**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

A logo of a university of transport technology

Description automatically generated

**BÁO CÁO BÀI TẬP TUẦN**

**BIG DATA**

**Tên đề tài:**

**Tìm hiểu Machine Learning with MLlib**

|  |  |
| --- | --- |
| **GIÁO VIÊN HƯỚNG DẪN:** | **Th.S ĐỖ BẢO SƠN** |
| **NHÓM SINH VIÊN THỰC HIỆN:** | **1. ĐẶNG NGỌC HIỆU**  **2. KIỀU MINH QUÂN**  **3. ĐỖ QUỐC TRƯỜNG**  **4. NGUYỄN HẢI DƯƠNG**  **5. BẠC ANH TUẤN** |
| **LỚP:** | **73DCTT23** |

**HÀ NỘI 03-2025**

**Bảng phân công nhiệm vụ**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| STT | Họ và Tên | Mã sinh viên | Nhiệm vụ | Ghi chú |
| 1 | Đặng Ngọc Hiệu | 73DCTT22349 | Tìm hiểu nội dung, DEMO |  |
| 2 | Kiều Minh Quân | 73DCTT22202 | Tìm hiểu nội dung, Word |  |
| 3 | Đỗ Quốc Trường | 73DCTT22120 | Tìm hiểu nội dung, Word |  |
| 4 | Nguyễn Hải Dương | 73DCTT22312 | Tìm hiểu nội dung, Powerpoint |  |
| 5 | Bạc Anh Tuấn | 73DCTT22257 | Tìm hiểu nội dung, DEMO |  |

**Mục Lục**

[**Mục lục ảnh** iii](#_Toc194961876)

[**Lời mở đầu** iv](#_Toc194961877)

[**Chương 1: Tổng quan về Apache spark** 1](#_Toc194961878)

[**1. Khái niệm:** 1](#_Toc194961879)

[**2. Các thành phần của Apache Spark:** 1](#_Toc194961880)

[**3. Kiến trúc của Apache Spark:** 2](#_Toc194961881)

[**Chương 2: Tổng quan về Machine Learning** 4](#_Toc194961882)

[**1. Giới thiệu về MLlib** 4](#_Toc194961883)

[**1.1. Khái quát chung về MLlib** 4](#_Toc194961884)

[**1.2. Lưu ý quan trọng về MLlib** 4](#_Toc194961885)

[**1.3. Giao diện MLlib qua các phiên bản Spark** 5](#_Toc194961886)

[**1.4. Các công cụ hỗ trợ cho các bài toán học máy phổ biến** 6](#_Toc194961887)

[**Chương 3: Các thuật toán trong MLlib** 7](#_Toc194961888)

[**2.1. Feature Extraction & Statistics** 7](#_Toc194961889)

[**2.2. Collaborative Filtering & Recommendation** 8](#_Toc194961892)

[**2.3. Thuật toán Alternating Least Squares** 9](#_Toc194961894)

[**2.4. Dimensionality Reduction** 10](#_Toc194961896)

[**2.5. Model Evaluation** 12](#_Toc194961898)

[**2.6. Tips and Performance Considerations** 12](#_Toc194961899)

[**2.6.1. Cấu hình thuật toán** 13](#_Toc194961900)

[**2.6.2. Tối ưu hóa bằng cách lưu trữ dữ liệu** 13](#_Toc194961901)

[**2.6.3. Nhận diện độ thưa của dữ liệu** 13](#_Toc194961902)

[**2.6.4. Điều chỉnh mức độ song song** 13](#_Toc194961903)

[**2.6.5. API Pipeline trong MLlib** 14](#_Toc194961904)

[**2.7. Other Algorithms** 16](#_Toc194961905)

[**Kết luận** 17](#_Toc194961906)

[**Tài liệu tham khảo:** 18](#_Toc194961907)

# **Mục lục ảnh**

[Hình 2.1: Demo File TF-IDF 4](#_Toc194087102)

[Hình 2.2: Demo Statistics 5](#_Toc194087103)

[Hình 2.3: Demo ColabFiltering\_recom 6](#_Toc194087104)

[Hình 2.4: Demo ModelEvalua-ALS\_RMSE 7](#_Toc194087105)

[Hình 2.5: Demo Dimensionality Reduction 9](#_Toc194087106)

# **Lời mở đầu**

Trong kỷ nguyên dữ liệu lớn (Big Data), nhu cầu khai thác và phân tích dữ liệu để đưa ra quyết định trở nên quan trọng hơn bao giờ hết. Các mô hình học máy (Machine Learning) không chỉ giúp tự động hóa quá trình phân tích mà còn có thể dự đoán xu hướng, nhận diện mẫu và tối ưu hóa hoạt động trong nhiều lĩnh vực khác nhau như tài chính, y tế, thương mại điện tử và khoa học dữ liệu. Tuy nhiên, một thách thức lớn khi triển khai học máy trên dữ liệu lớn là làm sao có thể xử lý lượng dữ liệu khổng lồ một cách nhanh chóng và hiệu quả.

MLlib, thư viện học máy của Apache Spark, được phát triển để giải quyết vấn đề này bằng cách cung cấp một bộ công cụ mạnh mẽ giúp xây dựng và triển khai các mô hình học máy trên quy mô lớn. MLlib tận dụng khả năng xử lý dữ liệu phân tán của Apache Spark, giúp tăng tốc quá trình huấn luyện mô hình so với các phương pháp truyền thống. Thư viện này hỗ trợ nhiều thuật toán phổ biến như hồi quy (regression), phân loại (classification), phân cụm (clustering), hệ thống gợi ý (recommendation systems) và nhiều công cụ tiền xử lý dữ liệu khác.

Bên cạnh đó, MLlib còn tích hợp chặt chẽ với Spark SQL, DataFrame API, và các công cụ xử lý dữ liệu khác, giúp các nhà khoa học dữ liệu dễ dàng thao tác với dữ liệu mà không cần viết quá nhiều mã phức tạp. Với khả năng mở rộng linh hoạt, MLlib không chỉ phù hợp cho các bài toán nhỏ mà còn có thể mở rộng để xử lý dữ liệu quy mô petabyte trên hệ thống phân tán. Nhờ đó, MLlib ngày càng trở thành một trong những công cụ quan trọng trong hệ sinh thái Apache Spark, được sử dụng rộng rãi trong các doanh nghiệp và tổ chức nghiên cứu trên toàn thế giới.

# **Chương 1: Tổng quan về Apache spark**

## **1. Khái niệm:**

Apache Spark là một framework mã nguồn mở tính toán cụm, được phát triển sơ khởi vào năm 2009 bởi AMPLab. Sau này, Spark đã được trao cho Apache Software Foundation vào năm 2013 và được phát triển cho đến nay.

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. Các thành phần của Apache Spark:**

Apache gồm có các thành phần chính sau đây:

* Spark Core:

Spark Core là thành phần cốt lõi của Apache Spark, các thành phần khác muốn hoạt động đều cần thông qua Spark Core.

Spark Core cung cấp các chức năng cơ bản để xử lý dữ liệu phân tán (distributed data processing - DDP) bao gồm quản lý bộ nhớ, thiết lập lịch tác vụ và khôi phục lỗi.

* Spark SQL:

Spark SQL cho phép người dùng truy vấn dữ liệu bằng ngôn ngữ SQL trên các tập dữ liệu lớn. Spark SQL cung cấp một cơ chế để tích hợp với các cơ sở dữ liệu quan hệ như MySQL, Oracle hoặc PostgreSQL và xử lý dữ liệu bằng cách sử dụng các tính năng phân tán của Spark.

Ngoài ra, Spark SQL còn tích hợp tốt với các công cụ ETL và Spark Streaming, điều đó giúp nó trở thành một phần quan trọng trong hệ thống phân tích dữ liệu phân tán và các ứng dụng trực tuyến trong thời gian thực.

* Spark Streaming:

Spark Streaming là một module cho phép xử lý dữ liệu trực tiếp trong thời gian thực (real-time). Spark Streaming cung cấp một cơ chế xử lý dữ liệu liên tục (stream processing) bằng cách chia nhỏ dữ liệu thành một chuỗi các microbatch nhỏ hơn và xử lý chúng thông qua API Apache Spark.

Bên cạnh đó, nó còn tích hợp với các module khác của Apache Spark như Spark SQL và MLlib. Spark Streaming được sử dụng rộng rãi cho các ứng dụng như phân tích dữ liệu trực tuyến, phát hiện sự cố hệ thống và phân tích dữ liệu giám sát trong thời gian thực.

* Spark MLlib (Machine Learning Library):

Spark MLlib là thư viện Machine Learning được tích hợp sẵn trong Apache Spark, cung cấp các thuật toán Machine Learning phổ biến giúp ích trong việc xử lý big data.

MLlib cung cấp các thuật toán Machine Learning phổ biến như: Regression, Classification, Clustering, Collaborative Filtering, Dimensionality Reduction, Feature Extraction and Transformation,...

Spark MLlib cũng tích hợp với các module khác của Apache Spark nhằm đáp ứng nhu cầu phân tích và xử lý dữ liệu lớn.

* GraphX

GraphX đi kèm với lựa chọn các thuật toán phân tán để xử lý cấu trúc đồ thị. Để thực hiện các tính toán trên đồ thị, GraphX cung cấp RDD, VertexRDD và EdgeRDD trong đó Vertex (đỉnh) và Edge (cạnh).

## **3. Kiến trúc của Apache Spark:**

Driver Program trong kiến ​​trúc Apache Spark sẽ quản lý, điều khiển quá trình xử lý trên cluster và tạo Spark Context. Spark Context sẽ bao gồm tất cả các chức năng cơ bản.

Spark Driver là một thành phần quan trọng trong kiến trúc của Apache Spark. Nó là chương trình chính của ứng dụng Spark, chạy trên một node trong cluster, quản lý và điều khiển quá trình xử lý trên cluster.

Spark Driver và Spark Context cùng giám sát quá trình thực thi cluster. Spark Driver hoạt động với Cluster Manage để quản lý nhiều công việc khác.

Cluster Manage giúp quản lý và phân phối tài nguyên trên các node của cluster, sau đó sẽ phân phối và giám sát tiến trình xử lý trên các node này để đảm bảo chúng hoạt động hiệu quả.

Bất cứ khi nào cấu trúc dữ liệu bất biến và phân tán (Resilient Distributed Datasets - RDD) được tạo trong Spark Context, thì nó sẽ được phân phối trên nhiều worker nodes và được cached ở đó.

Các worker nodes sẽ thực thi nhiệm vụ được giao bởi Cluster Manager và sau đó đưa nó trở lại Spark Context.

Executors sẽ lên tiến trình chạy trên các node khác trong cluster, được quản lý bởi driver program để thực hiện các nhiệm vụ xử lý dữ liệu. Mỗi Executor có thể chứa nhiều task (công việc) được giao để xử lý.

# **Chương 2: Tổng quan về Machine Learning**

## **1. Giới thiệu về MLlib**

MLlib (Machine Learning Library) là thư viện học máy của Apache Spark, cung cấp các công cụ và thuật toán mạnh mẽ để phân tích dữ liệu lớn. Thư viện này hỗ trợ nhiều thuật toán phổ biến trong học máy như phân loại, hồi quy, phân cụm, giảm chiều dữ liệu, và hệ thống gợi ý.

MLlib hoạt động tốt trên Spark và có thể xử lý dữ liệu trên nhiều nền tảng, bao gồm HDFS, Apache Cassandra, Apache HBase, và Amazon S3.

### **1.1. Khái quát chung về MLlib**

* MLlib được thiết kế đơn giản và linh hoạt, giúp người dùng triển khai các thuật toán học máy trên tập dữ liệu phân tán. Tất cả dữ liệu trong MLlib được biểu diễn dưới dạng RDDs (Resilient Distributed Datasets).
* MLlib chỉ giới thiệu một số kiểu dữ liệu đặc biệt như:
* Labeled Points (Điểm có nhãn)
* Vectors (Vector đặc trưng)
* Nhìn chung, MLlib là một tập hợp các hàm học máy có thể áp dụng trên RDDs.
* Quy trình sử dụng MLlib cho bài toán phân loại văn bản

Ví dụ: Xác định email spam

1. Bắt đầu với một RDD chứa chuỗi văn bản (danh sách email).
2. Trích xuất đặc trưng từ văn bản thành các số (vector hóa dữ liệu).
3. Áp dụng thuật toán học máy (ví dụ: hồi quy logistic) để tạo mô hình.
4. Đánh giá mô hình bằng tập dữ liệu kiểm tra (test dataset).

### **1.2. Lưu ý quan trọng về MLlib**

🔹 Chỉ hỗ trợ thuật toán phân tán:

* MLlib không bao gồm các thuật toán cổ điển không phù hợp với xử lý song song.
* Hỗ trợ các thuật toán hiện đại dành cho cụm máy tính, như:
  + Rừng ngẫu nhiên phân tán (Distributed Random Forests)
  + K-Means (Phiên bản cải tiến của K-Means truyền thống)
  + Alternating Least Squares (ALS) cho lọc cộng tác

🔹 Không nên dùng MLlib trong các trường hợp:

* Khi có nhiều tập dữ liệu nhỏ → Nên dùng thư viện Weka hoặc SciKit-Learn thay vì MLlib.
* Khi cần huấn luyện nhiều mô hình trên một tập dữ liệu nhỏ → Nên chạy SciKit-Learn trên từng node Spark (song song hóa với map()).
* Khi cần tinh chỉnh tham số (Hyperparameter Tuning) → Dùng parallelize() để huấn luyện nhiều mô hình với tham số khác nhau trên các node khác nhau.

### **1.3. Giao diện MLlib qua các phiên bản Spark**

|  |  |
| --- | --- |
| **Phiên bản Spark** | **Tính năng của Mllib** |
| Spark 1.0 - 1.1 | API cấp thấp, cung cấp các hàm cơ bản để gọi thuật toán. |
| Spark 1.2 | Thêm Pipeline API giúp tự động hóa quy trình học máy, giống SciKit-Learn. |

Bảng 1.1: Các phiên bản Spark

Pipeline API giúp xây dựng luồng làm việc tự động:

* Chia tập dữ liệu thành tập huấn luyện và kiểm tra.
* Tìm tham số tối ưu cho mô hình.
* Dễ dàng xây dựng pipeline học máy hoàn chỉnh.

Tóm lại: MLlib tối ưu cho xử lý dữ liệu lớn trên cụm máy tính, nhưng không phù hợp với bài toán nhỏ hoặc yêu cầu tinh chỉnh tham số chi tiết.

Cách cài đặt và triển khai

Yêu cầu hệ thống để sử dụng Mllib

* Cần thư viện đại số tuyến tính
* Cần thư viện runtime gfortran cho hệ điều hành của bạn.
* Nếu thiếu gfortran, hãy làm theo hướng dẫn trên trang web MLlib để cài đặt.
* Cần NumPy nếu sử dụng MLlib trong Python
* Nếu Python chưa có numpy (import numpy bị lỗi), hãy cài đặt bằng cách:
  + Linux: Dùng python-numpy hoặc numpy từ trình quản lý gói.
  + Windows/macOS: Dùng Anaconda hoặc cài bằng pip install numpy.
* Các thuật toán trong MLlib thay đổi theo phiên bản
* Các thuật toán được đề cập trong tài liệu này có mặt từ Spark 1.2 trở đi.
* Một số thuật toán có thể không tồn tại trong phiên bản Spark cũ hơn.

Tóm lại: Cần đảm bảo hệ thống có gfortran và numpy trước khi sử dụng MLlib. Ngoài ra, kiểm tra phiên bản Spark để tránh thiếu thuật toán.

### **1.4. Các công cụ hỗ trợ cho các bài toán học máy phổ biến**

 • Thuật toán học máy: Hỗ trợ các thuật toán như hồi quy (Regression), phân loại (Classification), phân cụm (Clustering), và hệ thống gợi ý (Recommendation).

 • Xử lý dữ liệu: Hỗ trợ chuẩn hóa dữ liệu, trích xuất đặc trưng (Feature Extraction), lựa chọn đặc trưng (Feature Selection), và biến đổi đặc trưng (Feature Transformation).

 • Pipeline học máy: Cung cấp các công cụ để xây dựng quy trình học máy từ tiền xử lý đến huấn luyện mô hình.

 • Đánh giá mô hình: Các chỉ số đánh giá hiệu suất mô hình như RMSE, MAE, Accuracy, Precision, Recall, v.v.

 • Hỗ trợ lưu và tải mô hình: Cho phép lưu trữ và tải lại các mô hình đã được huấn luyện để sử dụng sau này.

# **Chương 3: Các thuật toán trong MLlib**

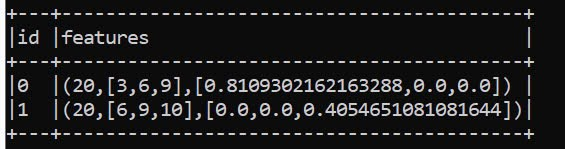
MLlib cung cấp nhiều thuật toán học máy khác nhau. Trong phạm vi tài liệu này, chúng ta sẽ tập trung vào các nhóm chính:

## **2.1. Feature Extraction & Statistics**

**a) Feature Extraction**

Feature Extraction (Trích xuất đặc trưng) là quá trình chuyển đổi dữ liệu thô thành dạng số để mô hình học máy có thể sử dụng. Trong MLlib (Spark ML), các thuật toán trích xuất đặc trưng giúp chuyển đổi văn bản, hình ảnh, hoặc dữ liệu phi cấu trúc khác thành các vector số.

* TF-IDF (Term Frequency - Inverse Document Frequency):Biểu diễn văn bản dưới dạng vector dựa trên tần suất từ. Phân loại văn bản, tìm kiếm thông tin.
* Word2Vec:Biểu diễn từ dưới dạng vector dựa trên ngữ cảnh. Xử lý ngôn ngữ tự nhiên (NLP).
* CountVectorizer:Đếm tần suất từ trong tài liệu và biểu diễn dưới dạng vector. Phân tích văn bản, phát hiện chủ đề.
* HashingTF:Ánh xạ văn bản thành vector sử dụng hàm băm. Tiết kiệm bộ nhớ so với CountVectorizer.
* PCA (Principal Component Analysis):Giảm số chiều của dữ liệu bằng cách giữ lại các thành phần quan trọng nhất. Giảm nhiễu, tăng hiệu suất tính toán.
* Rformula:Tự động chuyển đổi cột danh mục và số thành vector đặc trưng. Dễ dàng chuẩn bị dữ liệu cho mô hình.

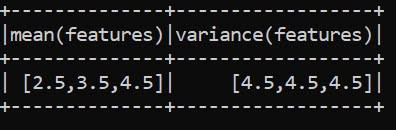
**

Hình 2.1: Demo File TF-IDF

**b) Statistics**

MLlib hỗ trợ các công cụ thống kê để phân tích dữ liệu:

* Summarizer: Tính toán các số liệu thống kê như trung bình, phương sai, độ lệch chuẩn, min, max.
* Correlation (Tương quan): Đánh giá mối quan hệ giữa các biến (Pearson, Spearman).
* Hypothesis Testing (Kiểm định giả thuyết): Sử dụng kiểm định t-test, Chi-square test để phân tích dữ liệu.
* Kernel Density Estimation (KDE): Ước lượng phân phối xác suất của dữ liệu.



Hình 2.2: Demo Statistics

## **2.2. Collaborative Filtering & Recommendation**

Lọc cộng tác (Collaborative Filtering) là một kỹ thuật trong hệ thống gợi ý, trong đó đánh giá và tương tác của người dùng với các sản phẩm được sử dụng để đề xuất các sản phẩm mới.

Ưu điểm của lọc cộng tác là nó chỉ cần một danh sách các tương tác giữa người dùng và sản phẩm, bao gồm:

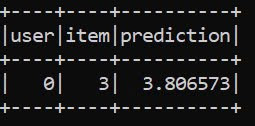
* Tương tác tường minh (explicit interactions): Đánh giá rõ ràng của người dùng (ví dụ: xếp hạng sản phẩm trên một trang mua sắm).
* Tương tác ẩn (implicit interactions): Các hành vi gián tiếp (ví dụ: một người dùng đã truy cập một trang sản phẩm nhưng không để lại đánh giá).

Dựa vào các tương tác này, thuật toán lọc cộng tác có thể học được:

* Sự tương đồng giữa các sản phẩm (vì cùng một nhóm người dùng có thể tương tác với chúng).
* Sự tương đồng giữa người dùng (vì họ có thể thích hoặc mua các sản phẩm giống nhau).
* Đưa ra các gợi ý mới dựa trên sở thích chung của người dùng.

Ngoài việc gợi ý sản phẩm, lọc cộng tác còn có thể áp dụng trong các tình huống khác như:

* Gợi ý người dùng để theo dõi trên mạng xã hội.
* Gợi ý các thẻ (tags) để gắn vào bài viết.
* Gợi ý bài hát để thêm vào danh sách phát.



Hình 2.3: Demo ColabFiltering\_recom

## **2.3. Thuật toán Alternating Least Squares**

* ALS (Alternating Least Squares) là một thuật toán phổ biến để thực hiện lọc cộng tác và có khả năng mở rộng tốt trên các cụm máy tính. Thuật toán này được triển khai trong lớp mllib.recommendation.ALS của MLlib.
* ALS hoạt động bằng cách tính toán một vector đặc trưng cho mỗi người dùng và mỗi sản phẩm sao cho tích vô hướng (dot product) giữa vector của người dùng và vector của sản phẩm gần với điểm đánh giá của họ.

Các tham số chính của ALS

* rank: Kích thước của vector đặc trưng (mặc định: 10). Rank lớn hơn có thể giúp mô hình tốt hơn nhưng tốn tài nguyên tính toán hơn.
* iterations: Số vòng lặp huấn luyện (mặc định: 10).
* lambda: Tham số điều chuẩn để tránh overfitting (mặc định: 0.01).
* alpha: Hệ số được sử dụng để tính toán mức độ tin cậy trong mô hình ALS với phản hồi ẩn (mặc định: 1.0).
* numUserBlocks, numProductBlocks: Số khối để chia dữ liệu người dùng và sản phẩm nhằm tối ưu tính song song. Nếu đặt -1, MLlib sẽ tự động xác định giá trị này.

Cách sử dụng ALS

* ALS yêu cầu một RDD chứa các đối tượng Rating(userID, productID, rating).
* ID phải là số nguyên 32-bit. Nếu ID là chuỗi hoặc số lớn hơn, có thể chuyển đổi bằng cách sử dụng hash code. Việc hai ID khác nhau ánh xạ vào cùng một hash code là rất hiếm nên vẫn đảm bảo chất lượng gợi ý.
* Ngoài ra, có thể sử dụng broadcast() để tạo bảng ánh xạ từ ID sản phẩm về số nguyên để đảm bảo mỗi sản phẩm có một ID duy nhất.



Hình 2.4: Demo ModelEvalua-ALS\_RMSE

## **2.4. Dimensionality Reduction**

Giảm chiều giúp tối ưu hóa hiệu suất mô hình và loại bỏ nhiễu:

* PCA (Principal Component Analysis): Biến đổi dữ liệu thành không gian có ít chiều hơn nhưng vẫn giữ được thông tin quan trọng.

Phân tích thành phần chính (PCA). Khi làm việc với một bộ dữ liệu gồm các điểm trong không gian chiều cao, chúng ta thường quan tâm đến việc giảm chiều dữ liệu của các điểm này để có thể phân tích dễ dàng hơn. Ví dụ, chúng ta có thể muốn vẽ các điểm trong không gian 2 chiều, hoặc chỉ giảm số lượng đặc trưng để huấn luyện mô hình hiệu quả hơn.

Kỹ thuật chính để giảm chiều dữ liệu được cộng đồng học máy sử dụng là phân tích thành phần chính (PCA). Trong kỹ thuật này, việc ánh xạ vào không gian chiều thấp được thực hiện sao cho phương sai của dữ liệu trong biểu diễn chiều thấp được tối đa hóa, từ đó loại bỏ các chiều không có thông tin quan trọng.

Để tính toán ánh xạ, ma trận tương quan chuẩn hóa của dữ liệu được xây dựng và các vectơ riêng và giá trị riêng của ma trận này được sử dụng. Các vectơ riêng tương ứng với giá trị riêng lớn nhất được sử dụng để tái tạo một phần lớn phương sai của dữ liệu gốc.

PCA hiện chỉ có sẵn trong Java và Scala (tính đến MLlib 1.2). Để sử dụng PCA, trước tiên bạn phải đại diện cho ma trận của mình bằng lớp mlib.linalg.distributed.RowMatrix, lớp này lưu trữ một RDD các vectors, mỗi vector đại diện cho một dòng.

Ví dụ: PCA trong Scala

import org.apache.spark.mllib.linalg.Matrix

import org.apache.spark.mllib.linalg.distributed.RowMatrix

val points: RDD[Vector] = // ...

val mat: RowMatrix = new RowMatrix(points)

val pc: Matrix = mat.computePrincipalComponents(2)

// Dự đoán điểm vào không gian chiều thấp

val projected = mat.multiply(pc).rows

// Huấn luyện mô hình k-means trên dữ liệu 2 chiều

val model = KMeans.train(projected, 10)

Đánh giá :Trong ví dụ này, RDD projected chứa một phiên bản 2 chiều của RDD points gốc và có thể được sử dụng để vẽ đồ thị hoặc thực hiện các thuật toán MLlib khác, như phân cụm qua K-means.

Lưu ý rằng computePrincipalComponents() trả về một đối tượng mllib.linalg.Matrix, là một lớp tiện ích đại diện cho ma trận dày, tương tự như Vector. Bạn có thể truy cập dữ liệu bên trong bằng cách sử dụng phương thức toArray

* Singular Value Decomposition (SVD): Phân rã ma trận để trích xuất đặc trưng quan trọng.

Phân tích giá trị ký dị (SVD) MLlib cũng cung cấp phương thức phân tích giá trị kỳ dị (SVD) cấp thấp. SVD phân tích một ma trận m × n thành ba ma trận A≈UΣVTA \approx U \Sigma V

TA≈UΣVT, trong đó:

* U là ma trận trực giao, các cột của nó được gọi là vectơ kỳ dị trái.
* Σ là ma trận đường chéo với các giá trị không âm trên đường chéo, các giá trị này được gọi là giá trị kỳ dị.
* V là ma trận trực giao, các cột của nó được gọi là vectơ kỳ dị phải.

Đối với các ma trận lớn, thường ta không cần phân tích đầy đủ mà chỉ cần các giá trị kỳ dị lớn nhất và các vectơ kỳ dị liên quan. Điều này giúp tiết kiệm bộ nhớ, giảm nhiễu và phục hồi cấu trúc bậc thấp của ma trận. Nếu chỉ giữ lại k giá trị kỳ dị lớn nhất, các ma trận kết quả sẽ có kích thước U:m×kU: m \times kU:m×k, Σ:k×k\Sigma: k \times kΣ:k×k, và V:n×kV: n \times kV:n×k.

Ví dụ :SVD trong Scala

// Tính toán 20 giá trị kỳ dị lớn nhất của một ma trận RowMatrix và các vectơ kỳ dị của chúng.

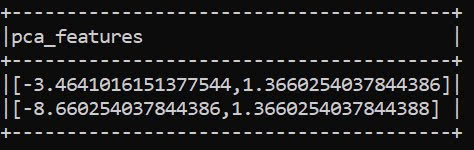
val svd: SingularValueDecomposition[RowMatrix, Matrix] =

    mat.computeSVD(20, computeU=true)

val U: RowMatrix = svd.U // U là một RowMatrix phân tán.

val s: Vector = svd.s // Giá trị kỳ dị là một vector dày cục bộ.

val V: Matrix = svd.V // V là một ma trận dày cục bộ.



Hình 2.5: Demo Dimensionality Reduction

## **2.5. Model Evaluation**

Đánh giá mô hình là một bước quan trọng trong quy trình học máy, vì nó giúp xác định hiệu quả của mô hình với bộ dữ liệu kiểm tra (khác với bộ dữ liệu huấn luyện). Quá trình đánh giá này giúp phát hiện vấn đề overfitting, khi mô hình quá phù hợp với dữ liệu huấn luyện nhưng lại không hoạt động tốt trên dữ liệu chưa thấy.

MLlib của Spark (phiên bản 1.2) cung cấp các chức năng đánh giá mô hình trong Java và Scala thông qua các lớp như BinaryClassificationMetrics và MulticlassMetrics. Những lớp này cho phép tính toán các chỉ số như độ chính xác (precision), độ nhạy (recall), và diện tích dưới đường cong ROC, thông qua một RDD của các cặp (dự đoán, giá trị thật). Để đánh giá mô hình chính xác, cần sử dụng bộ dữ liệu kiểm tra không được sử dụng trong quá trình huấn luyện.

Trong các phiên bản tương lai của Spark, API pipeline sẽ cung cấp chức năng đánh giá mô hình cho tất cả các ngôn ngữ, giúp người dùng có thể dễ dàng xây dựng pipeline, chọn lựa mô hình và tham số tốt nhất thông qua phương pháp xác thực chéo (cross-validation)

MLlib hỗ trợ các phương pháp đánh giá chất lượng mô hình:

* ROC, AUC: Đánh giá mô hình phân loại.
* RMSE, MAE: Đánh giá mô hình hồi quy.
* Cross-validation: Kiểm tra độ ổn định của mô hình bằng cách chia nhỏ dữ liệu để huấn luyện và kiểm tra.

## **2.6. Tips and Performance Considerations**

Chuẩn bị dữ liệu đặc trưng (Feature Engineering)

Mặc dù thuật toán đóng vai trò quan trọng trong học máy, nhưng chất lượng của dữ liệu đầu vào (đặc trưng - features) quyết định phần lớn kết quả.

* Thêm nhiều đặc trưng có ý nghĩa hơn bằng cách kết hợp dữ liệu từ nhiều nguồn khác nhau.
* Chuyển đổi dữ liệu đặc trưng thành dạng vector phù hợp, ví dụ:
  + Chuẩn hóa giá trị đặc trưng bằng StandardScaler để các đặc trưng có trọng số đồng đều.
  + Xử lý văn bản đúng cách bằng cách sử dụng thư viện như NLTK để rút gọn từ (stemming) và tính toán TF-IDF để xác định độ quan trọng của từ trong một tập dữ liệu lớn.
  + Mã hóa nhãn (labels) đúng định dạng: MLlib yêu cầu nhãn lớp phải nằm trong khoảng 0 đến C-1 (với C là số lượng lớp).

### **2.6.1. Cấu hình thuật toán**

* Thêm điều chuẩn (regularization) để giảm hiện tượng overfitting.
* Các thuật toán dựa trên Stochastic Gradient Descent (SGD) cần khoảng 100 vòng lặp (iterations) để đạt kết quả tốt.
* ALS mặc định có rank = 10, nhưng có thể tăng lên để cải thiện độ chính xác.
* Luôn kiểm tra hiệu quả trên tập dữ liệu kiểm thử (test data).

### **2.6.2. Tối ưu hóa bằng cách lưu trữ dữ liệu**

* Hầu hết các thuật toán của MLlib lặp lại dữ liệu nhiều lần → Nên cache() RDD trước khi truyền vào MLlib.
* Trong Python, MLlib tự động cache RDD nên không cần làm thủ công.
* Trong Scala và Java, cần cache thủ công nếu dữ liệu được tái sử dụng.

### **2.6.3. Nhận diện độ thưa của dữ liệu**

* Nếu vector đặc trưng chứa nhiều giá trị 0, nên sử dụng biểu diễn sparse để tiết kiệm bộ nhớ và tăng tốc xử lý.
* MLlib sử dụng biểu diễn sparse hiệu quả hơn nếu ít nhất ⅔ giá trị của vector là 0.
* Nếu chỉ khoảng 10% giá trị trong vector là khác 0, thì sử dụng sparse vector giúp tăng tốc tính toán.

→ Nếu việc dùng sparse vector giúp cache dữ liệu trong bộ nhớ thay vì lưu trên ổ đĩa, thì nên chuyển sang sparse representation ngay cả khi dữ liệu không quá thưa.

### **2.6.4. Điều chỉnh mức độ song song**

* Nên có ít nhất số phân vùng (partitions) trong RDD bằng số lõi CPU trên cụm máy để tận dụng tối đa hiệu suất.
* Mặc định, Spark chia dữ liệu theo kích thước khối (block size) ~64MB.
* Cảnh báo: Không nên tăng partitions quá mức vì có thể làm tăng chi phí giao tiếp giữa các nút trong cụm máy.

### **2.6.5. API Pipeline trong MLlib**

* Từ Spark 1.2, MLlib cung cấp API Pipeline giúp đơn giản hóa quá trình học máy, tương tự pipeline của SciKit-Learn.
* Pipeline là một chuỗi các bước gồm:
  + Biến đổi dữ liệu (Feature Transformation).
  + Huấn luyện mô hình (Model Fitting).
* Các thành phần chính của Pipeline API:
  + Tokenizer: Tách từ trong văn bản.
  + HashingTF: Chuyển đổi từ thành vector đặc trưng.
  + LogisticRegression: Thuật toán phân loại.
  + CrossValidator: Tìm kiếm tham số tốt nhất bằng cross-validation.
  + ParamGridBuilder: Xây dựng lưới tham số để thử nghiệm nhiều giá trị khác nhau.
  + BinaryClassificationEvaluator: Đánh giá mô hình dựa trên metric phân loại nhị phân.

Ví dụ: Pipeline API cho phân loại thư rác (Spam Classification)

import org.apache.spark.sql.SQLContext

import org.apache.spark.ml.Pipeline

import org.apache.spark.ml.classification.LogisticRegression

import org.apache.spark.ml.feature.{HashingTF, Tokenizer}

import org.apache.spark.ml.tuning.{CrossValidator, ParamGridBuilder}

import org.apache.spark.ml.evaluation.BinaryClassificationEvaluator

// Lớp biểu diễn tài liệu với ID, nội dung văn bản và nhãn

case class LabeledDocument(id: Long, text: String, label: Double)

// Tạo SQL Context

val sqlContext = new SQLContext(sc)

import sqlContext.\_

// Định nghĩa các bước trong pipeline

val tokenizer = new Tokenizer()

  .setInputCol("text")

  .setOutputCol("words")

val tf = new HashingTF()

  .setNumFeatures(10000)

  .setInputCol(tokenizer.getOutputCol)

  .setOutputCol("features")

val lr = new LogisticRegression() // Mặc định inputCol là "features"

// Tạo pipeline gồm các bước: Tokenizer → HashingTF → LogisticRegression

val pipeline = new Pipeline().setStages(Array(tokenizer, tf, lr))

// Huấn luyện mô hình

val model = pipeline.fit(documents)

// Dùng Grid Search và Cross-Validation để tìm tham số tối ưu

val paramMaps = new ParamGridBuilder()

  .addGrid(tf.numFeatures, Array(10000, 20000))

  .addGrid(lr.maxIter, Array(100, 200))

  .build()

val eval = new BinaryClassificationEvaluator()

val cv = new CrossValidator()

  .setEstimator(lr)

  .setEstimatorParamMaps(paramMaps)

  .setEvaluator(eval)

// Huấn luyện mô hình với Cross Validation

val bestModel = cv.fit(documents)

Các lợi ích của Pipeline API:

* Đơn giản hóa quá trình huấn luyện bằng cách tổ chức các bước trong một workflow thống nhất.
* Dễ dàng điều chỉnh tham số bằng Grid Search + Cross-Validation để tìm mô hình tốt nhất.
* Dùng SchemaRDDs từ Spark SQL, tương tự DataFrames trong R.
* Có thể mở rộng với nhiều bước tiền xử lý khác nhau.

## **2.7. Other Algorithms**

Ngoài các nhóm trên, MLlib còn hỗ trợ nhiều thuật toán khác như:

* Clustering (Phân cụm): K-Means, Gaussian Mixture Model (GMM).
* Classification (Phân loại): Logistic Regression, Decision Trees, Random Forest.
* Regression (Hồi quy): Linear Regression, Generalized Linear Models.

# 

# **Kết luận**

Trong bối cảnh dữ liệu lớn (Big Data) ngày càng phát triển, nhu cầu về các công cụ mạnh mẽ để xử lý và phân tích dữ liệu một cách nhanh chóng, chính xác trở nên cấp thiết. MLlib, thư viện học máy của Apache Spark, đã chứng tỏ vai trò quan trọng trong việc giúp các nhà khoa học dữ liệu và kỹ sư máy học triển khai các mô hình học máy trên quy mô lớn một cách hiệu quả. Với khả năng tận dụng sức mạnh tính toán phân tán của Spark, MLlib không chỉ giúp tối ưu hóa quá trình huấn luyện mô hình mà còn hỗ trợ xử lý dữ liệu lớn với tốc độ nhanh hơn nhiều so với các phương pháp truyền thống.

Một trong những điểm mạnh của MLlib là sự đa dạng của các thuật toán được tích hợp sẵn, bao gồm hồi quy (regression), phân loại (classification), phân cụm (clustering), hệ thống gợi ý (recommendation systems) và nhiều phương pháp tiền xử lý dữ liệu khác. Điều này giúp người dùng dễ dàng xây dựng các mô hình phức tạp mà không cần phải triển khai từ đầu. Bên cạnh đó, sự tương thích với các công cụ như Spark SQL, DataFrame API, Streaming, và khả năng tích hợp với các hệ thống lưu trữ dữ liệu lớn như HDFS, Cassandra, Hive càng giúp MLlib trở thành một lựa chọn lý tưởng cho các tổ chức và doanh nghiệp muốn tận dụng dữ liệu của mình để đưa ra các quyết định chiến lược.

Tuy nhiên, với sự phát triển không ngừng của công nghệ, MLlib đang dần chuyển sang Spark ML (DataFrame-based API) để tối ưu hóa việc xử lý dữ liệu và mô hình học máy. Thay vì làm việc với RDD (Resilient Distributed Dataset) như trước đây, Spark ML sử dụng DataFrame, giúp giảm bộ nhớ tiêu thụ và tăng tốc độ tính toán đáng kể. Do đó, những ai muốn ứng dụng MLlib vào thực tế nên cân nhắc sử dụng Spark ML để tận dụng những cải tiến mới nhất.

Tóm lại, MLlib đã và đang đóng vai trò quan trọng trong hệ sinh thái Apache Spark, giúp đơn giản hóa việc triển khai các mô hình học máy trên dữ liệu lớn. Dù có một số hạn chế, nhưng với khả năng mở rộng mạnh mẽ, tích hợp linh hoạt và hiệu suất cao, MLlib vẫn là một lựa chọn hàng đầu cho những ai muốn khai thác tiềm năng của học máy trong thời đại dữ liệu. Trong tương lai, với sự phát triển của AI, Machine Learning và các mô hình tiên tiến hơn, MLlib sẽ tiếp tục được cải tiến để đáp ứng nhu cầu ngày càng tăng của cộng đồng khoa học dữ liệu và doanh nghiệp trên toàn cầu.

# **Tài liệu tham khảo:**

Learning Spark Lightning-Fast Data Analysis (Holden Karau, Andy Konwinski, Patrick Wendell, Matei Zaharia)

<https://viblo.asia/p/tong-quan-ve-apache-spark-cho-he-thong-big-data-RQqKLxR6K7z>

<https://viblo.asia/p/machine-learning-trich-xuat-dac-trung-van-ban-part-1-oOVlYqzzl8W>

<https://spark.apache.org/mllib/?utm_source=chatgpt.com>