Nguyễn Anh Kiệt - 51703119

1. MACHINE LEARNING IN PYSPARK
   1. Khái niệm về machine learning library

MLlib là một triển khai Spark của một số chức năng máy học (ML) phổ biến, cũng như các thử nghiệm liên quan và trình tạo dữ liệu. MLlib hiện hỗ trợ bốn kiểu cài đặt vấn đề máy học phổ biến, đó là phân loại nhị phân, hồi quy, phân cụm và lọc cộng tác, cũng như nguyên thủy tối ưu hóa gradient cơ bản. Hướng dẫn này sẽ phác thảo các chức năng được hỗ trợ trong MLlib và cũng cung cấp một ví dụ về cách gọi MLlib.

* 1. Dependencies

MLlib sử dụng thư viện đại số tuyến tính [jblas](https://github.com/mikiobraun/jblas) , bản thân nó phụ thuộc vào các quy trình Fortran bản địa. Bạn có thể cần cài đặt [thư viện thời gian chạy gfortran](https://github.com/mikiobraun/jblas/wiki/Missing-Libraries) nếu nó chưa có trên các nút của bạn. MLlib sẽ gây ra lỗi liên kết nếu nó không thể tự động phát hiện các thư viện này.

Để sử dụng MLlib trong Python, bạn sẽ cần [NumPy](http://www.numpy.org/) phiên bản 1.7 hoặc mới hơn và Python 2.7.

* 1. Binary Classification (phân loại nhị phân)

Phân loại nhị phân là một vấn đề học tập có giám sát, trong đó chúng tôi muốn phân loại các thực thể thành một trong hai danh mục hoặc nhãn riêng biệt, ví dụ: dự đoán liệu email có phải là thư rác hay không. Vấn đề này liên quan đến việc thực hiện một Thuật toán học tập trên một tập hợp các ví dụ được gắn nhãn , tức là một tập hợp các thực thể được biểu diễn thông qua các tính năng (số) cùng với các nhãn danh mục cơ bản. Thuật toán trả về một Mô hình được đào tạo có thể dự đoán nhãn cho các thực thể mới mà nhãn bên dưới chưa được biết đến.

MLlib hiện hỗ trợ hai họ mô hình tiêu chuẩn để phân loại nhị phân, đó là [Máy vectơ hỗ trợ tuyến tính (SVM)](http://en.wikipedia.org/wiki/Support_vector_machine) và [Hồi quy logistic](http://en.wikipedia.org/wiki/Logistic_regression) , cùng với các biến thể [chính quy L1 và L2](http://en.wikipedia.org/wiki/Regularization_(mathematics)) của mỗi họ mô hình. Tất cả các thuật toán đào tạo đều tận dụng một nguyên thủy gradient cơ bản (được mô tả [bên dưới](https://spark.apache.org/docs/0.9.0/mllib-guide.html#gradient-descent-primitive) ) và lấy đầu vào là một tham số chính quy ( regParam ) cùng với các tham số khác nhau liên quan đến gradient descent ( stepSize , numIterations , miniBatchFraction ).

Các thuật toán có sẵn để phân loại nhị phân:

* [SVMWithSGD](https://spark.apache.org/docs/0.9.0/api/mllib/index.html#org.apache.spark.mllib.classification.SVMWithSGD)
* [LogisticRegressionWithSGD](https://spark.apache.org/docs/0.9.0/api/mllib/index.html#org.apache.spark.mllib.classification.LogisticRegressionWithSGD)
  1. Linear Regression(Hồi quy tuyến tính)

Hồi quy tuyến tính là một cài đặt học tập có giám sát cổ điển khác. Trong bài toán này, mỗi thực thể được liên kết với một nhãn có giá trị thực (trái ngược với nhãn nhị phân như trong phân loại nhị phân) và chúng tôi muốn dự đoán các nhãn càng chặt chẽ càng tốt với các đặc trưng số đại diện cho các thực thể. MLlib hỗ trợ hồi quy tuyến tính cũng như các biến thể chính quy L1 ( [lasso](http://en.wikipedia.org/wiki/Lasso_(statistics)#Lasso_method) ) và L2 ( [ridge](http://en.wikipedia.org/wiki/Ridge_regression) ). Các thuật toán hồi quy trong MLlib cũng tận dụng nguyên thủy gradient gốc [bên dưới](https://spark.apache.org/docs/0.9.0/mllib-guide.html#gradient-descent-primitive) (được mô tả [bên dưới](https://spark.apache.org/docs/0.9.0/mllib-guide.html#gradient-descent-primitive) ), và có các tham số tương tự như các thuật toán phân loại nhị phân được mô tả ở trên.

Các thuật toán có sẵn cho hồi quy tuyến tính:

* [LinearRegressionWithSGD](https://spark.apache.org/docs/0.9.0/api/mllib/index.html#org.apache.spark.mllib.regression.LinearRegressionWithSGD)
* [RidgeRegressionWithSGD](https://spark.apache.org/docs/0.9.0/api/mllib/index.html#org.apache.spark.mllib.regression.RidgeRegressionWithSGD)
* [LassoWithSGD](https://spark.apache.org/docs/0.9.0/api/mllib/index.html#org.apache.spark.mllib.regression.LassoWithSGD)
  1. Clustering (phân cụm)

Phân cụm là một vấn đề học tập không có giám sát, theo đó chúng ta nhằm mục đích nhóm các tập hợp con của các thực thể với nhau dựa trên một số khái niệm về sự giống nhau. Phân cụm thường được sử dụng để phân tích khám phá và / hoặc như một thành phần của đường dẫn học tập có giám sát phân cấp (trong đó các bộ phân loại hoặc mô hình hồi quy riêng biệt được đào tạo cho mỗi cụm). MLlib hỗ trợ phân cụm [k-](http://en.wikipedia.org/wiki/K-means_clustering) mean , một trong những thuật toán phân cụm được sử dụng phổ biến nhất để phân cụm các điểm dữ liệu thành số lượng cụm được xác định trước. Việc triển khai MLlib bao gồm một biến thể song song của phương thức [k-](http://en.wikipedia.org/wiki/K-means%2B%2B)[mean](http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf)[++](http://en.wikipedia.org/wiki/K-means%2B%2B) được gọi là [kmeans ||](http://theory.stanford.edu/~sergei/papers/vldb12-kmpar.pdf). Việc triển khai trong MLlib có các tham số sau:

* k là số cụm mong muốn.
* maxIterations là số lần lặp lại tối đa để chạy.
* InitializationMode chỉ định khởi tạo ngẫu nhiên hoặc khởi tạo thông qua k-means ||.
* số lần chạy là số lần chạy thuật toán k-mean (k-mean không được đảm bảo để tìm ra giải pháp tối ưu toàn cục và khi chạy nhiều lần trên một tập dữ liệu nhất định, thuật toán trả về kết quả phân cụm tốt nhất).
* InitializiationSteps xác định số bước trong k-mean || thuật toán.
* epsilon xác định ngưỡng khoảng cách mà chúng tôi coi là k-method đã hội tụ trong đó.

Các thuật toán có sẵn để phân cụm:

* [KMeans](https://spark.apache.org/docs/0.9.0/api/mllib/index.html#org.apache.spark.mllib.clustering.KMeans)
  1. Collaborative Filtering (lọc cộng tác)

[Lọc cộng tác](http://en.wikipedia.org/wiki/Recommender_system#Collaborative_filtering) thường được sử dụng cho các hệ thống khuyến nghị. Các kỹ thuật này nhằm mục đích điền vào các mục còn thiếu của ma trận liên kết mục người dùng. MLlib hiện hỗ trợ lọc cộng tác dựa trên mô hình, trong đó người dùng và sản phẩm được mô tả bằng một tập hợp nhỏ các yếu tố tiềm ẩn có thể được sử dụng để dự đoán các mục nhập bị thiếu. Đặc biệt, chúng tôi triển khai thuật toán [bình phương nhỏ nhất (ALS) xen kẽ](http://www2.research.att.com/~volinsky/papers/ieeecomputer.pdf) để tìm hiểu các yếu tố tiềm ẩn này. Việc triển khai trong MLlib có các tham số sau:

* numBlocks là số lượng màu đen được sử dụng để tính toán song song (đặt thành -1 để tự động cấu hình).
* thứ hạng là số lượng các yếu tố tiềm ẩn trong mô hình của chúng tôi.
* số lần lặp là số lần lặp lại để chạy.
* lambda chỉ định tham số chính quy hóa trong ALS.
* implicitPrefs chỉ định xem nên sử dụng biến thể ALS phản hồi rõ ràng hay một biến thể được điều chỉnh cho dữ liệu phản hồi ngầm hiểu
* alpha là một tham số áp dụng cho biến thể phản hồi ngầm của ALS điều chỉnh độ tin cậy cơ sở trong các quan sát ưu tiên
  1. Explicit vs Implicit Feedback

Phương pháp tiêu chuẩn để lọc cộng tác dựa trên phân tích nhân tử của ma trận coi các mục nhập trong ma trận mục người dùng là các tùy chọn rõ ràng do người dùng cung cấp cho mục đó.

Thông thường trong nhiều trường hợp sử dụng trong thế giới thực là chỉ có quyền truy cập vào phản hồi ngầm (ví dụ: lượt xem, lượt nhấp, lượt mua, lượt thích, lượt chia sẻ, v.v.). Cách tiếp cận được sử dụng trong MLlib để xử lý dữ liệu như vậy được lấy từ [Lọc cộng tác để tìm tập dữ liệu phản hồi ngầm](http://www2.research.att.com/~yifanhu/PUB/cf.pdf) . Về cơ bản thay vì cố gắng lập mô hình trực tiếp ma trận xếp hạng, cách tiếp cận này coi dữ liệu là sự kết hợp của các tùy chọn nhị phân và giá trị tin cậy . Sau đó, xếp hạng có liên quan đến mức độ tin cậy trong các tùy chọn của người dùng được quan sát, thay vì xếp hạng rõ ràng được đưa ra cho các mặt hàng. Sau đó, mô hình cố gắng tìm các yếu tố tiềm ẩn có thể được sử dụng để dự đoán sở thích mong đợi của người dùng đối với một mặt hàng.

Các thuật toán có sẵn để lọc cộng tác:

* ALS
  1. Gradient Descent Primitive

[Gradient descent](http://en.wikipedia.org/wiki/Gradient_descent) (cùng với các biến thể ngẫu nhiên của chúng) là các phương pháp tối ưu hóa bậc nhất rất phù hợp cho tính toán quy mô lớn và phân tán. Phương pháp gradient descent nhằm mục đích tìm điểm cực tiểu cục bộ của một hàm bằng cách thực hiện lặp đi lặp lại các bước theo hướng gradient âm của hàm tại điểm hiện tại, tức là giá trị tham số hiện tại. Gradient descent được bao gồm như một nguyên thủy cấp thấp trong MLlib, trên đó các thuật toán ML khác nhau được phát triển và có các tham số sau:

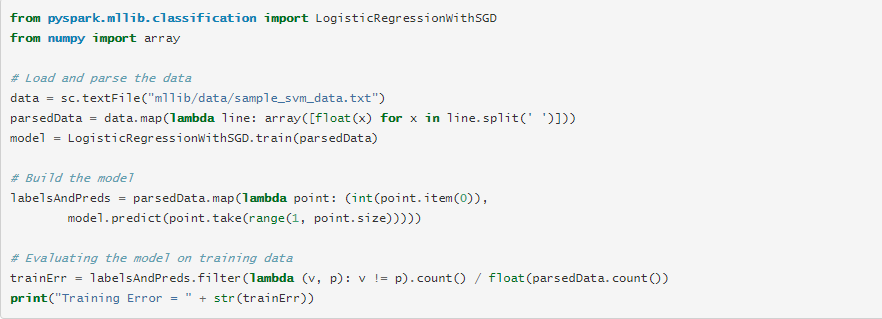
* gradient là một lớp tính toán gradient ngẫu nhiên của hàm đang được tối ưu hóa, tức là, đối với một ví dụ huấn luyện duy nhất, tại giá trị tham số hiện tại. MLlib bao gồm các lớp gradient cho các hàm mất mát thông thường, ví dụ: bản lề, logistic, bình phương nhỏ nhất. Lớp gradient lấy đầu vào làm ví dụ huấn luyện, nhãn của nó và giá trị tham số hiện tại.
* updater là một lớp cập nhật các trọng số trong mỗi lần lặp lại của gradient descent. MLlib bao gồm các bộ cập nhật cho các trường hợp không có chính quy, cũng như các bộ điều chỉnh L1 và L2.
* stepSize là một giá trị vô hướng biểu thị kích thước bước ban đầu cho gradient xuống. Tất cả các trình cập nhật trong MLlib sử dụng kích thước bước ở bước thứ t bằng stepSize / sqrt (t).
* numIterations là số lần lặp lại để chạy.
* regParam là tham số chính quy khi sử dụng chính quy L1 hoặc L2.
* miniBatchFraction là phần dữ liệu được sử dụng để tính gradient ở mỗi lần lặp.

Các thuật toán có sẵn cho gradient descent:

* [Gradient descent](http://en.wikipedia.org/wiki/Gradient_descent)
  1. Sử dụng ML bằng python

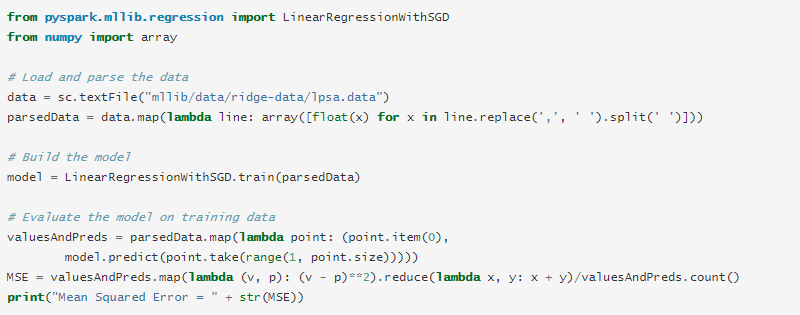
Binary Classification

Ví dụ sau đây cho thấy cách tải tập dữ liệu mẫu, xây dựng mô hình hồi quy logistic và đưa ra dự đoán với mô hình kết quả để tính toán lỗi huấn luyện.



Linear Regression

Ví dụ sau minh họa cách tải dữ liệu đào tạo, phân tích cú pháp nó dưới dạng RDD của LabeledPoint. Sau đó, ví dụ này sử dụng LinearRegressionWithSGD để xây dựng một mô hình tuyến tính đơn giản để dự đoán các giá trị nhãn. Chúng tôi tính toán sai số trung bình bình phương ở cuối để đánh giá mức [độ phù hợp](http://en.wikipedia.org/wiki/Goodness_of_fit)



Clustering

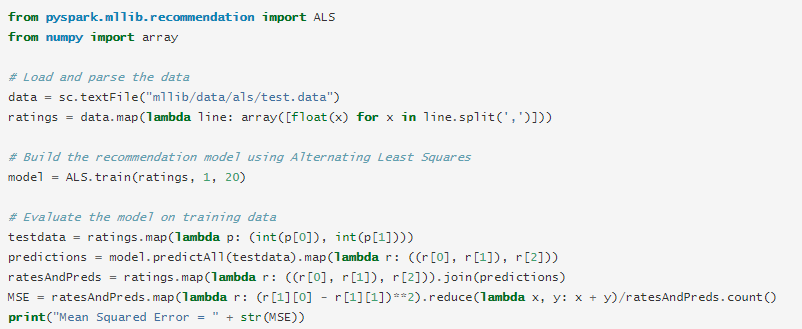
Trong ví dụ sau sau khi tải và phân tích dữ liệu, chúng tôi sử dụng đối tượng KMeans để phân cụm dữ liệu thành hai cụm. Số lượng các cụm mong muốn được chuyển đến thuật toán. Sau đó, chúng tôi tính toán Trong Tập hợp Tổng của Lỗi Bình phương (WSSSE). Bạn có thể giảm số đo sai số này bằng cách tăng k . Trong thực tế, k tối ưu thường là một trong đó có "khuỷu tay(elbow)" trong đồ thị WSSSE.



Tương tự, bạn có thể sử dụng RidgeRegressionWithSGD và LassoWithSGD và so sánh các lỗi trung bình bình phương khi làm.

Collaborative Filtering

Trong ví dụ sau, chúng tôi tải dữ liệu xếp hạng. Mỗi hàng bao gồm một người dùng, một sản phẩm và một xếp hạng. Chúng tôi sử dụng phương thức ALS.train () mặc định giả định xếp hạng là rõ ràng. Chúng tôi đánh giá đề xuất bằng cách đo Sai số trung bình bình phương của dự đoán xếp hạng.

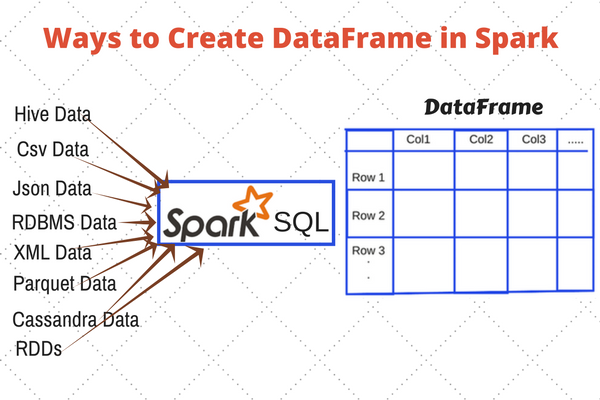


Nếu ma trận xếp hạng bắt nguồn từ nguồn thông tin khác (tức là nó được suy ra từ các tín hiệu khác), bạn có thể sử dụng phương pháp trainImplicit để có được kết quả tốt hơn.



1. DATAFRAME

Tập dữ liệu là một tập hợp dữ liệu phân tán. Dataset là một giao diện mới được thêm vào trong Spark 1.6 cung cấp các lợi ích của RDD (đánh máy mạnh, khả năng sử dụng các hàm lambda mạnh mẽ) với các lợi ích của công cụ thực thi được tối ưu hóa của Spark SQL. Một Dataset có thể được [xây dựng](https://spark.apache.org/docs/latest/sql-getting-started.html#creating-datasets) từ vật JVM và sau đó thao tác sử dụng biến đổi chức năng ( map, flatMap, filter, vv). API tập dữ liệu có sẵn trong [Scala](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html) và [Java](https://spark.apache.org/docs/latest/api/java/index.html?org/apache/spark/sql/Dataset.html) . Python không hỗ trợ Dataset API. Nhưng do bản chất động của Python, nhiều lợi ích của Dataset API đã có sẵn (tức là bạn có thể truy cập trường của một hàng theo tên một cách tự nhiên row.columnName). Trường hợp của R cũng tương tự.



DataFrame là một Tập dữ liệu được tổ chức thành các cột được đặt tên. Về mặt khái niệm, nó tương đương với một bảng trong cơ sở dữ liệu quan hệ hoặc một khung dữ liệu trong R / Python, nhưng với các tối ưu hóa phong phú hơn. DataFrames có thể được xây dựng từ nhiều [nguồn](https://spark.apache.org/docs/latest/sql-data-sources.html) như: tệp dữ liệu có cấu trúc, bảng trong Hive, cơ sở dữ liệu bên ngoài hoặc RDD hiện có. Các DataFrame API có sẵn trong Scala, Java, [Python](https://spark.apache.org/docs/latest/api/python/pyspark.sql.html#pyspark.sql.DataFrame) , và [R](https://spark.apache.org/docs/latest/api/R/index.html) . Trong Scala và Java, DataFrame được đại diện bởi một Dataset gồm Rows. Trong [API Scala](https://spark.apache.org/docs/latest/api/scala/org/apache/spark/sql/Dataset.html) , DataFramechỉ đơn giản là một loại bí danh của Dataset[Row]. Trong khi, trong [Java API](https://spark.apache.org/docs/latest/api/java/index.html?org/apache/spark/sql/Dataset.html) , người dùng cần sử dụng Dataset<Row>để biểu diễn a DataFrame.

Trong toàn bộ tài liệu này, chúng tôi thường đề cập đến Tập dữ liệu Scala / Java của Rows là DataFrames.