Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Економічна аналітика великих даних

Конспект лекцій

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня магістра
за освітньою програмою «Економічна аналітика»
спеціальності 051 Економіка

Укладач: І. С. Лазаренко

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2024

УДК 519.7

Укладач: *Лазаренко Ірина Сергіївна*, к.ф.-м. н.

Рецензент *Пишнограєв І.О.*, к.ф.-м.н., доц.
доцент кафедри штучного інтелекту
КПІ ім. Ігоря Сікорського

Відповідальний
редактор *Бояринова К. О*, д-р. економ. наук, проф., завідувачка кафедри
економічної кібернетикт КПІ ім. Ігоря Сікорського

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 7 від 09.05. 2024 р.)
за поданням вченої ради факультету менеджменту та маркетингу
(протокол № 9 від 29.04. 2024 р.)

Економічна аналітика великих даних: конспект лекцій [Електронний ресурс] : навч. посіб. для здобувачів ступеня магістра за освіт. програмою «Економічна аналітика» спец. 051 Економіка / КПІ ім. Ігоря Сікорського ; уклад.: І. С. Лазаренко. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2024. – 93 с.

Навчальний посібник відповідає силабусу дисципліни «Економічна аналітика великих даних» спеціальності 051 «Економіка» освітньої програми «Економічна аналітика» підготовки студентів факультету менеджменту та маркетингу. У навчальному посібнику викладено всі теми навчальної дисципліни, основні поняття та опорний конспект лекцій. Навчальний посібник призначений для студентів ФММ КПІ ім. Ігоря Сікорського спеціальності 051 Економіка.

УДК 519.7

Реєстр. № НП 23/24-508 . Обсяг 3 авт. арк.
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ПЕРЕДМОВА

Навчальний посібник «Економічна аналітика великих даних: конспект лекцій. включає загальні поняття, термінологію, методи та способи застосування технологій аналізу та алгоритмів великих даних в економіці. Посібник має на меті формування у студентів системних знань сучасних технологій для аналізу економічних моделей, розуміння принципів структурного підходу до підготовки та аналізу великих даних за допомогою загальних принципів та алгоритмів Big Data аналізу, а також розробленого функціоналу бібліотек різних мов.

Посібник спрямований на поглиблення знань з економічної аналітики великих даних, професійних навичок з питань економіко-математичного моделювання, якісної обробки великих даних на первинних етапах моделювання.

Навчальний посібник містить лекційний матеріал до тем, визначених у відповідній освітній компоненті, які спрямовані на розкриття та висвітлення термінології, основних принципів та парадигм, закладених в концепцію Big Data аналізу через призму економічної інформації. В лекціях розкриваються основні методи та принципи підготовки та аналізу великих масивів даних, методи визначення та боротьби з розповсюдженими проблемами, які впливають на якість результатів моделювання та надається вичерпний інструментарій для вивчення ключових аспектів аналітики великих даних та розвитку відповідних фахових компетенцій.

Посібник призначено для студентів денної, інтегральної та заочної форми навчання рівня вищої освіти ступеня «магістр». Його можна використати також для підготовки до занять, заліків, екзаменів студентам усіх форм навчання, які вивчають подібний матеріал. Курс відповідає сучасним ринковим потребам підготовки сучасних фахівців.

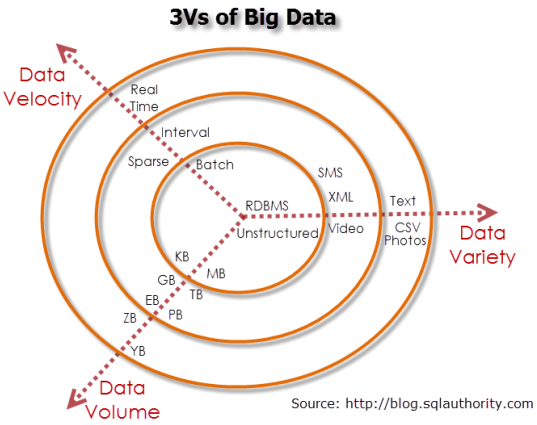
Лекція 1

Загальні відомості про великі дані. Основні особливості та огляд методів роботи з великими даними.

Термін "**великі дані**" описує величезну колекцію структурованих і неструктурованих даних, яка продовжує зростати в геометричній прогресії зі збільшенням рівня оцифрування. Однак через величезний обсяг і складність великих даних традиційне програмне забезпечення для обробки даних не здатне обробляти і витягувати з них корисну інформацію. Саме тому сьогодні підприємства звертаються до технологій великих даних.

З появою технологій великих даних компанії отримали можливість зберігати, обробляти та аналізувати величезні обсяги даних з метою виявлення релевантної інформації. На сьогодні існує цілий ряд надійних технологій великих даних, з яких можна вибирати. Питання полягає в тому, які з цих технологій будуть найбільш перспективними в майбутньому? Які інструменти великих даних можуть принести вам суттєві переваги?

Що таке технологія великих даних?



Термін "**великі дані**" вже багато років перебуває в моді. Під "великими даними" мається на увазі ***великий обсяг*** (**Volume**), ***швидкість*** (**Velocity**) і ***різноманітність інформаційних активів*** (**Variety**), які вимагають не традиційних методів обробки даних, а економічно ефективних, інноваційних методів обробки даних для отримання більш глибокого розуміння і прийняття рішень. Тому компанії впроваджують технології великих даних, щоб отримати більше інформації та приймати більш вигідні рішення. Технології великих даних визначаються як програмні утиліти, які в першу чергу призначені для аналізу,

обробки та вилучення інформації з великих масивів даних з надзвичайно складною структурою, які не можуть бути оброблені традиційними технологіями обробки даних.

Поява технологій великих даних почала заповнювати прогалину між традиційними технологіями роботи з даними (такими як СУБД, файлові системи тощо) та потребами бізнесу в даних, що швидко зростають. По суті, ці технології включають в себе специфічні структури даних, методи, інструменти та прийоми, що використовуються для зберігання, вивчення, ремоделювання, аналізу та оцінки даних. Величезні обсяги даних, що надходять у режимі реального часу, необхідно аналізувати за допомогою технологій обробки великих даних, щоб зробити висновки і прогнози, які допоможуть зменшити майбутні ризики. Такі можливості стають все більш важливими в епоху Інтернету.

Типи технологій великих даних

Загалом технології великих даних можна розділити на дві категорії:

Оперативні технології великих даних

Оперативні великі дані - це всі дані, які ми генеруємо в результаті повсякденної діяльності, наприклад, інтернет-транзакції, соціальні медіа-платформи або будь-яка інформація від конкретної компанії. Ці дані слугують сировиною для аналізу за допомогою оперативних технологій великих даних. Деякі приклади оперативних технологій великих даних включають

- Система онлайн-бронювання квитків, наприклад, на поїзди, рейси, автобуси, кіно тощо.
- Онлайн-торгівля або покупки на веб-сайтах електронної комерції, таких як Flipkart, Amazon, Myntra тощо.
- Онлайн-дані з соціальних мереж, таких як Instagram, Facebook, Messenger, Whatsapp тощо.
- Дані про співробітників або керівників у транснаціональних компаніях.

Аналітичні технології великих даних

Аналітичні великі дані можна розглядати як модифікований варіант технологій великих даних, який є більш складним, ніж оперативні великі дані. Аналітичні великі дані, як правило, застосовуються, коли йдеться про показники

ефективності, а також коли необхідно приймати критичні бізнес-рішення на основі звітів, отриманих в результаті оперативного аналізу великих даних. Таким чином, цей тип технологій великих даних стосується аналізу великих даних, що мають відношення до бізнес-рішень. Деякі приклади аналітичних технологій великих даних включають

- Біржові маркетингові дані.
- Дані прогнозу погоди.
- Медичні записи дозволяють лікарям контролювати стан здоров'я пацієнта.
- Ведення баз даних космічних місій, де важлива кожна деталь про місію.

Hard Skills

Навички повинні випливати з програмних і технологічних та операційних навичок. Нижче ми розглянемо основні навички. Цими навичками повинен володіти провідний аналітик з великих даних:

- Розуміти алгоритми, структури даних і вміти готувати дані для аналізу;
- Використовувати платформи для самообслуговування аналітики;
- Правильно застосовувати статистику та математику;
- Мати знання та досвід застосування:
    - Машинне навчання;
    - Data API;
    - ETL-інструментів;
    - Систем баз даних;
    - Рішень для зберігання даних.
- Розуміння основ розподілених систем.

Soft Skills

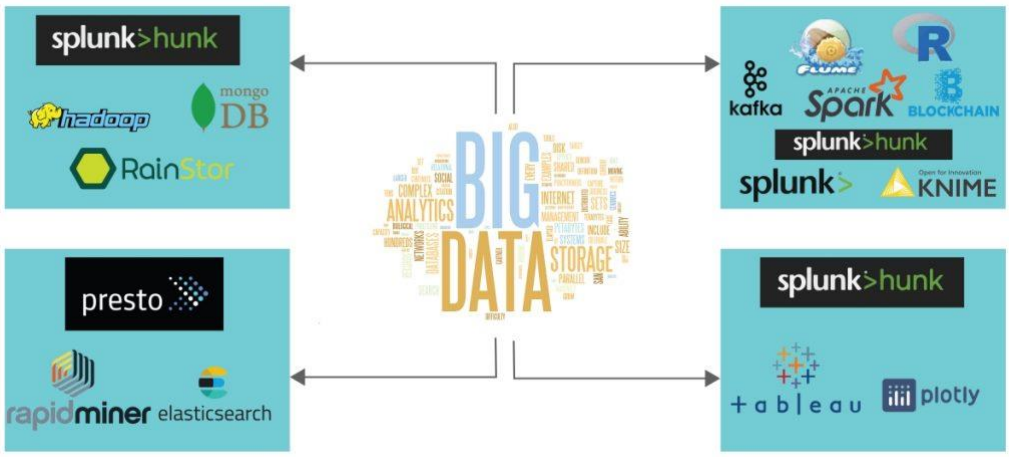
Ці знання допомагають зрозуміти, чи можна добре вписатися в культуру компанії чи ні:

Найкращі технології великих даних

Останнім часом багато технологій великих даних вплинули на ринок та IT-індустрію. Їх можна розділити на чотири великі категорії, а саме

- Data Storage
- Data Mining
- Data Analytics
- Data Visualization

Розглянемо технології, що підпадають під кожну з цих категорій, з фактами та особливостями, а також компанії, які їх використовують.



Зберігання даних (Data Storage)

Як правило, цей тип технологій великих даних включає в себе інфраструктуру, яка дозволяє отримувати, зберігати та управляти даними, і призначена для роботи з великими обсягами даних. Різноманітне програмне забезпечення дозволяє легко і швидко отримувати доступ до зібраних даних, використовувати їх і обробляти. Серед найбільш поширених технологій великих даних для цієї мети є такі:

1. Apache Hadoop

Apache Hadoop – це фреймворк з відкритим вихідним кодом на основі Java для зберігання та обробки великих даних, розроблений Apache Software Foundation. По суті, він забезпечує розподілену платформу для зберігання та обробки великих даних за допомогою програмної моделі MapReduce. Фреймворк Hadoop призначений для автоматичної обробки апаратних збоїв, оскільки вони часто трапляються. Фреймворк Hadoop складається з **п'яти модулів**, а саме: Hadoop Distributed File System (HDFS), Hadoop YARN (Yet Another Resource Negotiator), Hadoop MapReduce, Hadoop Common та Hadoop Ozone.

Компанії, що використовують Hadoop: LinkedIn, Intel, IBM, MapR, Facebook, Microsoft, Hortonworks, Cloudera та інші.

Ключові особливості:

Розподілена файлова система HDFS (Hadoop Distributed File System) забезпечує швидку передачу даних між вузлами. HDFS є принципово відмовостійкою файловою системою. У Hadoop дані, які зберігаються на одному вузлі, також реплікуються на інші вузли кластера, щоб запобігти втраті даних у разі апаратного або програмного збою.

Hadoop - це недорогий, відмовостійкий і надзвичайно гнучкий фреймворк, здатний зберігати і обробляти дані в будь-якому форматі (структурованому, напівструктурованому або неструктурованому).

MapReduce - це вбудований в Hadoop механізм пакетної обробки, який розподіляє великі обчислення між декількома вузлами для забезпечення оптимальної продуктивності та балансування навантаження.

2. MongoDB

MongoDB – це крос-платформна, документно-орієнтована база даних з відкритим вихідним кодом, призначена для зберігання та обробки великих обсягів даних, забезпечуючи при цьому високу доступність, продуктивність та

масштабованість. Оскільки MongoDB не зберігає і не отримує дані у вигляді таблиць, вона вважається базою даних NoSQL. Новачок у сфері зберігання даних, MongoDB користується великою популярністю завдяки своїм документно-орієнтованим функціям NoSQL, розподіленому сховищу ключів-значень та можливостям обчислень Map Reduce.

Компанії, що використовують MongoDB: MySQL, Facebook, eBay, MetLife, Google, Shutterfly, Aadhar тощо.

Ключові особливості:

Легко інтегрується з такими мовами, як Ruby, Python та JavaScript; ця безшовна інтеграція забезпечує високу швидкість кодування.

База даних MongoDB зберігає дані в документах JSON, які забезпечують багату модель даних, що легко відображається на рідні мови програмування.

MongoDB має кілька функцій, недоступних у традиційних СКБД, таких як динамічні запити, вторинні індекси, багаті оновлення, сортування та легке агрегування.

У документальних системах баз даних пов'язані дані зберігаються в одному документі, що дозволяє виконувати запити швидше, ніж у традиційній реляційній базі даних, де пов'язані дані зберігаються в декількох таблицях і пізніше об'єднуються за допомогою об'єднань.

3. RainStor

RainStor – це СУБД, яка управляє та аналізує великі дані, розроблена компанією RainStor. Для оптимізації зберігання великих обсягів даних для довідок використовується техніка дедуплікації. Завдяки своїй здатності сортувати та зберігати великі обсяги інформації для довідок, вона усуває дублікати файлів. Крім того, він підтримує хмарне зберігання та багатокористувацьку оренду. Продукт для роботи з базами даних RainStor доступний у двох редакціях: Big Data Retention та Big Data Analytics на Hadoop, які забезпечують високоефективне управління даними та прискорюють аналіз даних і виконання запитів.

Компанії, що використовують RainStor: Barclays, Reimagine Strategy, Credit Suisse та інші.

Ключові особливості:

За допомогою RainStor великі підприємства можуть керувати та аналізувати великі дані з найменшими загальними витратами.

Корпоративна база даних побудована на Hadoop для підтримки швидшої аналітики.

Це дозволяє запускати швидші запити та аналізи, використовуючи як SQL запити, так і MapReduce, що призводить до 10-100 разів швидших результатів.

RainStor забезпечує найвищий рівень стиснення. Дані стискаються до 40 разів (97,5%) або більше порівняно з необробленими даними і не потребують повторного роздування при доступі до них.

4. Cassandra

Cassandra – це розподілена NoSQL-база даних з відкритим вихідним кодом, яка дозволяє проводити глибокий аналіз багатьох наборів даних у режимі реального часу. Вона забезпечує високу масштабованість і доступність без компромісів у продуктивності. Для взаємодії з базою даних використовується мова CQL (Cassandra Structure Language). Завдяки масштабованості та відмовостійкості на хмарній інфраструктурі або товарному обладнанні, це ідеальна платформа для обробки критично важливих даних. Як основний інструмент для роботи з великими даними, вона підтримує всі типи форматів даних, включаючи структуровані, напівструктуровані та неструктуровані.

Компанії, що використовують Cassandra: Facebook, GoDaddy, Netflix, GitHub, Rackspace, Cisco, Hulu, eBay тощо.

Ключові особливості:

Децентралізована архітектура Cassandra запобігає виникненню єдиної точки відмови в кластері.

Чутливість даних робить Cassandra придатною для корпоративних додатків, які не можуть дозволити собі втрату даних, навіть якщо весь центр обробки

даних вийде з ладу. ACID (атомарність, узгодженість, ізоляція та довговічність) підтримуються Cassandra. Це дозволяє інтегрувати Hadoop з MapReduce. Вона також підтримує Apache Hive та Apache Pig.

Завдяки своїй масштабованості, Cassandra може бути масштабована для обслуговування більшої кількості клієнтів та більшого обсягу даних.

Інтелектуальний аналіз даних (Data Mining)

Інтелектуальний аналіз даних - це процес вилучення корисної інформації з необроблених даних та її аналізу. У багатьох випадках необроблені дані є дуже великими, дуже мінливими і постійно передаються зі швидкістю, яка робить вилучення даних майже неможливим без спеціальної техніки. Серед найпоширеніших технологій для інтелектуального аналізу даних - технології великих даних:

5. Presto

Розроблений компанією Facebook, Presto – це механізм запитів SQL з відкритим вихідним кодом, який дозволяє проводити інтерактивний аналіз великих обсягів даних. Ця розподілена пошукова система підтримує швидкі аналітичні запити до джерел даних різного розміру, від гігабайтів до петабайтів. Завдяки цій технології можна запитувати дані безпосередньо там, де вони знаходяться, без переміщення даних в окремі аналітичні системи. Можна навіть запитувати дані з декількох джерел в рамках одного запиту. Він підтримує як реляційні джерела даних (такі як PostgreSQL, MySQL, Microsoft SQL Server, Amazon Redshift, Teradata тощо), так і нереляційні джерела даних (такі як HDFS (Hadoop Distributed File System), MongoDB, Cassandra, HBase, Amazon S3 тощо).

Компанії, що використовують Presto: Repro, Netflix, Facebook, Airbnb, GrubHub, Nordstrom, Nasdaq, Atlassian та інші.

Основні можливості:

За допомогою Presto ви можете робити запити до даних, де б вони не знаходилися: в Cassandra, Hive, реляційних базах даних або навіть у власних

сховищах даних. За допомогою Presto можна робити запити до декількох джерел даних одночасно. Це дозволяє посилатися на дані з декількох баз даних в одному запиті.

Він не покладається на методи MapReduce і здатний отримувати дані дуже швидко, від декількох секунд до декількох хвилин. Відповіді на запити зазвичай повертаються протягом декількох секунд.

Presto підтримує стандартний ANSI SQL, що робить його простим у використанні. Можливість запитувати дані без вивчення спеціальної мови завжди є великим плюсом, незалежно від того, чи ви розробник, чи аналітик даних. Крім того, він легко підключається до найпоширеніших інструментів BI (Business Intelligence) за допомогою роз'ємів JDBC (Java Database Connectivity).

6. RapidMiner

RapidMiner – це передовий інструмент інтелектуального аналізу даних з відкритим вихідним кодом для предиктивної аналітики. Це потужна наукова платформа, яка дозволяє аналітикам даних та аналітикам великих даних швидко аналізувати свої дані. На додаток до інтелектуального аналізу даних, він дозволяє розгортати моделі та керувати ними. Завдяки цьому рішенню ви отримаєте доступ до всіх можливостей машинного навчання та підготовки даних, необхідних для впливу на ваші бізнес-операції. Надаючи уніфіковане середовище для підготовки даних, машинного навчання, глибокого навчання, інтелектуального аналізу тексту та предиктивної аналітики, воно спрямоване на підвищення продуктивності для корпоративних користувачів будь-якого рівня кваліфікації.

Компанії, що використовують RapidMiner: Domino's Pizza, McKinley Marketing Partners, Windstream Communications, George Mason University та інші.

Ключові особливості:

Інтегрована платформа для обробки даних, побудови моделей машинного навчання та їх розгортання. Крім того, він інтегрує фреймворк Hadoop з вбудованим RapidMiner Radoop. RapidMiner Studio забезпечує доступ,

завантаження та аналіз будь-якого типу даних, незалежно від того, чи це структуровані дані, чи неструктуровані, такі як текст, зображення та медіа.

У RapidMiner доступне автоматизоване прогнозне моделювання.

7. ElasticSearch

Побудований на Apache Lucene, Elasticsearch - це сучасна пошуково-аналітична система з відкритим вихідним кодом, яка дозволяє шукати, індексувати та аналізувати дані всіх типів. Деякі з найпоширеніших випадків його використання включають аналіз журналів, розвідку безпеки, оперативну розвідку, повнотекстовий пошук і бізнес-аналітику. Неструктуровані дані з різних джерел витягуються і зберігаються у форматі, який добре оптимізований для мовного пошуку. Користувачі можуть легко шукати та досліджувати великий обсяг даних на дуже високій швидкості. DB-Engines вважає Elasticsearch найкращою пошуковою системою для підприємств.

Компанії, що використовують ElasticSearch: Netflix, Facebook, Uber, Shopify, Linkedln, StackOverflow, GitHub, Instacart та інші.

Ключові особливості:

Використовуючи ElasticSearch, ви можете зберігати та аналізувати структуровані та неструктуровані дані обсягом до петабайтів.

Надаючи прості RESTful API та JSON-документи без схем, Elasticsearch полегшує пошук, індексування та запити до даних.

Більше того, він забезпечує пошук майже в реальному часі, масштабований пошук і багатокористувацькі можливості.

Elasticsearch написана на Java, що робить її сумісною майже з усіма платформами.

Як додаток з відкритим вихідним кодом, Elasticsearch дозволяє легко розширювати його функціональність за допомогою плагінів та інтеграцій.

Для повного контролю над даними, кластерними операціями, користувачами, користувачами і т.д. передбачено кілька інструментів

управління, інтерфейсів користувача і API (інтерфейсів прикладного програмування).

Аналітика даних (Data Analytics)

Аналітика великих даних передбачає очищення, перетворення та моделювання даних з метою вилучення важливої інформації, яка допоможе в процесі прийняття рішень. Ви можете отримати цінну інформацію з необроблених даних за допомогою методів аналізу даних. Серед інформації, яку можуть надати інструменти аналізу великих даних, є приховані закономірності, кореляції, вподобання клієнтів та статистична інформація про ринок. Нижче наведено кілька типів технологій аналізу даних, з якими вам варто ознайомитися.

8. Kafka

Apache Kafka – це популярне сховище подій з відкритим вихідним кодом і потокова платформа, розроблена Apache Software Foundation на мовах Java і Scala. Платформа використовується тисячами організацій для потокової аналітики, високопродуктивних конвеєрів даних, інтеграції даних та критично важливих додатків. Це відмовостійка система обміну повідомленнями, заснована на моделі "публікація-передплатник", яка може обробляти величезні обсяги даних. Для аналізу потокових даних у реальному часі Apache Kafka можна легко інтегрувати з Apache Storm та Apache Spark. По суті, Kafka - це система для збору, зберігання, читання та аналізу потокових даних в масштабі.

Компанії, що використовують Kafka: Netflix, Goldman Sachs, Shopify, Target, Cisco, Spotify, Intuit, Uber та інші.

Ключові особливості:

З Apache Kafka масштабованість може бути досягнута в чотирьох вимірах: процесори подій, виробники подій, споживачі подій та з'єднувачі подій. Це означає, що Kafka масштабується без особливих зусиль і без простоїв.

Kafka дуже надійна завдяки своїй розподіленій архітектурі, розбиттю на розділи, реплікації та відмовостійкості. Ви можете публікувати та підписуватися на повідомлення з високою пропускною здатністю.

Система гарантує нульовий час простою та відсутність втрати даних.

9. Splunk

Splunk – це масштабована, просунута програмна платформа, яка шукає, аналізує та візуалізує машинні дані з веб-сайтів, додатків, датчиків, пристроїв тощо, щоб надавати метрики, діагностувати проблеми та отримувати уявлення про бізнес-операції. У Splunk дані в режимі реального часу збираються, індексуються та корелюються у сховищі з можливістю пошуку, яке можна використовувати для створення звітів, сповіщень, графіків, інформаційних панелей та візуалізацій. На додаток до управління додатками, безпеки та відповідності нормативним вимогам, Splunk також забезпечує веб-аналітику та бізнес-аналітику. Поява великих даних робить Splunk здатним поглинати великі дані з різних джерел, які можуть включати або не включати машинні дані, і виконувати на них аналітику.

Компанії, що використовують Splunk: JPMorgan Chase, Lenovo, Wells Fargo, Verizon, BookMyShow, John Lewis, Domino's, Porsche та інші. Дізнайтеся більше.

Ключові особливості:

Підвищуйте ефективність вашого бізнесу за допомогою автоматизованих операцій, розширеної аналітики та наскрізних інтеграцій.

На додаток до структурованих форматів даних, таких як JSON і XML, Splunk може поглинати неструктуровані машинні дані, такі як веб-журнали і журнали додатків.

Splunk індексує отримані дані, щоб забезпечити швидший пошук і запити на основі різних умов.

Splunk надає аналітичні звіти, включаючи інтерактивні графіки, діаграми і таблиці, а також дозволяє ділитися ними з іншими людьми.

10. KNIME

KNIME (Konstanz Information Miner) - це безкоштовна платформа з відкритим вихідним кодом для аналітики, звітності та інтеграції великих наборів даних. Окрім інтуїтивності та відкритості, KNIME активно впроваджує нові ідеї та розробки, щоб зробити розуміння даних та розробку робочих процесів і багаторазових компонентів для науки про дані максимально простими та доступними. KNIME дозволяє користувачам візуально створювати і проектувати потоки даних (або конвеєри), вибірково виконувати кроки аналізу, а потім аналізувати результати і моделі за допомогою інтерактивних представлень і віджетів. Основна версія містить сотні модулів для інтеграції, перетворення даних (наприклад, фільтри, конвертери, розгалужувачі, об'єднувачі та з'єднувачі), а також методи, що використовуються для аналітики, статистики, інтелектуального аналізу даних та текстової аналітики.

Компанії, що використовують KNIME: Fiserv, Opplane, Procter & Gamble, Eaton Corporation та інші.

Ключові особливості:

Додаткові плагіни додаються за допомогою механізму розширення для розширення функціональності.

Крім того, додаткові плагіни забезпечують інтеграцію методів інтелектуального аналізу зображень, текстового аналізу, аналізу часових рядів і мережевого аналізу.

Робочі процеси KNIME можуть слугувати наборами даних для створення шаблонів звітів, які можна експортувати в різні формати файлів, включаючи doc, pdf, ppt, xls тощо.

Крім того, KNIME інтегрує різноманітні проекти з відкритим вихідним кодом, такі як алгоритми машинного навчання з проектів Spark, Weka, Keras, LIBSVM і R; а також ImageJ, JFreeChart і Chemistry Development Kit.

За допомогою нього можна виконувати прості ETL-операції.

11. Apache Spark

Найважливіша і найочікуваніша технологія вже в полі зору - Apache Spark. Це аналітичний движок з відкритим вихідним кодом, який підтримує обробку великих даних. Ця платформа має обчислення в пам'яті (IMC) для виконання швидких запитів до даних будь-якого розміру; узагальнену модель виконання (GEM), яка підтримує широкий спектр додатків, а також Java, Python і Scala API для простоти розробки. Ці API дозволяють приховати складність розподіленої обробки за простими високорівневими операторами. Spark був представлений Apache Software Foundation для прискорення обчислень Hadoop.

Компанії, що використовують Presto: Amazon, Oracle, Cisco, Netflix, Yahoo, eBay, Hortonworks та інші.

Ключові особливості:

Платформа Spark дозволяє виконувати програми в 100 разів швидше в пам'яті, ніж Hadoop MapReduce, або в 10 разів швидше на диску.

За допомогою Apache Spark ви можете запускати безліч робочих навантажень, включаючи машинне навчання, аналітику в реальному часі, інтерактивні запити та обробку графіків.

Spark має зручні інтерфейси розробки (API), доступні на мовах Java, Scala, Python та R для роботи з великими наборами даних.

До складу Spark входить низка високорівневих бібліотек, таких як підтримка SQL запитів, машинного навчання, потокової передачі даних та обробки графіків.

Візуалізація даних (Data Visualization)

Візуалізація даних - це спосіб візуалізації даних за допомогою графічного представлення. Методи візуалізації даних використовують візуальні елементи, такі як графіки, діаграми та карти, щоб забезпечити простий спосіб перегляду та інтерпретації тенденцій, закономірностей та відхилень у даних. Дані обробляються для створення графічних ілюстрацій, які дозволяють людям

зрозуміти великі обсяги інформації за лічені секунди. Нижче наведено кілька найкращих технологій для візуалізації даних.

12. Tableau

У галузі бізнес-аналітики та аналітики Tableau є найбільш швидкозростаючим інструментом для візуалізації даних. Він дозволяє користувачам легко створювати графіки, діаграми, карти та інформаційні панелі для візуалізації та аналізу даних, допомагаючи їм у просуванні бізнесу вперед. Використовуючи цю платформу, дані швидко аналізуються, в результаті чого створюються інтерактивні інформаційні панелі та робочі аркуші, які відображають результати. За допомогою Tableau користувачі можуть працювати з реальними наборами даних, отримуючи цінну інформацію та покращуючи процес прийняття рішень. Для початку роботи не потрібні знання з програмування; навіть ті, хто не має відповідного досвіду, можуть одразу створювати візуалізації за допомогою Tableau.

Компанії, що використовують Tableau: Accenture, Myntra, Nike, Skype, Coca-Cola, Wells Fargo, Citigroup, Qlik та інші.

Основні можливості:

У Tableau користувач може легко створювати візуалізації у вигляді гістограм, кругових діаграм, гістограм, деревовидних карт, бокс-діаграм, діаграм Ганта, точкових діаграм та інших інструментів.

Tableau підтримує широкий спектр джерел даних, включаючи локальні файли, CSV, текстові файли, Excel, електронні таблиці, реляційні та нереляційні бази даних, хмарні дані та великі дані.

Деякі з важливих функцій Tableau включають об'єднання даних та аналітику в реальному часі.

Це дозволяє обмінюватися даними в режимі реального часу у вигляді інформаційних панелей, таблиць тощо.

13. Plotly

Plotly – це бібліотека Python, яка полегшує інтерактивну візуалізацію великих даних. Цей інструмент дозволяє створювати чудові графіки швидше та ефективніше. Plotly має багато переваг, серед яких зручність у використанні, масштабованість, зниження витрат, передова аналітика та гнучкість. Він пропонує набагато багатший набір бібліотек та API, включаючи Python, R, MATLAB, Arduino, Julia тощо. Його можна використовувати в інтерактивному режимі в блокнотах Jupyter та Pycharm для створення інтерактивних графіків. За допомогою Plotly ми можемо включати інтерактивні функції, такі як кнопки, повзунки та випадаючі списки, щоб відображати різні перспективи на графіку.

Компанії, що використовують Plotly: Paladins, Bitbank та інші.

Ключові особливості:

Унікальною особливістю Plotly є його інтерактивність. Користувачі можуть взаємодіяти з графіками на екрані, забезпечуючи покращений досвід сторітелінгу.

Це як малювання на папері, ви можете намалювати все, що завгодно. У порівнянні з іншими інструментами візуалізації, такими як Tableau, Plotly забезпечує повний контроль над тим, що будується.

На додаток до діаграм Seaborn і Matplotlib, Plotly також пропонує широкий спектр графіків і діаграм, таких як статистичні діаграми, наукові діаграми, фінансові діаграми, географічні карти і так далі.

Крім того, Plotly пропонує широкий вибір діаграм ШІ та ML, які дозволять вам розширити можливості машинного навчання.

Лекція 2

Огляд технології обробки та збереження даних.

Огляд технологій Hadoop.

На початку нового тисячоліття, коли стартап Google, намагався проіндексувати весь інтернет, який зростав, перед ними стояли два основні виклики, які раніше ще ніхто не вирішував:

1. Як розмістити сотні терабайт даних на тисячах дисків, встановлених у більш ніж тисячі машин, без даунтаймів, втрати інформації та зі збереженням її постійної доступності?
2. Як розпаралелити обчислення ефективним і відмовостійким способом для обробки всіх цих даних на всіх машинах?


З 2003 по 2006 роки Google випустили три дослідницькі роботи, що пояснюють внутрішню архітектуру даних. Ці роботи назавжди змінили індустрію Big Data. Перша вийшла 2003 року під назвою "The Google File System". Друга пішла за нею 2004 року і називалася "MapReduce: Simplified Data Processing on Large Clusters". Згідно з Google Scholar, відтоді її процитували понад 21 000 разів. Третя наукова праця вийшла 2006 року і називалася "Bigtable: A Distributed Storage System for Structured Data".

Автори цих праць Джефф Дін та Санджай Гемават

Джефф Дін – американський інженер-програміст і вчений у галузі інформатики. Завдяки йому на світ з'явилося безліч "закулісних" продуктів Google, які допомогли корпорації пробитися на трон всього інтернету. Крім того Джефф, співзасновник компанії і глава команди інженерів Google з глибокого навчання, широко відомий своїми доповідями на тему технологій, інновацій та штучного інтелекту.

Працюючи в Compaq, Джефф брав участь у створенні інфраструктури безперервного профілювання і винайшов ProfileMe, апаратну технологію моніторингу продуктивності мікропроцесорів. Крім того, він встиг докласти руку до розробки та впровадження Swift, однієї з найшвидших у світі реалізацій Java. Також він був провідним технічним співробітником mySimon Inc., колись популярного сервісу для порівняння покупок. Він зробив свій внесок у створення Google News і AdSense, сервісу, який докорінно змінив світ реклами та інтернет-економіки. Однак незабаром після цього йому довелося зосередитися на вирішенні однієї з головних проблем Google - масштабування.

Разом із видатним програмістом Санджаєм Гемаватом він створив для Google особливу файлову систему, яка давала змогу ефективно розподіляти величезні обсяги даних по тисячах дешевих машин.

Потім у тандемі вони розробили революційний інструмент MapReduce, модель розподілених обчислень, використовувану в технологіях Big Data для паралельних обчислень, які виконують над дуже великими (до кількох петабайт) наборами даних. Після виходу в 2004 році статті MapReduce: Simplified Data Processing on Large Clusters інструмент став сенсацією і буквально підірвав галузь!

Джефф Дін був співзасновником Google Translate. Крім того, він зробив серйозний внесок у створення Google News. Сьогодні Дін очолює підрозділ Google AI і обіймає в компанії посаду Senior Fellow (еквівалент Senior Vice President) підрозділу Google Research and Health.

Засновник Hadoop, співробітник компанії Yahoo! Дуг Каттінг, який на той момент вже розробив Apache Lucene (пошукова бібліотека, що лежить в основі Apache Solr і ElasticSearch), працював над проектом сильно розподіленого пошукового модуля під назвою Apache Nutch. Подібно до Google, цей проєкт для досягнення широкого масштабу потребував розподіленого сховища і серйозних обчислювальних можливостей. Прочитавши роботи Google з Google File System і MapReduce, Дуг усвідомив

помилковість свого поточного підходу, а описана в тих роботах архітектура надихнула його на створення 2005 року дочірнього проекту для Nutch, який на честь іграшки (жовтого слона) свого сина він назвав Hadoop.

Разом із Дугом Каттінгом Майк Кафарелла є одним із перших співзасновників проєктів із відкритим вихідним кодом Hadoop і Nutch

Проєкт розпочався з двох ключових компонентів: розподіленої файлової системи Hadoop (HDFS) і реалізації фреймворка MapReduce. На відміну від Google, компанія Yahoo! вирішила відкрити вихідний код проекту в рамках Apache Software Foundation. У такий спосіб запросили всі інші провідні компанії до його використання та участі в розвитку, щоб скоротити технологічне відставання від своїх сусідів (Yahoo! розташована в Саннівейлі поруч із Маунтін-В'ю).

Досить скоро інші компанії, почавши використовувати Hadoop, зіткнулися з аналогічними проблемами обробки великих обсягів даних. Це означало величезні зобов'язання, оскільки їм потрібно було організувати і самостійно

керувати кластерами машин. Спроба Yahoo! зменшити складність програмування цих завдань реалізувалася у вигляді Apache Pig, ETL-інструменту, здатного перекладати власну мову Pig Latin у кроки MapReduce. Однак незабаром до розвитку цієї нової екосистеми долучилися й інші.

У 2007 році компанія Facebook випустила у відкритий доступ два нові проєкти під ліцензією Apache: Apache Hive і через рік Apache Cassandra. Apache Hive - це фреймворк, здатний перетворювати SQL-запити в завдання MapReduce для Hadoop. При цьому Cassandra є великим стовпчиковим сховищем, призначеним для широкомасштабного розподіленого доступу до контенту і його оновлення. Це сховище не вимагало для своєї роботи Hadoop, але незабаром, коли були створені конектори для MapReduce, стало частиною цієї екосистеми.

Водночас менш відома компанія Powerset, яка працювала над пошуковим движком, надихнулася роботою Google з Bigtable і розробила Apache Hbase, ще одне стовпчикове сховище, що спирається на HDFS. Незабаром після цього Powerset була поглинена корпорацією Microsoft, яка запустила на її основі новий проект під назвою Bing.

На швидке впровадження Hadoop рішуче вплинула ще одна компанія - Amazon. Запустивши Amazon Web Services, першу хмарну платформу з доступом до ресурсів на вимогу, і швидко додавши підтримку MapReduce через сервіс Elastic MapReduce, ця компанія дала змогу стартапам зі зручністю зберігати свої дані в S3, розподіленій файловій системі, а також розгортати та виконувати в ній завдання MapReduce.

Перший вендор (Вендор - це постачальник, який продає і просуває товари і послуги під власним брендом або торговельною маркою) послуг Hadoop під назвою Cloudera з'явився у 2008 році і незабаром був викуплений Дугом Каттінгом. Cloudera пропонувала готовий дистрибутив Hadoop, який називався CDH, а також інтерфейс для моніторингу кластерів Cloudera Manager, який, зрештою, спростив установку та управління кластерами і супутнім ПЗ на кшталт Hive і HBase. З тією ж метою незабаром були засновані компанії Hortonworks і MapR. Сховище Cassandra також отримало свого вендора Datastax, який з'явився 2010 року.

Через деякий час усі учасники ринку дійшли згоди, що, незважаючи на зручність Hive як SQL-інструменту для управління величезними ETL-пакетами, він погано підходить для інтерактивної та бізнес-аналітики (BI). Усі, хто звик до стандартних БД на мові SQL, очікує від них можливості сканувати таблиці з тисячами рядків протягом лічених мілісекунд. Hive же вимагав для цієї операції хвилини.

Google побічно зробила вирішальний вплив на світ великих даних, випустивши 2010 року четверту дослідницьку роботу під заголовком "Dremel:

Interactive Analysis of Web-Scale Datasets". У ній описувалися дві основні інновації:

Архітектура розподілених інтерактивних запитів, що надихнула появу більшості інтерактивних SQL-інструментів, про які буде сказано нижче.

Форма стовпчикового сховища, яка лягла в основу кількох нових форматів зберігання даних, таких як Apache Parquet, спільно розробленого Cloudera і Twitter, і Apache ORC, випущеного Hortonworks спільно з Facebook.

Cloudera в прагненні вирішити проблему високої затримки в Hive і відірватися від конкурентів у 2012 році вирішила запустити новий відкритий SQL-движок для інтерактивних запитів під назвою Apache Impala. Паралельно з цим MapR запустила власний опенсорсний інтерактивний движок, назвавши його Apache Drill. Керівництво Hortonworks, замість створення нового рушія з нуля, вважало за краще зайнятися прискоренням роботи Hive, запустивши проєкт Apache Tez, якусь подобу версії 2 для MapReduce, і адаптувавши Hive під виконання Tez замість MapReduce.

2010-2014: Hadoop 2.0 і революція Spark

У той час як Hadoop зміцнював свої позиції і був зайнятий впровадженням нового ключового компонента YARN для управління ресурсами. Spark виявиться чудовою заміною MapReduce з огляду на ширші можливості, простіший синтаксис і високу швидкодію, яку, зокрема, зумовлювала його здатність кешувати дані в ОЗП. Єдиним слабким місцем порівняно з MapReduce на перших порах була нестабільність Spark, але цю проблему в міру розвитку продукту було вирішено.

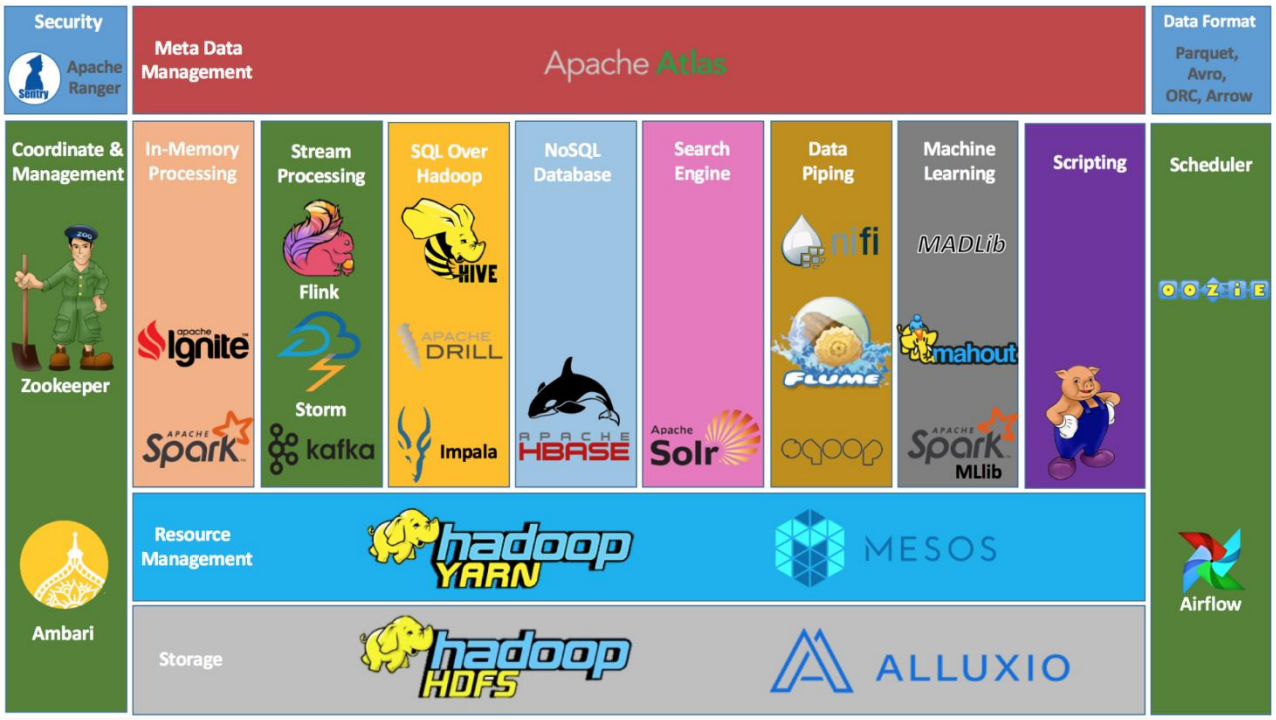
Цей інструмент також мав високу операційну сумісність із Hive, оскільки SparkSQL ґрунтувався на синтаксисі Hive (насправді від самого початку розробники Spark запозичили в Hive лексер і парсер), що істотно спрощувало перехід із Hive на SparkSQL. На додачу до цього - Spark привернув велику увагу у світі машинного навчання, оскільки колишні спроби писати алгоритми МО через MapReduce, наприклад, Apache Mahout, явно програвали реалізаціям Spark.

Для підтримки і монетизації швидкого зростання Spark його творці у 2013 році запустили Databricks. Метою цього проєкту стало надання кожному можливості обробки величезних обсягів даних. Для цього платформа реалізувала прості та ефективні API багатьма мовами (Java, Scala, Python, R, SQL і навіть .NET), а також нативні конектори для багатьох джерел і форматів даних (csv, json, parquet, jdbc, avro, etc.). Ринкова стратегія Databricks відрізнялася від стратегій попередників. Замість пропозиції локального розгортання Spark, компанія зайняла позицію виключно хмарної платформи, спочатку інтегрувавшись із сервісом AWS (який на той момент був найпопулярнішою хмарою), а потім з Azure і GCP. Дев'ять років потому, можна впевнено сказати, що це був грамотний хід.

Тим часом для обробки потокових подій з'являлися нові відкриті проєкти на кшталт Apache Kafka, розподіленої черги повідомлень, розробленої LinkedIn, та Apache Storm 3, рушія розподілених потокових обчислень від Twitter. Обидва інструменти вийшли в 2011 році. У той самий період платформа Amazon Web Services досягла небувалої популярності та успіху: це можна продемонструвати

одним лише різким стрибком розвитку Netflix у 2010 році, що став можливим переважно завдяки хмарі Amazon. У хмарній сфері почала зароджуватися конкуренція. Спочатку у 2010 році з'явився сервіс Microsoft Azure, слідом за яким у 2011 народилася Google Cloud Platform (GCP).

Екосистема Hadoop



2016-2020 на зміну Hadoop приходить контейнеризація та глибинне навчання

Першим трендом стало масове переміщення інфраструктур даних у хмару, під час якого HDFS було замінено хмарними сховищами на кшталт Amazon S3, Google Storage і Azure Blob Storage.

Другим трендом була контейнеризація. Docker - це фреймворк контейнеризації, який вийшов у 2011 році і з 2013 почав різко набирати популярність. У червні 2014 року Google випустили відкритий інструмент для оркестрації контейнерів під назвою Kubernetes (він же K8s), який одразу ж взяли на озброєння багато компаній для побудови нових розподілених/масштабованих архітектур. Docker і Kubernetes дали змогу розгортати нові види таких архітектур, уже стабільніші, надійніші та придатніші для безлічі випадків, включно з подієво-орієнтованими перетвореннями в реальному часі. Hadoop знадобився час, щоб встигнути за Docker, і підтримка запуску в ньому контейнерів з'явилася лише у версії 3.0 у 2018 році.

Третім трендом, як уже говорилося, стала поява повністю керованих розпаралелених сховищ для аналітики на базі SQL.

Цей момент добре демонструється формуванням "Сучасного дата-стека" і виходом у 2016 році інструменту командного рядка dbt.

Четвертим трендом, що вплинув на Hadoop, став розвиток глибокого навчання. У другій половині 2010-х усі вже чули про глибоке навчання і штучний інтелект.

По-перше, потрібні були GPU, яких у вузлах кластерів Hadoop зазвичай не було. По-друге, вченим були потрібні останні версії бібліотек для глибокого навчання, таких як TensorFlow і Keras, встановити які на весь кластер було проблематично, особливо коли багато користувачів просили різні версії однієї бібліотеки. Цю конкретну проблему чудово вирішував Docker, але інтеграція цього інструменту в Hadoop вимагала часу, а чекати аналітики даних були не

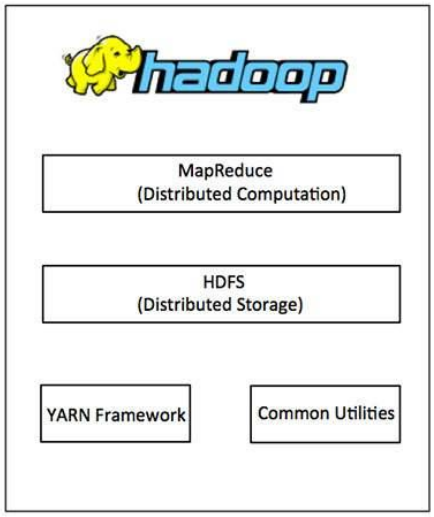
готові. У зв'язку з цим вони зазвичай вважали за краще замість використання кластера запускати одну потужну віртуальну машину з 8 GPU.

Cloudera у 2017 році зробила своє перше публічне розміщення акцій (IPO), компанія вже займалася розробкою і поширенням новітнього продукту, Data Science Workbench. Це рішення ґрунтувалося вже не на Hadoop або YARN, а на контейнеризації за допомогою Docker і Kubernetes, що давало змогу вченим за даними розгортати моделі у власному середовищі у вигляді контейнеризованого застосунку без ризику для безпеки та стабільності.

Хоча згасання Hadoop не означає повну кончину всієї екосистеми, оскільки багато великих компаній, як і раніше, її використовують, особливо для локальних розгортань. Використовують її і багато рішень, побудованих навколо цієї технології або її частин. При цьому також впроваджуються інновації, наприклад, Apache Hudi, випущений компанією Uber 2016 року, Apache Iceberg, запущений Netflix 2017 року, і відкритий продукт Delta Lake, який розробники Databricks представили 2019 року.

Hadoop - це фреймворк з відкритим вихідним кодом від Apache, який використовується для зберігання, обробки та аналізу великих обсягів даних. Hadoop написаний на Java і не є OLAP (онлайн-аналітичною обробкою). Він використовується для пакетної/офлайн обробки. Його використовують Facebook, Yahoo, Google, Twitter, LinkedIn та багато інших. Крім того, його можна масштабувати, просто додаючи вузли в кластер.

Модулі Hadoop



HDFS: розподілена файлова система Hadoop. Google опублікував документ GFS, на основі якого була розроблена HDFS. В ній йдеться про те, що файли будуть розбиватися на блоки і зберігатися у вузлах розподіленої архітектури.

Yarn: Ще один переговорник ресурсів використовується для планування завдань і управління кластером.

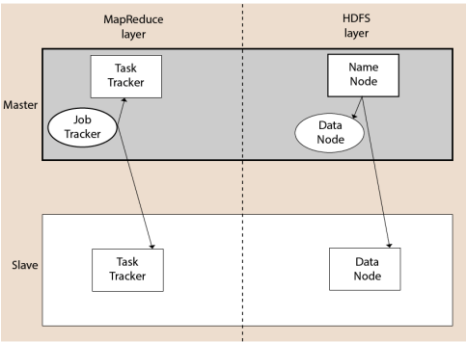
Map Reduce: Це фреймворк, який допомагає Java-програмам виконувати паралельні обчислення над даними з використанням пари ключ-значення. Задача Map приймає вхідні дані і перетворює їх у набір даних, які можуть бути обчислені в парі ключових значень. Вихідні дані Map-задачі споживаються задачею reduce, а потім на виході з редуктора отримується бажаний результат.

Hadoop Common: Ці бібліотеки Java використовуються для запуску Hadoop і використовуються іншими модулями Hadoop.

Архітектура Hadoop - це пакет файлової системи, механізму MapReduce та HDFS (Hadoop Distributed File System - розподілена файлова система Hadoop). Движок MapReduce може бути MapReduce/MR1 або YARN/MR2.

Кластер Hadoop складається з одного головного і декількох підлеглих вузлів. Головний вузол включає Job Tracker, Task Tracker, NameNode і DataNode, тоді як підлеглий вузол включає DataNode і TaskTracker.

Розподілена файлова система Hadoop



Hadoop Distributed File System (HDFS) - це розподілена файлова система для Hadoop. Вона містить архітектуру майстер/підлеглий. Ця архітектура складається з одного вузла NameNode, який виконує роль ведучого, і декількох вузлів DataNode, які виконують роль підлеглих.

NameNode, і DataNode можуть працювати на звичайних машинах. Для розробки HDFS використовується мова Java. Тому будь-яка машина, що підтримує мову Java, може легко запустити програмне забезпечення NameNode і DataNode.

NameNode Це єдиний головний сервер, що існує в кластері HDFS.

Оскільки це єдиний вузол, він може стати причиною одномоментної відмови. Він керує простором імен файлової системи, виконуючи такі операції, як відкриття, перейменування та закриття файлів. Це спрощує архітектуру системи.

DataNode Кластер HDFS складається з декількох вузлів даних (DataNode).

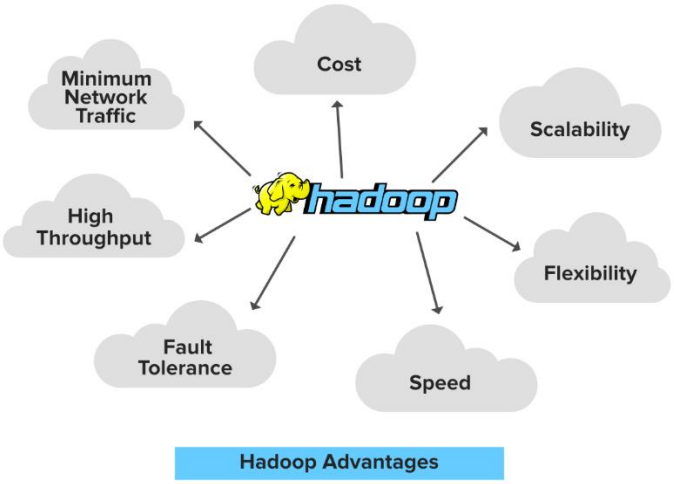
Кожен DataNode містить кілька блоків даних. Ці блоки даних використовуються для зберігання даних. DataNode відповідає за читання і запис запитів від клієнтів файлової системи. Він виконує створення, видалення та реплікацію блоків за інструкціями від NameNode.

Job Tracker Роль Job Tracker полягає в тому, щоб приймати завдання MapReduce від клієнта і обробляти дані, використовуючи NameNode. У відповідь NameNode надає метадані Job Tracker'у.

Task Tracker Працює як підлеглий вузол для Job Tracker. Він отримує завдання і код від Job Tracker і застосовує цей код до файлу. Цей процес також можна назвати Mapper.

Рівень MapReduce MapReduce починає існувати, коли клієнтська програма надсилає завдання MapReduce до Job Tracker. У відповідь Job Tracker надсилає запит відповідним Task Tracker'ам. Іноді TaskTracker виходить з ладу або завершує роботу. В такому випадку ця частина завдання переноситься на інший час.

Переваги Hadoop



1. Вартість

Hadoop має відкритий вихідний код і використовує недороге апаратне забезпечення, що забезпечує економічно ефективну модель, на відміну від традиційних реляційних баз даних, які вимагають дорогого апаратного забезпечення і високопродуктивних процесорів для роботи з великими обсягами даних. Проблема з традиційними реляційними базами даних полягає в тому, що зберігання величезних обсягів даних не є економічно ефективним, тому компанії починають видаляти необроблені дані, що може призвести до неправильного сценарію розвитку їхнього бізнесу. Засоби Hadoop надають нам 2 основні переваги з точки зору вартості, одна з яких полягає в тому, що це відкритий вихідний код, який є безкоштовним у використанні, а інша полягає в тому, що він використовує товарне обладнання, яке також є недорогим.

2. Масштабованість

Hadoop - це дуже масштабована модель. Велика кількість даних розподіляється на декілька недорогих машин у кластері, які обробляються паралельно. кількість цих машин або вузлів може бути збільшена або зменшена відповідно до вимог підприємства. У традиційних СКБД (реляційних системах управління базами даних) системи не можуть бути масштабовані для роботи з великими обсягами даних.

3. Гнучкість

Hadoop розроблений таким чином, що він може дуже ефективно працювати з будь-якими типами даних, такими як структуровані (MySql Data), напівструктуровані (XML, JSON), неструктуровані (зображення та відео). Це означає, що він може легко обробляти будь-який тип даних незалежно від їх структури, що робить його дуже гнучким. Це дуже корисно для підприємств, оскільки вони можуть легко обробляти великі набори даних, тому підприємства можуть використовувати Hadoop для аналізу цінної інформації з таких джерел, як соціальні мережі, електронна пошта і т.д. Завдяки цій гнучкості Hadoop можна використовувати для обробки журналів, зберігання даних, виявлення шахрайства тощо.

4. Швидкість

Hadoop використовує розподілену файлову систему для управління своїм сховищем, тобто HDFS (Hadoop Distributed File System). У DFS (Distributed File System) файл великого розміру розбивається на блоки файлів малого розміру, а потім розподіляється між вузлами, доступними в кластері Hadoop, оскільки ця величезна кількість блоків файлів обробляється паралельно, що робить Hadoop швидшим, завдяки чому він забезпечує високий рівень продуктивності в порівнянні з традиційними системами управління базами даних. Коли ви маєте справу з великою кількістю неструктурованих даних, швидкість є важливим фактором, з Hadoop ви можете легко отримати доступ до трильйонів даних всього за кілька хвилин.

5. Відмовостійкість

Hadoop використовує звичайне обладнання (недорогі системи), яке може вийти з ладу в будь-який момент. В Hadoop дані реплікуються на різні вузли даних в кластері Hadoop, що забезпечує доступність даних, якщо будь-яка з ваших систем вийде з ладу. Ви можете прочитати всі дані з однієї машини, якщо ця машина зіткнулася з технічною проблемою, дані також можуть бути прочитані з інших вузлів у кластері Hadoop, оскільки дані копіюються або

реплікуються за замовчуванням. Hadoop робить 3 копії кожного файлового блоку і зберігає їх на різних вузлах.

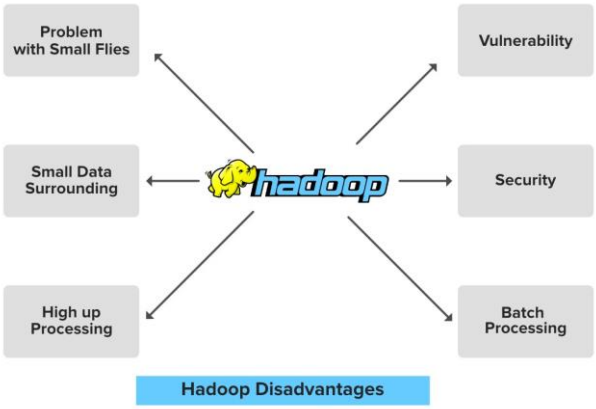
6. Висока пропускна здатність

Hadoop працює на розподіленій файловій системі, де різні завдання призначаються різним вузлам даних у кластері, блок цих даних обробляється паралельно в кластері Hadoop, що забезпечує високу пропускну здатність. Пропускна здатність - це не що інше, як завдання або робота, виконана за одиницю часу.

7. Мінімальний мережевий трафік

В Hadoop кожне завдання розбивається на різні невеликі підзадачі, які потім призначаються кожному вузлу даних, доступному в кластері Hadoop. Кожен вузол даних обробляє невелику кількість даних, що призводить до низького трафіку в кластері Hadoop

Недоліки Hadoop



1. Проблема з малими файлами

Hadoop може ефективно працювати над невеликою кількістю файлів великого розміру. Hadoop зберігає файл у вигляді файлових блоків розміром від 128 МБ (за замовчуванням) до 256 МБ. Hadoop зазнає невдачі, коли йому потрібно отримати доступ до файлу малого розміру у великій кількості. Така кількість маленьких файлів перевантажує Namenode і ускладнює роботу.

2. Вразливість

Hadoop - це фреймворк, написаний на java, а java є однією з найпоширеніших мов програмування, що робить його більш вразливим, оскільки він може бути легко використаний будь-яким кіберзлочинцем.

3. Низька продуктивність в умовах невеликих обсягів даних

Hadoop в основному призначений для роботи з великими масивами даних, тому він може бути ефективно використаний для організацій, які генерують великий обсяг даних. Його ефективність знижується при роботі з малими обсягами даних.

4. Відсутність безпеки

Дані - це все для організації, за замовчуванням функція безпеки в Hadoop недоступна. Тому драйвер даних повинен бути обережним з цим аспектом безпеки і повинен вжити відповідних заходів щодо нього. Hadoop використовує Kerberos для функції безпеки, якою непросто керувати. Зберігання та мережеве шифрування відсутні в Kerberos, що робить нас більш стурбованими з цього приводу.

5. Високошвидкісна обробка

Операції читання/запису в Hadoop є непомірними, оскільки ми маємо справу з даними великого розміру, які знаходяться в ТБ або ПБ. В Hadoop дані читаються або записуються з диска, що ускладнює виконання обчислень в пам'яті і призводить до накладних витрат на обробку або високопродуктивної обробки.

6. Підтримує тільки пакетну обробку

Пакетна обробка - це не що інше, як процеси, які виконуються у фоновому режимі і не мають жодної взаємодії з користувачем. Двигуни, що використовуються для цих процесів в ядрі Hadoop, не дуже ефективні. З ними неможливо отримати вихідні дані з низькою затримкою.

Лекція 3

Огляд обчислювальних технології великих даних. Огляд технологій MapReduce

MapReduce - це модель розподілених обчислень від компанії *Google*, яка використовується в технологіях Big Data для паралельних обчислень над дуже великими (до кількох петабайт) наборами даних у комп'ютерних кластерах, і фреймворк для обчислення розподілених завдань на вузлах (*node*) кластера.

Це фреймворк для кластерної обробки даних. Він складається з функцій *Map* і *Reduce*, розподіляє завдання обробки даних між різними комп'ютерами, а потім зводить результати в єдиний підсумок.

Завдання MapReduce розбиває вхідні дані на незалежні фрагменти. Ці незалежні фрагменти обробляються завданнями картографування паралельно. Потім фреймворк сортує виходи завдань відображення. Ці результати задач картографування потім надаються як вхідні дані для задач редукції. Вхідні та вихідні дані завдання MapReduce зберігаються у файловій системі. Фреймворк Hadoop піклується про планування, моніторинг та повторне виконання завдань, які не вдалося виконати.

Розподілена файлова система Hadoop і фреймворк MapReduce працюють на одному і тому ж наборі вузлів, тобто вузли зберігання і обчислювальні вузли однакові.

Така конфігурація дозволяє фреймворку Hadoop ефективно планувати завдання на вузлах, де присутні дані. Це призводить до дуже високої сукупної пропускної здатності кластера Hadoop.

Технологія практично універсальна: її можна використовувати для індексації веб-контенту, підрахунку слів у великому файлі, лічильників частоти

звернень до заданої адреси, обчислення об'єму всіх веб-сторінок з кожної URL-адреси конкретного хост-вузла, створення списку всіх адрес з необхідними даними та інших завдань обробки величезних масивів розподіленої інформації. Також до сфер застосування MapReduce належить розподілений пошук і сортування даних, звернення до графа веб-посилань, обробка статистики логів мережі, побудова інвертованих індексів, кластеризація документів, машинне навчання і статистичний машинний переклад. Також MapReduce адаптована під багатопроцесорні системи, добровільні обчислювальні, динамічні хмарні та мобільні середовища.

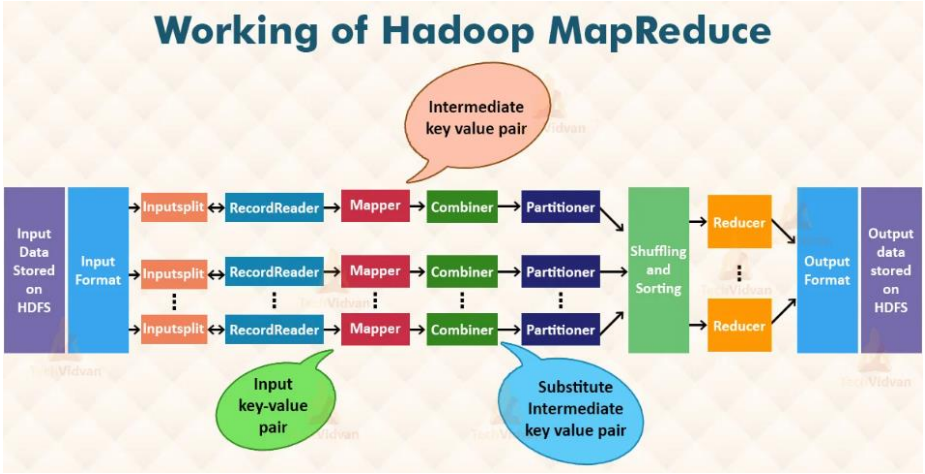
Спочатку назву MapReduce було запатентовано корпорацією Google, але в міру розвитку технологій Big Data стало загальним поняттям світу великих даних. Сьогодні безліч різних як комерційних, так і вільних продуктів, що використовують цю модель розподілених обчислень: Apache Hadoop, Apache CouchDB, MongoDB, MySpace Qizmt та інші Big Data фреймворки і бібліотеки, написані різними мовами програмування. Серед інших найбільш відомих реалізацій MapReduce варто відзначити такі:

- Greenplum - комерційна реалізація з підтримкою мов Python, Perl, SQL тощо;
- GridGain - безкоштовна реалізація з відкритим вихідним кодом мовою Java;
- Phoenix - реалізація мовою C з використанням поділюваної пам'яті;
- MapReduce реалізована в графічних процесорах NVIDIA з використанням CUDA;
- Qt Concurrent - спрощена версія фреймворка, реалізована на C++, для розподілу завдання між кількома ядрами одного комп'ютера;
- CouchDB використовує MapReduce для визначення уявлень поверх розподілених документів;
- Skynet - реалізація з відкритим вихідним кодом мовою Ruby;
- Disco - реалізація від компанії Nokia, ядро якої написане мовою Erlang, а додатки можна розробляти на Python;

- Hive framework - надбудова з відкритим вихідним кодом від Facebook, що дає змогу комбінувати підхід MapReduce і доступ до даних SQL-подібною мовою;
- Qizmt - реалізація з відкритим вихідним кодом від MySpace, написана на C#;
- DryadLINQ - реалізація від Microsoft Research на основі PLINQ і Dryad.
- Hadoop здатний запускати програми MapReduce, написані різними мовами: Java, Ruby, Python і C ++. Програми MapReduce паралельні за своєю природою, тому вони дуже корисні для виконання великомасштабного аналізу даних з використанням декількох машин у кластері.

Як працює Hadoop MapReduce?

Весь процес проходить через різні фази виконання MapReduce, а саме: розбиття, відображення, сортування та перетасування, а також редукцію.



Вхідні файли (InputFiles)

Дані, які має обробити задача MapReduce, зберігаються у вхідних файлах. Ці вхідні файли зберігаються у розподіленій файловій системі Hadoop. Формат файлів є довільним, хоча також можна використовувати рядкові лог-файли та двійковий формат.

InputFormat

Визначає специфікацію вхідних даних для завдання. InputFormat перевіряє специфікацію вхідних даних завдання MapReduce і розбиває вхідні файли на логічні екземпляри InputSplit. Кожен екземпляр InputSplit потім призначається окремому мапперу. TextInputFormat - це InputFormat за замовчуванням.

InputSplit

Представляє дані для обробки окремим маппером. InputSplit зазвичай представляє байт-орієнтоване представлення вхідних даних. Відповідальність за обробку та представлення орієнтованого на записи вигляду лежить на RecordReader. За замовчуванням InputSplit є FileSplit.

RecordReader

RecordReader зчитує пари <ключ, значення> з InputSplit. Він перетворює байт-орієнтоване представлення вхідних даних і представляє запис-орієнтоване представлення реалізаціям Mapper для обробки.

Він відповідає за обробку меж записів і представлення завдань Map з ключами і значеннями. Зчитувач записів розбиває дані на пари <ключ, значення> для введення у Mapper.

Маппер (Mapper)

Mapper відображає вхідні пари <ключ, значення> у набір проміжних пар <ключ, значення>. Він обробляє вхідні записи від RecordReader і генерує нові пари <ключ, значення>. Пари <ключ, значення>, згенеровані Mapper'ом, відрізняються від вхідних пар <ключ, значення>.

Згенеровані пари <ключ, значення> є виходом Mapper, відомим як проміжний вихід. Ці проміжні виходи мапперів записуються на локальний диск.

Вихідні дані мапперів не зберігаються у розподіленій файловій системі Hadoop, тому що це тимчасові дані, і запис цих даних на HDFS призведе до створення непотрібних копій. Вихідні дані мапперів передаються до комбінатора для подальшої обробки.

Комбайнер (Combiner)

Він також відомий як "міні-редуктор". Об'єднувач виконує локальну агрегацію на виході мапперів. Це допомагає мінімізувати передачу даних між маппером і редуктором.

Після виконання функції Об'єднувача вихідні дані передаються до Розділювача для подальшої обробки.

Розділювач (Partitioner)

Коли ми працюємо над програмою MapReduce з більш ніж одним редуктором, то в поле зору потрапляє тільки Розділювач (Partitioner). Для одного редуктора ми не використовуємо Partitioner. Він розділяє простір ключів. Він контролює розбиття ключів проміжних виходів Mapper.

Partitioner отримує вихід від Комбінатора і виконує розбиття. Ключ використовується для отримання розбиття, як правило, за допомогою хеш-функції. Кількість розбиттів аналогічна кількості завдань редукції. За замовчуванням використовується HashPartitioner.

Перетасування та сортування (Shuffling and Sorting)

На вхід редуктора завжди надходять відсортовані проміжні результати роботи мапперів. Після об'єднання і розділення фреймворк через HTTP отримує всі відповідні розділи виводу всіх мапперів.

Після того, як вихідні дані всіх мапперів перемішані, фреймворк групує входи редуктора на основі ключів. Це потім надається як вхідні дані для редуктора.

Редуктор (Reducer)

Редуктор зменшує набір проміжних значень, які мають спільний ключ, до меншого набору значень. На виході редуктора отримуємо кінцевий результат. Цей результат зберігається у розподіленій файловій системі Hadoop.

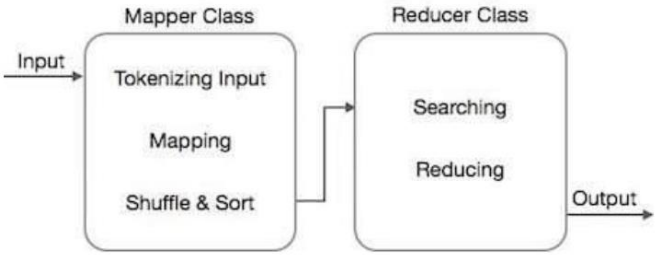
Записувач (RecordWriter)

RecordWriter записує вихідні дані (пари ключ-значення) редуктора у вихідний файл. Він записує результати роботи MapReduce до FileSystem.

OutputFormat

OutputFormat визначає спосіб, у який вихідні пари ключ-значення будуть записані у вихідні файли. Він перевіряє специфікацію виводу для завдання MapReduce. OutputFormat в основному забезпечує реалізацію RecordWriter, яка використовується для запису вихідних файлів завдання MapReduce. Вихідні файли зберігаються у файловій системі.

MapReduce – Алгоритм



Алгоритм MapReduce містить два важливих завдання, а саме ***Map*** і ***Reduce***.

Завдання карти виконується за допомогою **Mapper Class**

Завдання зменшення виконується за допомогою **Reducer Class**.

Клас Mapper приймає вхідні дані, маркує їх, відображає і сортує їх. Вихідні дані класу Mapper використовуються як вхідні дані класом Reducer, який, своєю чергою, шукає пари, що збігаються, і скорочує їх.

MapReduce реалізує різні математичні алгоритми, щоб розділити завдання на маленькі частини та призначити їх кільком системам. З технічного погляду алгоритм MapReduce допомагає відправляти завдання Map & Reduce на відповідні сервери в кластері.

Ці математичні алгоритми можуть включати в себе наступне:

- сортування
- пошук
- індексування
- TF-IDF

СОРТУВАННЯ

Сортування є одним з основних алгоритмів MapReduce для обробки та аналізу даних. MapReduce реалізує алгоритм сортування для автоматичного сортування вихідних пар ключ-значення з перетворювача за їхніми ключами.

Методи сортування реалізовані в самому класі *mapper*.

На етапі перемішування і сортування після токенізації значень у класі зіставлення клас *Context* (визначений користувачем клас) збирає значення ключів, що збігаються, у вигляді колекції.

Щоб зібрати схожі пари ключ-значення (проміжні ключі), клас Mapper використовує клас RawComparator для сортування пар ключ-значення.

Набір проміжних пар ключ-значення для даного редуктора автоматично сортується Hadoop для формування значень ключа (K2, {V2, V2, ... }) до їхнього подання редуктору.

ПОШУК

Пошук відіграє важливу роль в алгоритмі MapReduce. Це допомагає у фазі об'єднувача (опція) і у фазі редуктора

Фаза Map обробляє кожен вхідний файл і надає дані про співробітника в парах ключ-значення (<k, v>: <emp name, salary>)

Фаза об'єднувача (метод пошуку) прийматиме вхідні дані з фази карти у вигляді пари ключ-значення з ім'ям співробітника і зарплатою. Використовуючи техніку пошуку, комбінатор перевірить усю зарплату співробітника, щоб знайти співробітника з найбільшим окладом у кожному файлі.

Фаза скорочення - Сформуйте кожен файл, ви знайдете найбільш високооплачуваного співробітника. Щоб уникнути надмірності, перевірте всі пари <k, v> і видаліть записи, що дублюються, якщо такі є. Той самий алгоритм використовується між чотирма парами <k, v>, які надходять із чотирьох вхідних файлів

ІНДЕКСУВАННЯ

Зазвичай індексація використовується для вказівки на конкретні дані та їхню адресу. Він виконує пакетну індексацію вхідних файлів для певного Mapper.

Техніка індексування, яка зазвичай використовується в MapReduce, називається інвертованим індексом. Пошукові системи, такі як Google і Bing, використовують метод перевернутої індексації

TF-IDF

TF-IDF - це алгоритм обробки тексту, скорочений від Term Frequency - Inverse Document Frequency. Це один із поширених алгоритмів веб-аналізу. Тут термін "частота" відноситься до числа разів, коли термін з'являється в документі.

Термін частота (TF)

Він вимірює, як часто конкретний термін зустрічається в документі. Він розраховується за кількістю появ слова в документі, поділеною на загальну кількість слів у цьому документі.

Частота зворотних документів (IDF)

Він вимірює важливість терміна. Його розраховують за кількістю документів у текстовій базі даних, поділеною на кількість документів, у яких з'являється конкретний термін.

Під час обчислення TF усі терміни вважаються однаково важливими. Це означає, що TF підраховує частоту терміна для звичайних слів, таких як "є", "а", "що" тощо. Д. Таким чином, нам потрібно знати часті терміни при збільшенні кількості рідкісних, обчислюючи наступне:

```
TF(the) = (Number of times term the ‘the’ appears in a document) / (Total number of terms in the document)
```

```
IDF(the) = log_e(Total number of documents / Number of documents with term ‘the’ in it).
```

MapReduce - Установка

MapReduce працює лише в операційних системах зі смаком Linux і поставляється з Hadoop Framework. Нам потрібно виконати наведені нижче кроки для встановлення фреймворку Hadoop.

MapReduce - API

У цьому розділі ми детально розглянемо класи та їхні методи, які беруть участь в операціях програмування MapReduce. Насамперед ми зосередимося на наступному:

- Інтерфейс JobContext
- Job Class
- Mapper Class
- Reducer Class

Інтерфейс JobContext

Інтерфейс JobContext - це суперінтерфейс для всіх класів, який визначає різні завдання в MapReduce. Це дає вам доступ тільки для читання до завдання, яке надається завданням під час їхнього виконання.

MapContext <KEYIN, VALUEIN, KEYOUT, VALUEOUT>

Визначає контекст, який надається Mapper.

ReduceContext <KEYIN, VALUEIN, KEYOUT, VALUEOUT>

Визначає контекст, який передається редуктору.

Клас Job - це основний клас, який реалізує інтерфейс JobContext.

Клас Job є найважливішим класом в API MapReduce. Це дає змогу користувачеві налаштовувати завдання, надсилати його, контролювати його виконання і запитувати стан. Методи *set* працюють тільки до відправлення завдання, після чого вони генерують виняток *IllegalStateException*.

Зазвичай користувач створює додаток, описує різні аспекти завдання, а потім надсилає завдання і відстежує його виконання.

Нижче наводиться короткий опис конструктора класу Job.

1. Робота ()
2. Job (Конфіг конфігурації)
3. Job (Конфігурація conf, String jobName)

Ось деякі з важливих методів класу Job:

1. getJobName () – Задане користувачем ім'я завдання.
2. getJobState () – Повертає поточний стан завдання.
3. завершено() – Перевіряє, закінчена робота чи ні.
4. setInputFormatClass () – Встановлює InputFormat для роботи.
5. setJobName (ім'я рядка) – Встановлює задане користувачем ім'я завдання.
6. setOutputFormatClass () – Встановлює формат виведення для роботи.
7. setMapperClass (клас) – Встановлює Mapper для роботи.
8. setReducerClass (клас) – Встановлює Редуктор для роботи.
9. setPartitionerClass (клас) – Встановлює Partitioner для роботи.
10. setCombinerClass (клас) – Встановлює Combiner для роботи.

Mapper Class

Клас Mapper визначає завдання *Map*. Зіставляє вхідні пари ключ-значення з набором проміжних пар ключ-значення. Карти - це окремі завдання, які перетворюють вхідні записи в проміжні записи. Перетворені проміжні записи не обов'язково мають бути того самого типу, що й вхідні записи. Задана вхідна пара може відображатися на нуль або на безліч вихідних пар. Метод карта є найвідомішим методом класу Mapper.

```
    map(KEYIN key, VALUEIN value,
org.apache.hadoop.mapreduce.Mapper.Context context)
```

Цей метод викликається один раз для кожної пари ключ-значення у вхідному розбитті.

Reducer Class

Клас Reducer визначає завдання Reduce у MapReduce. Це зменшує набір проміжних значень, які розділяють ключ, до меншого набору значень. Реалізації редуктора можуть отримати доступ до Конфігурації для завдання через метод JobContext.getConfiguration (). Редуктор має три основні етапи - перемішування, сортування та зменшення.

Перемішати - Редуктор копіює відсортований висновок з кожного Mapper, використовуючи HTTP по всій мережі.

Сортувати - платформа об'єднує сортування входів Редуктора за ключами (оскільки різні Mappers можуть виводити один і той самий ключ). Фази тасування і сортування відбуваються одночасно, тобто під час вибірки виходів вони об'єднуються.

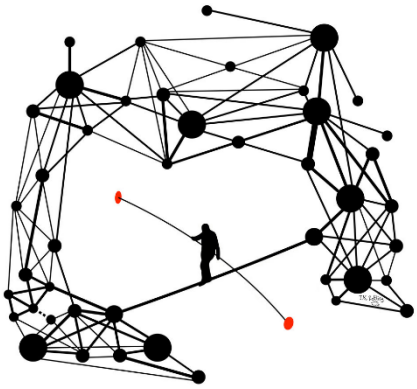
Скорочення - На цьому етапі метод Redu (Object, Iterable, Context) викликається для кожного <ключа, (набору значень)> у відсортованих вхідних даних.

```
reduce (KEYIN key, Iterable<VALUEIN> values,
org.apache.hadoop.mapreduce.Reducer.Context context)
```

Цей метод викликається один раз для кожного ключа в колекції пар ключ-значення.

Лекція 4

Існування структури у великих даних. Типи моделей представлення даних. Системи для представлення даних. Основні проблеми з відображенням результатів моделювання

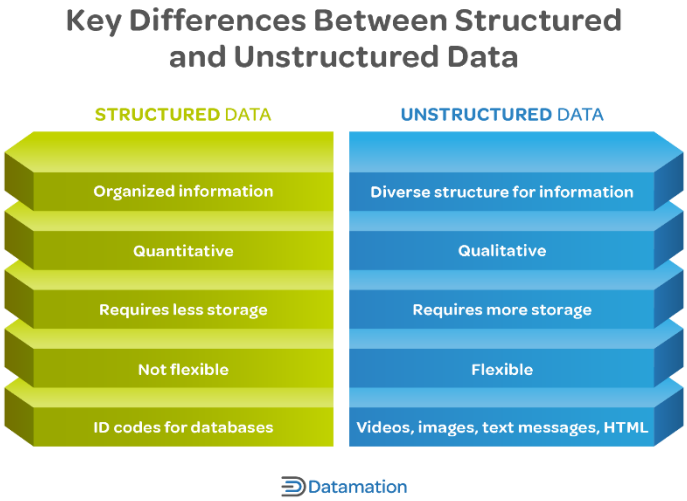


Категорії великих даних

Великі дані можуть містити три форми:

- Структуровані дані.
- Неструктуровані дані.
- Напівструктуровані дані.

Характеристикою цих форм є зберігання в системі сховищ даних.



Структуровані дані, або кількісні дані, - це інформація, яка добре організована і читається алгоритмами машинного навчання, що полегшує її пошук, маніпулювання та аналіз. Зазвичай ви знайдете структуровані дані в таблицях, рядках і стовпцях бази даних. Кожне поле містить певний тип даних, що відповідає його категорії та значенню.

Уявіть собі електронну таблицю з певними заголовками для кожного стовпчика. Такий формат дозволяє алгоритмам пошукових систем читати і розуміти дані. Структуровані дані можуть включати, наприклад, імена, адреси та дати - легко впізнавані та зрозумілі поля. Оскільки кожен запис має пошуковий

ключ, а кожне поле в цих записах чітко окреслене, ви можете шукати ці записи та поля даних за допомогою стандартного пошуку в базах даних або аналітичних програм.

Крім того, оскільки структуровані дані добре організовані, люди та автоматизовані інструменти можуть швидко сканувати, впорядковувати та аналізувати величезні обсяги даних.

Джерела структурованих даних

Структуровані дані можна генерувати різними способами і з різних джерел. Вони можуть надходити з корпоративного програмного забезпечення, такого як системи управління взаємовідносинами з клієнтами (CRM), бухгалтерські програми та інші додатки, що використовуються в критично важливих бізнес-операціях. Вона може бути отримана з онлайн-джерел, включаючи платформи соціальних мереж і веб-опитування. Вони також можуть бути отримані шляхом ручного введення даних людиною.

Крім того, структуровані дані можуть бути вилучені з неструктурованих за допомогою інструментів бізнес-аналітики (BI), які покладаються на штучний інтелект (ШІ) та обробку природної мови (NLP).

Неструктуровані дані - це інформація, яка не має внутрішньої структури або організації. Частини неструктурованих даних зазвичай називають «об'єктами», оскільки вони не мають ключів запису для їх ідентифікації. Для того, щоб організувати та ідентифікувати неструктуровані об'єктні дані, кожен окремий неструктурований об'єкт повинен бути позначений «тегом» або ідентифікатором, щоб його можна було шукати та знаходити.

Прикладами неструктурованих даних є відео, електронні листи, зображення та HTML-контент. Такі дані складають від 80 до 90 відсотків усіх даних, що генеруються у світі, але вони значно менш цінні, ніж структуровані, оскільки їх набагато складніше обробляти та витягувати з них інсайти.

Джерела неструктурованих даних

Неструктуровані дані надходять з широкого кола джерел. Об'єктом неструктурованих даних може бути текст у довільній формі, який не розбитий на

фіксований формат запису, що містить окремі поля даних. Неструктуровані дані можуть також надходити у вигляді фото, відео, креслення інженерного САПР, текстового потоку в соціальних мережах, HTML-документа або будь-якої іншої форми даних, які не зафіксовані у вигляді фіксованого запису, формату даних з визначеними полями.

Неструктуровані дані іноді можна знайти в записах структурованих даних. Наприклад, розглянемо форму, яка пропонує запитання з випадаючим списком відповідей, що також дозволяє користувачам додавати коментарі у довільній формі - відповіді, згенеровані зі списку, є структурованими даними, але поле для коментарів містить неструктуровані дані.

Напівструктуровані дані займають проміжне місце між структурованими і неструктурованими даними, оскільки вони мають певний ступінь організації, але не повністю організовані у фіксованому форматі запису, який можна знайти в традиційній системі або базі даних.

Наприклад, ви можете додати певної структури неструктурованому XML-документу, використовуючи метадані, щоб пояснити, хто і коли створив документ, і ключові слова, щоб описати вміст і зробити його доступним для пошуку. У випадку HTML-документів, які інакше були б неструктурованими, теги H1 використовуються для позначення заголовків, а H2 - для позначення підрозділів, що полегшує пошук.

Джерела напівструктурованих даних

Напівструктуровані дані бувають різних форматів і надходять з різних джерел. Датчики Інтернету речей (IoT) генерують величезні обсяги даних - вони можуть використовуватися на складах, наприклад, для оптимізації виконання замовлень, або на виробництві для моніторингу стану і функціональності обладнання. Значну частину цих даних можна зробити більш корисною, додавши до них теги, які зроблять їх доступними для пошуку.

Мови розмітки, такі як HTML і XML, також є поширеним джерелом напівструктурованих даних. Хоча пошук за тегами в напівструктурованих даних менш ефективний, ніж у структурованих, теги можуть допомогти сайтам і

технологіям, таким як штучний інтелект, ідентифікувати і знаходити документи та інші об'єкти неструктурованих даних у пошуковому запиті.

Як компанії використовують структуровані та неструктуровані дані?

Сучасні підприємства опинилися в незвичній ситуації, коли близько 80 відсотків їхньої щоденної IT-обробки здійснюється з використанням традиційних структурованих даних, тоді як 80 відсотків нових даних, що надходять, є неструктурованими. Така реальність вимагає від компаній гібридного підходу до управління даними, який включає в себе способи управління та обробки як неструктурованих, так і структурованих даних.

Найшвидший шлях до універсального управління даними полягає в тому, щоб забезпечити тегування всіх неструктурованих об'єктів даних - по суті, зробити їх напівструктурованими, щоб їх можна було індивідуально ідентифікувати та здійснювати пошук.

Тегування даних також є вимогою для організацій, які хочуть пов'язати або об'єднати як структуровані, так і неструктуровані дані у складений, гібридний запис - наприклад, пов'язати фотографію газонокосарки з фіксованим описом товару у файлі записів каталогу продажів або додати ідентифікаційний бейдж співробітника компанії до фіксованого опису цього співробітника в системі управління персоналом.

Здатність обробляти структуровані, неструктуровані та неструктуровані, але з тегами дані стає все більш важливою, оскільки компанії перекладають все більше роботи на аналітику та системи штучного інтелекту.

Сучасні компанії повинні використовувати гібридний підхід до управління даними, якщо вони хочуть успішно керувати, зберігати, курувати та використовувати величезні масиви даних, доступні для них. Цей гібридний підхід характеризується можливістю використовувати як структуровані, так і неструктуровані дані в повсякденних IT-операціях, з опціями інтелектуального аналізу даних, які здатні знаходити і пов'язувати ці різноманітні типи даних один з одним.

Ключовим елементом цього гібридного підходу до управління даними є напівструктуровані або неструктуровані дані, які були ідентифіковані та позначені, щоб зробити їх більш корисними, перетворивши на різновид структурованих даних.

Слово «***дані***» походить від латинського «datum» – факт, проте дані не завжди відповідають конкретним чи навіть реальним фактам. Іноді вони неточні або описують те, чого насправді не існує. Даними ми вважатимемо опис будь-якого явища (чи ідеї), що викликає зацікавленість через певні потреби. З даними нерозривно пов'язана їхня інтерпретація (або семантика), тобто той зміст, який їм приписується.

Дані описуються тією чи іншою мовою і фіксуються на певному носії (скажімо, папері). Зазвичай дані (факти) та їхня семантична інтерпретація фіксуються спільно. Проте в деяких випадках дані й інтерпретація розділяються. Так, у зведеній статистичній таблиці зверху, в її шапці, міститься інформація стосовно того, чим є дані (заголовок таблиці, описова інформація, назви стовпців тощо), а нижче розташовуються власне рядки чисел.

Застосування комп'ютерів для введення й обробки даних сприяло ще більшому розділенню даних та їхньої інтерпретації. Оскільки комп'ютери оперують даними як такими, велика частина інтерпретуючої інформації взагалі явно не фіксується. Програма одержує певні вхідні дані, обробляє їх і видає результат у вигляді сукупності вихідних даних.

З розвитком обчислювальної техніки з'явилася можливість фіксувати за допомогою комп'ютерів семантику даних, тобто закладати інтерпретацію в програми, що оперують даними. Проте за умов спільного використання даних різноманітними прикладними програмами цей підхід можна застосовувати лише частково. Інакше процес написання програм, які містять схожі, хоча й не ідентичні механізми інтерпретації, стає неефективним. У подібній ситуації доцільно зв'язувати дані з механізмами інтерпретації, що має забезпечувати уніфікованість подання семантичної інформації. У результаті змінюється роль даних: їх уже не можна розглядати лише як сукупність бітів, вони отримують

певне семантичне навантаження. Засоби інтерпретації мають бути гнучкими, аби разом з незмінними параметрами даних відображати й аспекти їхньої еволюції. Цього досягають двома способами. По-перше, можна відтворювати різні погляди на одні й ті самі дані. Наприклад, розглядати певну особу з точки зору кадрової системи як службовця, з погляду виробничих інтересів – як виконавця робіт, а з боку медичного обслуговування – як пацієнта. По-друге, різні дані можна подавати одноманітно. Так, адміністратори, клерки, агенти, секретарі незалежно від роду своєї діяльності можуть розглядатися в кадровій системі як службовці.

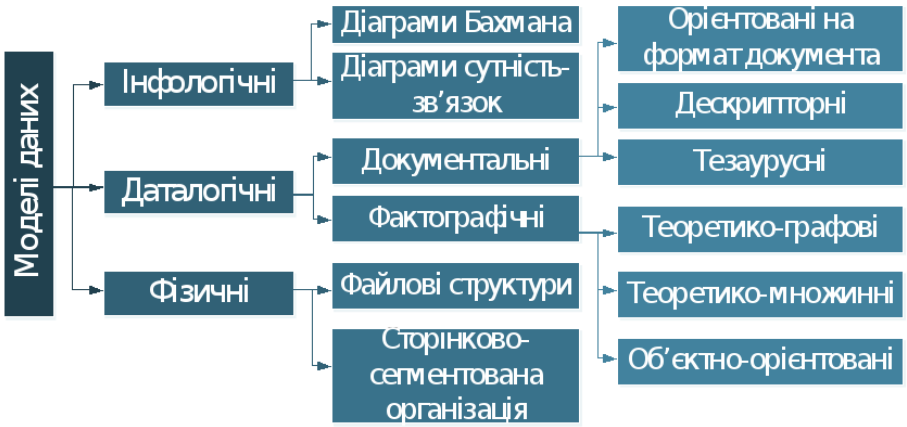
Класифікація моделей даних.

Модель даних відображує уявлення про реальний світ. Проте важливо, аби обсяг знань і семантика даних, відтворені в моделі, були адекватні способу використання даних. Вважатимемо, що **модель даних** це сукупність структури даних, операцій над ними (операції маніпулювання даними) та обмежень цілісності. Іншими словами, модель визначає, в який спосіб відбувається об'єднання даних у структури різної складності, які існують обмеження на значення даних і як здійснюється оперування цими даними

Основою для будь-якої структури даних є відображення елементарної одиниці даних у вигляді такої трійки: < ***об'єкт, властивість об'єкта, значення властивості*** >. Сукупність взаємопов'язаних між собою елементарних одиниць даних може відображуватися різноманітними способами, що приводить до формування різних структур, а відтак – різних моделей даних.

Поняття «дані» у концепції баз даних – це набір конкретних значень, параметрів, що характеризують об'єкт, умову, ситуацію або будь-які інші фактори.

Модель даних – це деяка абстракція, яка, будучи застосовна до конкретних даних, дозволяє користувачам і розроблювачам трактувати їх уже як інформацію, тобто відомості, що містять не тільки дані, але й взаємозв'язок між ними.



Інфологічні моделі даних – відбивають у природній і зручній для розроблювачів і інших користувачів формі інформаційно-логічний рівень абстрагування, пов'язаний з фіксацією й описом об'єктів предметної області, їх властивостей і їх взаємозв'язків.

Діаграма Бахмана являє собою орієнтований граф, у якому вершини відповідають групам (типам записів), а дуги – ієрархічним груповим відносинам.

Прийняті позначення:

- прямокутник – об'єкт, усередині прямокутника пишеться ім'я об'єкта й у круглих дужках можуть бути перераховані атрибути;
- об'єкти зв'язані між собою спрямованими ребрами, зв'язки підписуються;
- діаграми призначені для деревоподібних, мережних структур, тому завжди існує вихідний об'єкт і підлеглий об'єкт.

Даталогічна модель

Інфологічні моделі даних використовуються на ранніх стадіях проектування для опису структур даних у процесі розробки додатка, а *даталогічні* моделі вже підтримуються конкретними СУБД.

Документальні моделі даних – відповідають уявленню про слабоструктуровану інформацію, орієнтовану в основному на вільні формати документів, текстів природньою мовою.

Моделі, засновані на мовах розмітки документів, зв'язані насамперед зі стандартною загальною мовою розмітки – SGML (Standart Generalised Markup

Language), який був затверджений ISO у якості стандарту ще в 80-х роках. Ця мова призначена для створення інших мов розмітки, вона визначає припустимий набір тегів (посилань), їхні атрибути й внутрішню структуру документа. За допомогою SGML можна описувати структуровані дані, організовувати інформацію, що втримується в документах, представляти цю інформацію в деякому стандартизованому форматі. Але через деяку свою складність SGML використовувався в основному для опису синтаксису інших мов (найбільш відомим з яких є HTML), і деякі додатки працювали з Sgml-Документами прямо.

Набагато більш проста і зручна, ніж SGML, мова HTML дозволяє визначати оформлення елементів документа й має якийсь обмежений набір інструкцій – тегів, за допомогою яких здійснюється процес розмітки. Інструкції HTML у першу чергу призначені для керування процесом виводу вмісту документа на екрані програми-клієнта й визначають цим самим спосіб вистави документа, але не його структуру.

Однак HTML сьогодні вже не задовольняє повною мірою вимогам, пропонованим сучасними розроблювачами до мов подібного роду. І їй на зміну була запропонована нова мова гіпертекстової розмітки, потужний, гнучкий і, одночасно із цим, зручна мова XML.

XML (Extensible Markup Language) – це мова розмітки, що описує цілий клас об’єктів даних, названих XML-документами. Вона використовується як засіб для опису граматики інших мов і контролю над правильністю складання документів. Тобто сама по собі XML не містить ніяких тегів, призначених для розмітки, він просто визначає порядок їх створення.

Тезаурусні моделі – це моделі, які засновані на принципі організації словників, містять певні мовні конструкції й принципи їх взаємодії в заданій граматиці. Ці моделі ефективно використовуються в системах-перекладачах, особливо багатомовних перекладачах. Принцип зберігання інформації в цих системах і підкоряється тезаурусним моделям.

Дескрипторні моделі – найпростіші з документальних моделей, вони широко використовувалися на ранніх стадіях використання документальних баз

даних. У цих моделях кожному документу відповідає дескриптор – описатель. Цей дескриптор має тверду структуру й описує документ відповідно до тих характеристик, які потрібні для роботи з документами в розроблювальній документальній БД. Наприклад, для БД, що містить опис патентів, дескриптор містить назву області, до якої відноситься патент, номер патенту, дату видачі патенту й ще ряд ключових параметрів, які заповнюються для кожного патенту. Обробка інформації в таких базах даних велася винятково по дескрипторах, тобто по тим параметрам, які характеризували патент, а не по самому тексту патенту.

Фактографічні моделі – містять відомості, які представлені у вигляді спеціальним чином організованих сукупностей формалізованих записів даних.

Кожний з таких екземплярів структурних елементів або деяка їхня сукупність відбивають відомості про який-небудь факт, подію і т.д., відділеному (вичленованому) від усіх інших відомостей і фактів. Структура кожного типу інформаційного об’єкта складається з кінцевого набору реквізитів, що відбивають основні аспекти й характеристики відомостей для об’єктів даної предметної області. Приміром, фактографічна АІС, що накопичує відомості по особах, кожній конкретній особі в базі даних ставить у відповідність запис, що складається з певного набору таких реквізитів, як прізвище, ім’я, по батькові, рік народження, місце роботи і т.д.

Комплектування інформаційної бази у фактографічних АІС включає, як правило, обов’язковий процес структуризації вхідної інформації з документального джерела. Структуризація при цьому здійснюється через визначення (виділення, вичленовування) екземплярів інформаційних об’єктів певного типу, інформація про яких є в документі, і заповнення їх реквізитів.

Серед фактографічних інформаційних систем виділяють два класи:

- системи операційної обробки даних;
- системи, орієнтовані на аналіз даних і підтримку ухвалення рішення.

Системи операційної обробки даних

Для позначення таких систем використовується термін OLTP (On-Line Transaction Processing – оперативна обробка транзакцій або виконання транзакцій у режимі реального часу). Такі системи розраховані на обслуговування щодо простих запитів великої кількості користувачів.

Логічною одиницею функціонування таких систем є – **транзакція**, яка являє собою неподільну послідовність операцій маніпулювання даними (читання, видалення, вставка й зміна). Неподільність визначає факт фіксування змін, зроблених над даними, якщо всі операції, що входять у транзакцію виконані успішно. Якщо ж хоча б одна з операцій не виконана, то жодне зі змін е відбивається на стані даних.

Інформаційна система будується як система OLTP, якщо при її функціонуванні неминучі наступні аспекти роботи:

Одночасний доступ. Безліч користувачів звертаються до одного і того же запису, однак у конкретний момент часу тільки одному з них необхідно дозволити внесення змін. Приклад: продаж квитків.

Цілісність змін. Усі зміни в базі даних не повинні порушувати цілісності збережених даних, а для цього взаємозалежні кроки зміни повинні сприйматися системою як єдине ціле. Приклад: переклад коштів.

Системи, орієнтовані на аналіз даних

Для позначення таких систем використовується термін – OLAP (On-Line Analytical Processing – системи оперативної аналітичної обробки). Вони орієнтовані на аналіз даних і забезпечують одержання інформації, необхідної для розробки рішень у сфері керування. До систем OLAP можна віднести систему підтримки прийняття рішень – людино-машинний обчислювальний комплекс, орієнтований на аналіз даних, що й забезпечує одержання інформації, необхідної для розробки рішень у сфері керування.

До завдань, розв’язуваних такими системами, відносять: оцінку альтернатив рішень, прогнозування, класифікація, виявлення асоціацій і ін.

Однією з основних вимог до сучасної OLAP-систем є надійність зберігання даних, тобто здатність відновлювати пошкоджений стан бази даних після будь-яких апаратних або програмних збоїв.

Розв'язок цієї проблеми реалізується за допомогою *журналу транзакцій*, під яким розуміється особлива частина БД, недоступна користувачам і підтримувана з особливою старанністю (кілька копій, розташованих на різних фізичних дисках), що й містить послідовність дій про всі зміни в БД.

Основою побудови такої системи є концепція сховища даних, яке забезпечує можливість аналізу накопичених даних і представляється собою інтегровані набори даних, зібраних з різних джерел (наприклад: систем оперативного доступу до даних, електронних архівів і ін.)

Засновником даної концепції є Білл Инмон (1992 р. у книзі «Створення сховища даних» визначив *сховище даних* так – «предметно-орієнтований, інтегрований, незмінний і підтримуючий хронологію набір даних, призначений для підтримки прийняття розв'язків»).

Усім сховищам даних властиві такі властивості:

Предметна орієнтованість. Інформація в сховищі даних організована відповідно до основних аспектів діяльності підприємства (замовники, продажі, склад і т.п.); це й відрізняє сховище даних від даних системи оперативної обробка, де дані організовані відповідно до процесів (виписка рахунків, відвантаження товару й т.п.).

Інтегрованість. Вихідні дані вилучаються із систем оперативної обробки, перевіряються, очищаються, приводяться до єдиного виду, у потрібному ступені агрегуются

Прив'язка до часу. Дані в сховище завжди прямо пов'язані з певним періодом часу. Дані накопичуються в сховище у вигляді «історичних шарів», кожний з яких ставиться до конкретного періоду часу. Це дозволяє аналізувати тенденції розвитку.

Незмінюваність. Дані після завантаження в сховище даних не можуть бути змінені, тобто в цілому в сховище ніяких змін, крім додавання нових записів, не

передбачається. Важливою умовою незмінності даних є використання надійного встаткування, що забезпечує захист від збоїв.

Існують такі типи фактографічних моделей:

Теоретико-графова модель – це модель даних, у якій дозволені структури даних можуть бути представлені у вигляді граф загального або спеціального виду, наприклад дерева.

Теоретико-множинна модель – це модель даних, у якій використовується апарат реляційної алгебри, реляційного обчислення, а також операції над даними маніпулюються таблицями.

Об'єктно-орієнтована модель – це модель даних, яка базується на понятті об'єкта, тобто сутності, яка володіє станом й поведінкою.

Стан об'єкта визначається його атрибутами, а поведінка визначається сукупністю операцій, які визначені для цього об'єкта. Також передбачається можливість підтримки зв'язків між типами об'єктів.

Ієрархічна модель даних

Ієрархічна модель даних уперше була задіяна в системі IMS (Information Management System – інформаційна керуюча система) у межах проекту висадки на Місяць. У першій ієрархічній системі були повністю реалізовані функції СКБД, а саме: мови визначення та маніпулювання даними, опис і підтримка обмежень цілісності, паралелізм, відновлення, а також механізми ефективної обробки запитів. Слід сказати, що IMS і досі використовується на мейнфреймах.

Згодом було розроблено ще декілька ієрархічних СКБД, і кожна з них привносила в модель свою специфіку, зумовлену способом реалізації системи. Далі будуть розглянуті найбільш загальні та принципові аспекти моделі.

Ієрархічна структура даних визначається ієрархічною впорядкованістю своїх компонентів (або вузлів), тобто кожен вузол має не більше одного «батька» – старшого за ієрархією вузла.

Структура складається зі схем елементів даних (описова інформація) та їхніх екземплярів. Інакше кажучи, схема задає логічну структуру (або тип) елементу даних, а екземпляр – його значення.

Елементарним значенням структури є пойменоване поле даних, а його екземпляр – це елементарне значення.

Схема сегмента (яку називатимемо також просто сегментом) це пойменована впорядкована сукупність імен полів. Сегмент є одиницею доступу до даних ієрархічної структури під час взаємодії зовнішньої та оперативної пам'яті. Екземпляр сегмента – впорядкована сукупність значень полів.

Ієрархічна схема даних – це ієрархічно впорядкована сукупність сегментів, що має певні властивості:

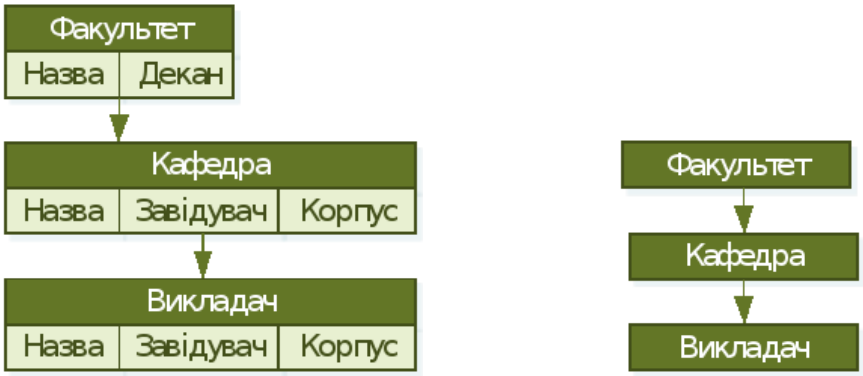
- на найвищому рівні ієрархії розташований єдиний сегмент, що називається кореневим;
- кожен інший сегмент, окрім кореневого, зв'язаний з одним і тільки одним сегментом вищого рівня, який є для нього сегмента батьківським (початковим);
- сегмент може бути зв'язаний з одним або кількома сегментами нижчого рівня, які називаються дочірніми (породженими);
- сегменти, що підпорядковані одному батьківському сегменту, називаються близнюками;
- сегменти, що не мають дочірніх, вважаються листковими, або їх ще називають листками.

Ієрархічний шлях (або просто шлях) – це послідовність сегментів, починаючи з кореневого, де кожний попередній є «батьком» наступного. Рівень сегмента визначається як кількість сегментів, що містяться на шляху, який веде від кореня до даного сегмента.

Для ієрархічної схеми використовується така графічна нотація:

Кожний сегмент зображується у вигляді пойменованого прямокутника. Усередині прямокутника записуються імена полів

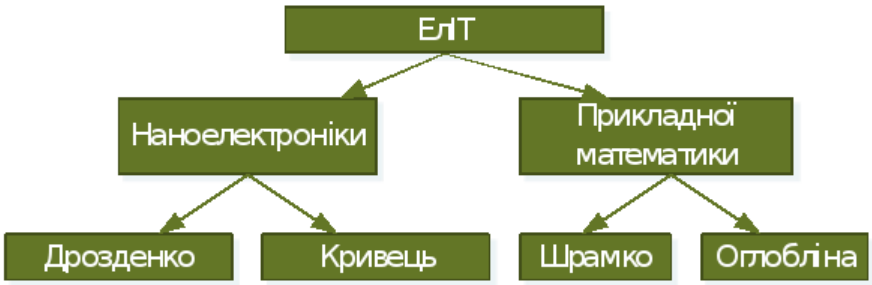
Ієрархічний зв'язок між сегментами позначається лініями зі стрілками, що проведені від батьківського сегмента до дочірнього. Батьківські сегменти, як правило, розміщують над дочірніми.



Екземпляр ієрархічної схеми даних складається з одного екземпляра кореневого сегмента і, можливо, кількох екземплярів дочірніх сегментів для кожного екземпляра батьківського сегмента. Припускається існування таких зв'язків між екземплярами сегментів:

- кожний екземпляр будь-якого сегмента підпорядкований одному екземпляру батьківського сегмента;
- екземпляр будь-якого сегмента (окрім кореневого) не може існувати без відповідного екземпляра батьківського сегмента;
- кожний екземпляр сегмента зв'язаний (підпорядковує собі) з усіма екземплярами дочірніх сегментів;
- екземпляри одного сегмента, зв'язані з одним екземпляром батьківського сегмента, можуть бути зв'язані між собою в ланцюжок, що дає змогу виконувати їхнє послідовне перебирання у межах усіх сегментів, породжених з одного початкового.

У такий спосіб ієрархічна впорядкованість сегментів створює зв'язок «один-до-багатьох» між екземплярами батьківського і дочірнього сегментів.



Ієрархічна схема інколи має розгалуження. У подібному випадку на рівні схеми батьківський сегмент може зв'язуватися з кількома дочірніми сегментами.

Ієрархічна структура даних – це сукупність ієрархічної схеми даних та всіх можливих екземплярів цієї схеми. Сукупність ієрархічних структур даних називається **ієрархічною базою даних**.

Операції над ієрархічною структурою.

Операції, що виконуються над ієрархічною структурою, поділяють на дві групи, операції пошуку (чи вибирання) та операції оновлення даних (або маніпулювання ними).

Зазначимо кілька базових принципів, характерних для мов маніпулювання даними ієрархічних систем.

Об’єктом маніпулювання є екземпляр сегмента.

У результаті успішного виконання пошукових операцій визначається як *поточний екземпляр сегмента,* який може відігравати роль стартової позиції для операцій маніпулювання; спочатку поточним є кореневий екземпляр сегмента.

Пошук необхідного сегмента відбувається за «навігаційним» принципом: необхідно прокласти шлях від кореневого екземпляра сегмента (або ж поточного) до шуканого, перевіряючи умови, накладені на значення полів сегментів, які розташовані на шляху, що будується. Операція в ієрархічній моделі визначається шляхом програмування «навігації» структурою даних.

Обмеження цілісності.

В ієрархічній моделі даних автоматично підтримується цілісність посилань між батьками та нащадками. Основне правило: ніякий нащадок не може існувати без свого батька.

Відмітимо, що аналогічна підтримка цілісності за посиланням між записами без зв'язку «батько-нащадок», не забезпечується. Прикладом такого зовнішнього посилання є вміст поля Посада в екземплярі типу запису Викладач.

Переваги та недоліки ієрархічної моделі.

До *переваг* ієрархічної моделі належать:

- розвинені низькорівневі засоби керування даними в зовнішній пам'яті;
- можливість побудови ефективних прикладних систем;
- економне використання пам'яті.

Зазначимо, що ієрархічна модель має також певні недоліки, які описані нижче.

Асиметрія пошуку за симетричними запитами. Програми пошуку викладачів за кафедрами та кафедр за викладачами принципово відрізняються: в першому випадку структура ієрархії збігається зі структурою пошуку, а в другому – ні.

Залежність між пошуком та відповідністю ієрархічної структури наявним зв'язкам у предметній області. Якщо ієрархічна структура відповідає структурі зв'язків ПО, то записувати пошукові вирази набагато легше.

Низький рівень мови запитів і маніпулювання даними. Оскільки маємо справу з мовою програмування процедурного типу, результатом однієї пошукової операції є один екземпляр сегмента даних, або ж один екземпляр ієрархічного шляху до заданого екземпляра сегмента. Це вказує на низьку селективну потужність мови.

Аномалії додавання, видалення та оновлення даних. Не можна здійснити операцію додавання сегмента ВИКЛАДАЧ без зазначення сегмента КАФЕДРА; не можна видалити сегмент КАФЕДРА, на видаляючи сегментів ВИКЛАДАЧ; після оновлення даних один і той самий екземпляр сутності зображується у вигляді багатьох екземплярів об'єктів бази даних.

Дублювання даних. Якщо об'єкти предметної області мають зв'язки типу «один-до-одного» або «один-до-багатьох», то ієрархічна структура дає змогу

зображувати дані без дублювання; проте, якщо є зв'язки типу «багато-до-багатьох», то дублювання даних під час відображення в ієрархічній моделі неминуче. Крім згаданих основних недоліків ієрархічної моделі слід зазначити також складність реалізації гнучких механізмів захисту даних, цілісності та несуперечності й «дружніх» інтерфейсів користувача.

Основні недоліки ієрархічної моделі пов'язані з тим, що не всі предметні області мають чітко виражену ієрархічну структуру

Мережева модель даних

Мережева модель даних є розширенням ієрархічної моделі й призначена для адекватного моделювання зв'язків між сутностями типу «багато-до-багатьох».

Окрім формальної нотації для мережевої моделі (мова опису даних – МОД) та пов'язаних з нею певних ключових концепцій, для мережевої моделі були запропоновані МОД підсхеми для означення зовнішнього відображення концептуальної схеми бази даних та мову опису збережених даних (МОЗД) для означення способів зберігання даних на носіях. Сама концептуальна схема описується за допомогою МОД. Запропонована була й мова маніпулювання даними (ММД) для написання прикладних програм, що взаємодіють з базою даних у термінах зовнішньої схеми (підсхеми).

Мережна структура даних.

Мережева структура даних є сукупністю схеми та екземпляра схеми. У свою чергу мережева схема формується з полів даних, типів записів і наборів, які також мають свої екземпляри. Власне з екземплярів полів, записів та наборів складається екземпляр схеми.

Елементарною одиницею даних мережної (так само, як ієрархічної) структури є пойменоване *поле даних.*

Тип запису – це пойменована впорядкована сукупність імен полів. Екземпляр запису (аналог сегмента в ієрархічній структурі даних) – це впорядкована сукупність значень полів запису. Екземпляр запису є одиницею доступу до даних мережної структури.

Набір – пойменований дворівневий ієрархічний зв'язок типів записів. Із дворівневих наборів можуть будуватися багаторівневі ієрархії та мережні структури. Кожний *тип набору* – це сукупність зв'язків між двома або кількома типами записів, де один тип запису оголошується власником, а інший (або кілька інших) – членами типу набору. Екземпляр набору містить один екземпляр запису-власника і довільну кількість екземплярів кожного типу запису-члена набору. Отже, набір описує дворівневий ієрархічний зв'язок типу «один-до-багатьох».

Типи наборів можуть використовуватися для створення багаторівневих ієрархічних або мережевих структур. Для опису будь-якої n-рівневої ієрархії потрібно принаймні n-1 наборів. Один тип запису може мати кілька батьківських записів, якщо вони є власниками різних типів наборів, тобто запис може бути членом багатьох наборів і мати декілька записів-власників. Так формуються мережеві структури.

Мережева структура може відображувати цикли та петлі. Циклом називається конфігурація, в якій предок типу запису є водночас його нащадком.

Петля – це структура, де один тип запису є водночас власником і членом в одному типі набору. Тоді навколо типу запису утворюється петля.

Операції над мережевою структурою.

Операції над мережевою структурою даних концептуально аналогічні до операцій над ієрархічною структурою. Мова маніпулювання даними в мережній структурі також є низькорівневою, тобто її використання передбачає програмування процесів «навігації» структурою. Кожний тип запису в цій моделі, на відміну від ієрархічної, може мати поточний екземпляр запису, відносно якого можна організовувати подальше перебирання екземплярів, окрім того, кожний тип набору може мати поточний екземпляр набору.

Згідно з пропозиціями CODASYL, навігація структурою та вибирання даних виконуються в два етапи. Спочатку за допомогою послідовності операторів FIND визначається розташування екземпляра запису необхідного типу, тобто бажаний запис стає поточним. Потім командою GET можна вибрати поточний запис, вказуючи (в разі потреби) його поля. Команда FIND насправді є сукупністю команд, об’єднаних спільною метою, – визначити місцезнаходження конкретного запису за вказаною стратегією. Оператори FIND дають змогу:

- знаходити запис за ключем або значеннями полів;
- послідовно переглядати записи вказаного типу, щоб знайти всі записи із заданими значеннями полів;

- послідовно переглядати всі записи-члени екземпляра набору;
- переглядати записи-члени екземпляра набору, які мають специфіковані значення в певних полях;
- знаходити власника даного запису в наборі.

Переваги та недоліки мережної моделі.

Для мережевої моделі даних властиві майже ті самі переваги та недоліки, що й для ієрархічної моделі. Принциповою відмінністю мережевої структури від ієрархічної є можливість безпосередньо відображувати складніші типи зв'язків. Натомість ієрархічні системи простіші в реалізації та експлуатації.

Лекція 5

Первинний аналіз даних. Виявлення основних структур. Вибір найвагоміших факторів моделі. Виявлення відхилень та аномалій. Робота з пропущеними значеннями. Статистична обробка даних

Процес первинної даних поділяється на такі етапи:

- Імпорт набору даних.
- Заповнення пропущених даних.
- Кодування категоріальних даних.
- Розбиття набору даних.
- Масштабування характеристик (нормалізація та стандартизація).

1. Імпорт даних

Досить часто статистичні дані зберігаються у форматі .csv, тому доцільно знати як імпортувати дані цього типу

2. Робота з пропущеною інформацією

Якщо в строкових даних є порожні комірки, необхідно їх замінити на NA значення

3. Кодування категоріальних змінних

Кодування означає перетворення строкових даних у числові дані.

Вона дозволяє створити новий масив числових значень, що ставляться у відповідність старому набору строкових значень певної факторної змінної (стовпця)

4. Розбиття даних на навчальну та тестову вибірки

Навчальна вибірка: частина даних, на якій реалізується наша модель (навчається).(≈80%)

Тестова набір: частина даних, на основі якої оцінюється якість моделі (≈20%).

5. Масштабування даних

Зазвичай у більшості наборів даних об’єкти, також відомі як вхідні дані, не мають однакового масштабу.

Є кілька способів для масштабування даних. Найбільш використовуваним є метод стандартизації та нормалізації.

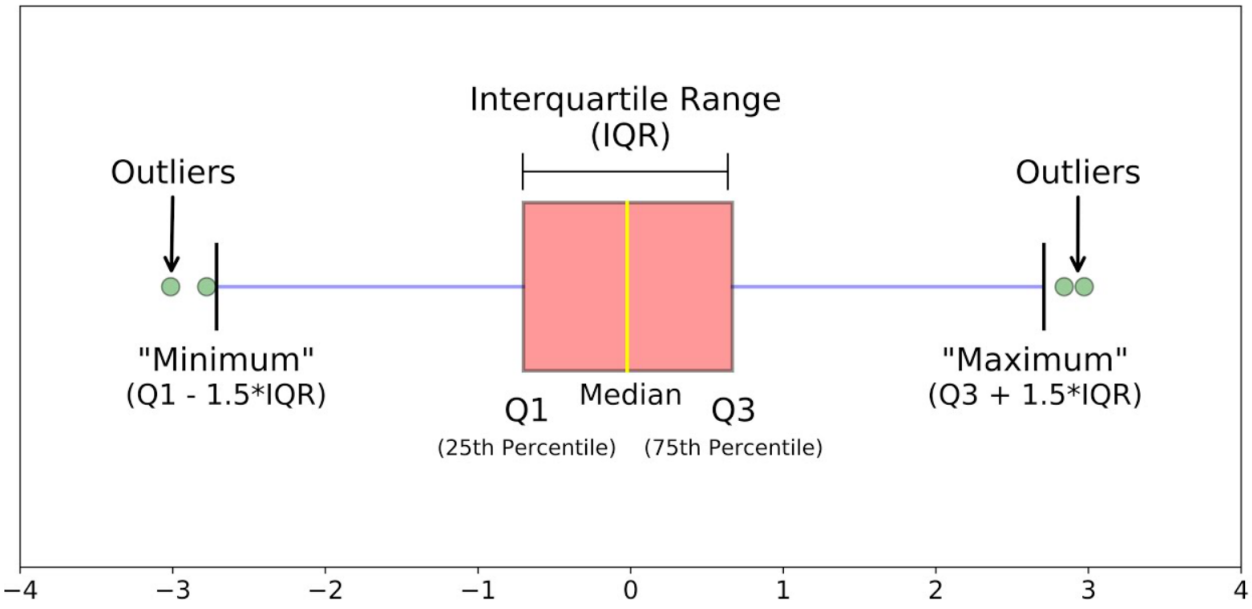
Техніка нормалізації використовується, коли дані розподілені нормально, тоді як стандартизація працює як з нормально розподіленими, так і з даними, які не є нормально розподіленими.

Standardisation	Normalisation
$x_{\text{stand}} = \dfrac{x - \text{mean}(x)}{\text{standard deviation }(x)}$	$x_{\text{norm}} = \dfrac{x - \min(x)}{\max(x) - \min(x)}$

6. Описова статистика

Графічно описова статистика добре подається такими типами діаграм: boxplot та violin plot.

Box plot ще називають «**ящик з вусами**»



Лінія по середині – ***медіана*** даних. Доцільно використовувати саме її, а не середнє значення, через її стійкість до викидів. Перший квартиль дорівнює 25% відсоткам, тобто 25% точок вибірки знаходяться нижче цього значення. Третій

квартиль дорівнює 75% відсоткам, тобто 75% точок вибірки знаходяться нижче цього значення. Мінімальне та максимальне значення є верхньою та нижньою межами діапазону даних.

Якщо ящик з вусами ***короткий***, то це означає, що більшість точок даних схожі, тому що ми маємо багато значень у маленькому діапазоні

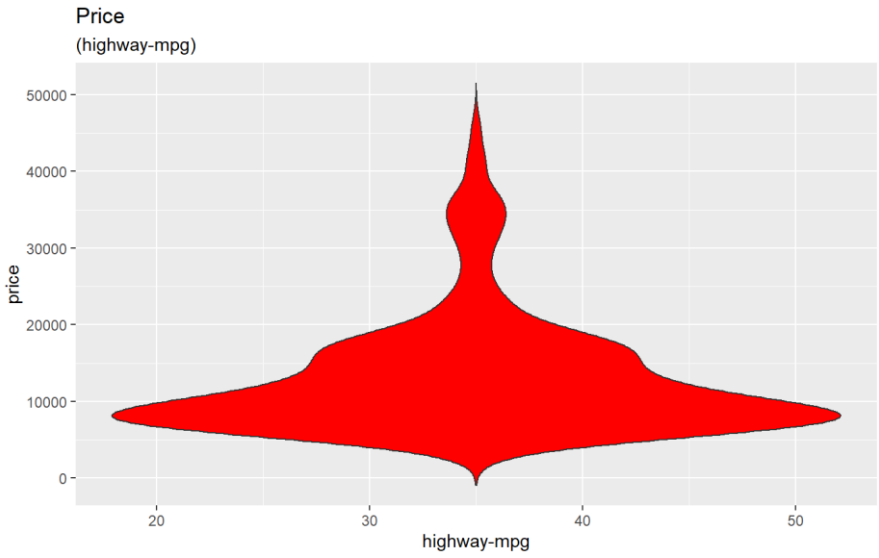
Якщо ящик з вусами ***довгий***, це означає, більшість точок вибірки відрізняються, оскільки значення поширені у широкому діапазоні

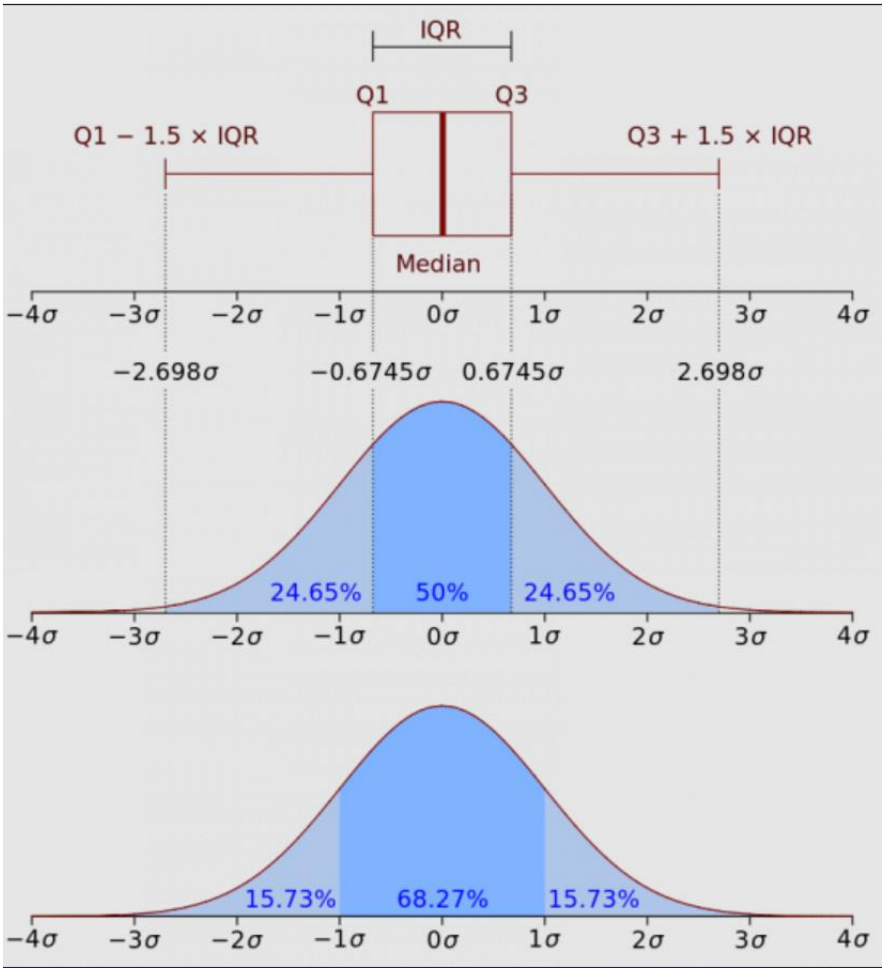
Якщо значення медіани *ближче до нижнього квартилю*, більшість даних має низьке значення. Якщо значення медіани *ближче до верхнього квартилю*, більшість даних має високе значення. Інакше кажучи, якщо лінія медіани знаходиться не в центрі скриньки, то це є показником того, що дані *асиметричні*.

Якщо *вуса дуже довгі*, то це означає, що дані мають високий рівень розкиду середньоквадратичного відхилення і дисперсії. Тобто значення дуже поширені і мають явні відмінності. Якщо вуса довгі тільки на одній стороні, дані можуть сильно відрізнятися тільки в одному напрямку.

Альтернативою до «ящика з вусами» є діаграма-«скрипка» **Violin plot**. В деякому сенсі вона більш репрезентативна, оскільки її основа – це графік щільності розподілу, який додатково дзеркально відображений.

Порівнюючи Boxplot та щільність розподілу, можна зрозуміти як розподілені дані, де знаходяться викиди відносно нормально розподілених даних.





7. Розвідувальний аналіз

Розвідувальний аналіз включає в себе наступні відомості про дані:

- виявлення викидів
- перевірка однорідності групових дисперсій
- перевірка на нормальність розподілу даних
- виявлення колінеарності між змінними

Виявлення викидів

Виявляти викиди можна використовуючи BoxPlot

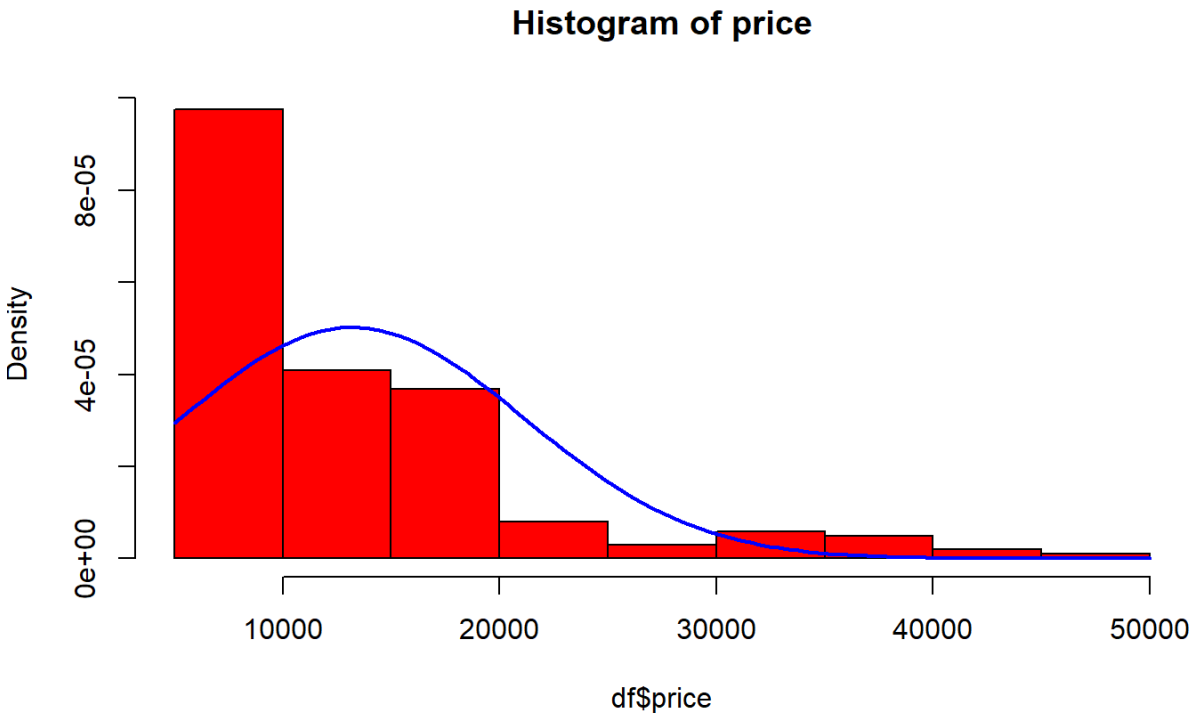
Перевірка однорідності групових дисперсій

Наприклад, можна використовувати ***тест Бартлетта або тест Флігнера*** для формальної перевірки нульової гіпотези про рівність дисперсій в k групах:

Оскільки p-value менше за рівень значущості (наприклад, 0.05), то це означає, що нульова гіпотеза (про однорідність дисперсій) може бути відхилена

Перевірка нормальності розподілу

Графічним способом можна побудувати гістограму.



Можна зробити висновок, що розподіл не схожий на нормальний

Можна перевірити нормальність розподілу за допомогою статистичних тестів

Скористаємося ***тестом Шапіро-Уілка***

Оцінка по p-value, якщо воно менше за рівень значущості (наприклад, 0.05), то це означає, що нульова гіпотеза (про нормальність розподілу даних) може бути відхилена

Альтернативні тести:

тест Андерсона-Дарлінга

тест Крамера фон Мізеса

тест Колмогорова-Смирнова в модифікації Лілієфорса

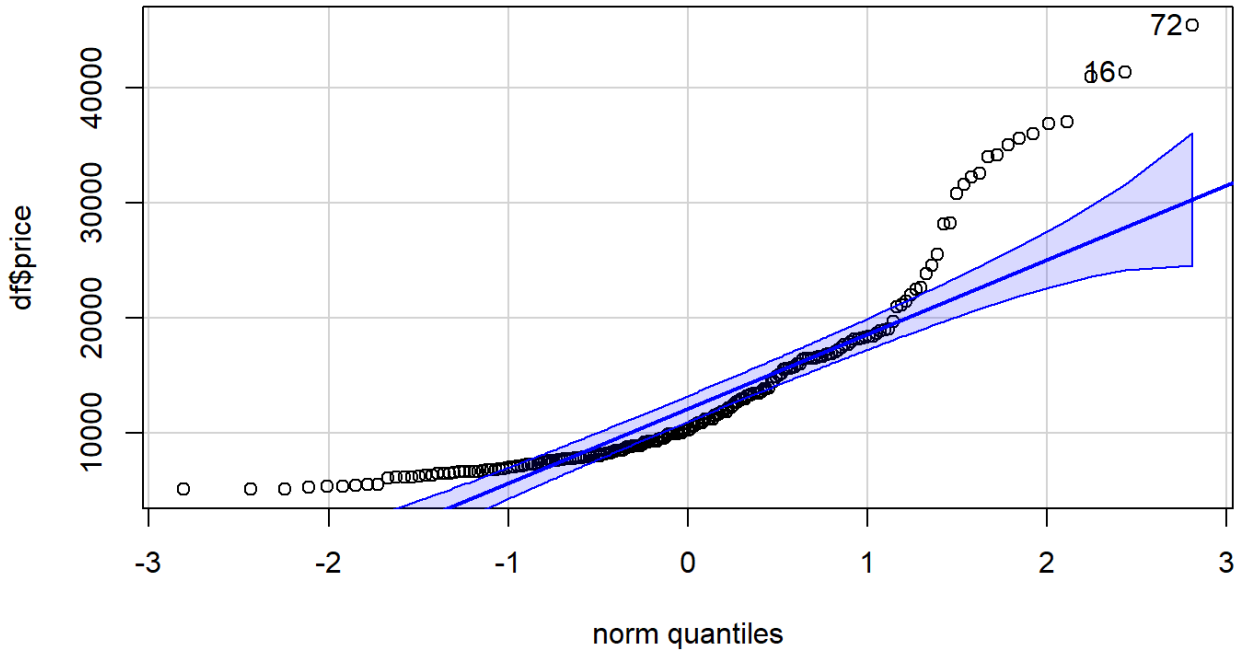
критерій хі-квадрат Пірсона

тест Шапіро-Франсія (див. Thode 2002)

Q-Q графік (квантиль-квантиль).

Квантиль – частина випадків, менша певного значення.

Також використовується графічний спосіб перевірки характеру розподілу даних – побудова графіків квантилей (Q-Q plots, Quantile-Quantile plots). На таких графіках зображуються кванти двох розподілів - емпіричного (тобто побудованого за аналізованими даними) і теоретично очікуваного стандартного нормального розподілу. При нормальному розподілі змінної точки, що перевіряється, на графіку квантилі ***повинні вишиковуватися в пряму лінію, що виходить під улом 45*** градусів з лівого нижнього кута графіку. Графіки квантилей особливо корисні при роботі з невеликими за розмірами сукупностями, для яких неможливо побудувати гістограми, що набувають будь-якої вираженої форми.



Виявлення колінеарності між змінними.

Коефіцієнт кореляції Пірсона (позначають «r») — в статистиці, показник кореляції (лінійної залежності) між двома змінними X та Y, який набуває значень від −1 до +1 включно

Коефіцієнт кореляції рангу Спірмена — непараметрична міра статистичної залежності між двома змінними. Він оцінює наскільки добре можна

описати відношення між двома змінними за допомогою монотонної функції. Якщо немає повторних значень даних, то коефіцієнт Спірмена дорівнює 1 або −1, це відбувається коли кожна змінна є монотонною функцією від іншої змінної. Кореляція Спірмена рівна 1, коли дві змінні монотонно пов'язані між собою, навіть якщо це відношення не є лінійним. Кореляція Спірмена є менш чутливою, ніж кореляція Пірсона відносно сильних викидів, які знаходяться в кінці обох зразків

8. Пониження розмірності

Вважається, що модель варто навчати на якомога більшому обсязі різноманітних даних. Але іноді, якщо інформації стає надто багато, вона гальмує чи зупиняє процес навчання. Щоб модель працювала нормально, потрібно зменшити кількість ознак, вибравши найефективніші. Для цього застосовують методи зниження розмірності.

Зниження розмірності – метод підготовки даних перед навчанням моделі. Воно може бути виконане після очищення та масштабування даних.

Модель, навчена на даних із широким набором ознак, наражається на ризик перенавчання. Це призводить до зниження точності. Боротьба з перенавчанням – основна мета зменшення розмірності. Що менше припущень робить модель, тим вона простіше.

Переваги зниження розмірності:

- більш висока швидкість навчання;
- потрібно менше місця для зберігання;
- видалення надмірного шуму.

Для зниження розмірності використовують методи вибору та проектування змінних.

Вибір змінних – це найпростіший спосіб зменшення розмірності. А ***проектування*** – створення нових змінних на основі існуючих шляхом їхнього перетворення.

Метод порогового відхилення (Variance Threshold) – простий спосіб вибору змінних. Він відкидає всі ознаки, у яких дисперсія не перевищує заданого порогового значення. Поріг вибирається з врахуванням даних.

Маленька дисперсія часто зустрічається у ознак, в яких значення багатьох рядків збігається.

Метод одновимірного вибору ознак (Univariate Feature Selection) використовує статистичні тести. Називається одновимірним, тому що аналізує змінні по черзі, порівнюючи їх із цільовим показником. Ознаки, які слабко корелюють з ним, відкидаються.

Приклади статистичних тестів – кореляція Пірсона, кореляція відстані, дисперсійний аналіз та хі-квадрат.

Найпоширеніші методи проектування змінних використовують лінійні перетворення. Наприклад, **Метод головних компонент PCA (Principal Component Analysis)**. Його застосовують для зменшення розмірності континуальних даних (температура, вага, рівень гемоглобіну у крові). Він дозволяє знизити втрату інформації при зменшенні розмірності.

PCA застосовується, якщо деякі ознаки корелюють одна з одною. Він створює їх незалежні лінійні комбінації. Часто дисперсії отриманих ознак сильно відрізняються. У такому разі ознаки з низькою дисперсією можна відкинути, оскільки вони менш інформативні.

Цей метод вирішує дві задачі: створює більш інформативні ознаки та зменшує шум у даних.

Факторний аналіз (Factor analysis), як і PCA створює нові ознаки, комбінуючи існуючі. Його відмінність у тому, що FA передбачає наявність реальних прихованих чинників, які визначають спостережувальні . Крім того, він пояснює частину дисперсії параметрів, що спостерігаються, помилками вимірювання, а частину – реальною дисперсією прихованих факторів.

Дані мають бути кількісними, вимірюватися за інтегральною шкалою чи шкалою відношень. Категоріальні дані (релігія, місце народження, стать) не підходять для цього методу.

LDA (Linear Discriminant Analysis) використовується для задач класифікації. Цей метод часто застосовують для зменшення розмірності при попередньої обробці даних. Мета – спроектувати набір даних на простір меншої розмірності з гарною роздільністю кластерів. В методі робиться припущення про нормально розподілені класи і рівні коваріації класів. Його можна застосовувати у випадках, коли класів більше ніж два.

Нелінійні методи зниження розмірності

Методи нелінійного перетворення використовуються, коли дані не лежать у лінійному підпросторі. До нелінійних методів відноситься, наприклад, **багатовимірне масштабування (MDS), ізометричне відображення об'єктів (Isomap), локально-лінійне вкладення (LLE), власне відображення Гессе** та інші.

Зведена інформація за результатами PCA містить наступне: стандартні відхилення головних компонент, частка дисперсії вихідних даних, яку пояснює головна компонента, та кумулятивна частку поясненої дисперсії – дисперсія вихідних даних, яка пояснюється поточною та попередніми головними компонентами.

Зазвичай вважається, що потрібно залишати стільки головних компонент, скільки можна змістовно проінтерпретувати. Але існують і більш формальні критерії. Так, наприклад, за правилом Кайзера потрібно залишати стільки головних компонент, скільки є компонент з дисперсією (власним числом) більше 1. У нашому випадку таких PC дві.

Ще існує правило Кеттела за яким потрібно побудувати графік, по горизонтальній осі якого відкладено номери головних компонент, а по вертикальній – їх дисперсії (власні значення коварійної матриці):

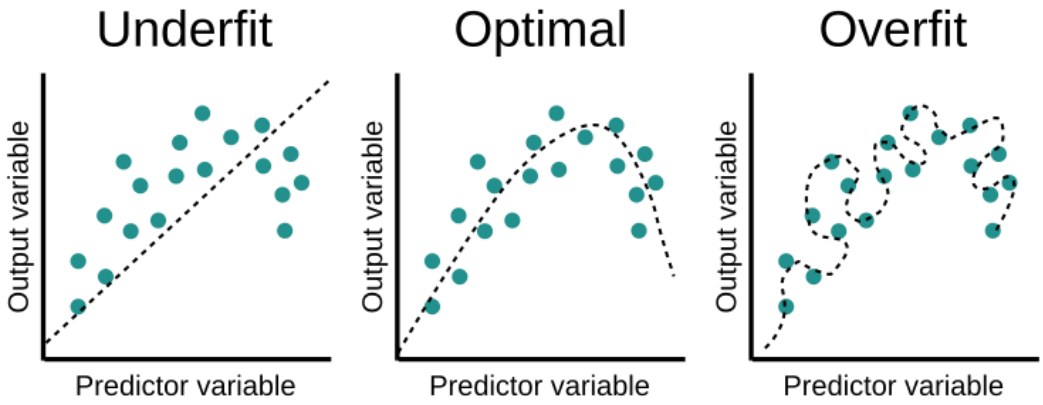
Зупинитись потрібною на тій головній компоненті, яка знаходиться на “згибі” або до нього.

Лекція 6.

Навчання моделі та регуляризація

1. Недонавчання та перенавчання моделі

Алгоритми машинного навчання використовуються для побудови моделі як параметричного сімейства функцій, що описують залежність, яка спостерігається в даних. Задача обрати із параметричного сімейства функцій модель, яка краще описує дані.



Недонавчання (Underfitting) – ситуація, коли в параметричному сімействі функцій не вдається знайти функцію, яка добре описує дані. Найчастіша причина недонавчання – коли складність природи даних вища, ніж складність природи моделі, яку створив дослідник. Прикладом недонавченої моделі дуже часто є проста лінійна регресія, яка вхоплює лише трендову компоненту зміни процесу, проте не описує можливі коливання результуючого фактору. Тобто така модель має низький рівень узагальнення, достатньо великий рівень зміщення (Bias) та може не вхопити достатню кількість ключових шаблонів в навчальній виборці для якісного моделювання. Рішенням для такої проблеми буде ускладнення моделі та пошук кращого опису ознак, які є у даних.

Перенавчання (Overfitting) – протилежний до недонавчання ефект, коли модель занадто складна та універсальна. Наприклад, моделі нейронних мереж, містять мільйони параметрів, такі мережі навчаються на великих даних, тому іноді обсягу даних може не вистачити, щоб одночасно добре налаштувати всі параметри. У момент перенавчання спостерігається ситуація, коли при

визначених оптимальних параметрах моделі, які добре описують дані, на нових даних ця модель починає часто помилятися. Проблема перенавчання часто зустрічається, і пов'язана вона з тим, що аналітик намагається зробити вибір керуючись неповною інформацією. Наша вибірка завжди не повністю описує шукану залежність, і при спробах підлаштувати модель під дані є ризик враховувати те, що нехарактерно даному процесу, тобто, шумові явища. Високий рівень дисперсії

Вирішити проблему перенавчання можна використовуючи процес ***регуляризації***.

2. Регуляризація

Регуляризація – це спосіб ввести додаткову інформацію в умову некоректно поставленої задачі, а також попередити вплив шумів та перенавчання. Це ніби «штраф» за складність моделі.

L_1-регуляризація (Lasso Regression) і L2-регуляризація (Ridge Regression). Такі методи регуляризації використовують на лінійних моделях регресії та класифікації. При створенні лінійної моделі завжди треба наближати вектор коефіцієнтів моделі до нульового вектора. Якщо цього не робити, то може виникнути ефект перенавчання, коли в моделі великі значення коефіцієнтів, які в сумі компенсують один одного за рахунок знаку, то на навчальній вибірці це працює добре, але на тестових даних помилка достатньо велика. Для усунення цього ефекту вектор коефіцієнтів наближають до нуля, і робить це регулювання.

Поширити підхід регуляризації з лінійних моделей на загальні класи моделей непросто, тому для цього використовують байєсівський підхід, який пов’язаний із введенням апріорного розподілу ймовірностей у просторі параметрів моделі. Байєсовський підхід складно втілити, тому що апріорний розподіл треба звідкись взяти, а отже, необхідно приблизно розуміти тип залежності, яку необхідно відновити.

Особливості роботи з регуляризацією:

1. Функція втрат з урахуванням регуляризації потрібна лише для підбору коефіцієнтів. Не можна порівнювати функції втрат різних моделей.

2. Результати звичайних метрик на тренувальних даних поступатимуться таким для лінійної регресії. Це типово. Мета — досягти за рахунок цього кращої генералізації інформації із даних та підвищити точність на валідаційних даних.

3. Ознаки (features) мають бути нормовані чи стандартизовані.

L_1-регуляризація (Lasso Regression)

Lasso — це абревіатура від (least absolute shrinkage and selection operator), а регресія Lasso додає до класичного рівняння сумарне абсолютне значення величини коефіцієнтів як штрафну компоненту до функції втрат.

$$J_{LASSO} = \sum_i (y_i - \hat{y})^2 + \lambda \|\omega\|_1 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=0}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} |\beta_j| \to min$$

$$\text{для деякого } t > 0\text{: } \sum_{j=0}^{p} |\beta_j| < t$$

λ – це гіперпараметр (штраф), який підбирається, якщо він дорівнює нулю, то ми отримаємо звичайну лінійну регресію.

Цей тип регуляризації (L1) може привести до нульових коефіцієнтів, тобто деякими характеристиками повністю нехтують для оцінки результату. Отже, регресія Lasso не тільки допомагає зменшити перенавчання, але й може допомогти у виборі значимих характеристик для побудови функцій.

L_2-регуляризація (Ridge Regression, регресія Тихонова)

У Ridge-регресії (L_2-регресії, або регресії Тихонова) штраф – це сума квадратів коефіцієнтів при змінних. Штрафи зменшують ваги (коефіцієнти) відповідних незалежних змінних, але ніколи не обнуляють їх. Це означає, що шуми завжди впливатимуть на результат, але незначно, тому, на відміну від L_1-регресії, всі характеристики беруть участь у моделі

$$J_{RIDGE} = \sum_i (y_i - \hat{y})^2 + \lambda \|\omega\|_2 = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=0}^{p} \beta_j x_{ij} \right)^2 + \lambda \sum_{j=0}^{p} \beta_j^2 \to min$$

Графік регресії має деяке зміщення (не дисперсія) за рахунок додавання статистичної помилки, яка не впливає на точність, але дозволяє не перенавчитися. Якщо встановити великий коефіцієнт (лямбда), то система ніколи не навчиться, просто через те, що сильно зміститься. Але якщо лямбда буде маленькою, то і зсув буде маленьким, і весь сенс регуляризації так само буде мінімальним.

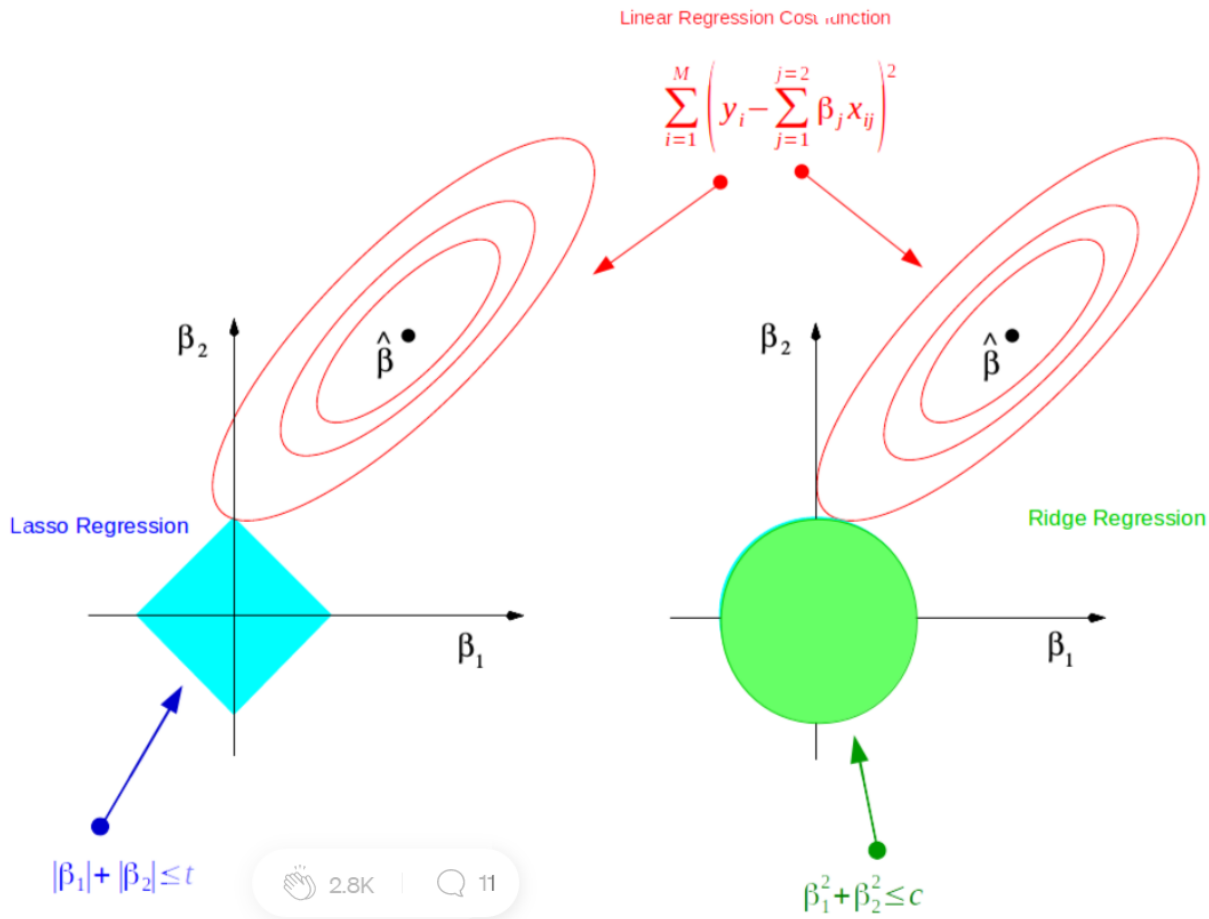
λ ще називають параметром згладжування. Він збалансовує крос-ентропійну функцію помилок та регуляризаційний штраф. Якщо значення λ велике, вагові коефіцієнти будуть прагнути до нуля, якщо значення λ мало або дорівнює нулю, то вагові коефіцієнти будуть просто прагнути до мінімізації крос-ентропійної функції помилок. Як правило, значення параметра встановлюють 0,1 або 1, або в районі між цими значеннями, але в основному його значення залежить від конкретних даних. Немає універсальної методики визначення значення параметра λ.

Вибір відповідної регресії

Lasso-регресію слід використовувати, коли є кілька характеристик з високою передбачувальною здатністю, а решта зайвих. Вона обнуляє непотрібні характеристики і залишає лише підмножину змінних.

Ridge-регресію краще застосовувати, коли передбачувана спроможність набору даних розподілена між різними характеристиками. Рідж-регресія не обнулює характеристики, які можуть бути корисними при складанні прогнозів, а просто зменшує вагу більшості змінних у моделі.

Обрати кращий варіант достатньо важко. Можна перебирати різні варіанти коефіцієнта λ, та оцінювати критерій якості, наприклад, показник MSE.



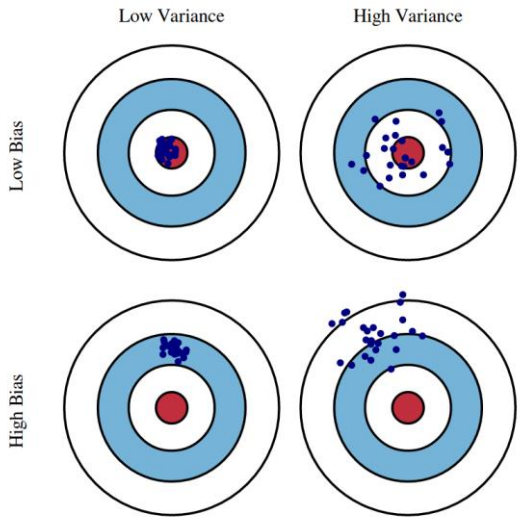
Elastic Net регуляризація – це гібридний підхід, який поєднує обидві регулярізації L_2 і L_1 методів Lasso та Ridge.

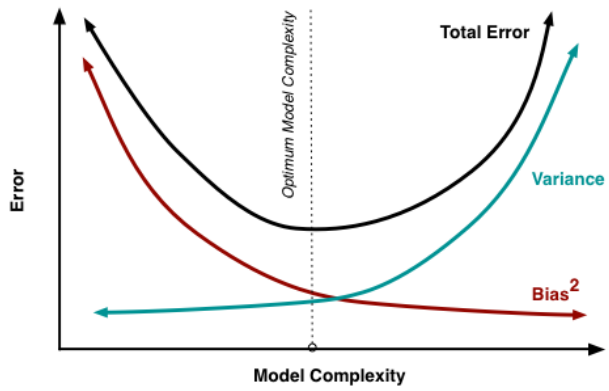
Elastic Net регресія завжди спрямована на мінімізацію такої функції втрат:

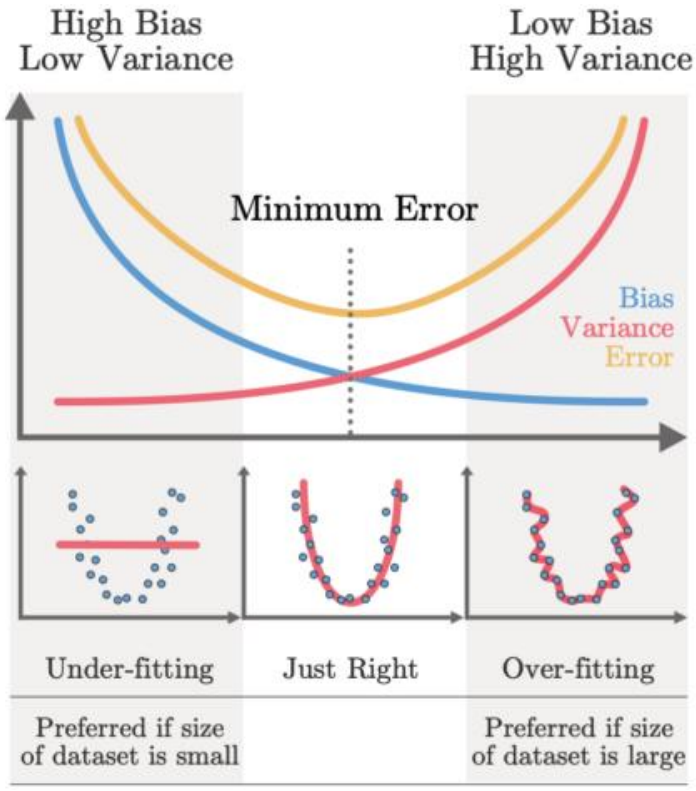
$$J_{ENet} = \frac{\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=0}^{p}\beta_j x_{ij}\right)^2}{2n} + \lambda\left(\frac{1-\alpha}{2}\sum_{j=0}^{p}\beta_j^2 + \alpha\sum_{j=0}^{p}|\beta_j|\right) \to min$$

Elastic Net також дозволяє нам налаштувати параметр альфа, де $\alpha = 0$ відповідає Ridge-регресії, а $\alpha = 1$ — Lasso-регресії. Таким чином, можна вибрати α -значення між 0 і 1 для оптимізації Elastic Net, і це зменшить деякі коефіцієнти, а деякі встановить 0 для розрідженого вибору. У регресії Elastic Net гіперпараметр λ здебільшого і сильно залежить від гіперпараметра α.

Процес регуляризації пов'язаний з дисперсією та зміщенням моделі, тому критерії якості оцінки регуляризованої моделі, такі як MSE, R^2 можуть не сильно покращуватись, а іноді і погіршуватись, проте це не свідчить про недоцільність процесу регуляризації.


Low Variance
High Variance
Low Bias
High Bias

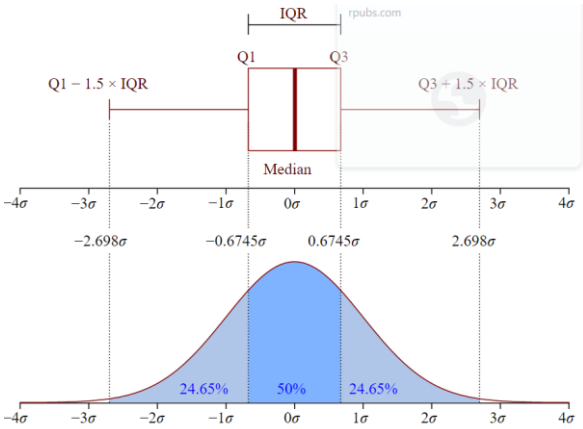

Total Error
Optimum Model Complexity
Variance
Bias^2
Error
Model Complexity

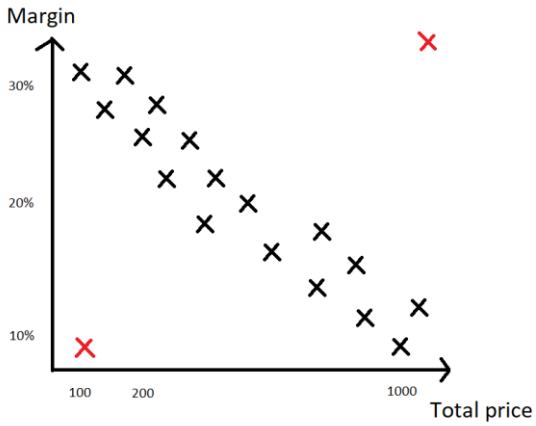

High Bias
Low Variance
Low Bias
High Variance
Minimum Error
Bias
Variance
Error
Under-fitting
Just Right
Over-fitting
Preferred if size of dataset is small
Preferred if size of dataset is large

Лекція 7

Виявлення викидів (outliers)

Викиди (outliers, anomalies) – це дані, які виходять за межі діапазону очікуваних значень. Поняття діапазону очікуваних значень суб'єктивне, з точки зору математики за визначенням математика Джона Тьюкі, це значення, яке виходить за межі діапазону $\pm 1.5 * IQR$, де IQR – це міжквартильний або інтерквартильний розмах (interquartile range), розраховується як різниця третього та першого квартилей





Причинами виникнення викидів можуть бути:

- «забруднені» зразки даних
- точки даних з різних вибірок
- неправильні методи відбору зразків даних,
- помилки при зборі або аналізі даних.

Викиди можуть значною мірою впливати на результати статистичних тестів, а отже, необхідно знаходити викиди в наборі даних.

Більшість статистичних тестів і методів моделювання чутливі до викидів, і їх необхідно позбуватися перед проведенням аналізу.

- наявність викидів може вплинути на нормальний розподіл набору даних, що є основним припущенням у більшості параметричних тестів на основі гіпотез

- наявність викидів спотворює середнє та стандартне відхилення набору даних
- в задачах машинного навчання наявність викидів може суттєво вплинути на кластеризацію деякими методами з лінійними алгоритмами, і може призвести до того, що кластер не буде добре відокремленим.

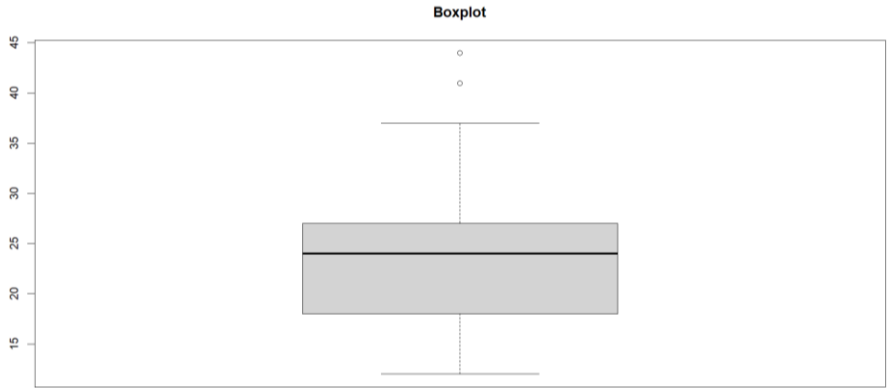
Видалення або збереження викидів здебільшого залежить від трьох факторів:

- Сфери/контексту вашого аналізу та дослідницького питання. У деяких галузях прийнято видаляти викиди, оскільки вони часто виникають через неправильне функціонування процесу. В інших галузях викиди зберігають, оскільки вони містять цінну інформацію. Також трапляється, що аналіз проводять двічі, один раз з викидами, а другий - без них, щоб оцінити їхній вплив на висновки. Якщо результати різко змінюються під впливом якихось впливових величин, це повинно застерегти дослідника від надмірно амбітних заяв.
- Чи є тести, які ви збираєтеся застосувати, стійкими до наявності викидів чи ні. Наприклад, кут нахилу простої лінійної регресії може суттєво змінюватися при наявності лише одного викиду, тоді як непараметричні тести, такі як тест Вілкоксона, зазвичай є стійкими до викидів.
- Наскільки віддалені викиди від інших спостережень? Деякі спостереження, які вважаються викидами (згідно з наведеними нижче методами), насправді не є екстремальними порівняно з іншими спостереженнями, тоді як інші потенційні викиди можуть бути дуже віддаленими від решти спостережень.

<u>Статистичні методи виявлення викидів</u>

Візуальні підходи, такі як гістограма, діаграма розсіювання (наприклад, Q-Q-діаграма) та бокс-діаграма, є найпростішим методом виявлення викидів.

1. ***Гістограма***
2. ***Ящик з вусами***



Бокс-діаграма допомагає візуалізувати кількісну змінну, відображаючи п'ять загальних підсумкових значень (мінімум, медіана, перший і третій квартилі та максимум), а також будь-яке спостереження, яке було класифіковано як ймовірний викид за допомогою критерію міжквартильного розмаху (IQR).

Критерій IQR означає, що всі спостереження вище $q_{0.75} + 1.5 \cdot IQR$ або нижче $q_{0.25} - 1.5 \cdot IQR$, (де $q_{0.25}$ та $q_{0.75}$ відповідають першому та третьому квартилю відповідно, а IQR – це різниця між третім та першим квартилем) вважаються потенційними викидами за R.

Іншими словами, всі спостереження за межами наступного інтервалу вважатимуться потенційними викидами:

$$I = [q_{0.25} - 1.5 \cdot IQR;\ q_{0.75} + 1.5 \cdot IQR]$$

Спостереження, які за критерієм IQR вважаються потенційними викидами, відображаються точками на полігональній діаграмі.

3. Перцентилі

За допомогою методу перцентилів усі спостереження, що лежать за межами інтервалу, утвореного 2,5 і 97,5 перцентилями, вважатимуться потенційними викидами. Інші перцентилі, такі як 1 і 99, або 5 і 95, також можуть бути використані для побудови інтервалу.

4. Z-scores

Якщо дані мають нормальний розподіл, можна використати z-score, який визначаються наступним чином

$$z_i = \frac{x_i - \bar{X}}{\sigma_X}$$

де $\bar{X}$ – середнє значення X, а σ_X – стандартне відхилення.

Це називається масштабуванням

Згідно з цим методом, будь-яке значення z-score:

- нижче -2 або вище 2 вважається *рідкісним*,
- нижче -3 або вище 3 вважається *вкрай рідкісним*

Іноді також використовують z-score нижче $-3{,}29$ або вище $3{,}29$ для виявлення викидів. Значення 3,29 походить від того, що 1 спостереження з 1000 виходить за межі цього інтервалу, якщо дані мають нормальний розподіл.

5. Фільтр Хемпеля (Hampel filter)

Фільтр Хемпеля, полягає у визначенні викидів значень за межами інтервалу I, утвореного медіаною, плюс/мінус 3 медіанними абсолютними відхиленнями (MAD)

$$I = [median - 3 \cdot MAD; median + 3 \cdot MAD]$$

де MAD визначається як медіана абсолютних відхилень від медіани даних

$$\tilde{X} = median\ (X)$$

$$MAD = median\left(\left|X_i - \tilde{X}\right|\right)$$

Наступні три статистичні тести:

- Тест Граббса
- Тест Діксона
- Тест Рознера

є частиною більш формальних методів виявлення викидів, оскільки всі вони передбачають обчислення тестової статистики, яка порівнюється з табличними критичними значеннями (які базуються на розмірі вибірки та бажаному рівні значущості). Використовувати їх можна у випадку, коли дані не мають пропущених значень та розподілені приблизно за нормальним законом.

6. Тест Граббса (Grubbs's Test)

Тест Граббса виявляє по одному викиду за раз (найбільше або найменше значення), тому нульова та альтернативні гіпотези виглядають наступним чином:

якщо ми хочемо перевірити найбільше значення:

$\boldsymbol{H_0}$*: Найбільше значення не є викидом*

$\boldsymbol{H_1}$*: Найбільше значення є викидом*

якщо ми хочемо перевірити найменше значення.

$\boldsymbol{H_0}$*: Найменше значення не є викидом*

$\boldsymbol{H_1}$*: Найменше значення є викидом*

Як і для будь-якого статистичного тесту, якщо p-value є меншим за обраний рівень значущості (зазвичай $\alpha = 0.05$), то нульова гіпотеза відхиляється, і ми робимо висновок, що найменше/найбільше значення є викидом.

І навпаки, якщо p-value більше або дорівнює рівню значущості, нульова гіпотеза не відхиляється, і ми робимо висновок, що на основі даних ми не відхиляємо гіпотезу про те, що найменше/найбільше значення не є викидом.

Тест Граббса не підходить для вибірок розміром 6 або менше ($n \leq 6$).

7. Q-тест Диксона (Dixon's test)

Q-тест Діксона використовується для перевірки того, чи є окреме низьке або високе значення викидом. Отже, якщо є підозра на наявність кількох викидів, тест слід виконувати для кожного з них окремо. Тест Діксона є найбільш корисним для невеликого розміру вибірки (зазвичай $n \leq 25$).

8. Тест χ^2(Хі-квадрат, Chi-squared) для викидів

Тест хі-квадрат для викидів можна використовувати для виявлення одного викиду у вхідному наборі даних. Наявність викидів у наборі даних може дати велику статистику тесту хі-квадрат і, отже, значуще p-value.

Тест хі-квадрат для викидів припускає, що дисперсія генеральної сукупності відома. Якщо вона не відома, дисперсію оцінюють за вибірковим набором даних. Він перевіряє гіпотези:

$\boldsymbol{H_0}$*: У наборі даних найбільше(найменше) значення не є викидом*

$\boldsymbol{H_1}$*: У наборі даних найбільше(найменше) значення є викидом*

9. Тест Рознера (Rosner's test)

Тест Рознера на викиди має наступні переваги:

- використовується для виявлення кількох викидів одночасно (на відміну від тестів Граббса і Діксона, які необхідно виконувати ітеративно для відсіювання кількох викидів),

- він призначений для уникнення проблеми маскування, коли викид, близький за значенням до іншого викиду, може залишитися невиявленим.

- тест Рознера є доцільним, коли розмір вибірки великий ($n \geq 20$).

Тест Рознера або узагальнений тест корисний для виявлення численних викидів в одновимірному наборі даних. Є достатньо точним для виявлення до 10 викидів.

Гіпотези тесту Рознера:

$\boldsymbol{H_0}$*: У наборі даних немає викидів*

$\boldsymbol{H_1}$*: У наборі даних є до k потенційних викидів*

Лекція 8

Поняття потокових даних та їх обробки. Огляд інструментів для роботи з потоковими даними

Потокові дані – це такі дані, які генеруються безперервно тисячами джерел та зазвичай надсилають дані одночасно, і невеликими розмірами.

Потокові дані включають велику кількість різноманітних даних, таких як логування дій створених клієнтами за допомогою мобільних або веб-додатків, купівля/продаж в e-commerce, активність гравців в іграх, інформацію з соціальних мереж, фінансові торговельні майданчики або також дані з давачів підключених пристроїв.

Прикладами областей, де застосування обробки потокових даних є доцільним:

– давачі транспортних засобів, промислового обладнання та сільськогосподарської техніки, що передають дані до системи, яка здійснює моніторинг та попередньо виявляє потенційні загрози та автоматично створює замовлення для заміни певних запчастин, що запобігає витримці обладнання;

– відстеження змін на фондовому ринку в режимі реального часу, обчислення ризику та автоматичний перерозподіл коштів на основі руху цін на акції;

– компанія, що займається розробкою онлайн-ігор збирає потокові дані про взаємодію між гравцями та грою і передає дані на свою ігрову платформу, яка аналізує дані в режимі реального часу та пропонує стимули для залучення своїх гравців на основі зібраних даних.

На сьогодні, існує широкий спектр технологій та засобів обробки потокових даних. Провідні компанії надають можливість здійснювати обробки таких даних за допомогою розроблених хмарних рішень, таких як: Amazon Kinesis, Apache Spark Streaming, Apache Storm, Apache Samza. На рисунку 1 зображено один зі способів організації обробки потокових даних з використанням Spark Streaming.

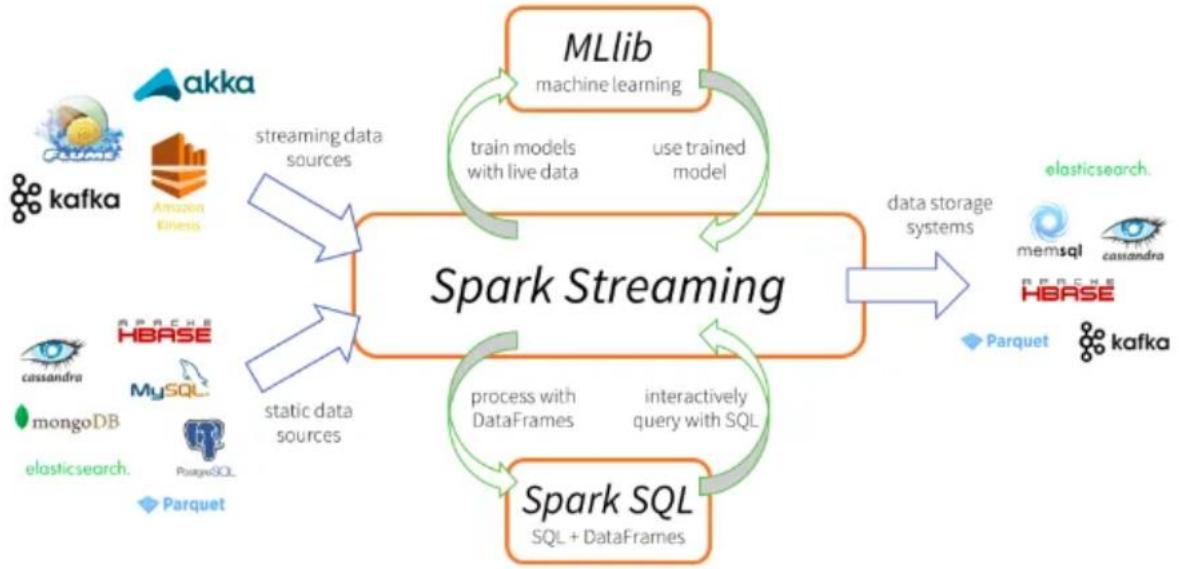


Рисунок 1 – Організація обробки потокових даних за допомогою Spark Streaming

При роботі з Apache Storm створюється граф обчислень реального часу, так звану топологію (topology), і передаємо його в кластер, де головний вузол розподіляє код між робочими вузлами для виконання. Основними елементами топології є spout і bolt. Spouts генерують потоки даних в формі незмінних пар ключ-значення, які називаються кортежами (tuple), а bolts виконують перетворення цих потоків (підрахунок, фільтрація, тощо). Bolts, у свою чергу, можуть передавати дані іншим bolt для виконання послідовних стадій обробки.

Spark Streaming це розширення базового Spark API, що дозволяє організувати високопродуктивну обробку потокових даних. Дані можуть надходити із багатьох джерел, таких як Kafka, Flume, Kinesis або TCP сокетів, і можуть бути оброблені за допомогою складних алгоритмів, виражених функціями високого рівня, такими як Map, Reduce, Join та Window. По завершенню процесу обробки, дані можна вивести у різні файлові системи, бази даних, або на приборні панелі. На рисунку 2 зображено архітектуру Spark Streaming.

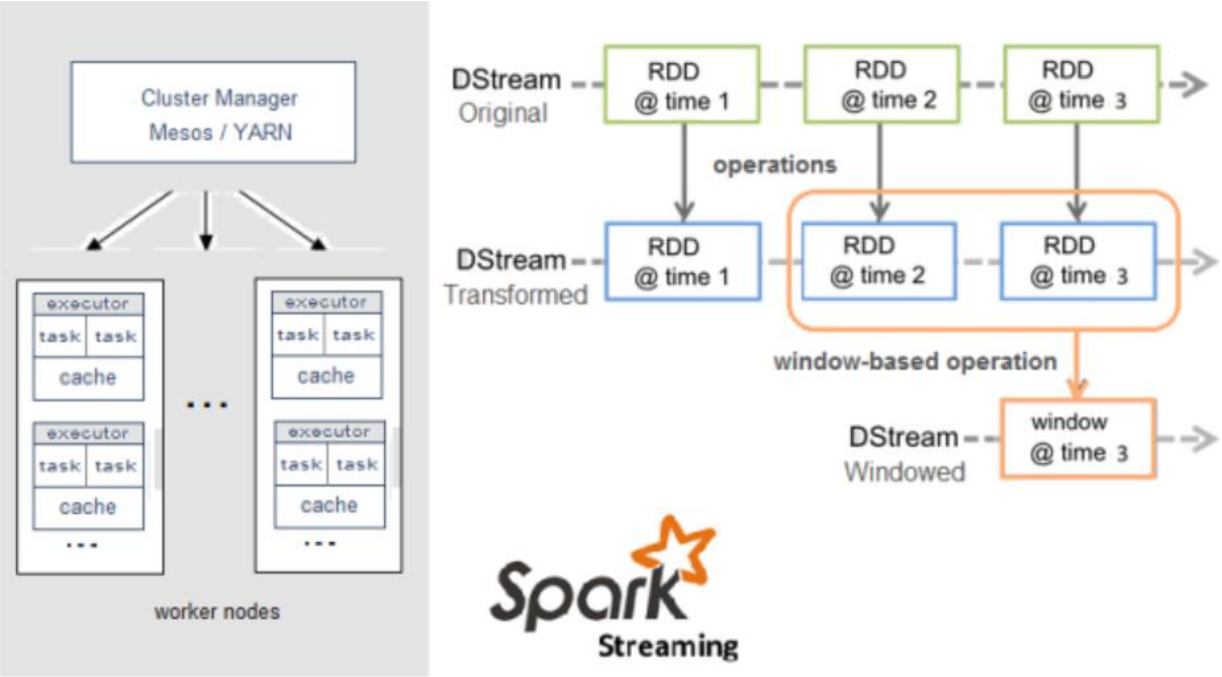


Рисунок 2 – Архітектура Spark Streaming

Концепція Apache Samza полягає в обробці повідомлень по мірі їх отримання. Потоковим примітивом Samza є не кортеж або DStream, а повідомлення (message). Потоки розбиваються на розділи (partition). Кожен розділ являє собою упорядковану послідовність доступних тільки для читання повідомлень. Кожне повідомлення має унікальний ідентифікатор. Система також підтримує пакетний режим (batching), який дозволяє послідовно приймати кілька повідомлень з одного розділу потоку. Модулі виконання і обміну повідомленнями Samza є можливими для підключення, тобто можуть бути замінені аналогами, але зазвичай використовуються YARN і Apache Kafka.

Всі три фреймворки чудово підходять для ефективної обробки потокових даних в реальному часі. Проте при виборі фреймворку слід керуватись наступними рекомендаціями.

Якщо потрібна високошвидкісна система обробки подій, що забезпечує інкрементні обчислення, Storm буде хорошим вибором. Якщо далі буде потрібно виконувати розподілені обчислення на вимогу, в той час як клієнт синхронно очікує результат, Storm надасть готову підсистему розподіленого віддаленого виклику процедур (distributed RPC).

Якщо необхідно збереження стану, в точності одноразова доставка повідомлень, і при цьому, не дуже турбує більш тривала затримка, тоді можемо обрати Spark Streaming. Цей фреймворк особливо підійде в тому випадку, якщо плануємо виконувати операції над графами, машинне навчання або доступ до SQL. Стек Apache Spark дозволяє використовувати спільно зі Streaming кілька інших бібліотек (Spark SQL, MLlib, GraphX) і реалізує зручну уніфіковану модель програмування. Зокрема, в поєднанні з потоковими алгоритмами, такими як потоковий метод k-середніх (streaming k-means), Spark може бути застосований для забезпечення прийняття рішень в реальному часі

РЕКОМЕНДОВАНИЙ ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Downloads – Oracle VM VirtualBox
2. https://downloads.cloudera.com/demo_vm/virtualbox/cloudera-quickstart-vm-5.4.2-0-virtualbox.zip
3. MapReduce Algorithm | Baeldung on Computer Science
4. A Beginners Introduction into MapReduce | by Dima Shulga | Towards Data Science
5. https://cran.r-project.org/bin/windows/base/
6. https://posit.co/download/rstudio-desktop/
7. https://www.tutorialspoint.com/r/index.htm
8. https://jtr13.github.io/cc20/data-preprocessing-and-feature-engineering-in-r.html
9. https://campus.datacamp.com/courses/introduction-to-machine-learning-in-r/how-much-will-i-earn?ex=3
10. https://baotramduong.medium.com/data-cleaning-and-preprocessing-in-r-efb7ef6c7ce6
11. https://medium.com/srijit-mukherjee/full-demo-of-overfitting-and-underfitting-in-r-70fd9b66a7d1
12. https://towardsdatascience.com/overfitting-vs-underfitting-a-complete-example-d05dd7e19765
13. https://www.geeksforgeeks.org/regularization-in-r-programming/
14. https://daviddalpiaz.github.io/r4sl/regularization.html
15. https://statsandr.com/blog/outliers-detection-in-r/
16. https://www.reneshbedre.com/blog/find-outliers.html
17. https://kristofsl.medium.com/anomaly-outlier-detection-using-isolation-forest-in-scala-9f5f4d6edff9
18. https://www.r-bloggers.com/2016/12/outlier-detection-and-treatment-with-r/
19. https://www.geeksforgeeks.org/outlier-analysis-in-r/
20. What is streaming data [Електронний ресурс] – Режим доступу до

ресурсу: https://aws.amazon.com/streaming-data

21. Spark Streaming [Електронний ресурс] – Режим доступу до ресурсу: https://databricks.com/glossary/what-is-spark-streaming

22. Сучасні методи обробки потокових даних [Електронний ресурс] – Режим доступу до ресурсу: https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/27300/7443.pdf?sequence=3&isAllowed=y