DSCI 6007 - Distributed and Scalable Engineering

Final Project Report

Topic : Netflix Recommendation System using Collaborative Filtering Approach

Introduction:

Out of the two topics that were given to us for the Final Project – I chose – Netflix Recommendation System Using Collaborative Filtering Approach in ALS.

What is a Recommendation System?

A recommendation system generates a compiled list of items in which a user might be interested, in the reciprocity of their current selection of item(s). It expands users' suggestions without any disturbance or monotony, and it does not recommend items that the user already knows.
For instance, the Netflix recommendation system offers recommendations by matching and searching similar users' habits and suggesting movies that share characteristics with films that users have rated highly.

Figure 1 – Netflix Recommendation Workflow

The recommendation system workflow shown in the diagram above shows the user's collaboration regarding the ratings of different movies or shows. New users get their recommendations based on the recommendations of existing users.

Complications

Recommender systems are machine learning-based systems that scan through all possible options and provides a prediction or recommendation. However, building a recommendation system has the below complications:

- Users' data is interchangeable.
- The data volume is large and includes a significant list of movies, shows, customers' profiles and interests, ratings, and other data points.
- New registered customers use to have very limited information.
- Real-time prediction for users.
- Old users can have an overabundance of information.
- It should not show items that are very different or too similar.
- Users can change the rating of items on change of his/her mind.

Types of Recommendation Systems

There are two types of recommendation systems:

1. Content filtering recommender systems.
2. Collaborative filtering based recommender systems. – We will only focus on this because our project is based on Collaborative Filtering Approach.

Collaborative Filtering:

The idea behind collaborative filtering is to consider users' opinions on different videos and recommend the best video to each user based on the user's previous rankings and the opinion of other similar types of users.

Pros:

- It does not need a movie's side knowledge like genres.
- It uses information collected from other users to recommend new items to the current user.
- **Cons:**
- It does not achieve recommendation on a new movie or shows that have no ratings.
- It requires the user community and can have a sparsity problem.

Different techniques of Collaborative filtering:

1. Non-probabilistic algorithm
    - User-based nearest neighbor.
    - Item-based nearest neighbor.
    - Reducing dimensionality.
2. Probabilistic algorithm
    - Bayesian-network model.
    - EM algorithm.

The underlying assumption of the collaborative filtering approach is that if a person A has the same opinion as a person 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 person chosen randomly. We have Used the Non Probabilistic Algorithm based on User based and Item based Nearest Neighbor using the Alternating Least Square method.

ALS Approach

In this part, we will use the Apache Spark ML Pipeline implementation of Alternating Least Squares, ALS. ALS takes a training dataset (DataFrame) and several parameters that control the model creation process. To determine the best values for the parameters, we will use ALS to train several models, and then we will select the best model and use the parameters from that model in the rest of this lab exercise.

The process we will use for determining the best model is as follows:

1. Pick a set of model parameters. The most important parameter to model is the rank, which is the number of columns in the Users matrix (green in the

diagram above) or the number of rows in the Movies matrix (blue in the diagram above). In general, a lower rank will mean higher error on the training dataset, but a high rank may lead to overfitting. We will train models with ranks of 4, 8, and 12 using the training_df dataset.

2. Set the appropriate parameters on the ALS object:

    3. - The "User" column will be set to the values in our userId Data Frame column.

    4. - The "Item" column will be set to the values in our movieId DataFr ame column.

    5. - The "Rating" column will be set to the values in our rating DataFr ame column.

    - We'll using a regularization parameter of 0.1.

    Note: Read the documentation for the ALS class carefully. It will help you accomplish this step.
6. Have the ALS output transformation (i.e., the result of ALS.fit()) produce a new column called "prediction" that contains the predicted value.
7. Create multiple models using ALS.fit(), one for each of our rank values. We'll fit against the training data set (training_df).
8. For each model, we'll run a prediction against our validation data set (validation_df) and check the error.
9. We'll keep the model with the best error rate.

Why are we doing our own cross-validation? A challenge for collaborative filtering is how to provide ratings to a new user (a user who has not provided any ratings at all). Some recommendation systems choose to provide new users with a set of default ratings (e.g., an average value across all ratings), while others choose to provide no ratings for new users. Spark's ALS algorithm yields a NaN (Not a Number) value when asked to provide a rating for a new user.

Using the ML Pipeline's CrossValidator with ALS is thus problematic, because cross validation involves dividing the training data into a set of folds (e.g., three sets) and then using those folds for testing and evaluating the parameters during the parameter grid search process. It is likely that some of the folds will contain users that are not in the other folds, and, as a result, ALS produces NaN values for those

new users. When the CrossValidator uses the Evaluator (RMSE) to compute an error metric, the RMSE algorithm will return NaN. This will make all of the parameters in the parameter grid appear to be equally good (or bad).

Motivation –

The project is about recommendation of movies on Netflix using the Collaborative Filtering approach. We created a cluster on Amazon EMR and considered the Training datasets to predict and test the recommendation using the Alternating Least Square Method.

:

Why is it important in the real world – Recommender system has the ability to predict whether a particular user would prefer an item or not based on the user's profile. Recommender systems are beneficial to both service providers and users [3]. They reduce transaction costs of finding and selecting items in an online movies.

How is it important for Data Engineering - Recommendation system provides the facility to understand a person's taste and find new, desirable content for them automatically based on the pattern between their likes and rating of different items

Documentation of approach

The Netflix recommendation system basically depends on the User. User plays an important role in it. The User watches the movie and rates them. So all the data from different users and their ratings were stored in the dataset which was further split into the Training Ratings dataset and the Testing Ratings Dataset. We were given these two respective files to work and predict the system along with the movie titles dataset to let us know the name of the movie of a particular movie id.
We preprocessed the data and stored all the data in Dataframes. We made Similar user comparisons and user to user comparisons and made a recommendation system which predicts a particular movie recommendation depending on the users interest.

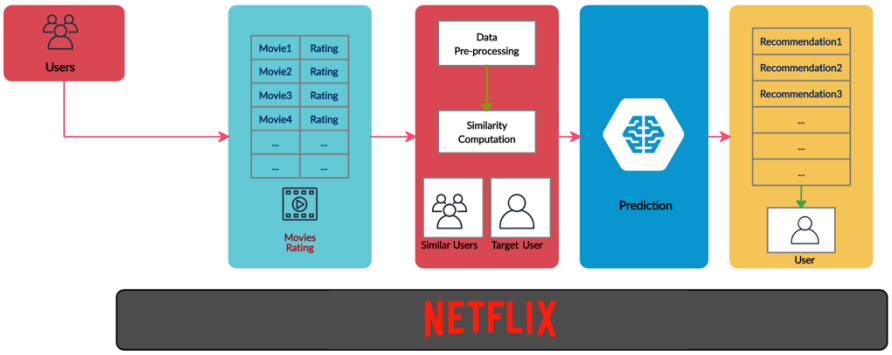


Figure 2: How did we approach the Project

SYSTEM CONFIGURATION:

Step 1: I first created the s3 bucket on AWS and uploaded the datasets.

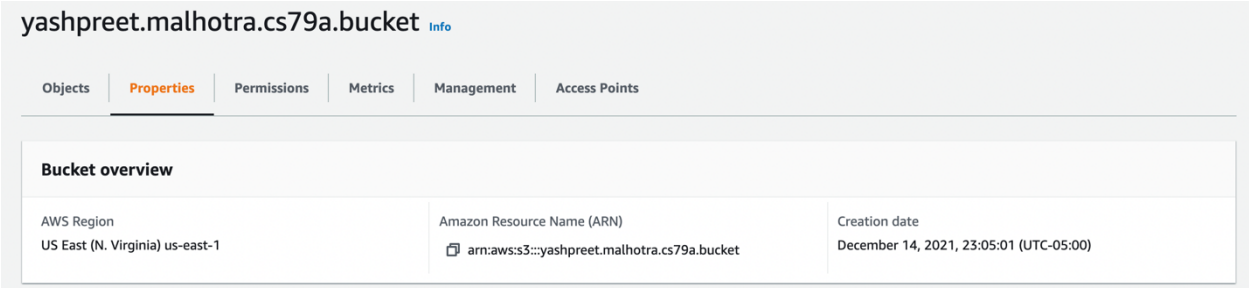
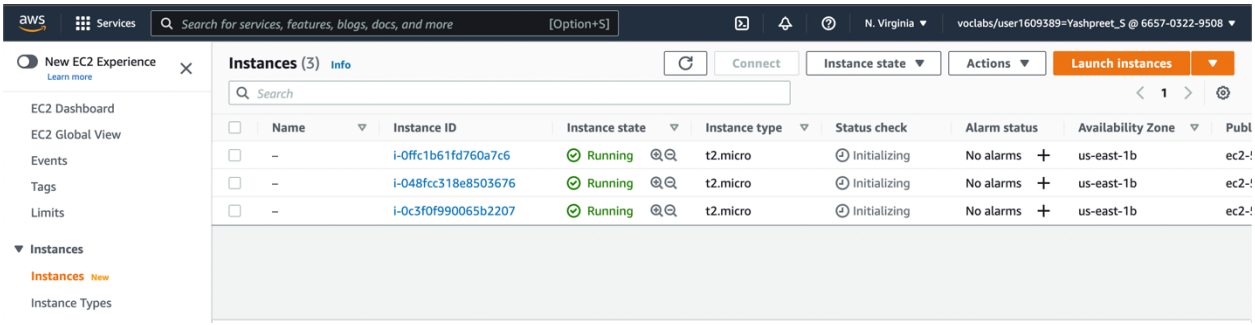


Figure 3: The Amazon s3 Bucket created on AWS

Step 2: I then created an EC2 instance and created a keypair named – Netflix_keypair.pem. I downloaded this pem file onto my system.



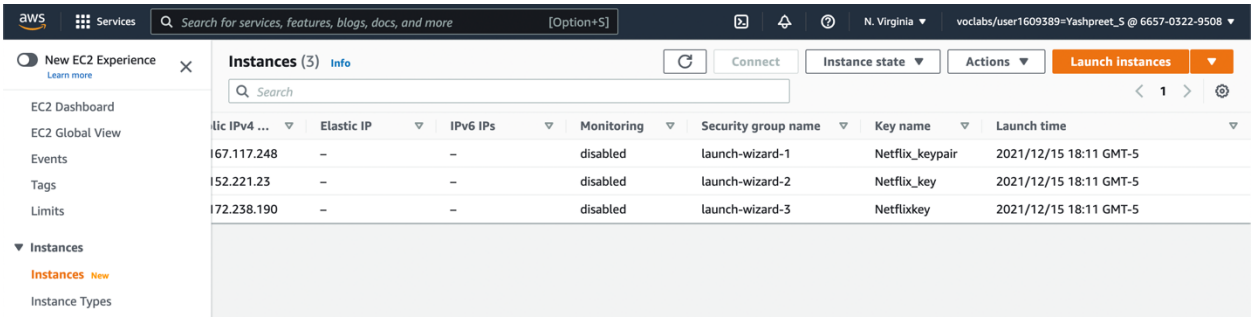


Figure 4: The instance that has the keypair – Netflix_keypair.pem

Step 3: I then created a cluster on Amazon EMR, that may have terminated after the cluster stops working. The cluster that has the final code running is the cluster with the name – yash.

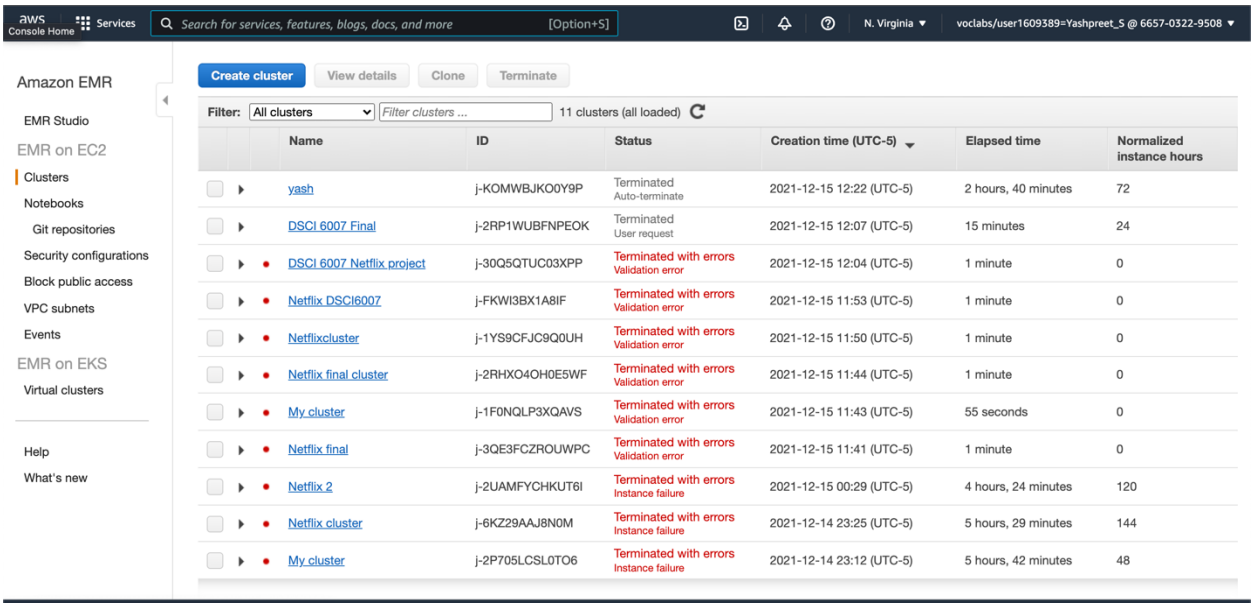


Figure 5: EMR Clusters. The cluster I worked on is the cluster with name yash

Step 4: Click on Create Cluster.

Step 5: Change the name of the cluster, choose the application Spark, the hardware configuration as m4.xlarge, and choose the EC2 instance keypair, in my case it would be Netflix_keypair.pem. Then click on Create Cluster.

Step 6: Give approx. 10-15 mins for the cluster to run. Once the cluster starts running, change the accessibility mode of the pem file on your terminal using the chmod 400 Netflix_keypair.pem.

Step 7: Go to the EC2 instance and in the Master instance add the inbound rules of 8000 port for both IPV4 and IPV6.

Step 8: Then run this command on you terminal on Mac OSX:

ssh -i `/Netflix_key.pem hadoop@ec2-54-211-208-225.compute-1.amazonaws.com.

Now this is for my case. The syntax is :

ssh -i '/your_keypair.pem hadoop@Master node'.

Here you basically are connecting your Master node to SSH.

Step 9: Once the cluster has been connected to your SSH, this is what you see on your terminal.

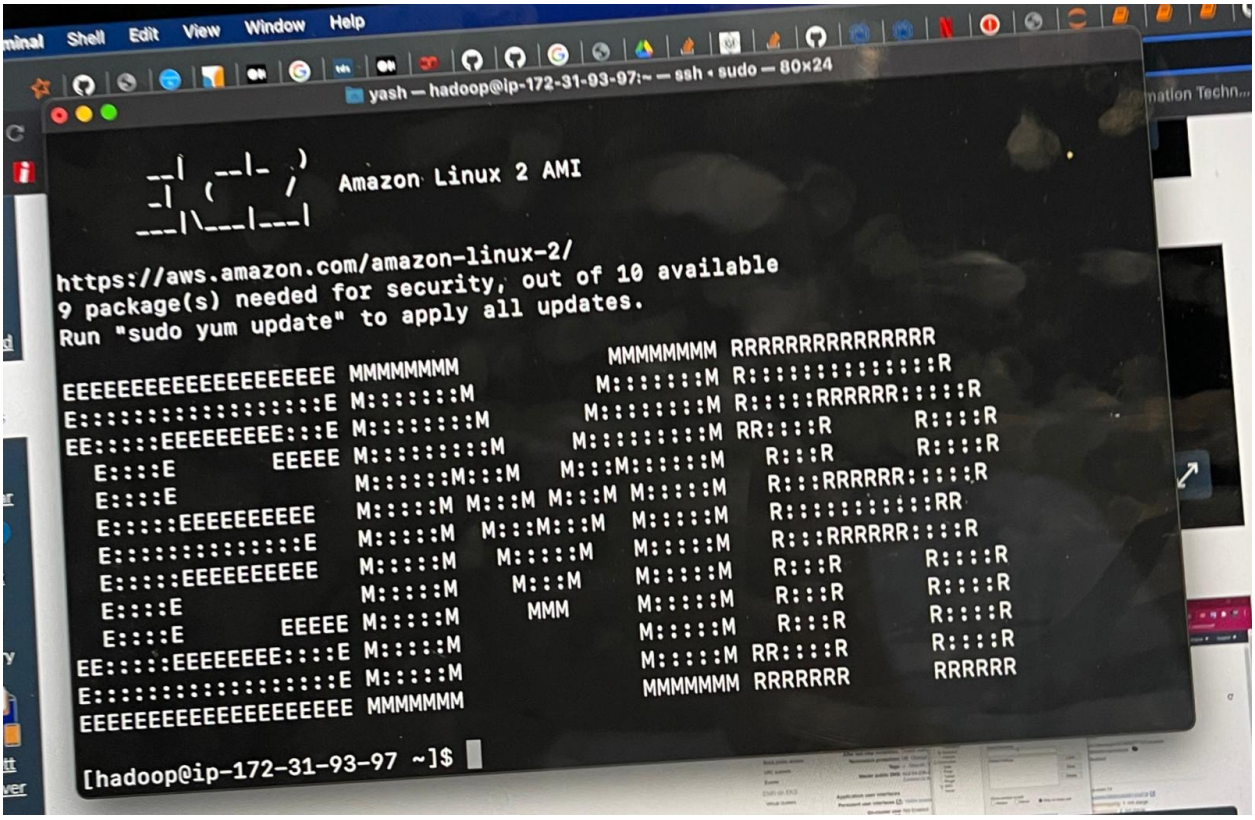


Figure 6: This is a screenshot of the EMR cluster when it runs on your machine.

Step 10: Type in the commands:
sudo yum update
sudo yum install pip
pip install python
python -m pip install jupyter
export PYSPARK_DRIVER_PYTHON = /usr/local/bin/jupyter
export PYSPARK_DRIVER_PYTHON_OPTS = “notebook –no-browser --ip=0.0.0.0 –port=8888
source ~/.bashrc

Step 11: And there you go! Go to your browser type in your Master node address in my case would be ec2-54-211-208-225.compute-1.amazonaws.com followed by the whole address you get when you run the bash src command. And you get you jupyter notebook to work on.

Big data application/dataset:

Description:

Step 1: Here, I called the s3 bucket in which all of the data is stored. In the Amazon S3 bucket, I named the bucket as – 'yashpreet.malhotra.cs79a.bucket' and uploaded the three respective datasets – TrainingRatings.txt, TestingRatings.txt and movie_titles.txt.

```
import os

# Change to the location of data files
dbfs_dir = 's3://yashpreet.malhotra.cs79a.bucket'
movie_titles_filename = dbfs_dir + '/movie_titles.txt'
TrainingRatings_filename = dbfs_dir + '/TrainingRatings.txt'
TestingRatings_filename = dbfs_dir + '/TestingRatings.txt'
```

Figure 7: Calling the dataset from the Amazon S3 Bucket

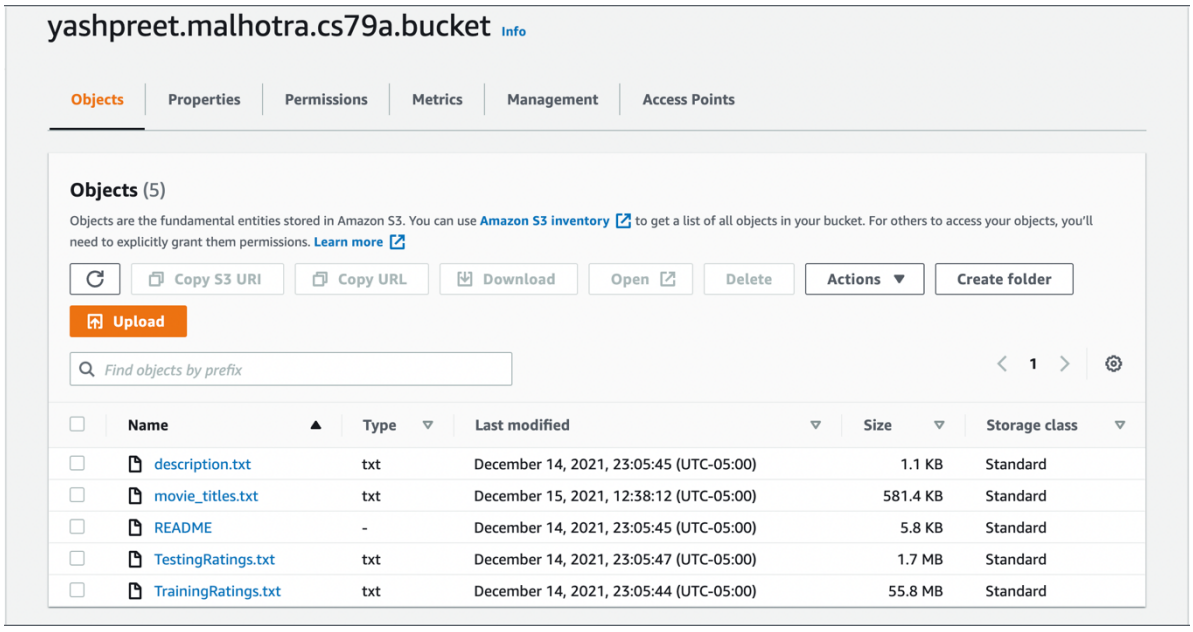


Figure 8: The datasets uploaded on S3 bucket

Step 2 – The next step I did was to change the datatypes of the columns that had strings because the recommender doesn't understand Strings or object datatype. Thus we change the schemas for all the three data sets.

```
from pyspark.sql.types import *

movie_titles_df_schema = StructType(
  [StructField('ID', IntegerType()),
   StructField('movie_year', IntegerType()),
   StructField('movie_title', StringType())]
)
TrainingRatings_df_schema = StructType(
  [StructField('movie_Id', IntegerType()),
   StructField('user_Id', IntegerType()),
   StructField('movie_ratings', FloatType())]
)
TestingRatings_df_schema = StructType(
  [StructField('movie_Id', IntegerType()),
   StructField('user_Id', IntegerType()),
   StructField('movie_ratings', FloatType())]
)
```

Figure 9: Changing the datatypes of the columns of each dataset that contains string / object.

Step 3 - Here, next was to display the count of the movie ratings in both the Training and the Testing datasets which are 3,255,352 ratings in the Training dataset and 100,478 in the testing dataset. Along with this we even counted the number of movie titles in the movie_titles dataset that is 17770. I thus converted all the txt files into pandas dataframes and since it's a huge data, I am displaying only top 3 rows using the head() command.

```
There are 17770 movietitle, 3255352 trainingRatings and 100478 testingRatings in the datasets
Movie_titles:
+---+----------+--------------------+
| ID|movie_year|         movie_title|
+---+----------+--------------------+
|  1|      2003|     Dinosaur Planet|
|  2|      2004|Isle of Man TT 20...|
|  3|      1997|           Character|
+---+----------+--------------------+
only showing top 3 rows

TrainingRatings:
+--------+-------+-------------+
|movie_Id|user_Id|movie_ratings|
+--------+-------+-------------+
|8       |1744889|1.0          |
|8       |1395430|2.0          |
|8       |1205593|4.0          |
+--------+-------+-------------+
only showing top 3 rows

TestingRatings:
+--------+-------+-------------+
|movie_Id|user_Id|movie_ratings|
+--------+-------+-------------+
|8       |573364 |1.0          |
|8       |2149668|3.0          |
|8       |1089184|3.0          |
+--------+-------+-------------+
only showing top 3 rows
```

Figure 10: Storing the datasets on a Pandas Dataframe and displaying it along with the counts respectively.

Step 4 – The next step was to display the movie id's of all the movie that one user has rated and what exactly didh he rate. So the top 20 rows are displayed and the user id selected was – 2149668. The ratings given by the user ranges from 1.0 to 5.0 as shown in figure ()

And thus, we did the exact alternate thing here. By keeping the Movie ID constant – 992, we checked how many different users rated for this movie id and what was their rating. The rating ranges from 1.0 to 5.0 as shown in figure ().

```
+--------+-------+-------------+
|movie_Id|user_Id|movie_ratings|
+--------+-------+-------------+
|     992|2149668|          3.0|
|    1202|2149668|          3.0|
|    1289|2149668|          1.0|
|    1305|2149668|          3.0|
|    2015|2149668|          5.0|
|    2212|2149668|          3.0|
|    2342|2149668|          4.0|
|    2601|2149668|          3.0|
|    2675|2149668|          3.0|
|    2755|2149668|          3.0|
|    2913|2149668|          5.0|
|    2955|2149668|          5.0|
|    3151|2149668|          4.0|
|    3253|2149668|          4.0|
|    3274|2149668|          2.0|
|    3290|2149668|          5.0|
|    3355|2149668|          5.0|
|    3538|2149668|          4.0|
|    4847|2149668|          3.0|
|    4849|2149668|          3.0|
+--------+-------+-------------+
only showing top 20 rows
```

Figure 11-a: Display different movie ratings based on a particular user.

```
+--------+-------+-------------+
|movie_Id|user_Id|movie_ratings|
+--------+-------+-------------+
|     992| 306466|          3.0|
|     992| 765331|          4.0|
|     992|  41412|          3.0|
|     992|1331887|          3.0|
|     992|1276913|          3.0|
|     992| 887273|          4.0|
|     992|1663216|          3.0|
|     992|1778851|          5.0|
|     992|2630287|          5.0|
|     992| 530441|          5.0|
+--------+-------+-------------+
only showing top 10 rows
```

Figure 11-b: Display different movie ratings of a particular movie based on different users.

Step 5: The next step was to take the average of the ratings on a particular movie.

```
movie_ids_with_avg_ratings_df:

[Stage 25:============================>                            (4 + 4) / 8]21/12/15 18:51:05 WARN Utils: Truncated the string representation of a plan since it was too large. This behavior can be adjusted by setting 'spark.debug.maxToStringFields' in SparkEnv.conf.

+-------+-----+------------------+
|user_Id|count|average           |
+-------+-----+------------------+
|128389 |42   |3.0238095238095237|
|2250628|25   |3.24              |
|279120 |24   |3.5416666666666665|
+-------+-----+------------------+
only showing top 3 rows
```

Figure 12: Average Ratings of various movies of a particular user.

The Same thing was done again, by keeping the movie_id constant, that is 8, and based on that how many users rated that movie with movie_id = 8 and what was their rating.

movie_Id	user_Id	movie_ratings
8	1744889	1.0
8	1395430	2.0
8	1205593	4.0
8	1488844	4.0
8	1447354	1.0
8	306466	4.0
8	1331154	4.0
8	1818178	3.0
8	991725	4.0
8	1987434	4.0
8	1765381	4.0
8	433803	3.0
8	1148143	2.0
8	1174811	5.0
8	1684516	3.0
8	754781	4.0
8	567025	4.0
8	1623132	4.0
8	1567095	3.0
8	1666394	5.0

only showing top 20 rows

Figure 13: Display the ratings of various users on a particular movie

Step 6: The next step was to check the count of all the movies that each user has rated as in Figure ()

The same thing is done on an alternative basis, where the count is displayed of all the users that rated a particular movie as in Figure ().

```
user_Id
305344     1757
387418     1744
2439493    1640
1664010    1535
2118461    1481
Name: movie_ratings, dtype: int64
```

Figure 14-a: Count of all movies each user has rated.

```
movie_Id
6971    25468
6287    24393
4640    23525
9728    23184
8596    23005
Name: movie_ratings, dtype: int64
```

Figure 14-b: Count of all the movies that the user has rated.

Step 7: The next step is to create pivot tables for both the movie_id and the user_id respectively. Additionally, a train test split is performed on the Training dataset so that we can apply the Alternating Least Square method to predict the model.

```
Training: 2439360, validation: 815992

+--------+-------+-------------+
|movie_Id|user_Id|movie_ratings|
+--------+-------+-------------+
|       8|      7|          5.0|
|       8|   1333|          3.0|
|       8|   3363|          2.0|
+--------+-------+-------------+
only showing top 3 rows

+--------+-------+-------------+
|movie_Id|user_Id|movie_ratings|
+--------+-------+-------------+
|       8|   3321|          1.0|
|       8|   7284|          5.0|
|       8|   9321|          3.0|
+--------+-------+-------------+
only showing top 3 rows
```

Figure 15: The Train test split on the Training Dataset

Step 8: The model is then predicted after the split using the ALS technique with which we compute the Root Mean Square Error and the Mean Absolute Error. We calculate these errors to check the accuracy for the predictions done by the model.

```
als = ALS(maxIter=10, regParam=0.5, userCol="user_Id",
                      itemCol = "movie_Id", ratingCol = "movie_ratings", coldStartStrategy = "drop")

model = als.fit(train_df)
#Generating Predictions
prediction = model.transform(validation_df)
evaluate1 = RegressionEvaluator(metricName="mse", labelCol="movie_ratings",
                                predictionCol="prediction")

evaluate2 = RegressionEvaluator(metricName="rmse", labelCol="movie_ratings",
                                predictionCol="prediction")



rmse = evaluate1.evaluate(prediction)
mse = evaluate2.evaluate(prediction)

print('The model had a MSE on the test set of {0}'.format(mse))
print('The model had a RMSE on the test set of {0}'.format(rmse))
```

```
[Stage 157:==============================================>      (174 + 8) / 200]

The model had a MAE on the test set of 1.0336890583191667
The model had a RMSE on the test set of 1.068513069288766
```

Figure 16: The ALS MODEL with its Root Mean Square Error and Mean Absolute Error.

Step 9: Similarly we do this on the Test dataset, to check if the model is Overfitting or underfitting. As we can see, that the bias and the variance are equally trade-off, thus our model fits the data normally.

```
##TestData
predict_df = model.transform(TestingRatings_df)

# Remove NaN values from prediction (due to SPARK-14489)
predicted_test_df = predict_df.filter(predict_df.prediction != float('nan'))

# Run the previously created RMSE evaluator, reg_eval, on the predicted_test_df DataFrame
test_MAE = evaluate1.evaluate(predicted_test_df)
test_RMSE = evaluate2.evaluate(predicted_test_df)

print('The model had a MAE on the test set of {0}'.format(test_MAE))
print('The model had a RMSE on the test set of {0}'.format(test_RMSE))
```

```
[Stage 217:=============================================>        (168 + 8) / 200]
The model had a MAE on the test set of 1.0741172384754827
The model had a RMSE on the test set of 1.0363962748270963
```

Figure 17: Prediction on the Testing Dataset using the ALS Approach

Step 10: Now the model recommends movies based on user's interests. Here the model recommends the list of movies for each user based on what the user rated previous movies. Here the model gives a list of recommendation based on movie-id's. Here the model is recommending what kind of users will be recommended for each and every movie.

```
+-------+--------------------+
|user_Id|     recommendations|
+-------+--------------------+
|    481|[[3033, 4.833468]...|
|   2678|[[3033, 3.6789656...|
|   3595|[[3033, 3.377369]...|
|   6460|[[3033, 3.994896]...|
|   7284|[[3033, 4.2301435...|
|   7576|[[3033, 4.207594]...|
|   9597|[[3033, 3.633531]...|
|  15191|[[3033, 4.096875]...|
|  15846|[[3033, 4.4358387...|
|  20461|[[3033, 3.6945784...|
|  20774|[[3033, 3.4982104...|
|  26258|[[3033, 3.945142]...|
|  27608|[[3033, 4.0977364...|
|  28346|[[3033, 3.7633572...|
|  30941|[[3033, 3.856298]...|
|  30976|[[3033, 4.1473007...|
|  31203|[[3033, 3.6661737...|
|  36822|[[3033, 4.1111326...|
|  40851|[[3033, 4.0672035...|
|  41068|[[3033, 3.1798801...|
+-------+--------------------+
only showing top 20 rows
```

Figure 18-a: Movie Recommendations done by the model for each user.

```
+--------+--------------------+
|movie_Id|     recommendations|
+--------+--------------------+
|     481|[[1482568, 3.8480...|
|    6911|[[1482568, 4.2477...|
|   11041|[[1482568, 3.9991...|
|   15841|[[1482568, 4.6041...|
|   15051|[[1482568, 4.6994...|
|    8851|[[1482568, 4.5978...|
|    1061|[[1482568, 4.3810...|
|   16283|[[1482568, 3.2875...|
|   12293|[[1482568, 5.5656...|
|   12273|[[1482568, 3.7682...|
|   12755|[[1482568, 4.9801...|
|   12705|[[1482568, 3.5804...|
|    2675|[[1482568, 3.1100...|
|    5465|[[1482568, 3.7763...|
|   10845|[[1482568, 3.7558...|
|    8825|[[1482568, 4.5972...|
|   13965|[[1482568, 3.8801...|
|    5025|[[1482568, 4.7460...|
|   11317|[[1482568, 3.9508...|
|    9517|[[1482568, 4.1833...|
+--------+--------------------+
only showing top 20 rows
```

Figure 18-b: User Recommendations on each movie

Step 11: I thus add myself as a user and rate movies for respective movie_ids based on my interests and thus the model predicts the movies based on my ratings for previous movies.

```
My movie ratings:

DataFrame[user_Id: bigint, movie_Id: bigint, movie_ratings: bigint]

+-------+--------+-------------+
|user_Id|movie_Id|movie_ratings|
+-------+--------+-------------+
|      1|   12293|            3|
|      1|    2290|            2|
|      1|   11812|            5|
|      1|   25468|            4|
|      1|   10947|            5|
|      1|   14648|            3|
|      1|   14185|            5|
|      1|   11088|            4|
|      1|    5326|            4|
|      1|    3928|            2|
+-------+--------+-------------+
```

Figure 19: My Movie Ratings

CONCLUSION:

Over the years, Machine learning has solved several challenges for companies like Netflix, Amazon, Google, Facebook, and others. The recommender system for Netflix helps the user filter through information in a massive list of movies and shows based on his/her choice. A recommender system must interact with the users to learn their preferences to provide recommendations.

Collaborative filtering (CF) is a very popular recommendation system algorithm for the prediction and recommendation based on other users' ratings and

collaboration. User-based collaborative filtering was the first automated collaborative filtering mechanism.

Future Work: So as I took only me as a user and predicted my movies of my interest, in future I will try to develop a UI which gets users of different types and predict their recommendations accurately.