Movie Recommendation System using PySpark

Venkata Subba Rao Inti, Siva Rama Krishna Bheemanaboyana, Sai Krishna Malakalapalli
School of Computing and Engineering
University of Missouri-Kansas City, Missouri, USA
{vikcc, sbp2m, vmfvn}@umsystem.edu

ABSTRACT

Recommendation systems are data-filtering programs that are mainly used to predict user and item ratings, primarily from big data, to recommend their interests. Users can classify users with similar interests using these types of systems. This makes recommendation systems as a central part of websites, Online Streaming Platforms, and other ecommerce applications as well.

There are so many approaches in the market to make new or improve the existing recommendation systems. In this report, we will discuss about various method of filtering techniques along with the method that we used in our project. This report clearly demonstrates collection of datasets, dataflow of this project, application of ALS methodology, results, and enhancements.

KEYWORDS

Users, Ratings, Movies, Alternating Least Squares (ALS), Root Mean Squared Error (RMSE), Prediction, tags, links, recommendations, filtering, training, testing, similarity, matrix factorization

1. Introduction

Everyone, regardless of their age, gender, race, color, or geographic area, enjoys movies. Through this incredible medium, we are all connected in some manner. What's most intriguing is how unique our movie preferences and combinations are. Some people prefer genre films, such as thrillers, romances, or science fiction, while others are more interested in the leading stars and filmmakers. With all of this in mind, it's quite impossible to generalize a film and declare that everyone will enjoy it. Despite this, similar films are enjoyed by a specific group of the population.

Recommender systems are a set of strategies and algorithms that help users find "relevant" items. They use a variety of strategies to forecast future behavior based on historical data. For example, Amazon can recommend new shopping items to buy, Netflix can recommend new movies to watch, and Google can recommend news that a user might be interested in.

In this report, we will discuss about the 27 million Data Records from Movie Lens, Various methods of filtering techniques, Project Workflow, Project Illustration, Plotting results and few recommendations for users. In the end, we will also discuss about the results, scope, enhancements, and contributions for this project.

2. Related Work:

In movie recommender systems, users are asked to rate movies that they have already seen, and then these ratings are used to recommend other movies to them that they have not seen, using collaborative filtering based on similar ratings. Collaborative filtering is gaining attraction to the point where it now influences many recommender systems. Memory-based collaborative filtering and model-based collaborative filtering are the two most common types of collaborative filtering. Memory-based collaborative filtering searches for an active user's nearest neighbors in the user space and dynamically recommends movies. The computation complexity and data sparsity are two drawbacks of this strategy.

Many writers attempted to alleviate the computational complexity and memory bottleneck difficulties by computing relationships between objects for a neighborhood region surrounding a target object. Various researchers applied clustering-based methods on recommender systems that delivered expert recommendations. The purpose of clustering is to partition objects into groups known as clusters in such a way that two objects within the same cluster have a minimum distance between them to identify similar objects then clustering process is performed offline to build the model. It is also seen that the RMSE value of the proposed system is achieving the same value as the existing technique used in this project.

They demonstrated in their empirical experiments that the item-based strategy could save computation time while also providing rationally correct prediction and accuracy. The model-based collaborative technique generates a pre-built model for collecting rating patterns based on a database of users and ratings, which can address data sparsity and scalability difficulties. Model-based strategies were used by several researchers on recommender systems that provided expert recommendations.

3. Dataset

Group Lens Research has collected and made available rating data sets from the Movie Lens web site (https://grouplens.org/datasets/movielens/). This dataset describes 5-star rating and free-text tagging activity from Movie Lens, a movie recommendation service. It contains 27753444 (27M) ratings and 1108997 tag applications across 58098 movies. These data were created by 283228 users between January 09, 1995, and September 26, 2020. The dataset that we used was generated on September 26, 2020. Users were selected at random for inclusion. All

selected users had rated at least 1 movie. No demographic information is included. Each user is represented by an id, and no other information is provided.

All ratings are contained in the file ***ratings.csv***. Each line of this file after the header row represents one rating of one movie by one user, and has the following format:

userId - Unique Id for each user,
movieId - Unique Id for each movie
rating – rating given by userId for the movieId
timestamp - Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970.

Movie information is contained in the file ***movies.csv***. Each line of this file after the header row represents one movie, and has the following format:

movieId – Unique Id for each movie
title – Title of the movie
genres – Genre of the movie

Identifiers that can be used to link to other sources of movie data are contained in the file ***links.csv***. Each line of this file after the header row represents one movie, and has the following format:

movieId – Unique Id for each movie
imdbId – IMDB rating for that movie
tmdbId – TMDB rating for that movie

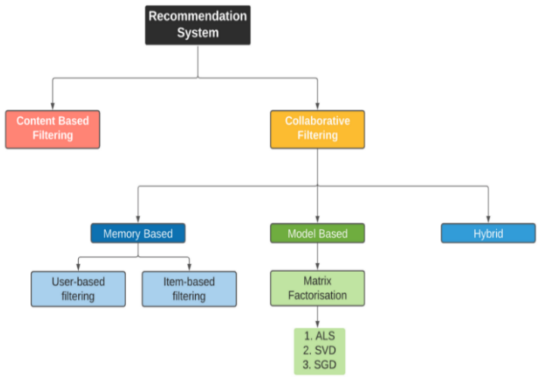
The dataset files are written as comma-separated values files with a single header row. User ids are consistent between *ratings.csv and tags. csv* i.e., the same id refers to the same user across the two files. Only movies with at least one rating or tag are included in the dataset. These movie ids are consistent with those used on the Movie Lens web site. Movie ids are consistent between *ratings.csv, tags.csv, movies.csv, and links. csv* i.e., the same id refers to the same movie across these four data files.

4. Methodology

The goal of this project is to create a movie recommendation system that incorporates explicit user feedback on movie ratings. There are mainly two main approaches in recommendation systems. They are Collaborative Filtering and Content Based Filtering.

4.1. Collaborative Filtering: By gathering preferences or taste information from many users, we can create predictions about a user's interests. The basic concept is that if a user A and a user B have the same view on one issue, A is more likely to have B's opinion on a different topic x than a random user's opinion on x.

4.2. Content Based Filtering: Content Based Filtering methods are based on a description of the item and a profile of the user's preferences. In a content-based recommender system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are like those that a user liked in the past or is examining in the present. Various candidate items are compared with items previously rated by the user and the best-matching items are recommended.



Collaborative Filtering has some major advantages over CBF in that it can perform in domains where there is not much content associated with items and where content is difficult for a computer system to analyze (such as opinions and ideal). Also, CF technique has the ability to provide serendipitous recommendations, which means that it can recommend items that are relevant to the user even without the content being in the user's profile.

In this project we will be focusing on a type of Collaborative Filtering called Alternating Least Squares (ALS) using Matrix Factorization.

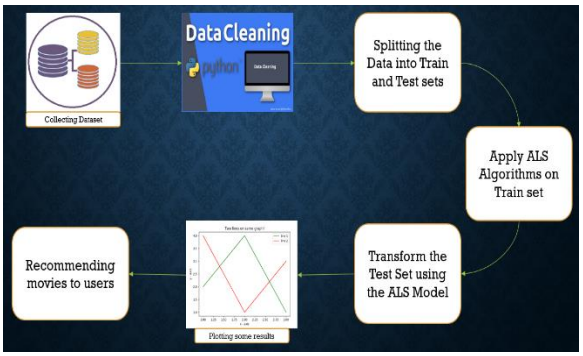
4.3. ALS Approach: ALS is one of the low rank matrix approximation algorithms for collaborative filtering. ALS decomposes user-item matrix into two low rank matrixes: user matrix and item matrix. In collaborative filtering, users and products are described by a small set of latent factors that can be used to predict missing entries. And ALS algorithm learns these latent factors by matrix factorization.

Spark MLlib library for Machine Learning provides a Collaborative Filtering implementation by using Alternating Least Squares. The implementation in MLlib has these parameters:

numBlocks-number of blocks used to parallelize operations.
Rank-number of latent factors in the model.
Iterations-number of iterations to run.
Lambda-regularization parameter in ALS.
ImplicitPrefs-whether to use the explicit feedback ALS variant or one adapted for implicit feedback data.
Alpha- parameter applicable to the implicit feedback variant of ALS that governs the baseline confidence in preference observations.

5. Project WorkFlow

In this section, we will discuss the step-by-step process that we followed while doing this project.



Firstly, we collected the huge dataset of 27M+ records from Movie Lens website as mentioned in the previous section. Once we got the dataset, we will do necessary preprocessing steps. As this data is comma separated and textual data, we don't have much to clean in the data. Once data was cleaned, we will split the dataset into training dataset and testing dataset. And then we will apply the ALS approach on the training dataset and then we will apply that model on the testing dataset. In the end, we will plot some data results and we will also give recommendations for some specific users and for some group of users.

6. Project Illustration

We implemented this project in Google Colab Platform where we installed Apache Spark and Java using the commands and configured the home paths to each.

```
[2] import os
    os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
    os.environ["SPARK_HOME"] = "/content/spark-3.0.3-bin-hadoop3.2"
```

6.1. Importing Libraries and Creating Spark Session: Once environmental variables were set, you can start running spark scripts. We imported all the required libraries for establishing spark session, sql context, applying ALS algorithm and to do some computations. And we created the spark configuration with port number 4050 and created spark context using the created configuration.

In order to run PySpark in your machine, you need to import findspark package and you need to run init() in it. ALS is present in Spark Machine Learning Library and imported spark sql functions like desc, col, udf, when. We will create a SparkSession with the master node with function getOrCreate().

```
import findspark
findspark.init()
from pyspark import SparkContext, SparkConf
from pyspark.sql import SQLContext, SparkSession
```

```
[6] from pyspark.ml.evaluation import RegressionEvaluator
    from pyspark.ml.recommendation import ALS
    from pyspark.sql.functions import udf,col,when,desc

    import numpy as np
    from IPython.display import Image
    from IPython.display import display
```

```
[7] conf = SparkConf().set('spark.ui.port', '4050')
    sc = SparkContext(conf=conf)
    spark = SparkSession.builder.master('local[*]').getOrCreate()
```

6.2. Data Loading into DataFrames: Once Spark session was established, we can start runing PySpark functions. Now in the first step, we want to load the dataset downloaded into the individual dataframes as follows:

```
[22] ratings=spark.read.load('/content/ml-latest/ratings.csv', format='csv', inferSchema=True, header=True)
     ratings.printSchema()

root
 |-- userId: integer (nullable = true)
 |-- movieId: integer (nullable = true)
 |-- rating: double (nullable = true)
 |-- timestamp: integer (nullable = true)
```

```
movies=spark.read.load('/content/ml-latest/movies.csv', format='csv', inferSchema=True, header=True)
movies.printSchema()

root
 |-- movieId: integer (nullable = true)
 |-- title: string (nullable = true)
 |-- genres: string (nullable = true)
```

```
[10] tags=spark.read.load('/content/ml-latest/tags.csv', format='csv', inferSchema=True, header=True)
     tags.printSchema()
```

```
[11] links=spark.read.load('/content/ml-latest/links.csv', format='csv', inferSchema=True, header=True)
     links.printSchema()

root
 |-- movieId: integer (nullable = true)
 |-- imdbId: integer (nullable = true)
 |-- tmdbId: integer (nullable = true)
```

6.3. Splitting & Applying ALS method: Now, we need to use these dataframes for furthur. Firstly, we need to split the dataset into training set and testing set to apply ALS algorithm on training data and later on test data to transform. A series of actions are done using the below code:

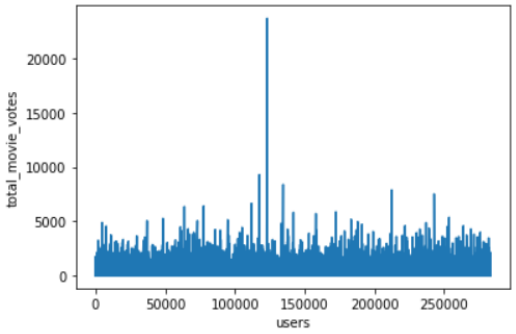
```
[12] training_set,testing_set=ratings.randomSplit([0.8,0.2])
     training_set.count()

22201324
```

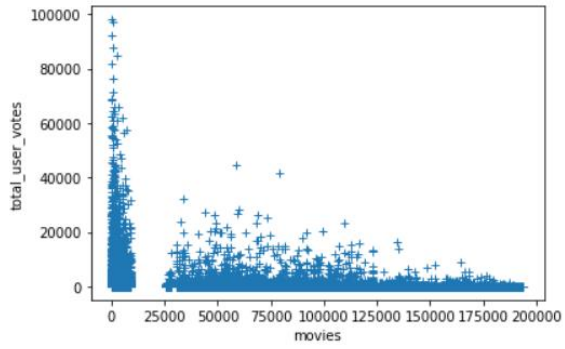
```
[13] als=ALS(maxIter=10,regParam=0.1,rank=4,userCol="userId",itemCol="movieId",ratingCol="rating")
     model=als.fit(training_set)
     predictions=model.transform(testing_set)
     new_predictions=predictions.filter(col('prediction')!=np.nan)
     evaluator=RegressionEvaluator(metricName="rmse",labelCol="rating",predictionCol="prediction")
     rmse=evaluator.evaluate(new_predictions)
     print("root mean square error"+ str(rmse))

root mean square error0.8300382368827274
```

6.4. Plotting comparision between Users and Movies: From the dataset that we have, we tried to find out that how many movies that a user is rating from the entire ratings.csv dataset.



Based on these Plot, we have observed that each user on an average, he rated for atleast 2000 movies in the given timespan. Similarly, if you want to find how many votes that each movie got from the ratings.csv, and movies.csv



Based on the above plot, we can say that many movies didn't get any ratings from users. But very few movies are getting the more votes.

6.5. Recommendations to users: Based on the algorithm that we applied, we can find movie recommendations for individual users, for subset of users.

```
[15] ##Recommending movies for each particular user
particular_user=predictions.filter(col("userId")==599).join(movies,"movieID").join(links,"movieId").select("userId","title","genres",
                                                                                                            "tmdbId","prediction")
particular_user=particular_user.sort(desc("prediction"))
particular_user.show(10)
```

```
+------+--------------------+--------------------+------+----------+
|userId|               title|              genres|tmdbId|prediction|
+------+--------------------+--------------------+------+----------+
|   599|   Fight Club (1999)|Action|Crime|Dram...|   550|   3.94882|
|   599|Lord of the Rings...|   Adventure|Fantasy|   120| 3.9418998|
|   599|Star Wars: Episod...|Action|Adventure|...|    11| 3.8700497|
|   599|American Beauty (...|       Drama|Romance|    14| 3.8612788|
|   599|Dark Knight, The ...|Action|Crime|Dram...|   155| 3.763387|
|   599|Little Miss Sunsh...|Adventure|Comedy|...|   773| 3.7005336|
|   599|Green Mile, The (...|         Crime|Drama|   497| 3.5630517|
|   599|Ferris Bueller's ...|              Comedy|  9377|  3.537811|
|   599|        Juno (2007)|Comedy|Drama|Romance|  7326| 3.5125322|
|   599|Casino Royale (2006)|Action|Adventure|...| 36557|  3.393381|
+------+--------------------+--------------------+------+----------+
only showing top 10 rows
```

Here, we tried to find the recommendations for userId-599. Based on our approach, the highest rated prediction was 3.94882 out of 5 for the movie Fight Club(1999). This result might differ if we take different parameters while calling ALS() method.

6.6. Recommendation to all users: Similarly, with builtin functionality *recommendForAllUsers()*, we can also implement recommendations for all users using the following query and it will show output as follows:

```
[16] ##Generating top 5 recommendations for each user

     user_recommendations=model.recommendForAllUsers(5)

     #Generating top 5 user recommendations for each movie

     movie_recommendations=model.recommendForAllItems(5)
```

```
[17] user_recommendations.select("userId","recommendations.movieId").show(10,False)
```

```
+------+-----------------------------------------+
|userId|movieId                                  |
+------+-----------------------------------------+
|148   |[151989, 185211, 107434, 153184, 192089]|
|463   |[185211, 122222, 153184, 151989, 131586]|
|471   |[151989, 177209, 107434, 190707, 183947]|
|496   |[107434, 193369, 151989, 87798, 192089] |
|833   |[151989, 107434, 171851, 125279, 140351]|
|1088  |[107434, 178953, 183475, 87798, 154917] |
|1238  |[107434, 87798, 178953, 183475, 151989] |
|1342  |[149700, 101058, 169606, 154947, 140353]|
|1580  |[151989, 177209, 185645, 136892, 193759]|
|1591  |[151989, 177209, 120821, 107434, 159761]|
+------+-----------------------------------------+
only showing top 10 rows
```

For all these userIds, our recommendation system was suggesting those movieId's to watch based on their previous interests and other users likes of interest. Similarly, we can also use another functionality to provide suggestions for subset of people. *recommendationForUserSubset()* is used to provide these suggestions for a subset.

```
#User recommendations for a group of user subset
users=ratings.select("userId").distinct().limit(5)
users.show()
```

```
[26] user_subset_recommendations=model.recommendForUserSubset(users,10)
```

```
[ ] user_subset_recommendations.show()
```

```
[27] user_subset_recommendations.select("userId","recommendations.movieId").show(10,False)
```

```
+------+-----------------------------------------------------------------------------+
|userId|movieId                                                                      |
+------+-----------------------------------------------------------------------------+
|471   |[151989, 177209, 107434, 190707, 183947, 157789, 157791, 139074, 192089, 159761]|
|463   |[185211, 122222, 153184, 151989, 131586, 152711, 192089, 139074, 180293, 183947]|
|833   |[151989, 107434, 171851, 125279, 140351, 188113, 84838, 175275, 181811, 141016] |
|496   |[107434, 193369, 151989, 87798, 192089, 178953, 183475, 131586, 122222, 184299] |
|148   |[151989, 185211, 107434, 153184, 192089, 183947, 139074, 190707, 87798, 162436] |
+------+-----------------------------------------------------------------------------+
```

Finally, this is how we recommended movies for individual users, all users at a time, subset of users. We have mentioned title and genre in one output and only movieIds in other outputs.

7. Results

- We built a movie recommendation system based on the user-user similarity, movie-movie similarity, global averages, and matrix factorization.
- On the Test Data, our model resulted 0.83 RMSE Value, which is a good-to-go model. A model with RMSE < 2 is considered good.
- What happens if a user never watched a movie or is a new user? This problem is called as "Cold Start in Recommendation System".

8. Enhancements

- This model can be further enhanced by including features like *Location, genre and interests* while recommending a movie to users.
- *"Cold Start Problem"* can be eliminated if we apply *"Cross Validation"* on the ALS Model. This is one of the major enhancement that can be done.
- Using *"Hybrid Filtering"* will improve the performance of proposed system.

9. Future Scope

- Recommender systems are used in a wide range of applications, including movies, music, news, books, research articles, search queries, social tagging, and general items.
- In future, we can apply the cross validation to the existing model that we developed to provide the better results but applying cross validation will takes a longer time and it will keep on checking for all available values and provide the best results.
- Any Online Streaming Platforms, Ecommerce Applications can use this project to improve their customer service.
- It can be latter extended to advertising messages, suggesting music tracks, best courses in e-learning, e.t.c.

10. Team Contributions

Based on the suggestions that we got from the Professor Dr. Uddin, Siva Rama Krishna worked on collecting the huge data related to movie ratings from various sources. As data was almost clean, he also verified it and replaced NaN values with 0. Venkata Subba Rao, who is a lead developer for this project did the major part of this project. He took the cleaned data and splitted data into train and test datasets. He also did rigorous research on these filtering algorithms and finally came up with ALS method. He applied this method on train data and latter transformed on test data. Venkata Subba Rao also did the movie recommendations for individual users as well as for the group of users. He also took the responsibility of preparing the Project Report and Paper Presentation. Venkata Sai Krishna, who is responsible for plotting graphs for users and movies. His role also includes the identifying the RMSE value. And all three of us worked on preparing slides for project presentations.

11. Acknowledgements

We sincerely want to thank Professor Dr. Uddin and Our Teaching Assistant for constantly replying to our emails regarding concerns on the project right from the Project selection to preparing the project report. Help from Professor made the project requirements quite understandable. And this assistance guided us to collect the relevant and huge dataset for these type of projects from various sources.

12. References

[1] Merve Acilar, Ahmet Arslan. A collaborative filtering method based on artificial immune network. Expert Systems with Application, 2009.

[2] Bhavik Pathak, Robert Garfinkel, Ram D. Gopal, Rajkumar Venkatesan & Fang Yin. Empirical Analysis of the Impact of Recommender Systems on Sales, Journal of Management Information Systems, December 2014.

[3] Chenguang Pan, Wenxin Li. Research Paper Recommendation with Topic Analysis. In Computer Design and Applications IEEE, 2010.

[4] https://files.grouplens.org/datasets/movielens/ml-latest-README.html for collecting Movie Lens Dataset.

[5] Xiangnan He Hanwang Zhang Min-Yen Kan Tat-Seng Chua Fast Matrix Factorization for Online Recommendation with Implicit Feedback, 2017

[6] Rishabh Ahuja, Arun Solanki, Anand Nayyar, Movie Recommender System Using K-Means Clustering and K-Nearest Neighbor, 2019