Movie Recommender Engine

Guided by Dr. Jianwu Wang

GitHub Repository Link -
https://github.com/nagasushmitha-macheri/Movie-Recommender-System---Apache-Spark

Akansha Singh
QK90578
qk90578@umbc.edu

Ashish Vishwanathan
BL29932
ashishv1@umbc.edu

Naga Sushmitha Macheri
KC79893
nagasum1@umbc.edu

There has been a surge in the number of people subscribing to online streaming services such as Netflix, Amazon Prime and Hulu. These services offer a variety of movies and TV shows to its subscribers so that they can stream it anywhere. Due to the increase in demand, it is a necessity that these streaming services have a strong recommendation engine that recommends the users movies based on what they have watched previously or how they have rated a movie. A movie recommendation engine is a system that identifies and provides recommended content to users that they might like. For this Independent Study, we have chosen to develop such a recommendation engine using Collaborative Filtering. This report contains our results.

Introduction:

Most of the web products that we use today are driven by recommender systems. Be it Amazon, YouTube, Pinterest, or Netflix, every website has a “recommended for you” section. These companies use recommendation engines to target customers with specific products based on their previous purchases or previously watched or the rating they might have given to a previously bought product or previously watched movie. These recommendation engines have been proven to add value to businesses. In streaming services, recommendation engines are used to suggest customers with new movies based on their previous activities which means that the probability of the user liking the recommended movie is very high. Therefore, companies like Netflix, Amazon Prime and Hulu have a very high financial motive to build a recommendation that is as accurate as possible.

Apache Spark:

Apache Spark is an open-source software which is a fast and a general engine for large scale data processing. Spark offers many high-level operators which makes it very easy to build apps in parallel using Java, Scala, R and Python. It utilizes in-memory caching and executes query in an optimized way so that analytic queries run faster for data of any size. It supports code reuse of code across multiple workloads like batch processing, interactive queries, real-time analytics, machine learning and graph processing. Many organizations from various industries are using Spark.

Dataset:

Link to the dataset - https://github.com/databricks/spark-training/tree/master/data/movielens/medium

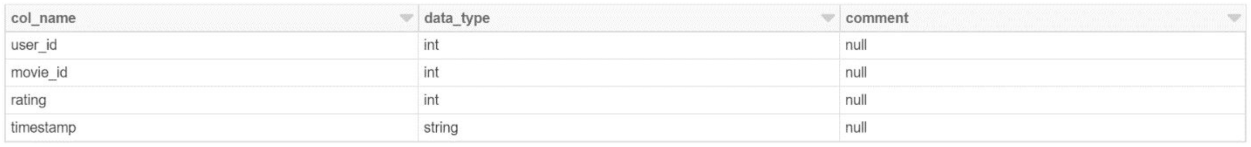
For this project, we have chosen to implement a recommendation engine using the Spark Training Dataset given by MovieLens.

These files contain 1,00,209 anonymous ratings for approximately 3900 movies given by 6040 users.

The dataset has 3 files:

Ratings.dat –

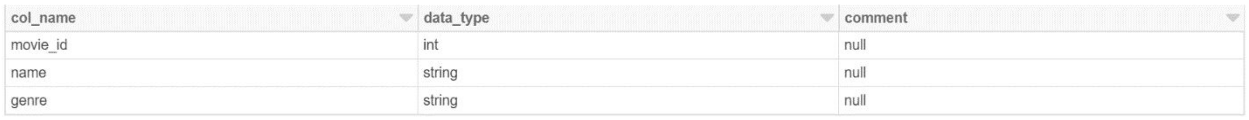
UserID::Movieid::Rating::Timestamp



col_name	data_type	comment
user_id	int	null
movie_id	int	null
rating	int	null
timestamp	string	null

Movies.dat-

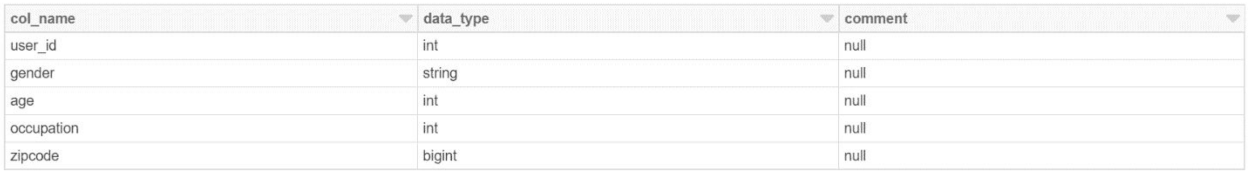
MovieID::Title::Genres



col_name	data_type	comment
movie_id	int	null
name	string	null
genre	string	null

Users.dat-

UserID::Gender::Age::Occupation::Zip-code



col_name	data_type	comment
user_id	int	null
gender	string	null
age	int	null
occupation	int	null
zipcode	bigint	null

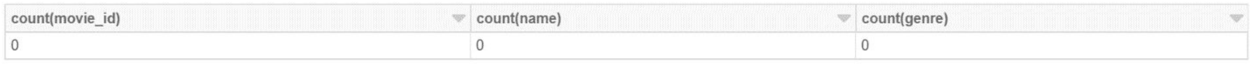
Environment:

For this project, we have decided to implement our code in the community edition environment provided by Databricks.

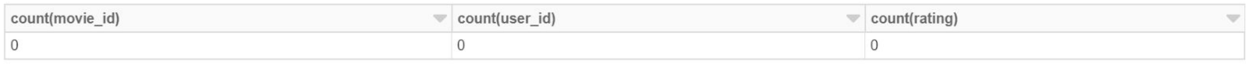
Data Preprocessing:

Some of the fields like Movie_ID, User_ID and age were of **String** datatype. We have casted it as **Integer** as it deemed appropriate. Since Ratings were whole numbers from 1-5 we casted it as **Integer** type instead of **Float**.

Further we checked if there are any missing movie_id or ratings and found that there weren't any missing values in the dataset.



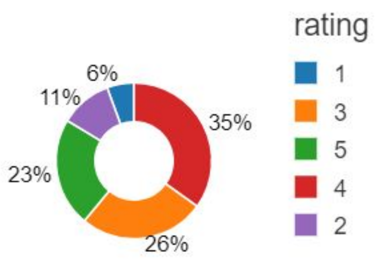
count(movie_id)	count(name)	count(genre)
0	0	0



count(movie_id)	count(user_id)	count(rating)
0	0	0

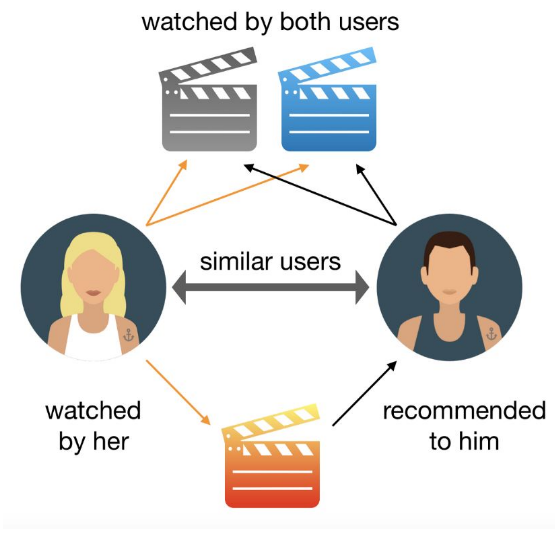
Exploratory analysis:

This pie chart tells us that the users have given 4 ratings for most of the movies.



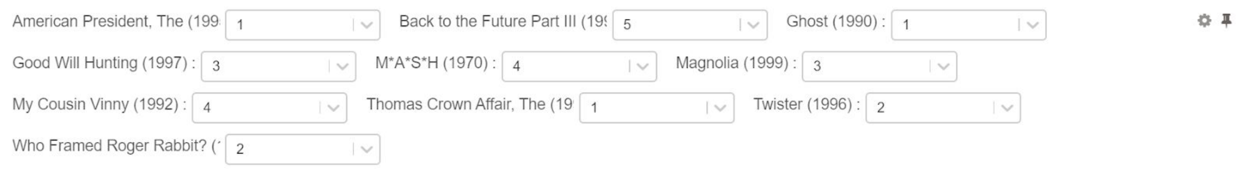
Algorithm and Implementation:

We are using Collaborative Filtering, a common technique used for recommendation. Collaborative filtering aims to fill any missing entries in the user-item matrix. In Spark, both users and products are described by a set of latent factors that can be used to predict missing values. Spark uses Alternating Least Squares method to learn these factors. Collaborative filtering is a method of predicting the interests of a user by learning about their interests with a set of items in the past and interests of others with the same set of items. The basic idea of CF is that if one user has liked movies A, B and C and another user has liked movies B, C and D, there is a high possibility that the second user will like movie A and the first user will like movie D.



In our project, we have used the times_rated which is the number of times a movie has been rated to build the association matrix. We have then used Alternating Least Squares(ALS) method to fill up the missing values i.e. movies with no users and users with no movies.

We have used Databricks Widgets to make it more interactive and easier for the user to give their personalized ratings to the movies. After the user gives their ratings, we have processed the ratings and predicted movies based on these ratings.

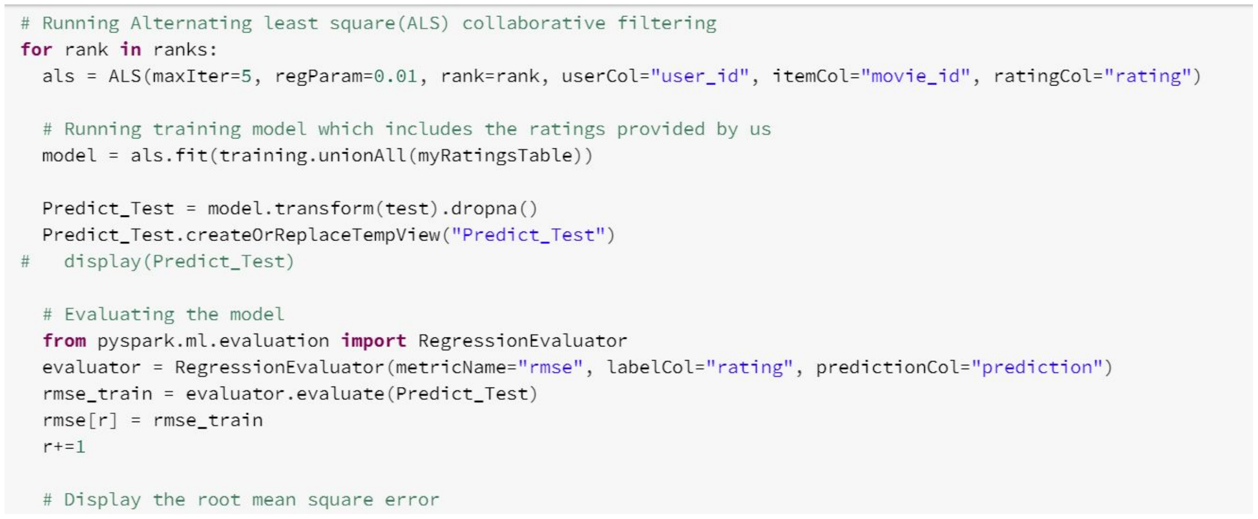


American President, The (199 : 1 | Back to the Future Part III (19! 5 | Ghost (1990) : 1
Good Will Hunting (1997) : 3 | M*A*S*H (1970) : 4 | Magnolia (1999) : 3
My Cousin Vinny (1992) : 4 | Thomas Crown Affair, The (19 1 | Twister (1996) : 2
Who Framed Roger Rabbit? (' 2

After processing these new ratings, we have split the dataset into training and test dataset with 8:2 ratio.

We then trained the model using ALS() from the PySpark library. We tuned a few hyperparameters like maxIter, which is the maximum number of iterations to run, regParam, which specifies the regularization parameter in ALS, and rank, which is the number of latent factors.

We have used Root Mean Square Error(RMSE) to determine the accuracy of the model. We have trained our model using different values for ranks and each time we got a different rmse value.



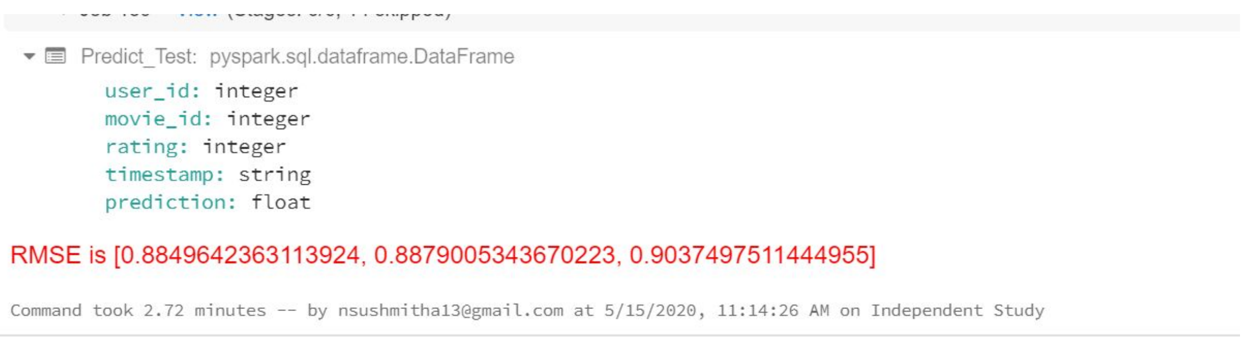
```
# Running Alternating least square(ALS) collaborative filtering
for rank in ranks:
  als = ALS(maxIter=5, regParam=0.01, rank=rank, userCol="user_id", itemCol="movie_id", ratingCol="rating")

  # Running training model which includes the ratings provided by us
  model = als.fit(training.unionAll(myRatingsTable))

  Predict_Test = model.transform(test).dropna()
  Predict_Test.createOrReplaceTempView("Predict_Test")
#   display(Predict_Test)

  # Evaluating the model
  from pyspark.ml.evaluation import RegressionEvaluator
  evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating", predictionCol="prediction")
  rmse_train = evaluator.evaluate(Predict_Test)
  rmse[r] = rmse_train
  r+=1

  # Display the root mean square error
```

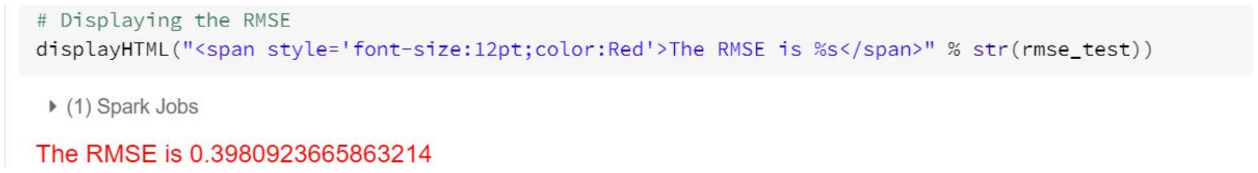


```
Predict_Test: pyspark.sql.dataframe.DataFrame
    user_id: integer
    movie_id: integer
    rating: integer
    timestamp: string
    prediction: float

RMSE is [0.8849642363113924, 0.8879005343670223, 0.9037497511444955]

Command took 2.72 minutes -- by nsushmitha13@gmail.com at 5/15/2020, 11:14:26 AM on Independent Study
```

We finally ran this model on our test data to get an rmse value of 0.39.



```
# Displaying the RMSE
displayHTML("<span style='font-size:12pt;color:Red'>The RMSE is %s</span>" % str(rmse_test))

(1) Spark Jobs

The RMSE is 0.3980923665863214
```

Reflection:

Initially, we were trying to install Apache Spark on our local machine, but we faced many difficulties. In every step, there was an error like Java version error, a Java path error, etc. After looking up solutions for these on the internet, we were unsuccessful in solving them and finally decided to use the community edition offered by Databricks as our environment.

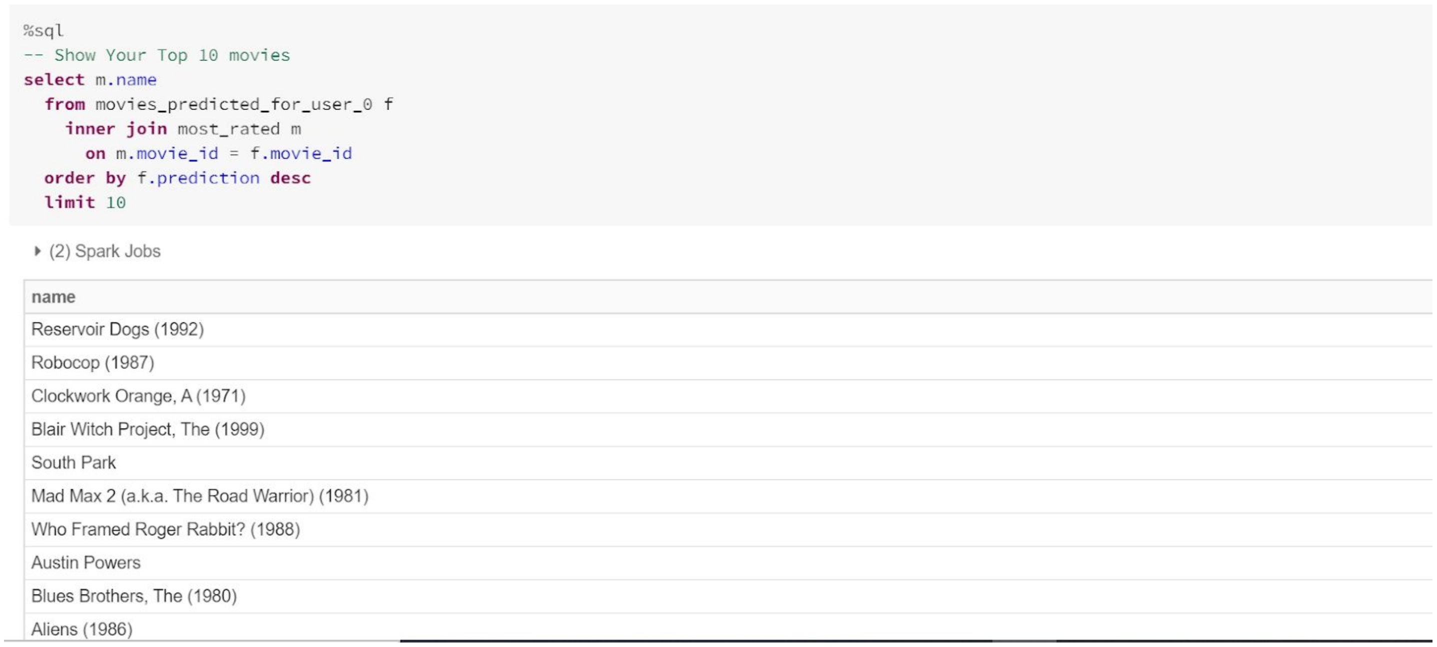
When we were trying to load our dataset on the Databricks environment, we were unaware that it allows only one character as a delimiter. Since our dataset had 2 character delimiter (::), the table being loaded was taking the second colon as a separate column. We decided to replace :: with : using the 'find and replace' feature of notepad.

Our initial dataset was the Netflix Prize Data, that had approximately 80 million rows for ratings, for over 17000 movies given by more than 500,000 customers. During implementation of the algorithm, it came to light that our train and test data should contain at least 1 instance of the same user_id. We learnt that ALS cannot make predictions on unseen user_ids. Since our dataset was very sparse, it was possible that there was no overlap of user_ids in the train set and the user_ids in the test set. We then decided to move forward with the Spark Training Dataset.

Conclusion:

This movie recommender system uses alternating least squares(ALS) on the MovieLens dataset. Since we had limited processing power, we used a maximum of 5 iterations with regularization parameters set to 0.01. The regularization parameter helps to reduce the variance and increases the bias to our estimated regularization parameters thus avoiding overfitting. For the test dataset, we got an rmse value of 0.39 which means that our predictions are 39% variation from the actual ratings.

These are our final recommendations for user 0 based on their ratings that we retrieved from the widget -



```
%sql
-- Show Your Top 10 movies
select m.name
  from movies_predicted_for_user_0 f
    inner join most_rated m
      on m.movie_id = f.movie_id
  order by f.prediction desc
  limit 10
```

▸ (2) Spark Jobs

name
Reservoir Dogs (1992)
Robocop (1987)
Clockwork Orange, A (1971)
Blair Witch Project, The (1999)
South Park
Mad Max 2 (a.k.a. The Road Warrior) (1981)
Who Framed Roger Rabbit? (1988)
Austin Powers
Blues Brothers, The (1980)
Aliens (1986)

References:

1. Dataset
   https://github.com/databricks/spark-training/tree/master/data/movielens/medium

2. Collaborative Filtering
   https://spark.apache.org/docs/latest/ml-collaborative-filtering.html

3. Spark MLlib
   https://www.edureka.co/blog/spark-mllib/

4. What is rank in ALS
   https://www.howtobuildsoftware.com/index.php/how-do/iGP/algorithm-machine-learning-apache-spark-mllib-what-is-rank-in-als-machine-learning-algorithm-in-apache-spark-mllib

5. How do you build a “People who bought this also bought that” -style recommendation engine
   https://datasciencemadesimpler.wordpress.com/tag/alternating-least-squares/

6. Image for Collaborative Filtering
   https://www.bing.com/images/search?view=detailV2&ccid=JNYIml9T&id=602E646C49F396C30535B3C9712D6D6B49FC5218&thid=OIP.JNYIml9TfBXCr51-gTwOogHaHG&mediaurl=https%3a%2f%2fcdn-images-1.medium.com%2fmax%2f1200%2f1*x8gTiprhLs7zflmEn1UjAQ.png&exph=1150&expw=1200&q=System+Recomendation+Modeling+Filtering+Collaborative&simid=608051387548895658&selectedIndex=5&ajaxhist=0

7. Intro to Recommender Systems
   https://www.ethanrosenthal.com/2015/11/02/intro-to-collaborative-filtering/

8. Slides provided by Dr. Wang.