикладів задач, що потребують агрегації даних. Створіть кадр даних із вигаданими даними та застосуйте необхідні операції агрегації.
6. В який спосіб можна об’єднати дані з декількох таблиць в PySpark аналогічно до команди SQL JOIN?
7. Після операції групування даних необхідно порахувати кількість результатів, а потім, якщо результатів багато, візуалізувати лише перші 10 рядків отриманого кадру даних. Які операції PySpark для цього необхідно використати?
8. Що таке файл CSV? Як провести читання даних з такого файлу?

Лекція 4 – Інструменти машинного навчання в PySpark. Регресія

Класифікація і регресія – найстаріші і найбільш добре вивчені види предиктивної аналітики. Більшість алгоритмів, з якими ви, ймовірно, зіткнетеся в аналітичних пакетах і бібліотеках, є методами класифікації або регресії, такі як метод опорних векторів, логістична регресія, нейронні мережі та глибоке навчання. Спільна нитка, що пов'язує регресію і класифікацію, полягає в тому, що обидві вони припускають прогнозування одного (або декількох) значень на підставі одного (або декількох) інших значень. Для цього їм потрібен набір вхідних і вихідних даних, на яких можна вчитися. Їх потрібно забезпечити як запитаннями, так і еталонними відповідями. З цієї причини вони належать до різновидів навчання з учителем.

PySpark MLlib пропонує реалізації низки алгоритмів класифікації та регресії. До них належать дерева рішень, наївний байєсівський метод, логістична регресія і лінійна регресія. Найприкметніше в цих алгоритмах те, що вони допомагають передбачити майбутнє або, принаймні, передбачити те, чого ми ще не знаємо напевно, наприклад, імовірність того, що ви купите автомобіль, ґрунтуючись на вашій поведінці в Інтернеті; чи є електронний лист спамом, з огляду на слова, які він містить; чи на яких ділянках землі буде зібрано найкращий врожай сільськогосподарських культур з огляду на їхнє місцезнаходження та хімічний склад ґрунту.

У цьому розділі ми зосередимося на популярному і гнучкому типі алгоритму як для класифікації (випадковий ліс), так і для регресії (лінійна регресія). Хоча з реалізацією PySpark можна швидко приступити до роботи, буде корисно зрозуміти основи таких алгоритмів.

4.1. Проста лінійна регресія

Важливим питанням аналізу даних є відповідь на запитання: чи пов'язана змінна X (або, що більш ймовірно, $X_1, \ldots, X_p$) зі змінною Y, і якщо так, то в чому цей зв'язок полягає, і чи можемо ми його використати для того, щоб передбачити Y?

Проста лінійна регресія, або парна лінійна регресія, моделює зв'язок між величиною однієї змінної та величиною другої, наприклад у міру збільшення X збільшується і Y. Або ж у міру збільшення X зменшується і Y. Кореляція – ще один спосіб виміряти те, яким чином дві змінні зв'язані між собою. Різниця між ними полягає в тому, що кореляція вимірює силу зв'язку між двома змінними, тоді як регресія оцінює природу зв'язку кількісно.

4.2. Рівняння регресії

Проста лінійна регресія оцінює, наскільки саме зміниться Y коли X змінюється на деяку величину. Для коефіцієнта кореляції змінні X та Y

взаємозамінні. У разі регресії ми намагаємося передбачити змінну Y зі змінної X, використовуючи лінійний зв'язок (тобто прямий):

$$Y = b_0 + b_1 X$$

Ця формула читається, як «Y дорівнює b_1, помножене на X плюс константа b_0». Компонент рівняння b_0 називається перетином (або константою), а b_1 – нахилом по відношенню до осі x. Змінна Y називається *відгуком* або *залежною змінною*, оскільки вона залежить від X. Змінна X називається *провісником* (предиктором), або *незалежною змінною*. Спільнота машинного навчання тяжіє до використання інших термінів, називаючи Y *метою* та X – *вектором ознак*.

У регресійному аналізі важливими поняттями є *підігнані значення* та *залишки*. Як правило, дані не лягають рівно на пряму, тому рівняння регресії має включати явний залишковий член e_i:

$$Y_i = b_0 + b_1 X_i + e_i.$$

Підігнані значення, також звані передбаченими значеннями, у типовій ситуації позначаються як $\hat{Y}_i$ (Y з капелюхом). Вони задаються такою формулою:

$$\hat{Y}_i = \hat{b}_0 + \hat{b}_1 X_i.$$

Позначення $\hat{b}_0$ і $\hat{b}_1$ свідчить, що ці коефіцієнти оцінюються відносно відомим, тобто є оціночними.

4.3. Найменші квадрати

Яким чином виконується підгонка моделі до даних? Коли існує чіткий зв'язок, ви можете подумки уявити підгонку прямої вручну. Насправді пряма регресії є оцінкою, яка мінімізує значення суми квадратичних залишків, також іменованих *сумою квадратів залишків* чи *залишковою сумою квадратів* (residual sum of squares, RSS):

$$RSS = \sum_{i=1}^{n} \left(Y_i - \hat{Y}_i\right)^2 = \sum_{i=1}^{n} \left(Y_i - \hat{b}_0 - \hat{b}_1 X_i\right)^2$$

Оцінки $\hat{b}_0$ та $\hat{b}_1$ – це значення, які мінімізують суму квадратів залишків (RSS).

Метод мінімізації суми квадратів залишків називається *регресією на основі найменших квадратів*, чи регресією *на основі звичайних найменших квадратів* (ЗНК). Цей метод часто приписується Карлу Фрідріху Гауссу, німецькому математику, але він був вперше опублікований в 1805 французьким математиком Андре-Марі Лежандром (Adrien-Marie Legendre). Регресію на основі найменших квадратів можна легко та швидко обчислити за допомогою стандартної статистичної обчислювальної системи.

Ключові терміни для простої лінійної регресії

- Відгук
    - Змінна, яку ми намагаємось передбачити.
    - *Синоніми:* залежна змінна, Y-змінна, ціль, результат.
- Незалежна змінна
    - Змінна, яка використовується для передбачення відгуку.
    - *Синоніми:* X-змінна, провісник, предиктор, передбачувальна змінна, ознака, атрибут.
- Запис
    - Вектор, який складається з значень провісників і значення результату для окремого елемента даних чи випадку.
    - *Синоніми:* рядок, випадок, прецедент, зразок, екземпляр, приклад.
- Перетин
    - Перетин регресійної прямої з віссю y, тобто передбачене значення, коли $X = 0$.
    - *Синоніми:* b_0, β_0, точка перетину, коефіцієнт зсуву на осі y.
- Коефіцієнт регресії
    - Нахил регресійної прямої по відношенню до осі x.
    - *Синоніми:* нахил, b_1, β_1, оцінки параметрів, ваги, кутовий коефіцієнт.
- Підігнані значення
    - Оцінки $\hat{Y}_i$, отримані з регресійної прямої.
    - *Синонім:* передбачені значення.
- Залишки
    - Різниця між значеннями, що спостерігаються, і підігнаними значеннями.
    - *Синонім:* помилки.
- Найменші квадрати
    - Метод підгонки регресії шляхом мінімізації суми квадратів залишків.
    - *Синоніми:* стандартний метод найменших квадратів, стандартний МНК.

4.4. Множинна лінійна регресія

Коли провісників кілька, рівняння регресії просто розширюється їх розміщення:

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_pX_p + e$$

Замість прямої тепер у нас лінійна модель – зв'язок між кожним коефіцієнтом та його змінною (ознакою) є лінійним.

Всі інші поняття з простої лінійної регресії, такі як підгонка найменшими квадратами, підігнані значення та залишки, відносяться і до умов множинної лінійної регресії. Наприклад, підігнані значення задаються наступною формулою:

$$\hat{Y}_i = \hat{b}_0 + \hat{b}_1X_{1,i} + \hat{b}_2X_{2,i} + \cdots + \hat{b}_pX_{p,i}$$

Ключові терміни для множинної лінійної регресії

- Формула множинної лінійної регресії:

$$Y = b_0 + b_1X_1 + b_2X_2 + \cdots + b_pX_p + e$$

- Підігнані значення:

$$\hat{Y}_i = \hat{b}_0 + \hat{b}_1X_{1,i} + \hat{b}_2X_{2,i} + \cdots + \hat{b}_pX_{p,i}$$

- Корінь із середньоквадратичної помилки:
  - Квадратний корінь із середньоквадратичної помилки регресії (це найбільш широко використовується метрика порівняння регресійних моделей).
  - *Синонім:* RMSE.
- Стандартна помилка залишків:
  - Те саме, що і середньоквадратична помилка, але скоригована для степенів свободи.
  - *Синонім:* RSE.
- R-квадрат
  - Частка дисперсії, яка пояснюється моделлю, зі значеннями в інтервалі від 0 до 1.
  - *Синоніми:* коефіцієнт детермінації, R^2.
- t-Статистика
  - Коефіцієнт для провісника, поділений на стандартну помилку коефіцієнта, що дає метрику для порівняння важливості змінних моделі
- Зважена регресія
  - Регресія, в якій записам поставлені у відповідність різні ваги

4.5. Оцінювання результативності моделі

Найважливішою метрикою результативності з погляду науки даних є *корінь із середньоквадратичної помилки* (RMSE). RMSE – це квадратний корінь із середньоквадратичної помилки в передбачених $\hat{y}_i$ значеннях:

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}.$$

Вона вимірює сукупну точність моделі і є основою її порівняння з іншими моделями (включаючи моделі, підігнані з допомогою спеціальних технічних прийомів машинного навчання). Схожою на RMSE є *стандартна помилка залишків* (RSE). У цьому випадку ми маємо p провісників, RSE задається такою формулою:

$$\text{RSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n - p - 1}}.$$

Єдина різниця полягає в тому, що знаменник є ступенями свободи, на противагу числу записів (p – кількість змінних). Насправді різниця між RMSE і RSE для лінійної регресії є дуже малою, особливо для застосунків великих даних.

Ще одна корисна метрика, яку ви побачите на виході з обчислювальних систем, називається *коефіцієнтом детермінації* або *статистикою R-квадрат* або R^2. R-квадрат варіює в інтервалі від 0 до 1 і вимірює частку варіації даних, пояснювану в моделі. Він корисний головним чином в пояснювальних застосуваннях регресії, де ви хочете визначити, наскільки добре модель підігнана до даних. Формула для R^2 така:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}.$$

Знаменник пропорційний дисперсії відгуку Y.

Оцінювання результативності моделі

- Корінь із середньоквадратичної помилки (RMSE):

$$\text{RMSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}.$$

- Стандартна помилка залишків (RSE):

$$\text{RSE} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n - p - 1}}.$$

- Коефіцієнт детермінації (статистика R^2):

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \overline{y})^2}.$$

4.6. Регресія у PySpark

Регресійний аналіз є потужним інструментом для моделювання зв'язку між залежною змінною та однією або кількома незалежними змінними. У цьому розділі ми заглибимося в деталі реалізації регресійних моделей за допомогою PySpark.

По суті, регресія передбачає оцінку параметрів математичної моделі, яка найкраще прогнозує значення безперервної результуючої змінної на основі набору вхідних даних. Лінійна регресія є, мабуть, найвідомішим типом регресії, де зв'язок між незалежними змінними та залежною змінною вважається лінійним.

PySpark використовує модель стійкого розподіленого набору даних (RDD) і DataFrame для управління даними на декількох вузлах кластера. Для задач регресії особливо корисним є модуль *pyspark.ml*, який пропонує набір інструментів для побудови, навчання та оцінки моделей машинного навчання. Основним класом, який використовується для регресії в PySpark, є *LinearRegression*, який реалізує лінійну регресію за методом найменших квадратів.

Регресія – це фундаментальна техніка в машинному навчанні, яка використовується для прогнозування безперервних результатів на основі вхідних даних. У сфері великих даних такі інструменти, як Apache Spark, а саме його Python API, відомий як PySpark, надають потужні можливості для ефективної обробки великих наборів даних. У цій главі ми розглянемо, як можна реалізувати регресію за допомогою PySpark, зосередившись на ключових поняттях, алгоритмах і практичних міркуваннях для реальних застосувань.

Щоб почати виконувати завдання регресії в PySpark, вам спочатку потрібно підготувати дані. Підготовка даних передбачає завантаження набору даних у DataFrame, обробку відсутніх значень і перетворення категоріальних ознак у числові формати. Цей крок є дуже важливим, оскільки він гарантує, що дані будуть у форматі, придатному для алгоритмів машинного навчання. Наприклад, якщо ви працюєте з файлом CSV, ви можете завантажити його за допомогою методу *spark.read.csv*, вказавши такі параметри, як *header* і *inferSchema*, для правильного аналізу даних.

Після завантаження необхідно обрати ознаки та перетворити в єдиний вектор. PySpark надає різні трансформатори в модулі *pyspark.ml.feature* для цих завдань. Наприклад, *VectorAssembler* можна використовувати для об'єднання декількох стовпців в один вектор ознак, що вимагається більшістю алгоритмів машинного навчання.

Навчання регресійної моделі передбачає її підгонку до ваших навчальних даних за допомогою методу *fit*. Цей процес обчислює коефіцієнти, які мінімізують функцію втрат, яка зазвичай є середньоквадратичною помилкою (MSE) для лінійної регресії. Після навчання модель можна використовувати для прогнозування нових даних за допомогою методу *transform*.

Оцінка продуктивності регресійної моделі має важливе значення для того, щоб переконатися, що вона відповідає бажаній точності та надійності. PySpark надає оцінювачі в модулі *pyspark.ml.evaluation*, такі як *RegressionEvaluator*, які можуть обчислювати такі метрики, як MSE, середня абсолютна похибка (MAE) та R-квадрат. Ці показники допоможуть вам зрозуміти, наскільки добре працює ваша модель, і визначити області для покращення.

На додаток до лінійної регресії, PySpark підтримує інші алгоритми регресії, включаючи регресію дерева рішень, регресію випадкового лісу та градієнтного бустінгу. Кожен з цих алгоритмів має свої сильні сторони і підходить для різних типів даних і завдань. Наприклад, регресія на основі дерева рішень може обробляти нелінійні зв'язки між ознаками та цільовою змінною, тоді як регресія на основі випадкового лісу підвищує надійність шляхом усереднення декількох дерев рішень.

При роботі з великими наборами даних важливо враховувати методи оптимізації продуктивності. PySpark дозволяє робити паралельними операції в кластері, що може значно прискорити обробку даних і навчання моделей. Такі методи, як розбиття даних на розділи, кешування фреймів даних, до яких часто звертаються, та оптимізація кількості розділів, можуть допомогти підвищити ефективність ваших регресійних задач.

Загалом, PySpark забезпечує надійну основу для реалізації регресійних моделей на великих даних. Використовуючи його потужні API та можливості розподілених обчислень, ви можете ефективно готувати, навчати та оцінювати регресійні моделі для отримання цінної інформації з великих наборів даних. Незалежно від того, чи працюєте ви з лінійною регресією, чи з більш складними алгоритмами, PySpark пропонує інструменти та гнучкість, необхідні для вирішення широкого спектру регресійних задач у реальному світі.

4.7. Навчання регресії у PySpark

Розглянемо використання інструментарію PySpark для визначення рейтингу товару за певними ознаками. Нехай початкові дані задано файлі products.csv, що надано нижче.

products.csv

```
id,f1,f2,f3,f4,f5,rating,category
1,0.8,0.9,0.7,0.6,0.4,0.85,3
2,0.9,0.5,0.8,0.7,0.3,0.75,1
3,0.7,0.6,0.5,0.4,0.8,0.65,2
```

```
4,0.6,0.7,0.9,0.5,0.1,0.55,4
5,0.3,0.4,0.6,0.8,0.9,0.6,5
6,0.5,0.8,0.7,0.9,0.2,0.75,3
7,0.4,0.1,0.3,0.6,0.7,0.45,2
8,0.9,0.6,0.5,0.7,0.4,0.75,4
9,0.1,0.3,0.4,0.8,0.6,0.45,1
10,0.7,0.2,0.5,0.9,0.3,0.55,5
11,0.6,0.4,0.8,0.5,0.1,0.45,2
12,0.8,0.7,0.3,0.6,0.9,0.65,3
13,0.5,0.9,0.4,0.1,0.7,0.55,4
14,0.2,0.8,0.6,0.3,0.5,0.45,1
15,0.9,0.5,0.7,0.4,0.2,0.65,3
16,0.7,0.6,0.8,0.9,0.1,0.75,5
17,0.4,0.3,0.5,0.2,0.6,0.35,2
18,0.6,0.7,0.9,0.8,0.3,0.75,4
19,0.3,0.1,0.4,0.5,0.7,0.45,1
20,0.5,0.2,0.6,0.9,0.8,0.55,3
```

Для побудови та навчання моделі регресії використаємо інструментарії модуля pyspark.ml з бібліотеки PySpark.

Крок 1. Імпорт необхідних залежностей.

Код

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator
```

Крок 2. Створення сесії PySpark.

Код

```
spark = SparkSession.builder.appName("LinearRegressionExample").getOrCreate()
```

Крок 3. Читання даних з файлу csv. Файл csv має знаходитись в тій же директорії, що і скрипт Python.

Код

```
df = spark.read.csv("data.csv", header=True, inferSchema=True)
```

Крок 4. Створення вектору ознак (незалежних змінних) використовуючи клас `VectorAssembler`.

Код

```
assembler = VectorAssembler(inputCols=["f1", "f2", "f3", "f4", "f5"],
outputCol="features")
```

```
assembled_df = assembler.transform(df)
```

Крок 5. Розбиття даних на тренувальну та тестову частину використовуючи функцію `randomSplit`. В прикладі нижче відкладаємо 80% даних для тренування і 20% для тестування моделі.

Код

```
train_df, test_df = assembled_df.randomSplit([0.8, 0.2])
```

Крок 6. Будуємо модель лінійної регресії та вказуємо, що вхідні ознаки збережені в `features`, а вихідні в `rating`.

Код

```
lr = LinearRegression(labelCol="rating", featuresCol="features")
```

Крок 7. Здійснюємо навчання моделі регресії.

Код

```
model = lr.fit(train_df)
```

Крок 8. Здійснюємо передбачення на тестових даних та виводимо перші 5 рядків.

Код

```
predictions = model.transform(test_df)
predictions.show(5)
```

Крок 9. Оцінюємо отримані передбачення як корінь із середньоквадратичної помилки.

Код

```
evaluator = RegressionEvaluator(labelCol="rating", predictionCol="prediction",
metricName="rmse")
rmse = evaluator.evaluate(predictions)
print(rmse)
```

4.8. Контрольні питання

1. Яка відмінність між задачами регресії та класифікації? Наведіть приклади задач кожного типу.
2. За якою формулою обчислюється результат лінійної регресії?
3. Як можна оцінити результативність моделі?
4. В яких межах може знаходитись результат метрики R^2? Яке значення є найкращим, яке найгіршим?
5. Відшукайте додаткові метрики за якими можна оцінити моделі регресії? Проаналізуйте та наведіть приклади для яких даних, які метрики є більш доречними. В яких з них краще значення є найбільшим, а в яких – найменшим?
6. Для чого використовується *VectorAssembler* в PySpark?
7. Як після навчання регресії можна застосувати її до нових, раніше невідомих даних?
8. Проведіть дослідження в яких випадках для побудови регресії використовується PySpark, а в яких більш прості прикладні пакети.

Лекція 5 – Машинне навчання в PySpark. Класифікація

Сьогодні багато галузей мають справу з дуже великими масивами даних різних типів. Обробка всієї цієї інформації вручну може зайняти багато часу і навіть не принести користі в довгостроковій перспективі. Для підвищення рентабельності інвестицій застосовується багато стратегій, від простої автоматизації до методів машинного навчання. У цьому розділі ми розглянемо одну з найважливіших концепцій – класифікацію в машинному навчанні.

Ми почнемо з визначення того, що таке класифікація в машинному навчанні, перш ніж пояснити два типи учнів у машинному навчанні та різницю між класифікацією та регресією. Потім ми розглянемо деякі реальні сценарії, в яких можна використовувати класифікацію. Після цього ми представимо всі різні типи класифікації та глибоко зануримося в деякі приклади алгоритмів класифікації.

5.1. Що таке класифікація в машинному навчанні?

Класифікація – це керований метод машинного навчання, коли модель намагається передбачити правильну мітку для заданих вхідних даних. При класифікації модель повністю навчається на навчальних даних, а потім оцінюється на тестових даних перед тим, як використовувати її для прогнозування на нових, ще не бачених даних.

Наприклад, алгоритм може навчитися передбачати, чи є даний електронний лист спамом або ні, як показано на рис. 5.1.

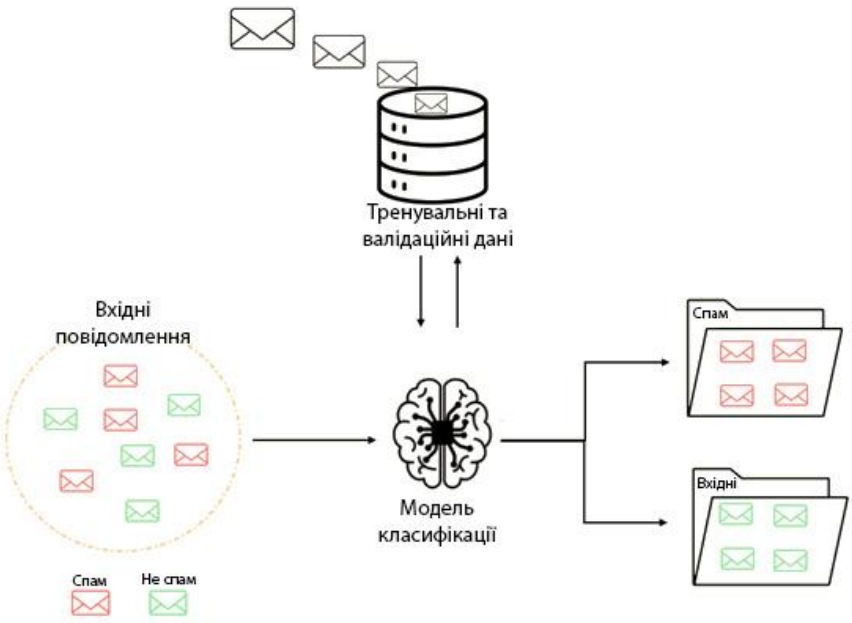

Рис. 5.1. Вирішення спам/не спам – один з прикладів задачі класифікації.

Перш ніж зануритися в концепцію класифікації, ми спочатку зрозуміємо різницю між двома типами учнів у класифікації: лінивими та охочими до навчання. Потім ми прояснимо помилкове уявлення про класифікацію та регресію.

5.2. Ліниві учні та учні, які прагнуть вчитися

У класифікації машинного навчання існує два типи учнів: ліниві та активні учні.

Активні учні – це алгоритми машинного навчання, які спочатку будують модель на основі навчального набору даних, перш ніж робити будь-які прогнози щодо майбутніх наборів даних. Вони витрачають більше часу на процес навчання через своє бажання отримати краще узагальнення під час навчання від вивчення вагових коефіцієнтів, але їм потрібно менше часу, щоб робити прогнози.

Більшість алгоритмів машинного навчання – це активні учні – алгоритми, що швидко навчаються, і нижче наведено кілька прикладів:

- логістична регресія;
- метод опорних векторів;
- дерева рішень;
- штучні нейронні мережі.

З іншого боку, ліниві учні або учні на основі прикладів не створюють жодної моделі одразу на основі навчальних даних, і саме звідси походить їхня лінивість. Вони просто запам'ятовують навчальні дані, і кожного разу, коли виникає необхідність зробити прогноз, вони шукають найближчого сусіда з усіх навчальних даних, що робить їх дуже повільними під час прогнозування. Деякі приклади такого роду:

- метод K-найближчих сусідів;
- міркування на основі конкретних випадків.

5.3. Класифікація в машинному навчанні проти регресії

Хоча класифікація і регресія відносяться до категорії керованого навчання, вони не однакові.

- Завданням прогнозування є *класифікація*, коли цільова змінна є дискретною. Прикладом може слугувати ідентифікація основної думки, що лежить в основі тексту.
- Завданням прогнозування є *регресія*, коли цільова змінна є неперервною. Прикладом може бути прогнозування заробітної плати людини, враховуючи її освіту, попередній досвід роботи, географічне розташування та рівень стажу.

5.4. Приклади класифікації машинного навчання в реальному житті

Класифікація на основі машинного навчання має різні застосування в різних сферах нашого повсякденного життя:

Охорона здоров'я. Навчання моделі машинного навчання на історичних даних пацієнтів може допомогти фахівцям у галузі охорони здоров'я точно аналізувати їхні діагнози:

- під час пандемії COVID-19 моделі машинного навчання були впроваджені для ефективного прогнозування того, чи хворіє людина на COVID-19 чи ні;
- дослідники можуть використовувати моделі машинного навчання для прогнозування нових захворювань, які з більшою ймовірністю з'являться в майбутньому.

Освіта. Освіта – одна з галузей, яка має справу з найбільшою кількістю текстових, відео- та аудіоданих. Ця неструктурована інформація може бути проаналізована за допомогою технологій природної мови для виконання різних завдань, таких як:

- класифікація документів за категоріями;
- автоматична ідентифікація коректності документів.

Транспорт. Транспорт є ключовим компонентом економічного розвитку багатьох країн. Як наслідок, галузі використовують моделі машинного та глибинного навчання:

- прогнозувати, в якій географічній точці буде спостерігатися зростання інтенсивності руху;
- прогнозувати потенційні проблеми, які можуть виникнути в певних місцях через погодні умови.

Стале сільське господарство. Сільське господарство – одна з найцінніших основ виживання людства. Впровадження сталого розвитку може допомогти підвищити продуктивність фермерів на новому рівні, не завдаючи шкоди навколишньому середовищу:

- використовуючи класифікаційні моделі, щоб передбачити, який тип землі підходить для певного типу насіння.
- прогнозувати погоду, щоб допомогти їм вжити належних превентивних заходів.

5.5. Різні типи задач класифікації в машинному навчанні

У машинному навчанні існує чотири основних типи задач класифікації: бінарна, багатокласова, класифікація з декількома мітками та незбалансована класифікація.

Бінарна класифікація. У задачі бінарної класифікації метою є класифікація вхідних даних на дві взаємовиключні категорії. Навчальні дані в такій ситуації позначаються у двійковому форматі: істина і хибність; позитивний

і негативний; 0 і 1; спам і не спам і т.д., залежно від проблеми, яку ми вирішуємо. Наприклад, нам може знадобитися визначити, чи є дане зображення вантажівкою або човном.

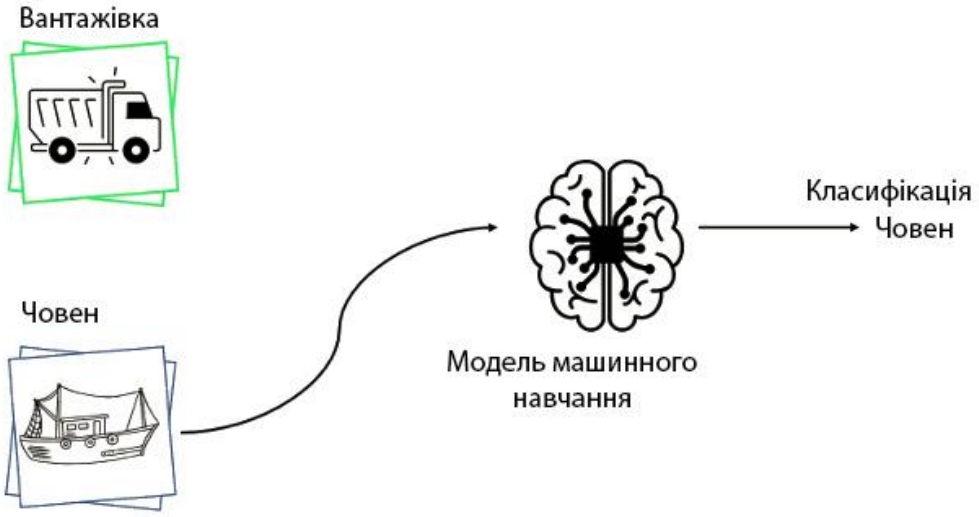

Рис. 5.2. Приклад бінарної класифікації.

Алгоритми логістичної регресії та метод опорних векторів з самого початку були призначені для вирішення задачі бінарної класифікації. Однак інші алгоритми, такі як K-найближчих сусідів та дерева рішень, також можна використовувати для бінарної класифікації.

Багатокласова класифікація. З іншого боку, багатокласова класифікація має щонайменше дві взаємовиключні мітки класів, де мета полягає в тому, щоб передбачити, до якого класу належить заданий вхідний приклад. У наступному прикладі модель правильно класифікувала зображення як літак (рис. 5.3).

Більшість алгоритмів бінарної класифікації можна також використовувати для багатокласової класифікації. Ці алгоритми включають, але не обмежуються ними:

- випадковий ліс;
- наївний Байєс;
- метод K-найближчих сусідів;
- градієнтний бустінг;
- метод опорних векторів;
- логістична регресія.

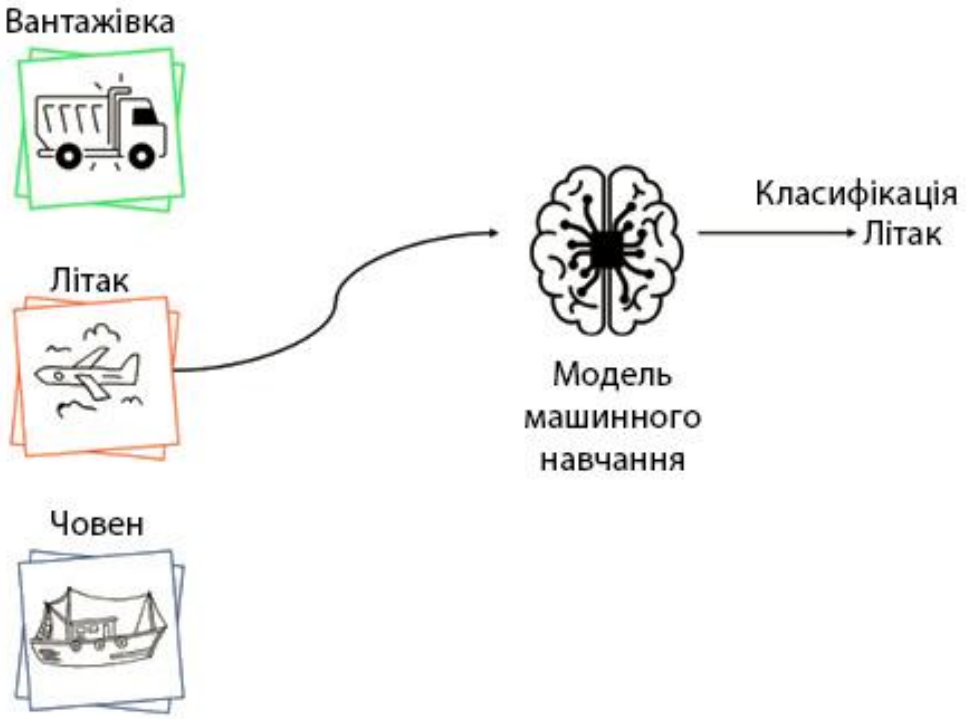

Рис. 5.3. Багатокласова класифікація.

Але зачекайте! Хіба ви не казали, що SVM та логістична регресія не підтримують багатокласову класифікацію за замовчуванням?

Так, це так. Однак ми можемо застосувати підходи бінарного перетворення, такі як «один проти одного» та «один проти всіх», щоб адаптувати власні алгоритми бінарної класифікації для задач багатокласової класифікації.

Один проти одного. Ця стратегія навчає стільки класифікаторів, скільки є пар міток. Якщо ми маємо 3-класову класифікацію, ми матимемо три пари міток, тобто три класифікатори, як показано нижче (рис. 5.4).

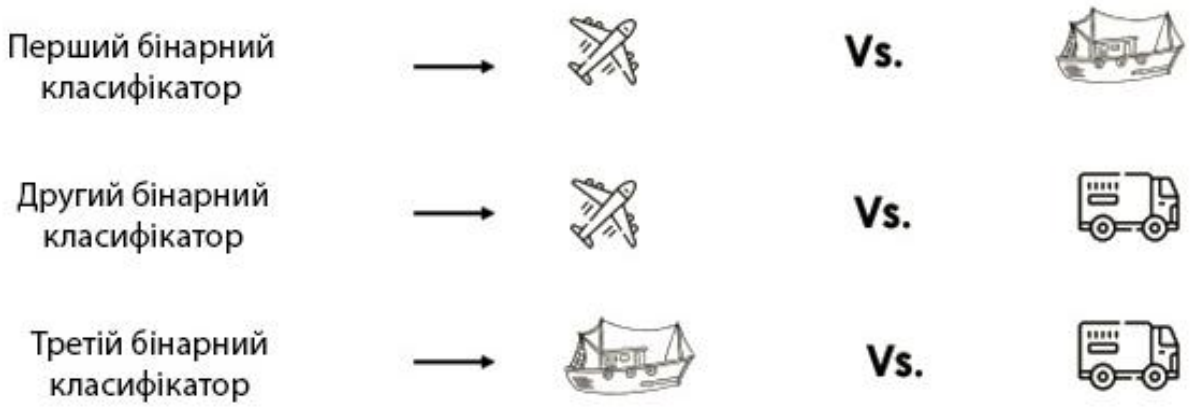

Рис. 5.4. Підхід «один проти одного».

Загалом, для N міток ми матимемо $\frac{N(N-1)}{2}$ класифікаторів. Кожен класифікатор навчається на одному двійковому наборі даних, і остаточний клас передбачається більшістю голосів між усіма класифікаторами. Підхід «один проти одного» найкраще працює для методу опорних векторів та інших ядерних алгоритмів.

Один проти решти. На цьому етапі ми починаємо з розгляду кожної мітки як незалежної мітки, а решту розглядаємо разом як одну мітку. З 3-ма класами ми матимемо три класифікатори.

Загалом, для N міток ми матимемо N бінарних класифікаторів (рис. 5.5).

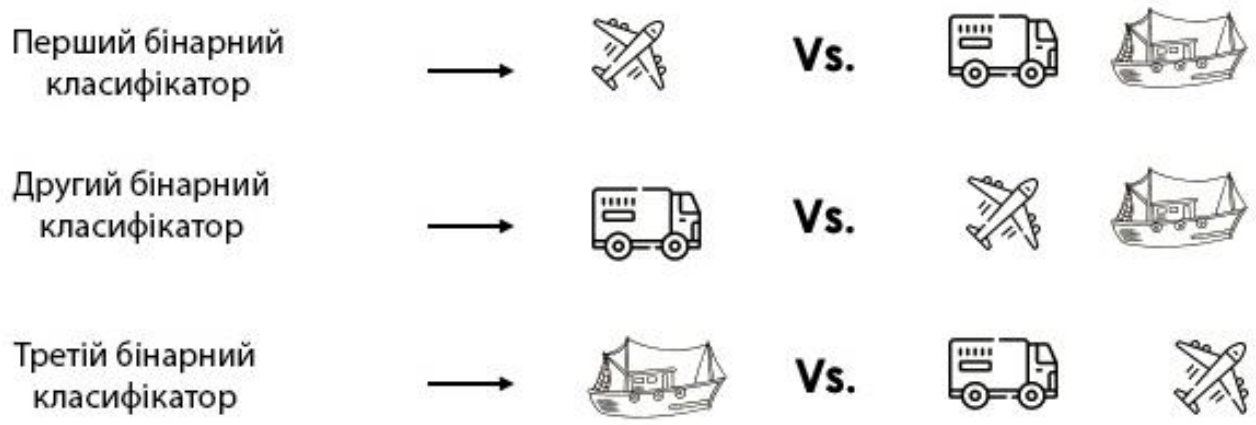

Рис. 5.5. Підхід «один проти решти».

Класифікація з декількома мітками. У задачах класифікації з декількома мітками ми намагаємося передбачити 0 або більше класів для кожного вхідного прикладу. У цьому випадку немає взаємного виключення, оскільки вхідний приклад може мати більше однієї мітки.

Такий сценарій можна спостерігати в різних сферах, наприклад, при автоматичному тегуванні в обробці природної мови, де заданий текст може містити кілька тем. Подібно до комп'ютерного зору, зображення може містити кілька об'єктів, як показано нижче: модель передбачила, що зображення містить літак, човен, вантажівку і собаку (рис. 5.6).

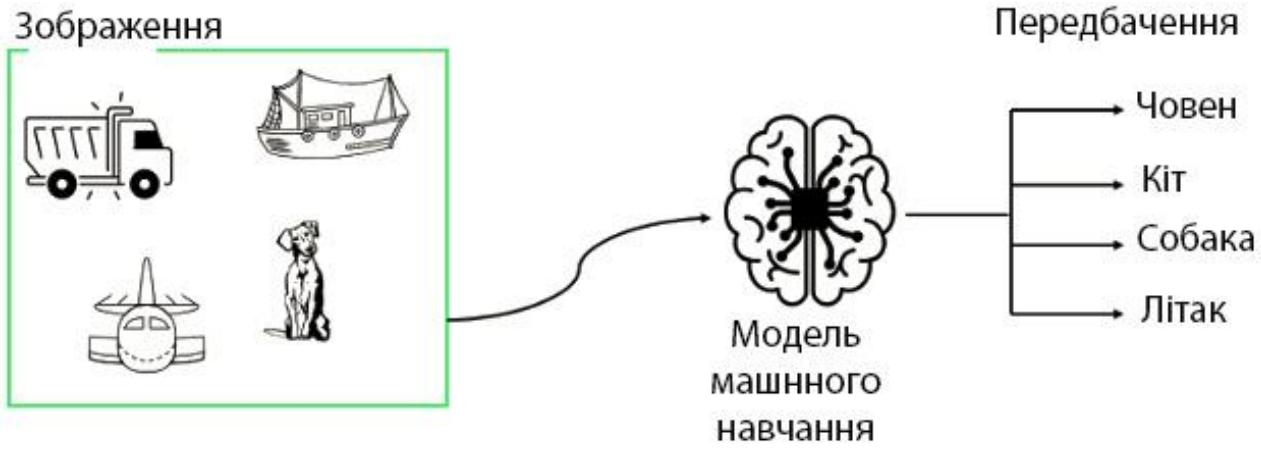

Рис. 5.6. Класифікація з декількома мітками.

Неможливо використовувати багатокласові або бінарні моделі класифікації для виконання класифікації з декількома мітками. Однак більшість алгоритмів, що використовуються для стандартних задач класифікації, мають свої спеціалізовані версії для класифікації з декількома мітками. Наведемо декілька прикладів:

- дерева рішень з декількома мітками;
- багатомітоковий градієнтний бустинг;
- багатоміточні випадкові ліси.

Незбалансована класифікація. Для незбалансованої класифікації кількість прикладів нерівномірно розподілена в кожному класі, що означає, що ми можемо мати більше прикладів одного класу, ніж інших у навчальних даних. Розглянемо наступний сценарій 3-класової класифікації, де навчальні дані містять 60% вантажівок, 25% літаків і 15% човнів.

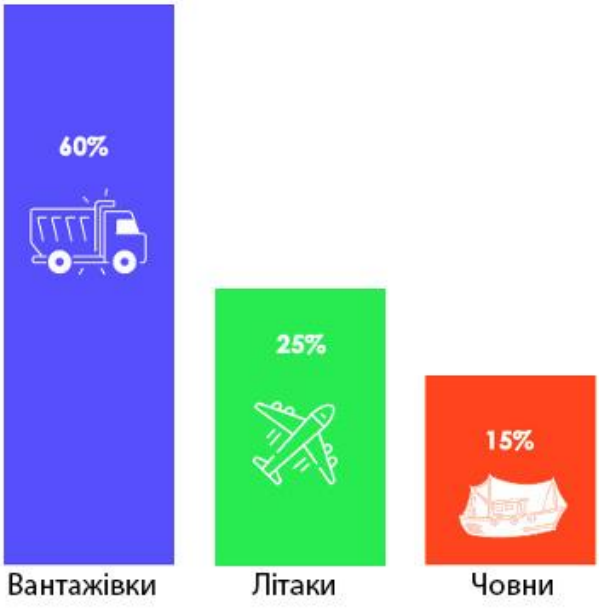

Рис. 5.7. Незбалансована класифікація.

Проблема незбалансованої класифікації може виникнути за наступним сценарієм:

- виявлення шахрайських транзакцій у фінансових галузях;
- діагностика рідкісних захворювань;
- аналіз відтоку клієнтів.

Використання звичайних моделей прогнозування, таких як дерева рішень, логістична регресія тощо, не може бути ефективним при роботі з незбалансованим набором даних, оскільки вони можуть бути упередженими до прогнозування класу з найбільшою кількістю спостережень, а ті, що мають меншу кількість спостережень, вважатимуться шумом.

Чи означає це, що такі проблеми не мають рішення?

Звісно, ні! Ми можемо використовувати кілька підходів для вирішення проблеми дисбалансу в наборі даних. Найпоширеніші з них включають методи вибірки або використання можливостей алгоритмів, чутливих до вартості.

Методи вибірки. Ці методи мають на меті збалансувати розподіл вихідних даних:

- перевибірка на основі кластерів;
- випадкова неповна вибірка: випадкове вилучення прикладів з класу більшості;
- перевибірка SMOTE: випадкове повторення прикладів з класу меншості.

Алгоритми, чутливі до вартості. Ці алгоритми враховують вартість помилкової класифікації. Вони спрямовані на мінімізацію загальних витрат, що генеруються моделями.

- дерева рішень з урахуванням вартості;
- логістична регресія з урахуванням вартості;
- вартісно-чутливий метод опорних векторів.

Розглянемо більш детально деякі популярні підходи класифікація, а саме дерева рішень та випадкових ліс.

5.6. Дерева рішень

Дерево рішень передбачає значення цільової змінної за допомогою застосування послідовності простих розв'язувальних правил (які називаються предикатами). Цей процес у певному сенсі узгоджується з природним для людини процесом ухвалення рішень.

Хоча узагальнювальна здатність дерев, що вирішують, невисока, їхні передбачення обчислюються доволі просто, через що дерева рішень часто використовують як цеглинки для побудови ансамблів – моделей, що роблять передбачення на основі агрегації передбачень інших моделей. Про них ми поговоримо далі.

Приклад вирішального дерева. Почнемо з невеликого прикладу. На рис. 5.8 зображено дерево, побудоване для задачі класифікації на п'ять класів.

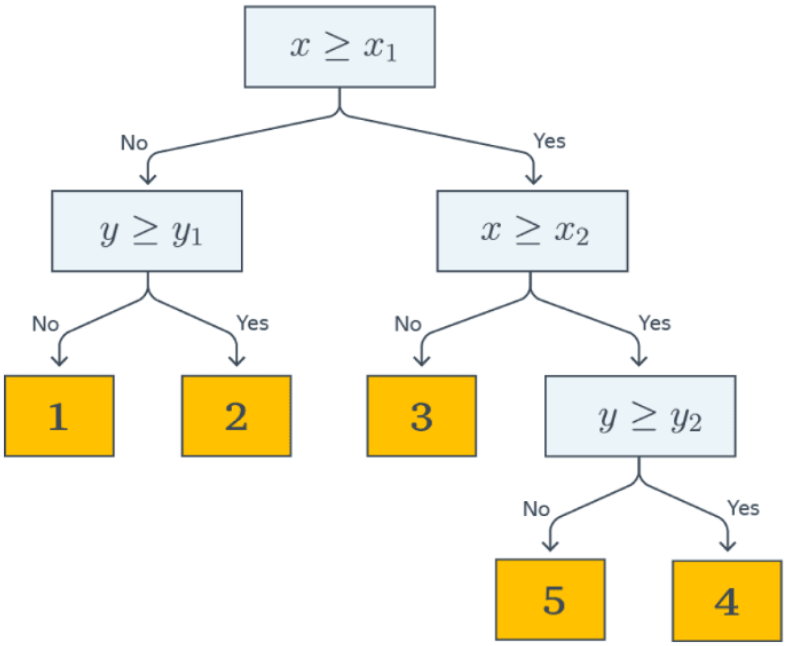

Рис. 5.8. Приклад дерева рішень.

Об'єкти в цьому прикладі мають дві ознаки з дійсними значеннями: X і Y. Рішення про те, до якого класу буде віднесено поточний об'єкт вибірки, ухвалюватиметься за допомогою проходу від кореня дерева до деякого листа.

У кожному вузлі цього дерева міститься предикат. Якщо предикат вірний для поточного прикладу з вибірки, ми переходимо до правого нащадка, якщо ні – до лівого. У цьому прикладі всі предикати – це просто взяття порога за значенням якоїсь ознаки:

$$B(x, j, t) = [x_j \leq t] \tag{5.1}$$

У листках записані передбачення (наприклад, мітки класів). Щойно ми дійшли до листка, ми присвоюємо об'єкту відповідь, записану у вершині.

На рис. 5.9 візуалізовано процес побудови розв'язувальних поверхонь, породжуваних деревом (права частина картинки).

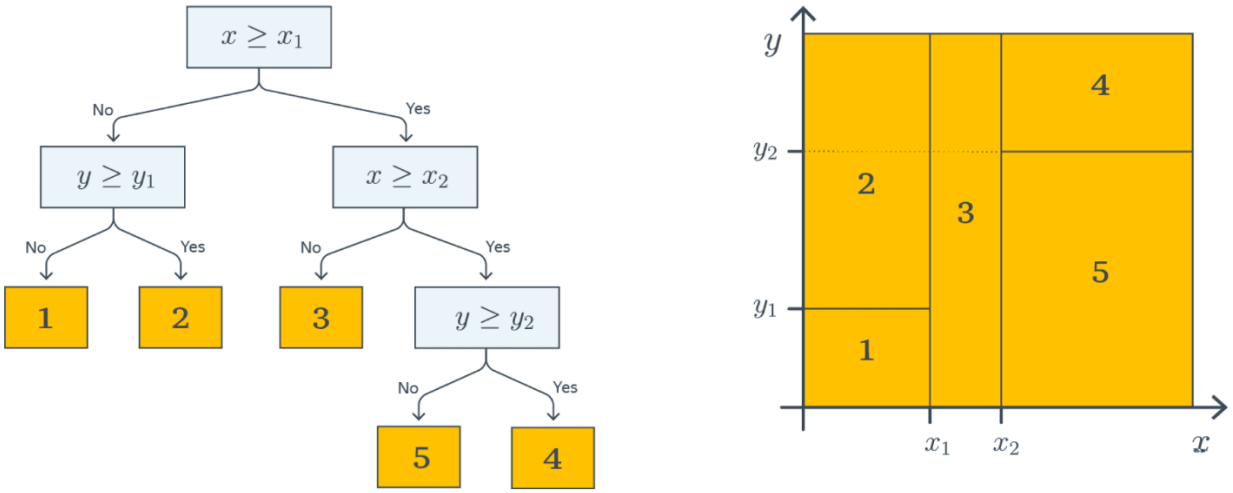

Рис. 5.9. Процес побудови розв’язувальних поверхонь.

Кожен предикат породжує поділ поточної підмножини простору ознак на дві частини. На першому етапі, коли відбувався поділ за $[X \leq X_1]$, всю площину було поділено на дві відповідні частини. На наступному рівні частина площини, для якої виконується $[X \leq X_1]$, була поділена на дві частини за значенням другої ознаки $Y \leq Y_1$ – так утворилися області 1 і 2. Те саме повторюється для правої частини дерева – і так далі до листових вершин: вийде п'ять областей на площині. Тепер будь-якому об'єкту вибірки буде присвоюватися один із п'яти класів залежно від того, в яку з утворених областей він потрапляє. Цей приклад добре демонструє, зокрема, те, що дерево здійснює кусково-постійну апроксимацію цільової залежності.

5.7. Випадковий ліс

Передбачення одного дерева є не надто точним. Для підвищення якості окремі дерева рішень можна об’єднати в так званий «випадковий ліс». Для об’єднання достатньо, щоб окремі дерева рішень були певною мірою не схожі один на одного.

Побудуємо ансамбль алгоритмів, де базовий алгоритм – це дерево рішень. Будемо будувати за такою схемою:

1. Для побудови i-го дерева:
    a. Спочатку з навчальної вибірки X вибирають із поверненням випадкову підвибірку X^i такого самого розміру, що й X.
    b. У процесі навчання кожного дерева в кожній вершині випадково вибирають $n < N$ ознак, де N – повне число ознак (метод випадкових підпросторів), і серед них шукають оптимальне розбиття. Такий прийом якраз дає змогу керувати ступенем скорельованості базових алгоритмів.

2. Щоб отримати передбачення ансамблю на тестовому об'єкті, усереднюємо окремі відповіді дерев (для регресії) або беремо найпопулярніший клас (для класифікації).
3. Ми побудували випадковий ліс (Random Forest), що узагальнює передбачення окремих дерев рішень.

У PySpark дерева рішень можна реалізувати за допомогою класифікатора *DecisionTreeClassifier* з модуля *pyspark.ml.classification*. Цей класифікатор будує єдине дерево рішень, вибираючи найкращі розбиття для максимізації інформативності. Процес побудови дерева рішень включає оцінку різних атрибутів та їхніх потенційних точок розбиття, щоб знайти найефективніший спосіб розділити дані на класи.

У PySpark випадковий ліс можна реалізувати за допомогою класифікатора *RandomForestClassifier* з модуля *pyspark.ml.classification*. Цей класифікатор будує кілька дерев рішень і комбінує їхні прогнози для прийняття остаточного рішення.

Однією з важливих переваг використання PySpark для цих алгоритмів є його здатність ефективно обробляти великі набори даних, використовуючи розподілені обчислення. DataFrame API PySpark надає високорівневий інтерфейс для маніпулювання даними, що полегшує підготовку та попередню обробку даних перед навчанням моделей. Крім того, PySpark легко інтегрується з іншими компонентами Spark, такими як MLlib (бібліотека машинного навчання), яка пропонує широкий спектр алгоритмів та інструментів для побудови та оцінки моделей машинного навчання.

Працюючи з деревами рішень і випадковими лісами в PySpark, важливо враховувати компроміс між складністю моделі та її інтерпретованістю. Дерева рішень відносно прості і зрозумілі, що робить їх корисними для пояснення прогнозів нам. Однак вони можуть бути схильні до перенавчання. Випадкові ліси, з іншого боку, пропонують краще узагальнення завдяки своїй ансамблевій природі, але можуть пожертвувати певною інтерпретованістю.

Дерева рішень і випадкові ліси є потужними алгоритмами класифікації, які можна ефективно реалізувати за допомогою PySpark. Використовуючи можливості розподілених обчислень Spark, ці моделі можуть ефективно працювати з великими наборами даних, забезпечуючи при цьому надійні та інтерпретовані результати.

5.8. Контрольні питання

1. Які підходи класифікації просто запам’ятовують навчальну вибірку? Подумайте, для яких даних такий підхід є доречним.
2. Проаналізуйте приклади задач класифікації, наведені в пункті 5.4. Які інші задачі з вашого досвіду можна було б вирішити в аналогічний спосіб?
3. Яка відмінність між бінарною та багатокласовою класифікацією? Чи однаково застосовними є різні методи для кожного типу класифікації?
4. Що таке підходи «один про одного», «один проти решти»? Проведіть дослідження для яких методів класифікації використання даних є необхідним, а які можуть відрізу вирішувати задачу багатокласової класифікації?
5. Які проблеми виникають, коли вибірка даних є суттєво незбалансованою за кількістю прикладів в кожному з класів? Які способи вирішення такої проблеми існують?
6. Застосуйте дерево рішень до певної задачі. Проаналізуйте отриману якість. Чи є результат задовільним? Які способи покращення якості ви бачите?
7. Наведіть переваги та недоліки використання окремого дерева рішень проти випадкового лісу.
8. За допомогою яких класів PySpark реалізуються дерева рішень та випадкових ліс? Які переваги використання PySpark над іншими програмними пакетами?

Лекція 6 – Нереляційні БД. Переваги та недоліки над реляційними БД

Очікується, що у 2024 році обсяг ринку великих даних вперше в історії досягне 84 мільярдів доларів. Враховуючи, що щодня станом на 2024-й рік у світі генерується приблизно 2,5 квінтильйона байт даних, таке зростання ринку є цілком логічним – і є одним з рушійних факторів експоненціального розвитку та використання систем баз даних.

Два найпоширеніші типи баз даних – це бази даних SQL (наприклад, реляційні бази даних) та бази даних NoSQL. Далі буде описано бази даних SQL та NoSQL, те, як вони працюють, поширені випадки використання, а також переваги та недоліки кожного типу баз даних.

6.1. Мова структурованих запитів (SQL)

Що таке SQL? SQL, що розшифровується як Structured Query Language (мова структурованих запитів), – це мова програмування, орієнтована на конкретну задачу або проблему, яка зазвичай використовується для таких завдань, як вставка, оновлення, запит і видалення даних в базі даних. SQL також використовується для створення та модифікації схем бази даних (наприклад, правил форматування даних, структури таблиць/індексів), а також для визначення параметрів доступу до бази даних та адміністрування.

Що таке структуровані дані? Структуровані дані – це дані, які організовані в послідовному, заздалегідь визначеному форматі і часто складаються з алфавітно-цифрових символів. Прикладами таких даних є фінансові транзакції, інвентаризаційні записи або списки клієнтів, які часто зберігаються в базах даних SQL (наприклад, реляційних базах даних).

Що таке база даних SQL? Коли використовується термін «база даних SQL», це означає тип бази даних, де SQL є основною мовою програмування, що використовується для створення та управління цією базою даних. Інтерфейси прикладного програмування (API) SQL містять групи функцій, які дозволяють розробникам виконувати операції з базами даних і керувати ними без необхідності створювати окремі команди SQL знову і знову.

Незалежно від того, чи використовується база даних SQL для зберігання транзакцій для роздрібної торгівлі або фінансової інформації для корпорації, бази даних SQL відносяться до типу баз даних, які називаються реляційними базами даних.

Реляційні бази даних. Реляційні бази даних, або системи керування реляційними базами даних (СКБД), зберігають дані в рядках і стовпцях, які використовуються для формування таблиць. Зв'язок між двома таблицями (або більше) можна створити за допомогою зовнішнього ключа. Ці зовнішні ключі (наприклад, унікальні ідентифікатори) підтримують заздалегідь визначені зв'язки, які існують між таблицями.

Приклад: реляційна база даних електронної комерції, що містить інформацію про клієнтів, товари та замовлення (рис. 6.1).

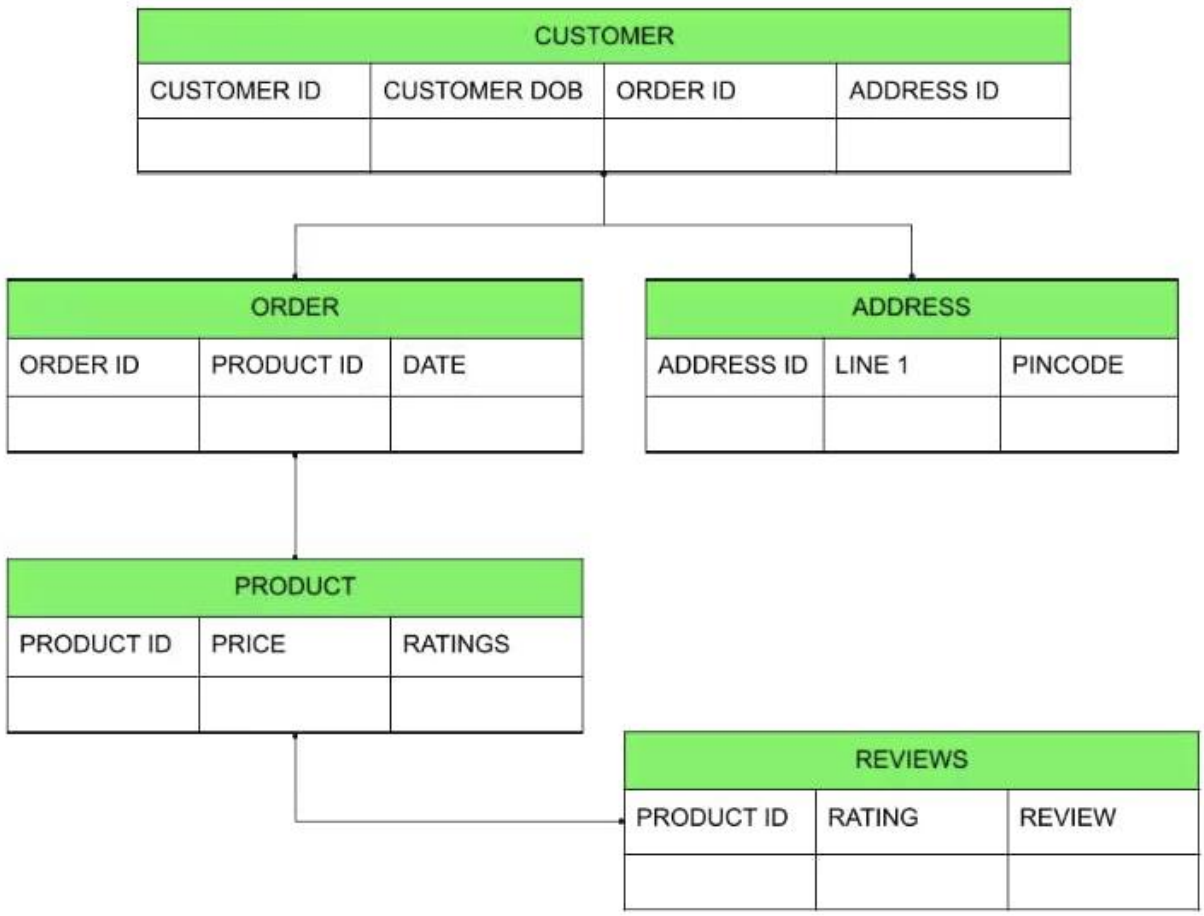

Рис. 6.1. Приклад реляційної бази даних електронної комерції.

Важливо зазначити, що реляційні бази даних створюються та управляються за допомогою фіксованої схеми. Фіксована схема означає, що всі дані, які потрапляють до бази даних, повинні бути точно приведені у відповідність до визначених стандартів форматування, що обмежує типи структур даних, які можуть зберігатися в реляційних базах даних. Наприклад, реляційні бази даних не здатні обробляти неструктуровані дані (наприклад, інформацію, яка має різний формат і не відповідає заданій моделі даних), але чудово підтримують транзакційну або фінансову інформацію, яка включає структуровані дані або напівструктуровані типи даних (наприклад, дані, які мають однаковий формат і відповідають заданій моделі даних).

Приклади баз даних SQL. Існує безліч різних прикладів баз даних SQL, зокрема:

- Oracle Database – це реляційна система управління базами даних (СКБД), розроблена і продається корпорацією Oracle і є однією з найпоширеніших корпоративних систем баз даних у світі.
- MySQL – це загальновживана реляційна система управління базами даних з відкритим вихідним кодом, яка використовується для створення та адміністрування баз даних. Розроблена і поширювана корпорацією Oracle, MySQL відома простотою використання, широкою підтримкою спільноти та надійністю.
- Microsoft SQL Server є реляційною системою управління базами даних, розробленою компанією Microsoft. Ця платформа баз даних зазвичай використовується у великих корпоративних середовищах для підтримки великих обсягів обробки транзакцій, бізнес-аналітики та аналітичних додатків.

Важливо зазначити, що інші типи баз даних також можуть встановлювати зв'язки між частинами даних. У випадку нормалізованих табличних баз даних (наприклад, SQL або реляційних баз даних) ці зв'язки виражаються за допомогою зовнішніх ключів або таблиць перетину. У випадку систем керування базами даних (СКБД), таких як MongoDB (база даних NoSQL), ці зв'язки встановлюються шляхом вбудовування або посилання на дані.

6.2. Не тільки мова структурованих запитів (NoSQL)

Що таке NoSQL? NoSQL, що розшифровується як «не тільки SQL», – це підхід до системи управління базами даних, який використовується для прийому, зберігання та вилучення неструктурованих і напівструктурованих даних у базі даних. Це означає, що дані, які не можуть бути проаналізовані або підраховані за допомогою традиційних реляційних баз даних (наприклад, SQL), можуть залишатися у своєму рідному форматі і бути перенесені в базу даних NoSQL. Причина, по якій вони називаються NoSQL, полягає в тому, щоб підкреслити, що ці бази даних можуть працювати з нетабличними, нереляційними моделями даних, а також підтримувати SQL-подібні мови запитів.

Що таке неструктуровані дані? Неструктуровані дані – це дані, які не мають наперед визначеної моделі даних або послідовної організації. Крім того, неструктуровані дані, такі як пости в соціальних мережах, можуть швидко оновлюватися і змінюватися, в той час як структуровані дані, такі як банківські транзакції, мають набагато меншу швидкість змін. Прикладами неструктурованих даних є зображення, аудіофайли, відео та карти.

Що таке база даних NoSQL? Бази даних NoSQL – це бази даних, які використовують гнучку схему, що вміщує неструктуровані та напівструктуровані дані, а також використовують нетабличний метод зберігання даних.

Використання гнучкої схеми дозволяє базам даних NoSQL приймати неструктуровані дані у їхньому природному форматі (наприклад, .txt, .JPG, MP3), що неможливо для баз даних SQL через вимогу, щоб усі дані відповідали заздалегідь визначеному формату. Крім того, коли бази даних NoSQL зберігають дані, використовуються гнучкі моделі даних, так що неструктуровані файли даних можуть мати різні структури даних, але зберігатися в одній колекції.

6.3. Типи баз даних NoSQL

Існують різні типи баз даних NoSQL, зокрема:

- **Документні бази даних.** Документні бази даних, які іноді називають об'єктно-орієнтованими базами даних, зберігають дані в документах, схожих на об'єкти JSON (JavaScript Object Notation), хоча вони не є сховищами JSON. Вони використовують драйвери, що повертаються від нативних об'єктів до мови програмування, яку використовує розробник, не потребуючи об'єктно-реляційного маппера (ORM). Кожен документ сам по собі розглядається як запис і може містити значення, включаючи числа, масиви, об'єкти, рядки або навіть логічні символи. Крім того, можуть бути включені пари ключ-значення, вкладені документи або інші структуровані дані. Популярним постачальником цих баз даних NoSQL є MongoDB.
- **Бази даних «ключ-значення».** Бази даних «ключ-значення» збирають, отримують і зберігають дані у вигляді груп пар «ключ-значення». Це означає, що кожен запис даних представлений унікальним ключем і пов'язаним з ним значенням. Ключ використовується для отримання відповідного значення з бази даних. Наприклад, у базі даних про дизайн інтер'єру ключ-значення може бути «колір», а значення – «фіолетовий». Популярними постачальниками цих систем баз даних NoSQL є AWS та ScyllaDB.
- **Колонкові бази даних.** Колонкові бази даних організовують дані в стовпці, а не в рядки, що корисно при роботі з великими наборами даних, які розріджені в глибину. Насправді, сховища колонкових баз даних іноді називають «сховищами з широкими стовпцями». У сховищах сімейств стовпців кожен рядок має свій набір стовпців, а стовпці потім збираються в «сім'ї». Ці моделі даних корисні при роботі з великими наборами даних, які виграють від горизонтального масштабування для оптимізації продуктивності. Популярними постачальниками цих баз даних NoSQL є Apache Cassandra та HBase.
- **Графові бази даних.** Графові бази даних зберігають дані у вузлах і ребрах. Вузли зазвичай зберігають інформацію про людей, місця та речі, тоді як

ребра зберігають інформацію про зв'язки між вузлами. Графові бази даних є чудовим інструментом для запитів до графових структур (наприклад, соціальних мереж, ієрархій). До популярних постачальників цих баз даних NoSQL належать Neo4j, AWS та Kibana.

6.4. Ключові відмінності між базами даних SQL та NoSQL

Хоча і SQL, і NoSQL бази даних пропонують цінну функціональність, важливо розуміти ключові відмінності між ними.

Модель зберігання бази даних. Різниця між системами баз даних SQL та NoSQL, пов'язана зі зберіганням даних, є разючою. Зокрема, бази даних SQL зберігають дані в таблицях, що містять рядки і стовпці, тоді як системи NoSQL зберігають дані різними методами залежно від типу неструктурованих даних (наприклад, документи JSON, пари ключ-значення, групування сімейств, вузли/ребра графа).

Тип даних. У той час як бази даних NoSQL, які іноді називають нереляційними базами даних, здатні приймати, зберігати та отримувати неструктуровані дані, бази даних SQL (традиційні реляційні бази даних) не можуть цього робити. Бази даних SQL здатні приймати, зберігати та отримувати тільки структуровані дані. Це пов'язано з різницею між використовуваними схемами SQL та NoSQL.

Схеми. Бази даних SQL покладаються на сувору, заздалегідь визначену схему даних, з якою повинні узгоджуватися дані, що поглинаються. Натомість бази даних NoSQL використовують гнучкі схеми, які дозволяють їм приймати дані в різних власних форматах.

6.4.1. Масштабованість

Для адміністраторів баз даних важливо планувати зростання і розширення своїх систем баз даних – це ще одна чітка відмінність між SQL і NoSQL базами даних.

Бази даних SQL. Бази даних SQL традиційно масштабуються вертикально. Це означає, що ресурси (наприклад, процесори, сховище або пам'ять) додаються до одного сервера. І хоча це може призвести до обмеження можливого зростання, оскільки масштабується лише один сервер з обмеженою потужністю, є кілька причин для такого вибору масштабування:

- Відповідність стандартам атомарності, узгодженості, ізоляції та витривалості (ACID). Відповідність ACID відноситься до набору властивостей, які гарантують надійність, узгодженість і цілісність даних транзакцій бази даних. Це дуже важливо, оскільки багато баз даних SQL містять банківську та фінансову інформацію, яка повинна відповідати державним та галузевим стандартам. Підтримувати відповідність ACID складніше у розподіленій системі (наприклад, багато комп'ютерів,

пов'язаних мережею), де ресурси збільшуються за рахунок горизонтального масштабування проти одного комп'ютера і одного сервера, які масштабуються вертикально.

- Управління транзакціями. Механізми управління транзакціями використовуються в базах даних SQL для підтримки цілісності даних і узгодженості бази даних. Керування одночасними транзакціями на декількох вузлах у середовищі розподіленої бази даних, ймовірно, створить додаткову складність і використання ресурсів, що може вплинути як на цілісність даних, так і на загальну продуктивність бази даних. Якщо використовувати горизонтальне масштабування, ці проблеми можуть стати можливими.
- Жорсткість схеми. Бази даних SQL використовують жорсткі, заздалегідь визначені схеми, за допомогою яких вони отримують дані. Хоча це легко підтримувати в середовищі з одним комп'ютером/сервером, складність додасться, якщо використовувати розподілену систему з горизонтальним масштабуванням. Зокрема, кожен вузол потенційно може мати свою версію схеми, що збільшить витрати на адміністрування і може спричинити додаткові проблеми з узгодженістю даних.

Бази даних NoSQL. Системи баз даних NoSQL часто конфігуруються у так званій розподіленій системі. Це означає, що кілька незалежних комп'ютерів (наприклад, вузлів) пов'язані через мережу і працюють разом для досягнення спільних цілей. Приналежність до розподіленої системи також означає, що можна використовувати горизонтальне та вертикальне масштабування.

Горизонтальне масштабування передбачає збільшення доступних ресурсів і потужності розподіленої системи шляхом додавання до неї нових вузлів (наприклад, комп'ютерів, серверів). Таким чином, більша кількість вузлів стає доступною для підтримки робочого навантаження системи. Крім того, немає практично ніяких обмежень на те, наскільки велика база даних може зростати з точки зору ємності, оскільки можна продовжувати додавати додаткові вузли.

6.5. Варіанти використання технологій баз даних SQL та NoSQL

У той час як бази даних SQL відмінно справляються з управлінням структурованими реляційними даними, зберігаючи цілісність транзакцій і виконуючи складні запити, бази даних NoSQL неперевершені в запитах і зберіганні неструктурованих або напівструктурованих даних, забезпечуючи при цьому необмежену масштабованість і гнучкість.

З огляду на це, рушійною силою у виборі між SQL та NoSQL базами даних (наприклад, між реляційними та нереляційними базами даних) є сценарії використання. Ось короткий огляд переваг та відповідних варіантів використання баз даних SQL та NoSQL.

6.5.1. Варіанти використання баз даних SQL

Ключові характеристики баз даних SQL підходять для наступних варіантів використання:

1. **Відповідність нормативним вимогам.** Оскільки структура баз даних SQL відповідає вимогам ACID, їх часто використовують для зберігання даних, які повинні відповідати певним державним або галузевим стандартам.
2. **Транзакційні бази даних.** Транзакційні бази даних зберігають дані, які є результатом взаємодії між двома або більше сторонами.
   Приклади:
   - бази даних точок продажу роздрібної торгівлі;
   - медичні бази даних рецептів і замовлень;
   - комерційні банки;
   - бази даних бухгалтерського та фінансового обліку.
3. **Системи планування ресурсів підприємства (ERP).** ERP-системи допомагають підприємствам керувати процесами, які є ключовими для операційної діяльності, управління персоналом, виробництва тощо.
   Приклади:
   - бази даних людських ресурсів;
   - системи управління ланцюгами поставок;
   - системи управління ризиками.

6.5.2. Випадки використання баз даних NoSQL

Хоча існує поширена думка, що NoSQL (наприклад, нереляційні) бази даних не є ACID-сумісними, деякі з них насправді є такими. MongoDB є яскравим прикладом NoSQL-сумісної ACID-бази даних.

Маючи це на увазі, ось кілька прикладів найпоширеніших випадків використання NoSQL-баз даних:

1. **Транзакційні бази даних.** Транзакційні бази даних також можуть підтримуватися базами даних NoSQL, оскільки вони використовуються для зберігання неструктурованих даних, які є результатом взаємодії між двома або більше сторонами.
   Приклади:
   - медичні картки пацієнтів, які потребують можливостей нереляційної бази даних (наприклад, історії хвороби, рентгенівські/скановані фотографії та відео);
   - страхові справи (наприклад, фотографії з автокатастроф, документація про травми);
   - бази даних юридичних документів (наприклад, свідчення, заяви, матеріали судових справ).

2. **Бази даних документів та управління цифровими активами.** Бази даних документів та управління цифровими активами зберігають і керують документами, зображеннями, мультимедійним контентом, відео тощо.
   Приклади:
   - онлайн-бібліотеки (наприклад, юридичні бібліотеки, онлайн-бібліотека Конгресу);
   - цифрові видавничі платформи, такі як Kindle або Nook;
   - потокові медіа-сервіси, такі як Netflix і Hulu;
   - онлайн-платформи для обміну фотографіями, такі як Instagram або Meta.
3. **Аналіз графів та мереж.** Графові та мережеві бази даних чудово підходять для управління такими структурами даних, як системи рекомендацій, соціальні мережі та пов'язаний з ними мережевий аналіз, оскільки вони здатні виявляти та аналізувати неінтуїтивно зрозумілі взаємозв'язки між взаємопов'язаними елементами даних.
   Приклади:
   - аналіз соціальних мереж (наприклад, метрики постів і користувачів);
   - виявлення шахрайства, яке ізолює незвичні транзакції або інші аномалії;
   - графи знань (наприклад, квадранти продуктів/послуг Gartner).
4. **Платформи Інтернету речей (IoT).** Платформи IoT часто використовуються для зберігання та аналізу даних датчиків і метаданих пристроїв у режимі реального часу.
   Приклади:
   - системи розумного дому;
   - системи «розумного міста» (наприклад, робота світлофорів);
   - збір метеорологічної інформації.

6.6. Контрольні питання

1. Проведіть порівняльний аналіз реляційних та нереляційних баз даних. Чи можна визначити тип баз даних, що є строго кращим за інших?
2. Яким чином інформація про товари та відгуки зберігалася б в реляційних та нереляційних БД?
3. Які є типи нереляційних баз даних? До якого з них відноситься MongoDB?
4. Які переваги та недоліки має MongoDB у порівнянні з іншими нереляційними БД?
5. Яким чином зазвичай відбувається масштабування нереляційних БД?
6. Розгляньте розділ 6.5. Які інші приклади використання баз даних ви можете навести? Який тип БД для кожної задачі ви би використали?

Лекція 7 – MongoDB – відкрита документна нереляційна БД

MongoDB – це потужна, гнучка і масштабована система управління базами даних (СУБД) загального призначення. Вона поєднує в собі можливість масштабування з такими функціями, як вторинні індекси, запити за діапазоном, сортування, агрегування та геопросторові індекси. У цьому розділі розглядаються основні проектні рішення, які зробили MongoDB тим, чим вона є.

Простота використання. MongoDB – це нереляційна, а документоорієнтована система управління базами даних. Основною причиною відмови від реляційної моделі є спрощення горизонтального масштабування, але є й інші переваги.

Документоорієнтована СУБД замінює концепцію «рядка» більш гнучкою моделлю, «документом». Дозволяючи використовувати вкладені документи і масиви, документоорієнтований підхід дає можливість представляти складні ієрархічні відносини за допомогою одного запису. Це природним чином вписується в те, як розробники, що працюють із сучасними об'єктно-орієнтованими мовами, розглядають свої дані.

Також немає зумовлених схем: ключі та значення документа не мають фіксованих типів або розмірів. Коли немає фіксованої схеми, додавати або видаляти поля в міру необхідності стає простіше. Як правило, це прискорює розробку, оскільки розробники можуть швидко виконувати ітерації. Експериментувати також простіше. Розробники можуть випробувати десятки моделей для даних, а потім вибрати найкращу.

Розроблено для масштабування. Розміри наборів даних для додатків зростають неймовірними темпами. Збільшення доступної пропускної здатності та дешеві сховища створили середовище, у якому навіть невеликим застосункам необхідно зберігати більше даних, ніж здатні обробити багато баз даних. Терабайт даних, колись нечуваний обсяг інформації, тепер став звичайним явищем.

У міру зростання обсягу даних, які необхідно зберігати розробникам, останні стикаються з важким рішенням: як масштабувати свої бази даних? Масштабування бази даних зводиться до вибору між вертикальним масштабуванням (отримання більшої машини) і горизонтальним масштабуванням (партиціонування даних на декількох машинах). Вертикальне масштабування часто є шляхом найменшого опору, але в нього є свої недоліки: більші машини нерідко дуже дорогі, і зрештою досягається фізична межа, коли потужнішу машину не можна купити за будь-яку ціну. Альтернативою є горизонтальне масштабування: додати місце для зберігання або збільшити пропускну спроможність для операцій читання та запису, придбати додаткові сервери та додати їх у свій кластер. Це і дешевше, і більш масштабовано; однак адмініструвати тисячу машин складніше, ніж піклуватися про одну.

MongoDB була розроблена для горизонтального масштабування. Документоорієнтована модель даних полегшує розподіл даних між кількома

серверами. MongoDB автоматично піклується про балансування даних і навантаження в кластері, автоматично перерозподіляючи документи і спрямовуючи операції читання і запису в потрібні машини, як показано на рис. 7.1.

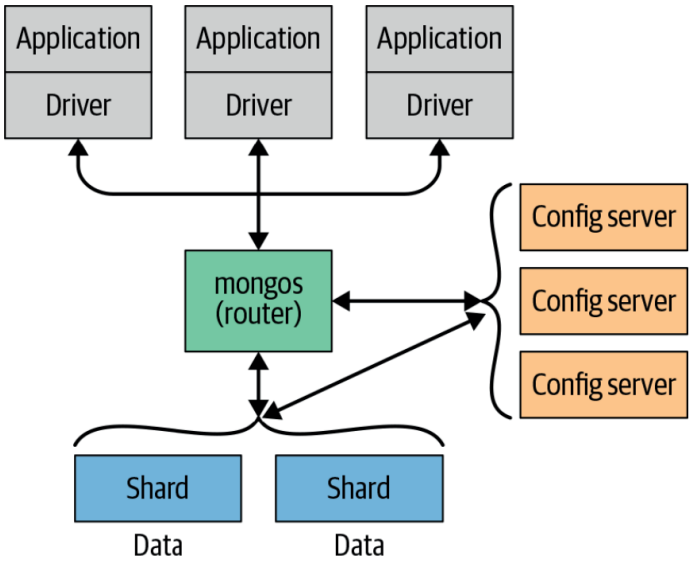

Рис. 7.1. Масштабування MongoDB з використанням шардингу на кількох серверах.

Топологія кластера MongoDB, або ж це фактично кластер, а не один вузол на іншому кінці з'єднання з базою даних, прозора для застосунку. Це дає змогу розробникам зосередитися на програмуванні застосунку, а не на його масштабуванні.

Аналогічним чином, якщо необхідно змінити топологію наявного розгортання, наприклад для масштабування, щоб підтримати велике навантаження, логіка застосунку може залишатися незмінною.

Багатство функцій. MongoDB – СУБД загального призначення, тому, крім створення, читання, оновлення та видалення даних, вона надає більшість тих функцій, які можна очікувати від системи керування базами даних, і багато інших, які виділяють її. Серед них:

- індексування: MongoDB підтримує загальні вторинні індекси та надає унікальне, складене, геопросторове та повнотекстове індексування. Також підтримуються вторинні індекси в ієрархічних структурах, як-от вкладені документи та масиви, що дають змогу розробникам повною мірою використовувати можливість моделювання в такий спосіб, що найбільше підходить для їхніх застосунків;
- агрегація: MongoDB надає фреймворк для агрегації на базі концепції конвеєрів обробки даних. Конвеєри агрегації дають змогу створювати складні аналітичні механізми, обробляючи дані через низку відносно

простих етапів на стороні сервера, використовуючи всі переваги оптимізації бази даних.

Спеціальні типи колекцій та індексів. MongoDB підтримує колекції даних TTL (time-to-live), термін дії яких має спливти в певний час, як-от сеанси та колекції фіксованого розміру, для зберігання нещодавно отриманих даних, наприклад журналів. MongoDB також підтримує часткові індекси, обмежені тільки тими документами, які відповідають фільтру критеріїв, щоб підвищити ефективність і зменшити необхідний обсяг дискового простору.

Файлове сховище. MongoDB підтримує простий у використанні протокол для зберігання великих файлів і метаданих файлів.

Деякі функції, поширені в реляційних СУБД, у MongoDB відсутні, особливо складні з'єднання таблиць. Обробка з'єднань у MongoDB була архітектурним рішенням, що забезпечує більшу масштабованість, оскільки обидві ці функції складно ефективно реалізувати в розподіленій системі.

Продуктивність є основною метою MongoDB і багато в чому сформувала її дизайн. Хоча MongoDB є потужним засобом, що містить безліч функцій реляційних систем, вона не призначена для виконання всього того, що робить реляційна СУБД. У випадку з деякими функціями сервер бази даних переносить обробку і логіку на клієнтський бік (обробка здійснюється або драйверами, або кодом програми користувача). Підтримка цього обтічного дизайну є однією з причин, через яку MongoDB може досягати такої високої продуктивності.

7.1. Основні концепції MongoDB

MongoDB – потужний інструмент, з яким легко почати роботу. У цьому розділі ми познайомимося з деякими основними поняттями MongoDB:

- *документ* є основною одиницею даних у MongoDB і приблизно еквівалентний рядку в реляційній системі керування базами даних (але набагато виразніший);
- аналогічно *колекцію* можна розглядати як таблицю з динамічною схемою;
- один примірник MongoDB може містити кілька незалежних *баз даних*, кожна з яких містить свої власні колекції;
- кожен документ має спеціальний ключ «_id», що є унікальним у межах колекції;
- MongoDB поширюється за допомогою простого, але потужного інструменту під назвою оболонка *mongo*. Оболонка *mongo shell* надає вбудовану підтримку для адміністрування екземплярів MongoDB і маніпулювання даними з використанням мови запитів MongoDB. Це також повнофункціональний інтерпретатор JavaScript, який дає змогу користувачам створювати та завантажувати власні сценарії для різних цілей.

Документи. В основі MongoDB лежить документ: упорядкований набір ключів зі зв'язаними значеннями. Подання документа залежить від мови програмування, але більшість мов мають природну структуру даних, наприклад асоціативний масив або словник. Наприклад, у JavaScript документи представлені у вигляді об'єктів:

```
{"greeting" : "Hello, world!"}
```

Цей простий документ містить єдиний ключ `"greeting"` зі значенням `"Hello, world!"`. Більшість документів будуть складнішими порівняно з цим і часто міститимуть кілька пар типу «ключ/значення»:

```
{"greeting" : "Hello, world!", "views" : 3}
```

Як видно, значення в документах – це не просто «BLOB-об'єкти». Вони можуть належати до одного з декількох типів даних (або навіть до всього вкладеного документа). У цьому прикладі значення `"greeting"` є рядком, тоді як значення `"views"` – це ціле число.

Ключі в документі – це рядки. У ключі допустиме використання будь-якого символу в кодуванні UTF-8, за кількома помітними винятками: символи . і $ володіють деякими спеціальними властивостями і повинні використовуватися тільки за певних обставин. Загалом їх слід вважати зарезервованими, і драйвери скаржитимуться, якщо ці символи будуть використовуватися не за призначенням.

MongoDB чутлива до типу і регістру. Наприклад, ці документи відрізняються:

```
{"count" : 5}
{"count" : "5"}
```

так само як і ці:

```
{"count" : 5}
{"Count" : 5}
```

І ще одна важлива річ: документи в MongoDB не можуть містити дублікати ключів. Наприклад, наведений нижче документ не є допустимим:

```
{"greeting" : "Hello, world!", "greeting" : "Hello, MongoDB!"}
```

Колекції. *Колекція* являє собою групу документів. Якщо в MongoDB документ є аналогом рядка в реляційній СУБД, то колекцію можна розглядати як аналог таблиці.

Динамічні схеми. Колекції мають динамічні схеми. Це означає, що документи в одній колекції можуть мати будь-яку кількість різних «форм». Наприклад, обидва наведені нижче документи можуть зберігатися в одній колекції:

```
{"greeting" : "Hello, world!", "views": 3}
{"signoff": "Good night, and good luck"}
```

Зверніть увагу на те, що попередні документи мають різні ключі, різну кількість ключів і значення різних типів. Оскільки будь-який документ можна помістити в будь-яку колекцію, часто виникає запитання: «Навіщо взагалі потрібні окремі колекції?» Якщо немає потреби в окремих схемах для різних видів документів, навіщо використовувати додаткові колекції? На те є низка вагомих причин:

- зберігання різних видів документів в одній колекції може стати кошмаром для розробників і адміністраторів. Розробники повинні переконатися, що кожен запит повертає тільки документи, прив'язані до певної схеми, або що код програми, який виконує запит, може обробляти документи різної форми. Якщо ми запитуємо пости в блозі, дуже складно відсіяти документи, що містять дані про авторів;
- отримати список колекцій набагато швидше, ніж витягти список типів документів у колекції. Наприклад, якби в кожному документі було поле «type», в якому вказувалося, чи був це документ «персонал», «платежі» або «інший», було б набагато повільніше шукати ці три значення в одній колекції, ніж мати три окремі колекції та запитати правильну;
- угруповання документів одного і того ж виду в одній колекції допускає локальність даних. Отримання кількох постів у блозі з колекції, що містить тільки пости, імовірно, потребуватиме менше операцій пошуку на диску, ніж отримання тих самих постів із колекції, де містяться пости і дані про авторів;
- ми починаємо нав'язувати своїм документам певну структуру при створенні індексів. (Це особливо вірно у випадку з унікальними індексами.) Ці індекси визначаються для кожної колекції. Поміщаючи в одну колекцію тільки документи одного типу, можна ефективніше індексувати свої колекції.

Існують вагомі причини для створення схеми та групування пов'язаних типів документів. Хоча цього і не потрібно за замовчуванням, визначення схем для вашого додатка є гарною практикою і може бути реалізовано за допомогою функцій перевірки документації MongoDB і бібліотек об'єктно-документного відображення, доступних для безлічі мов програмування.

Іменування. Колекція ідентифікується за ім'ям. Імена колекцій можуть бути будь-яким рядком у кодуванні UTF-8 з деякими обмеженнями:

- порожній рядок («») не є допустимим ім'ям колекції;
- не слід створювати колекції з іменами, що починаються зі слова system. Цей префікс зарезервовано для внутрішніх колекцій. Наприклад, колекція system.users містить користувачів бази даних, а колекція system.namespaces містить інформацію про всі колекції бази даних;
- створені користувачем колекції не повинні містити зарезервований символ $ у своїх іменах. Різні драйвери, доступні для бази даних, все ж підтримують застосування цього символу в іменах колекцій, тому що він

міститься в деяких згенерованих системою колекціях, але ви не повинні використовувати $ в імені, тільки якщо ви не виконуєте доступ до однієї з цих колекцій.

Вкладені колекції. Одна з угод для організації колекцій полягає в тому, щоб використовувати вкладені колекції простору імен, розділені символом «.». Наприклад, додаток, що містить блог, може мати колекцію з ім'ям blog.posts і окрему колекцію з ім'ям blog.authors. Це слугує лише організаційним цілям – немає жодного зв'язку між колекцією blog (вона навіть не повинна існувати) та її «нащадками». Хоча вкладені колекції не мають якихось спеціальних властивостей, вони корисні і включені в багато інструментів MongoDB. Наприклад:

- GridFS, протокол для зберігання великих файлів, використовує вкладені колекції для зберігання метаданих файлів окремо від блоків вмісту;
- більшість драйверів надають певний «синтаксичний цукор» для доступу до вкладеної колекції. Наприклад, в оболонці бази даних db.blog надасть вам колекцію blog, а db.blog.posts – колекцію blog.posts.

Вкладені колекції – хороший спосіб організувати дані в MongoDB для безлічі випадків використання.

Бази даних. На додаток до групування документів за колекцією MongoDB групує колекції в бази даних. Один екземпляр MongoDB може містити кілька баз даних, кожна з яких об'єднує нуль або більше колекцій. Є гарне практичне правило – зберігати всі дані одного застосунку в одній і тій самій базі даних. Окремі бази даних корисні при зберіганні даних для декількох додатків або користувачів на одному сервері MongoDB.

Як і колекції, бази даних ідентифікуються за ім'ям. Імена баз даних можуть бути будь-яким рядком у форматі UTF-8 з такими обмеженнями:

- порожній рядок («») не є допустимим ім'ям бази даних; - ім'я бази даних не може містити такі символи: /, \, ., », *, , :, |, ?, $, (один пробіл). Здебільшого дотримуватися буквено-цифрової таблиці ASCII;
- імена баз даних нечутливі до регістру;
- імена баз даних обмежені максимум 64 байтами.

Традиційно, до використання підсистеми зберігання WiredTiger, імена баз даних ставали файлами у вашій файловій системі. Тепер цього більше немає. Це пояснює, чому багато які з попередніх обмежень взагалі існують.

Є також деякі зарезервовані імена баз даних, до яких можна отримати доступ, але які мають особливу семантику. Ось вони:

- *admin* – база даних *admin* відіграє роль в автентифікації та авторизації. Крім того, доступ до цієї бази даних необхідний для низки адміністративних операцій.
- *local* – у цій базі даних зберігаються дані, що належать до одного сервера. У наборах реплік у базі *local* зберігаються дані, що використовуються в процесі реплікації. Сама база даних *local* ніколи не реплікується.

– *config* – розділені (сегментовані) кластери MongoDB використовують базу даних *config* для зберігання інформації про кожен шард.

Об'єднуючи ім'я бази даних із колекцією в цій базі даних, ви можете отримати повне ім'я колекції, яке називається простором імен. Наприклад, якщо ви використовуєте колекцію blog.posts у базі даних cms, простір імен цієї колекції буде таким: cms.blog.posts. Довжина просторів імен обмежена 120 байтами, а на практиці має бути менше 100 байт.

7.2. Налаштування MongoDB на локальному комп'ютері

Першим кроком необхідно завантажити MongoDB за посиланням (рис. 7.2). Далі йде стандартна процедура інсталяції із стандартними налаштуваннями (рис. 7.3).

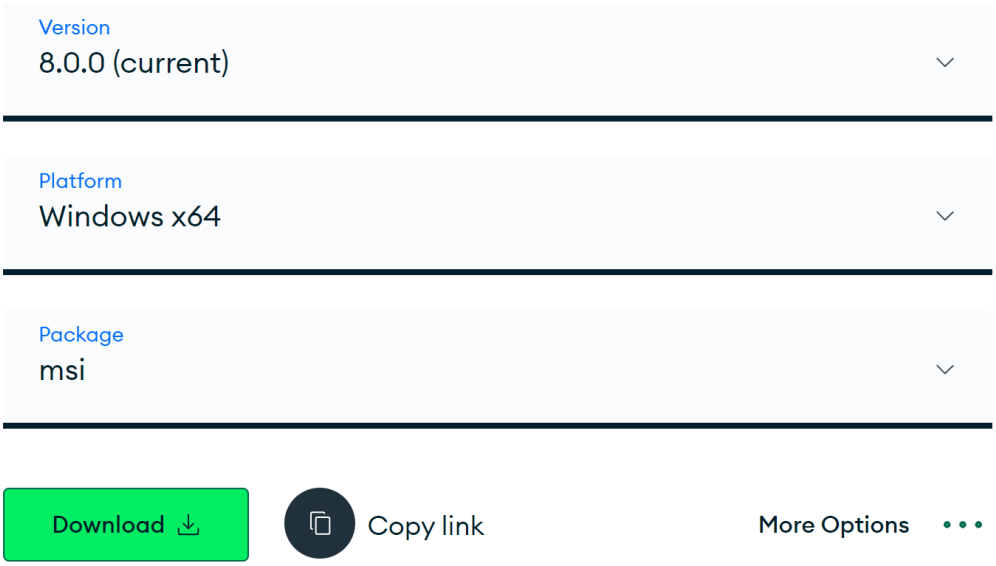

Рис. 7.2. Завантаження MongoDB.

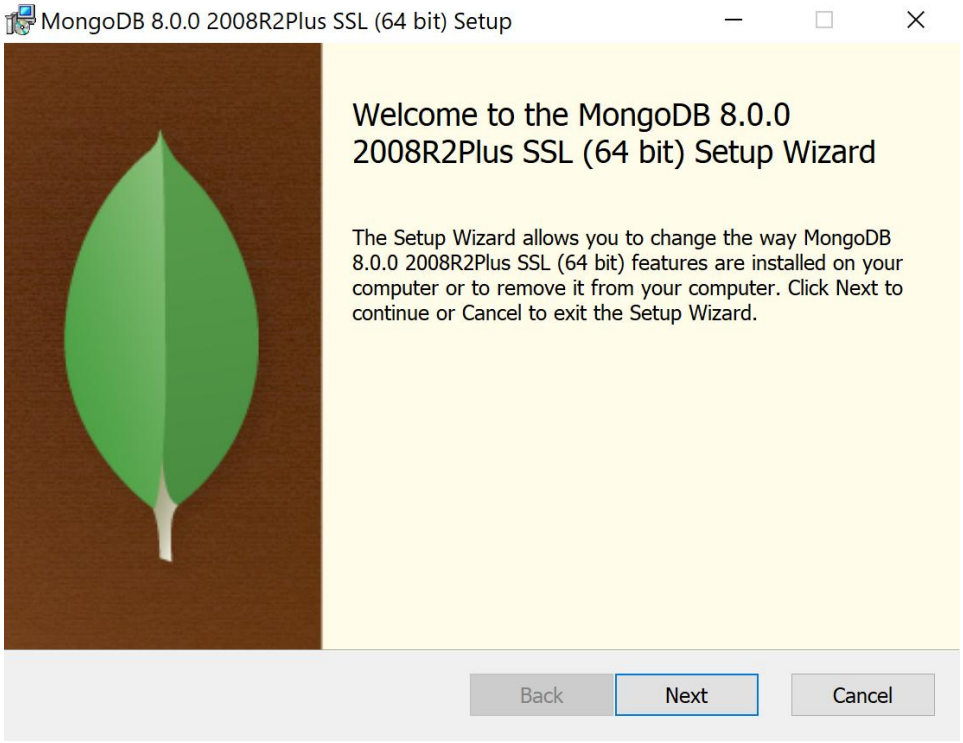

Рис. 7.3. Початок встановлення MongoDB.

7.3. MongoDB Compass – візуальний інструмент роботи з MongoDB

MongoDB Compass – це безкоштовний, відкритий і потужний клієнт з графічним інтерфейсом для MongoDB, популярної бази даних NoSQL. Він надає інтуїтивно зрозумілий і зручний інтерфейс для взаємодії з базами даних MongoDB, полегшуючи керування, запити та аналіз даних.

Програма також надає функції для дослідження даних, такі як фільтрація, сортування та агрегування, які допоможуть вам досліджувати та аналізувати ваші дані. Крім того, Compass дозволяє створювати, оновлювати та видаляти документи безпосередньо в додатку.

За допомогою MongoDB Compass ви можете підключатися до локальних або віддалених екземплярів MongoDB. Інструмент також дозволяє імпортувати та експортувати дані у форматах JSON або CSV.

Загалом, MongoDB Compass – це універсальний і потужний клієнт, який спрощує процес роботи з базами даних MongoDB, що робить його незамінним інструментом для розробників, адміністраторів та аналітиків, які працюють з MongoDB на щоденній основі.

Початок роботи з MongoDB Compass. Після відкриття MongoDB Compass необхідно створити з’єднання із БД. Для цього натискаємо «Add New Connection» (рис. 7.4).

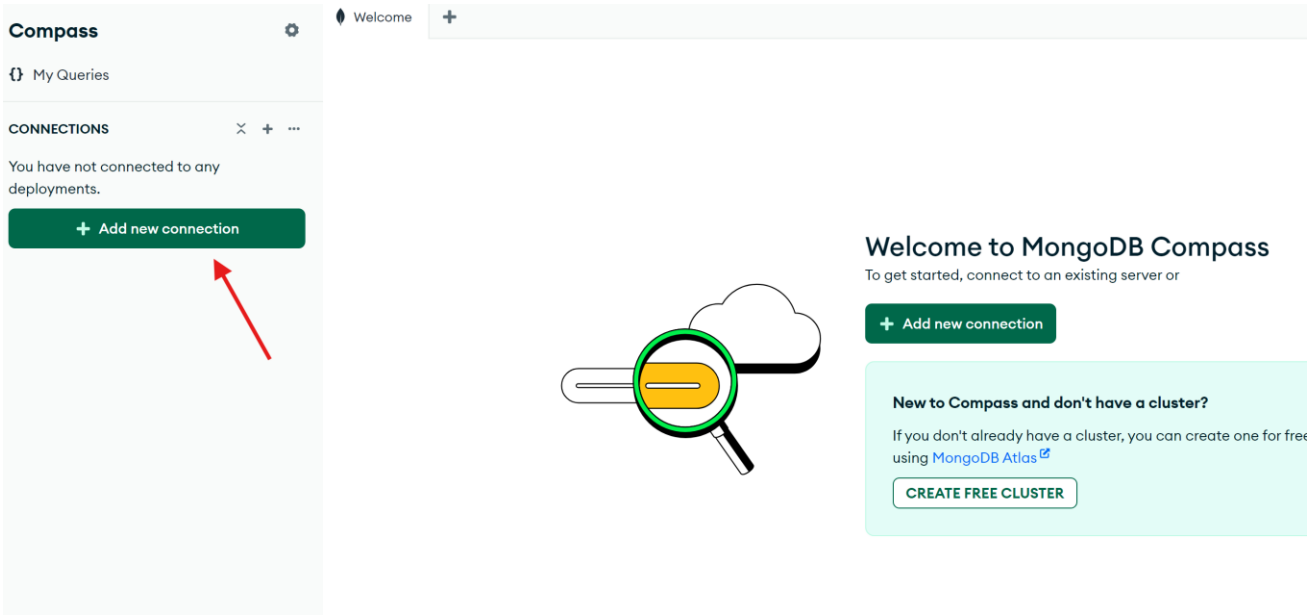

Рис. 7.4. Створення нового з’єднання з БД.

У вікні «Add New Connection» залишаємо всі налаштування без змін та натискаємо «Connect» (рис. 7.5).

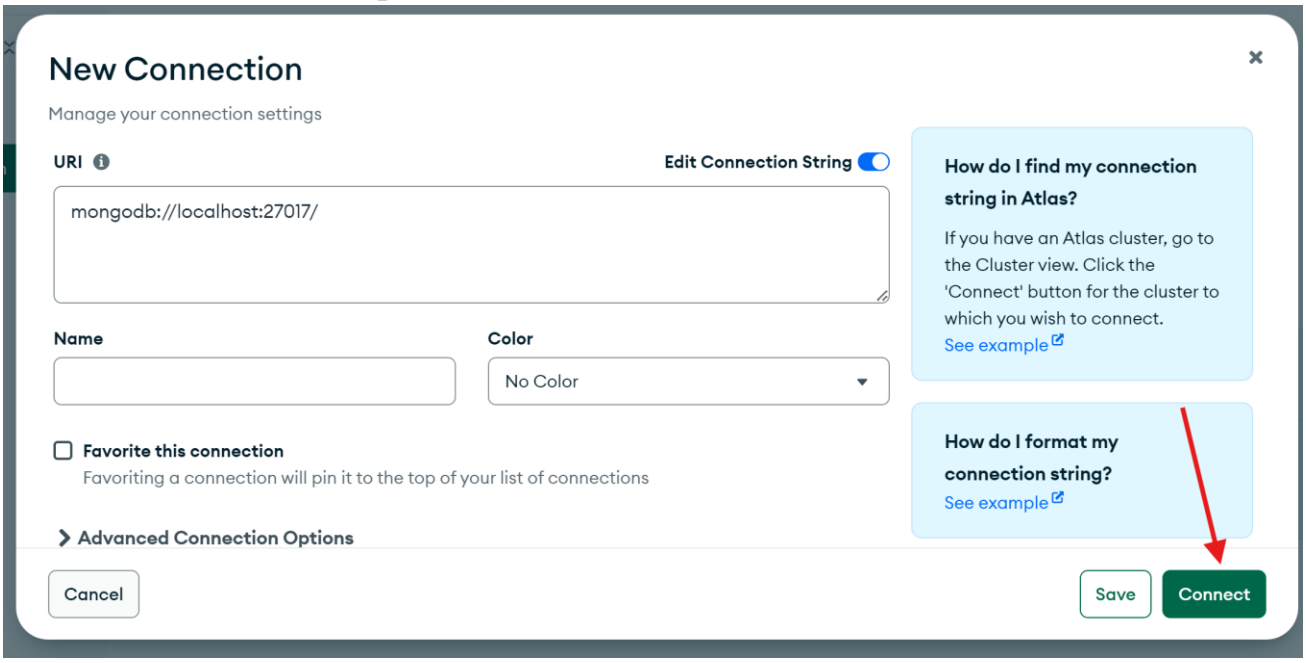

Рис. 7.5. З’єднання з БД.

Натиснувши «+» можна створити нову базу даних «MyDb» та колекцію в ній «MyCollection» (рис. 7.6 та рис. 7.7).

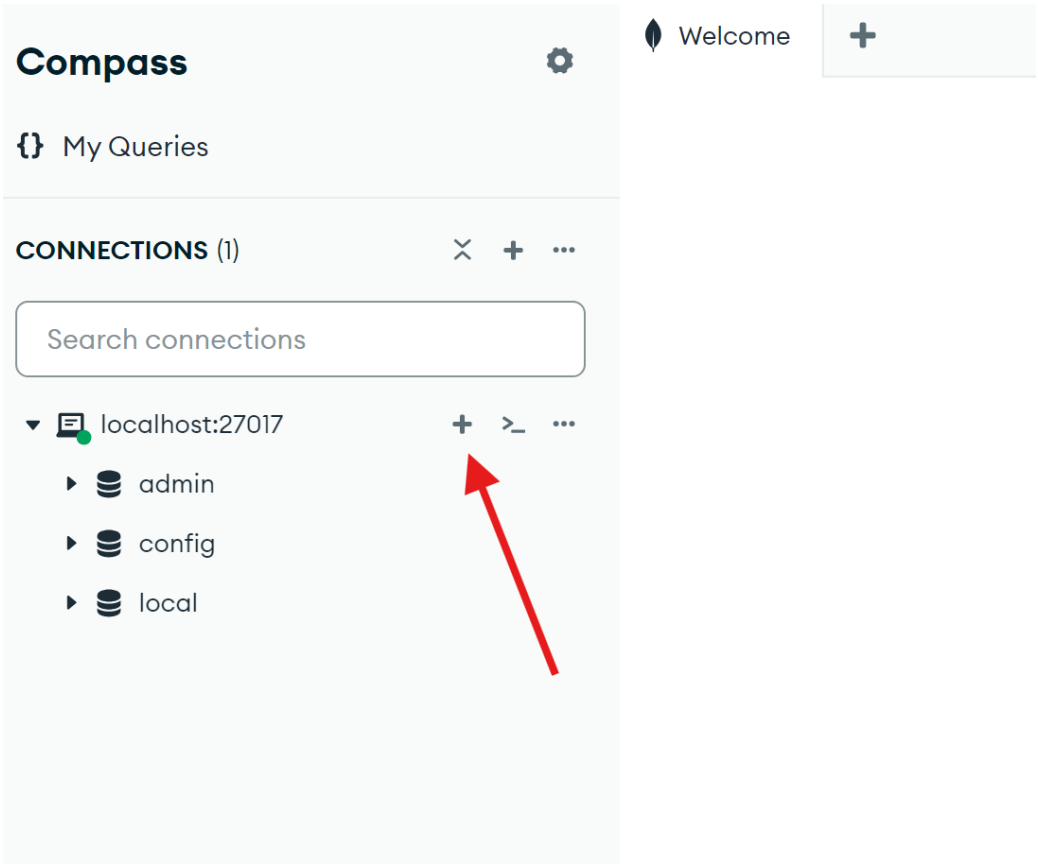

Рис. 7.6. Кнопка для створення нової бази даних.

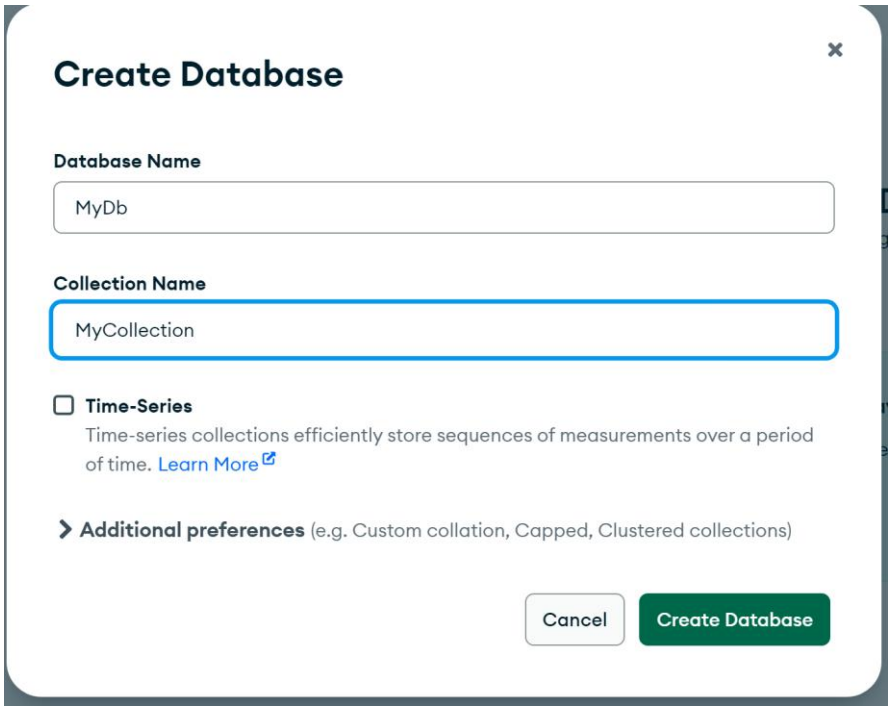

Рис. 7.7. Створення бази даних та колекції в ній.

Імпорт даних. Для імпорту даних натискаємо «Add Data» і після цього «Import JSON or CSV file» (рис. 7.8). Обираємо файл із готовими даними. У вікні,

що з'явилось, натискаємо «Import» для завершення імпорту. Після цього, ви отримаєте колекцію бази даних із доданими документами з обраного файлу.

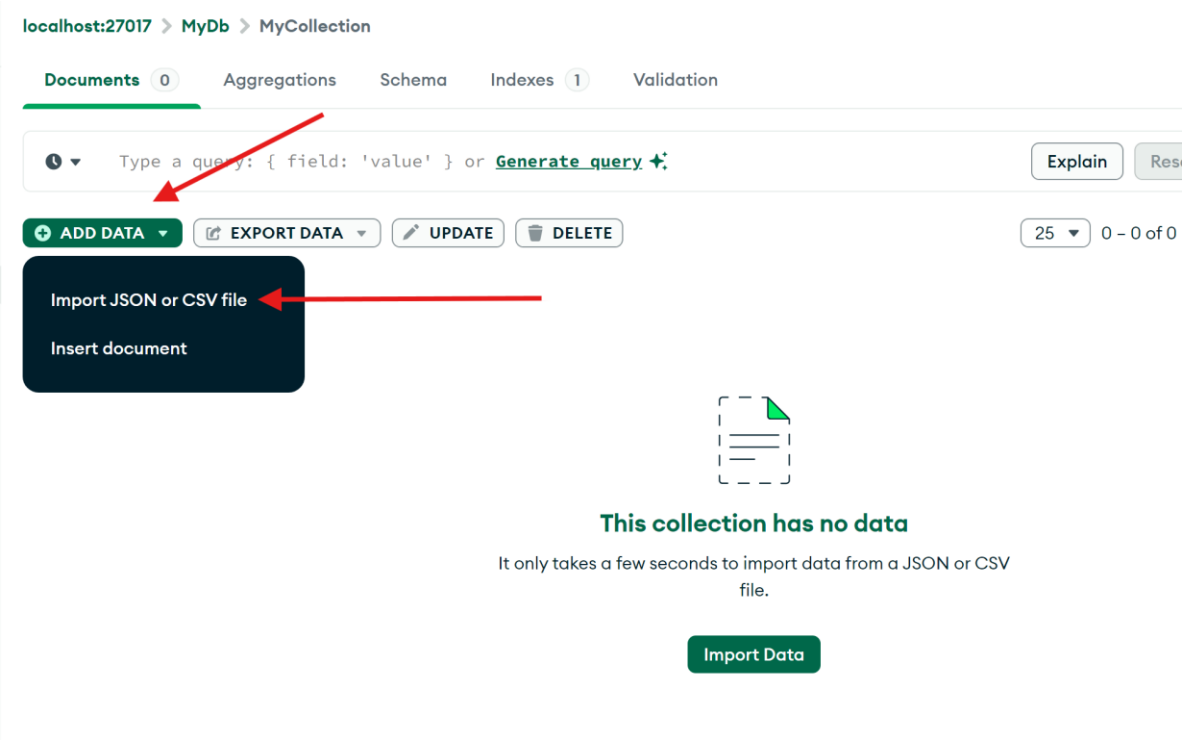

Рис. 7.8. Додавання даних.

7.4. Контрольні питання

1. Використовуючи які інструменти можна під’єднатись до MongoDB?
2. Які є зарезервовані імена баз даних?
3. Що таке вкладені колекції?
4. Чи необхідно зводити всі документи в MongoDB до однакової форми? Проаналізуйте переваги та недоліки обох випадків.
5. Які концепції використовує MongoDB для структурних елементів бази даних? Які подібні концепції існують в реляційних базах даних?
6. Як імпортувати дані в БД через MongoDB Compass?
7. Чому зберігання кількох застосунків у одній базі може бути невигідним?
8. Що таке простір імен в контексті MongoDB? Яке обмеження на його довжину існує?

Лекція 8 – Засоби роботи з MongoDB в Python

MongoDB, популярна NoSQL-база даних, стала вибором для багатьох розробників завдяки своїй гнучкості, масштабованості та високій продуктивності. Щоб ефективно взаємодіяти з екземпляром MongoDB з програми на Python, нам потрібно покладатися на сторонні бібліотеки, які надають зручний інтерфейс до бази даних.

У цій лекції ми розглянемо бібліотеку pymongo, яка є офіційним драйвером Python для MongoDB. Будуть наведені практичні приклади та фрагменти коду, які демонструють, як використовувати ці інструменти для виконання поширених завдань з базами даних, таких як вставка даних, запити, агрегування та інше. Наприкінці цього розділу ви будете добре розуміти, як ефективно працювати з MongoDB у ваших додатках на Python, що дозволить вам створювати масштабовані та ефективні системи, керовані даними.

Для роботи з MongoDB із мовою програмування Python необхідно встановити відповідний пакет:

Командний рядок

```
pip install pymongo
```

8.1. Створення баз даних та колекцій

Створення бази даних. Щоб створити базу даних у MongoDB, почніть із створення об’єкта MongoClient, а потім укажіть URL-адресу з’єднання з правильною ip-адресою та назвою бази даних, яку ви хочете створити.

MongoDB створить базу даних, якщо вона не існує, і підключиться до неї.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

mydb = myclient["mydatabase"]
```

Важливо: у MongoDB база даних не створюється, доки вона не отримає вміст!

MongoDB чекає, доки ви не створите колекцію (таблицю) з принаймні одним документом (записом), перш ніж фактично створити базу даних (і колекцію).

Перевірка існування бази даних

Пам’ятайте: у MongoDB база даних не створюється, доки вона не отримає вміст, тому, якщо ви створюєте базу даних уперше, вам слід завершити наступні два розділи (створення колекції та створення документа), перш ніж перевіряти, чи існує база даних!

Ви можете перевірити, чи існує база даних, перерахувавши всі бази даних у вашій системі:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")

print(myclient.list_database_names())
```

Вивід

```
['admin', 'config', 'local']
```

Створення колекції. Колекція в MongoDB — це те саме, що таблиця в базах даних SQL. Щоб створити колекцію в MongoDB, використовуйте об'єкт бази даних і вкажіть назву колекції, яку ви хочете створити.

MongoDB створить колекцію, якщо вона не існує.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]

mycol = mydb["customers"]
```

Важливо: у MongoDB колекція не створюється, доки вона не отримає вміст! MongoDB чекає, поки ви вставите документ, перш ніж фактично створити колекцію.

Перевірте, чи існує колекція. Пам'ятайте: у MongoDB колекція не створюється, доки вона не отримає вміст, тому, якщо ви створюєте колекцію вперше, вам слід завершити наступний розділ (створити документ), перш ніж перевіряти, чи існує колекція!

Ви можете перевірити, чи існує колекція в базі даних, перерахувавши всі колекції:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]

print(mydb.list_collection_names())
```

8.2. Додавання даних

Вставлення в колекцію

Документ у MongoDB – це те саме, що запис у базах даних SQL.

Щоб вставити запис або документ, як це називається в MongoDB, у колекцію, ми використовуємо метод insert_one().

Перший параметр методу insert_one() – це словник, що містить імена та значення кожного поля в документі, який потрібно вставити.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydict = { "name": "John", "address": "Highway 37" }

x = mycol.insert_one(mydict)
```

Поле _id. Метод insert_one() повертає об'єкт InsertOneResult, який має властивість inserted_id, яка містить ідентифікатор вставленого документа.

Вставте інший запис у колекцію «клієнти» та поверніть значення поля _id:

Код

```
mydict = { "name": "Peter", "address": "Lowstreet 27" }

x = mycol.insert_one(mydict)

print(x.inserted_id)
```

Вивід

```
66fadf5465f84ef9c38a7b94
```

Якщо ви не вкажете поле _id, MongoDB додасть його для вас і призначить унікальний ідентифікатор для кожного документа.

У наведеному вище прикладі не було вказано поле _id, тому MongoDB призначив унікальний _id для запису (документа).

Вставлення декількох документів. Щоб вставити кілька документів у колекцію в MongoDB, ми використовуємо метод insert_many().

Перший параметр методу insert_many() – це список словників із даними, які потрібно вставити:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "name": "Amy", "address": "Apple st 652"},
  { "name": "Hannah", "address": "Mountain 21"},
  { "name": "Michael", "address": "Valley 345"},
  { "name": "Sandy", "address": "Ocean blvd 2"},
  { "name": "Betty", "address": "Green Grass 1"},
  { "name": "Richard", "address": "Sky st 331"},
  { "name": "Susan", "address": "One way 98"},
  { "name": "Vicky", "address": "Yellow Garden 2"},
  { "name": "Ben", "address": "Park Lane 38"},
  { "name": "William", "address": "Central st 954"},
  { "name": "Chuck", "address": "Main Road 989"},
  { "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)

# Друк ідентифікаторів _id вставлених документів:
print(x.inserted_ids)
```

Вивід

```
[ObjectId('66fadfa8266aaa9f6110e31f'), ObjectId('66fadfa8266aaa9f6110e320'),
ObjectId('66fadfa8266aaa9f6110e321'), ObjectId('66fadfa8266aaa9f6110e322'),
ObjectId('66fadfa8266aaa9f6110e323'), ObjectId('66fadfa8266aaa9f6110e324'),
ObjectId('66fadfa8266aaa9f6110e325'), ObjectId('66fadfa8266aaa9f6110e326'),
ObjectId('66fadfa8266aaa9f6110e327'), ObjectId('66fadfa8266aaa9f6110e328'),
ObjectId('66fadfa8266aaa9f6110e329'), ObjectId('66fadfa8266aaa9f6110e32a')]
```

Метод insert_many() повертає об’єкт InsertManyResult, який має властивість inserted_ids, яка містить ідентифікатори вставлених документів.

Вставлення кількох документів із зазначеними ідентифікаторами. Якщо ви не хочете, щоб MongoDB призначав унікальні ідентифікатори для вашого документа, ви можете вказати поле _id під час вставлення документа(ів).

Пам’ятайте, що значення мають бути унікальними. Два документи не можуть мати однаковий _id.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mylist = [
  { "_id": 1, "name": "John", "address": "Highway 37"},
  { "_id": 2, "name": "Peter", "address": "Lowstreet 27"},
  { "_id": 3, "name": "Amy", "address": "Apple st 652"},
  { "_id": 4, "name": "Hannah", "address": "Mountain 21"},
  { "_id": 5, "name": "Michael", "address": "Valley 345"},
  { "_id": 6, "name": "Sandy", "address": "Ocean blvd 2"},
  { "_id": 7, "name": "Betty", "address": "Green Grass 1"},
  { "_id": 8, "name": "Richard", "address": "Sky st 331"},
  { "_id": 9, "name": "Susan", "address": "One way 98"},
  { "_id": 10, "name": "Vicky", "address": "Yellow Garden 2"},
  { "_id": 11, "name": "Ben", "address": "Park Lane 38"},
  { "_id": 12, "name": "William", "address": "Central st 954"},
  { "_id": 13, "name": "Chuck", "address": "Main Road 989"},
  { "_id": 14, "name": "Viola", "address": "Sideway 1633"}
]

x = mycol.insert_many(mylist)

# Друк ідентифікаторів _id вставлених документів:
print(x.inserted_ids)
```

Вивід

```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

8.3. Операції пошуку даних

У MongoDB ми використовуємо методи find() і find_one() для пошуку даних у колекції.

Так само, як оператор SELECT використовується для пошуку даних у таблиці в базі даних MySQL.

Пошук одного елемента. Щоб вибрати дані з колекції в MongoDB, ми можемо використати метод find_one().

Метод find_one() повертає перше входження у вибірку.

Знайдіть перший документ у колекції клієнтів:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

x = mycol.find_one()

print(x)
```

Вивід

```
{'_id': ObjectId('66fadf27e360b98a5cd8b009'), 'name': 'John', 'address': 'Highway
37'}
```

Знайти всі елементи. Щоб вибрати дані з таблиці в MongoDB, ми також можемо використовувати метод find().

Метод find() повертає всі елементи.

Першим параметром методу find() є об'єкт запиту. У цьому прикладі ми використовуємо порожній об'єкт запиту, який вибирає всі документи в колекції.

Метод find() без параметрів є еквівалентним SELECT * в SQL.

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

for x in mycol.find():
  print(x)
```

Вивід

```
{'_id': 1, 'name': 'John', 'address': 'Highway 37'}
{'_id': 2, 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 4, 'name': 'Hannah', 'address': 'Mountain 21'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 6, 'name': 'Sandy', 'address': 'Ocean blvd 2'}
(вивід скорочено)
```

8.4. Запити до даних

Фільтрування результату. Під час пошуку документів у колекції ви можете відфільтрувати результат за допомогою об'єкта запиту.

Перший аргумент методу find() є об’єктом запиту, який використовується для обмеження пошуку.

Знайти документ(и) з адресою «Park Lane 38»:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Park Lane 38" }

mydoc = mycol.find(myquery)

for x in mydoc:
  print(x)
```

Вивід

```
{'_id': ObjectId('66fadfa8266aaa9f6110e327'), 'name': 'Ben', 'address': 'Park Lane
38'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
```

Розширений запит. Щоб робити розширені запити, ви можете використовувати модифікатори як значення в об’єкті запиту. Наприклад щоб знайти документи, у яких поле «адреса» починається з літери «S» або вище (за алфавітом), використовуйте модифікатор «більше ніж»: {"$gt": "S"}.

Знайдіть документи, адреса яких починається з літери "S" або вище:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": { "$gt": "S" } }

mydoc = mycol.find(myquery)

for x in mydoc:
  print(x)
```

Вивід

```
{'_id': ObjectId('66fadfa8266aaa9f6110e321'), 'name': 'Michael', 'address':
'Valley 345'}
```

```
{'_id': ObjectId('66fadfa8266aaa9f6110e324'), 'name': 'Richard', 'address': 'Sky
st 331'}
{'_id': ObjectId('66fadfa8266aaa9f6110e326'), 'name': 'Vicky', 'address': 'Yellow
Garden 2'}
{'_id': ObjectId('66fadfa8266aaa9f6110e32a'), 'name': 'Viola', 'address': 'Sideway
1633'}
{'_id': 5, 'name': 'Michael', 'address': 'Valley 345'}
{'_id': 8, 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
```

8.5. Сортування результатів

Використовуйте метод sort() для сортування результату за зростанням або спаданням. Метод sort() отримує один параметр «назва поля» і один параметр «напрямок» (за замовчуванням напрямок за зростанням).

Сортування результату в алфавітному порядку за назвою:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name")

for x in mydoc:
  print(x)
```

Вивід

```
{'_id': ObjectId('66fadfa8266aaa9f6110e31f'), 'name': 'Amy', 'address': 'Apple st
652'}
{'_id': 3, 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': 11, 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('66fadfa8266aaa9f6110e327'), 'name': 'Ben', 'address': 'Park Lane
38'}
{'_id': ObjectId('66fadfa8266aaa9f6110e323'), 'name': 'Betty', 'address': 'Green
Grass 1'}
{'_id': 7, 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': 13, 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': ObjectId('66fadfa8266aaa9f6110e329'), 'name': 'Chuck', 'address': 'Main
Road 989'}
{'_id': ObjectId('66fadfa8266aaa9f6110e320'), 'name': 'Hannah', 'address':
'Mountain 21'}
(вивід скорочено)
```

Сортування за спаданням. Використовуйте значення -1 як другий параметр для сортування за спаданням.

```
sort("name", 1) # за зростанням
sort("name", -1) # за спаданням
```

Сортування результату у зворотному порядку за алфавітом за назвою:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

mydoc = mycol.find().sort("name", -1)

for x in mydoc:
  print(x)
```

Вивід

```
{'_id': 12, 'name': 'William', 'address': 'Central st 954'}
{'_id': ObjectId('66fadfa8266aaa9f6110e328'), 'name': 'William', 'address':
'Central st 954'}
{'_id': 14, 'name': 'Viola', 'address': 'Sideway 1633'}
{'_id': ObjectId('66fadfa8266aaa9f6110e32a'), 'name': 'Viola', 'address': 'Sideway
1633'}
{'_id': 10, 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('66fadfa8266aaa9f6110e326'), 'name': 'Vicky', 'address': 'Yellow
Garden 2'}
{'_id': 9, 'name': 'Susan', 'address': 'One way 98'}
{'_id': ObjectId('66fadfa8266aaa9f6110e325'), 'name': 'Susan', 'address': 'One way
98'}
{'_id': ObjectId('66fadfa8266aaa9f6110e322'), 'name': 'Sandy', 'address': 'Ocean
blvd 2'}
(вивід скорочено)
```

8.6. Видалення даних

Видалення документа. Щоб видалити один документ, ми використовуємо метод delete_one().

Перший параметр методу delete_one() — це об’єкт запиту, який визначає, який документ потрібно видалити.

Примітка: якщо запит знаходить більше одного документа, видаляється лише перший екземпляр.

Видалити документ з адресою «Mountain 21»:

Код

```
myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Mountain 21" }

mycol.delete_one(myquery)
```

Вивід

Видалення багатьох документів. Щоб видалити кілька документів, скористайтеся методом delete_many().

Перший параметр методу delete_many() — це об’єкт запиту, який визначає, які документи потрібно видалити.

Видалити всі документи, адреса яких «Yellow Garden 2»:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]

myquery = { "address": "Yellow Garden 2" }

x = mycol.delete_many(myquery)

print(x.deleted_count, " documents deleted.")
```

Вивід

```
2  documents deleted.
```

Видалити всі документи в колекції. Щоб видалити всі документи в колекції, передайте порожній об’єкт запиту методу delete_many():

Видалити всі документи в колекції «customers»:

Код

```
import pymongo

myclient = pymongo.MongoClient("mongodb://localhost:27017/")
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
```

```
x = mycol.delete_many({})

print(x.deleted_count, " documents deleted.")
```

Вивід

```
25 documents deleted.
```

8.7. Контрольні питання

1. Який пакет Python необхідно використати для під’єднання до MongoDB?
2. Чи можливе створення порожньої БД в MongoDB?
3. Що таке «драйвер» в термінах MongoDB?
4. Які особливості створення БД та колекцій в MongoDB?
5. Як здійснити пошук елемента в MongoDB? Як вказати, що потрібен більший або менший елемент?
6. У який спосіб зручно імпортувати дані із стороннього джерела?
7. Як відсортувати дані за спаданням певного поля?
8. Що повертає функція видалення елементів в MongoDB?

Лекція 9 – Ланка між PySpark та MongoDB

В даній лекції буде розглянуто яким чином можливо поєднати дві ключові технології великих даних – Apache Spark та MongoDB. Apache Spark з його потужним двигуном для великомасштабної обробки даних і MongoDB, провідна база даних NoSQL, розроблена для сучасних додатків, все частіше використовуються разом для вирішення складних проблем з даними.

PySpark, Python API для Apache Spark, забезпечує елегантний спосіб використання можливостей Spark з середовища Python. Тим часом, MongoDB стала популярним вибором для розробників, які шукають гнучку схему та високу масштабованість своїх баз даних. Однак інтеграція цих двох технологій може бути складним завданням, особливо коли мова йде про використання сильних сторін обох світів.

У цьому розділі ми розглянемо зв'язок між PySpark і MongoDB, продемонструємо, як ефективно інтегрувати ці два потужні інструменти. Розглянемо наступний практичний приклад пайплайну ETL, що використовує обидві ці технології: створення єдиної бази даних клієнтів.

Уявімо, що ми – компанія, яка займається електронною комерцією і має кілька інтернет-магазинів у різних регіонах, кожен з яких має власну систему баз даних. Наша мета – створити єдину базу даних клієнтів, яку можна використовувати для аналізу та маркетингу.

Поточний стан:

1. Магазин A: цей магазин використовує базу даних MongoDB для зберігання інформації про клієнтів, включаючи ім'я, електронну пошту, адресу та історію покупок.
2. Магазин B: цей магазин використовує базу даних MySQL для зберігання аналогічної інформації про клієнтів.
3. Магазин C: цей магазин не має централізованої системи баз даних; дані про клієнтів зберігаються в декількох електронних таблицях і CRM-системах.

Проблеми такої системи:

1. Неузгодженість даних: база даних кожного магазину має власну унікальну структуру, що ускладнює інтеграцію даних з різних джерел.
2. Відсутність уніфікованого аналізу: через розрізненість баз даних наша маркетингова команда не може легко отримати доступ та проаналізувати поведінку клієнтів у всіх магазинах.
3. Труднощі з ідентифікацією цінних клієнтів: без централізованого перегляду інформації про клієнтів нам важко ідентифікувати клієнтів з найбільшими витратами або прогнозувати майбутні моделі покупок.

Ми згадати підхід ETL, але що ж таке ETL?

9.1. Витягнути, перетворити, завантажити, або ETL

Що таке ETL? ETL (Extract, Transform, Load) – означає «витягнути, перетворити, завантажити» – це процес інтеграції даних, який об'єднує, очищає та організовує дані з декількох джерел в єдиний, узгоджений набір даних для зберігання в сховищі даних або іншій цільовій системі.

Конвеєри даних ETL забезпечують основу для аналізу даних і робочих потоків машинного навчання. За допомогою низки бізнес-правил ETL очищає та організовує дані для задоволення конкретних потреб бізнес-аналітики, таких як щомісячна звітність, але вона також може вирішувати більш складні аналітичні завдання, які можуть покращити внутрішні процеси та досвід кінцевих користувачів. Конвеєри ETL часто використовуються організаціями для:

- вилучення даних із застарілих систем;
- очищення даних для покращення якості та узгодженості даних;
- завантаження даних у цільову базу даних.

Як розвивався ETL. Компанії генерують дані ще з часів рахівниць, але сучасна аналітика стала можливою лише з появою цифрових комп'ютерів і сховищ даних.

Значний крок вперед був зроблений у 1970-х роках, коли відбувся перехід до великих централізованих баз даних. Тоді було запроваджено ETL як процес інтеграції та завантаження даних для обчислень і аналізу, який з часом став основним методом обробки даних для проектів зі створення сховищ даних.

Наприкінці 1980-х років сховища даних і перехід від транзакційних баз даних до реляційних баз даних, які зберігали інформацію у форматах реляційних даних, набули популярності. Старіші транзакційні бази даних зберігали інформацію від транзакції до транзакції, причому дублікати інформації про клієнтів зберігалися з кожною транзакцією, тому не було простого способу отримати доступ до даних про клієнтів в уніфікованому вигляді з плином часу. З появою реляційних баз даних аналітика стала основою бізнес-аналітики (BI) і важливим інструментом для прийняття рішень.

До появи більш досконалого програмного забезпечення ETL, перші спроби були здебільшого ручними зусиллями IT-команди для вилучення даних з різних систем і з'єднувачів, перетворення даних у загальний формат, а потім завантаження їх у взаємопов'язані таблиці. Тим не менш, перші кроки ETL були варті зусиль, оскільки вдосконалені алгоритми, а також розвиток нейронних мереж створювали дедалі глибші можливості для аналітичних досліджень.

Ера великих даних настала в 1990-х роках, коли швидкість обчислень і ємність сховищ продовжували стрімко зростати, а великі обсяги даних надходили з нових джерел, таких як соціальні мережі та Інтернет речей (IoT). Обмежуючим фактором залишається те, що дані часто зберігаються в локальних сховищах даних.

Наступним важливим кроком як в обчисленнях, так і в ETL були хмарні обчислення, які стали популярними наприкінці 1990-х років. Завдяки таким сховищам даних, як Amazon Web Services (AWS), Microsoft Azure і Snowflake, доступ до даних тепер можна отримати з усього світу і швидко масштабувати, що дозволяє рішенням ETL надавати чудову детальну інформацію та нові конкурентні переваги.

Останнім досягненням є рішення ETL, що використовують потокові дані для отримання оперативної інформації з величезних масивів даних.

ETL в порівнянні з ELT. Найбільш очевидна різниця між ETL і ELT – витягти, завантажити, перетворити – полягає в порядку виконання операцій. ELT копіює або експортує дані з джерела, але замість того, щоб завантажувати їх у проміжну зону для перетворення, він завантажує необроблені дані безпосередньо в цільове сховище даних, де вони будуть перетворені за потреби.

Хоча обидва процеси використовують різні сховища даних, кожен з них має свої переваги і недоліки. ELT корисний для поглинання великих обсягів неструктурованих наборів даних, оскільки завантаження може відбуватися безпосередньо з джерела. ELT може бути більш ідеальним для управління великими даними, оскільки не потребує попереднього планування вилучення та зберігання даних.

Процес ETL вимагає більшої визначеності на самому початку. Потрібно визначити конкретні точки даних для вилучення, а також будь-які потенційні «ключі» для інтеграції між різними системами джерел. Джерело вхідних даних часто відстежується за допомогою метаданих. Навіть після завершення цієї роботи необхідно створити бізнес-правила для перетворення даних. Ця робота зазвичай може залежати від вимог до даних для певного типу аналізу даних, що визначатиме рівень узагальнення, який повинні мати дані.

Хоча конвеєри ELT стають все більш популярними з впровадженням хмарних баз даних, технологія ELT все ще розвивається, а це означає, що найкращі практики все ще формуються.

Як працює ETL. Найпростіший спосіб зрозуміти, як працює ETL, – це зрозуміти, що відбувається на кожному кроці процесу.

Витягування даних (Extract). Під час видобування даних необроблені дані копіюються або експортуються з місця їхнього походження до місця обробки. Команди управління даними можуть витягувати дані з різних джерел, які можуть бути структурованими або неструктурованими. Ці типи даних включають, але не обмежуються ними:

- SQL або NoSQL сервери;
- CRM і ERP системи;
- JSON та XML;
- плоскі бази даних;
- електронна пошта;
- веб-сторінки.

Перетворення. На етапі обробки вихідні дані проходять обробку. Тут дані трансформуються і консолідуються для цільового аналітичного використання. Ця фаза процесу трансформації може включати:

- фільтрацію, очищення, агрегування, дедуплікацію, перевірку достовірності та автентичності даних;
- виконання розрахунків, перекладів або узагальнень на основі вихідних даних. Це може включати зміну заголовків рядків і стовпців для узгодженості, конвертацію валют або інших одиниць виміру, редагування текстових рядків тощо;
- проведення аудиту для забезпечення якості та відповідності даних, а також обчислення метрик;
- видалення, шифрування або захист даних, що регулюються галузевими або державними регуляторами;
- форматування даних у таблиці або об'єднані таблиці відповідно до схеми цільового сховища даних.

Завантаження. На цьому останньому етапі перетворені дані переміщуються з місця зберігання до цільового сховища даних. Зазвичай це передбачає початкове завантаження всіх даних, а потім періодичне завантаження інкрементних змін даних і, рідше, повне оновлення для стирання і заміни даних у сховищі. Для більшості організацій, які використовують ETL, цей процес є автоматизованим, чітко визначеним, безперервним і пакетним. Зазвичай процес завантаження ETL відбувається в неробочий час, коли трафік у вихідних системах і сховищі даних найнижчий.

9.2. Об’єднання та уніфікація даних з багатьох джерел засобами PySpark

Для створення єдиної бази даних клієнтів ми впровадили ETL-конвеєр, який витягує дані з бази даних кожного магазину, перетворює їх у стандартизований формат і завантажує в центральну колекцію MongoDB. Це дозволяє нам:

1. Уніфікувати інформацію про клієнтів: ми витягуємо відповідні поля (наприклад, ім'я, електронну пошту, адресу) з усіх баз даних і перетворюємо їх у єдиний формат.
2. Агрегувати історію покупок: ми агрегуємо дані про покупки в усіх магазинах, щоб створити повне уявлення про купівельну поведінку кожного клієнта.
3. Створення єдиної уніфікованої бази даних: конвеєр ETL завантажує перетворені дані в нову колекцію в нашій базі даних MongoDB, що робить її легко доступною для аналізу та маркетингових цілей.

Переваги ETL:

1. Уніфікований аналіз: наша команда маркетологів тепер має доступ до єдиного уніфікованого перегляду інформації про клієнтів у всіх магазинах.

2. Покращене таргетування: визначаючи цінних клієнтів і прогнозуючи майбутні моделі покупок, ми можемо адаптувати наші маркетингові кампанії для кращого залучення цих цінних клієнтів.
3. Удосконалене прийняття рішень на основі даних: маючи всебічне розуміння поведінки клієнтів, ми можемо приймати більш обґрунтовані рішення щодо товарних пропозицій, стратегій ціноутворення та розподілу ресурсів.

Етапи конвеєру ETL:

1. Витяг: ми витягуємо дані про клієнтів з бази даних MongoDB магазину A, бази даних MySQL магазину B та інтегруємо їх з CRM-системою, що використовується магазином C.
2. Перетворення: ми перетворюємо витягнуті дані в стандартизований формат за допомогою PySpark, агрегуючи історію покупок і обчислюючи середню вартість замовлення.
3. Завантаження: перетворені дані завантажуються в нашу центральну колекцію MongoDB, створюючи єдину базу даних клієнтів.

Впровадивши це ETL-рішення, ми створили єдину базу даних клієнтів, яка дозволяє нам приймати більш обґрунтовані рішення щодо маркетингових стратегій і товарних пропозицій, що в кінцевому підсумку сприяє зростанню бізнесу.

Для прикладу розглянемо побудову ETL конвеєру із PySpark для бази даних клієнтів магазину C.

1. Розглянемо дані. Інформацію про клієнтів розділено на файли customers2023.csv та customers2024.csv.

customers2023.csv

```
"id","name","address","subscription"
1,"John Smith","123 Main St, Anytown, CA 12345",19.99
2,"Jane Doe","456 Elm St, Othertown, NY 67890",9.99
3,"Bob Johnson","789 Oak St, Hometown, IL 54321",29.99
4,"Maria Rodriguez","321 Pine St, Smalltown, TX 90123",0.0
5,"David Lee","901 Maple St, Bigcity, FL 34567",0.0
6,"Emily Chen","234 Cedar St, Suburbiatown, OH 45678",24.99
7,"Michael Brown","567 Spruce St, Ruraltown, GA 89012",39.99
```

customers2024.csv

```
"id","name","address","subscription"
8,"Emily Patel","987 Walnut St, Apt 3, Springfield, IL 62704",14.99
10,"Sophia Lee","1234 Oakwood Dr, Sunnyvale, CA 94087",24.99
```

```
13,"Jackson Hall","456 Elm St, Othertown, TX 76543",29.99
14,"Isabella Garcia","321 Pine St, Smalltown, FL 32456",39.99
3,"Bob Johnson","789 Oak St, Hometown, IL 54321",29.99
4,"Maria Rodriguez","321 Pine St, Smalltown, TX 90123",13.0
5,"David Lee","901 Maple St, Bigcity, FL 34567",12.0
```

2. Імпортуємо необхідні для роботи пакети

Код

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import lit
```

3. Створимо сесію PySpark. Сконфігуруємо під'єднання до нашої БД MongoDB.

Код

```
spark = SparkSession.builder.appName("Application")
spark = spark.config("spark.mongodb.output.uri",
"mongodb://mongodb:27017/productdb")
spark = spark.config("spark.jars.packages", "org.mongodb.spark:mongo-spark-
connector_2.12:10.4.0")
spark = spark.getOrCreate()
```

4. Завантажимо дані в PySpark.

Код

```
customers2023 = spark.read.csv('customers2023.csv', header=True, inferSchema=True)
customers2024 = spark.read.csv('customers2024.csv', header=True, inferSchema=True)
```

5. Створимо колонку, що визначає рік реєстрації клієнта (в залежності від назви файлу):

Код

```
customers2023 = customers2023.withColumn('year', lit(2023))
customers2024 = customers2024.withColumn('year', lit(2024))
```

6. Об'єднаємо отримані кадри даних.

Код

```
customers = customers2023.union(customers2024)
```

7. Приберемо дублікати. Для клієнтів з однаковим id оберемо максимальну вартість підписки.

Код

```
customers = customers.groupBy(customers.id).max()
```

8. Відобразимо отриманий кадр даних.

Код

```
customers.show()
```

Вивід

```
+---+-------+-----------------+---------+
| id|max(id)|max(subscription)|max(year)|
+---+-------+-----------------+---------+
|  1|      1|            19.99|     2023|
|  6|      6|            24.99|     2023|
|  3|      3|            29.99|     2024|
|  5|      5|             12.0|     2024|
|  4|      4|             13.0|     2024|
|  7|      7|            39.99|     2023|
|  2|      2|             9.99|     2023|
| 13|     13|            29.99|     2024|
|  8|      8|            14.99|     2024|
| 10|     10|            24.99|     2024|
| 14|     14|            39.99|     2024|
+---+-------+-----------------+---------+
```

9. Подальший аналіз показав, що інтереси клієнтів було записано в окремий файл customer_interests.csv (зміст файлу показано нижче). Додамо їх в поточний кадр даних.

customer_interests.csv

```
id,interests
1,"Reading, Music, Traveling"
2,"Cooking, Photography, Writing"
3,"Gaming, Theater, Painting, Yoga"
4,"Dancing, Karaoke, Sports, Art"
5,"Sports, Art, Traveling, Hiking"
6,"Yoga, Meditation, Reading, Writing"
7,"Swimming, Kayaking, Cycling, Biking"
8,"Painting, Sculpture, Gardening, Hiking"
10,"Painting, Sculpture, Reading, Writing"
13,"Rock Climbing, Swimming, Kayaking"
14,"Surfing, Snowboarding, Cycling, Biking"
```

Код

```
customers_interests = spark.read.csv('customers_interests.csv', header=True,
inferSchema=True)
customers = customers.join(customers_interests, 'id')
customers.show()
```

Вивід

```
+---+-------+----------------+---------+--------------------+
| id|max(id)|max(subscription)|max(year)|           interests|
+---+-------+----------------+---------+--------------------+
|  1|      1|           19.99|     2023|Reading, Music, T...|
|  6|      6|           24.99|     2023|Yoga, Meditation,...|
|  3|      3|           29.99|     2024|Gaming, Theater, ...|
|  5|      5|            12.0|     2024|Sports, Art, Trav...|
|  4|      4|            13.0|     2024|Dancing, Karaoke,...|
|  7|      7|           39.99|     2023|Swimming, Kayakin...|
|  2|      2|            9.99|     2023|Cooking, Photogra...|
| 13|     13|           29.99|     2024|Rock Climbing, Sw...|
|  8|      8|           14.99|     2024|Painting, Sculptu...|
| 10|     10|           24.99|     2024|Painting, Sculptu...|
| 14|     14|           39.99|     2024|Surfing, Snowboar...|
+---+-------+----------------+---------+--------------------+
```

10. Також виявилось, що ідентифікатори клієнтів не відповідають формату основної бази даних. Зчитаємо файл customer_ids.csv для перетворення ідентифікаторів

customer_ids.csv

```
id,new_id
1,5cafe5f8e6a0a21d4b2a8f7c
2,5cae9b3ce6a0a21d4b2a8f76
3,5cd0ff8de6a0a21d4b2a8f74
4,5cbaf8e1e6a0a21d4b2a8f7b
5,5cc9c0e4e6a0a21d4b2a8f79
6,5cd3bbaee6a0a21d4b2a8f75
7,5caed1cbe6a0a21d4b2a8f78
8,5ce3e9a4e6a0a21d4b2a8f77
10,5cb8c4b3e6a0a21d4b2a8f7a
13,5cd1db45e6a0a21d4b2a8f72
14,5cf3ba9de6a0a21d4b2a8f7d
```

Код

```
customer_ids = spark.read.csv('customer_ids.csv', header=True, inferSchema=True)
customer_ids.show()
```

Вивід

```
+---+--------------------+
| id|              new_id|
+---+--------------------+
|  1|5cafe5f8e6a0a21d4...|
|  2|5cae9b3ce6a0a21d4...|
|  3|5cd0ff8de6a0a21d4...|
|  4|5cbaf8e1e6a0a21d4...|
|  5|5cc9c0e4e6a0a21d4...|
|  6|5cd3bbaee6a0a21d4...|
|  7|5caed1cbe6a0a21d4...|
|  8|5ce3e9a4e6a0a21d4...|
| 10|5cb8c4b3e6a0a21d4...|
| 13|5cd1db45e6a0a21d4...|
| 14|5cf3ba9de6a0a21d4...|
+---+--------------------+
```

11.Додамо нові ідентифікатори до основного кадру даних

Код

```
customers = customers.join(customer_ids, 'id')
customers.show()
```

Вивід

```
+---+-------+----------------+---------+--------------------+--------------------+
| id|max(id)|max(subscription)|max(year)|           interests|              new_id|
+---+-------+----------------+---------+--------------------+--------------------+
|  1|      1|           19.99|     2023|Reading, Music, T...|5cafe5f8e6a0a21d4...|
|  6|      6|           24.99|     2023|Yoga, Meditation,...|5cd3bbaee6a0a21d4...|
|  3|      3|           29.99|     2024|Gaming, Theater, ...|5cd0ff8de6a0a21d4...|
|  5|      5|            12.0|     2024|Sports, Art, Trav...|5cc9c0e4e6a0a21d4...|
|  4|      4|            13.0|     2024|Dancing, Karaoke,...|5cbaf8e1e6a0a21d4...|
|  7|      7|           39.99|     2023|Swimming, Kayakin...|5caed1cbe6a0a21d4...|
|  2|      2|            9.99|     2023|Cooking, Photogra...|5cae9b3ce6a0a21d4...|
| 13|     13|           29.99|     2024|Rock Climbing, Sw...|5cd1db45e6a0a21d4...|
|  8|      8|           14.99|     2024|Painting, Sculptu...|5ce3e9a4e6a0a21d4...|
| 10|     10|           24.99|     2024|Painting, Sculptu...|5cb8c4b3e6a0a21d4...|
| 14|     14|           39.99|     2024|Surfing, Snowboar...|5cf3ba9de6a0a21d4...|
+---+-------+----------------+---------+--------------------+--------------------+
```

12.Оберемо колонки, необхідні для БД MongoDB та перейменуємо їх.

Код

```
customers = customers.select('new_id', 'max(subscription)', 'max(year)',
'interests')
customers = customers.withColumnRenamed('new_id', '_id')
customers = customers.withColumnRenamed('max(subscription)', 'subscription')
customers = customers.withColumnRenamed('max(year)', 'year')
customers.show()
```

Вивід

```
+--------------------+------------+----+--------------------+
|                 _id|subscription|year|           interests|
+--------------------+------------+----+--------------------+
|5cafe5f8e6a0a21d4...|       19.99|2023|Reading, Music, T...|
|5cd3bbaee6a0a21d4...|       24.99|2023|Yoga, Meditation,...|
|5cd0ff8de6a0a21d4...|       29.99|2024|Gaming, Theater, ...|
|5cc9c0e4e6a0a21d4...|        12.0|2024|Sports, Art, Trav...|
|5cbaf8e1e6a0a21d4...|        13.0|2024|Dancing, Karaoke,...|
|5caed1cbe6a0a21d4...|       39.99|2023|Swimming, Kayakin...|
|5cae9b3ce6a0a21d4...|        9.99|2023|Cooking, Photogra...|
|5cd1db45e6a0a21d4...|       29.99|2024|Rock Climbing, Sw...|
|5ce3e9a4e6a0a21d4...|       14.99|2024|Painting, Sculptu...|
|5cb8c4b3e6a0a21d4...|       24.99|2024|Painting, Sculptu...|
|5cf3ba9de6a0a21d4...|       39.99|2024|Surfing, Snowboar...|
+--------------------+------------+----+--------------------+
```

13.Вивантажимо дані в MongoDB. На рис. 9.1. показано вивантажені дані.

Код

```
customers.write.format("mongodb").option("database",
"productdb").option("collection", "customers").mode("overwrite").save()
```

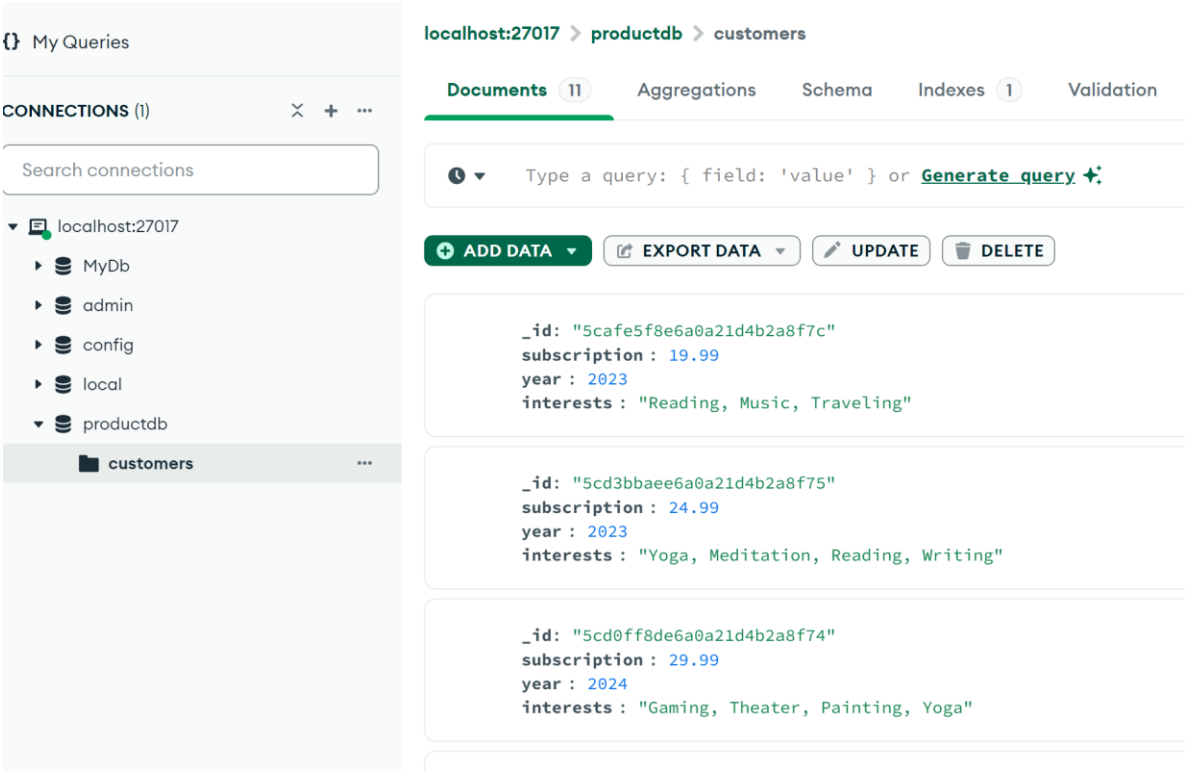

Рис. 9.1. Оброблені та вивантажені дані в MongoDB.

Тепер, коли ми успішно інтегрували наші клієнтські дані з різних джерел в єдину уніфіковану базу даних за допомогою процесу ETL, ми можемо почати використовувати можливості машинного навчання (ML), щоб отримати глибше розуміння і приймати обґрунтовані рішення. Для цього можливо використати підходи класифікації та регресії, що були вже нами розглянуті на попередніх лекціях.

9.3. Контрольні питання

1. Як спільне використання баз даних з PySpark дозволяє покращувати аналіз та передбачення на даних?
2. В яких ситуаціях базового інструменту імпорту в MongoDB недостатньо та необхідно використовувати PySpark?
3. Що таке ETL? Які етапи в себе включає ETL?
4. Чому підхід ETL набув популярності? Яке місце в реалізації цього підходу займає PySpark?
5. Які функції необхідно використати для експорту даних з довільного джерела в MongoDB засобами PySpark?
6. Яку функцію PySpark необхідно використати для об’єднання даних з декількох джерел подібної структури?
7. Поясніть в чому подібності та відмінності між підходами ETL та ELT? Який підхід є доречним для якої задачі?

Рекомендовані джерела інформації

1. Хабарлак К.С. Аналіз та обробка великих даних [Електронний ресурс] : методичні рекомендації до виконання практичних робіт для здобувачів ступеня магістра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 82 с.
2. Pedro Duarte Faria. Introduction to pyspark. – 2024. [Online] URL: https://pedropark99.github.io/Introd-pyspark/
3. Apache Spark documentation [Online]. URL: https://spark.apache.org/docs/latest/index.html
4. MongoDB documentation [Online]. URL: https://www.mongodb.com/docs/
5. Python MongoDB [Online]. URL: https://www.w3schools.com/python/python_mongodb_getstarted.asp
6. D. Paper Data Science Fundamentals for Python and MongoDB / Apress Media – 2018. – 201c.
7. Хабарлак К.С. Самонавчання складних систем [Електронний ресурс] : конспект лекцій для здобувачів ступеня магістра освітньо-професійної програми «Системний аналіз» зі спеціальності 124 Системний аналіз / К.С. Хабарлак, Т.А. Желдак ; М-во освіти і науки України, Нац. техн. ун-т «Дніпровська політехніка». – Дніпро : НТУ «ДП», 2024. – 112 с.
8. K. Khabarlak Tomato Plant Treatment Smart Suggestions Using Big Data Analytics // Materials of the XII International Scientific-Practical Conference «Information Control Systems and Technologies», Odesa, Ukraine, 2024. – p. 214-216.
9. Practical Statistics for Data Scientists / P. Bruce, A. Bruce, P. Gedeck // O'Reilly Media, 2020.

Навчальне видання

Хабарлак Костянтин Сергійович
Хом'як Тетяна Валеріївна

АНАЛІЗ ТА ОБРОБКА ВЕЛИКИХ ДАНИХ

Конспект лекцій
для здобувачів ступеня магістра
освітньо-професійної програми «Системний аналіз»
зі спеціальності 124 Системний аналіз

Видано в авторській редакції.

Електронний ресурс.
Підписано до видання 12.11.2024. Авт. арк. 7,9.

Національний технічний університет «Дніпровська політехніка».
49005, м. Дніпро, просп. Дмитра Яворницького, 19.