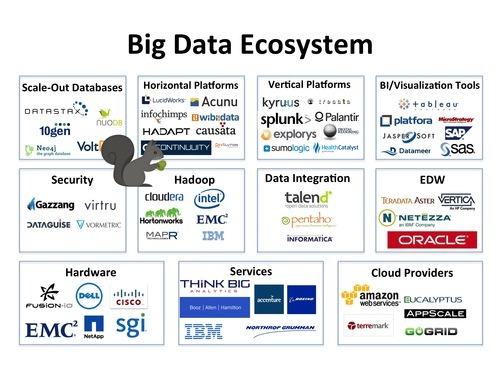
# CHƯƠNG 1 GIỚI THIỆU

**Big Data** là các tập dữ liệu có khối lượng lớn và phức tạp. Độ lớn đến mức các phần mềm xử lý dữ liệu truyền thống không có khả năng thu thập, quản lý và xử lý dữ liệu trong một khoảng thời gian hợp lý.

Những tập dữ liệu lớn này có thể bao gồm các dữ liệu có cấu trúc, không có cấu trúc và bán cấu trúc, mỗi tập có thể được khai thác để tìm hiểu insights.



Hình1.1 Hình ảnh hệ sinh thái Big Data

Dưới đây là một số ứng dụng trong mỗi chuyên môn khác nhau:

* Netflix sử dụng Big Data để cải thiện trải nghiệm của khách hàng
* Phân tích chiến dịch và kế hoạch xúc tiến của Sears Holding
* Kết hợp và quảng cáo theo thời gian thực

**Công nghệ Big datas**

Có rất nhiều công nghệ để giải quyết vấn đề lưu trữ và xử lý Big Data. Những công nghệ đó là Apache Hadoop, Apache Spark, Apache Kafka, v.v … Dưới đây là thông tin tổng quan về các công nghệ này.

* **Apache Hadoop**

Hadoop là một Apache framework mã nguồn mở được viết bằng Java, cho phép xử lý phân tán (distributed processing) các tập dữ liệu lớn trên các cụm máy tính (clusters of computers) thông qua mô hình lập trình đơn giản. Hadoop được thiết kế để mở rộng quy mô từ một máy chủ đơn sang hàng ngàn máy tính khác có tính toán và lưu trữ cục bộ (local computation and storage).

* **Apache Spark**

Sự cải tiến hơn nữa của công nghệ này đã dẫn đến sự phát triển của Apache Spark – công cụ tính toán nhanh và có mục đích chung để xử lý quy mô lớn. Nó có thể xử lý dữ liệu nhanh hơn 100 lần so với MapReduce.

* **Apache Kafka**

Apache Kafka là một bổ sung khác cho hệ sinh thái Big Data, một hệ thống nhắn tin phân tán lượng thông tin cao thường được sử dụng với Hadoop.

* *Tóm lại, Big Data là thách thức đặt ra cho các tổ chức, doanh nghiệp trong thời đại số hiện nay. Một khi làm chủ được Big Data thì họ sẽ có cơ hội thành công lớn hơn trong bối cảnh cạnh tranh ngày nay. Thế giới thì sẽ được hưởng lợi hơn từ việc trích xuất thông tin một cách chính xác hơn, hữu ích hơn với chi phí thấp hơn.*

## Mục tiêu nghiên cứu

* Mục tiêu nghiên cứu chính của đề tài này là: Sử dụng Logistic Regression trong thư viện PySpark để phân loại khách hàng đăng kí tài khoản tiền gửi có kì hạn.
* Tìm hiểu cách các thuật toán học máy áp dụng như thế nào trong những hệ thống phân tán.
* Tiến hành chạy và thực nghiệm bộ dữ liệu thực tế trên những thuật toán được chọn để so sánh và đưa ra kết luận về hiệu suất của các mô hình khi áp dụng với những tham số khác nhau.
* Rút ra kết luận về đề tài nghiên cứu, đưa ra định hướng trong tương lai.

## Phương pháp nghiên cứu

* Áp dụng những kiến thức đã học trong môn Bigdata để tìm hiểu các thuật toán và cách triển khai của chúng trên những hệ thống như HDFS hay PySpark.
* Nghiên cứu từ những nguồn thông tin hữu ích trên nền tảng mạng để triển khai phần code thực nghiệm.
* Thực nghiệm và so sánh giữa những khía cạnh khác nhau của các công cụ là phương thức chính trong bài nghiên cứu này.

# CHƯƠNG 2 CƠ SỞ LÝ THUYẾT

## 2.1 Giới thiệu về HDFS

HDFS (Hadoop Distributed File System) là hệ thống tập tin phân tán, một nền tảng lưu trữ dữ liệu đáp ứng cho một khối lượng dữ liệu lớn và chi phí rẻ.

Đã có rất nhiều Hadoop cluster chạy HDFS

* Các công ty khác nhưng Facebook, Adobe, Amazon cũng đã xây dựng các cluster chạy HDFS với dung lượng hàng trăm, hàng nghìn TB.

**Về tổng quan thiết kế của HDFS:**

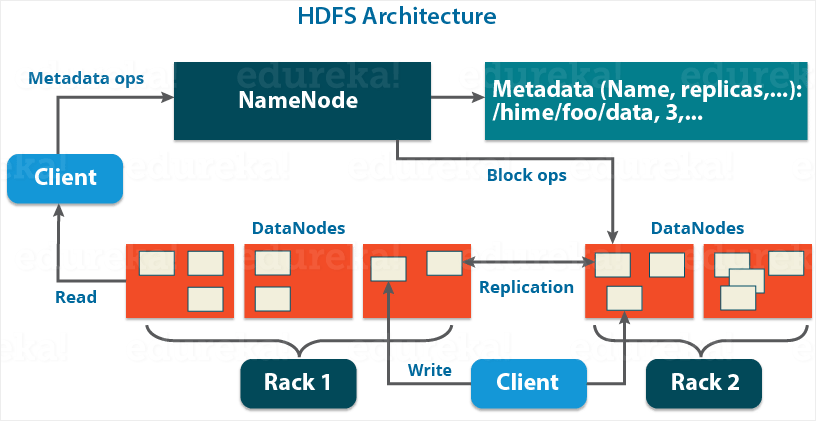
Giống như các hệ thống file khác, HDFS duy trì một cấu trúc cây phân cấp các file, thư mục mà các file sẽ đóng vai trò là các node lá.

Mỗi file được chia ra làm một hay nhiều block và mỗi block có một block ID để nhận diện.

Mỗi block của file được lưu trữ thành nhiều bản sao khác nhau vì mục đích an toàn dữ liệu.

Mỗi cluster có một NameNode và một/chiều DataNode

NameNode đóng vai trò là một master, chịu trách nhiệm duy trì thông tin về cấu trúc cây phân cấp các file, thư mục của hệ thống file và ccs metadata khác của hệ thống file.



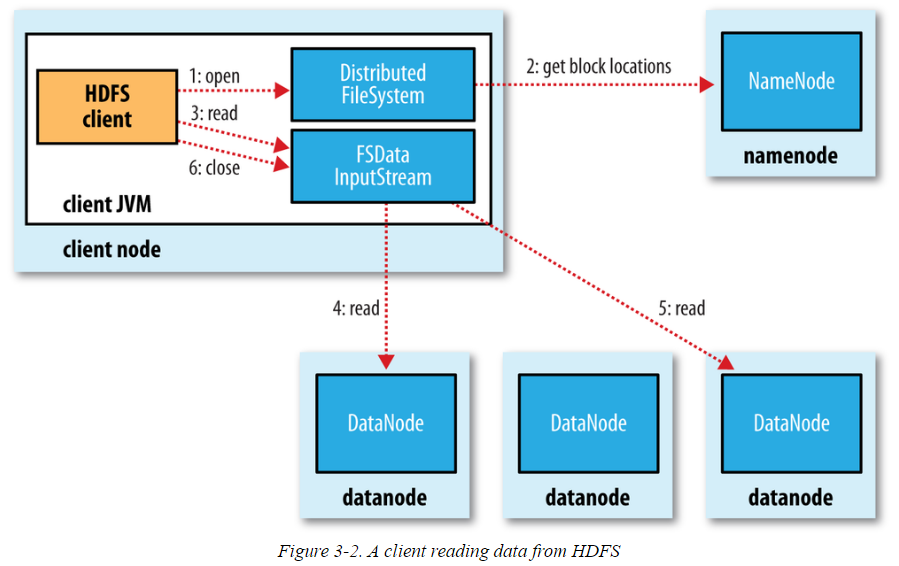
Hình 2.1 Hình ảnh HDFS Architecture

Metadata mà Namenode lưu trữ gồm có:

* File System Namespace: là hình ảnh cây thư mục của hệ thống file tại một thời điểm nào đó, thể hiện tất cả các file thư mục có trên hệ thống file và quan hệ giữa chúng.
* Thông tin để ánh xạ từ tên file ra thành danh sách các block với mỗi file, ta có một danh sách có thứ tự các block (đại diện bởi block id) của file đó.
* Thông tin nơi lưu trữ các block: Mỗi block ta có một danh sách các Datanode lưu trữ các bản sao của block đó.

Datanode:

* Chịu trách nhiệm lưu trữ các block thật sự của từng file của hệ thống file phân tán lên hệ thống file cục bộ của Datanode
* Mỗi 1 block được lưu trữ như là 1 file riêng biệt trên hệ thống file cục bộ của Datanode



Hình 2.2 Hình ảnh tương tác của client với NameNode và DataNode

Khi client của hệ thống muốn đọc 1 file trên hệ thống HDFS:

* Client này sẽ thực hiện một request đến Namenode để lấy các metadata của file cần đọc => biết được danh sách các block của file và vị trí của các Datanode chứa các bản sao của từng block
* Client kết nối trực tiếp với các Datanode để thực hiện các request đọc dữ liệu các block

Định kỳ, mỗi Datanode báo cáo cho Namenode biết về danh sách tất cả các block mà nó đang lưu trữ. Namnode dựa vào những thông tin này để cập nhật lại các Metadata trong nó. Metadata trên namenode sẽ đạt được tình trạng thống nhất với dữ liệu trên các Datanode.Metadata ở trạng thái thống nhất được dùng để nhân bản metadata dùng cho mục đính phục hồi lại Namenode nếu Namenode bị lỗi.

Khả năng chịu lỗi của HDFS: NameNode và Datanode đều được thiết kết để có thể phục hồi nhanh chóng.

NameNode và Datanode liên lạc thông qua HeartBeat

* HeartBeat là tín hiệu được gửi bởi Datanode đến Namenode sau một khoảng thời gian thông thường để biểu thị sự hiện diện của nó (tức là nó còn sống)
* Nếu sau khoảng thời gian nhất định, Namenode không nhận được bất kỳ phản hồi nào từ Datanode thì Datanode đã ngưng hoạt động

Trường hợp Namenode ngừng hoạt động

* Chỉ cần phục hồi lại Namenode mà không cần phải restart tất cả các Datanode
* Namenode sau khi phục hồi sẽ tự động liên lạc lại với các Datanode => hệ thống lại phục hồi (thực chất là Namenode chỉ đứng yên và lắng nghe các HeartBeat từ các Datanode)

Nếu một Datanode bất kỳ bị ngưng hoạt động

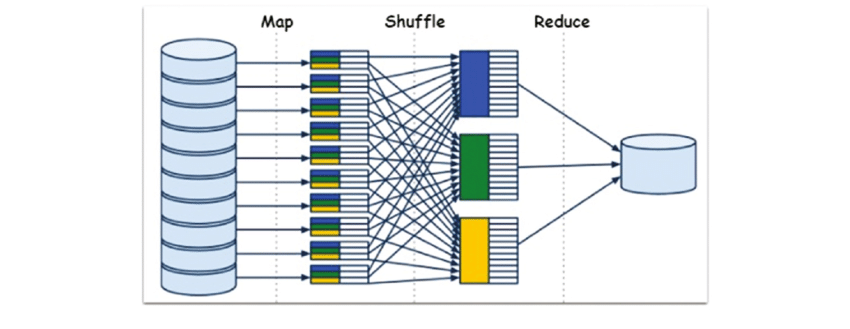
* Chỉ cần khởi động lại DataNode này
* Datanode sẽ tự động liên lạc với Namenode thông qua các HeartBeat để cập nhật lại tình trạng của mình trên Namenode.

Nhân bản metadata trên Namenode với SecondaryNameNode:

* SecondaryNameNode là một node duy nhất trên Hadoop cluster. Nhiệm vụ của SecondaryNameNode là lưu trữ lại checkpoint mới nhất trên NameNode.
* Khi NameNode gặp sự cố, checkpoint mới nhất này được import vào NameNode => NameNode trở lại hoạt động bình thường như thời điểm SecondaryNameNode tạo checkpoint.

## 2.2 Giới thiệu về Hadoop và MapReduce

MapReduce là mô hình được thiết kế độc quyền bởi Google, nó có khả năng lập trình xử lý các tập dữ liệu lớn song song và phân tán thuật toán trên 1 cụm máy tính. MapReduce trở thành một trong những thành ngữ tổng quát hóa trong thời gian gần đây.



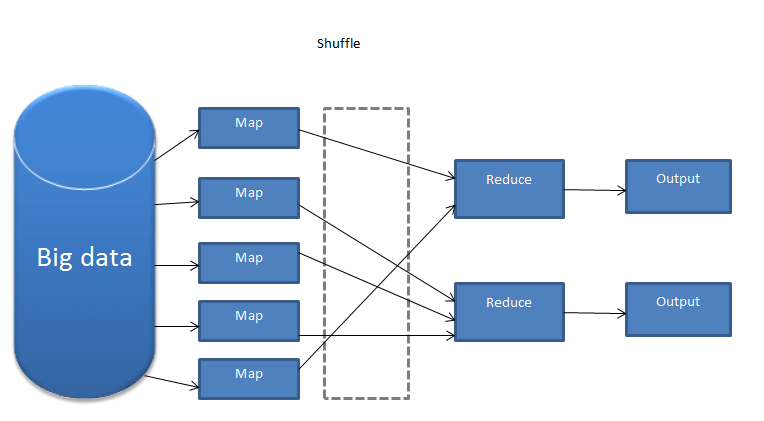
Hình 2.3 Hình ảnh kiến trúc MapReduce

MapReduce sẽ bao gồm những thủ tục sau: thủ tục 1 Map() và 1 Reduce(). Thủ tục Map() bao gồm lọc (filter) và phân loại (sort) trên dữ liệu khi thủ tục khi thủ tục Reduce() thực hiện quá trình tổng hợp dữ liệu. Đây là mô hình dựa vào các khái niệm biển đối của bản đồ và reduce những chức năng lập trình theo hướng chức năng.

Các hàm chính của MapReduce:

MapReduce có 2 hàm chính là Map() và Reduce(), đây là 2 hàm đã được định nghĩa bởi người dùng và nó cũng chính là 2 giai đoạn liên tiếp trong quá trình xử lý dữ liệu của MapReduce. Nhiệm vụ cụ thể của từng hàm như sau:

* **Hàm Map():** có nhiệm vụ nhận Input cho các cặp giá trị/  khóa và output chính là tập những cặp giá trị/khóa trung gian. Sau đó, chỉ cần ghi xuống đĩa cứng và tiến hành thông báo cho các hàm Reduce() để trực tiếp nhận dữ liệu.
* **Hàm Reduce():** có nhiệm vụ tiếp nhận từ khóa trung gian và những giá trị tương ứng với lượng từ khóa đó. Sau đó, tiến hành ghép chúng lại để có thể tạo thành một tập khóa khác nhau. Các cặp khóa/giá trị này thường sẽ thông qua một con trỏ vị trí để đưa vào các hàm reduce.
* Ở giữa Map và Reduce thì còn 1 bước trung gian đó chính là Shuffle. Sau khi Map hoàn thành xong công việc của mình thì Shuffle sẽ làm nhiệm vụ chính là thu thập cũng như tổng hợp từ khóa/giá trị trung gian đã được map sinh ra trước đó rồi chuyển qua cho Reduce tiếp tục xử lý.



Hình 2.4 Hình ảnh kiến trúc của MapReduce

Các ưu điểm nổi bật của MapReduce

Mapreduce được ưa chuộng sử dụng như vậy bởi nó sở hữu nhiều ưu điểm vượt trội như sau:

* MapReduce có khả năng xử lý dễ dàng mọi bài toán có lượng dữ liệu lớn nhờ khả năng tác vụ phân tích và tính toán phức tạp.
* Với khả năng hoạt động độc lập kết hợp phân tán, xử lý các lỗi kỹ thuật để mang lại nhiều hiệu quả cho toàn hệ thống.
* MapReduce có khả năng thực hiện trên nhiều nguồn ngôn ngữ lập trình khác nhau như: [Java ,](https://itnavi.com.vn/blog/java-la-gi-nhung-dieu-can-biet-truoc-khi-lap-trinh-java/) [C/ C++](https://itnavi.com.vn/blog/ngon-ngu-lap-trinh-c-va-c/), [Python](https://itnavi.com.vn/blog/python-la-gi-nhung-dieu-ban-can-biet-de-hoc-ngon-ngu-python/),... tương ứng với nó là những thư viện hỗ trợ.

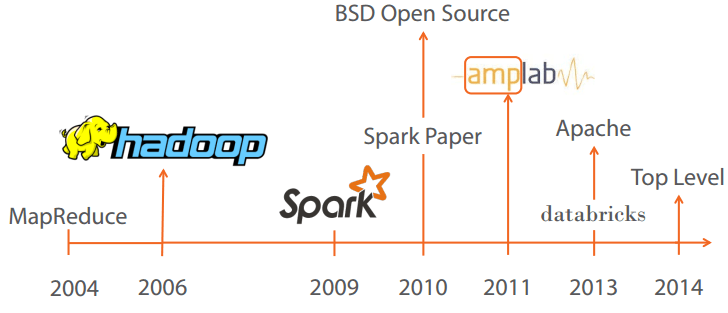
## 2.3 Apache Spark

Ưu điểm lớn nhất của Hadoop là được dựa trên một mô hình lập trình song song với xử lý dữ liệu lớn là MapReduce, mô hình này cho phép khả năng tính toán có thể mở rộng, linh hoạt, khả năng chịu lỗi, chi phí rẻ.

Apache Haddop có một nhược điểm là tất cả các thao tác đều phải thực hiện trên ổ đĩa cứng điều này đã làm giảm tốc độ tính toán đi gấp nhiều lần.

Để khắc phục được nhược điểm này thì Apache Spark được ra đời. Apache Spark có thể chạy nhanh hơn 10 lần so với Haddop ở trên đĩa cứng và 100 lần khi chạy trên bộ nhớ RAM.

### 2.3.1 giới thiệu về Apache Spark



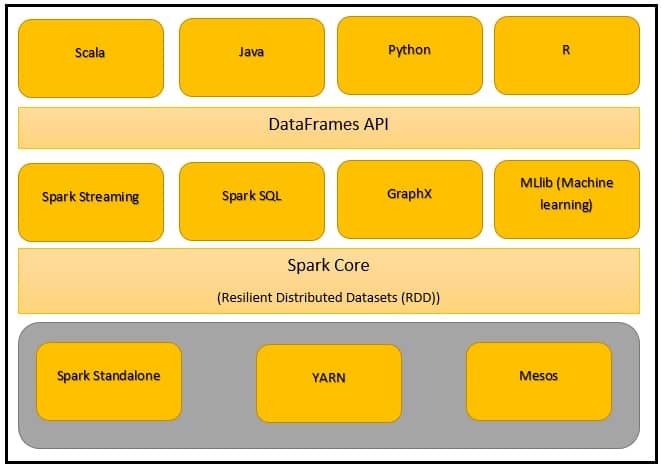
Hình 2.5 Thời gian ra đời của tool Big Data

Tốc độ xử lý của Spark có được do việc tính toán được thực hiện cùng lúc trên nhiều máy khác nhau. Đồng thời việc tính toán được thực hiện ở bộ nhớ trong (in-memories) hay thực hiện hoàn toàn trên RAM.

Spark cho phép xử lý dữ liệu theo thời gian thực, vừa nhận dữ liệu từ các nguồn khác nhau đồng thời thực hiện ngay việc xử lý trên dữ liệu vừa nhận được (Spark Streaming).

Spark không có hệ thống file của riêng mình, nó sử dụng hệ thống file khác như: HDFS, Cassandra, S3, …. Spark hỗ trợ nhiều kiểu định dạng file khác nhau (text, csv, json…) đồng thời nó hoàn toàn không phụ thuộc vào bất cứ một hệ thống file nào.

### 2.3.2 Các thành phần của Apache Spark



Hình 2.6 Hình ảnh kiến trúc của Apache Spark

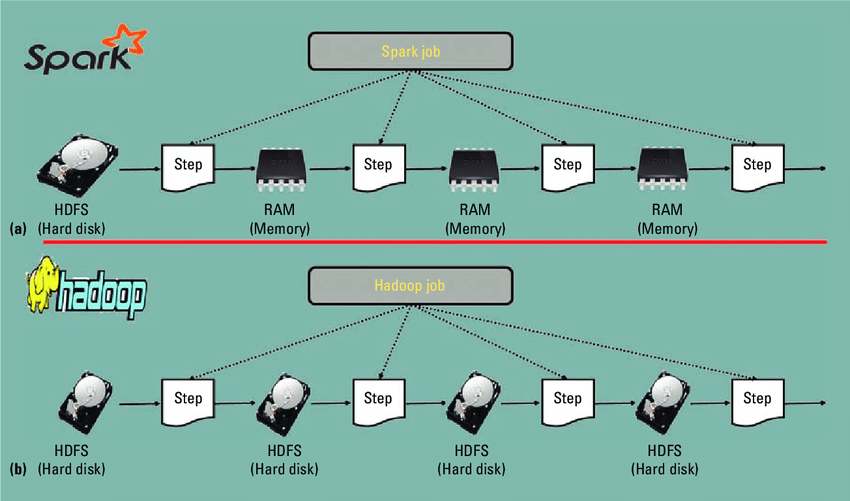
**Apache Spark** gồm có 5 thành phần chính: Spark Core, Spark Streaming, Spark SQL, MLlib và GraphX, trong đó:

* **Spark Core** là nền tảng cho các thành phần còn lại và các thành phần này muốn khởi chạy được thì đều phải thông qua Spark Core do Spark Core đảm nhận vai trò thực hiện công việc tính toán và xử lý trong bộ nhớ (In-memory computing).
* **Spark SQL** cung cấp một kiểu data abstraction mới (SchemaRDD) nhằm hỗ trợ cho cả kiểu dữ liệu có cấu trúc (structured data) và dữ liệu nửa cấu trúc (semi-structured data). **Spark SQL** hỗ trợ DSL (Domain-specific language) để thực hiện các thao tác trên DataFrames bằng ngôn ngữ Scala, Java hoặc Python.
* **Spark Streaming** được sử dụng để thực hiện việc phân tích stream bằng việc coi stream là các mini-batches và thực hiệc kỹ thuật RDD transformation đối với các dữ liệu mini-batches này.
* **MLlib** (Machine Learning Library): MLlib là một nền tảng học máy phân tán bên trên Spark do kiến trúc phân tán dựa trên bộ nhớ. Theo các so sánh benchmark Spark MLlib nhanh hơn 9 lần so với phiên bản chạy trên Hadoop (Apache Mahout).
* **GrapX**: Grapx là nền tảng xử lý đồ thị dựa trên Spark. Nó cung cấp các Api để diễn tảcác tính toán trong đồ thị bằng cách sử dụng Pregel Api.

### 2.3.3 Những điểm nổi bật của Spark

* Xử lý dữ liệu: Spark xử lý dữ liệu theo lô và thời gian thực
* Tính tương thích: Có thể tích hợp với tất cả các nguồn dữ liệu và định dạng tệp được hỗ trợ bởi cụm Hadoop.
* Hỗ trợ ngôn ngữ: hỗ trợ Java, Scala, Python và R.
* Phân tích thời gian thực: Apache Spark có thể xử lý dữ liệu thời gian thực tức là dữ liệu đến từ các luồng sự kiện thời gian thực với tốc độ hàng triệu sự kiện mỗi giây.

### 2.3.4 So sánh giữa Spark và Hadoop MapReduce



Hình 2.7 Hình ảnh hoạt động của Spark Job vs Hadoop Job

|  |  |
| --- | --- |
| MAP REDUCE | SPARK |
| Map Reduce hì nhanh hơn những hệ thống truyền thống nhưng nó không tận dụng tối đa bộ nhớ của các hadoop cluster | Spark được chứng minh là có thể thực thi xử lí các công việc nhanh hơn gấp 10 cho đến 100 lần. |
| Map Reduce thực hiện hoàn toàn trên đĩa cứng, có độ trễ cao hơn. Không hỗ trợ caching dữ liệu. | Spark chắc chắn có độ trễ tính toán thấp hơn bằng cách caching các phần kết quả thông qua bộ nhớ ram của ô cứng phân tán. Lưu trữ dữ liệu trong bộ nhớ ram |
| MapReduce là lựa chọn rẻ hơn | Spark giá thành đắt hơn |
| Viết xử lí cho Map Reduce rất phức tạp và dài bởi lẽ nó thuần Java | Viết code Spark thì luôn dễ hơn và chúng ta có thể viết với 4 ngôn ngữ |
| Batch processing | Batch /Iteractive /Real Time /Interactive Processing |
| MapReduce không tích hợp cho những thứ khác như SQL, ML, RT | Spark được tích hợp sẵn cho SQL, ML, RT |
| Nó xử lí dữ liệu đơn giản | Hỗ trợ engine phân tích dữ liệu. Vì thế mà được lựa chọn rất nhiều trong ngành khoa học dữ liệu. |

Bảng 2.1 So sánh giữa Apache Hadoop và Apache Spark

Từ những ưu điểm trên của Spark mà trong bài nghiên cứu này hầu hết các thuật toán sẽ được sử dụng và tính toán trên spark.

**Resilient Distributed Datasets**

Ở cấp độ cao, mọi ứng dụng Spark bao gồm một chương trình trình điều khiển chạy chức năng chính của người dùng và thực hiện các hoạt động song song khác nhau trên một cụm. Tính trừu tượng chính mà Spark cung cấp là tập dữ liệu phân tán có khả năng phục hồi (RDD), là tập hợp các phần tử được phân vùng trên các nút của cụm có thể hoạt động song song. RDD được tạo bằng cách bắt đầu bằng một tệp trong hệ thống tệp Hadoop (hoặc bất kỳ hệ thống tệp nào khác được Hadoop hỗ trợ) hoặc một bộ sưu tập Scala hiện có trong chương trình trình điều khiển và chuyển đổi nó. Người dùng cũng có thể yêu cầu Spark duy trì một RDD trong bộ nhớ, cho phép nó được sử dụng lại một cách hiệu quả trong các hoạt động song song. Cuối cùng, các RDD tự động phục hồi sau các lỗi của nút.

Sự trừu tượng thứ hai trong Spark là các biến được chia sẻ có thể được sử dụng trong các hoạt động song song. Theo mặc định, khi Spark chạy một hàm song song dưới dạng một tập hợp các tác vụ trên các nút khác nhau, nó sẽ gửi một bản sao của từng biến được sử dụng trong hàm cho mỗi tác vụ. Đôi khi, một biến cần được chia sẻ giữa các tác vụ hoặc giữa các tác vụ và chương trình điều khiển. Spark hỗ trợ hai loại biến được chia sẻ: biến quảng bá, có thể được sử dụng để lưu trữ một giá trị trong bộ nhớ trên tất cả các nút và bộ tích lũy, là những biến chỉ được “thêm vào”, chẳng hạn như bộ đếm và tổng.

Hướng dẫn này hiển thị từng tính năng trong số các ngôn ngữ được hỗ trợ của Spark. Cách dễ dàng nhất để làm theo nếu bạn khởi chạy trình bao tương tác của Spark - bin / spark-shell cho Scala shell hoặc bin / pyspark cho Python.



Hình 2.8: Hoạt động của RDD trong Spark

## 2.4 Thuật toán LR

### 2.4.1 Giới thiệu thuật toán LR

Trong phân tích hồi quy, hồi quy logistic (hoặc hồi quy logit) là ước tính các tham số của mô hình logistic (các hệ số trong tổ hợp tuyến tính). Về mặt hình thức, trong hồi quy logistic nhị phân có một biến phụ thuộc nhị phân duy nhất, được mã hóa bởi một biến chỉ số, trong đó hai giá trị được gắn nhãn "0" và "1", trong khi các biến độc lập có thể là một biến nhị phân (hai lớp, được mã hóa bởi một biến chỉ số) hoặc một biến liên tục (bất kỳ giá trị thực nào).

### 2.4.2 Thuật toán phân lớp LR

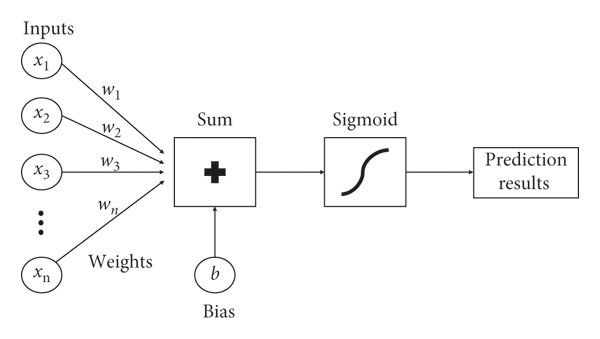
Logistic Regression là 1 thuật toán phân loại được dùng để gán các đối tượng cho 1 tập hợp giá trị rời rạc (như 0, 1, 2, ...). Một ví dụ điển hình là phân loại Email, gồm có email công việc, email gia đình, email spam, ... Giao dịch trực tuyến có là an toàn hay không an toàn, khối u lành tính hay ác tính. Thuật toán trên dùng hàm sigmoid logistic để đưa ra đánh giá theo xác suất

Hàm sigmoid

 (2-4)

Trong đó: z là hàm số của mô hình Logistic Regression.

### 2.4.3 Lưu đồ thuật toán LR



Hình 2.10 Lưu đồ thuật toán Logistic Regression

# CHƯƠNG 3 TRIỂN KHAI THỰC NGHIỆM

## 1. Về bộ dữ liệu

Bỗ dữ liệu thô được lấy ở trên kaggle. Dữ liệu có liên quan đến các chiến dịch tiếp thị trực tiếp của một tổ chức ngân hàng Bồ Đào Nha. Các chiến dịch tiếp thị dựa trên các cuộc gọi điện thoại. Thông thường, cần có nhiều liên hệ với cùng một khách hàng, để truy cập xem sản phẩm (tiền gửi có kỳ hạn ngân hàng) sẽ được ('có') hay không ('không') được đăng ký.

Thông tin thuộc tính:

Các biến đầu vào:

**Dữ liệu khách hàng của ngân hàng:**

* Age (số)
* Job: loại công việc (phân loại: 'quản trị viên.', ‘Blue-collar’, 'doanh nhân', 'người giúp việc', 'quản lý', 'nghỉ hưu', 'tự kinh doanh', 'dịch vụ', 'sinh viên' , 'kỹ thuật viên', 'thất nghiệp', 'không xác định')
* Marital: tình trạng hôn nhân (phân loại: 'đã ly hôn', 'đã kết hôn', 'độc thân', 'không rõ'; lưu ý: 'đã ly hôn' có nghĩa là đã ly hôn hoặc góa vợ)
* Education (phân loại: 'basic.4y', 'basic.6y', 'basic.9y', 'high.school', 'illiterate', 'professional.course', 'university.degree', 'chưa biết')
* default: có tín dụng trong tình trạng vỡ nợ? (phân loại: 'không', 'có', 'không xác định')
* house: có cho vay mua nhà không? (phân loại: 'không', 'có', 'không xác định')
* loan: có khoản vay cá nhân? (phân loại: 'không', 'có', 'không xác định')
* contact: kiểu liên lạc liên lạc (phân loại: 'cellular', 'phone')
* month: tháng liên hệ cuối cùng trong năm (phân loại: 'jan', 'feb', 'mar', ..., 'nov', 'dec')
* day\_of\_week: ngày liên hệ cuối cùng trong tuần (phân loại: 'mon', 'tue', 'wed', 'thu', 'fri')

Biến đầu ra (mục tiêu mong muốn):

* y - khách hàng đã đăng ký tiền gửi có kỳ hạn chưa? (nhị phân: 'yes', 'no')

## 2. Tiền xử lý dữ liệu bằng PySpark

Môi trường sử dụng: Google Colab / Anaconda Enviroment sử dụng Visual Studio và Terminal để code

Tải pyspark nếu chưa có pyspark. Sau đó khởi tạo SparkSession. Điểm vào tất cả các chức năng trong Spark là lớp SparkSession. Để tạo một SparkSession cơ bản, chỉ cần sử dụng SparkSession.builder

Đọc file dữ liệu csv bằng spark.read.csv()

!pip install pyspark

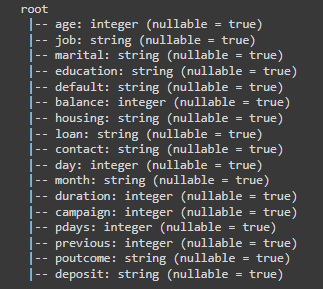
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('ml-bank').getOrCreate()

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

df.printSchema()

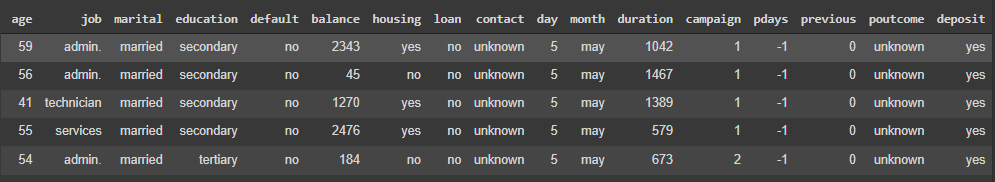
In ra xem tập dữ liệu và data type của từng field



Hình 3.1: Tập các trường dữ liệu

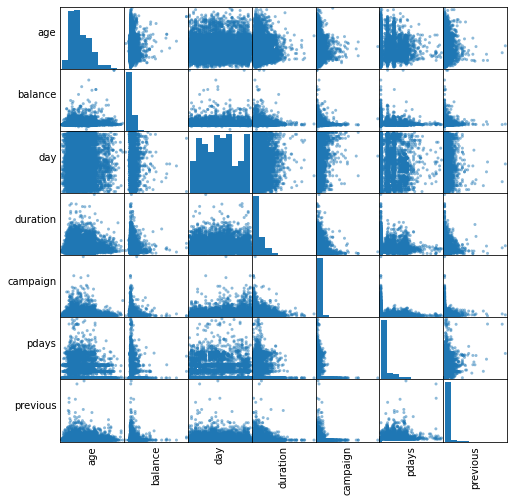
Sử dụng pandas để load ra 5 rows đầu tiên của bộ dữ liệu. Xem qua năm quan sát đầu tiên. Khung dữ liệu Pandas đẹp hơn Spark DataFrame.show ().

pd.DataFrame(df.take(5), columns=df.columns)



Hình 3.2: Top 5 rows trong tập dữ liệu

Mối tương quan giữa các biến độc lập.



Hình 3.3: Biểu đồ tương quan độc lập giữa các trường

Rõ ràng là không có các biến số tương quan cao. Do đó, chúng tôi sẽ giữ lại tất cả chúng cho mô hình. Tuy nhiên, cột ngày và tháng không thực sự hữu ích, chúng tôi sẽ loại bỏ hai cột này.

df = df.select('age', 'job', 'marital', 'education', 'default', 'balance', 'housing', 'loan', 'contact', 'duration', 'campaign', 'pdays', 'previous', 'poutcome', 'deposit')

cols = df.columns

df.printSchema()

Quá trình này bao gồm Category Indexing, One-Hot Encodingvà VectorAssembler – Transformer có tính năng hợp nhất nhiều cột thành một cột vectơ.

Nó lập chỉ mục từng cột phân loại bằng cách sử dụng StringIndexer, sau đó chuyển đổi các indexed categories thành các biến được mã one-hot encoded. Kết quả đầu ra có các vectơ nhị phân được nối vào cuối mỗi hàng. Chúng tôi sử dụng lại StringIndexer để mã hóa các nhãn của chúng tôi để gắn nhãn các chỉ số. Tiếp theo, chúng tôi sử dụng VectorAssembler để kết hợp tất cả các cột đối tượng thành một cột vectơ duy nhất.

categoricalColumns = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'poutcome']

stages = []

for categoricalCol in categoricalColumns:

    stringIndexer = StringIndexer(inputCol = categoricalCol, outputCol = categoricalCol + 'Index')

    encoder = OneHotEncoder(inputCols=[stringIndexer.getOutputCol()], outputCols=[categoricalCol + "classVec"])

    stages += [stringIndexer, encoder]

label\_stringIdx = StringIndexer(inputCol = 'deposit', outputCol = 'label')

stages += [label\_stringIdx]

numericCols = ['age', 'balance', 'duration', 'campaign', 'pdays', 'previous']

assemblerInputs = [c + "classVec" for c in categoricalColumns] + numericCols

assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")

stages += [assembler]

Chúng tôi sử dụng Pipeline để liên kết nhiều Transformer và Estimators lại với nhau để chỉ định quy trình công việc học máy của chúng tôi. Các giai đoạn của Pipeline được chỉ định dưới dạng một mảng có thứ tự.

pipeline = Pipeline(stages = stages)

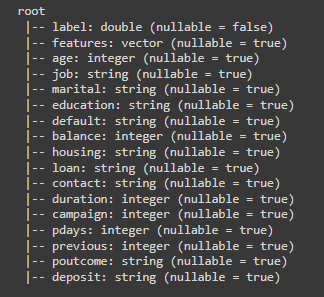
pipelineModel = pipeline.fit(df)

df = pipelineModel.transform(df)

selectedCols = ['label', 'features'] + cols

df = df.select(selectedCols)

df.printSchema()

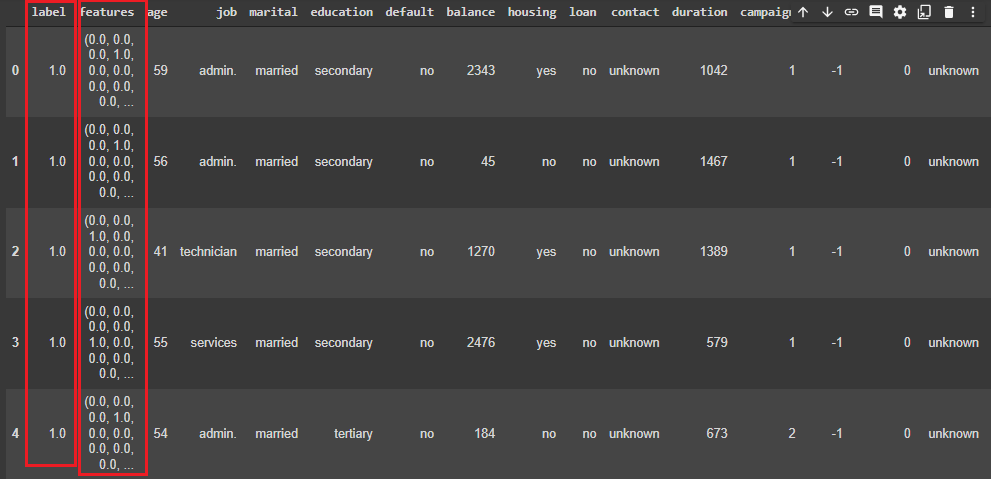


Hình 3.4: Dữ liệu sau khi làm sạch và có trường features và label

Như bạn có thể thấy, bây giờ chúng ta có cột tính năng và cột nhãn.

Tách ngẫu nhiên dữ liệu thành các tập huấn luyện và thử nghiệm.

pd.DataFrame(df.take(5), columns=df.columns)



Hình 3.5: Hiển thị dữ liệu 5 hàng đầu dưới dạng bảng

Sử dụng trường features và label để train model

train, test = df.randomSplit([0.7, 0.3], seed = 2018)

print("Training Dataset Count: " + str(train.count()))

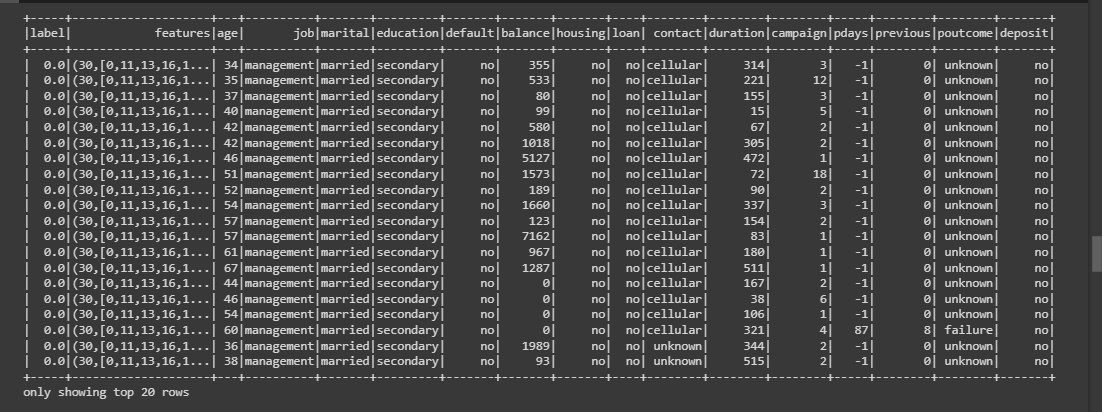
print("Test Dataset Count: " + str(test.count()))

Training Dataset Count: 7855

Test Dataset Count: 3307

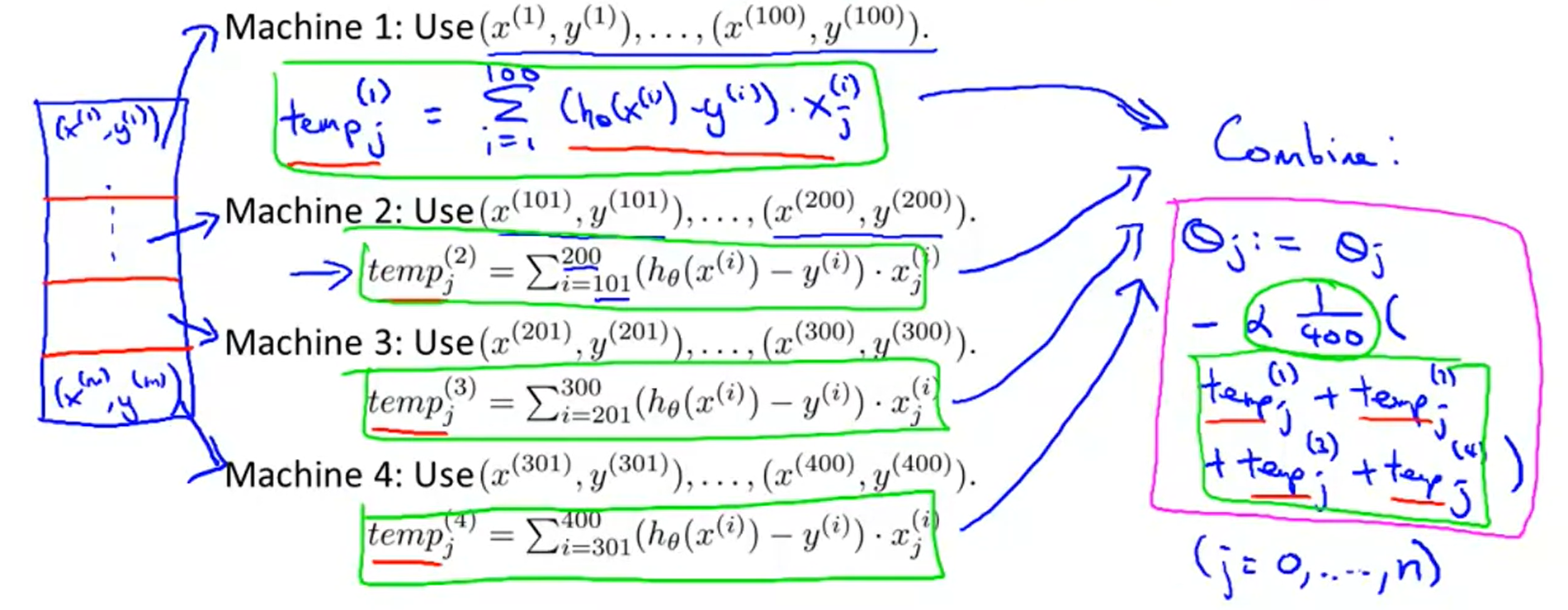
## 3.3 Huấn luyện mô hình

Dữ liệu để train model



Hình 3.6: Tập dữ liệu train 20 dòng đầu

### 3.3.1 LR



Hình 3.8: Tính toán phân tán mô hình LR

Train model

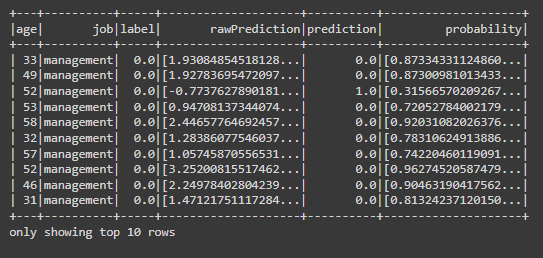
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label', maxIter=10)

lrModel = lr.fit(train)

Đưa ra dự đoán trên tập kiểm tra.

predictions = lrModel.transform(test)

predictions.select('age', 'job', 'label', 'rawPrediction', 'prediction', 'probability').show(10)



Hình 3.9: Dữ liệu được dự đoán bằng mô hình Logistic Regression

## 3.4 Đánh giá kết quả

## 3.4.1 LR

Evaluate our Logistic Regression model.

evaluator = BinaryClassificationEvaluator()

print('Test Area Under ROC', evaluator.evaluate(predictions))

Test Area Under ROC 0.885643189559481

# CHƯƠNG 4 KẾT LUẬN

# 

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |