Final Project Report

Cobbinah Beatrice Yaa Yeboah

May 11th 2020

Spring 2020 – Distributed and Scalable Data

Motivation:

People may be interested in a product but would often not comb through multitudes of information just to find what they need. In other words, the sales of several companies are dependent on their ability to present the user preferences in their eyesight. This project intends to predict what the users' opinion is on Netflix movies and as such recommend movies the user may have never seen.

System Configuration:

Configuring the system consisted of setting up a Standard Amazon EMR cluster – m4.xlarge, 2 slave nodes, 1 master node and Amazon S3 bucket

Documentation of Approach and Big Data Application Analysis:

Problem 1

Step 1

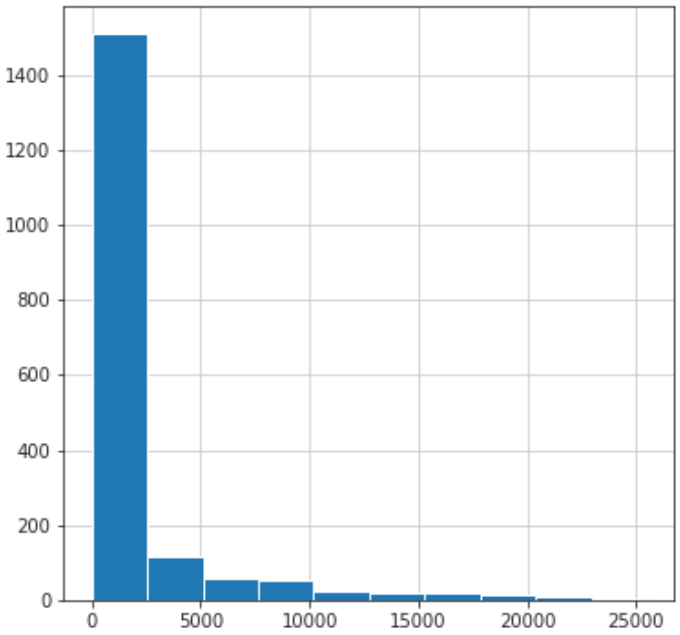
Data uploads and pre-processing:

1. Create an Amazon S3 bucket and upload the data. The data consist of one training, one testing set and a movie data that has the titles and the year of the release.
2. SSH into the master node of the EMR and set up your Jupyter Notebook with pyspark.
3. Load the data from the S3 and set up an schema to hold the data.
4. Create a dataframe to hold the data
5. We map attempt to cast the latitude and longitude to floats.

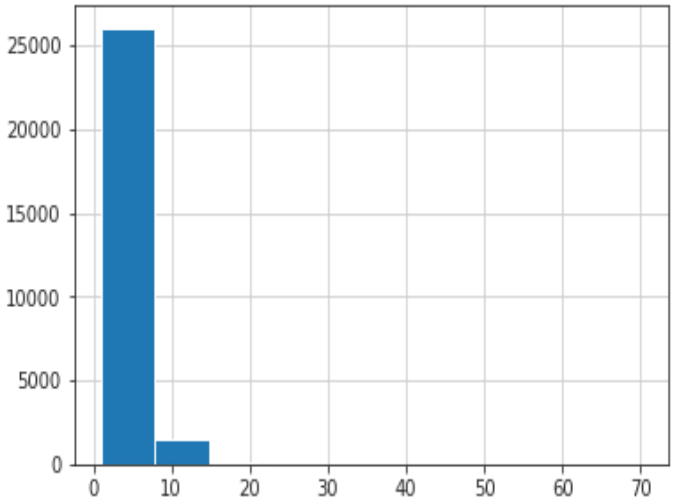
Step 2

Now that we have all the data set up, we will get a view our data is distributed.

First, we will look at a graph of the counts of movies in the training dataset

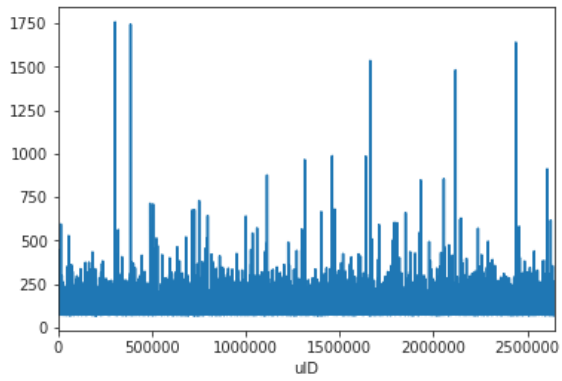


Next, lets look at a graph of the counts of users in the test set based on their rating



It seems that most of the users in our testing dataset did not rate the movies they watched.

Finally, we will take a look at how many movies users in the training dataset seem to be watching.



Well, now that we have seen that we will start the actual work.

1. We will implement the ALS model to predict the ratings of users in the testing dataset
2. To get the best parameters that can predict for the model, I varied the rank data and the maxiter parameters

```
predictions = model.transform(testing)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="ratings",predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
```

```
Root-mean-square error = 0.864900801896
```

```
ranks = [6,8,10, 12]
for i in range(len(ranks)):
    print(i)
    als = ALS(maxIter=10,rank=ranks[i], regParam=0.01, userCol="uID", itemCol="mID", ratingCol="ratings",coldStartStrategy="drop")
    model = als.fit(training)
    predictions = model.transform(testing)
    evaluator = RegressionEvaluator(metricName="rmse", labelCol="ratings",predictionCol="prediction")
    rmse = evaluator.evaluate(predictions)
    print("Root-mean-square error = " + str(rmse))
    predictions.show()
```

The best model with a rank of 0.8, maxIter of 10 and a regParam of 0.09 yielded an accuracy score of 0.849.

```
als = ALS(maxIter=10,rank=8, regParam=0.09, userCol="uID", itemCol="mID", ratingCol="ratings",coldStartStrategy="drop")
model = als.fit(training)
predictions = model.transform(testing)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="ratings",predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
predictions.show()
```

```
Root-mean-square error = 0.8491849282
+----+-------+-------+----------+
| mID|    uID|ratings|prediction|
+----+-------+-------+----------+
|2366| 799672|    4.0| 3.3448634|
|2366|1704384|    3.0|  2.838464|
|2366|1172778|    4.0| 2.8995314|
|2366|1312530|    2.0| 2.5765352|
|2366|1684416|    1.0| 2.1967905|
|2366|2360117|    3.0|  3.163447|
|2366| 461110|    2.0| 2.6697888|
|2366| 477387|    1.0| 1.5132037|
|2366|1256465|    3.0|  2.731005|
```

Problem 2

Step 1

I would be attempting to build a better model ie a model whose rate prediction is more likely to be true. But first- some digging.

```
1821
```

```
#finding out the distinct users in the training set
training_df.select('uID').distinct().count()
```

```
28978
```

```
#finding out the distinct movies in the testing set
testing_df.select('mID').distinct().count()
```

```
1701
```

```
testing_df.select('uID').distinct().count()
```

```
27555
```

There are 1821 different movies and 28978 different users in our train dataset, while there are 1701 movies and 27555 different users in the train data

- To determine the best model for this system, I will be finding the estimated average overlap of items for users and the estimated average overlap of users
- To find these, I will be creating a dataframe whose row index is the user, and column is the movies and value represent the rate for that movie by the user or null if there is no rate present

```
#now lets create a new dataframe with userID as index and the movieId as columns for our training data
user_movie_rating = trainingPandas.pivot_table(index='uID', columns='mID', values='ratings')
user_movie_rating.head()
```

mID	8	28	43	48	61	64	66	92	96	111	...	17654	17660	17689	17693	17706	17725	17728	17734	17741	17742
uID																					
7	5.0	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
79	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
199	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
481	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	5.0	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
769	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 1821 columns

-

To find the average for a user with id 1664010, simply select the row that corresponds to this user and sum up all the values in that row and divide by the count of non-zeros.

```
user1664010= list(user_movie_rating.loc[1664010])
avg1664010= get_average(user1664010)
print(avg1664010)
```

```
4.23843648208
```

We will do this for 4 more users and average them and this will be our estimated average overlap of item rated by our 5 selected users. We will do this for the items in our movie_user_rating dataframe and from this we will find our estimated average overlap of users for items.

My Analysis

As expected, the overlap of users that rated items is higher than the overlap of items rated by users. This is because, often a user's rate on an item is influenced mostly by personal preference. ie a user that likes the cartoons is likely to rate the movie Tom and Jerry relatively high. Likewise, in the inverse, a user that hates cartoons is likely to rate the cartoon with a lower rate. the average overall for both is these users is like to be fairly even, However, the overlap of items rated by users will have a combination of very high and very low rates and this is likely to result in an overall high average

Section B -The Selection

- I selected the Matrix factorization approach using SVD simply because of its ability to scale across the large data.
- The similarity measure- the user-user or item-item matrix for our dataset with more than 1000 distinct users would require a lot of computation and memory. This approach considering the size of our data would not be the best option.
- I will be normalizing my dataset because, not only does our testing data contains several nans as a result of not being rated, but also, every time a new user is entered, without any prior rated information, without normalizing the model most likely cannot predict any movie to this new user.
- Finally, my SVD approach is because this approach allows for the dataset to be scaled and appropriately reduces the large dimensions to a smaller set that is easily scalable through

Section C -Normalization

To normalize my data, I found the mean of each user by movie row and subtracted the mean value for each rating

```
] #Now lets create a normalized dataframe from our training set
  normalised_mat =  user_movie_rating - np.asarray([(np.mean(user_movie_rating, 1))]).T
  normalised_mat.head()
```

mID	8	28	43	48	61	64	66	92	96	111	...	17654	17660	17685
uID														
7	4.777046	3.777046	-0.222954	-0.222954	-0.222954	-0.222954	-0.222954	-0.222954	-0.222954	-0.222954	...	-0.222954	-0.222954	-0.222954
79	-0.167490	-0.167490	-0.167490	-0.167490	-0.167490	-0.167490	-0.167490	-0.167490	-0.167490	-0.167490	...	-0.167490	-0.167490	-0.167490
199	-0.153762	-0.153762	-0.153762	-0.153762	-0.153762	-0.153762	-0.153762	-0.153762	-0.153762	3.846238	...	-0.153762	-0.153762	-0.153762
481	-0.176826	-0.176826	-0.176826	-0.176826	-0.176826	-0.176826	-0.176826	-0.176826	-0.176826	4.823174	...	-0.176826	-0.176826	-0.176826

Problem 3

My approach: Step 1

- To ensure my mode, will be predicting correcting, I further split my training data and created a testing set from this.
- Using the SVD model simply consist of factorizing the matrix into smaller dimensions of based on relatedness and using this relatedness to predict the rate of new ones.
- I will be using the SVD model found in the Scipy library
- The data is pivoted before it is passed in the model.

```
preds_df = pd.DataFrame(all_user_predicted_ratgs, columns = train.columns, index= train.index)
preds_df.head()
```

mID	8	28	43	48	61	64	66	92	96	111	...	17654	17660	17689	1
uID															
7	0.034177	0.749242	-0.001626	-0.018925	-0.002373	-0.007050	-0.003369	-0.003599	-0.000042	0.412604	...	0.259888	-0.004119	-0.001759	0.14
79	0.004857	0.812900	0.016973	0.071726	0.013448	0.012980	0.014458	0.015650	0.008345	0.334227	...	0.178450	0.013557	0.009924	0.11
199	0.196017	-0.662838	-0.008799	0.016020	-0.008925	-0.005025	-0.006814	0.001732	-0.001292	0.416214	...	0.008295	-0.014459	-0.006628	-0.04
481	0.060348	1.181981	-0.002816	0.046674	-0.000397	-0.001576	-0.006943	0.004081	-0.011065	0.835635	...	0.224740	-0.004276	-0.003616	0.16
769	0.142250	0.348481	0.006319	0.077472	0.011265	0.008808	0.003678	0.007945	-0.005922	0.117017	...	0.002384	0.008751	0.010659	-0.02

5 rows × 1820 columns

```
U, sigma, Vt = svds(tested_demeaned, k = 50)
```

```
sigma = np.diag(sigma)
```

```
all_user_predicted_ratgs = np.dot(np.dot(U, sigma), Vt) +usertested_ratings_mean.reshape(-1, 1)
```

```
#that didnt seem bad enough
math.sqrt(mean_squared_error(all_user_predicted_ratgs,tested_demeaned))
```

```
0.5307017722376987
```

Once we have the predicted ratings, it is important that we add the mean value back to it. Once that is done we can view our predictions and check how well they performed. The accuracy score above was received from our split, train data set. Now let's try this on our actual testing.

```
finaltested = userTst_movie_rating.as_matrix()
fusertested_ratings_mean = np.mean(finaltested, axis = 1)
ftested_demeaned = finaltested - fusertested_ratings_mean.reshape(-1, 1)
```

```
/usr/local/lib/python2.7/site-packages/ipykernel_launcher.py:1: FutureWarning: Method .as_matrix will be removed in a future version. Use .value
  """Entry point for launching an IPython kernel.
```

```
U, sigma, Vt = svds(ftested_demeaned, k = 50)
sigma = np.diag(sigma)
final_all_user_predicted_ratgs = np.dot(np.dot(U, sigma), Vt) +fusertested_ratings_mean.reshape(-1, 1)
```

ets evaluate the approach for our overall testing data

```
#Using our Mean squared error
math.sqrt(mean_squared_error(final_all_user_predicted_ratgs,ftested_demeaned))
```

```
0.13843667198102874
```

Finally

I tried to predict new movies for a user based on the model.

```
predictions
```

	mID	yearofrelease	title
5745	5760.0	2001.0	The Sopranos: Season 3
9456	9481.0	1972.0	Deliverance
14667	14712.0	2001.0	Tomb Raider
5712	5727.0	1984.0	The Last Starfighter
3625	3638.0	2003.0	Bad Boys II
8827	8851.0	1998.0	Rounders
4748	4761.0	1981.0	The Cannonball Run
7125	7145.0	1984.0	Star Trek III: The Search for Spock
11806	11837.0	1987.0	Three Men and a Baby
11582	11613.0	1974.0	The Man with the Golden Gun

Final Conclusions:

Factorizing the data seems to make the data easily manageable, however, our testing data is predominantly 0 and as such scaling will be relatively easier. It shows significantly as the trained_test data seemed to have had a relatively worse performance than the test data. Also, normalizing seemed to have made it easier to predict for user who only have a 0 ratings.