Big data with Hadoop and Spark

PROJECT

BUILDING A MOVIES RECOMMENDATION SYSTEM USING COLLABORATIVE FILTERING AND ALTERNATE LEAST SQUARE(ALS)

By
Meghana Vasavada
Date: 09-07-2017

CONTENTS

Recommender System

What is a Recommender System?

- **Subclass of information filtering system that seeks to predict the "rating" or "preference" that a user would give to an item E.g. music, books and movies**

Examples:

In eCommerce recommend *items*
In eLearning recommend *content*
In search and navigation recommend *links*
Use *items* as generic term for what is recommended

How does it Help?

- **Help people (customers, users) make *decisions***
- **Recommendation is based on *preferences***
  - **Of an individual**
  - **Of a group or community**

Recommender System

TYPES OF RECOMMENDER SYSTEM

- *Content-Based* (CB) – use personal preferences to match and filter items
  - E.g. what sort of books do I like?
- *Collaborative Filtering* (CF) – match ‘like-minded’ people
  - E.g. if two people have similar ‘taste’ they can recommend items to each other
- *Social Software* – the recommendation process is supported but not automated
  - E.g. Weblogs provide a medium for recommendation
- *Social Data Mining* – Mine log data of social activity to learn group preferences
  - E.g. web usage mining

Collaborative filtering

Collaborative filtering: An efficient algorithm to match people with similar interests

- In Collaborative filtering we make predictions (filtering) about the interests of a user by collecting preferences or taste information from many users(collaborating).
- The underlying assumption is that if a user A has the same opinion as a user B on an issue, A is more likely to have B's opinion on a different issue x than to have the opinion on x of a user chosen randomly.
- Commonly used for recommender systems.
- These techniques aim to fill in the missing entries of a user-item association matrix.

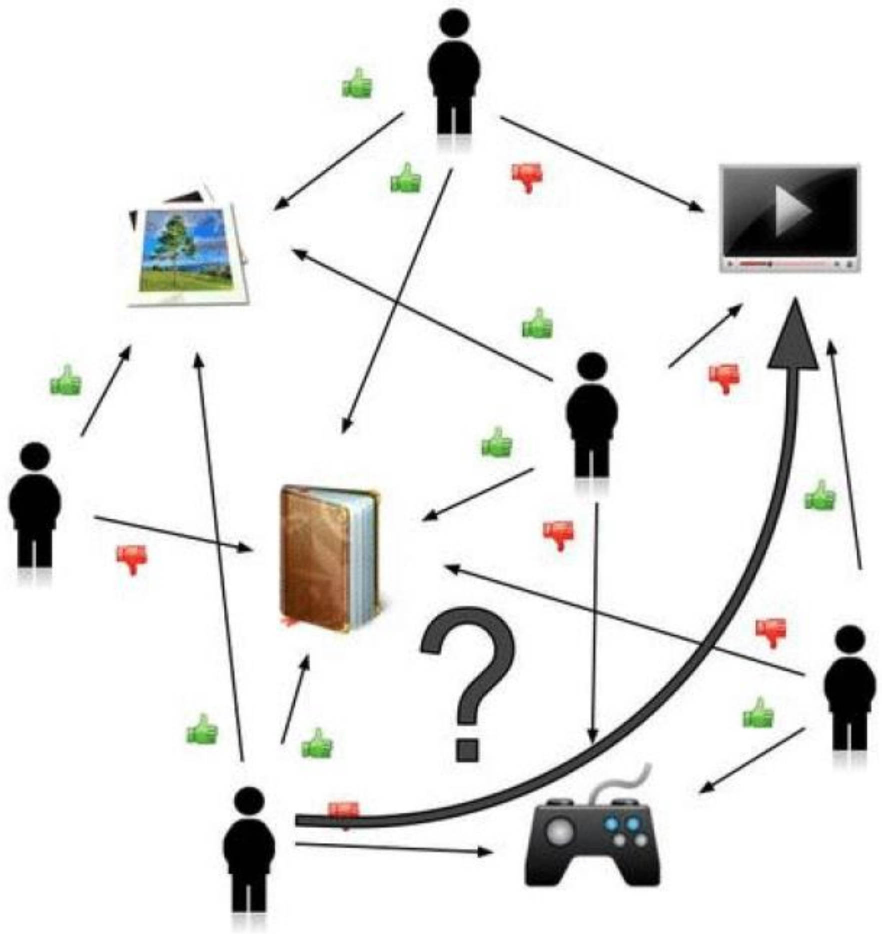
Collaborative filtering

How does Collaborative filtering work?

- Users rate items – user interests recorded. Ratings may be:
  - Explicit:- buying or rating an item
  - Implicit:- browsing time, no. of mouse clicks

- *Nearest neighbour* matching used to find people with similar interests
- Items that neighbours rate highly but that you have *not* rated are recommended to you
- User can then rate recommended items

The image below shows an example of collaborative filtering. At first, people rate different items (like videos, images, games). Then, the system makes predictions about a user's rating for an item not rated yet. The new predictions are built upon the existing ratings of other users with similar ratings with the active user. In the image, the system predicts that the user will not like the video



Collaborative filtering

Spark MLlib library for Machine Learning provides a Collaborative Filtering implementation by using Alternating Least Squares (ALS). The implementation in MLlib has the following parameters:

1. numBlocks is the number of blocks used to parallelize computation (set to 1to autoconfigure).
2. rank is the number of latent factors in the model.
3. iterations is the number of iterations to run.
4. lambda specifies the regularization parameter in ALS.
5. implicitPrefs specifies whether to use the explicit feedback ALS variant or one adapted for implicit feedback data. (optional)
6. alpha is a parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations.

Alternating Least Squares(ALS)

- Matrix R can be factorized into two nonnegative matrices, a user-preference matrix U and a preference-rating matrix M
- The loss function used in ALS is so called *rooted mean square error (RMSE)* defined as

$$\mathcal{L}(R, U, M) = \frac{1}{n} \sum_{i,j} (r_{i,j} - < u_i, m_j >)^2,$$

where n is the number of entries in the rating matrix R.

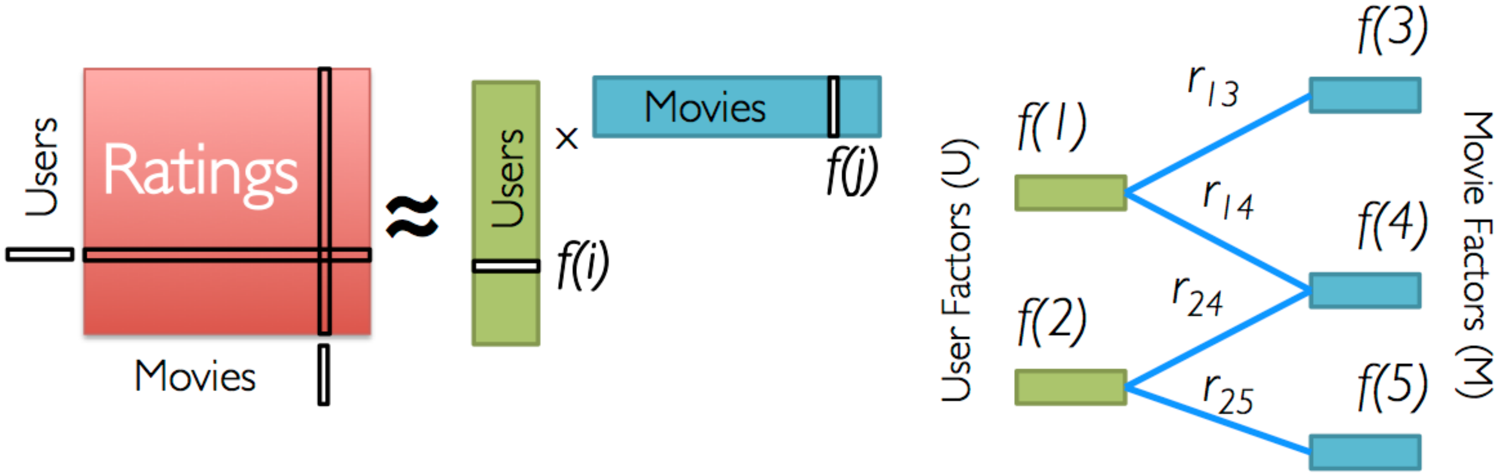
- In addition, ALS applies L norm regularization on the parameter spaces and U and M.
- Combine the loss function, the objective of ALS can be formulated as

$$\min_{U,M} \frac{1}{n} \sum_{i,j} (r_{i,j} - < u_i, m_j >)^2 + \lambda (\sum_i n_{n_i} u_i^2 + \sum_i n_{m_i} m_i^2),$$

- where lambda is the regularization parameter that controls the balance of the loss term and the regularization term, *nui* is the number of movies rated by user *i,* and *nmi* is the number of users that rate movies .

Alternating Least Squares(ALS)

Low-Rank Matrix Factorization:



Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda \|w\|_2^2$$

MovieLens Dataset

- UserId: MovieLens users were selected at random for inclusion. Their ids have been anonymized. User ids are consistent between ratings.csv and tags.csv (i.e., the same id refers to the same user across the two files)
- MovieId: Only movies with at least one rating or tag are included in the dataset. Movie ids are consistent between ratings.csv, tags.csv, movies.csv, and links.csv
- Ratings: All ratings are contained in the file ratings.csv. Ratings are made on a 5-star scale
- Tags: All tags are contained in the file tags.csv. Each line of this file after the header row represents one tag applied to one movie by one user
- Title: Title of the movies from movies.csv
- Genres: Genres are a pipe-separated list, e.g Action, Animation

Movies Recommendation-Pipeline

Create a Dataframe from Movies.csv and Ratings.csv

↓

Create a Training and Test Dataset (80:20)

↓

Train the Training model with ALS training parameters

↓

Transform the model to predict the test dataset and compute the Root Mean Square Error (RMSE)

↓

Transform the model to predict the test dataset

↓

Evaluate the model by Cross Validation approach by fine tuning the parameters- New model

↓

Transform the validated model to predict the test dataset

↓

Movies Recommendation-Pipeline

Create the new dataset of 10 new User ratings with UserId as 0

Add the new dataset to the training dataset

Train the new training dataset with ALS training parameters from best model

Repeat the steps for testing the prediction and RMSE

Transform against the unrated movies

Get the top 20 recommended movies having atleast > 500 reviews

Save the model and export the csv file

ALS model and movies Prediction using SPARK in Databricks

Creating a Dataframe from movies.csv

```
moviesDF = spark.sql("select * from movies")
moviesDF.show(truncate = False)
```

```
+-------+------------------------------------+-------------------------------------------+
|movieId|title                               |genres                                     |
+-------+------------------------------------+-------------------------------------------+
|1      |Toy Story (1995)                    |Adventure|Animation|Children|Comedy|Fantasy|
|2      |Jumanji (1995)                      |Adventure|Children|Fantasy                 |
|3      |Grumpier Old Men (1995)             |Comedy|Romance                             |
|4      |Waiting to Exhale (1995)            |Comedy|Drama|Romance                       |
|5      |Father of the Bride Part II (1995)  |Comedy                                     |
|6      |Heat (1995)                         |Action|Crime|Thriller                      |
|7      |Sabrina (1995)                      |Comedy|Romance                             |
|8      |Tom and Huck (1995)                 |Adventure|Children                         |
|9      |Sudden Death (1995)                 |Action                                     |
|10     |GoldenEye (1995)                    |Action|Adventure|Thriller                  |
|11     |American President, The (1995)      |Comedy|Drama|Romance                       |
|12     |Dracula: Dead and Loving It (1995)  |Comedy|Horror                              |
|13     |Balto (1995)                        |Adventure|Animation|Children               |
|14     |Nixon (1995)                        |Drama                                      |
|15     |Cutthroat Island (1995)             |Action|Adventure|Romance                   |
|16     |Casino (1995)                       |Crime|Drama                                |
|17     |Sense and Sensibility (1995)        |Drama|Romance                              |
|18     |Four Rooms (1995)                   |Comedy                                     |
|19     |Ace Ventura: When Nature Calls (1995)|Comedy                                     |
|20     |Money Train (1995)                  |Action|Comedy|Crime|Drama|Thriller         |
+-------+------------------------------------+-------------------------------------------+
only showing top 20 rows
```

ALS model and movies Prediction using SPARK in Databricks

Creating a Dataframe from ratings.csv

ratingsDF = spark.sql("select * from ratings")
ratingsDF.show()

```
ratingsDF.show(10)
```

▸ (1) Spark Jobs

```
+------+-------+------+-------------------+
|userId|movieId|rating|          timestamp|
+------+-------+------+-------------------+
|     1|      2|   3.5|2005-04-02 23:53:47|
|     1|     29|   3.5|2005-04-02 23:31:16|
|     1|     32|   3.5|2005-04-02 23:33:39|
|     1|     47|   3.5|2005-04-02 23:32:07|
|     1|     50|   3.5|2005-04-02 23:29:40|
|     1|    112|   3.5|2004-09-10 03:09:00|
|     1|    151|   4.0|2004-09-10 03:08:54|
|     1|    223|   4.0|2005-04-02 23:46:13|
|     1|    253|   4.0|2005-04-02 23:35:40|
|     1|    260|   4.0|2005-04-02 23:33:46|
+------+-------+------+-------------------+
only showing top 10 rows
```

ALS model and movies Prediction using SPARK in Databricks

EDA: Compute statistics to get a better understanding of the data

```
1  moviesDF.describe().show()
```

▸ (1) Spark Jobs

```
+-------+-----------------+--------------------+------------------+
|summary|          movieId|               title|            genres|
+-------+-----------------+--------------------+------------------+
|  count|            27278|               27278|             27278|
|   mean|59855.48057042305|                null|              null|
| stddev|44429.31469707313|                null|              null|
|    min|                1|"""Great Performa...|(no genres listed)|
|    max|            99999|          貞子3D (2012)|           Western|
+-------+-----------------+--------------------+------------------+
```

```
1  ratingsDF.describe().show()
```

▸ (1) Spark Jobs

```
+-------+-----------------+-----------------+------------------+
|summary|           userId|          movieId|            rating|
+-------+-----------------+-----------------+------------------+
|  count|         20000263|         20000263|          20000263|
|   mean|69045.87258292554|9041.567330339605|3.5255285642993797|
| stddev|40038.62665316182|19789.47744541297|  1.05198891929425|
|    min|                1|                1|               0.5|
|    max|           138493|           131262|               5.0|
+-------+-----------------+-----------------+------------------+
```

ALS model and movies Prediction using SPARK in Databricks

Analysis to display movies with Highest ratings with more than 500 reviews before training

Movies with highest ratings:

average	title	count	movieId
4.446990499637029	Shawshank Redemption, The (1994)	63366	318
4.364732196832306	Godfather, The (1972)	41355	858
4.334372207803259	Usual Suspects, The (1995)	47006	50
4.310175010988133	Schindler's List (1993)	50054	527
4.275640557704942	Godfather: Part II, The (1974)	27398	1221
4.2741796572216	Seven Samurai (Shichinin no samurai) (1954)	11611	2019
4.271333600779414	Rear Window (1954)	17449	904
4.263182346109176	Band of Brothers (2001)	4305	7502
4.258326830670664	Casablanca (1942)	24349	912
4.256934865900383	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	6525	922
4.24807897901911	One Flew Over the Cuckoo's Nest (1975)	29932	1193
4.247286821705426	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	23220	750
4.246001523229246	Third Man, The (1949)	6565	1212
4.235410064157069	City of God (Cidade de Deus) (2002)	12937	6016
4.2347902097902095	Lives of Others, The (Das leben der Anderen) (2006)	5720	44555
4.233538107122288	North by Northwest (1959)	15627	908
4.2326233183856505	Paths of Glory (1957)	3568	1178
4.227123123722136	Fight Club (1999)	40106	2959
4.224281931146873	Double Indemnity (1944)	4909	3435
4.224137931034483	12 Angry Men (1957)	12934	1203

only showing top 20 rows

ALS model and movies Prediction using SPARK in Databricks

Collaborative Filtering:

- Splitting Data into Train and test datasets
- Generate a Recommendation model using ALS on the training data and transforming it with Test dataset to generate predictions

(training, test) = ratingsDF.randomSplit([0.8, 0.2])

```
print "Training set size: ", training.count()
print "Test set size: ", test.count()
#print "Validation set size: ", test.count()
```

(2) Spark Jobs

```
Training set size:  15998929
Test set size:  4001334
```

Command took 1.13 minutes -- by meghana.rwgsql@gmail.com at 8/31/2017, 10:39:13 AM on movies_project

Cmd 32

```
# Build the recommendation model using ALS on the training data
# Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row

als = ALS(rank=10, maxIter=10, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating")
#als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating", coldStartStrategy="drop")
model = als.fit(training)
```

(5) Spark Jobs

Command took 2.11 minutes -- by meghana.rwgsql@gmail.com at 8/31/2017, 10:40:44 AM on movies_project

Cmd 33

```
#predictions = model.transform(test).dropna()
predictions = model.transform(test)
predictions = predictions.dropna()
predictions.registerTempTable("predictions_table")
```

ALS model and movies Prediction using SPARK in Databricks

Collaborative Filtering:

The script below generates top 10 user recommendations for each movie. Here 10 movies

movieRecs = model.recommendForAllItems(10)
movieRecs.show(10,truncate = False)

▸ (1) Spark Jobs

```
+-------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|movieId|recommendations                                                                                                                                                                               |
+-------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|148    |[[33314,8.701766], [1425,8.180933], [81312,7.373311], [68156,7.2083273], [106990,6.8708425], [86490,6.8576126], [131667,6.6769357], [113811,6.571302], [42589,6.48554], [36374,6.4662094]]  |
|463    |[[120759,7.827287], [103795,7.3661604], [1425,7.3247886], [7637,7.0286155], [41182,6.988009], [65884,6.946594], [46236,6.7009373], [101608,6.600505], [117304,6.5729437], [130899,6.54625]] |
|471    |[[15629,7.3206534], [39244,7.006679], [68156,6.9869514], [55809,6.806746], [35429,6.7478523], [17999,6.739544], [107667,6.633784], [132964,6.6239376], [112794,6.5889277], [54192,6.460428]]|
|496    |[[1425,11.919976], [33314,8.572311], [42589,8.134817], [120759,7.6434937], [101255,7.342348], [26714,7.2738714], [86490,7.260195], [59542,7.174942], [72203,7.1624765], [36374,7.149855]]   |
|833    |[[7245,6.9702134], [138324,6.848807], [86490,6.721907], [121230,6.658245], [126728,6.6005583], [29030,6.476948], [29388,6.4507437], [114637,6.3349886], [51105,6.30784], [67679,6.2839894]] |
|1088   |[[61007,9.486544], [92390,8.747037], [7332,7.796366], [101608,7.435592], [95345,7.182036], [99906,7.117282], [8892,6.961412], [65884,6.9582014], [132685,6.84749], [116845,6.832096]]       |
|1238   |[[61315,7.817972], [81312,6.8869853], [101255,6.847694], [110290,6.76576], [14403,6.6632776], [105116,6.627554], [27841,6.5338273], [7326,6.5334315], [92724,6.456632], [73969,6.453871]]   |
|1342   |[[1425,7.136017], [125248,6.5628214], [97609,6.553137], [122900,6.551678], [49826,6.5504475], [11941,6.4981246], [22946,6.486262], [124859,6.455388], [9635,6.372327], [9340,6.367054]]     |
|1580   |[[36089,6.003181], [861,5.9432993], [112095,5.7447734], [46278,5.7132716], [127916,5.7111936], [71611,5.683817], [117250,5.68226], [115837,5.658524], [52622,5.6481333], [22718,5.6437683]] |
|1591   |[[116848,6.179776], [1425,6.070424], [35823,6.035642], [69716,5.946124], [96611,5.924816], [95981,5.9094105], [86490,5.883312], [112156,5.8413234], [28134,5.7672577], [27735,5.762802]]    |
+-------+----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
only showing top 10 rows
```

ALS model and movies Prediction using SPARK in Databricks

Collaborative Filtering: Computing Root Mean Square error (RMSE) from the first model

Cmd 42

```
 # Evaluate the model by computing the RMSE on the test data
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
```

(1) Spark Jobs

```
Root-mean-square error = 0.804667235236
```

Command took 39.97 seconds -- by meghana.rwgsql@gmail.com at 8/31/2017, 11:17:09 AM on movies_project

Cmd 43

```
displayHTML("<h4>The Root-mean-square error is %s</h4>" % str(rmse))
```

The Root-mean-square error is 0.804667235236

```
predictions.describe().show()
```

(1) Spark Jobs

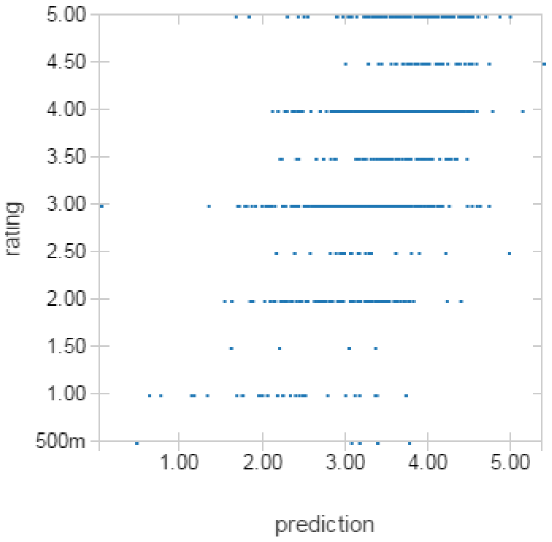
summary	userId	movieId	rating	prediction
count	4000316	4000316	4000316	4000316
mean	69017.89363490284	9013.73503618214	3.525570604922211	3.4412456563136664
stddev	40043.73561717234	19720.687189242326	1.0519828325185538	0.7479698634538245
min	1	1	0.5	-6.137381
max	138493	131013	5.0	9.837434

Command took 38.73 seconds -- by meghana.rwgsql@gmail.com at 8/31/2017, 11:20:32 AM on movies_project

Plot: Ratings vs Predictions

```
display (spark.sql("select rating, prediction from predictions_table"))
```

(1) Spark Jobs



Showing sample based on the first 1000 rows.

Plot Options...

ALS model and movies Prediction using SPARK in Databricks

Collaborative Filtering: Validating the model by fine tuning the parameters

```
#MODEL TUNING AND CROSS VALIDATION

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder

# 1. you first need to build a parameter grid based on your ALS model

paramGrid = ParamGridBuilder() \
                    .addGrid(als.rank, [10, 30]) \
                    .addGrid(als.maxIter, [10, 20]) \
                    .build()

# 2. use your ALS estimator and parameter grid to build cross validator
crossval = CrossValidator(estimator=als,
                          estimatorParamMaps=paramGrid,
                          evaluator=evaluator,
                          numFolds=3)  # use 3+ folds in practice

# Run cross-validation, and choose the best set of parameters.
cvModel = crossval.fit(training)
```

▸ (6) Spark Jobs

Command took 1.11 hours -- by meghana.rwgsql@gmail.com at 8/31/2017, 11:29:07 AM on movies_project

Cmd 79

```
# Make predictions on test documents. cvModel uses the best model
cv_predictions = cvModel.transform(test)
```

ALS model and movies Prediction using SPARK in Databricks

Collaborative Filtering: Computing RMSE and generating plot of data from Validated model model

```
cv_predictions = cv_predictions.dropna()
cv_predictions.registerTempTable("cv_predictions_table")
```

```
#Calculating root mean square
cv_rmse = evaluator.evaluate(cv_predictions)
print("Root-mean-square error = " + str(cv_rmse))
```

▸ (1) Spark Jobs

```
Root-mean-square error = 0.804667235236
```

Command took 38.75 seconds -- by meghana.rwgsql@gmail.com at 8/31/2017, 12:42:07 PM on movies_project

Cmd 85

```
displayHTML("<h4>The Root-mean-square error is %s</h4>" % str(cv_rmse))
```

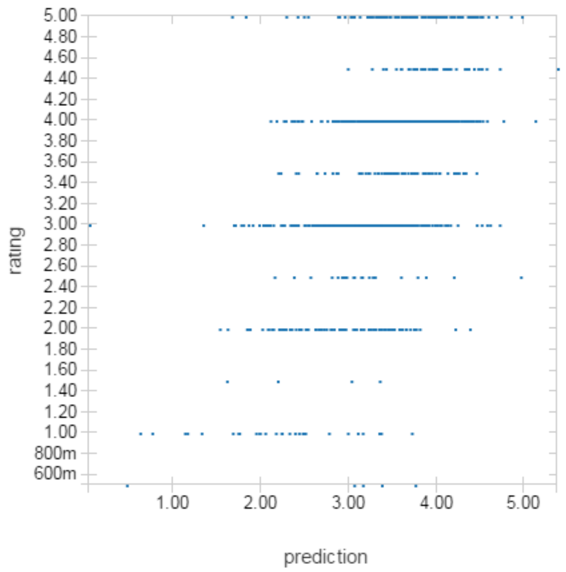
The Root-mean-square error is 0.804667235236

Command took 0.07 seconds -- by meghana.rwgsql@gmail.com at 8/31/2017, 12:43:23 PM on movies_project

Plotting Ratings vs Predictions

```
display (spark.sql("select rating, prediction from cv_predictions_table"))
```

▸ (1) Spark Jobs



ALS model and movies Prediction using SPARK in Databricks

Collaborative Filtering: Predicting new user ratings

We need to rate some movies for the new user. We will put them in a new RDD and we will use the user ID 0, that is not assigned in the MovieLens dataset.

```
#Adding new user ratings.
#Now we need to rate some movies for the new user. We will put them in a new RDD and we will use the user ID 0, that is not assigned in the MovieLens dataset

new_user_ID = 0
# The format of each line is (userID, movieID, rating)
new_user_ratings = [
(0,318,4.5,1112484580), # Shawshank Redemption (1994)
(0,858,4.7,1112484940), # Godfather (1972)
(0,50,5.0,1094785709), # Usual suspects (1995)
(0,527,5.0,1094785691), # Schindlers List (1993)
(0,1221,5.0,1094785759), # Godfather: Part II (1974)
(0,2019,4.8,1112484735), # Seven Samurai (1954)
(0,904,4.5,1094786062), # Rear window (1954)
(0,7502,4.7,1094785764), # Band of Brothers (2001)
(0,912,4.2,1112486150) , # Casablanca (1942)
(0,922,4.0,1112486098) # Sunset blvd (1950)
]
new_user_ratings_RDD = sc.parallelize(new_user_ratings)
print 'New user ratings: %s' % new_user_ratings_RDD.take(10)

```

ALS model and movies Prediction using SPARK in Databricks

Collaborative Filtering: Predicting new user ratings

New Training data = new user ratings + training dataset

Building a new model with ALS on new Training dataset

```
# TODO: Replace <FILL IN> with appropriate code
training_with_my_ratings = training.unionAll(new_user_ratings)

print ('The training dataset now has %s more entries than the original training dataset' %
       (training_with_my_ratings.count() - training.count()))
assert (training_with_my_ratings.count() - training.count()) == new_user_ratings.count()
#The training dataset now has 11 more entries than the original training dataset
```

▸ (5) Spark Jobs

```
The training dataset now has 10 more entries than the original training dataset
```

Command took 1.96 minutes -- by meghana.rwgsql@gmail.com at 8/31/2017, 1:21:36 PM on movies_project

Cmd 59

```
#Training  new dataset with ALS model

# Build the new recommendation model using ALS on the training data
# Note we set cold start strategy to 'drop' to ensure we don't get NaN evaluation metrics
from pyspark.sql import SparkSession
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
#from pyspark.mllib.recommendation import ALS, Rating
from pyspark.sql import Row

als = ALS(rank=10, maxIter=10, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating")
#als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating", coldStartStrategy="drop")
new_rating_model = als.fit(training_with_my_ratings)

#print "New model trained in %s seconds" % round(tt,3)

```

ALS model and movies Prediction using SPARK in Databricks

Collaborative Filtering: Testing the new model with test dataset

```
#predictions = model.transform(test).dropna()
predictions_with_my_ratings = new_rating_model.transform(test)
predictions_with_my_ratings = predictions_with_my_ratings.dropna()
predictions_with_my_ratings.registerTempTable("predictions_with_my_ratings_table")
```

Command took 0.07 seconds -- by meghana.rwgsql@gmail.com at 8/31/2017, 1:28:27 PM on movies_project

Computing RMSE

```
 # Evaluate the model by computing the RMSE on the test data
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions_with_my_ratings)
print("Root-mean-square error for new ratings = " + str(rmse))*
```

▸ (1) Spark Jobs

```
Root-mean-square error for new ratings = 0.805098946653
```

Command took 38.66 seconds -- by meghana.rwgsql@gmail.com at 8/31/2017, 1:30:03 PM on movies_project

ALS model and movies Prediction using SPARK in Databricks

Predicting new user ratings: Top 10 user recommendations for each movie from the new model

```python
# Generate top 10 user recommendations for each movie
movieRecsnew = new_rating_model.recommendForAllItems(10)
movieRecsnew.show(10,truncate = False)
```

▸ (1) Spark Jobs

```
+-------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|movieId|recommendations                                                                                                                                                                                         |
+-------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
|148    |[[124002,8.557087], [89066,8.09482], [43815,7.5858655], [33314,7.496185], [128821,7.2536316], [56145,6.8170147], [112318,6.7748556], [5747,6.746767], [61007,6.746116], [87447,6.7400374]]     |
|463    |[[101608,7.5712504], [65884,7.5201693], [92390,7.3081822], [118248,7.1546474], [79298,6.9978046], [7637,6.960828], [28022,6.9503565], [54584,6.9177628], [67339,6.9005127], [112156,6.639238]] |
|471    |[[107804,6.9638305], [68156,6.7373343], [75545,6.385509], [72458,6.2377696], [35429,6.223061], [55716,6.198926], [34369,6.1974516], [53118,6.1938286], [107667,6.1792274], [120483,6.1639795]] |
|496    |[[1425,8.402818], [5625,8.163208], [58845,8.043345], [84580,7.989399], [119988,7.8339767], [17816,7.8166494], [130693,7.7461033], [4273,7.668422], [112469,7.6296096], [74030,7.615759]]       |
|833    |[[97184,7.3606925], [75306,6.9260845], [138215,6.9125247], [67679,6.848727], [5474,6.839116], [86166,6.765509], [5747,6.7330513], [121534,6.7165036], [121230,6.4955673], [50529,6.48988]]     |
|1088   |[[74576,7.644407], [133404,7.607803], [76772,7.459447], [120759,7.444519], [24829,7.4256725], [93821,7.417887], [62342,7.396893], [93809,7.390879], [7245,7.360171], [110831,7.3552527]]       |
|1238   |[[61315,7.6330214], [53192,6.8270493], [19366,6.5271854], [5747,6.521721], [84889,6.5086412], [14403,6.4422936], [14571,6.4190235], [51055,6.383786], [87447,6.3422785], [9241,6.3329163]]     |
|1342   |[[97609,7.4944735], [124859,6.8256826], [133374,6.8231425], [63539,6.642704], [63122,6.6319804], [122900,6.5695662], [114432,6.5227604], [1425,6.5136213], [68007,6.2603736], [115976,6.259104]]|
|1580   |[[115837,6.137461], [42714,5.9726377], [54192,5.9595323], [20349,5.8790793], [72725,5.8467574], [34369,5.836102], [88922,5.812204], [36089,5.791688], [44957,5.7126417], [22718,5.70847]]      |
|1591   |[[1425,7.4258857], [138215,6.4615006], [86490,6.347809], [99021,6.153923], [112156,6.141174], [97136,6.1349545], [69716,6.087041], [27735,6.070429], [107650,5.999553], [54192,5.922649]]      |
+-------+--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------+
only showing top 10 rows
```

ALS model and movies Prediction using SPARK in Databricks

Filter out movies with high ratings > 500 reviews from the predicted dataset

```python
#We want to filter our movies with high ratings but greater than or equal to 500 reviews.
#movies_with_500_ratings_or_more = movie_names_with_avg_ratings_df.<FILL_IN>
new_predicted_movies_with_500_ratings_or_more = (new_movie_names_with_avg_pred_ratings_df
                                    .where("count >= 500")
                                    .orderBy(new_movie_names_with_avg_pred_ratings_df.average.desc()))

print 'Movies with highest ratings:'
new_predicted_movies_with_500_ratings_or_more.show(20, truncate=False)
```

▸ (2) Spark Jobs

```
Movies with highest ratings:
+-----------------+-----------------------------------------------------------------------------+-----+-------+
|average          |title                                                                        |count|movieId|
+-----------------+-----------------------------------------------------------------------------+-----+-------+
|4.254283139357548|Shawshank Redemption, The (1994)                                             |12513|318    |
|4.22503672482484 |Third Man, The (1949)                                                        |1317 |1212   |
|4.195024231735141|Godfather, The (1972)                                                        |8345 |858    |
|4.175523698708929|Seven Samurai (Shichinin no samurai) (1954)                                  |2359 |2019   |
|4.17406071068002 |Rear Window (1954)                                                           |3538 |904    |
|4.173247438199274|Usual Suspects, The (1995)                                                   |9424 |50     |
|4.172001171513332|Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)                                |1322 |922    |
|4.170191933998413|Casablanca (1942)                                                            |4949 |912    |
|4.152172995341341|Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)|4591 |750    |
|4.140182386042633|Schindler's List (1993)                                                      |9804 |527    |
|4.138639154275057|North by Northwest (1959)                                                    |3161 |908    |
|4.138031953557074|Rashomon (Rashômon) (1950)                                                   |738  |5291   |
|4.135877628541436|Big Sleep, The (1946)                                                        |1131 |1284   |
|4.134845475108205|Double Indemnity (1944)                                                      |963  |3435   |
|4.131054048945816|Paths of Glory (1957)                                                        |737  |1178   |
|4.130510358326207|12 Angry Men (1957)                                                          |2649 |1203   |
|4.127649324721303|Thin Man, The (1934)                                                         |638  |950    |
|4.126916712084419|Notorious (1946)                                                             |966  |930    |
|4.125684980212188|All About Eve (1950)                                                         |943  |926    |
|4.125092605320183|Touch of Evil (1958)                                                         |925  |1248   |
+-----------------+-----------------------------------------------------------------------------+-----+-------+
only showing top 20 rows
```

ALS model and movies Prediction using SPARK in Databricks

Filter out movies with high ratings > 500 reviews from the predicted dataset

Cmd 72

```
predictions_with_my_ratings.registerTempTable("predictions_with_my_ratings_table")
```

Cmd 73

```
new_predicted_movies_with_500_ratings_or_more.registerTempTable("Recommended_movies")
```

Command took 0.07 seconds -- by meghana.rwgsql@gmail.com at 8/31/2017, 3:03:27 PM on movies_project

Cmd 74

```
spark.sql("select * from Recommended_movies limit 10").show(truncate = False)
```

▸ (2) Spark Jobs

```
+-----------------+-------------------------------------------------------------------+-----+-------+
|average          |title                                                              |count|movieId|
+-----------------+-------------------------------------------------------------------+-----+-------+
|4.254283139357548|Shawshank Redemption, The (1994)                                   |12513|318    |
|4.22503672482484 |Third Man, The (1949)                                              |1317 |1212   |
|4.195024231735141|Godfather, The (1972)                                              |8345 |858    |
|4.175523698708929|Seven Samurai (Shichinin no samurai) (1954)                        |2359 |2019   |
|4.17406071068002 |Rear Window (1954)                                                 |3538 |904    |
|4.173247438199274|Usual Suspects, The (1995)                                         |9424 |50     |
|4.172001171513332|Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)                      |1322 |922    |
|4.170191933998413|Casablanca (1942)                                                  |4949 |912    |
|4.152172995341341|Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)|4591 |750    |
|4.140182386042633|Schindler's List (1993)                                            |9804 |527    |
+-----------------+-------------------------------------------------------------------+-----+-------+
```

ALS model and movies Prediction using SPARK in Databricks

average	title	count	movieId
4.254283139	Shawshank Redemption, The (1994)	12513	318
4.225036725	Third Man, The (1949)	1317	1212
4.195024232	Godfather, The (1972)	8345	858
4.175523699	Seven Samurai (Shichinin no samurai) (1954)	2359	2019
4.174060711	Rear Window (1954)	3538	904
4.173247438	Usual Suspects, The (1995)	9424	50
4.172001172	Sunset Blvd. (a.k.a. Sunset Boulevard) (1950)	1322	922
4.170191934	Casablanca (1942)	4949	912
4.152172995	Dr. Strangelove or: How I Learned to Stop Worrying and Love the Bomb (1964)	4591	750
4.140182386	Schindler's List (1993)	9804	527

ALS model and movies Prediction using SPARK in Databricks

Getting the movies from Recommended_movies table and Predictions Dataframe

```
Rec_movies_gt_500_reviews = spark.sql("select Recommended_movies.movieId, Recommended_movies.title,
Recommended_movies.count, Recommended_movies.average,predictions_with_my_ratings_table.prediction \
        FROM Recommended_movies \
        JOIN predictions_with_my_ratings_table ON predictions_with_my_ratings_table.movieId=Recommended_movies.movieId \
        ORDER BY predictions_with_my_ratings_table.prediction DESC\
        LIMIT 20")
Rec_movies_gt_500_reviews.show(truncate = False)
```

movieId	title	count	average	prediction
1035	Sound of Music, The (1965)	2818	3.710417884368267	7.885456
1354	Breaking the Waves (1996)	756	3.7397369243322856	7.354319
913	Maltese Falcon, The (1941)	2451	4.0977760245974135	7.315525
327	Tank Girl (1995)	1441	2.8083723716174127	7.1648254
1923	There's Something About Mary (1998)	4882	3.4361935991432175	7.1481495
1721	Titanic (1997)	6576	3.1962290323677034	7.135368
39	Clueless (1995)	5173	3.3595627621147495	7.1215887
39	Clueless (1995)	5173	3.3595627621147495	7.068882
2700	South Park: Bigger, Longer and Uncut (1999)	3442	3.521844895531676	7.043368
231	Dumb & Dumber (Dumb and Dumber) (1994)	6354	2.8606806995677223	7.019651
1088	Dirty Dancing (1987)	2220	3.164470894111169	6.9840746
1	Toy Story (1995)	9991	3.779025166437064	6.9619803
34	Babe (1995)	6462	3.547983020373987	6.9606657
920	Gone with the Wind (1939)	2895	3.7159833018489454	6.943983
6503	Charlie's Angels: Full Throttle (2003)	862	2.463607352559742	6.9119782
899	Singin' in the Rain (1952)	2064	4.011479699966144	6.868468
5618	Spirited Away (Sen to Chihiro no kamikakushi) (2001)	2665	4.095016690501576	6.836734
7153	Lord of the Rings: The Return of the King, The (2003)	6349	3.969171227554878	6.835625
1721	Titanic (1997)	6576	3.1962290323677034	6.8279424
4993	Lord of the Rings: The Fellowship of the Ring, The (2001)	7490	3.984718856287098	6.8215823

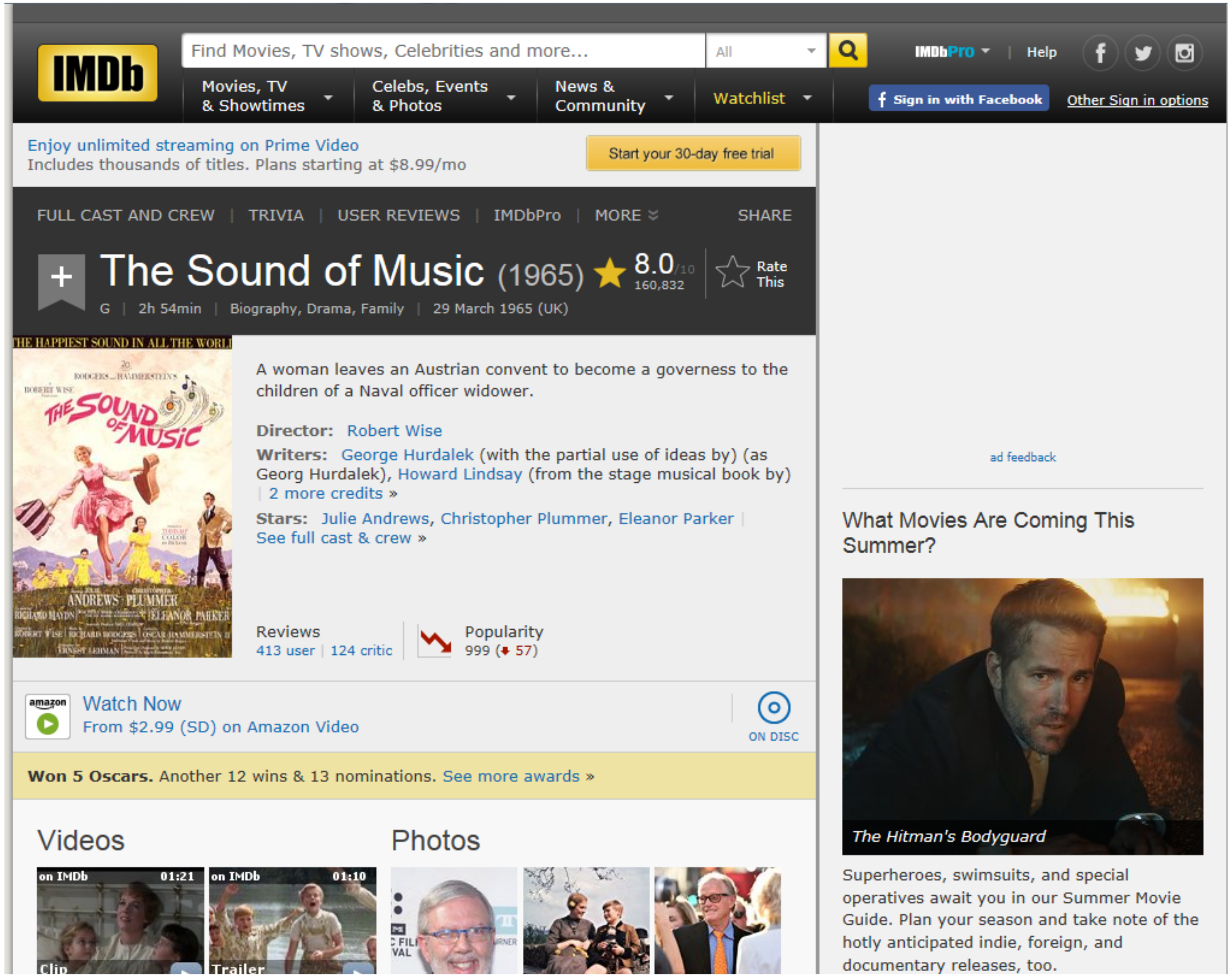
ALS model and movies Prediction using SPARK in Databricks

Recommended movies with > 500 reviews and ratings: From Recommended_movies table and Predictions Dataframe

movieId	title	count	average	prediction
1035	Sound of Music, The (1965)	2818	3.710417884	7.885456
1354	Breaking the Waves (1996)	756	3.739736924	7.354319
913	Maltese Falcon, The (1941)	2451	4.097776025	7.315525
327	Tank Girl (1995)	1441	2.808372372	7.1648254
1923	There's Something About Mary (1998)	4882	3.436193599	7.1481495
1721	Titanic (1997)	6576	3.196229032	7.135368
39	Clueless (1995)	5173	3.359562762	7.1215887
39	Clueless (1995)	5173	3.359562762	7.068882
2700	South Park: Bigger, Longer and Uncut (1999)	3442	3.521844896	7.043368
231	Dumb & Dumber (Dumb and Dumber) (1994)	6354	2.8606807	7.019651
1088	Dirty Dancing (1987)	2220	3.164470894	6.9840746
1	Toy Story (1995)	9991	3.779025166	6.9619803
34	Babe (1995)	6462	3.54798302	6.9606657
920	Gone with the Wind (1939)	2895	3.715983302	6.943983
6503	Charlie's Angels: Full Throttle (2003)	862	2.463607353	6.9119782
899	Singin' in the Rain (1952)	2064	4.0114797	6.868468
5618	Spirited Away (Sen to Chihiro no kamikakushi) (2001)	2665	4.095016691	6.836734
7153	Lord of the Rings: The Return of the King, The (2003)	6349	3.969171228	6.835625
1721	Titanic (1997)	6576	3.196229032	6.8279424
4993	Lord of the Rings: The Fellowship of the Ring, The (2001)	7490	3.984718856	6.8215823

Movies Recommendation

Sound of Music, The (1965)



Conclusion

- Spark's MLlib library provides scalable data analytics through a rich set of methods.
- Its Alternating Least Squares implementation for Collaborative Filtering is one that fits perfectly in a recommendation engine.
- Due to its very nature, collaborative filtering is a costly procedure since requires updating its model when new user preferences arrive.
- Therefore, having a distributed computation engine such as Spark to perform model computation is a must in any real world.

References

- https://github.com/apache/spark/blob/master/examples/src/main/python/ml/als_example.py
- http://spark.apache.org/docs/preview/mllib-collaborative-filtering.html#collaborative-filtering

THANK YOU