**Контрольні питання**

**1. Поняття великих даних. Програмний засіб Hadoop**

Великі дані — набори інформації (як структурованої, так і неструктурованої) настільки великих розмірів, що традиційні способи та підходи, здебільшого засновані на рішеннях класу бізнесової аналітики та системах управління базами даних, не можуть бути застосовані до них. Альтернативне визначення називає великими даними феноменальне прискорення нагромадження даних та їх ускладнення.

Apache Hadoop — вільна програмна платформа і каркас для організації розподіленого зберігання і обробки наборів великих даних з використанням моделі програмування MapReduce, при якій завдання ділиться на багато дрібніших відособлених фрагментів, кожен з яких може бути запущений на окремому вузлі кластера, що складається з серійних комп'ютерів.

**2. Методи і техніка анализу великих даних**

Методи классу Data Mining (видобуток даних, інтелектуальний аналіз даних, глибинний аналіз данних) — сукупність методів виявлення у даних раніше невідомих, нетривіальних, практично корисних знань, необхідних для прийняття рішень. До таких методів, зокрема, належать: навчання асоціативним правилам (association rule learning), класифікація (разгалуження на категорії), кластерний аналіз, регресійний аналіз, виявлення і аналіз відхилень тощо.

Краудсорсинг — класифікація і збагачення даних силами широкого, неозначеного кола особистостей, що виконують цю роботу без вступу у трудові стосунки.

Змішання та інтеграція даних (data fusion and integration) — набір технік, що дозволяють інтегрувати різнорідні дані з розмаїття джерел з метою проведення глибинного аналізу (наприклад, цифрова обробка сигналів, обробка природньої мови, включно з тональним аналізом).

Машинне навчання, включаючи навчання з учителем і без учителя – використання моделей, побудованих на базі статистичного аналізучи машинного навчання для отримання комплексних прогнозів на основі базових моделей.

Штучні нейронні мережі, мережевий аналіз, оптимізація, у тому числі генетичні алгоритми (genetic algorithm — евристичні алгоритми пошуку, що використовуються для розв’язання задач оптимізації і моделювання шляхом випадкового підбору, комбінування і варіації потрібних параметрів з використанням механізмів, аналогічних натуральному відбору у природі)

Розпізнавання образів

Прогнозна аналітика

Імітаційне моделювання (simulation) — метод, що дозволяє будувати моделі, що описують процеси так, як вони би проходили у дійсності. Імітаційне моделювання модна розглядати як різновид експериментальних випробувань.

Просторовий аналіз (spatial analysis) — клас методів, що використовують топологічну, геометричну і географічну інформацію, що вилучається із даних.

Статистичний аналіз — аналіз часових рядів, A/B-тестування (A/B testing, split testing — метод маркетингового дослідження; при його використанні контрольна група елементів порівнюється із набором тестових груп, у яких один чи кілька показників були змінені, щоб з’ясувати, які зі змін покращують цільовий показник.

Візуалізація аналітичних даних — подання інформації у вигляді малюнків, діаграм, з використанням інтерактивних можливостей і анімації, як для отримання результатів, так і для використання у якості вихідних даних для подальшого аналізу. Дуже важливий етап аналізу великих даних, що дозволяє показати найважливіші результати аналізу у найбільш зручному для сприйняття вигляді

**3. Дайте визначення MapReduce**

MapReduce - це фреймворк, розроблений Google, який дозволяє використовувати велику кількість недорогих комп'ютерів для проведення розподіленої паралельної обробки великої кількості задач. В MapReduce комп'ютери називають “нодами”. Вони діляться на “робочі ноди”, та на “мастер ноду” (worker node, master node відповідно).

Метод MapReduce складається з двох кроків. Map і Reduce. На етапі Map відбувається попередня обробка даних головною нодою, а саме, обробка вхідних пар ключ/значення та генерація проміжних пар ключ/значення.

Після цього, на етапі Reduce головний вузол отримує результати згортання попередньо оброблених даних від робочих вузлів, та на їх основі формує результат. Тобто виконується зведення докупи всіх проміжних значень, пов’язаних із проміжним ключем.

Плюсом цього методу є те що він дозволяє проводити ці операції розподілено й паралельно, якість обчислень залежить лише від вхідних даних.

**4. Дайте визначення Apache Spark**

Apache Spark — високопродуктивна система для оброблення великих об’ємів даних, що зберігаються в кластері Hadoop. Spark може використовуватися як у типових сценаріях оброблення даних, схожих на MapReduce, так і для реалізації специфічних методів, таких як потокове оброблення, SQL, інтерактивні та аналітичні запити, рішення задач машинного навчання і робота з графами.

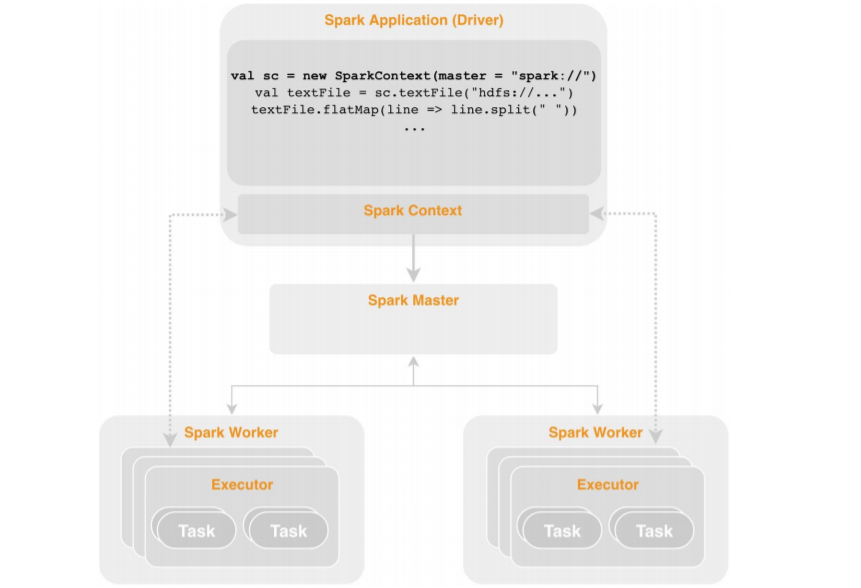
**5. Архітектура розподіленого додатку Spark. Ком’ютерний кластер. Вузол**

Комп’ютерний кластер — це набір взаємопов’язаних вузлів, які працюють разом і з деякими обмеженнями можуть розглядатись як єдина система.

Вузол — це окремий комп’ютер, який має певні обчислювальні ресурси і ресурси для збереження даних.

Окремий вузол може ефективно обробити тільки частину даних, саме тому, для “горизонтального” масштабування завдань, вузли об’єднуються в кластери.

Для ефективного використання кластера необхідно забезпечити його представлення як єдиної системи, щоб робота з кластером нагадувала роботу з одним комп’ютером. Для такого наближення і використовуються засіб Apache Spark.



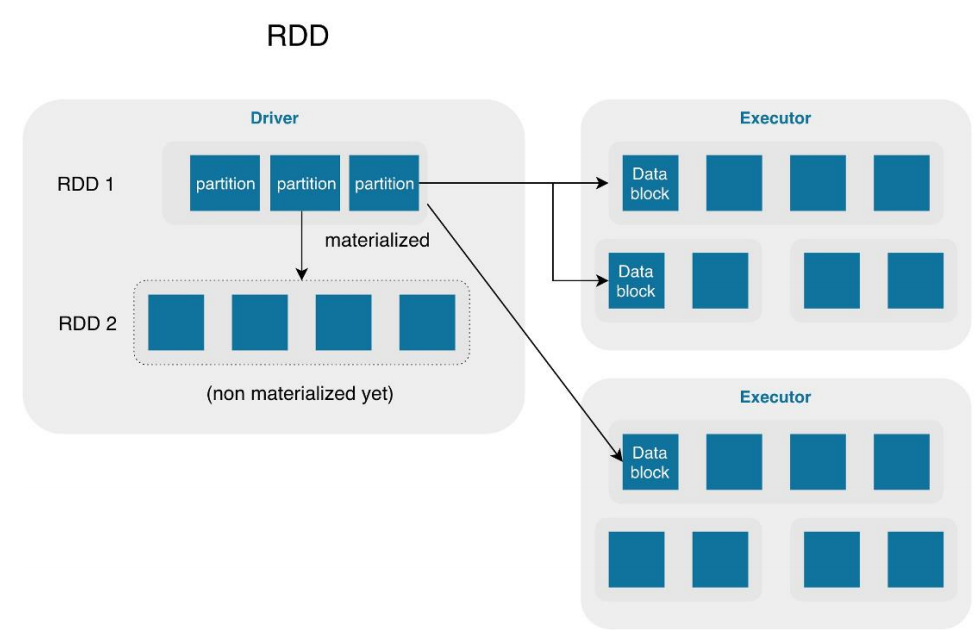
Загальна архітектура застосунку Spark

Основні компоненти архітектури — це Driver та Executor. Driver — це компонент, що відповідає за оркестровку, контроль за обчислювальними ресурсами, відстеження та управління станом обчислень. Executor відповідає за виконання обчислень над даними на ресурсах обчислювального вузла, де він знаходиться.

**6. Основні концепції Spark. RDD та граф перетворень**

Для представлення даних Spark використовує концепцію RDD (Resilient Distributed Datasets). RDD - це абстракція набору даних, що складається із записів одного довільного типу і розділеного на частини, які розміщені по всьому кластеру.

RDD містить метаінформацію про набір даних, але не самі дані, і зберігається в driver додатка Spark під час його виконання і тільки під час його виконання. Безпосередньо дані, розбиті на блоки, зберігаються на executor'ах або в зовнішньому сховищі (якщо датасета не був ще завантажений для обробки, тобто так можуть бути представлені, умовно кажучи, вхідні дані програми Spark).



**7. Основні етапи обробки даних**

· Краудсорсинг (класифікація і збагачення даних силами широкого, неозначеного кола особистостей)

· Змішання та інтеграція даних

· Машинне навчання

· Розпізнавання образів

· Прогнозна аналітика

· Імітаційне моделювання (simulation)

· Просторовий аналіз

· Статистичний аналіз

· Візуалізація аналітичних даних

**8. Загрузка даних із зовнішнього сховища**

Как происходит загрузка данных, а также сколько партиций будет создано в процессе нее, определяется адаптером конкретного хранилища. В целом, в типичные задачи такого адаптера входит следующее: выполнение служебных запросов для определения количества партиций и получения всей необходимой метаинформации (например, схемы таблиц или идентификация схемы с помощью сэмплирования json-документов); определение местоположения партиций (что особенно актуально, если кластер хранилища и вычислительный кластер являются одним целым); подготовка и оптимизация запросов, специфичных для хранилища, для выборки данных из него, включая передачу необходимых параметров для фильтрации на стороне хранилища (если хранилище позволяет это); трансфер данных из источника и их запись во внутреннее представление, которое сможет использовать для вычислений Apache Spark; При чтении данных из хранилища сначала будет выполнена операция подсчета, что тоже может потребовать достаточно интенсивных вычислений. Особенное внимание тут следует уделять при работе с источниками, где схема данных не известна наперед: json-файлы в hdfs, данные в mongodb и т.п. Загрузка данных также является операцией вычислений над данными и происходит при материализации датасетов.

**9. Зміна розміщення даних та кількості партицій**

В процесі обчислень може виникнути потреба у зміні розміщення даних і / або кількості партіцій в датасета.

Наприклад, після завантаження даних з файлів в hdfs ми хочемо, щоб всі дані, що належать одному і тому ж користувачеві, потрапили на один вузол, так як всі подальші операції будуть відбуватися тільки в рамках одного користувача.

Щоб домогтися такого ефекту, ми можемо скористатися функцій repartition із зазначенням кількості партіцій або конкретного partitioner. В Spark за замовчуванням є два основних partitioner, що дозволяють досягти цієї цілі:

* HashPartitioner - розміщує запис в партіціі у відповідність з хешом від ключа цього запису (актуально для key-value RDD).
* RangePartitioner - розміщує запис в партіціі відповідно до діапазону, в який потрапляє ключ цього запису (актуально для keyvalue RDD). Такий partitioner може бути корисний, наприклад, в ситуації коли нам необхідно агрегувати дані з продажу за окремі тижні - тут RangePartitioner може допомогти з розміщенням всіх даних, що відносяться до конкретної тижня тільки в одній партіціі, тим самим приберемо необхідність в реальному мережевий передачі даних.

У разі якщо RDD не призначено partitioner, розподіл записів в партіціі відбувається рівномірно. Відзначимо, що в разі DataFrame API репартіцірованіе може бути застосоване до кількох колонок. Також існує можливість дописати свій власний partitioner, наприклад, для ситуації, коли нам має сенс розбивати дані і за ідентифікатором клієнта, і по інтервалу здійснення операції - таким чином всі дані одного користувача за один тиждень опиняться в одній партіціі. При цьому, якщо користувач має дуже багато активності, його обробку можна буде распараллелить (так як розбиття - це мінімальна одиниця послідовної обробки).

Слід зазначити, що розміщувати таким чином дані можна не тільки для одного датасета, а відразу для декількох, наприклад, мають один і той же первинний ключ. У разі якщо нам знадобиться проводити над ними операцію join (див. Приклад в лістингу 3.7), за рахунок однакових partitioner'ов і відповідно однаковому розташуванню партіцій з тими ж ключами на вузлах, вдасться уникнути мережевий передачі в операції shuffle (за умови однакової кількості партіцій в обох RDD).

**10. Як відбувається обчислення над даними в Spark**

Дані над якими працює Spark зазвичай групуються у спеціальну структуру даних - DataFrame. Вона містить таблицю яка складається з рядків та колонок.  
Всі операції в Spark можна умовно поділити на дві основі групи - трансформатори та дії:  
 Дії застосовуються тоді, коли необхідно матеріалізувати результат - як правило, зберегти дані на диск, або вивести частину даних в консоль). Ось приклад дій, які можна застосовувати над RDD): :  
 - saveAsTextFile (path) - зберігає дані в текстовий файл (в hdfs, на локальну машину або в будь-яку іншу підтримувану файлову систему - повний список можна подивитися в документації)

- collect () - повертає елементи датасета у вигляді масиву. Як правило, це застосовується у випадках, коли даних в датасета вже мало (застосовані різні фільтри і перетворення) - і необхідна візуалізація, або додатковий аналіз даних, наприклад засобами пакета Pandas.

Траносформації , результатом застосування яких завжди є новий RDD. Як правило, це операції, які будь-яким чином перетворюють елементи даного датасета. Ось приклади поширених перетворень, кожне з яких повертає новий датасета (RDD)

* map (function) - застосовує функцію function до кожного елементу датасета.
* filter (function) - повертає всі елементи датасета, на яких функція function повернула справжнє значення.
* distinct ([numTasks]) - повертає датасета, який містить унікальні елементи вихідного датасета

**11. Розгалуження та ітеративні обчислення**

Розгалуження - другий з типових різновидів алгоритмів. Розгалуження починається з певної умови. Якщо ця умова виконується, відбувається одна дія (чи набір дій); якщо ні - інша. Прикладом такого алгоритму може слугувати, наприклад, спілкування контролера з пасажиром автобуса: якщо пасажир пред'явив квиток, треба подякувати пасажиру за оплачений проїзд, якщо ні - оштрафувати. Метод або ж алгоритм ітерації — рекурсивний алґоритм, що реалізує в деякому топологічному просторі послідовність точково-множинних відображень, з допомогою яких, за початковою точкою вичисляють послідовність точок згідно формул. Ця операція називається ітерація, а послідовність називається ітераційна послідовність. Такі методи (що ще називаються методами послідовних наближень) можуть використовуватись для находження розв'язків операторного рівняння чи мінімуму деякого функціонала, чи власних значень і елементів рівняння, а також для доведення існування розв'язків цих задач.

**12. Shuffle механізм**

Процесс shuffling является одним из фундаментальных, т. к. позволяет изменять размещение данных на узлах, лежа в основе операций объединения (join) и группировки (group by, reduce) данных. Для задач машинного обучения shuffle является неотъемлемой частью, например, для алгоритма Expectation Maximization или word2vec. На вход shuffle поступает RDD c n партициями, а выходом будет RDD с m партициями. В силу исторических причин мы будем называть входной RDD map-стороной, а выходной – reduce-стороной. В Spark количество выходных партиций не зависит напрямую от количества уникальных ключей в конкретном датасете, а определяется параметром spark.sql.shuffle.partitions, который задается в настройках SparkContext. Каждая запись из партиций первого RDD будет оценена по некоторому ключу, и относительно него будет назначена партиция из выходного RDD, в которую эта запись попадет. Таким образом, все записи с одинаковыми ключами попадут в одну и ту же партицию выходного RDD. На рис 2.7 приведена иллюстрация данного процесса.

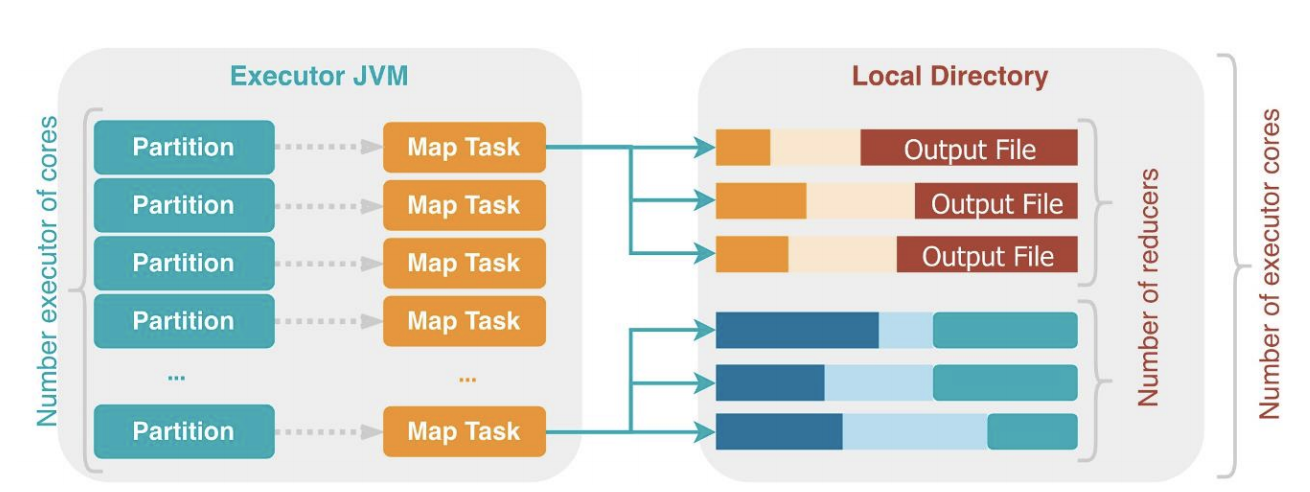


Рисунок 2.7 – Shuffle – процесс изменяющий размещение данных

Для реализации shuffle необходима подготовка данных на map-стороне – группировка входных записей по их ключам, чтобы их затем можно было отослать на узел с нужной партицией. Существует два основных способа, как это можно сделать. Первый способ (hash shuffle) подразумевает использование отдельного набора файлов для каждой map-задачи. Его оптимизированная версия, используемая в текущих 20 версиях Spark, подразумевает переиспользование набора файлов для каждого из слотов. Sort Shuffle используют другую логику обработки. Hash shuffle на выходе производит по одному отдельному файлу для каждой выходной партиции и, как следствие, для каждого из «редьюсеров». С помощью же sort shuffle возможно сократить их количество: выходные файлы упорядочены по id «редьюсера» и и содержат индекс в самом начале, состоящий из указателей позиций (offsets) на начало блока записей с конкретным id. Это позволяет легко получить блок данных, относящихся к «редьюсеру х», просто используя информацию о положении связанного блока данных в файле. Но, конечно, для небольшого количества «редьюсеров» очевидно, что хеширование отдельных файлов будет работать быстрее, чем сортировка, поэтому сортировка в случайном порядке имеет план «отката»: когда количество «редьюсеров» меньше, чем «spark.shuffle. sort.bypassMergeThreshold» (по умолчанию 200), используется hash shuffle

**13. Управління памяттю в Apache Spark**

Модель пам'яті Spark передбачає наявність 3 основних областей пам'яті:

Reserved Memory, User Memory, Spark Memory.

Зарезервована пам'ять (Reserved Memory). Це пам'ять,

зарезервована системою, і її розмір жорстко закодований. Починаючи з

версії Spark 1.6.0, його значення становить 300 МБ, що означає, що ці 300

МБ RAM не беруть участі в обчисленнях розміру області пам'яті Spark, і його

розмір не можна змінити будь-яким чином без перекомпіляції Spark або

установки spark.testing.reservedMemory, що не рекомендується, так як це

параметр тестування, що не призначений для використання під час

реальних обчислень.

Пам'ять користувача (User Memory). Це пул пам'яті, який

залишається після виділення Spark Memory, і може бути використаний для

зберігання структури даних користувача, які створюються в процесі

обчислень за допомогою трансформацій RDD і функцій, що задаються

визначених користувачем (user-defined function, UDF).

Пам'ять Spark (Spark Memory). Це пул пам'яті, керований самим

Spark.

Пам'ять зберігання (Storage memory). Цей пул використовується як для

зберігання кешованих даних Spark, так і для тимчасового розгортання

серіалізовать даних.

Пам'ять виконання (Execution Memory). Цей пул використовується для

зберігання об'єктів, необхідних під час виконання завдань Spark. наприклад,

він використовується для зберігання проміжного буфера на map етапі, а також

для зберігання хеш-таблиці для етапу агрегації по хешу. Цей пул також

підтримує розміщення даних на диску, якщо недостатньо пам'яті

**14. DataFrame API та Spark SQL. Датафрейми**

Spark SQL - це розширення Apache Spark для роботи зі структурованими даними. Spark SQL дозволяє використовувати переваги реляційної обробки даних. Одним з засобів роботи зі Spark SQL є DataFrame.

DataFrame - це основна абстракція для Spark SQL. Розподілена колекція даних, впорядкована за іменованими стовпцями, в Spark називається DataFrame.

DataFrame пов’язаний з поняттям DataSet. Dataset - це розподілений набір даних. Інтерфейс Dataset доданий в Spark 1.6 для забезпечення переваг RDD (таких як сувора типізація, можливість використовувати лямбда-функції) спільно з перевагами механізму виконання Spark SQL. Dataset може бути побудований з JVM-об'єктів і змінений за допомогою функціональних перетворень (map, flatMap, filter та т. Д.). Dataset API доступний в Scala і Java. В Python немає підтримки Dataset API, але через динамічного харакатера Python, багато можливостей Dataset API вже доступні (наприклад, звернення до полю в рядку по імені). Аналогічно в разі мови R.

DataFrame - це Dataset, представлений іменованими стовпцями. DataFrame - це аналог таблиці в реляційної базі даних або DataFrame'а в Python / R, але з поліпшеною оптимізацій для розподілених обчислень. DataFrame може бути створений з великої кількості джерел: csv-файлів, таблиць в Hive, зовнішніх баз даних або існуючих RDD. DataFrame доступний з Scala, Java, Python і R.

**15. Робота з DataFrame API в рамках побудови сценаріїв обробки даних на Spark**

map (function) - застосовує функцію function до кожного елементу датасета;

filter (function) - повертає всі елементи датасета, на яких функція function повернула справжнє значення;

distinct ([numTasks]) - повертає датасета, який містить унікальні елементи вихідного датасета.

Також варто відзначити про операції над множинами, зміст яких зрозумілий з назв:

union (otherDataset);

intersection (otherDataset);

cartesian (otherDataset) - новий датасета містить в собі всілякі пари (A, B), де перший елемент належить вихідного датасета, а другий - датасета-аргументу.

saveAsTextFile (path) - зберігає дані в текстовий файл (в hdfs, на локальну машину або в будь-яку іншу підтримувану файлову систему - повний список можна подивитися в документації);

collect () - повертає елементи датасета у вигляді масиву. Як правило, це застосовується у випадках, коли даних в датасета вже мало (застосовані різні фільтри і перетворення) - і необхідна візуалізація, або додатковий аналіз даних, наприклад засобами пакета Pandas;

take (n) - повертає у вигляді масиву перші n елементів датасета

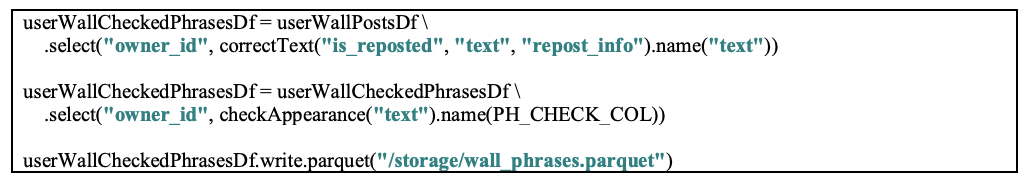
count () - повертає кількість елементів в датасета;

reduce (function) – функція агрегації даних.

**16. Використання функцій користувачів(UDF)**

Ниже, в листинге 3.7, приведен пример более сложной программы обработки данных, использующей еще 2 принципиальные возможности Spark: возможность пользователю писать собственные функции обработки (user – defined functions или UDF) в дополнение к тому, что есть в стандартной библиотеке Spark; использование broadcast переменных для работы с крупными объемами данных без замыкания их в UDF, что позволяет избежать расходов на дорогостоящие сериализацию и передачу данных, так как каждая задача обладает собственной копией используемой функции и всех ее данных.





Листинг 3.7 – Пример обработки данных с использованием UDF и broadcast переменных (приведено для PySpark, но код для Spark существенно не отличается).

**17. Функції користувачів агрегації в Spark**

[-IN, BUF, OUT]

Базовий клас для визначених користувачем агрегацій, який можна використовувати в Dataset операціях, щоб взяти всі елементи групи та звести їх до одного значення.

IN - Тип введення для агрегування.

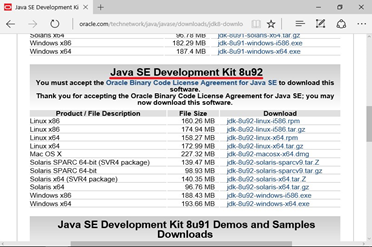
BUF - Тип проміжного значення зменшення.

OUT - Тип кінцевого результату.

* bufferEncoder: Encoder[BUF]  
  Визначає Encoder для типу проміжного значення.
* finish(reduction: BUF): OUT  
  Перетворює результат зменшення.
* merge(b1: BUF, b2: BUF): BUF  
  Об’єднує два проміжні значення.
* outputEncoder: Encoder[OUT]  
  Визначає Encoder для остаточного типу вихідного значення.
* reduce(b: BUF, a: IN): BUF  
  Агрегує вхідне значення а у поточне проміжне значення. Для підвищення продуктивності функція може змінити b і повернути його, замість того, щоб створювати новий об'єкт для b.
* zero: BUF  
  Початкове значення проміжного результату для цього агрегування.

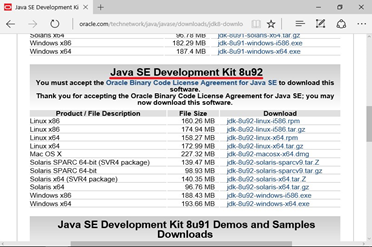
**18. Створення, налаштування та запуск Spark проекту. Налаштування оточення**

Для початку необхідно завантажити JDK 8 (перевірте останню підтримувану Spark версію) для вашої версії ОС: http://www.oracle.com/technetwork/java/javase/downloads/index.html.



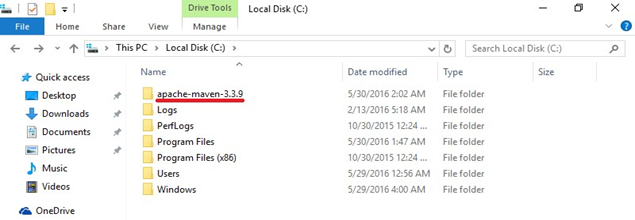
Java SE Development Kit.

Потім пройти інструкцій інсталятора і використовувати стандартні параметри. Необхідно також завантажити Apache Maven http://maven.apache.org/download.cgi:



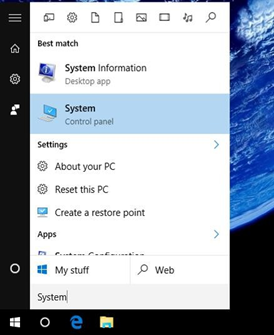
Завантаження Maven з головного сайту.

Вийміть файли з архіву в корінь диска C:.



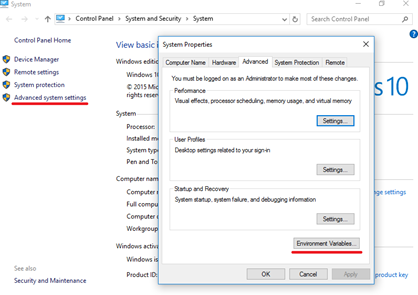
Директорія з Apache Maven.

У пошуку (натисніть клавішу Windows) знайдіть і виберіть: Система (Панель управления)/ System (Control Panel).



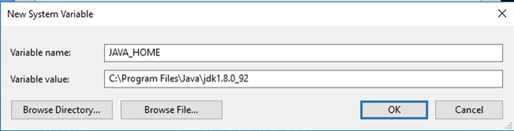
Панель управління.

Натисніть Расширенные настройки системы (Advanced system settings), далі Переменные среды (Environment Variables).

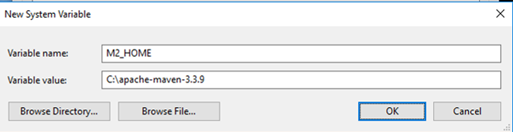


Вікно системних властивостей.

Додайте дві нові системні змінні: JAVA\_HOME і M2\_HOME натиснувши New в розділі System Variables.

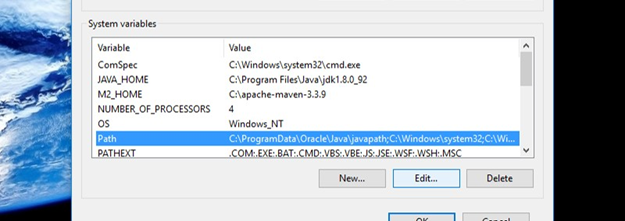


Створення змінної JAVA\_HOME.



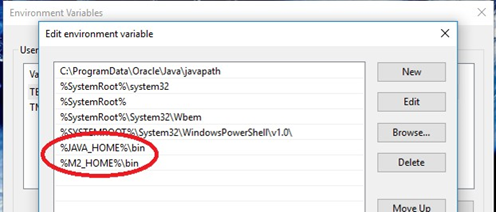
Створення змінної M2\_HOME.

Після цього знайдіть змінну Path і натисніть Edit.



Налаштування змінної PATH

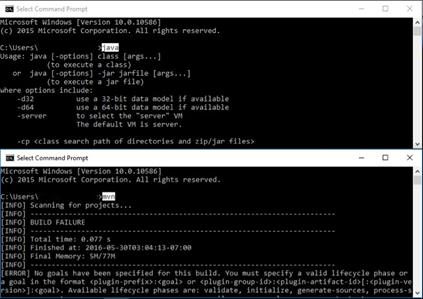
Додайте дві наступні рядки:% JAVA\_HOME% \ bin і% M2\_HOME% \ bin, натиснувши New і заповнивши пробіли.



Налаштування змінних оточення.

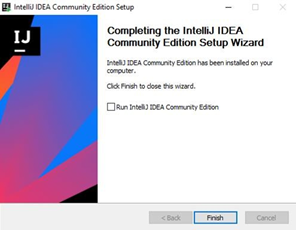
Закрийте всі вікна, натиснувши ОК.

Відкрийте командний рядок, натиснувши клавішу Windows, і введіть cmd. Спробуйте виконати наступні команди в відкрилася консолі: java і mvn. Команди повинні бути доступні.



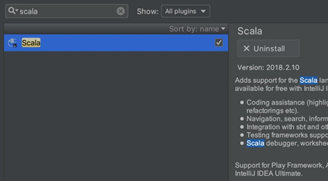
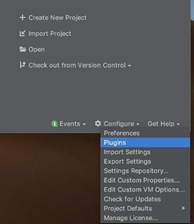
Перевірка команд java і mvn в терміналі

Скачайте і встановіть IntelliJ IDEA Community Edition, якщо вона не встановлена: http://www.jetbrains.com/idea/download. Використовуйте стандартні параметри.



IntelliJ IDEA Вікно установки.

Також необхідно встановити Scala плагін для IntelliJ IDEA.



IntelliJ IDEA, установка Scala плагіна.

**19. Створення нового проекту в Spark**

Для того щоб створити новий проект Spark ми використаємо IntelliJ IDEA. Для цього в меню натиснемо кнопку створити новий проект. На треба вибрати опцію Maven та натиснити створити. При створені потрібно вказати домашній каталог для JDK и вказати в якому каталозі створити новий проект. Після цього ми створюємо новий каталог у нашому проекті, наприклад, src/main та вказуємо, що цей каталог є корневим для проекту. Потім створюємо файл app.scala, та натискаємо на кнопку Налаштувати Scala SDK, це потрібно щоб інсталювати та використовувати бібліотеку scala у проекті

**20. Основи машинного навчання. Spark Mllib**

MLlib - це основна бібліотека Spark, яка надає безліч службових програм, корисних для задач машинного навчання, таких як:

* Класифікація;
* Регресія;
* Кластеризація;
* Моделювання;
* Сингулярного розкладання і аналізу за методом головних компонент;
* Перевірки гіпотез і статистичної вибірки.

MLlib використовують для створення додатків з можливістю прогнозного аналізу на відкритому наборі даних.

**21. Загальні відомості про класифікацію та логістичну регресію**

Класифікація - це поширена задача машинного навчання, яка являє собою процес сортування вхідних даних за категоріями. Це завдання алгоритму класифікації, що дозволяє визначити, як призначити "мітки" вхідними даними, які ви надаєте. Наприклад, можна уявити алгоритм машинного навчання, який приймає в якості вхідних даних дані про акції. Потім ділить біржову акцію на дві категорії: акції, які слід продавати і акції, які слід зберігати.

Логістична регресія - один з алгоритмів класифікації. API Spark для логістичної регресії підходить для задач двійковій класифікації або поділу вхідних даних на дві групи. Додаткові відомості про логістичної регресії см. В статті Вікіпедії.

В цілому, процес логістичної регресії створює логістичну функцію. Використовуйте функцію для прогнозування ймовірності того, що вхідний вектор належить одній групі або інший.

**22. Створення додатку машинного навчання Apache Spark MLlib. Створення**

вхідного кадру даних

Створення додатку машинного навчання Apache Spark MLlib:

1. Створіть записну книжку, використовуючи ядро PySpark.

2. Імпортуйте типи, необхідні для цього додатка. Скопіюйте та вставте наступний код в порожню клітинку, а потім натисніть клавіші SHIFT + ENTER або запустіть осередок за допомогою синього значка відтворення зліва від коду.

Python

import matplotlib.pyplot as plt

from datetime import datetime

from dateutil import parser

from pyspark.sql.functions import unix\_timestamp, date\_format, col, when

from pyspark.ml import Pipeline

from pyspark.ml import PipelineModel

from pyspark.ml.feature import RFormula

from pyspark.ml.feature import OneHotEncoder, StringIndexer,

VectorIndexer

from pyspark.ml.classification import LogisticRegression

from pyspark.mllib.evaluation import BinaryClassificationMetrics

from pyspark.ml.evaluation import BinaryClassificationEvaluator

Ядро PySpark дозволяє не ставити контексти явно. Контекст Spark буде створений автоматично при виконанні першого осередку коду.

Створення вхідного кадру даних

Оскільки необроблені дані мають формат Parquet, можна використовувати контекст Spark, щоб витягти файл в пам'ять безпосередньо у вигляді таблиці даних. Незважаючи на те, що в наведеному нижче коді використовуються параметри за замовчуванням, при необхідності можна примусово зіставити типи даних і інші атрибути схеми.

1. Виконайте наступні рядки, щоб створити кадр даних Spark, вставивши код в новий осередок. При цьому дані витягуються через API Відкритих наборів даних. Отримання всіх цих даних призводить до створення близько 1 500 000 000 строк. Залежно від розміру пулу Spark (попередня версія), необроблені дані можуть бути занадто великими або займати занадто багато часу для обробки. Ці дані можна відфільтрувати до меншого обсягу. У наступному прикладі коду використовуються параметри start\_date і end\_date для застосування фільтра, який повертає дані за один місяць.

from azureml.opendatasets import NycTlcYellow

end\_date = parser.parse('2018-06-06')

start\_date = parser.parse('2018-05-01')

nyc\_tlc = NycTlcYellow(start\_date=start\_date, end\_date=end\_date)

filtered\_df = nyc\_tlc.to\_spark\_dataframe()

2. Недолік простий фільтрації полягає в тому, що з точки зору статистичної схеми це може призвести до зміщення даних. Інший підхід полягає у використанні вибірки, вбудованої в Spark. Наступний код скорочує набір даних до 2000 рядків, якщо застосовується після наведеного вище коду. Цей крок вибірки можна використовувати замість простої фільтрації або в поєднанні з простим фільтром.

# To make development easier, faster and less expensive down sample for now

sampled\_taxi\_df = filtered\_df.sample(True, 0.001, seed=1234)

3. Тепер можна переглянути дані, щоб дізнатися, що було лічено. Зазвичай краще розглянути дані з підмножиною, а не з повним набором, в залежності від розміру набору даних. У наступному коді представлені два способи перегляду даних: перший з них є базовим, а другий надає більш широкі можливості за рахунок сітки, а також можливість візуалізувати дані в графічному вигляді.

#sampled\_taxi\_df.show(5)

display(sampled\_taxi\_df)

4. В залежності від розміру створеного набору даних і необхідності повторно запускати експеримент або записну книжку, може бути корисним кешування набору даних в локальну робочу область. Існує три способи для явного кешування:

- Збереження кадру даних локально у вигляді файлу

- Збереження кадру даних у вигляді тимчасової таблиці або подання

- Збереження кадру даних у вигляді постійної таблиці

Перші два з цих способів включені в наступні приклади коду. При створенні тимчасової таблиці або подання використовуються різні шляхи доступу до даних, які діють лише під час розмови примірника Spark.

sampled\_taxi\_df.createOrReplaceTempView("nytaxi")

**23. Створення моделі логистичної регресії**

Логістична регресія - це відповідний регресійний аналіз, коли залежна змінна є дихотомічною (двійковою). Як і всі регресійні аналізи, логістична регресія є прогностичним аналізом. Логістична регресія використовується для опису даних та для пояснення взаємозв'язку між однією залежною двійковою змінною та однією або кількома іменними, порядковими, інтервальними або незалежними змінними на рівні співвідношення. Логістична регресія використовується, коли залежна змінна (ціль) є категоричною.

from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(labelCol="Outcome", featuresCol="Aspect",weightCol="classWeights",maxIter=10)

model=lr.fit(train)

predict\_train=model.transform(train)

predict\_test=model.transform(test)

predict\_test.select("Outcome","prediction").show(10)

Гіперпараметри

До цього моменту ми розробили модель класифікації з використанням логістичної регресії. Однак робота логістичної регресії залежить від ряду параметрів. На даний момент ми працювали лише з параметрами за замовчуванням. Тепер спробуємо налаштувати гіперпараметри і подивитись на результат.

Список налаштованих параметрів у LR

1) aggregationDepth: запропонована глибина для treeAggregate (> = 2). (за замовчуванням: 2)

2) elasticNetParam: параметр змішування ElasticNet, в діапазоні [0, 1]. Для альфа = 0 - це штраф L2. Для альфа = 1 це штраф L1. (за замовчуванням: 0,0)

3) family: назва родини, яка буде використана в моделі. Підтримувані параметри: авто, біном, багаточлен (за замовчуванням: авто)

4) featuresCol: назва стовпця особливостей. (за замовчуванням: функції, поточний: Aspect)

5) fitIntercept: чи відповідає термін перехоплення. (за замовчуванням: True)

6) labelCol: назва стовпця мітки. (за замовчуванням: ярлик, поточний: Result)

7) maxIter: максимальна кількість ітерацій (> = 0). (за замовчуванням: 100, поточний: 10)

8) predictionCol: назва стовпця передбачення. (за замовчуванням: Prediction)

9) ProbaCol: Назва стовпця для передбачуваних умовних ймовірностей класу. Примітка: Не всі моделі виводять добре відкалібровані оцінки ймовірності! Ці ймовірності слід розглядати як ступінь впевненості, а не точні ймовірності. (за замовчуванням: ймовірність)

10) rawPredictionCol: ім'я стовпця непередбаченого прогнозування (так званої впевненості). (за замовчуванням: rawPrediction)

11) regParam: параметр регуляризації (> = 0). (за замовчуванням: 0,0)

12) standardization: чи слід стандартизувати навчальні особливості, перш ніж підбирати модель. (за замовчуванням: True)

13) threshold: поріг при двійковому прогнозуванні класифікації, в діапазоні [0, 1].

14) Якщо поріг і порогові значення встановлені, вони повинні збігатися, наприклад. якщо поріг дорівнює p, то порогові значення повинні дорівнювати [1-p, p]. (за замовчуванням: 0,5)

15) thresholds: пороги в багатокласовій класифікації для регулювання ймовірності прогнозування кожного класу. Масив повинен мати довжину, рівну кількості класів, зі значеннями> 0, за винятком того, що не більше одного значення може бути 0. Прогнозується клас з найбільшим значенням p / t, де p - вихідна ймовірність цього класу, а t - значення поріг класу. (невизначено)

16) tol: допуск збіжності для ітераційних алгоритмів (> = 0). (за замовчуванням: 1e-06)

17) weightCol: назва стовпця ваги. Якщо цей параметр не встановлений або порожній, ми розглядаємо всі ваги екземплярів як 1,0. (поточний: classWeights)

Тепер давайте налаштуємо деякі з цих параметрів і спостерігатимемо їх вплив на ефективність алгоритму.

З метою налаштування гіперпараметрів ми розглянемо такі параметри:

1) agregationDepth [2, 5, 10]

2) elasticNetParam [0,0, 0,5, 1,0]

3) fitIntercept [True / False]

4) maxIter [10, 100, 1000]

5) regParam [0,01, 0,5, 2,0]

Визначимо сітку параметрів наступним чином:

from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

paramGrid = ParamGridBuilder()\

.addGrid(lr.aggregationDepth,[2,5,10])\

.addGrid(lr.elasticNetParam,[0.0, 0.5, 1.0])\

.addGrid(lr.fitIntercept,[False, True])\

.addGrid(lr.maxIter,[10, 100, 1000])\

.addGrid(lr.regParam,[0.01, 0.5, 2.0]) \

.build()

**24. Навчання моделі логістичної регресії**

Навчання моделі логістичної регресії у PySpark відбувається за допомогою методу fit().

LogisticRegression.fit(trainingData, iterations=10)

Метод fit() може приймати наступні параметри:

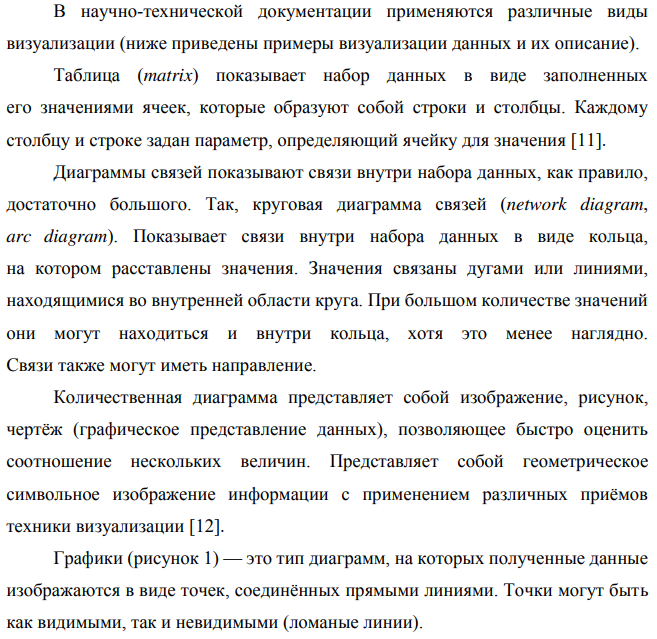
* data – дані для навчання, RDD.
* iterations – кількість ітерацій (за замовчуванням: 100)
* initialWeights – початкові ваги. (за замовчуванням відсутні)
* regParam – регуляризатор. ((за замовчуванням: 0.0)
* regType –Тип регуляризатора, що використовується для навчання моделіl. Можливі значення:
  + ”l1” для регуляризації типу L1
  + ”l2” для регуляризації типу L2(викор. За замовчуванням)
  + None - регуляризація вимкнена
* intercept – Логічний параметр, який вказує на використання розширеного представлення для навчальних даних (тобто чи активовані функції баєса чи ні). (за замовчуванням: False)
* corrections – Кількість виправлень, використаних в оновленні BFGS. Якщо для двійкової класифікації використовується відомий оновлювач, він викликає реалізацію ml, і цей параметр не матиме ефекту. (за замовчуванням: 10)
* tolerance – Показник збіжності ітерацій. (за замовчуванням: 1e-6)
* validateData – Булевий параметр, який вказує, чи повинен алгоритм перевіряти дані перед тренуванням. (за замовчуванням: True)
* numClasses – кількість класів (тобто результатів), яку мітка може пройти в багаточисельній логістичній регресії. (за замовчуванням: 2)

Після навчання моделі дані міток, ознак і тд. можуть бути отримані наступним чином:

prediction = model.transform(testData)

result = prediction.select("features", "label", "prediction").collect()

**25. Створення візуального представлення прогнозу**



**26. Обробка текстової інформації та проектування ознак засобами Spark**

Проект Apache Spark підтримує основні категорії науки про дані (Data Science) за якими здійснюється обробка текстової інформації та проектування ознак:

* коллаборативна (collaborative) фільтрація (CF) - видобуток, вилучення знань про поведінку користувача і вироблення рекомендацій - рекомендаційна система по відвідувань (сайти, документи, новини та ін.), товарам, рекомендаційна система на основі рейтингів; система аналізу текстів (завдання тематичного моделювання (topic modeling) - пошук наукової інформації, виявлення трендів і фронту досліджень та ін.);
* кластеризація (таксономія) - аналіз предметів в певному класі (наприклад, веб-сторінки, газетні статті) і організація їх у вигляді груп за подібністю ознак;
* класифікація або дискримінантний аналіз - навчання з учителем по існуючих класифікацій (навчальній вибірці), а потім віднесення некласифікованих елементів до класу, до якого в категорії кращий належить цей об'єкт. Наприклад, завдання медичної діагностики - (класифікація виду захворювання, визначення найбільш доцільного способу лікування, прогнозування тривалості і результату захворювання та ін.), оцінювання кредитоспроможності позичальників та ін .;
* визначення спільної появи ознак, часто які з набору елементів (frequent itemset mining) - аналізує предмети в групах або терміни в сесії запитів, а потім визначає, які елементи, як правило, з'являються разом.

**27. Для аналитики больших данных використовуються компоненти:**

MLLib – машинное обучение, GraphX – работа с графами, Spark-SQL

– интерфейс, SparkStreaming – потоковая аналитика. Дайте визначення

цих компонент

MLLib - проста і масштабована реалізація машинного навчання. Бібліотека містить загальні алгоритми навчання, в тому числі класифікації, кластеризації і коллаборатівной фільтрації, додаткові інструменти у вигляді набору утиліт. MLLib має відмінну інтегрувальна завдяки можливості підключення до розвинених прикладним інтерфейсів (доступний в Java, Scala, Python і SparkR) і простим розгортанням - бібліотека інсталюється на кластері Hadoop і працює з уже наявними даними. Забезпечує високу швидкість навчання в порівнянні з MapReduce і тестуванням.

GraphX ​​- компонент Spark для роботи з графами. На високому рівні GraphX ​​розширює RDD шляхом введення нової абстракції: спрямований мультіграф з властивостями, прикріпленими до кожної вершині і ребру. Для підтримки обчислень графів, Graphx надає набір основних операторів (наприклад, subgraph - повертає граф, що містить тільки вершини, що задовольняють певній умові; joinVertices - приєднує нові вершини; aggregateMessages - агрегація повідомлень до зазначеної вершині і ін.). Крім того, GraphX ​​включає в себе зростаючу колекцію алгоритмів на графах для спрощення аналітики відповідних завдань [47], наприклад, PageRank6, Personalized PageRank - алгоритми посилання ранжирування; Shortest Path - проблема знаходження найкоротшого шляху; Graph Coloring - маркування (розфарбування) вершин графа. Прикладний інтерфейс GraphX ​​забезпечує роботу з графами і графопараллельнимі обчисленнями. Модуль володіє такими важливими перевагами як гнучкість ( «бесшовность» роботи як з графами, так і з колекціями даних) і висока швидкість роботи.

Spark SQL призначений для структурованої обробки даних і реляційних запитів. Він вводить нову абстракцію даних під назвою DataFrames, яка забезпечує швидкі інтерактивні запити до структурованим і частково структурованих даних. DataFrame - розподілена колекція даних, організованих в названих колонках. Це концептуально еквівалентно таблиці в реляційної базі даних або кадру даних в R / Python, але з більш багатою внутрішньою оптимізацією. DataFrames можуть бути побудовані з широкого спектру джерел: структурованих файлів даних, таблиць у зовнішніх базах даних або існуючих RDD. Ця функція об'єднує кращі практики створення таблиць даних.

DataFrames і SQL забезпечують загальний спосіб доступу до різних джерел даних і інтеграцію SQL-запитів з усіма елементами фреймворка. Стандарт підключення до даних на сторонніх носіях: з'єднання через JDBC або ODBC. За цим же інтерфейсів забезпечується також з'єднання для інструментів бізнес-аналітики (BI). DataFrame API доступний в Scala, Java, Python і R.

SparkStreaming надає користувачеві можливість створювати і запускати додатки в потоковому режимі, а так само без істотних змін в коді може здійснювати пакетну обробку.

Модуль SparkStreaming - розширення ядра Spark API, що надає масштабуючу, відмовостійку потокову обробку реальних даних c високою пропускною спроможністю. Джерелами даних можуть бути різні системи: розподілена система передачі повідомлень Kafka, платформа для роботи з потоковими даними Kinesis, TCP-сокети або соціальні мережі, наприклад Twitter та ін. Ці дані можна обробляти за допомогою складних алгоритмів, скажімо, машинного навчання, роботи з графами і графопараллельними обчисленнями, що дозволяють виконувати розрахунки в режимі реального часу (або близькому до реального часу). Отримані результати можуть бути поміщені в файлові системи, бази даних та інформаційні панелі (live dashboards), що відображають існуючий (поточний) стан речей, близьке до реального часу вимірюваних процесів.

**28. Мова програмування Scala. Призначення.**

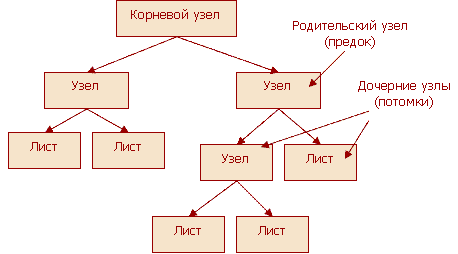
Scala - це типобезпечна мова програмування, яка поєднує функціональний й об’єктно орієнтований підхід у створенні програмного додатку. Назва походить від scalable і language, що означає - масштабована мова, тобто така, що росте разом із вимогами розробника.

Scala в основному використовується для веб розробок. Серед прикладів, на Scala написані веб-фреймворки Play, Lift, які використовуються сервісами Git, Coursera та FourSquare. Майже вся серверна частина соціальної мережі Twitter була переписана з Ruby на Scala.

**29. Алгоритм “дерево рішень”.**

Дерево прийняття рішень - це засіб підтримки прийняття рішень при прогнозуванні, широко застосовується в статистиці і аналізі даних. Нас з вами більше цікавить другий варіант використання, однак перш ніж перейти до питань про користь дерев рішень для світу великих даних і конкретним алгоритмам, хотілося б згадати про найважливіші поняттях, пов'язаних з методом в цілому.

Отже, дерево рішень, подібно до його «прототипу» з живої природи, складається з «гілок» і «листя». Гілки (ребра графа) зберігають в собі значення атрибутів, від яких залежить цільова функція; на листках ж записується значення цільової функції. Існують також і інші вузли - батьківські і нащадки - за якими відбувається розгалуження, і можна розрізнити випадки.



Загальний алгоритм побудови дерева прийняття рішення

**30. Застосування випадкових лісів для класифікації даних**

Алгоритм полягає у використанні ансамблю дерев рішень. Алгоритм поєднує в собі дві основні ідеї: метод баггінга (bagging – від bootstrap aggregation, тобто дерева навчаються по випадковим підвибіркам) і метод випадкових підпросторів.

Для беггінга є сенс використовувати велику кількість дерев рішень з досить великою глибиною. Під час класифікації фінальним результатом буде той клас, за який проголосувало більшість дерев, за умови, що одне дерево має один голос.

**31. Як працюють ансамблі.**

Стекінг. Беггінг. Бустінг  
 Ансамблеві методи - це парадигма машинного навчання, де кілька моделей (часто званих «слабкими учнями») навчаються для вирішення однієї і тієї ж проблеми і об'єднуються для отримання кращих результатів. Основна гіпотеза полягає в тому, що при правильному поєднанні слабких моделей ми можемо отримати більш точні і / або надійні моделі.  
 Основні типи мета-алгоритмів, які спрямовані на об'єднання слабких учнів:

* Беггінг. У цьому випадку часто розглядають однорідних слабких учнів, навчають їх паралельно і незалежно один від одного, а потім об'єднують їх, слідуючи деякому детермінованому процесу усередненню.
* Бустінг. У цьому випадку часто розглядають однорідних слабких учнів, навчають їх послідовно адаптивним способом (слабкий учень залежить від попередніх) і об'єднують їх, слідуючи детермінованій стратегії.
* Стекінг. У цьому випадку часто враховують різнорідних слабких учнів, вивчають їх паралельно і об'єднують їх, навчаючи метамодель для виведення прогнозу, заснованого на прогнозах різних слабких моделей.

**32. Алгоритми кластеризації. Задача кластеризації. Типи методів кластеризації.**

Задача кластеризації полягає в розбитті об’єктів з x на декілька кластерів, у яких об’єкти більш схожі між собою, ніж з об’єктами інших кластерів. У метричному просторі «схожість» звичайно визначають через відстань.

Грубо можна виділити такі кластеризації:

* Жорстка кластеризація. Кожен об'єкт або належить кластеру або ні.
* М'яка кластеризація (також нечітка кластеризація). Кожен об'єкт належить кожному кластеру до певної міри. Наприклад, це ймовірність належності кластеру.

Серед них виділяють декілька доладних:

* Жорстке розбиття на кластери. Кожен об'єкт належить рівно одному кластеру.
* Жорстке розбиття на кластери з викидами. Об'єкт може не належати жодному кластеру і розглядається як викид.
* Кластери з перетином. Об'єкт може належати більш ніж одному кластеру.
* Ієрархічна кластеризація. Якщо об'єкт належить нащадку, то він також належить і предку.
* Підпросторова кластеризація. Хоч кластери і можуть перетинатись, проте в межах визначеного підпростору кластери не перетинаються.

Алгоритми квадратичної помилки

Задачу кластеризації можна розглядати як побудова оптимального розбиття об'єктів на групи. При цьому оптимальність може бути визначена як вимога мінімізації середньоквадратичної помилки розбиття:



де cj - «центр мас» кластера j (точка з середніми значеннями характеристик для даного кластера).

Алгоритми квадратичної помилки відносяться до типу плоских алгоритмів. Найпоширенішим алгоритмом цієї категорії є метод k-середніх. Цей алгоритм будує задане число кластерів, розташованих якнайдалі один від одного. Робота алгоритму ділиться на кілька етапів:

1. Випадково вибрати k точок, які є початковими «центрами мас» кластерів.
2. Віднести кожен об'єкт до кластеру з найближчим «центром мас».
3. Перерахувати «центримас» кластерів відповідно до їх поточним складом.
4. Якщо критерій зупинки алгоритму не задоволений, повернутися до п. 2.

Як критерій зупинки роботи алгоритму зазвичай вибирають мінімальне зміна среднеквадратической помилки. Так само можливо зупиняти роботу алгоритму, якщо на кроці 2 не було об'єктів, що перемістилися з кластера в кластер.

До недоліків даного алгоритму можна віднести необхідність задавати кількість кластерів для розбиття.

**33. Ієрархічна кластеризація. Застосування методів кластеризації.**

Ієрархічна кластеризація — метод кластерного аналізу, який намагається побудувати ієрархію кластерів. Стратегії побудови ієрархічної кластеризації діляться на два типи: агломератові та розділювальні.

Ієрархічні алгоритми дозволяють одержувати послідовну розбивку сукупності об'єктів за певним правилом. Вони підрозділяються на подільні й агломеративні. Подільні алгоритми починають роботу з розгляду вихідної сукупності як одного кластера й послідовно розділяють її на більш дрібні групи, аж до розбивки, коли кожний об'єкт вважається окремим кластером . В результаті утворюється ієрархічне дерево кластерів, і аналітик може вибрати ту її конфігурацію, яка краще відповідає розв'язанню задачі. В агломеративній кластеризації також формується ієрархічне дерево, але шляхом об'єднання об'єктів в більш великі кластери з більш дрібних. Спочатку кожен об'єкт вихідної безлічі розглядається як окремий кластер, потім шукаються два об'єкти, відстань між якими мінімальна, і об'єднуються в один і т.д. Дана процедура продовжується до тих пір, поки всі об'єкти не будуть зібрані в єдиний кластер.

**34. Алгоритми зниження розмірності**

Алгоритми зменшення розмірності 2.1 PCA PCA метод, який проектує дані на нову координатну систему меншої розмірності, яка визначається власними векторами і власними числами матриці. Хоча існує кілька алгоритмів, що працюють таким чином, PCA є найбільш популярним з них. З цієї причини він включений в даний порівняння. PCA включає в себе обчислення ковариационной матриці даних, щоб мінімізувати надмірність і максимізувати дисперсію. Математично PCA визначається як ортогональное лінійне перетворення і вважає всі базисні вектори ортонормованій матрицею [1] [2].

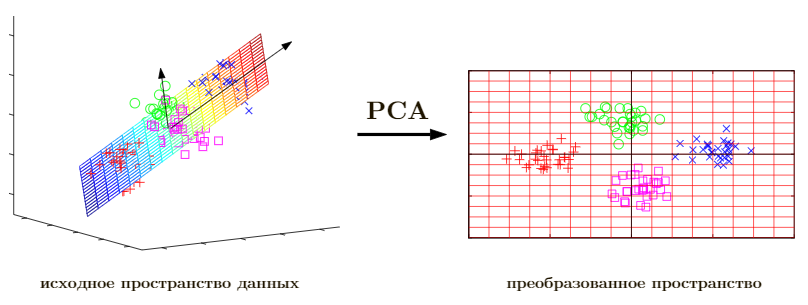
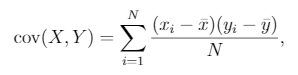


Рис. 1: Principal Component Analisys

PCA працює за допомогою знаходження власних чисел і власних векторів матриці коваріації. Матриця ковариации використовується, щоб виміряти, як багато размерностей відрізняються від середнього по відношенню один до одного. Коваріація двох випадкових величин (розмірностей) міра їх лінійної залежності:

cov (x, Y) = E [(X EX) (Y EY)],

де E [X] і E [Y] математичне сподівання випадкової величини X і Y відповідно. Для певного набору даних ми можемо записати:

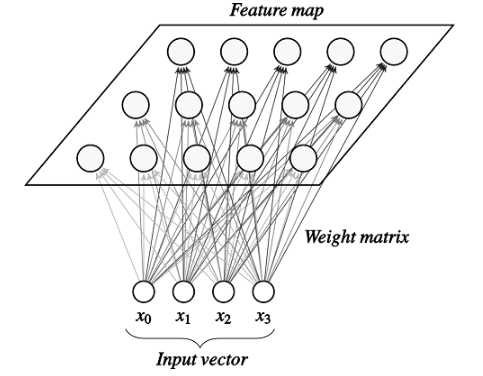


де x середнє X, y середнє Y, а N розмірність даних. Матриця ковариации матриця A з елементами A ij = cov (i, j). Вона центрує дані, віднімаючи середнє з кожного об'єкта.

**35. Карти самоорганізації. Основні концепції.**

Карта самоорганізаціїї - це штучна нейронна мережа заснована на навчанні без учителя. У картах самоорганізації нейрони помещени в вузлах решітки, зазвичай одно- або двовимірної.Всі нейрони цієї решітки пов'язані з усіма вузлами вхідного шару.

Створює дискретне представлення вхідних просторів навчальних вибірок, які називаються картою і тому використання цього типу нейронної мережі є методом для зниження дискретності



**36. Конкурентне навчання. Архітектура. Топологія. Алгоритм навчання. Підходи до визначення відстаней. Застосування.**

Конкурентне навчання є формою навчання без нагляду у штучних нейронних мережах, в яких вузли конкурують за право реагувати на підмножину вхідних даних. Варіант навчання за Геббовою теорією, конкурентне навчання працює шляхом підвищення спеціалізації кожного вузла в мережі. Воно добре підходить для пошуку кластерів у межах даних. Конкурентне навчання зазвичай реалізується з нейронними мережами, які містять прихований шар, який широко відомий як «конкурентний шар» . Кожен конкурентний нейрон описується вектором вагів і обчислює коефіцієнт подібності між вхідними даними та ваговим вектором. Наведемо простий алгоритм конкурентного навчання, щоб знайти три кластера в межах деяких вхідних даних.

1. Налаштування: Нехай набір датчиків подається в три різні вузли, так що кожен вузол підключається до кожного датчика. Нехай ваги, які кожен вузол дає своїм датчикам, встановлюються випадковим чином від 0,0 до 1,0. Нехай вихід кожного вузла є сумою всіх його датчиків, причому потужність сигналу кожного датчика помножується на його вагу.

2. Коли у мережу подаються вхідні дані, вузол з найбільшим виходом вважається переможцем. Вхідні дані класифікується так, якби вони були всередині кластера, що відповідає цьому вузлу.

3. Вузол-переможець оновлює кожну свою вагу, переміщаючи вагу від з'єднань тих датчиків, що дали йому слабкі сигнали, до тих, що дали йому сильніші сигнали.

Таким чином, при отриманні великої кількості даних, кожен вузол сходиться докупи в центрі кластера, який він представляє і активізується з більшою ймовірністю для вхідних даних у цьому кластері та з меншою ймовірністю для вхідних даних з інших кластерів.

**37. Створення рекомендаційних систем засобами Spark MLlib**

Рекомендаційні системи-програми, які намагаються передбачити, які об'єкти (фільми, музика, книги, новини, веб-сайти) будуть цікаві користувачеві, маючи певну інформацію про його профілі [1]. Рекомендаційні системи порівнюють однотипні дані від різних людей і обчислюють список рекомендацій для конкретного користувача. Це альтернатива пошуковим алгоритмам, так як дозволяють виявити об'єкти, які не можуть бути знайдені останніми [2] .виду рекомендаційних систем:

* Статістіческіе (statistical approach) -це системи, які ґрунтуються на статистичних даних, собраннихс користувачів;
* Демографіческіе (demographic recommendation) порівнюють характеристики об'єкта з характеристиками користувача;
* Ассоціатівние (association rules) будують рекомендації на основі даних про те, які об'єкти використовуються разом;
* Інформаціонние (content based) -це системи, які шукають об'єкти, які схожі на ті, чтопользователь вже позитивно оцінив;
* Коллектівние (коллаборатівного) (collaborative filtering) -самі поширені системи, які для передбачення рейтингу певного об'єкта керуються оцінками інших користувачів;
* Гібрідние (hybrid) - використовують трохи вище згаданих технологій.

Розглянемо завдання розробки рекомендаційної системи для інтернетсервіса великої мережі ресторанів. Сервіс надає можливість зробити замовлення через Інтернет. Близько 1 000 000 користувачів протягом останніх 6 років зробили близько 2 500 000 замовлень. До сервісу постійно приєднуються нові ресторани, реєструються нові користувачі і, відповідно, обсяги даних мають тенденцію до увеліченію.Необходімо провести аналіз замовлень і надати рекомендації про те, коли відправити користувачеві повідомлення і які додаткові продукти можуть його заінтересовать.Данние зберігаються в MongoDB -документооріентірованной системи управління базами даних з відкритим вихідним кодом, що не вимагає опису схеми таблиць. У замовленні відображено хто, коли зробив замовлення, найменування продуктів і т.д. Враховуючи обсяг інформації, обгрунтовано буде звернутися до алгоритмів машинного навчання і аналізу даних. В світі, де інформації все більше, машинне навчання часто єдиний спосіб якось її усвідомити [6]. Для вирішення поставленого завдання обраний Apache Spark -швидкий і потужний механізм для великомасштабної обробки даних [3]. Spark надає бібліотеку (ML) для роботи з алгоритмами машинного навчання. Однак, перш за все необхідно налаштувати інтегрування даних в Spark, т.к.они зберігаються не в HDFS (розподілена файлова система Apache Hadoop, яку використовує Apache Spark). Даний процес спростився з переходом на нову версію Apache Spark (2.2.0). Використовуючи mongo-spark connector, імпорт даних виглядає наступним чином:

spark = SparkSession

.builder

.appName("ALS")

.config("spark.mongodb.input.uri",

"mongodb://hadoop.master:27017:restaurants.orders")

.getOrCreate()

df = spark.read.format("com.mongodb.spark.sql.DefaultSource")

.load(

**38. Перспективні напрямки розвитку програмних засобів обробки надвеликих даних і машинного навчання.**

В зв’язку із стрімким ростом ІТ ринку, багато великих компаній збирають велику кількість даних, що призведе до попиту на нові програмні засоби обробки великих даних не тільки в науковій сфері, а й у більшості інших галузей. Також цьому сприяє розуміння компаніями того, що аналіз даних споживачів їхньої продукції моде серйозно підвищити їх конкурентоздатність.

В той же час розвиток апаратної частни призведе до збільшення обчислювальної потужності, що дозволить використовувати нові способи програмної обробки великих даних, що раніше відкидалися або вважалися неможливими через занадто високу вимогливість до обчислювальних ресурсів.

**39. Сучасні напрямки в машинному навчанні.**

Існує багато напрямків розвитку машинного навчання, але найголовніші з них 4:

* Економічна ефективність
* спеціалізоване обладнання
* Робота над призначеними для користувача інтерфейсами
* вертикальні додатки

**40. Програмні рішення обробки надвеликих даних в задачах машинного навчання від лідерів ринку (Amazon, Google, Facebook).**

Платформи для машинного навчання та роботи з надвеликими даними. Машинне навчання як сервіс. Машинне навчання на AWS. Azure Machine Learning Packages. Google Cloud ML Engine. IBM Data Science Experience.

FBLearner. Amazon Web Services або AWS (читається як ей дабл ю ес) є дочірньою компанією Amazon.com, що надає платформу хмарних обчислень в оренду приватним особам, компаніям та урядам на основі платної підписки. Існує і безкоштовна підписка, яка доступна протягом перших 12 місяців. Технологія дозволяє абонентам мати у своєму розпорядженні повноцінний віртуальний кластер комп'ютерів, який завжди доступний через Інтернет. Віртуальні комп'ютери AWS мають більшість атрибутів реального комп'ютера, включаючи апаратні пристрої (процесор, відеокарту, локальну та оперативну пам'ять, жорсткий диск або SSD-накопичувач); операційну систему на вибір; мережу; і попередньо встановлені прикладні програми, такі як веб-сервер, база даних, CRM і т. д. Кожна система AWS також віртуалізує консольний ввід/вивід (клавіатура, дисплей і миша), що дозволяє користувачам AWS підключитися до своєї системи AWS за допомогою браузера. Браузер виступає як вікно у віртуальний комп'ютер, дозволяючи користувачу входити в систему, налаштовувати та використовувати свої віртуальні системи так само, як справжній, фізичний комп'ютер. Це дозволяє їм налаштувати систему так, щоб надавати інтернет-орієнтовані сервіси та послуги своїм клієнтам.

Технологія AWS базується на серверних кластерах (фермах), розташованих по всьому світі. Плата за користування базується на комбінації використання апаратних засобів/ОС/програмного забезпечення/мережевих функцій, вибраних користувачем, а також вимог до доступності[en], надлишковості (redundancy), безпеки та додаткових параметрів. Виходячи з того, що користувач потребує і оплачує, він може зарезервувати один віртуальний комп'ютер (VM), кластер віртуальних комп'ютерів (VM Cluster), фізичний (реальний) комп'ютер (Server), призначений для його виняткового використання, або навіть кластер фізичних комп'ютерів (Server Cluster). Компанія Amazon зобов'язується керувати та оновлювати програмне та апаратне забезпечення для дотримання необхідних стандартів безпеки. AWS працює в багатьох географічних регіонах, у тому числі в Канаді, Німеччині, Ірландії, Сінгапурі, Токіо, Сіднеї, Пекіні, Лондоні і т. д.

Google Cloud Platform — запропонований компанією Google набір хмарних служб, які виконуються на тій же самій інфраструктурі, яку Google використовує для своїх продуктів призначених для кінцевих споживачів, таких як Google Search та YouTube. Окрім інструментів для керування, також надається ряд модульних хмарних служб, таких як обчислення, зберігання даних, аналіз даних та машинне навчання. Для реєстрації потрібно мати банківську карту або банківський рахунок.

Google Cloud Platform надає такі оточення як інфраструктура як послуга, платформа як послуга, та безсерверні обчислення.

У квітні 2008, Google анонсувала App Engine — платформу для розробки та хостингу веб-застосунків у дата-центрах Google. Це був перший хмарний сервіс запропонований компанією. Для загалу сервіс став доступним у грудні 2011 року. З моменту анонсу App Engine, Google встигла додати чисельні хмарні служби до своєї платформи.

Microsoft Azure реалізує Хмарний моделі платформи як сервісу (PaaS) та інфраструктури як сервісу (IaaS). Можливе використання як сторонніх, так і сервісів Microsoft в якості моделі ПО як сервісу (SaaS). Працездатність платформи Microsoft Azure забезпечує глобальна мережа розподілених дата-центрів Microsoft.

Machine Learning As a Service (MLaaS) - це набір послуг, що надають засоби машинного навчання як частину послуг хмарних обчислень. MLaaS допомагає клієнтам отримувати вигоди від машинного навчання без спільних витрат, часу та ризику створення внутрішньої внутрішньої команди машинного навчання.

Машинне навчання Azure (Azure ML) - це хмарний сервіс для створення та управління рішеннями машинного навчання. Він покликаний допомогти вченим даних та інженерам машинного навчання використовувати свої існуючі навички обробки даних та розробки моделей. Крім того, допоможіть їм масштабувати, розподіляти та розгортати свої робочі навантаження в хмарі. SDK Azure ML для Python надає класи, які ми можемо використовувати для роботи з Azure ML у нашій передплаті Azure.

Google Cloud ML Engine - це розміщена платформа для масштабного запуску навчальних завдань та прогнозів машинного навчання. Служба розглядає ці два процеси (навчання та прогнозування) незалежно.

IBM Data Science Experience - це інтерактивне спільне хмарне середовище, де вчені можуть використовувати різноманітні інструменти для активації своєї ідеї. Вчені-дослідники можуть використовувати найкраще з відкритого коду, використовувати унікальні функції IBM, розвивати свої можливості та ділитися своїми успіхами.