This is the forth article of the "Big Data Processing with Apache Spark" series. Please see also: Part 1: Introduction, Part 2: Spark SQL, Part 3: Spark Streaming, Part 5: Spark ML Data Pipelines, Part 6: Graph Data Analytics with Spark GraphX.

Machine learning, predictive analytics, and data science topics are getting a lot of attention in recent years for solving real world problems in different business domains in several organizations. Spark MLlib, Spark's Machine Learning library, includes several different machine learning algorithms for Collaborative Filtering, Clustering, Classification and other machine learning tasks.

In the previous articles in "Big Data Processing with Apache Spark" series, we have looked at what Apache Spark framework is (Part 1), how to leverage the SQL interface to access data using Spark SQL library (Part 2) and real-time data processing & analytics of streaming data using Spark Streaming (Part 3).

In this article, we'll discuss machine learning concepts and how to use Apache Spark MLlib library for running predictive analytics. We will use a sample application to illustrate the powerful API Spark provides in machine learning area.

Spark Machine Learning API includes two packages called `spark.mllib` and `spark.ml`.

The spark.mllib package contains the original Spark machine learning API built on Resilient Distributed Datasets (RDDs). It offers machine learning techniques which include correlation, classification and regression, collaborative filtering, clustering, and dimensionality reduction.

On the other hand, spark.ml package provides machine learning API built on the DataFrames which are becoming the core part of Spark SQL library. This package can be used for developing and managing the machine learning pipelines. It also provides Feature Extractors, Transformers, Selectors, and machine learning techniques like classification and regression, and clustering.

We'll focus on Spark MLlib package in this article and discuss various machine learning techniques this package brings to the table. In the next article, we'll cover the Spark ML package and look at how to create and manage data pipelines.

Machine Learning & Data Science

Machine Learning is about learning from existing data to make predictions about the future. It's based on creating models from input data sets for data-driven decision making.

Data science is the discipline of extracting the knowledge from large data sets (structured or unstructured) to provide insights to business teams and influence the business strategies and roadmaps. The role of Data Scientist is more critical than ever in solving the problems that are not easy to solve using the traditional numerical methods.

There are different types of machine learning models like:

- Supervised learning
- Unsupervised learning
- Semi-supervised Learning
- Reinforcement learning

Let’s briefly look at each of these machine learning models and how they compare with each other.

Supervised learning: This technique is used to predict an outcome by training the program using an existing set of training data (labeled data). Then we use the program to predict the label for a new unlabeled data set.

There are two sub-models under supervised machine learning, Regression and Classification.

Unsupervised learning: This is used to find hidden patterns and correlations within the raw data. No training data used in this model, so this technique is based on unlabeled data.

Algorithms like k-means and Principle Component Analysis (PCA) fall into this category.

Semi-supervised Learning: This technique uses both supervised and unsupervised learning models for predictive analytics. It uses labeled and unlabeled data sets for training. It typically involves using a small amount of labeled data with a large amount of unlabeled data. It can be used for machine learning methods like classification and regression.

Reinforcement learning: The Reinforcement Learning technique is used to learn how to maximize a numerical reward goal by trying different actions and discovering which actions result in the maximum reward.

Following table shows some examples where these machine learning models can be used to solve the problems.

ML Model	Examples
Supervised learning	Fraud detection
Unsupervised learning	Social network applications, language prediction
Semi-supervised Learning	Image categorization, Voice recognition
Reinforcement learning	Artificial Intelligence (AI) applications

Table 1. Machine Learning Models and Real-world Examples

Machine Learning Algorithms

There are several algorithms to help with machine learning solutions. Let’s look at some of the algorithms supported by machine learning frameworks.

Naive Bayes: Naive Bayes is a supervised learning algorithm used for classification. It's based on applying Bayes theorem and a set of conditional independence assumptions.

k-means Clustering: k-means algorithm creates k groups from a set of objects so that the members of a group are more similar.

Support vector machines: Support vector machines (SVMs) is a supervised learning algorithm used to find the boundary that separates classes by as wide a margin as possible. Given a set of training examples, each marked for belonging to one of two categories, an SVM training algorithm builds a model that assigns new examples into one category or the other. Applications of SVM include bioinformatics, text, and image recognition.

Decision Trees: Decision trees are used in many types of machine learning problems including multi-class classification. MLlib supports both basic decision tree algorithm and ensembles of trees. Two ensemble algorithms are available, Gradient-Boosted Trees and Random Forests.

As documented on this website, here is a summary of all the machine learning styles, problems, and methods to solve them, shown in Table 2 below.

ML Model	Problems	Algorithms
Supervised Learning	Classification, Regression, Anomaly Detection	Logistic Regression, Back Propagation Neural Network
Unsupervised Learning	Clustering, Dimensionality reduction	k-Means , Apriori algorithm
Semi-Supervised Learning	Classification, Regression	Self training, Semi-supervised Support Vector Machines (S3VMs)

Table 2. Machine learning styles, problems, and methods

Steps in a Machine Learning Program

When working machine learning projects, other tasks like data preparation, cleansing and analysis are also very important tasks in addition to the actual learning models and algorithms used to solve the business problems.

Following are the steps performed in a typical machine learning program.

- Featurization
- Training
- Model Evaluation

Figure 1 below shows the process flow of a typical machine learning solution.

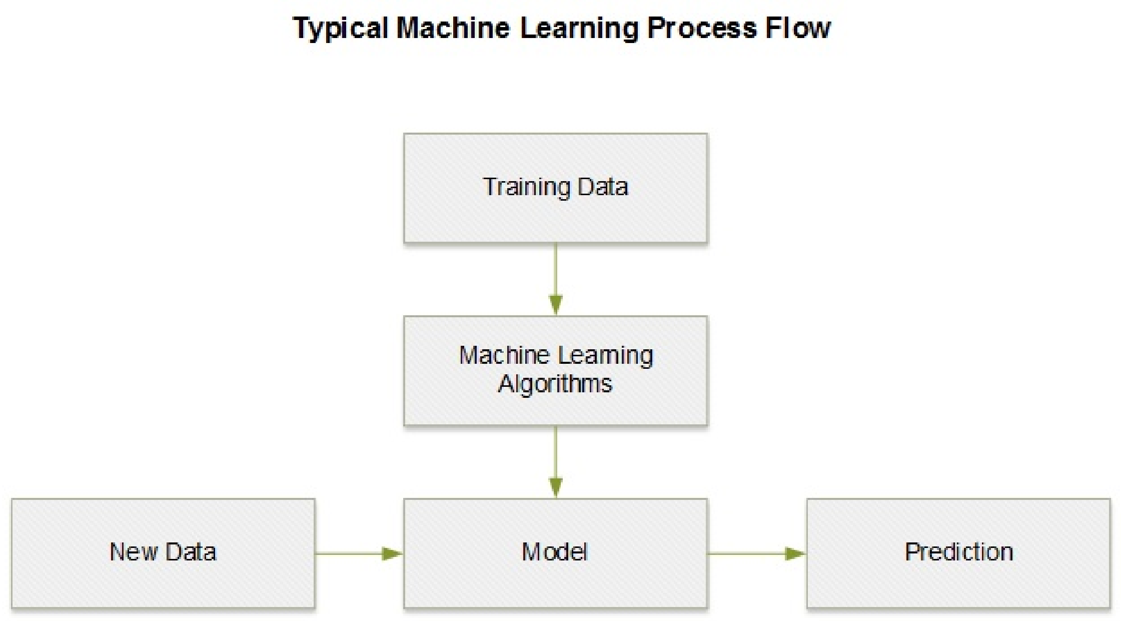

Typical Machine Learning Process Flow
Training Data
Machine Learning Algorithms
New Data
Model
Prediction

Figure 1. Machine Learning Process Flow

It's important to know that if the raw data isn't cleaned or prepared before running machine learning algorithms, the resulting patterns will not be accurate or useful as well as some anomalies may be missed.

The quality of training data we provide to machine learning programs also plays a critical role in the prediction results. If the training data is not random enough, resulting patterns won't be accurate. And if the data is too small, the machine learning program may give inaccurate predictions.

Use Cases

The business use cases for machine learning span different domains and scenarios including recommendation engines (food recommendation engine), predictive analytics (stock price prediction or predicting flights delay), targeted advertising, fraud detection, image and video recognition, self-driving cars and various other artificial intelligence.

Let's look at two popular machine learning applications, recommendation engine and fraud detection, in more detail.

Recommendation Engines

Recommendation engines use the attributes of an item or a user or the behavior of a user or their peers, to make the predictions. There are different factors that drive an effective recommendation engine model. Some of these factors are list below:

- Peer based
- Customer behavior
- Corporate deals or offers
- Item clustering
- Market/Store factors

Recommendation engine solutions are implemented by leveraging two algorithms, content-based filtering and collaborative filtering.

Content-based filtering: This is based on how similar a particular item is to other items based on usage and ratings. The model uses the content attributes of items (such as categories, tags, descriptions and other data) to generate a matrix of each item to other items and calculates similarity based on the ratings provided. Then the most similar items are listed together with a similarity score. Items with the highest score are most similar.

Movie recommendation is a good example of this model. It recommends that "Users who liked a particular movie liked these other movies as well".

These models don’t take into account the overall behavior of other users, so they don't provide personalized recommendations compared to other models like collaborative filtering.

Collaborative Filtering: On the other hand, collaborative filtering model is based on making predictions to find a specific item or user based on similarity with other items or users. The filter applies weights based on the "peer user" preferences. The assumption is users who display similar profile or behavior have similar preferences for items.

An example of this model is the recommendations on ecommerce websites like Amazon. When you search for an item on the website you would see something like "Customers Who Bought This Item Also Bought."

Items with the highest recommendation score are the most relevant to the user in context.

Collaborative filtering based solutions perform better compared to other models. Spark MLlib implements a collaborative filtering algorithm called Alternating Least Squares (ALS). There are two variations of the entries in collaborative filtering, called explicit and implicit feedback. Explicit feedback is based on the direct preferences given by the user to the item (like a movie). Explicit feedback is nice, but many times it's skewed because users who strongly like or dislike a product provide reviews on it. We don't get the opinion of many people towards the center of the bell shaped curve of data points.

Implicit feedback examples are user's views, clicks, likes etc. Implicit feedback data is used a lot in the industry for predictive analytics because of the ease to gather this type of data.

There are also model based methods for recommendation engines. These often incorporate methods from collaborative and content-based filtering. Model-based approach gets the best of both worlds, the power and performance of collaborative filtering and the flexibility and adaptability of content-based filtering. Deep learning techniques are good examples of this model.

You can also integrate other algorithms like K-Means into the recommendation engine solution to get more refined predictions. K-Means algorithm works by partitioning "n" observations into "k" clusters in which each observation belongs to the cluster with the nearest mean. Using K-Means technique, we can find similar items or users based on their attributes.

Figure 2 below shows different components of a recommendation engine, user factors, other factors like market data and different algorithms.

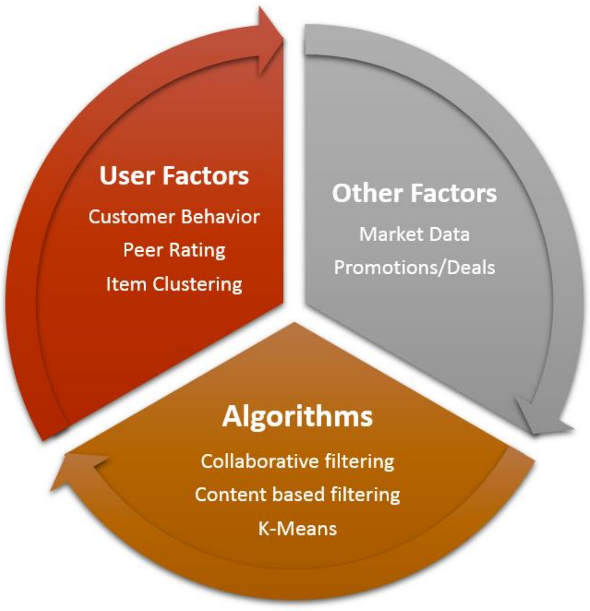


Figure 2. Components of a Recommendation Engine

Fraud Detection

Fraud detection is another important use case of using machine learning techniques because it addresses a critical problem in financial industry quickly and accurately. Financial services organizations only have few hundred milliseconds to determine if a particular online transaction is legitimate or a fraud.

Neural network techniques are used for point-of-sale (POS) fraud detection use cases. Organizations like PayPal use different types of machine learning algorithms for risk managementlike linear, neural network, and deep learning.

Spark MLlib library provides several algorithms for solving this use case, including linear SVMs, logistic regression, decision trees, and naive Bayes. In addition, ensemble models (which combine the predictions of a set of models) such as random forests or gradient-boosting trees are also available.

When you are working on a machine learning project in your organization, as recommended in this article, you can follow the steps listed below to implement machine learning solutions in your own projects.

- Select the programming language
- Select the appropriate algorithm or algorithms
- Select problem
- Research the algorithms
- Unit test all the functions in the ML solution

Now, let’s look at how Apache Spark framework implements the machine learning algorithms.

Spark MLlib

MLlib is Spark’s machine learning (ML) library. Its goal is to make practical machine learning scalable and easy. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs.

Like we learned earlier, there are two different ways of using Spark Machine Learning API, Spark MLlib and Spark ML.

In addition to various algorithms Spark MLlib supports, there are also data processing functions and data analytics utilities and tools available in this library.

- Frequent itemset mining via FP-growth and association rules
- Sequential pattern mining via PrefixSpan
- Summary statistics and hypothesis testing
- Feature transformations
- Model evaluation and hyper-parameter tuning

Figure 3 below shows Apache Spark framework with Spark MLlib library.

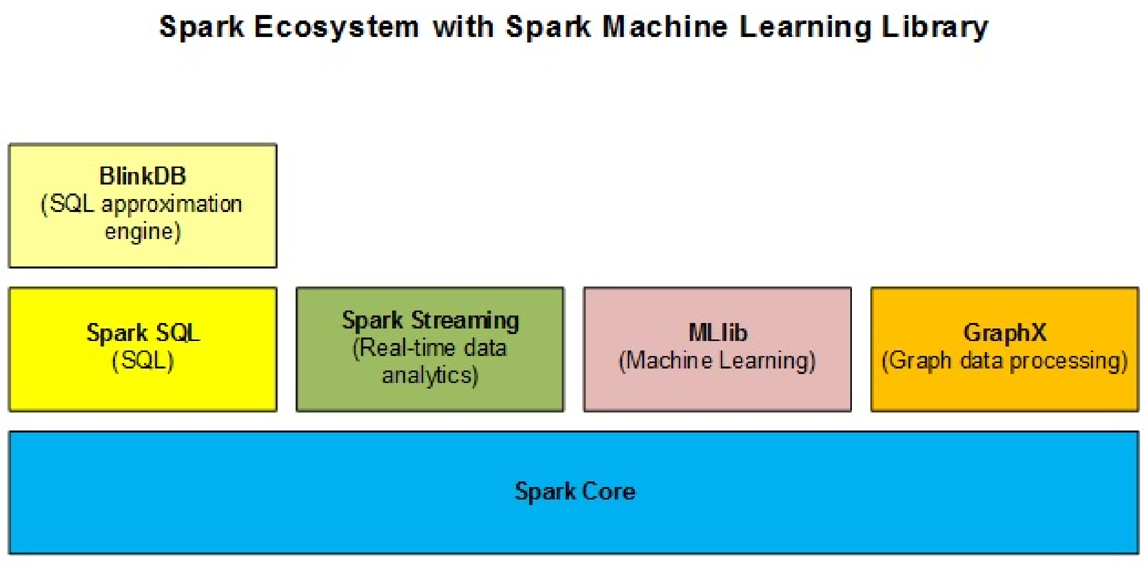

Spark Ecosystem with Spark Machine Learning Library
BlinkDB (SQL approximation engine)
Spark SQL (SQL)
Spark Streaming (Real-time data analytics)
MLlib (Machine Learning)
GraphX (Graph data processing)
Spark Core

Figure 3. Spark Ecosystem with Spark Machine Learning Library

Spark MLlib API is available in Scala, Java, and Python programming languages. Here are the links to API in each of these languages.

- Spark MLlib Scala API
- Java API
- Python API

Sample Application

We'll develop a sample Machine Learning application using classification technique, specifically collaborative filtering method, to predict the movies to recommend to a user based on other users' ratings on different movies.

Our recommendation engine solution will use Alternating Least Squares (ALS) machine learning algorithm.

Even though the data sets used in the code example in this article are not very large in size and complexity, Spark MLlib can be used for any type of real world problems that are complex in nature and deal with data containing several dimensions, and very complex predictor functions.

Remember machine learning can be used to solve problems that cannot be solved by numerical means alone.

To keep the machine learning application simple so we can focus on Spark MLlib API, we'll follow the Movie Recommendations example discussed in Spark Summit workshop. This exercise is a very good resource to learn more about Spark MLlib library. For more details on the application, visit their website.

Use Case

The use case we want to implement Spark based machine learning solution is a recommendation engine.

Recommendation engines are used to make predictions for unknown user-item associations (e.g. movie ratings) based on known user-item associations. They can make predictions based on based on user's affinity to other items and other users' affinity to this specific item. The engine builds a prediction model based on the known data (called Training Data) and then make predictions for unknown user-item associations (called Test Data).

Our program includes the following steps to arrive at the top movie recommendations for the user.

- Load movies data file
- Load the data file with ratings provided by a specific user (you)
- Load the ratings data provided by other users (community)
- Join the user ratings data with community ratings into a single RDD
- Train the model using ALS algorithm using ratings data
- Identify the movies not rated by a particular user (userId = 1)
- Predict the ratings of the items not rated by user
- Get top N recommendations (N=5 in our example)
- Display the recommendation data on the console

If you want to process or run further analysis on output data, you can store the results in a NoSQL database like Cassandra or MongoDB.

Data Sets

We will use the movie datasets provided by MovieLens group. There are few different data files we need for the sample application. These datasets are available for download from GroupLens website. We will use one of the latest datasets (smaller version with 100K ratings). Download the dataset zip file from the website.

Following table shows the different datasets used in the application.

#	Dataset	File Name	Description	Data Fields
1	Movies Data	`movies.csv`	Movie details.	movieId,title,genres
2	User Ratings Data	`user-ratings.csv`	Ratings by a specific user.	userId,movieId,rating,timestamp
3	Community Ratings Data	`ratings.csv`	Ratings by other users.	userId,movieId,rating,timestamp

User ratings file is for the user in context. You can update the ratings in this file based on your movie preferences. We'll assign a user id called "User Id 0" to represent these ratings.

When we run the recommendation engine program, we'll join the specific user ratings data with ratings from the community (other users).

Technologies

We will use Java to write the Spark MLlib program which can be executed as a stand-alone application. The program uses the following MLlib Java classes (all are located in `org.apache.spark.mllib.recommendation` package):

- `ALS`
- `MatrixFactorizationModel`
- `Rating`

We will use the following technologies in the sample application to illustrate how Spark MLlib API is used for performing predictive analytics.

- Apache Spark 1.6.1
- Spark Java API
- JDK version 1.8.0_65
- Apache Maven 3.3.3
- Eclipse IDE

Architecture components of the sample application are shown in Figure 4 below.

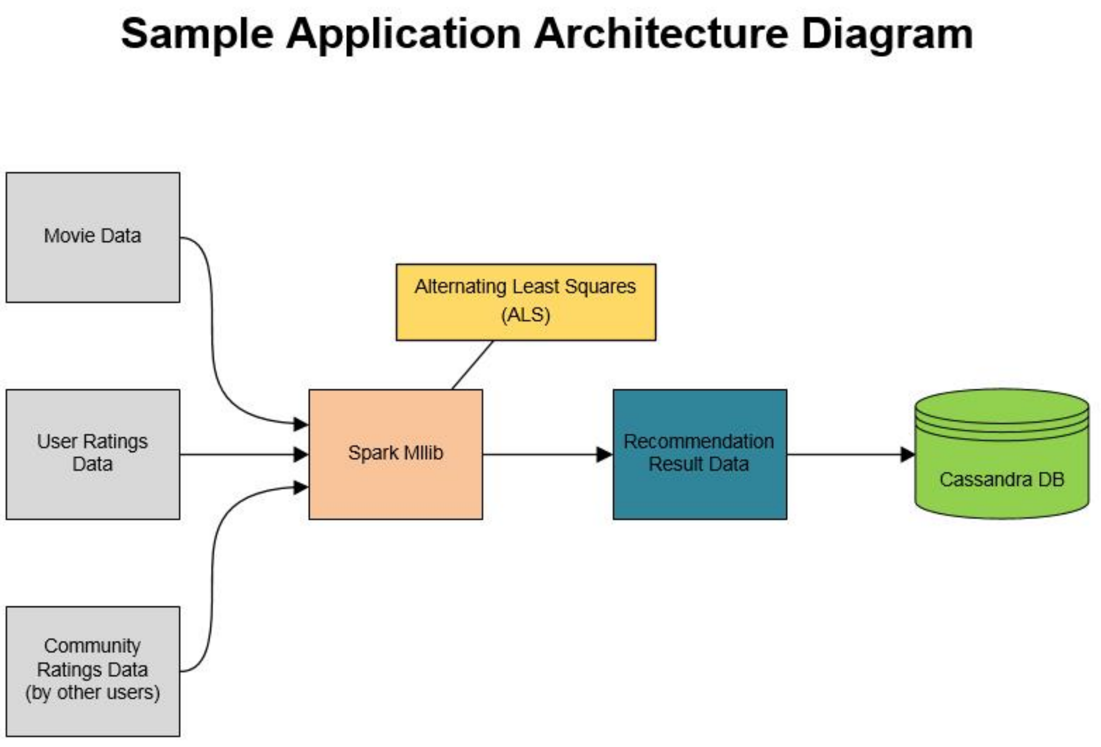

Sample Application Architecture Diagram
Movie Data
User Ratings Data
Community Ratings Data (by other users)
Alternating Least Squares (ALS)
Spark Mllib
Recommendation Result Data
Cassandra DB

Figure 4. Spark Machine Learning Sample Application Architecture

There are several implementations of movie recommendation example available in different languages supported by Spark, like Scala (Databricks and MapR), Java (Spark Examplesand Java based Recommendation Engine), and Python. We'll use the Java solution in our sample application. Download the Java program from Spark examples website to run the example on your local machine. Create a new Maven based Java project called `spark-mllib-`

sample-app and copy the Java class into the project. Modify the Java class to pass in the data sets discussed in the previous section.

Make sure you include the required Spark Java libraries in the dependencies section of Maven pom.xml file. To do a clean build and download Spark library JAR files, you can run the following commands.

Set environment parameters for JDK (JAVA_HOME), Maven (MAVEN_HOME), and Spark (SPARK_HOME)

For Windows Operating System:

```
set JAVA_HOME=[JDK_INSTALL_DIRECTORY]
set PATH=%PATH%;%JAVA_HOME%\bin

set MAVEN_HOME=[MAVEN_INSTALL_DIRECTORY]
set PATH=%PATH%;%MAVEN_HOME%\bin

set SPARK_HOME=[SPARK_INSTALL_DIRECTORY]
set PATH=%PATH%;%SPARK_HOME%\bin

cd c:\dev\projects\spark-mllib-sample-app

mvn clean install
mvn eclipse:clean eclipse:eclipse
```

If you are using a Linux or Mac OSX system, you can run the following commands:

```
export JAVA_HOME=[JDK_INSTALL_DIRECTORY]
export PATH=$PATH:$JAVA_HOME/bin

export MAVEN_HOME=[MAVEN_INSTALL_DIRECTORY]
export PATH=$PATH:$MAVEN_HOME/bin

export SPARK_HOME=[SPARK_INSTALL_DIRECTORY]
export PATH=$PATH:$SPARK_HOME/bin

cd /Users/USER_NAME/spark-mllib-sample-app

mvn clean install
mvn eclipse:clean eclipse:eclipse
```

If application build is successful, the packaged JAR file will be created in target directory.

We will use spark-submit command to execute the Spark program. Here are the commands for running the program in Windows and Linux/Mac respectively.

Windows:

```
%SPARK_HOME%\bin\spark-submit --class
"org.apache.spark.examples.mllib.JavaRecommendationExample" --master local[*]
```

```
target\spark-mllib-sample-1.0.jar
```

Linux/Mac:

```
$SPARK_HOME/bin/spark-submit --class
"org.apache.spark.examples.mllib.JavaRecommendationExample" --master local[*]
target/spark-mllib-sample-1.0.jar
```

Monitoring of Spark MLlib Application

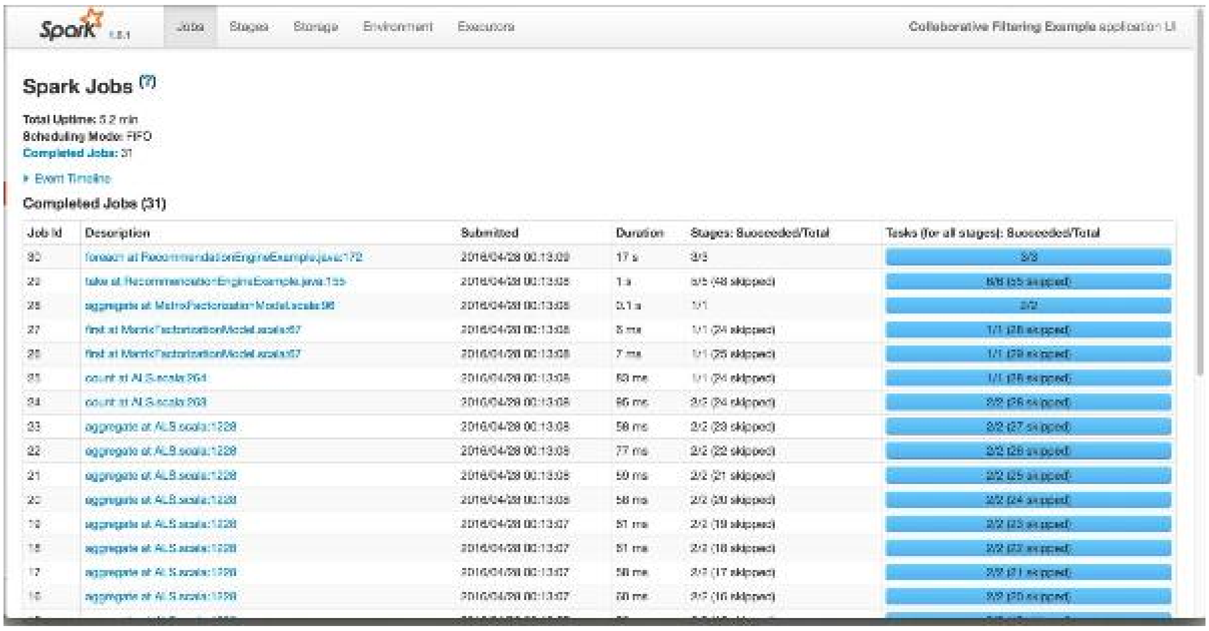
We can monitor the Spark program status on the web console which is available at URL.

Let's look at some of these visualization tools showing the Spark machine learning program statistics.

We can view the details of all jobs in the sample machine learning program. Click on "Jobs" tab on the web console screen to navigate the Spark Jobs web page that shows these job details.

Figure 5 below shows the status of the jobs from the sample program.

(Click on the image to enlarge it)



Spark Jobs (?)

Total Uptime: 5.2 min
Scheduling Mode: FIFO
Completed Jobs: 31

▸ Event Timeline

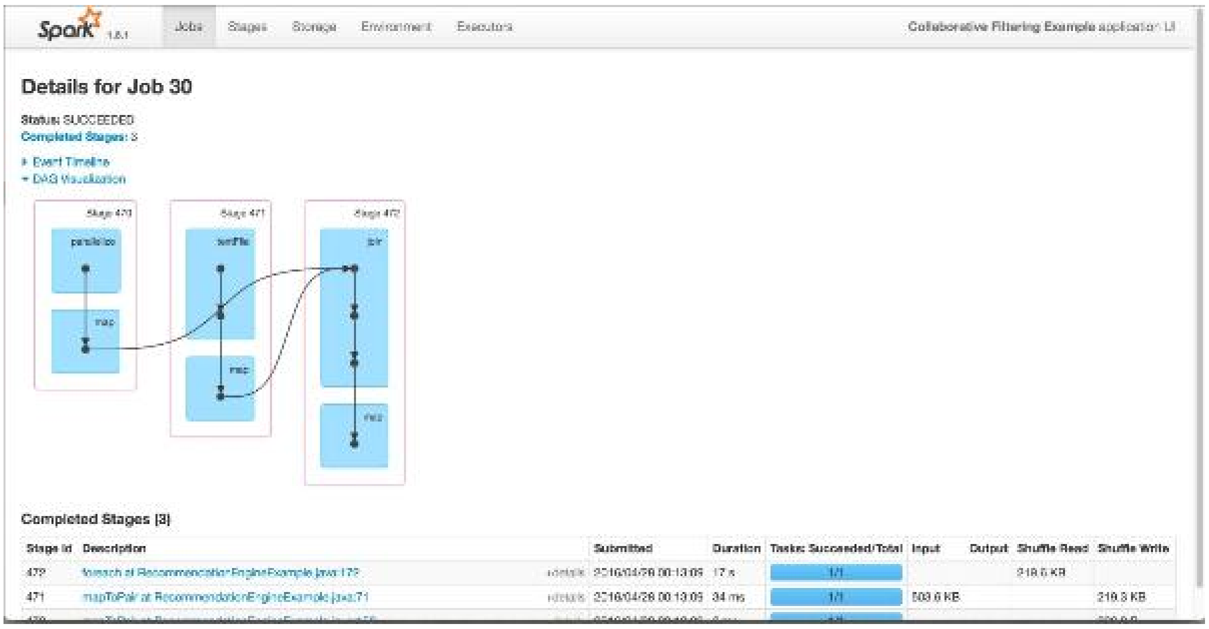
Completed Jobs (31)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
30	foreach at RecommendationEngineExample.java:172	2016/04/28 00:13:09	17 s	3/3	3/3
29	take at RecommendationEngineExample.java:155	2016/04/28 00:13:08	1 s	5/5 (48 skipped)	6/6 (55 skipped)
28	aggregate at MatrixFactorizationModel.scala:96	2016/04/28 00:13:08	0.1 s	1/1	2/2
27	first at MatrixFactorizationModel.scala:67	2016/04/28 00:13:08	5 ms	1/1 (24 skipped)	1/1 (28 skipped)
26	first at MatrixFactorizationModel.scala:67	2016/04/28 00:13:08	7 ms	1/1 (25 skipped)	1/1 (29 skipped)
25	count at ALS.scala:264	2016/04/28 00:13:08	63 ms	1/1 (24 skipped)	1/1 (28 skipped)
24	count at ALS.scala:263	2016/04/28 00:13:08	96 ms	2/2 (24 skipped)	2/2 (28 skipped)
23	aggregate at ALS.scala:1228	2016/04/28 00:13:08	58 ms	2/2 (23 skipped)	2/2 (27 skipped)
22	aggregate at ALS.scala:1228	2016/04/28 00:13:08	77 ms	2/2 (22 skipped)	2/2 (26 skipped)
21	aggregate at ALS.scala:1228	2016/04/28 00:13:08	50 ms	2/2 (21 skipped)	2/2 (25 skipped)
20	aggregate at ALS.scala:1228	2016/04/28 00:13:08	58 ms	2/2 (20 skipped)	2/2 (24 skipped)
19	aggregate at ALS.scala:1228	2016/04/28 00:13:07	51 ms	2/2 (19 skipped)	2/2 (23 skipped)
18	aggregate at ALS.scala:1228	2016/04/28 00:13:07	51 ms	2/2 (18 skipped)	2/2 (22 skipped)
17	aggregate at ALS.scala:1228	2016/04/28 00:13:07	58 ms	2/2 (17 skipped)	2/2 (21 skipped)
16	aggregate at ALS.scala:1228	2016/04/28 00:13:07	60 ms	2/2 (16 skipped)	2/2 (20 skipped)

Figure 5. Spark jobs statistics for the machine learning program

Direct Acyclic Graph (DAG) shows the dependency graph of different RDD operations we ran in the program. Figure 6 below shows the screenshot of this visualization of Spark machine learning job.

(Click on the image to enlarge it)



Details for Job 30

Status: SUCCEEDED
Completed Stages: 3

Completed Stages (3)

Stage Id	Description		Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
472	foreach at RecommendationEngineExample.java:172	+details	2016/04/28 00:13:09	17 s	1/1			219.5 KB	
471	mapToPair at RecommendationEngineExample.java:71	+details	2016/04/28 00:13:09	34 ms	1/1	503.6 KB			219.3 KB

Figure 6. DAG visualization graph of Spark machine learning program

Conclusions

Spark MLlib is one of the two machine learning libraries from Apache Spark. It's used for implementing predictive analytics solutions for business use cases like recommendation engine and fraud detection system. In this article, we learned about how to use Spark MLlib to create a recommendation engine application to predict the movies that a user might like.

Make sure you perform sufficient testing to assess the effectiveness as well as performance of different machine learning techniques to find out the best solution for your requirements and use cases.

What's Next

In the next article, we will focus on the other Machine Learning API from Spark, called Spark ML, which is the recommended solution for big data projects developed on data pipelines.

References

- Big Data Processing using Apache Spark - Part 1: Introduction
- Big Data Processing using Apache Spark - Part 2: Spark SQL
- Big Data Processing using Apache Spark - Part 3: Spark Streaming
- Apache Spark Main Website
- Spark Machine Learning Website
- Spark MLlib Programming Guide
- Spark Summit 2014 Machine Learning Training Exercise

About the Author

Srini Penchikala currently works as Software Architect and is based out of Austin, Texas. He has over 20 years of experience in software architecture, design and development. Srini is currently authoring a book on Apache Spark. He is also the co-author of Spring Roo in Action book from Manning Publications. He has presented at conferences like JavaOne, SEI Architecture Technology Conference (SATURN), IT Architect Conference (ITARC), No Fluff Just Stuff, NoSQL Now, Enterprise Data World, and Project World Conference. Srini also published several articles on software architecture, security & risk management, NoSQL and Big Data topics on websites like InfoQ, The ServerSide, OReilly Network (ONJava), DevX Java, java.net and JavaWorld. He is the Lead Editor for Data Science community at InfoQ.

This is the forth article of the "Big Data Processing with Apache Spark" series. Please see also: Part 1: Introduction, Part 2: Spark SQL, Part 3: Spark Streaming, Part 5: Spark ML Data Pipelines, Part 6: Graph Data Analytics with Spark GraphX.