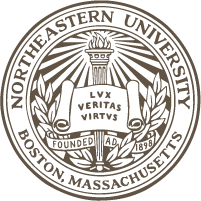
**INFO 6101 Data Science Engineering with Python**



**MOVIE RECOMMENDATION SYSTEM**

**Progress Draft**

**Harshad Pardeshi**

**Krutika Deshpande**

**Prithvi Kamath**

**Under the guidance of**

**Prof. Konstantopoulos, Dino**

# **Abstract**

Recommender systems have become ubiquitous in our lives. They have become very popular in many aspects in real social networks, such as e-commerce services Amazon.com, movie rating website IMDB, and DVD rental service company Netflix. This project focuses on signed link prediction of recommending movies. In this project, we attempt to understand the different kinds of recommendation systems and compare their performance on the MovieLens dataset. We attempt to build a scalable model to perform this analysis. We start by preparing and comparing the various models on a smaller dataset of 100,000 ratings. Then, we try to scale the algorithm so that it is able to handle 20 million ratings by using Apache Spark. We find that for the smaller dataset, using user-based collaborative filtering results in the lowest Mean Squared Error on our dataset.

# **Introduction**

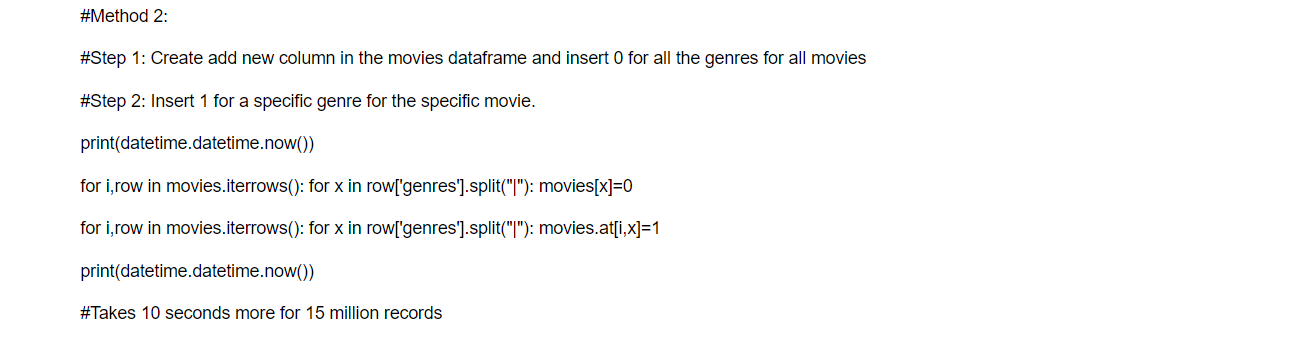
In today’s world where internet has become an important part of human life, users often face the problem of too much choice. Right from looking for a motel to looking for good investment options, there is too much information available. To help the users cope with this information explosion, companies have deployed recommendation systems to guide their users. The research in the area of recommendation systems has been going on for several decades now, but the interest still remains high because of the abundance of practical applications and the problem rich domain. A number of such online recommendation systems implemented and used are the recommendation system for books at Amazon.com, for movies at MovieLens.org, CDs at CDNow.com (from Amazon.com), etc.

Due to the advances in recommender systems, users constantly expect good recommendations. They have a low threshold for services that are not able to make appropriate suggestions. If a music streaming app is not able to predict and play music that the user likes, then the user will simply stop using it. This has led to a high emphasis by tech companies on improving their recommendation systems. However, the problem is more complex than it seems. Every user has different preferences and likes. In addition, even the taste of a single user can vary depending on a large number of factors, such as mood, season, or type of activity the user is doing. For example, the type of music one would like to hear while exercising differs greatly from the type of music he’d listen to when cooking dinner. Another issue that recommendation systems have to solve is the exploration vs exploitation problem. They must explore new domains to discover more about the user, while still making the most of what is already known about of the user. Two main approaches are widely used for recommender systems. 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.

# **Code with Documentation**

## **Exploratory Data Analysis**

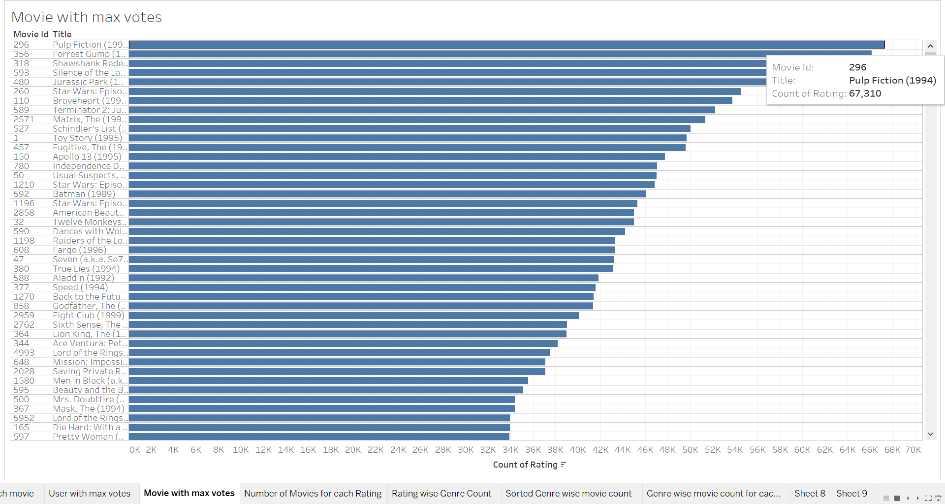
Created dataframe using the movies.csv file. Dataframe consisted of three columns viz. 'movieid', 'title', 'genres'. The 'genres' column had a combination of the all the genres a movie is about. These genres were extracted from a single column to multiple column with values as 1 and 0 if the particular movie is of that genre. For example, the movie 'Toy Story' has the genres 'Adventure', 'Animation' , 'Childern' , 'Comedy' and 'Fantasy'. So, for those columns the cell value is set to 1 and rest are 0. The cleaned dataset is then writing into a csv file.





## **Tableau Visualization**

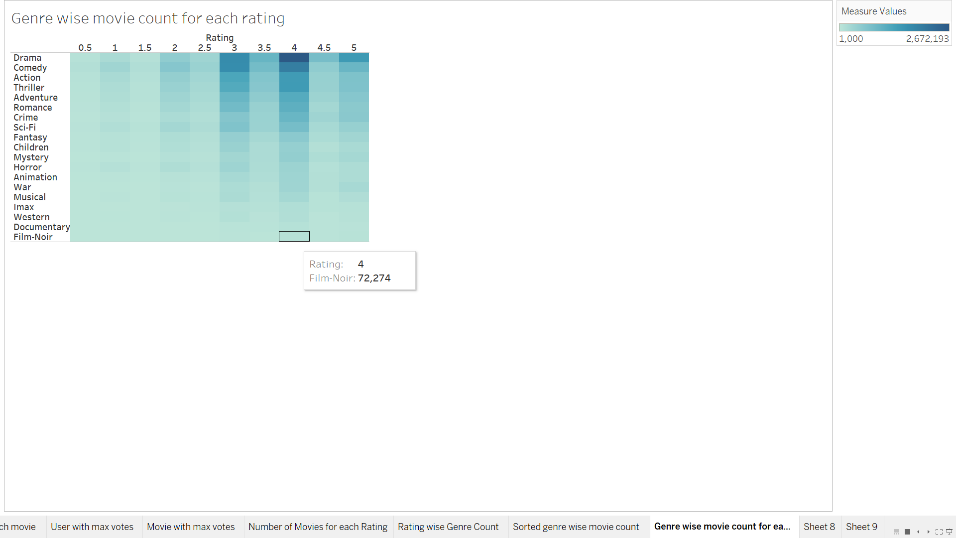
* Movie with maximum votes



* Word cloud of Genres

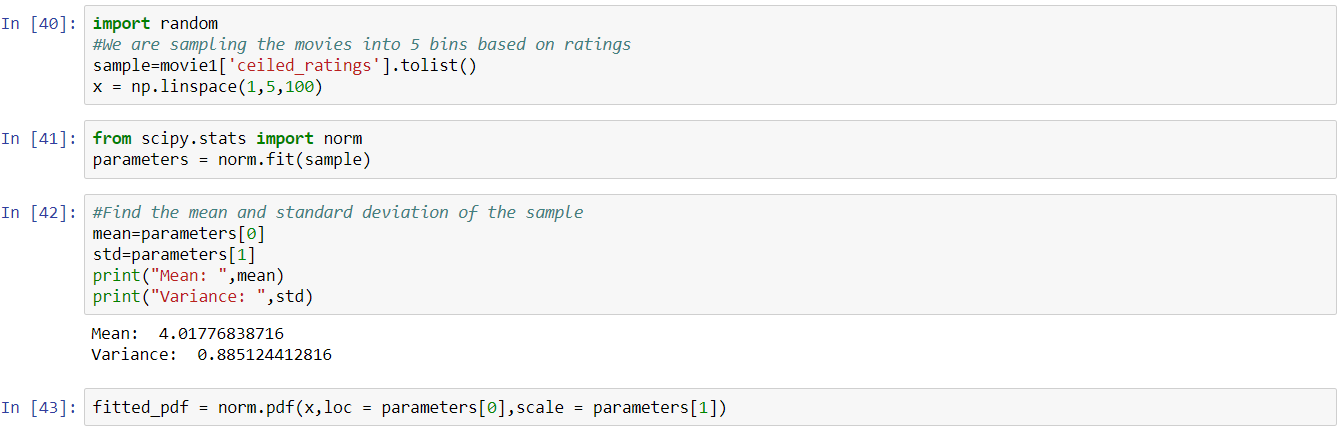


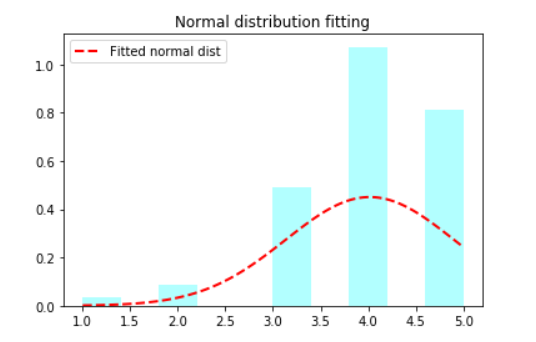
* Genre wise movie count for each rating



## **Maximum Likelihood Estimation**

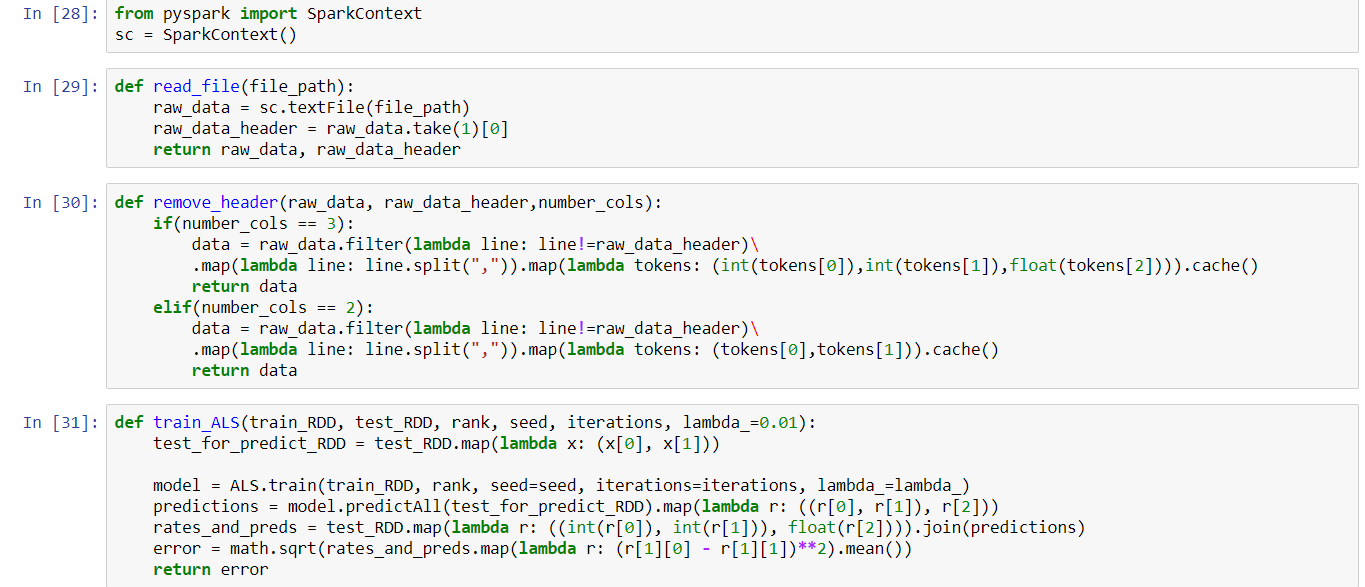
Filtered the ratings for movie with movieId equal to 1 i.e. ratings for 'Toy Story'. Sampled the movies into 5 bins based on ratings. Fitted the sample into normal distribution. The sample appears to be left skewed indicating a unimodal distribution. Plotted the maximum likelihood estimation for the sampled data and estimated the parameters viz. mean and variance for the sample.





## **Collaborative Filtering using Spark RDD**

Initially we trained and tested our model on a small dataset with around 10,000 ratings. Using Spark RDD, read those ratings and movies file. First of all, created spark object using SparkContext(), which tells Spark how to use a cluster. Performed transformations on the dataset using lambda functions where we extract the cell values from the dataset and map those into tuples. The header for the csv files is removed.



Focusing on collaborative filtering models which can be generally split into two classes: user- and item-based collaborative filtering. In either scenario, one builds a similarity matrix. For user-based collaborative filtering, the user-similarity matrix will consist of some distance metric that measures the similarity between any two pairs of users. Likewise, the item-similarity matrix will measure the similarity between any two pairs of items.

We can turn our matrix factorization approximation of a *k*-attribute user into math by letting a user *u* take the form of a *k*-dimensional vector *x\_u*. Similarly, an item *i* can be *k*-dimensional vector *y\_i*. User *u*’s predicted rating for item *i* is just the dot product of their two vectors.

https://cdn-images-1.medium.com/max/1000/1*C02BnJr9kV0NxTu3SzoXZA.png

where *r\_ui*hat represents our prediction for the true rating *r\_ui*, and *y\_i*(*x⊺\_u*) is assumed to be a column (row) vector. These user and item vectors are often called latent vectors or low-dimensional embeddings in the literature. The *k* attributes are often called the latent factors.

For ALS minimiztion, we hold one set of latent vectors constant. For this example, we’ll pick the item vectors. We then take the derivative of the loss function with respect to the other set of vectors (the user vectors). We set the derivative equal to zero (we’re searching for a minimum) and solve for the non-constant vectors (the user vectors). Now comes the alternating part: With these new, solved-for user vectors in hand, we hold them constant, instead, and take the derivative of the loss function with respect to the previously constant vectors (the item vectors). We alternate back and forth and carry out this two-step dance until convergence.

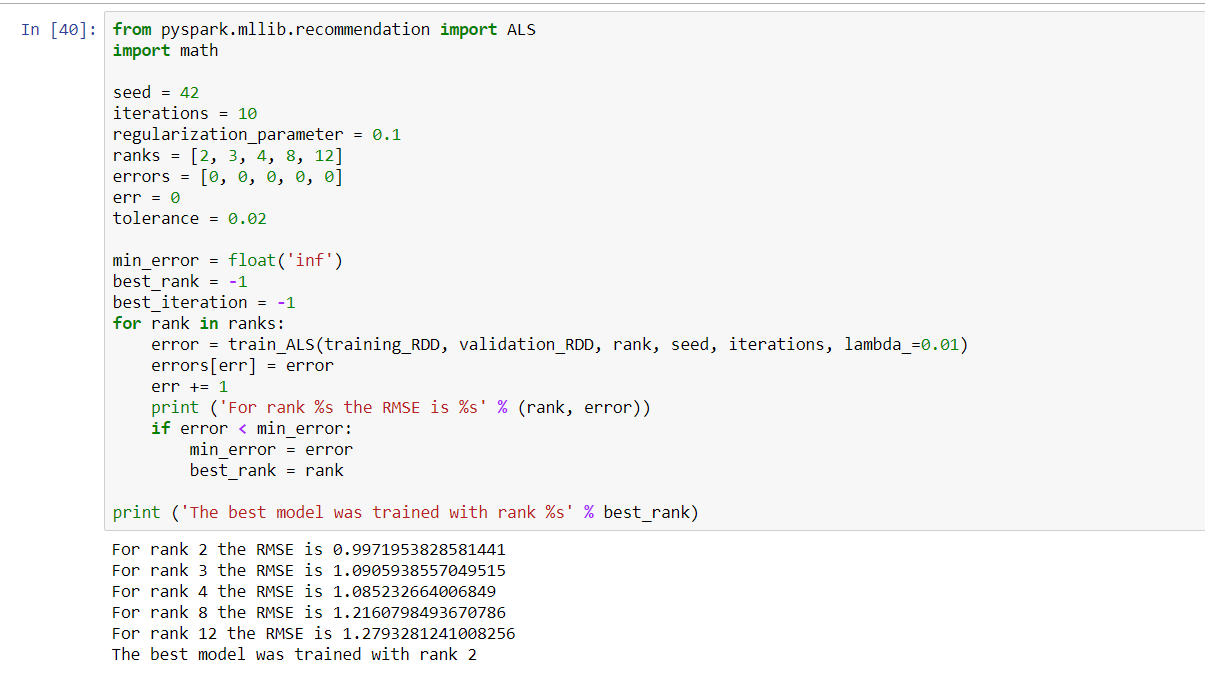
We multiply each feature of the user by the corresponding feature of the movie and add everything together, this will be a good approximation for the rating the user would give that movie.

Opting for the best model with the lowest RMSE value.

For rank 2 we get the low RMSE value of 0.99719.

Calculating the error on the testing dataset.

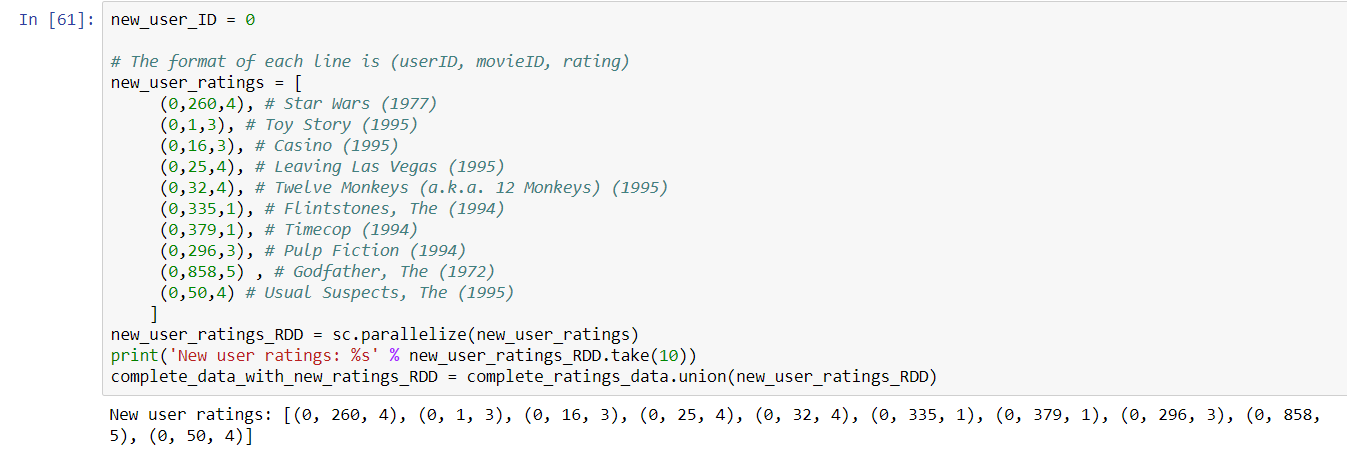
All the operations are initially being tested on the small dataset.



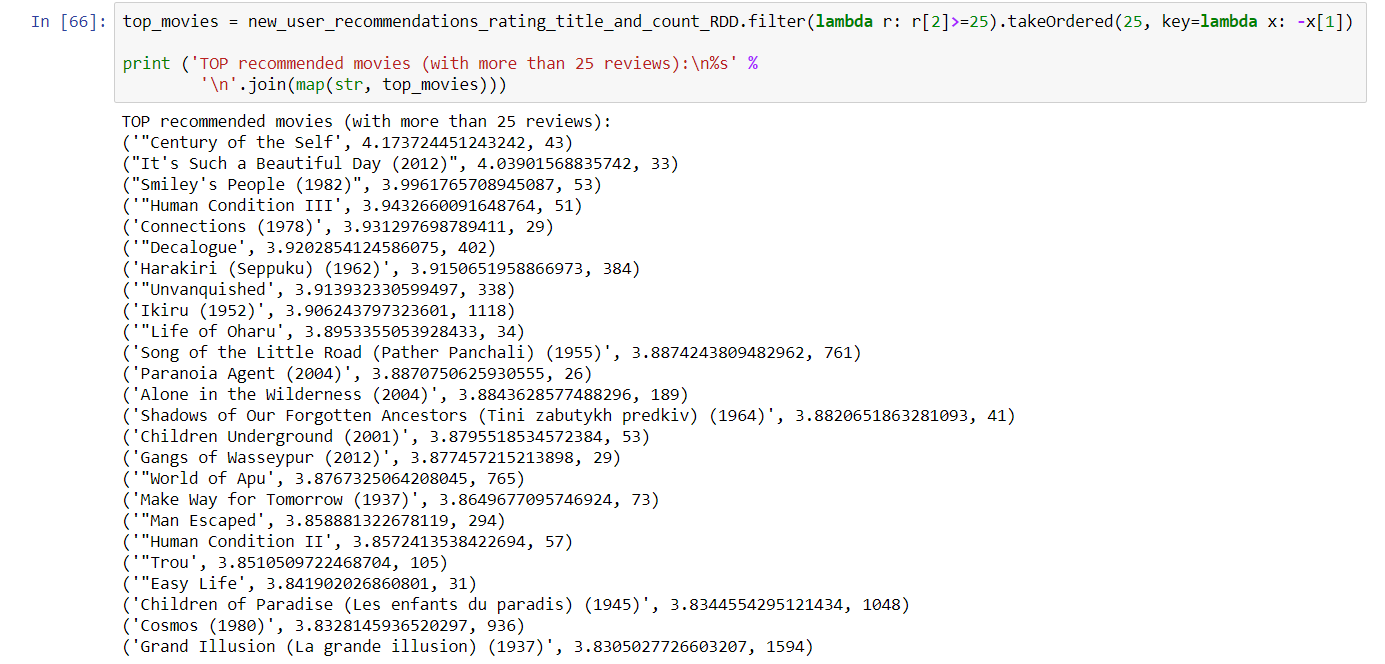
# **Results**

Once the model is trained, the model is tested by adding up a new user and the ratings he gave to the movies he has watched.

Based on this, the system now recommends movies to the user.



Recommended movies to the new user



# **Conclusion**

We were successfully able to implement the movie recommendation system with minimum error. The new user who rated movies like GodFather, Star Wars, Leaving Las Vegas was recommended movies like Century of the Self, It’s Such a Beautiful Day and Smiley People. The objective of this project to work and implement Spark framework, recommendation system was achieved successfully.

# **References**

# <https://blog.insightdatascience.com/explicit-matrix-factorization-als-sgd-and-all-that-jazz-b00e4d9b21ea>

1. <https://spark.apache.org/docs/latest/rdd-programming-guide.html>
2. <https://grouplens.org/datasets/movielens/20m/>
3. <https://towardsdatascience.com/probability-concepts-explained-maximum-likelihood-estimation-c7b4342fdbb1>
4. <https://www.cs.cmu.edu/~mgormley/courses/10601-s17/slides/lecture25-mf.pdf>