CS-GY 6513-I:Big Data

MOVIE RECOMMENDATION SYSTEM

Name: Ratnam Srivastava
N Number: N14670882
NetID: rs6436
Email: rs6436@nyu.edu

Name: Mohammed Fareed Uddin
N Number: N16671975
NetID: mfu212
Email: mfu212@nyu.edu

Name: Arjan Singh Narula
N Number: N15292730
NetID:asn419
Email:asn419@nyu.edu

Content

Introduction

The Project focus on Recommending Movies to the upcoming or new User by considering the previous preferences of the Users. We are using Collaborative Filtering technique with Spark's Alternating Least Squares which give automatic recommendation of interest of a user by collecting taste or preferences from many users.

There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on Google.

Often, these systems are able to collect information about users choices and can use this information to improve their suggestions in the future. For example, Facebook can monitor your interaction with various stories on your feed in order to learn what types of stories appeal to you. Sometimes, the recommender systems can make improvements based on the activities of a large number of people. For example, if Amazon observes that a large number of customers who buy the latest Apple Macbook also buy a USB-C-to USB Adapter, they can recommend the Adapter to a new user who has just added a Macbook to his cart.

Due to the continuous improvement in these system, users constantly expect good recommendation. For example, if a music player is not able to suggest song which user might like, than user might simply stop using it. So, that's why these engine plays major role in industries.

The major problem with these engine you cannot move on with the single prediction. As user might have different demands at different day like sometimes he is interested in listening the happy music or sometimes sad music. These swings in user emotions make the prediction of these engine more Complex.

There are two possible methods with which we can build Recommendation Engine.The first one is content-based filtering, where we try to profile the users interests using information collected, and recommend items based on that profile. The other is collaborative filtering, where we try to group similar users together and use information about the group to make recommendations to the user.

Algorithms Used:

Collaborative Filtering

Collaborative filtering is based on the assumption that people who agreed in the past will agree in the future, and that they will like similar kinds of items as they liked in the past.The system generates recommendations using only information about rating profiles for different users or items. By locating peer users/items with a rating history similar to the current user or item, they generate recommendations using this neighborhood.

A key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself. Many algorithms have been used in measuring user similarity or item similarity in recommender systems. For example, the K Nearest Approach and Pearson Correlation.

Dataset Description and Collection

We have collected the dataset from the GroupLens Research.The author of dataset has collected data over various period of time depending on the size of the dataset. The first dataset consist of 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users.The other dataset consist of 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. Each file of dataset consist of different folders of ratings, movies etc. So, at first we extract them into each individual folder so that we can use each later file for our recommendation.

Libraries Used

We have used Spark Ml **Alternating Least Square** algorithm to train our model. The parameters used for training our model are:

- *numBlocks* is the number of blocks used to parallelize computation (set to -1 to auto-configure).
- *rank* is the number of features to use (also referred to as the number of latent factors).
- *iterations* is the number of iterations of ALS to run. ALS typically converges to a reasonable solution in 20 iterations or less.
- *lambda* specifies the regularization parameter in ALS.
- *implicitPrefs* specifies whether to use the *explicit feedback* ALS variant or one adapted for *implicit feedback* data.
- *alpha* is a parameter applicable to the implicit feedback variant of ALS that governs the *baseline* confidence in preference observations.

Analysis & Methodology

Our analysis happened in distinct stages which we have explained below.

- **Data Collection**:
  We have taken the data from the GroupLens research which have taken the data from the MovieLens site. They have collected data ver various period of time. We have used two dataset. The first one is small dataset which is used for the purposes finding out the best parameters for the training purposes. The other one is the large dataset which is used for the purposes of training and testing. It consist of 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users.

- **Data Manipulation and Loading:**
  Both dataset consist of multiple folder such as ratings, movies etc. So, at first we extract them into each individual folder so that we can use it for our recommendation engine. We have dropped the timestamp column from our dataset as it doesn't have major impact on our recommendation.

- **Training and Building Model:**
  We have used the Spark Ml library Alternating Least Squares for the training purposes. This library uses the collaborative filtering technique in which users and products are described by a small set of latent factors that can be used to predict missing entries. After that we apply Spark Ml library to our smaller dataset first to get the best hyper parameters which can be used for training our model. After finding out the best hyper parameters we apply these parameters for training our large dataset which will be used for giving recommendation to new or upcoming users.

Data Analysis

The data we used was from Movie Lens dataset were messy and the column names were not consistent among different files for the same dataset.
Moreover, we had to deal with nulls, nan and missing values. All of them were represented differently, for e.g., nulls were represented as “nan”, “*”, “/”. The preprocessing included:

i) **Detecting columns with the necessary information:**
In our dataset, For each line in the rating dataset, We drop the *timestamp* because we do not need it for this recommender.
We also drop the genres column from the movie dataset as it not needed for this recommender

ii)**Loading the raw ratings data:**
Here we filter out the header which is included in each file.

```
small_ratings_data = small_ratings_raw_data.filter(lambda line:
line!=small_ratings_raw_data_header)\
    .map(lambda line: line.split(",")).map(lambda tokens: (toke
ns[0],tokens[1],tokens[2])).cache()
```

```
small_ratings_file = os.path.join(datasets_path, 'ml-latest-sma
ll', 'ratings.csv')

small_ratings_raw_data = sc.textFile(small_ratings_file)
small_ratings_raw_data_header = small_ratings_raw_data.take(1)[
0]
```

Model development using a smaller dataset

In order to determine the best ALS parameters which can be used for training our recommender, we used the small dataset. We needed first to split it into train, validation, and test datasets.

```
training_RDD, validation_RDD, test_RDD = small_ratings_data.ran
domSplit([6, 2, 2], seed=0L)
validation_for_predict_RDD = validation_RDD.map(lambda x: (x[0]
, x[1]))
test_for_predict_RDD = test_RDD.map(lambda x: (x[0], x[1]))
```

After Splitting our smaller dataset into train, test and validation set, we train our model to find out the best hyper tuning parameters which can be used for training our recommender.

```
from pyspark.mllib.recommendation import ALS
import math
seed = 250
iterations = 10
regularization_parameter = 0.1
ranks = [4, 8, 12]
errors = [0, 0, 0]
err = 0
tolerance = 0.02

min_error = float('inf')
best_rank = -1
best_iteration = -1
for rank in ranks:
    model = ALS.train(training_RDD, rank, seed=seed, iterations=iterations,
                      lambda_=regularization_parameter)
    predictions = model.predictAll(validation_for_predict_RDD).map(lambda r: ((r[0], r[1]), r[2]))
    rates_and_preds = validation_RDD.map(lambda r: ((int(r[0]), int(r[1])), float(r[2]))).join(predictions)
    error = math.sqrt(rates_and_preds.map(lambda r: (r[1][0] - r[1][1])**2).mean())
    errors[err] = error
    err += 1
    print('For rank %s the RMSE is %s' % (rank, error))
    if error < min_error:
        min_error = error
        best_rank = rank

print('The best model was trained with rank %s' % best_rank)
```

We then pass these best hyper parameters to our large dataset for training and recommendation purposes.

Final model development using Larger Dataset

In order to build the final recommender, we used the complete dataset and we processed it the same way as we did the small dataset. But here we used the best hyper parameters which we find out during training on our smaller dataset.

```
training_RDD, test_RDD = complete_ratings_data.randomSplit([7, 3], seed=42)

complete_model = ALS.train(training_RDD, best_rank, seed=seed,
                           iterations=iterations, lambda_=regularization_parameter)
```

Model Performance

We have used the Root Mean Square error to see how our model is performing on dataset. It is basically a difference between the predicted value produced by the model and the expected value. It allowed understanding how our model is performing on the test data.

```
For testing data the RMSE is 0.8302558863048242
```

Visualization

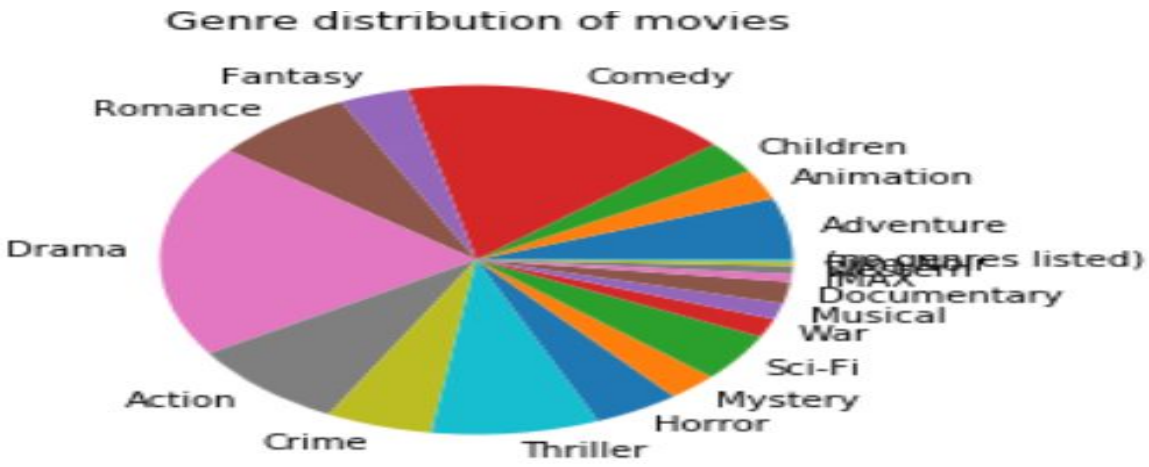
1: Dataset Table

	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy

2: Genre Distribution of Movies

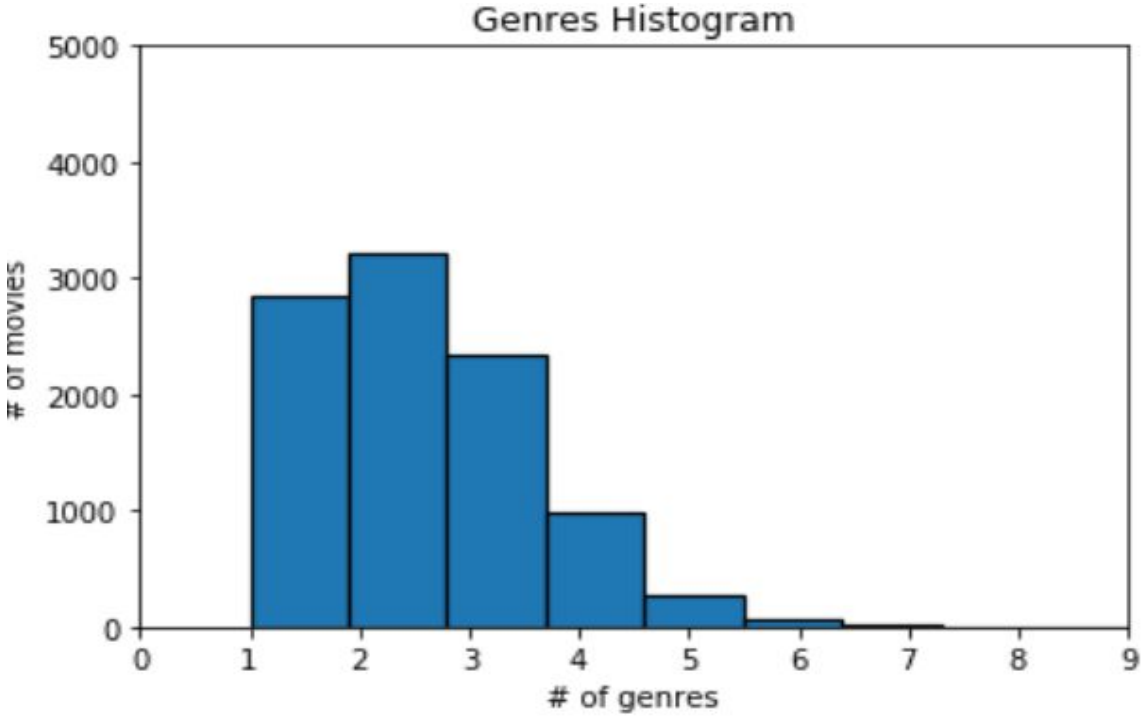
	movieId	title	genres	genres_arr	genre_count
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	[Adventure, Animation, Children, Comedy, Fantasy]	5
1	2	Jumanji (1995)	Adventure|Children|Fantasy	[Adventure, Children, Fantasy]	3
2	3	Grumpier Old Men (1995)	Comedy|Romance	[Comedy, Romance]	2
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	[Comedy, Drama, Romance]	3
4	5	Father of the Bride Part II (1995)	Comedy	[Comedy]	1

3:Number of Genres against Number of Movies

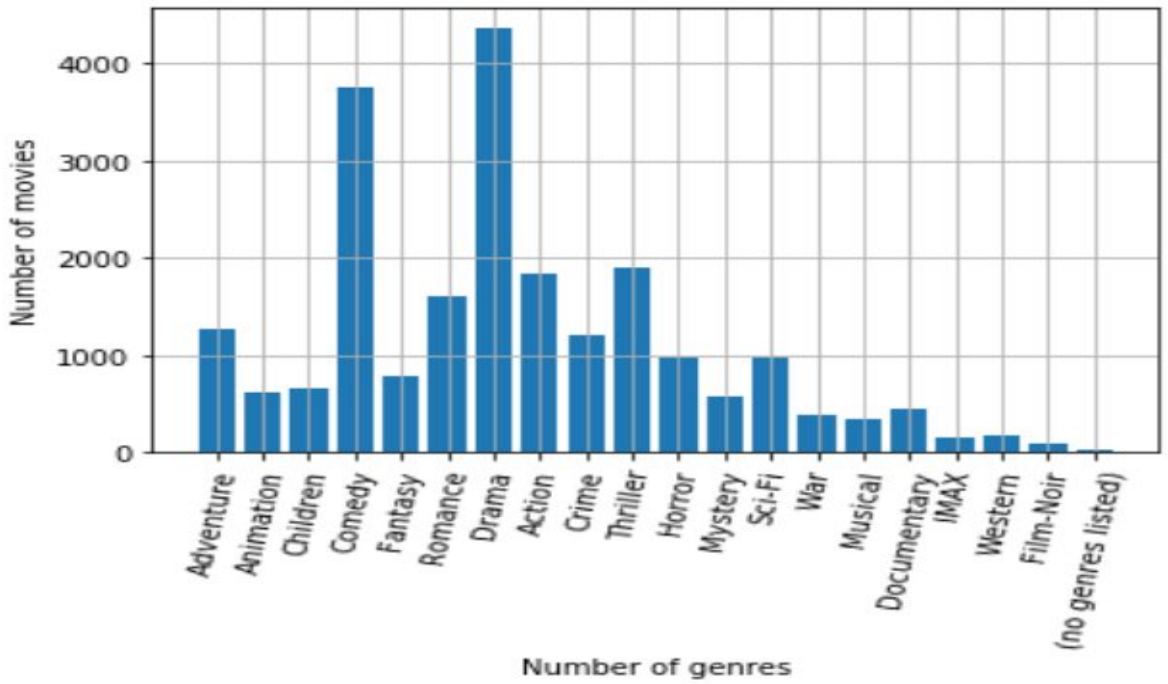



```
<Figure size 432x288 with 0 Axes>
```

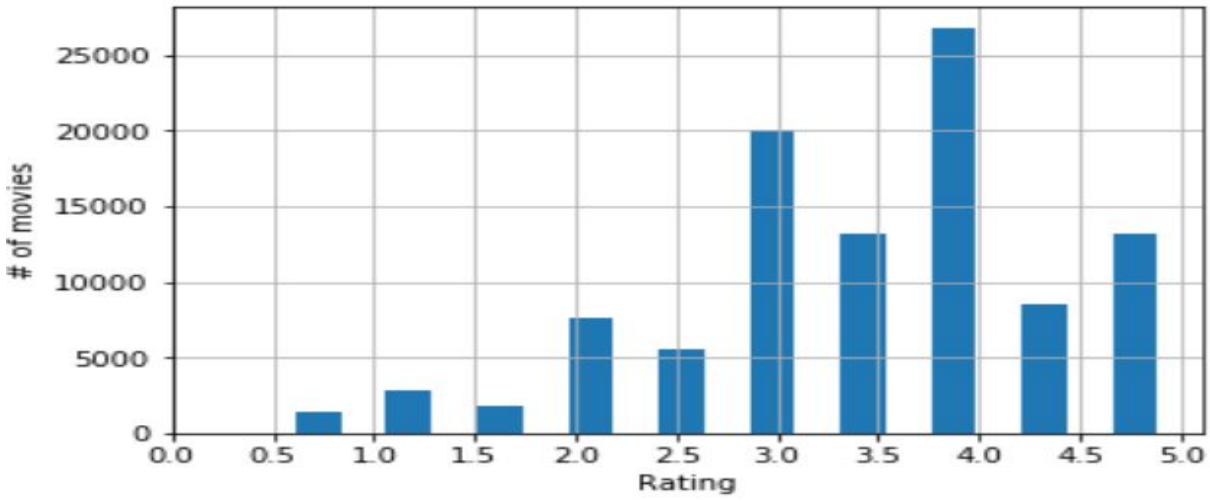
4: Total Number of Movies against Total Number of Genres



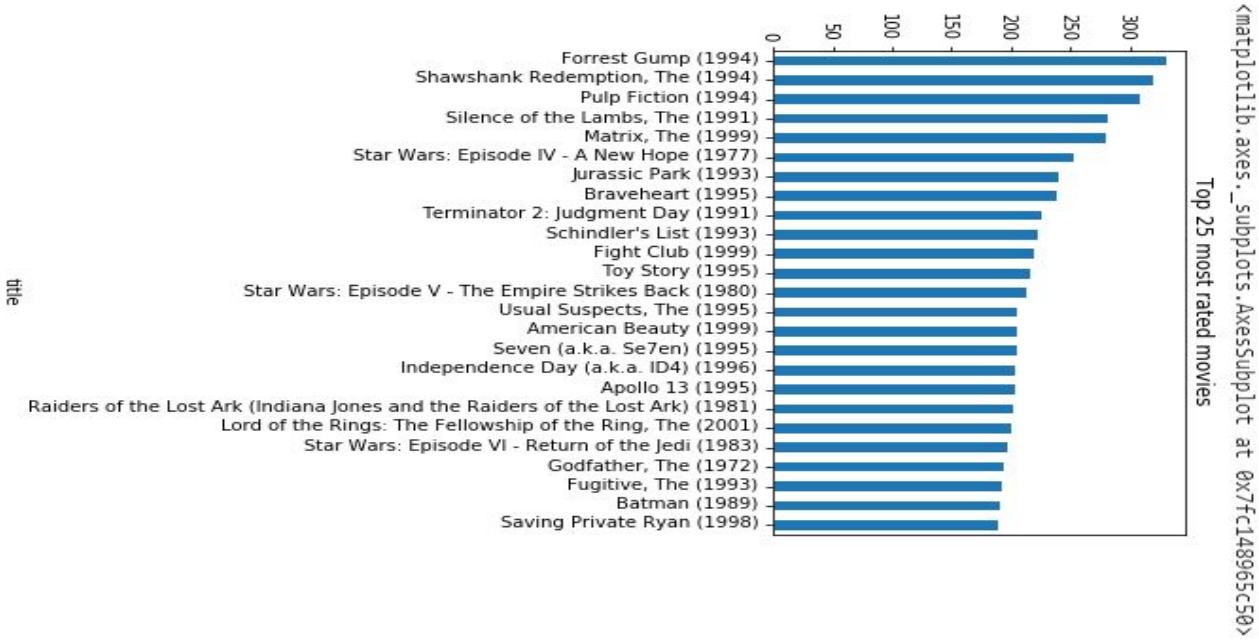
5: Total Number of Movies for Each Genre



6:Histogram of Movie Rating



7: Top 25 Rated Movies



Result

1: **Root Mean Square Error on Small Validation Dataset:** We choose the value with lowest RMSE value which will further be used for training our large dataset.

```
For rank 4 the RMSE is 0.9219502380669411
For rank 8 the RMSE is 0.9240152640633886
For rank 12 the RMSE is 0.9223173115607685
The best model was trained with rank 4
```

2: **Root Mean Square Error on Small Test Dataset:** We receive the RMSE value of 0.972342381898 when we test our trained model on small dataset.

```
For testing data the RMSE is 0.972342381898
```

3: **Root Mean Square on Large Dataset**: After using best hyper parameters for small dataset for training our large dataset mode.We receive the RMSE of 0.8302558863048242

```
For testing data the RMSE is 0.8302558863048242
```

Future Scope

There are plenty of way to expand on the work done in this project.

- Firstly, the content based method can be expanded to include more criteria to help categorize the movies. The most obvious ideas is to add features to suggest movies with common actors, directors or writers.
- We could also try to develop hybrid methods that try to combine the advantages of both content-based methods and collaborative filtering into one recommendation system.
- We can extend also extend our model by deploying on the web service. For this,We need to use Flask web application that defines as a Restful like API for our engine. We then Dockerize our Web App which we can deploy to the cloud using Kubernetes.

References

- https://github.com/jadianes
- https://spark.apache.org/docs/2.2.0/mllib-collaborative-filtering.html
- https://en.wikipedia.org/wiki/Recommender_system#Collaborative_filtering
- https://en.wikipedia.org/wiki/Recommender_system#Collaborative_filtering
- https://medium.com/@madasamy/introduction-to-recommendation-systems-and-how-to-design-recommendation-system-that-resembling-the-9ac167e30e95
- https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html
- https://spark.apache.org/docs/latest/api/python/index.html