Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Використання методів машинного навчання до аналізу Big Data'

Текстова частина до курсової роботи
за спеціальністю „Програмна інженерія” 6.050103

Керівник курсової роботи
к.т.н., доц. Жежерун О. П
(прізвище та ініціали)

_____
(підпис)

“____” _____ 2021 р.

Виконав студент Савкін Г.І.
(прізвище та ініціали)
“____” _____ 2021 р.

м. Київ – 2021 рік

Календарний план виконання курсової роботи

Тема: Розробка нейронної мережі для розпізнавання графічних об’єктів

Календарний план виконання роботи:

№ п\п	Назва етапу дипломного проекту(роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень - листопад 2020р.	
2.	Огляд літератури за темою роботи	Листопад-лютий 2020р.	
3.	Аналіз методів машинного навчання	Лютий-березень 2020р.	
4.	Аналіз екосистеми Hadoop	Березень 2020р.	
5.	Написання текстової частини	Квітень 2020р.	
6.	Перегляд змісту роботи керівником		
7.	Створення презентації		
8.	Захист курсової роботи		

Студент Савкін Г.І. _____

Керівник Жежерун О.П. _____

“_____”_____ 2021 р.

Зміст

ВСТУП

Кількість інформації у світі зростає щодня, збільшується швидше ніж очікували аналітики. Соціальні мережі, активність на веб сторінках, банківські транзакції, тощо – кожна людина генерує масу інформації, до того ж часто дуже корисної для бізнесів.

Звичайно зберегти всю цю масу даних є непростою задачею, але ще складнішою часто є що саме робити з усім цим. Навіть якщо знайти охочих аналізувати отримане, ціна утримання таких робітників буде набагато вищою за отримані прибутки, до того ж аналіз з високою вірогідністю не буде точним.

Тут у гру вступає машинне навчання. Винайдене ще в минулому сторіччі, тільки зараз ця сфера починає розвиватися по справжньому. Тепер аналіз Big Data не є чимось неймовірним і може бути проведений за копійки на сучасних хмарних кластерах. З розвитком сучасних технологій Big Data та машинне навчання будуть продовжувати ставати невід’ємною частиною суспільства.

Мета цієї роботи – проаналізувати середу для роботи з цими технологіями, які переваги та недоліки мають ті чи інші підходи і як саме варто робити Big Data Аналіз.

РОЗДІЛ 1: Big Data

1.1 Історія Big Data

У природі завжди перемагає найсильніший, найспритніший та найвитриваліший, за єдиним виключенням. Людина змогла не тільки перемогти, а й підкорити собі цілий світ, за допомогою розуму. Наші древні пращури не тільки спостерігали за навколишнім світом, а й навчались від нього. Помітивши як маленька іскра висікається при ударі двох каменів вони навчились керувати вогнем, дослідивши гострі кігті диких тварин – навчились робити ножі та списи. Ця жага до знань ніколи не покинула людство. Коли в природі все, що можна було дослідити було досліджено, наш вид почав записувати і досліджувати власні дії, для збільшення кількості ресурсів, грошей, тощо. Так, наприклад, в 18000 р. до н. е. населення древньої Африки, в регіоні на території сучасної Уганди, підраховувало здобуту їжу та інші запаси, відмічаючи їх кількість на палицях та кістках, аби мати змогу передбачувати здобутий об’єм заздалегідь[1]. Схожі підрахунки робили й у Месопотамії, нотуючи кількість забитої худоби, зберігаючи глиняні маркери у відповідній кількості. Ці дії були першим відомим збереженням інформації з метою подальшого оброблення, першим кроком у сторону Big Data. Інші приклади включають у себе детальні записи про стан та склад армії в Римській Імперії для кращого розподілення військових сил, та створення першого великого масиву інформації у вигляді Александрійської бібліотеки.

З розвитком математики з’являлись все нові можливості для використання накопичених даних. Купці та підприємці ще у 3000 р до н. е. використовували ці знання для аналізу вигоди у торгівлі, але справжньої популярності ці методи набули лише після винайдення «подвійного запису» у 14 ст. У 1663 році англієць Джон Граунт, якого потім назвуть «батьком статистики», видав у світ книгу під назвою «Natural and Political Observations Made Upon the Bills of Mortality»[2]. В ній він аналізував так звані «Bills of Mortality» - документи з інформацією про

вмерлих, що друкувалися та поширювалися щочетверга – з ціллю виявити ознаки початку епідемії бубонної чуми задовго до її серйозного розповсюдження. Цю роботи вважають найпершим прикладом використання статистичних методів дослідження. Крім того її де факто можна назвати першим прикладом використання Big Data на практиці.

Але інформація у справді великих об’ємах з’явилась лише у 20 сторіччі, разом з винайденням комп’ютера. Завдяки цьому новому пристрою збір та обробку даних стало можливо проводити без виснажливої роботи зі сторони людини. Першим серйозним проектом, пов’язаним з великим об’ємом інформації (на цьому етапі з метою лише її збереження), стала система збереження персональних внесків громадян, з метою надання справедливої пенсії усім працівникам[3]. Замовлено проект було в 1937 році американським урядом, реалізувала його компанія IBM у вигляді масивної перфокартної машини.

Перші ж приклади обробки Big Data відносяться до сфери дешифрування. У 1943 році, в розпал Другої світової війни, британські вчені створили машину під назвою «Colossus», що була здатна розшифровувати німецькі повідомлення в багато десятків разів швидше за людину[4]. А після війни американська розвідка почала використовувати автоматичну оброку інформації для простішого аналізу розвідданих своїх шпигунів. Окрім цих великих проектів збір та обробка інформації були присутні хіба що у великих бізнесах, і то в більшості випадків лише з метою збереження інформації без будь якого конкретного застосування в майбутньому. Розвиток сфери був сильно обмежений тодішніми технологіями – малий максимальний об’єм збережених даних, повільні методи збору та швидкість обробки не давали вченим розв’язувати складніші завдання.

Найпростішою проблемою для вирішення було зберігання інформації. Навіть з найпримітивнішими методами можна було отримати достатньо велике сховище просто поєднавши багато малих сховищ в одне. Разом з ростом потрібного об’єму покращувались і характеристики кожного окремого елементу, такі як ємкість, швидкість та розмір. Так на перфокартах що використовувались

для перших інформаційних систем можна було вмістити всього лише 0.08 кілобайтів інформації, а вже у 1982 році з'явилися перші компакт-диски з об'ємом в 700000 кілобайт, та жорсткі диски, що могли вмістити в себе 10 мегабайтів інформації, яку до того ж можна було змінювати. Крім цього змінювався і підхід до збереження. Від групи файлів на диску, до реляційних баз даних, до data warehouse-ів, до сучасних data lake-ів, ці підходи постійно покращувалися та адаптувалися під все більшу кількість інформації.

Наступна проблема, швидкість обчислень, була подолана і схожий спосіб. У 1976 році комп'ютерний інженер Сеймур Крей, після ряду невдалих проектів створив перший у світі суперкомп'ютер Cray-1[5]. Обчислювальна потужність системи була феноменальною для того часу, досягала 133 мегафлопсів, що було в багато разів більше за альтернативи. Цей та наступні суперкомп'ютери дозволили обробляти набагато більше інформації ніж альтернативи. Навіть сьогодні для найсерйозніших проектів у сфері Big Data та машинного навчання використовують наступників Cray-1, наприклад японський Fugaku (з потужністю 415 петафлопсів) або американський Summit (266.7 петафлопсів).

Остання проблема – збір інформації – також була вирішена ще в 20-му сторіччі. У 1989 році британський вчений Тім Бернерс-Лі створив концепт Всесвітньої павутини – системи, в якій інформація, збережена на одних комп'ютерах, може бути отримана іншими через деяку середу передачі. Всесвітня павутина дозволяла отримувати та надсилати листи, зображення, музику, тощо. Її створення – дуже важливий крок для науки Big Data. Після її поширення кожний користувач став створювати інформацію – своєю поведінкою на веб сторінках, своїми інтернет покупками, своїми постами в соціальних мережах, тощо. Все більше і більше комп'ютерів підключались до мережі, і відповідно з'являлося все більше і більше інформації. Інформації, яку можна було зберегти, обробити та використати отриманий результат. Крім того і передача інформації стала тривіальною задачею. На даний момент все більше і більше інформації поступає з ще одного нового джерела – пристроїв Internet of things, різноманітних датчиків та

сенсорів, кількість яких постійно зростає, при тому з дуже високою швидкістю. Наприклад за минуле десятиріччя кількість даних згенерованих людиною зросла в 10 разів, проте кількість даних сенсорів за цей час зросла в 50 разів. Загальна ж кількість інформації досягла відмітки в 44 зета байти у 2020 році, а за прогнозами у 2025 році ця кількість збільшиться до 175 зета байтів[6].

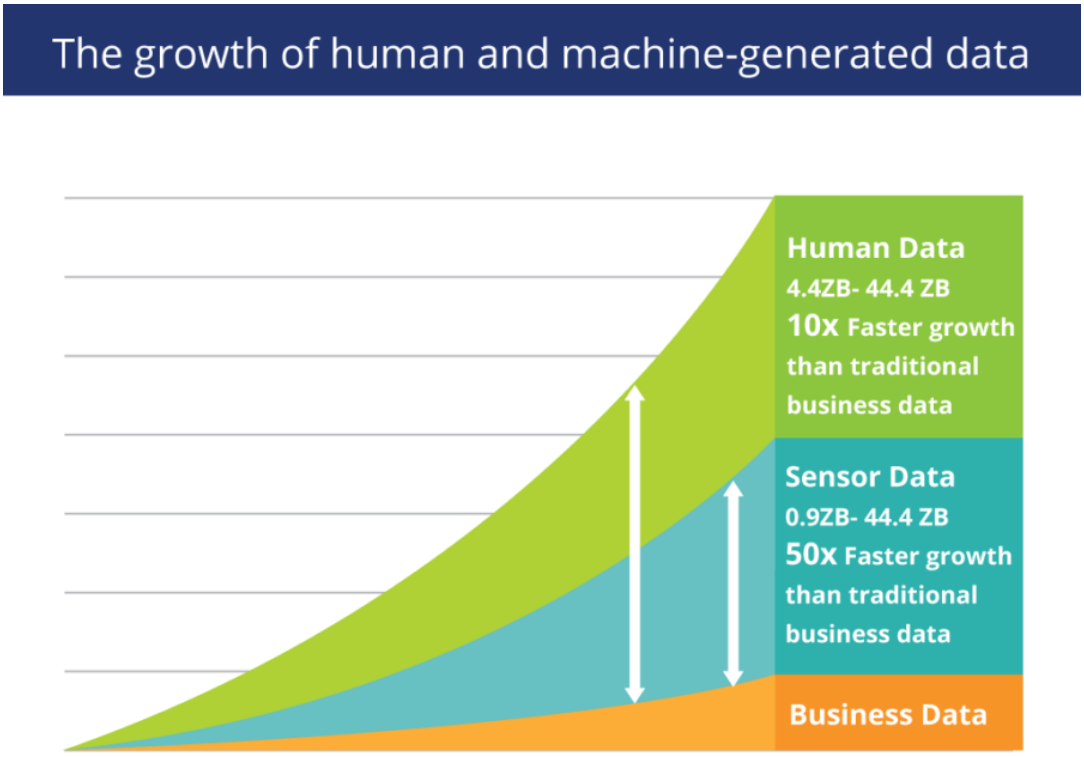


Рисунок 1. Ріст кількості даних за типом

1.2 Визначення терміну Big Data

Коли саме було вперше використано термін Big Data точно не відомо, але використовували його ще з ранніх років 1990-х. Через це й чіткого визначення в нього немає, що в свою чергу спричиняє певну незгоду серед спілки Big Data, а разом з цим і велику кількість визначень. Найпопулярніше - інтерпретація Gartner, яке ще називають правилом трьох V, визначене у 2001 році Дугласом Лейні[7]:

«Big data це такі дані, які поєднують в собі 3 фактори – дуже великий об’єм (Volume), дуже висока швидкість (Velocity), та дуже велика варіативність (Variety)»

У цьому визначені кожна V означає наступне:

- Volume (об’єм) – оброблювана інформація є постійним потоком, або накопиченням потоку
- Velocity (швидкість) – інформація має мати необхідність у швидкій обробці
- Variety (варіативність) – інформація перебуває у багатьох різних формах, а не лише в одній постійній.

Але це визначення, на думку багатьох, не покривало поняття повністю, тому було створено багато альтернативних, кількість V в яких варіюється від 4 до 11. Наприклад 4Vs від IBM, в якому було додано ще й правдивість (Veracity), оскільки багато клієнтів IBM зіткнулися з проблемою великої кількості проблем з джерелами інформації.

Ще одне V визначення від Microsoft, 6Vs, включає в себе ще 2 параметри: видимість (Visibility) та мінливість (Variability), які позначають наступне:

- Visibility – необхідність мати уявлення про усю існуючу інформацію щоб зробити правильні висновки
- Variability – висока складність датасету, тобто дані з великою кількістю змінних

На жаль і ці визначення не покривають повністю усю сферу, або описують її неточно. Наприклад, жодне з Vs визначень не торкається самої причини збору даних. Вони збираються не через те, що є можливість обробити більший об’єм з більшою швидкістю, а тому що є певна наукова чи бізнес задача, яку є потреба виконати.

Альтернативу пропонує Тімо Еліотт. Він зібрав та проаналізував велику кількість альтернативних визначень і розділив на 7 категорій[8], відповідно до того, що намагається визначити кожне. Вони є наступними:

- Оригінальне визначення Big Data (Vs) – визначення через певну кількість V
- Big Data як технологія – визначення через певні технології, такі як Hadoop та NoSQL
- Big Data у порівнянні з даними минулого – визначення концентруються на контрасті двох понять, проводять більш-менш чітку межу між даними та Big Data
- Big Data як сигнали – визначення орієнтоване на бізнеси, в яких Big Data визначається як інструмент для швидкого реагування на певні сигнали, або, як їх називає автор, «Дзеркало заднього виду для компаній»
- Big Data як можливість – розглядає термін як набір втрачених у минулому можливостей, які з сучасними технологіями більш ніж реальні
- Big Data як метафора – більш романтичне представлення терміну, порівнює процес аналізу Big Data з людським мозком або з нервовою системою планети
- Big Data як нова модна назва для старих речей – найбільш цинічне визначення, за яким усі Big Data проекти це просто старі Business Intelligence проекти, які з ростом у популярності нового терміну просто змінили назву

Що добре видно з такої великої кількості дуже різних визначень, так це то що поле Big Data не є чимось простим. Це не просто обробка великої кількості інформації, оскільки з розвитком технологій і кількість даних буде зростати, а щось більше, що поєднує в собі багато вимог та проблем.

1.3 Технології аналізу Big Data

Доцільно буде використати одне з зазначених вище визначень для подальшого розгляду цієї сфери. Найкращим варіантом у цій ситуації буде аналізувати Big Data через призму технологій, що були створені для її потреб. Звичайно, найбільшою та найважливішою частиною Big Data аналізу є машинне

навчання, проте його особливості та аплікацію буде розглянуто у окремому розділі. Натомість важливо розглянути іншу важливу частину процесу аналізу, а точніше розподілені обчислення та, як його продовження, хмарні обчислення.

1.3.1 Розподілені та хмарні обчислення

Розподілені обчислення – це технологія підвищення ефективності окремого комп’ютера шляхом об’єднання його в певний кластер з кількома іншими. Пристрої у кластері називають воркерами (від англійського слова «працівник»). Сформована таким чином система має обчислювальну потужність майже еквівалентну сумарній потужності кожного окремого воркера. Альтернативно термін може бути використано для опису функціонування багатоядерного процесора, в якому фактично відбувається такий самий процес, тільки замість окремих пристроїв в тандемі працюють ядра. Порівняно з альтернативою – централізованими обчисленнями, при яких створюється один дуже потужний воркер– розподілені обчислення мають велику перевагу у масштабованості та у надмірності. Масштабованість дозволяє просто підвищити потужність системи шляхом додання нових воркерів. Надмірність полягає у наявності кількох воркерів з однаковим функціоналом, які можуть змінити одне одного у випадку несправності. Крім того невелика ціна окремого воркера робить цей метод ще й більш вигідним з точки зору фінансів. Розподілення обчислення не є чимось новим чи ексклюзивним для великих бізнесів та наукових досліджень. Наприклад, найбільшим представником цієї технології є добре відома усім Всесвітня мережа Інтернет.

Хмарні обчислення є логічним продовженням розподілених обчислень. Замість того, щоб налаштовувати власну інфраструктуру для кластеру, організовувати його технічну підтримку, тощо, компанії можуть орендувати комп’ютерні ресурси в Інтернеті за необхідністю.

1.3.2 MapReduce

Наступна технологія, без якої аналіз Big Data був би майже неможливим, це MapReduce – найпопулярніша модель для розподілених обчислень. MapReduce оригінально складалась з 5 етапів, які потім було спрощено до трьох етапної моделі. Етапи є наступними:

- Splitting – розділяє вхідні дані рівномірно між усіма воркерами
- Mapping – кожен воркер використовує надану користувачем функцію до своїх даних
- Shuffling – воркери обмінюються обробленими даними так, щоб в кожному воркері містились усі дані з певним ключем, отриманим при обробці
- Reducing – кожен воркер оброблює надані йому дані, об’єднуючи їх
- Aggregating – результати обробки кожного воркера збираються та об’єднуються в один фінальний результат

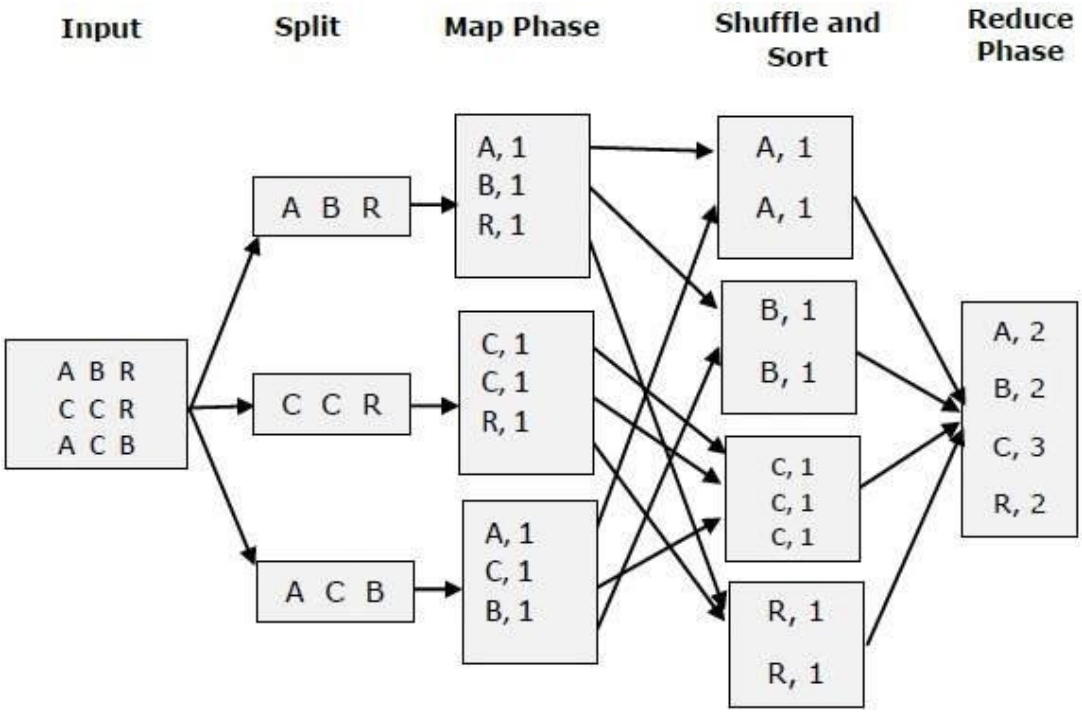


Рисунок 2, Приклад роботи MapReduce

Кожен воркер у фреймворку MapReduce є незалежним від інших, і усі вони можуть працювати паралельно (за умови що робота одного воркера не залежить від результатів роботи інших). Це дозволяє дуже просто підвищити кількість воркерів, що використовуються для обчислень, тим самим пришвидшуючи обробку.

Для усіх серйозних проектів Big Data дуже важливо мати хоча б якусь кількість обчислювальних воркерів, інакше навіть найпростіші обчислення можуть займати кілька годин, а більш складні можуть тривати багато тижнів. Але організація такої екосистеми зазвичай задача дуже складна, з великою кількістю можливих проблем. Тому стандартом індустрії є готова імплементація розподілених обчислень Hadoop.

1.3.3 Massively parallel processing (MPP)

MPP – ще одна модель розподілених обчислень. До популяризації Hadoop усі бізнес рішення виконувались саме на MPP. Крім того і деякі елементи екосистеми Hadoop все ще використовують цю модель через її набагато більшу швидкість в порівнянні з MapReduce (з іншої сторони стабільність та можливості для розвитку у MPP набагато нижчі), як наприклад Cloudera Impala.

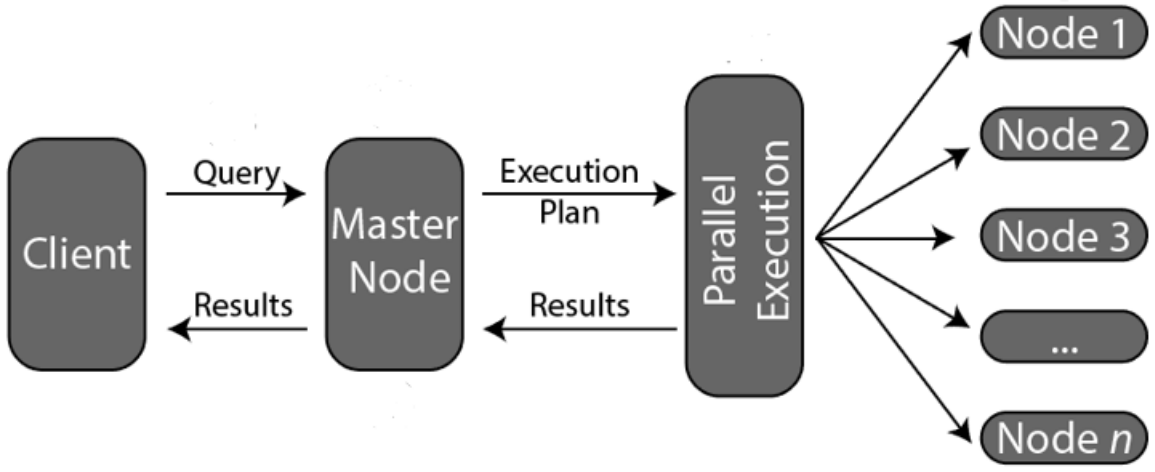


Рисунок 3, модель MPP

В MPP запит користувача оброблюється майстер нодою, яка розроблює план виконання, і передає його на кожну з нод. Після закінчення роботи результат повертається користувачу.

1.4 Big Data в Hadoop

На сьогоднішній день складно уявити Big Data проект що не використовує платформу Hadoop. Оригінально Hadoop був розроблений інженером «Yahoo!» Доугому Каттінгом, для задоволення потреб компанії у методах оброки великої кількості даних. Зараз Hadoop є open-source проектом компанії Apache, і має багато імплементацій від різних груп та компаній. Проект складається з двох великих частин – з файлової системи HDFS (Hadoop Distributed File System) та з «двигуна» (у класичному представлені (Hadoop версія 1) це був MapReduce, але на даний момент він вважається морально застарілим в порівнянні з альтернативами). Починаючи з Hadoop 2 до стандартного комплекту входить також й YARN

1.4.1 HDFS

HDFS – файлова система, що була заснована на принципах GFS (Google File System) і не дуже суттєво відрізняється від неї. Велика різниця між класичними файловими системами та HDFS з GFS, це те що останні не гарантують успішне виконання операцій. Навпаки, існує очікувана кількість помилок, які виправляються використанням високої надмірності системи. Крім того можлива кількість об’єктів в GFS набагато більша і сягає кількох мільярдів. І остання відмінність – класичні файлові системи зазвичай змінюють файли шляхом повного перезапису усіх даних, HDFS та GFS натомість додають інформацію напряму.

Архітектура GFS складається з трьох компонентів: одного майстер-серверу (або nameNode в HDFS термінології), багатьох чанк серверів (або dataNode-ів в HDFS), та багатьох клієнтів.

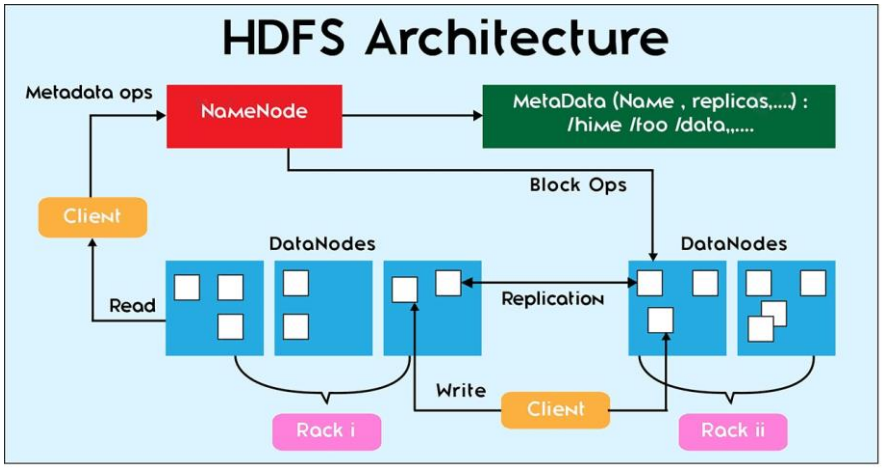


Рисунок 4. Архітектура Hadoop Distributed File System

Майстер сервер зберігає метадані, необхідну для доступу до чанк серверів, на яких зберігаються дані у блоках, зазвичай розміром 64 або 128 мегабайт. Крім того в метаданих зберігається лог системних процесів, таких як прибирання зайвих «сміттєвих» чанків, міграція чанків на інші сервера, тощо, а також записи «heartbeat» повідомлень, якими майстер підтверджує правильну роботу кожного чанка. Усі дані чанків дублюються для підтримання необхідної надмірності (гарною ідеєю є розміщення копій даних на фізичних пристроях у іншій географічній локації). Клієнти – це користувачі чи програми, які контактують з системою через інтерфейс API.

GFS було створено з п’ятьма постулатами[9]:

- GFS завжди очікує несправність апаратного забезпечення – кожний окремий сервер може перестати функціонувати в будь-який час
- GFS добре підтримує збереження кількох мільйонів великих файлів, з приблизним очікуванням розміру в 100 мегабайт на файл. GFS підтримує і менші файли, але не оптимізується під них
- Типовий розмір читання потоку сягає від кількох сотень кілобайт до 1 мегабайту

- GFS добре підтримує одночасну роботу кількох клієнтів з мінімальним синхронізаційним навантаженням
- Постійна пропускна здатність мережі для зберігання великих файлів важливіша за низьку затримку

Перед тим як розглядати різновиди «двигунів» для Hadoop, доцільно почати з розгляду різних дистрибутивів проекту, адже часто двигуни прив'язані саме до них.

1.4.2 YARN

YARN – Yet Another Resource Negotiator, сервіс для керування ресурсами в кластері. Починаючи з Hadoop 2 є основною частиною проекту, разом з HDFS.

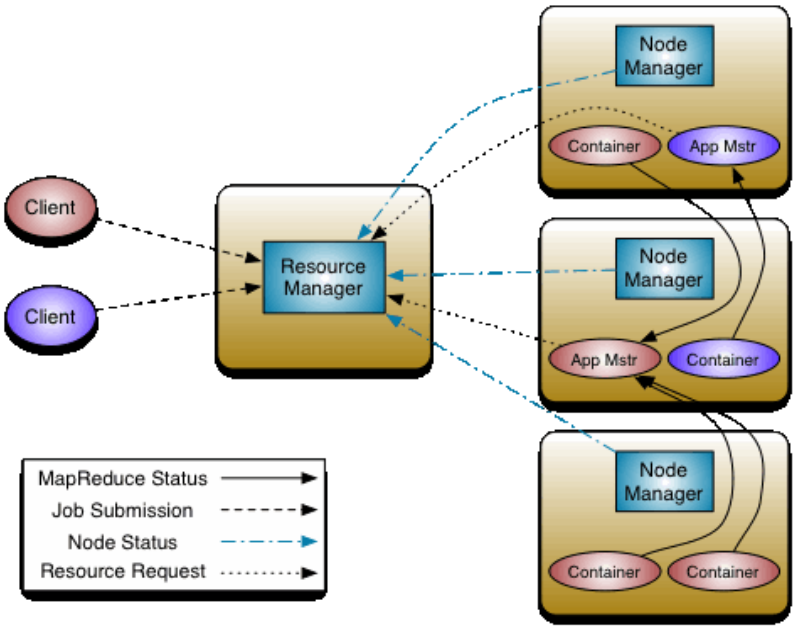


Рисунок 5, архітектура YARN

Центральна частина YARN – менеджер ресурсів (Resource Manager). Він відповідає за розподіл ресурсів між машинами в кластері, крім того поділяється на дві складові, Планувальник (YarnScheduler), та менеджер застосунків (Application Manager). Крім того на кожній ноді є свій власний менеджер (Node Manager), відповідальний за запуск нових контейнерів, а також Application Master, який

спілкується з менеджером ресурсів, слідкує за правильним виконанням процесу в контейнері, а по завершенню процесу повідомляє про це менеджера ресурсів та дерегеструє себе в ньому.

YarnScheduler відповідальний за відслідковування статусів завдань які зараз виконуються, за запуск нових завдань, та за розбиття ресурсів між ними. Працює в одному з 3-ох режимів: FIFO scheduling, capacity scheduling або fair scheduling.

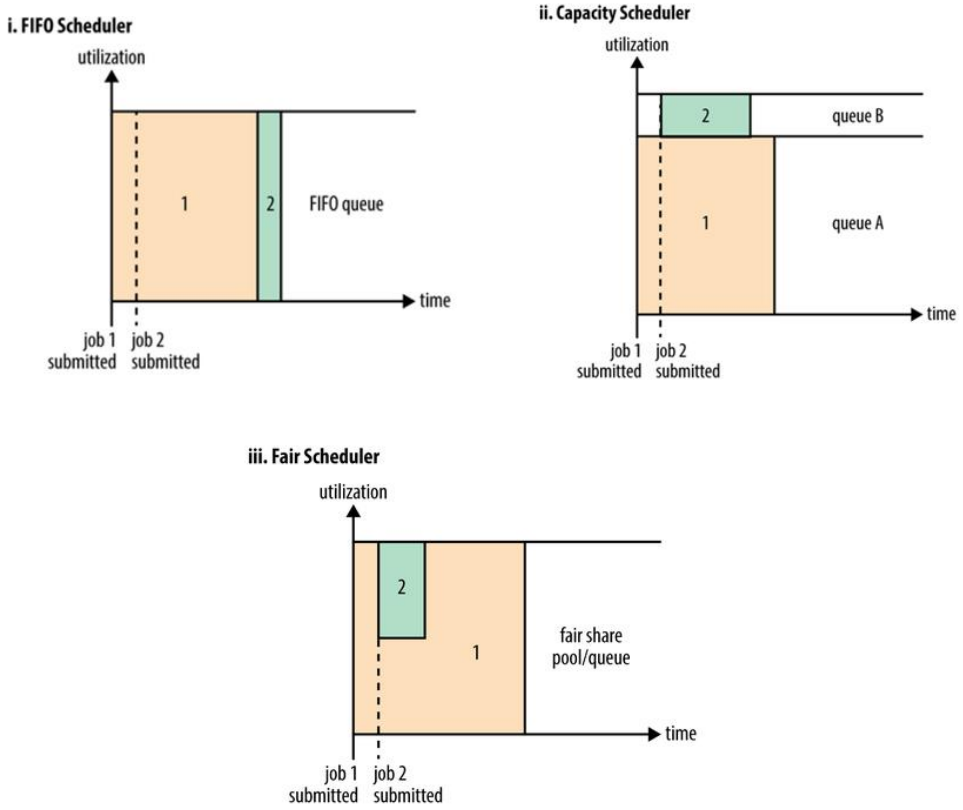


Рисунок 6, різні варіанти функціонування YarnScheduler-а

- FIFO scheduling – працює по структурі черги для завдань. В кожен момент праці кластер працює лише над одним завданням.
- Capacity scheduling – використовується якщо є кілька клієнтів що використовують кластер. Завдання розташовуються у кілька черг, так щоб сумарно усі черги використовували усі ресурси кластеру. Адміністратору необхідно налаштувати правильне розбиття на черги щоб коректно відобразити необхідність у ресурсах.

- Fair scheduling – спроба розділити усі ресурси між завданнями так, щоб кожне використовувало однакову потужність кластера у середньому. Коли одне з завдань закінчує виконання, YARN розподіляє ресурси між іншими. Стандартно це працює лише з вимогами до пам’яті, але може бути налаштовано працювати і з обчислювальною потужністю процесора або з використанням дискового простору.

Application Manager відповідає з утримання списку створених користувачами застосувань. При створені цих застосунків Application Manager має перевірити чи є Application Master з необхідними ресурсами, і в разі успіху передає завдання планувальнику для додання в чергу.

1.4.3 Дистрибутиви Hadoop

Найпопулярніші дистрибутиви Hadoop є наступними: Cloudera CDH Hadoop Distribution, Hortonworks Data Platform (HDP), MapR Hadoop Distribution, Amazon Elastic MapReduce. Крім них вартим уваги є ще офіційний дистрибутив Apache Hadoop, який є основним проектом від якого усі інші беруть свій початок. Не зважаючи на те, що Apache Hadoop існує на ринку довше конкурентів, він вже довго не користується значною популярністю серед експертів у сфері. Часто про нього кажуть що він дуже складний та заплутаний. Навіть простий запуск цього дистрибутива потребує великої кількості змінених файлів конфігурації та правильно налаштованих змінних середовища. Крім того велика кількість різних пакетів потребує певного часу для опанування.

Cloudera Distribution including Apache Hadoop (CDH) – дистрибутив компанії Cloudera, яка спеціалізується саме на цьому продукті, є найпопулярнішим на даний момент. Відомий своєю високою швидкістю в доданні нових авангардних функцій, навіть якщо функції ще не повністю стабільні. Мають кілька додаткових продуктів як альтернативи таким самим продуктам Apache. Наприклад Cloudera

Manager – менеджер, який відповідає за розгортання та за контроль системи – має дуже гарні відгуки серед як новачків так і експертів, надаючи простий та зрозумілий інтерфейс для контролю над усіма частинами екосистеми.

Hortonworks Data Platform – трохи менш популярний за попередній, але все одно дуже гарний дистрибутив. На відміну від Cloudera команда концентрує свої зусилля на поліпшені продуктів Apache, заміст того щоб розробляти свої з нуля (наприклад замість згаданого раніше Cloudera Manager використовують Apache Ambari).

MapR Hadoop Distribution – популярний через свою високу оптимізацію, по чистій потужності часто займає перше місце серед інших дистрибутивів. Так наприклад швидкість запису в нього в кілька разів швидше за Cloudera і Hortonworks. Крім того MapR також є й дуже надійним. Найбільшою проблемою MapR є цінова політика – безкоштовна версія програми має набагато менше функціоналу за повну.

1.4.4 «Двигуни» Hadoop

Довгий час після релізу Hadoop єдиним фреймворком на якому він працював був Apache MapReduce. Проте у нього був один великий недолік – необхідність запису усіх тимчасових результатів на диск. Це дуже сповільняло процес оброки Big Data, не задовольняло одну з V – Velocity, або швидкість. Для

hadoop MapReduce	TEZ	Spark	Apache Flink
• Batch	• Batch • Interactive	• Batch • Interactive • Near-Real time	• Batch • Interactive • Real-Time • Iterative
• 1st Generation	• 2nd Generation	• 3rd Generation	• 4th Generation

Рисунок 7, покоління двигунів Hadoop

багатьох ситуацій таке сповільнення є недопустимим, тому з’явилося багато альтернативних «двигунів». На даний момент існує 4 покоління, 1G – 4G.

Apache Spark – найпопулярніший фреймворк на момент 2021 року, відноситься до 3G. Написано його на Scala, випущено як top-level проект в 2014 році. Підтримка in-memory обробки дозволила Spark працювати в 100 разів швидше у RAM, та в 10 разів швидше у пам’яті ніж MapReduce. Це у свою чергу дозволило проводити обробку в майже реальному часі – важливий функціонал для багатьох бізнесів. Крім цього Spark за роки свого існування обріс великою кількістю додаткового функціоналу. Наприклад в ядрі фреймворку є вже готова бібліотека для машинного навчання MLlib, оптимізована під розподілені обчислення, або бібліотека GraphX для швидкої підтримки графів. Spark також має модифікований під нього REPL (Read evaluate print loop), модифікований Scala REPL, що дозволяє запускати код інтерактивно – дуже важлива функція для дебагінгу. Писати код для фреймворку можна на трьох мовах – Scala, Java та Python

Tez - йде в комплекті з екосистемою Hortonworks, дуже схожий за функціоналом до Spark, але, оскільки відноситься до другого покоління, на даний момент фактично застарілий. З найбільших проблем фреймворку – велика затримка, та висока складність коду. Код на Tez більше нагадує код асемблера (для порівняння, програма WordCount в офіційних прикладах займає усього 5 стрічок в Spark, і майже 250 в Tez). Одна з небагатьох ситуацій, коли Tez є кращим за Spark, якщо оброблюваний чанк інформації є більшим за об’єм доступної пам’яті, до того ж в два чи більше разів. У такій ситуації Spark буде робити overflow до диску, і в результаті ефективність фреймворку сильно падає. В Tez така ситуація оброблюється краще через інші методи збереження тимчасових результатів. Проте уникнути цього дуже просто шляхом зменшення оброблюваного чанку.

Apache Flink – відносно новий двигун, наступне покоління фреймворків Hadoop, 4G. Перша версія була випущена ще в 2011 році, але Flink все ще не отримав достатньо популярності щоб серйозно конкурувати з лідером Spark. Відповідно й спілка Flink набагато менш активна та розвинута, проекту все ще не вистачає зрілості. Попри те Flink має кілька значних переваг над конкурентами:

- Дуже низька затримка при обробці даних в реальному часі, усього кілька мілісекунд. Для порівняння затримка в Spark може сягати кількох секунд.
- Набагато краща підтримка роботи з потоками. В Spark потік ділиться на дуже малі частини і оброблюється як велика кількість мікро пакетів. В Flink існує підтримка як такого micro-batching підходу, так й повноцінного потокового (для цього в ньому є два різних API, DataSet API та DataStream API).

1.4.5 Інструменти роботи з SQL

Як і у випадку зі звичайними даними, Big Data зазвичай дуже зручно зберігати в табличному форматі. Для ефективної роботи з ними та для аналітики використовуються два продукти, часто одночасно – Apache Hive та Cloudera Impala.

Apache Hive – фреймворк для контролю даних на основі MapReduce. Перетворює запити написані на HiveQL, SQL-подібній мові запитів, в ланцюги команд для MapReduce, які потім передаються у вигляді списку команд до Hadoop чи Spark. Має високий потенціал до масштабованості, є відмово стійким. Функція LLAP (Live Long And Process) дозволяє кешувати запити у пам’яті. Починаючи з Hive 3+ присутні ефективні та повноцінні ACID операції. З іншої сторони, через використання MapReduce фреймворк є відносно повільним, до того ж не

підтримує обробку трансакційних даних і працює лише з пакетами, відповідно не здатен працювати з OLTP (Online Transaction Processing). Враховуючи це, зазвичай використовується для побудови потужних ETL (Extract-Transform-Load) систем в яких відсутня необхідність моментального результату і цінується надійність, можливість працювати з великим об’ємом інформації одночасно та значна кількість доступних форматів даних.

Cloudera Impala – як і Hive є фреймворком, націленим на роботу з даними, але у зовсім іншому ключі аніж перший. Основана на архітектурі Massive Parallel Processing, Impala працює дуже швидко, з мінімальними затримками. Весь фреймворк оптимізований під швидкі результати, і націлений саме на аналітиків Big Data.

1.4.6 Інструменти роботи з NoSQL

Інколи для певних ситуацій типові SQL рішення не підходять, або підходять, але гірше ніж NoSQL. В цих випадках використовують HBase.

HBase – база даних «ключ-значення», що працює поверх HDFS. На відміну від команд MapReduce в Hive та MPP в Impala, HBase зберігає дані у власну базу даних. До того ж робить це не при кожній операції запису, а тільки при досягненні певного об’єму – до того інформація тримається у пам’яті. Це надає HBase унікальну можливість працювати з окремими записами в реальному часі. Має дуже гарну швидкість обробки даних, підтримує роботу з транзакціями. Крім очевидного функціоналу як NoSQL база даних в екосистемі Hadoop, HBase має ще й високу цінність як середа для часто оновленої інформації. Зважаючи на розмір блока в 64 мегабайти, переписування інформації в Hive спричиняє дуже високе навантаження на ресурси, в HBase такої проблеми немає.

1.4.7 Інші корисні технології в екосистемі Hadoop

Крім вже обговорених існує ще багато надзвичайно корисних фреймворків та сервісі для різних аспектів BDA.

Apache Kafka – система, розроблена для обміну та для збереження великої кількості повідомлень з потоків інформації. Може функціонувати як зв'язок між клієнтами та системою, або пов’язувати окремі мікросервіси. На відміну від альтернатив, як наприклад RabbitMQ, зберігає повідомлення не в пам’яті, а на диску, що критично для Big Data застосунків. З плюсів: Kafka гарантує збереження усіх даних що були отримані, з мінусів: заплутана та неточна документація, що вимагає багато часу на вивчення. Фреймворк часто використовують в комбінації з надбудовою над Spark, Spark Streaming, який дозволяє працювати з потоками даних. Spark Streaming має в собі підтримку Kafka що дуже полегшує процес стрімінгу інформації.

Zookeeper – сервер для координації усіх частин Hadoop. Зазвичай використовується для конфігурації середи. Кілька сервісів залежні від Zookeeper-а та не можуть працювати без нього. Надійний сервіс, який використовується в багатьох серйозних проектах.

Ambari – менеджер для середовища Hadoop. Дозволяє створювати, налаштовувати кластери, стартувати та зупиняти кожний з сервісів. Також має аналітичну панель.

Amazon Web Services Elastic MapReduce Hadoop Distribution (AWS EMR)– платформа для швидкого та зручного розгортання хмарних кластерів Hadoop. Надає усі необхідні інструменти для роботи з Big Data, до того ж має гарну аналітичну підтримку на базі HDFS архітектури. Amazon називають «королем хмарних технологій», і не дарма - AWS покриває собою 32 відсотки хмарного ринку. Низька ціна та можливість швидкого надання додаткових ресурсів роблять цю платформу дуже привабливою для малих бізнесів та персональних проектів. До того ж Amazon має власний сервіс для збереження даних у хмарі S3, який входить до екосистеми EMR, що прибирає необхідність зовнішнього рішення.

РОЗДІЛ 2: МАШИННЕ НАВЧАННЯ

2.1 Визначення машинного навчання

Існує два популярних визначення машинного навчання. Перше запропонував Артур Семюель, піонер у сфері штучного інтелекту, у своїй роботі[10] 1959 року (він же й вигадав термін). Його визначення було наступним: «Машинне навчання – це сфера науки, що надає комп'ютеру можливість навчатись, не будучи напряму запрограмованими». Інше, більш сучасне визначення задав у 1997 році професор Університету Карнегі-Меллон та засновник першої в світі кафедри машинного навчання, Том Мітчелл: «Комп'ютерна програма навчається з певного досвіду Д завданням З і оцінкою ефективності О, якщо її ефективність в виконанні завдань З, яка оцінена оцінкою О, покращується з досвідом Д»

2.1.1 Supervised і unsupervised learning

Усі алгоритми машинного навчання поділяються на 2 категорії – supervised learning (навчання з учителем) та unsupervised learning (навчання без вчителя).

Supervised learning має певний стартовий датасет та вже отримані результати обробки цього датасету. Маючи ці два параметри алгоритм має розробити інструменти для отримання результатів з даних які ще таких не мають. Наприклад, маючи інформацію про продані автомобілі (марка, модель, колір, тощо), і відповідну вартість продажу, можливо розрахувати приблизну вартість будь якого іншого авто.

Supervised learning також ділиться на два підкласи – алгоритми регресії та алгоритми класифікації. Відповідь регресії – певне число, як наприклад у випадку з вартістю автомобіля. Класифікація ж ставить собі у ціль віднести елемент до одного з класів.

2.2 Особливості функціювання алгоритмів машинного навчання

Алгоритми машинного навчання багато у відрізняються, але в них є кілька важливих незмінних елементів, кожен з яких варто розглянути.

2.2.1 Ціна та гіпотеза

В більшості моделей машинного навчання є певна функція ціни. Чим менша ціна окремого екземпляру з датасету, тим ближче результат, отриманий алгоритмом до правильного. При ціні 0 відповідь є ідентичною очікуваній. Загальна ціна функції дорівнює сумі цін усіх екземплярів з датасету.

Гіпотеза – це функція, яку намагається вивести алгоритм. Саме її результат порівнюється з очікуваним при оцінці ціни.

Зазвичай функція ціни виглядає наступним чином:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

де h_0 – гіпотеза цієї моделі, m – кількість елементів в датасеті.

Або якщо модель потребує регуляризації:

$$min_\theta \ \frac{1}{2m} \ \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \ \sum_{j=1}^{n} \theta_j^2$$

(часто замість просто лямбда використовують лямбда ділене на 2m)

2.2.2 Векторизація

Алгоритми машинного навчання, особливо у контексті Big Data, мають мати дуже високий рівень оптимізації. Навіть обробка невеликого датасету на кілька тисяч елементів з менше 100 змінних кожен може зайняти суттєвий час. Крім того машинне навчання спирається на лінійну алгебру і часто вимагає певних маніпуляцій з елементами матриці. Для таких ситуацій часто варто

використовувати певні вбудовані в бібліотеки засоби, наприклад знайти суму квадратів усіх елементів вектору буде у багато разів швидше, якщо транспонувати вектор а потім помножити його на оригінальний, аніж якщо проходити через увесь вектор в циклі.

```
[[1.0000000e+00 2.0000000e+00 3.0000000e+00 ... 9.9999998e+07
  9.9999999e+07 1.0000000e+08]]
Результат векторизованого:  3.333333383333975e+23 , виконано за 0.08391310000000374 секунд
Результат без векторизації:  3.333333383334632e+23 , виконано за 59.357925 секунд

Process finished with exit code 0
```

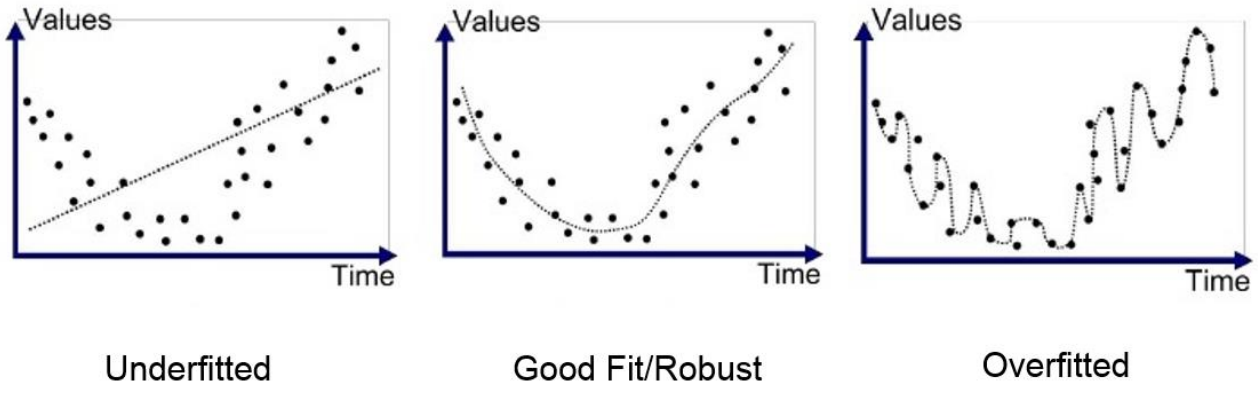
Рисунок 8. Порівняння швидкостей з векторизацією та без на векторі з 100 мільйонів елементів

Крім звичайних обчислень існують цілі алгоритми, які можна замінити діями лінійної алгебри

2.2.3 Регуляризація

Для алгоритмів регресії велика проблема – overfitting датасету. У цьому випадку алгоритм машинного навчання пристосовується до даних занадто добре, і в результаті працює гірше на реальних даних пізніше.

Для вирішення цієї проблеми до функції ціни додається сума усіх змінних в квадраті окрім першої, помножена на певний коефіцієнт лямбда. Таким чином алгоритм карається при використанні завеликих значень, через що результат приймає набагато більш натуральну форму, більш логічну для поставленої задачі.



Проте існує і зворотна сторона нормалізації. Якщо обрати лямбда занадто великою, отримана функція буде страждати від underfitting-у. Underfitting – це феномен, коли використання великих значень змінних в алгоритмі карається

настільки сильно, що найменшою вартістю є значення дуже далекі від оптимальних. В такому випадку функція має близький до лінійного вигляд, і дає погані результати на реальних даних.

Для вирішення проблеми обрання гарного значення лямбда зазвичай робиться наступне:

- Датасет ділиться на 2 (або інколи на 3) групи – тренувальний датасет та тестовий датасет (зазвичай тренувальний складає 60% від загального, а тестовий 40%)
- Обирається певний набір значень лямбда
- Тренуються моделі, основуючись на тренувальному датасеті для кожного значення лямбда
- Для кожної моделі вираховується її точність на тестовому датасеті. Найточніша модель обирається як основна

2.2.3 Feature scaling та Mean normalization

Одним з часто вживаних алгоритмів в машинному навчанні є градієнтний спуск. Суть алгоритму полягає в знаходженні найкращого напряму для зміни значень гіпотези і вираховується шляхом інтегрування функції для знаходження її напряму.

Для цього та кількох інших алгоритмів важливою умовою для ефективного функціонування є приблизно однакові значення усіх змінних датасету. Без цього

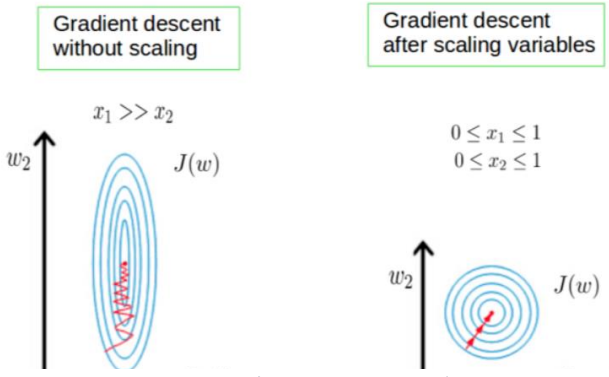


Рисунок 9, кроки алгоритму градієнтного спуску без та з нормалізацією

через специфічну роботу алгоритму обрахунки можуть зайняти набагато більше часу. Перерахунок значень змінних відбувається за допомогою двох підходів – feature scaling та mean normalization.

Feature scaling полягає у діленні кожної змінної на проміжок в якому вона знаходиться (на різницю максимального та мінімального значень), mean normalization до того ж віднімає від кожного значення середнє усіх інших.

Поєднана формула має вигляд:

$\frac{x-\mu}{xmax-xmin}$, де

xmax та xmin - максимальне та мінімальне значення змінної в датасеті,
μ - середнє значення змінної

2.3 Різновиди алгоритмів машинного навчання

Існує багато різних алгоритмів машинного навчання, але для оброки Big Data 4 найпопулярніших - лінійна регресія, логічна регресія, SVM (support vector machine) та K-means.

2.3.1 Лінійна регресія (Linear Regression)

Лінійна регресія – одна з найвідоміших моделей машинного навчання, яка маючи тестовий датасет може передбачити значення для ще не аналізованих параметрів. Задача лінійної регресії з математичної сторони – знайти такі значення θ для рівняння $\theta_1x_1 + \theta_2x_2 + \theta_3x_3 + \ldots$ (x – атрибути елементу датасету) щоб при підстановці нових значень x відповідь була близькою до очікуваної. Для цього треба мінімізувати значення функції вартості.

Ця модель використовує наступну формулу для своєї гіпотези:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \cdots + \theta_n x_n$$

До того ж використовується алгоритм градієнтного спуску, зміна значень обраховується наступним чином (Альфа – швидкість спуску. Завелике значення може призвести до того, що алгоритм не знайде мінімум, замале буде обраховуватись дуже довго):

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{for j} := 0...\text{n}$$

Одним з важливих методів для лінійної регресії є формування поліноміальних атрибутів, шляхом піднесення атрибутів у степінь, або множення кількох атрибутів. Це дозволяє формувати нелінійні функції, не змінюючи спосіб, у який працює модель.

2.3.2 Логічна регресія (Logistic regression)

Логічна регресія використовується для відношення нових даних до певної групи, основуючись на попередній інформації. Функціонує за схожим з лінійною регресією, з відмінністю лише у тому, що замість числа повертає 0 чи 1, відноситься до групи чи ні відповідно. Має видозмінену формулу для ціни:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

Не має методів регуляризації, оскільки вони не працюють на логічній моделі, але також використовує градієнтний спуск:

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

2.3.3 SVM

SVM, або Support Vector Machine – модель, схожа на логічну регресію. Зазвичай також використовується для класифікації елементів, але має в собі додаткову ціль – розділити елементи датасету такою лінією, щоб відстані між

граничними точками і нею були найбільшими.

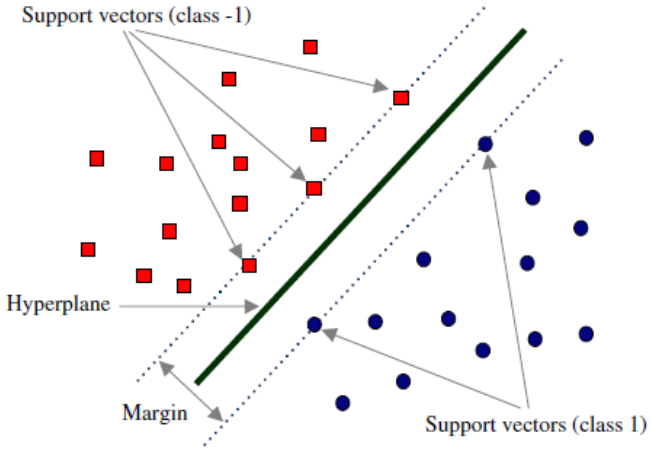


Рисунок 10, приклад SVM

Зазвичай провести таку лінію не є складним завданням, але часто елементи розділені не лінійно, а якимось іншим чином. В такому випадку датасет

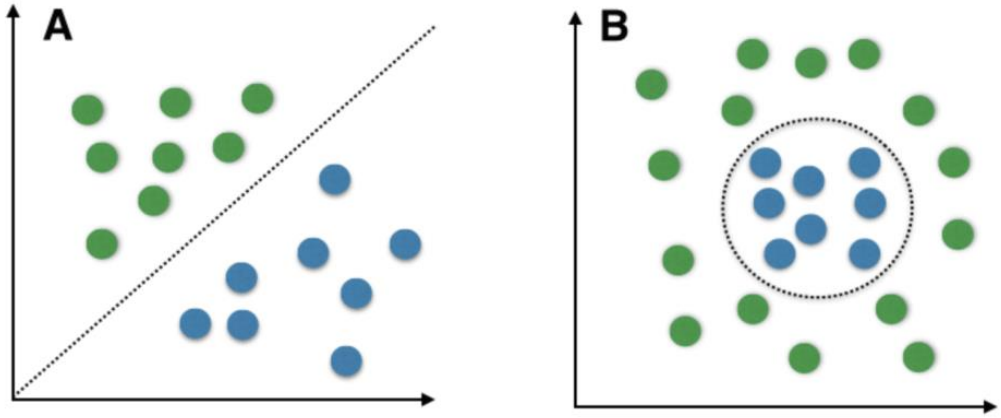


Рисунок 11, лінійно подільний та неподільний датасети

називають лінійно неподільним.

Для досягнення цього використовують ядра, або kernels, та функцію відстані точки до ядра (є багато способів обрахунку цієї відстані – лінійно, за Гаусом, поліноміально, тощо, тип зазвичай обирається експертом у області). Ядра обираються серед точок, усі інші точки обраховують сумарну відстань до обраних

ядр. Якщо відстань менша за 0.5, точка вважається такою, що не відноситься до шуканого класу.

2.3.4 K-means алгоритм

K-means є найпопулярнішою моделлю для навчання без вчителя. Простий і ефективний, k-means призначений для розділення датасету на будь яку кількість груп без їх чіткого визначення. Має багато застосувань, як наприклад поділ ринку, аналіз соціальних мереж, тощо.

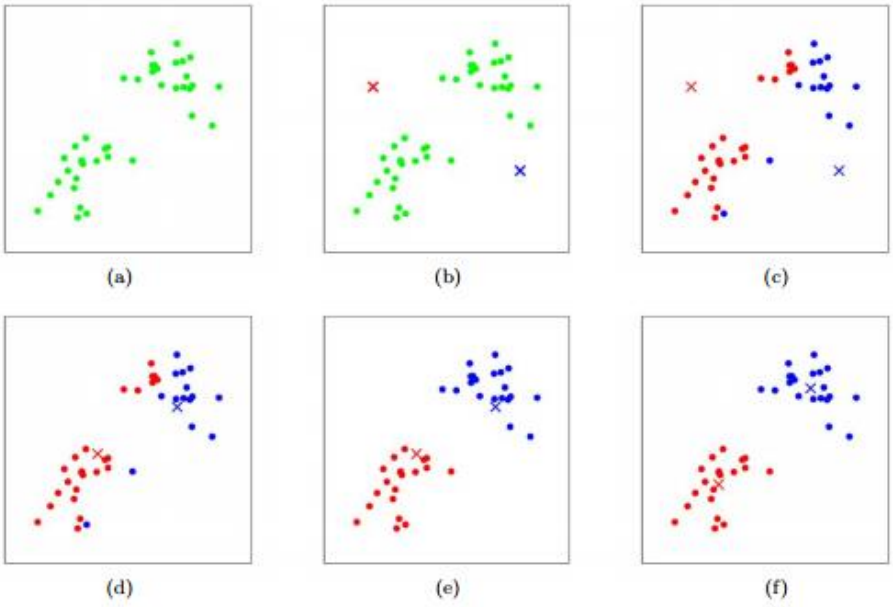


Рисунок 12, кроки k-means

Алгоритм k-means є наступним:

- Випадковим чином обрати випадкові точки з датасету, відповідно до кількості класів поділу, якірні точки. Якщо вибір невдалий то на цьому етапі є можливість що групування елементів пройде неправильно. Наприклад якщо дві точки дуже близько одна до одної. Для вирішення цієї проблеми алгоритм запускається багато разів, і для кожного результату обраховується його вартість. Фінальним результатом стає результат з найменшою ціною, який з найбільшою імовірністю відповідає реаліям.
- Для кожного елементу датасету обрахувати, яка з точок ближче до нього в просторі, й запам’ятати цю точку.

- Коли усі точки було оцінено, обраховується їх середнє значення, і отримані точки стають новими якорями. Процес повторюється доки якірні точки не залишаються такими ж що й на попередньому циклі, що свідчить про завершення.

Одна з проблем k-means це визначення найкращої кількості класів для поділу. В деяких випадках ця кількість зрозуміла з поставленої задачі, але не завжди. Для вирішення використовують «метод ліктя». Для кожної кількості класів обраховується k-means разом з ціною фінального результату. Побудувавши графік від ціни/кількості класів можна чітко побачити фігуру, віддалено схожу на лікоть

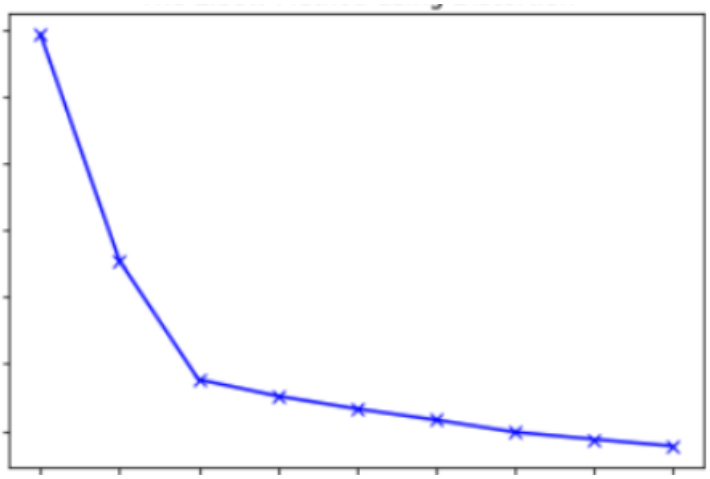


Рисунок 13, приклад фігури "лікоть"

Оскільки зі збільшенням числа класів ціна буде завжди падати, необхідно знайти момент, в якому збільшення числа класів буде давати тільки малу зміну в ціні. Ця точка знаходиться на місці згину «ліктя».

РОЗДІЛ 3: РЕАЛІЗАЦІЯ МАШИННОГО НАВЧАННЯ НА БАЗІ HADOOP

Для даного проекту найкращим вибором буде створити кластер на базі AWS Elastic Map Reduce, оскілки він поєднує у собі дистрибутив Hadoop (з native підтримкою Spark) та систему хмарних обчислень.

Першим кроком буде створення кластеру. Налаштування створеного кластеру:

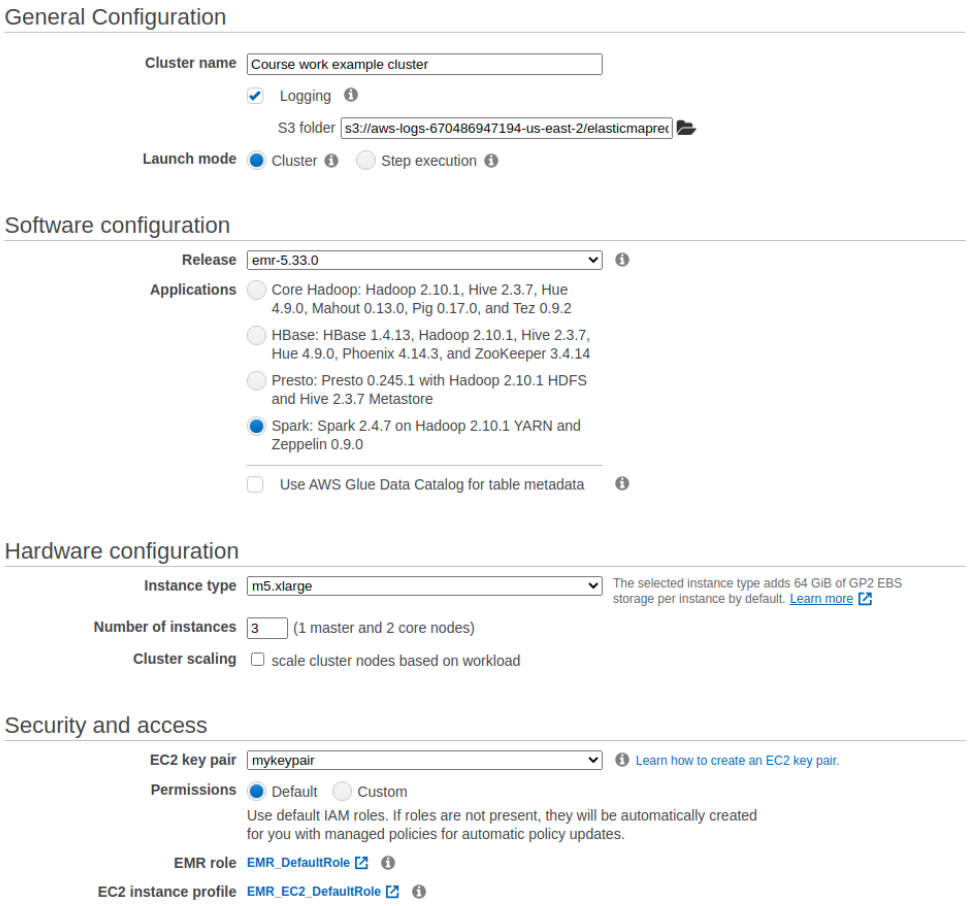


Рисунок 14, налаштування кластеру

Використовуємо найстабільнішу версію EMR, Spark. Використано усього лише 3 ноди, так як для більшої кількості Amazon вимагає переходу на інший план EC2. Також при створенні вказаний набір ключів який потім буде використано для SSH підключення.

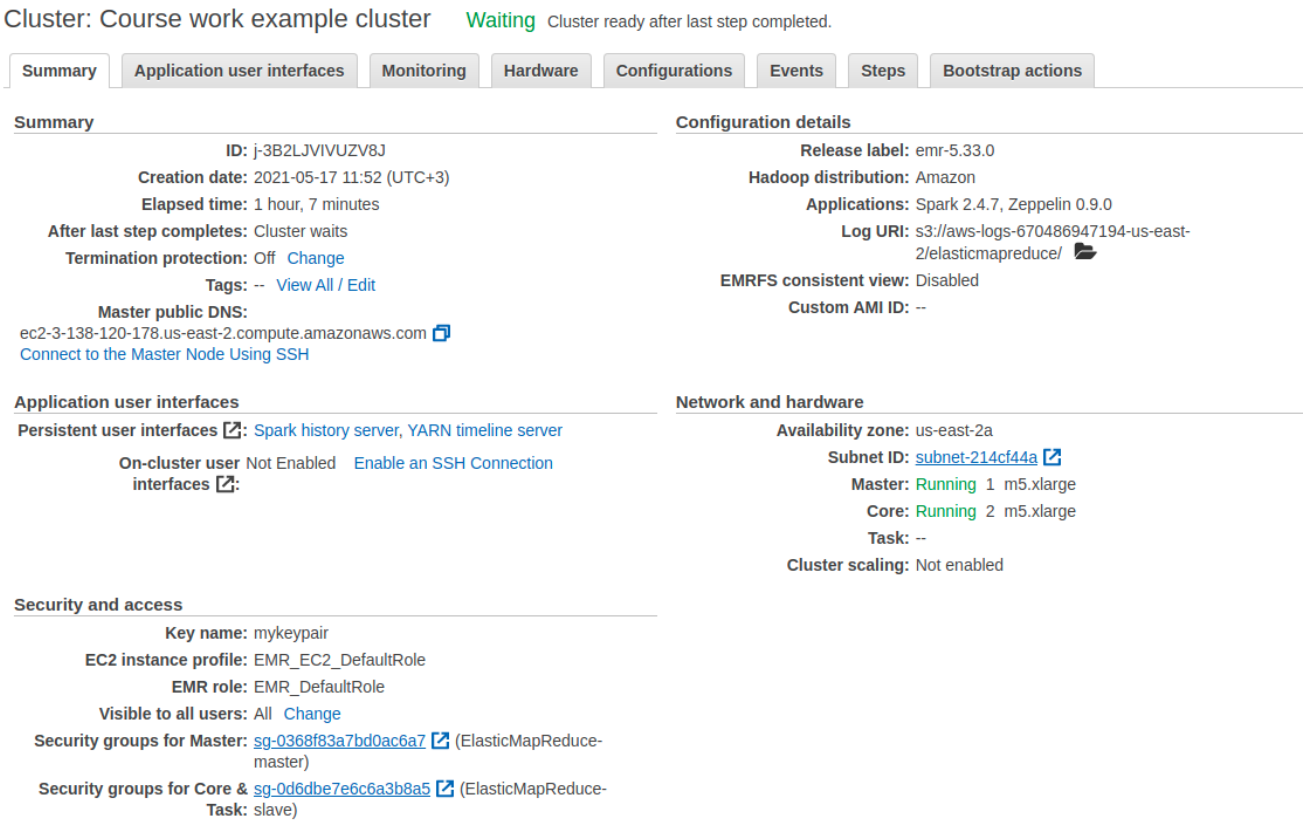


Рисунок 15, стан кластеру після ініціалізації

Наступний крок – підключення до кластеру через SSH. Для цього використано наступну команду (команда специфічна до цього кластеру)

ssh -i mykeypair.pem hadoop@ec2-3-138-120-178.us-east2.compute.amazonaws.com

Далі необхідно обрати датасет над яким буде проходити машинне навчання. Тут є кілька варіантів. Перший, найпростіший – знайти датасет, який вже має публічний інстанс в хмарному сховищі Amazon S3. Так наприклад Million Song Dataset з великою кількість метадати музикальних творів різних періодів має можливість бути скопійованим звідти. Оскільки EMR використовує S3 для збереження даних, такий підхід може бути дуже зручним.

Іншим варіантом (у випадку якщо датасет не має інстансу в S3) є завантаження датасету локально, перенесення його в систему.

Отримані дані необхідно перетворити на такі, що сприймає EMR, наприклад на Spark RDD:

Спершу визначається конфігурація spark:

```
from pyspark import SparkContext, SparkConf

conf = SparkConf().setAppName("Coursework test").setMaster('local[*]')

conf.set('spark.executor.memory', executor_memory)\
    .set('spark.driver.memory', driver_memory)\
    .set('spark.driver.maxResultSize', max_result_size)\
    .set('spark.yarn.executor.memoryOverhead', memory_overhead)

spark_context = SparkContext(conf=conf)
```

В цьому випадку використовувались наступні параметри: executor_memory = '5G', driver_memory = '5G', max_result_size = '5G', memory_overhead = '10G'

Використовуючи можливості паралелізації отримуємо RDD тренувальних даних:

```
rdd_train_data = spark_context.parallelize(list(train_data))
```

Перетворюємо RDD на Dataframe та ділимо на тренувальний та тестовий датасет:

```
df_train_full = rdd_train_data.toDF()
df_train, df_test = df_train_full.randomSplit([0.6, 0.4])
```

Запускаємо логічну регресію (як приклад) над тренувальним датасетом:

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol='features', labelCol='decade', regParam=0.01, maxIter=1000, fitIntercept=True)
lr_model = lr.fit(df_train)
```

Після цього програму можна запускати і отримувати результат машинного навчання над необхідним датасетом. У схожий спосіб можна виконати й інші алгоритми, особливо ті що підтримуються за стандартом в Spark.

Висновок

Під час роботи було проведено дослідження щодо роботи екосистеми обробки Big Data, зокрема проекту Hadoop, його файлової системи, різних дистрибуцій Hadoop та їх переваг та недоліків, рушіїв та інших частин екосистеми.

Також було досліджено поняття та методи методи машинного навчання, їх математичне представлення та способи використання.

В останній частині роботи було наведено приклад роботи у сфері Big Data Аналізу з використанням хмарного кластеру EMR та методів машинного навчання MapReduce.

Список джерел

1. de Heinzelin de Braucourt J., *Exploration du Parc National Albert: Les fouilles d'Ishango*, Fascicule 2, Institut des Parcs Nationaux du Congo Belge, Brussels, 1957
2. Graunt, John. *Natural and political observations made upon the bills of mortality*. No. 2. Johns Hopkins Press, 1939.
3. Robert McMillan, *How Social Security Rescued IBM From Death by Depression*, Wired, 2012
4. Copeland B. J. Colossus: *The secrets of Bletchley Park's code-breaking computers*. – OUP Oxford, 2010.
5. Kolodzey J. *Cray-1 computer technology* IEEE Transactions on Components, Hybrids, and Manufacturing Technology, 1981
6. Rydning D. R. J. G. J. *The digitization of the world from edge to core*, Framingham: International Data Corporation, 2018.
7. Laney, Doug. *3D data management: Controlling data volume, velocity and variety.* META group research note 6.70, 2001.
8. Elloitt, Timo. *7 Definitions of Big Data You Should Know About*, 2013
9. Ghemawat S., Gobioff H., Leung S. T. *The Google file system. Proceedings of the nineteenth*, ACM symposium on Operating systems principles, 2003
10. Samuel, Arthur L. *"Some studies in machine learning using the game of checkers."* IBM Journal of research and development 3.3, 1959
11. Mitchell, Tom M. "Machine learning.", 1997
12. Buyya, Rajkumar, Rodrigo N. Calheiros, and Amir Vahid Dastjerdi, eds. *Big data: principles and paradigms*. Morgan Kaufmann, 2016.
13. Dr Mark van Rijmenam. *A Short History Of Big Data*, Datafloq, 2013