Movie Recommendation System

Akhila Reddy Dodda [1] **Akshay Krishna Sheshadri** [1] **Muni Lohith Krishna Mohan Konidala** [1]

Abstract

Recommender systems are important tools that analyze data on user preferences and behavior to make personalized recommendations. They can help improve the user experience, increase engagement, and drive revenue for businesses and they are widely used in e-commerce, social media, and other applications. We plan to use PySpark to build recommender systems to suggest products, or services, or content to users based on their preferences and behavior. In particular, we plan to build a Movie recommendation system and evaluate it using various performance metrics such as accuracy, Mean average precision @ K (MAP@K), and Mean Average Recall @ K (MAR@K).

1 Introduction

Recommender systems are software tools and algorithms that analyze data on user preferences and behavior to make personalized recommendations for items such as products, services, or content. The goal of these systems is to predict what items a user might be interested in and present them with recommendations that are relevant and useful. They are a powerful tool for businesses looking to improve their customer experience and gain a competitive edge.

Recommender systems have become increasingly important in recent years due to the explosion of digital content and the growing amount of data available on user behavior. They are used in a variety of applications, from e-commerce and online advertising to music and video streaming services, and they can help improve customer engagement, increase sales, and enhance the user experience.

For our use-case, the primary objective of our recommendation system is to predict and filter only those movies that a user is likely to prefer, based on relevant user and movie data.

2 Data

We plan to use 'The Movies Dataset'. This dataset has both a *small* and *full* data split. The *Small* split consists of 100,000 ratings and 3,600 tag applications applied to 9,000 movies by 600 users. The *Full* split consists of 27,000,000 ratings and 1,100,000 tag applications applied to 58,000 movies by 280,000 users. Each of these have the files containing information as shown in table 1: We will load these different files into pyspark to perform efficient filters, joins, and other transformations to prepare the data that will be subsequently processed by our recommender system.

File	Description
Metadata	Features include posters, backdrops, budget, revenue, release dates, languages, production countries and companies
Keywords	Contains movie plot keywords.
Credits	Cast and Crew Information
Links	The TMDB and IMDB IDs of all movies
Ratings	The ratings of all users on the movies

Table 1. Data present in The Movies Dataset

3 Methodology

3.1 Data preparation, exploration, and pre-processing

To begin, we make sure that the data that includes information about movies, users, and their interactions (user ratings of movies) is in a format that can be loaded into PySpark. After loading the data, we will apply appropriate joins and transformations based on the different data sources to prepare the dataset in the required format for training our recommender system.

After this, we will perform exploratory data analysis (EDA) to visualize and comprehend the data's structure and characteristics.

Finally, we will preprocess and query the data using Spark-SQL, which involves tasks such as data cleaning, handling

*Equal contribution [1]Manning College of Information & Computer Sciences, University of Massachusetts Amherst. Correspondence to: Akhila Reddy Dodda <akhilareddyd@umass.edu>, Akshay Krishna Sheshadri <asheshadri@umass.edu>, Muni Lohith Krishna Mohan Konidala <mkonidala@umass.edu>.

missing values, feature engineering, and creating train-validation-test splits. The processed data will then be used as input by the Machine Learning Model.

The project workflow is presented in Figure 2, and an interactive dashboard will be provided for users to access the system by entering their ID and receiving personalized recommendations.

3.2 Training (filtering strategy)

Recommender systems come in various types, including content-based filtering, collaborative filtering, and hybrid approaches that blend the two techniques. Content-based filtering evaluates item characteristics like genre, category, or keywords to offer recommendations, while collaborative filtering analyzes user preferences and behavior, such as ratings or past purchases, to identify patterns and suggest similar items. Hybrid approaches combine both techniques to suggest items that match both item characteristics and user behavior.

Movie recommendation systems aim to suggest movies that match the user's preferences based on their data. Here, we opt to use collaborative filtering to get movie recommendations. In particular, we plan to use the Alternating Least Squares (ALS) matrix factorization algorithm that is available within the Spark's Machine Learning Library - MLLib.

3.3 Generating Recommendations and evaluation

During training and validation, the model's accuracy is estimated using metrics such as Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). Their formulaes are as follows:

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - f(x_i))}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - f(x_i)|$$

where y_i is the actual recommendation, $f(x_i)$ is the predicted recommendation. x_i.

Apart from these accuracy metrics, we also plan to evaluate the recommender system's performance using metrics such as mean reciprocal rank (MRR), mean average precision @ K (MAP@K), and mean average recall @ K (MAR@K) to measure the quality of a ranked list of recommendations.

4 First Milestones

4.1 Dataset Accumulation

We download The Movies Dataset which was last updated on 9/2018. This includes information about movies released before July 2017 such as cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, and countries. Although the dataset has 7 files, we only intend to use 3 of them: ”movies-metadata”, ”links”, and ”ratings”, as they contain all the columns required for our recommendation system. We have loaded all the data into 3 separate Spark dataframes.

“movies-metadata” : file is the file that contains information on 45,000 movies in the Full MovieLens dataset, including posters, backdrops, budget, revenue, release dates, languages, production countries, and companies.

“links” : file has the TMDB and IMDB IDs of all the movies in the Full MovieLens dataset.

“ratings” : file contains 27,000,000 ratings applied to 58,000 movies by 280,000 users.

The ‘links’ file is the bridge between ‘ratings’ and ’movie metadata’ and is used to perform filters/joins among the data.

4.2 Data Pre-Processing

4.2.1 Missing/Wrong Values:

We performed a thorough assessment of our selected datasets and checked for missing values in each column. These missing values are the ‘noise’ of the dataset and can cause our model to perform poorly. While we found all the rows with missing values, we only eliminated the rows that had missing values in specific columns that would cause issues in our training and exploratory data analysis.

“Ratings”: Thankfully, we discovered no missing values in the ”ratings.csv” file.

“Links” : Figure 3 shows the first 20 rows which contained missing values in the “tmbId” column. Figure 4 shows the total number of missing values in the dataset.

“Metadata”: Figure 5 shows the total number of missing values in each column of the dataset.

There were a few cases where some rows had to be dropped due to ‘wrong’ values such as having a release year of 22, 11, 1 and so on. These were discovered and dealt with during EDA.

4.2.2 Lowercase:

The next step we undertake is to convert all of our text in the ‘overview’ and ’title’ columns to lower text. This will help ensure that words with different capitalization will still be treated the same during EDA and word cloud generation.

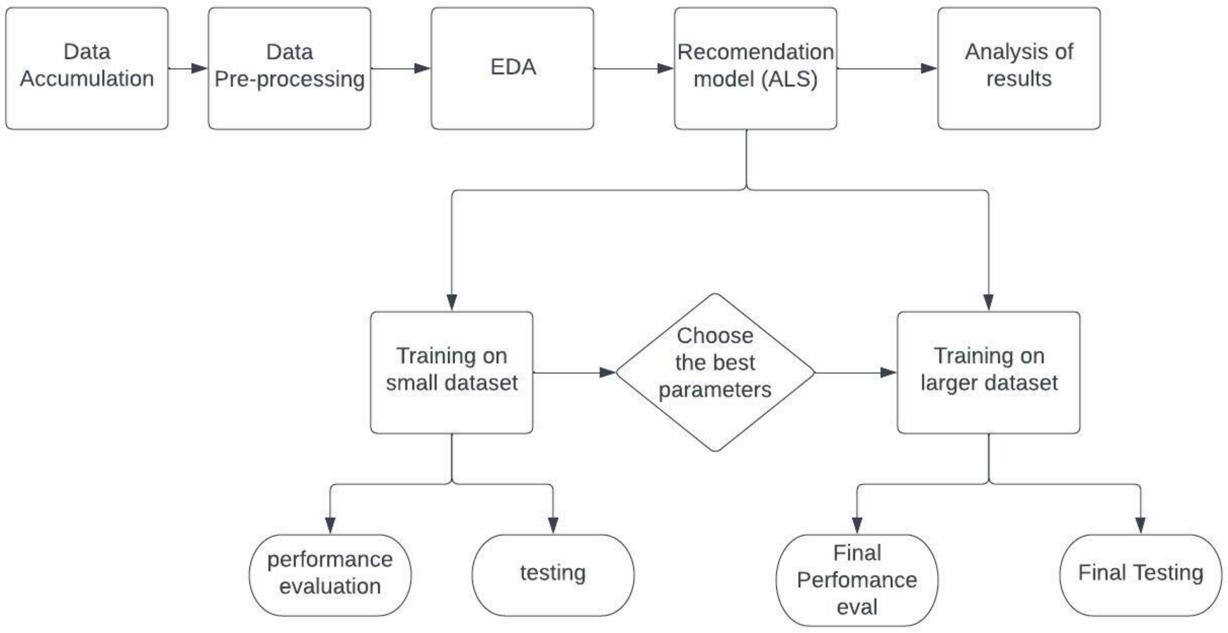


Figure 1. Work Flow of the diagram

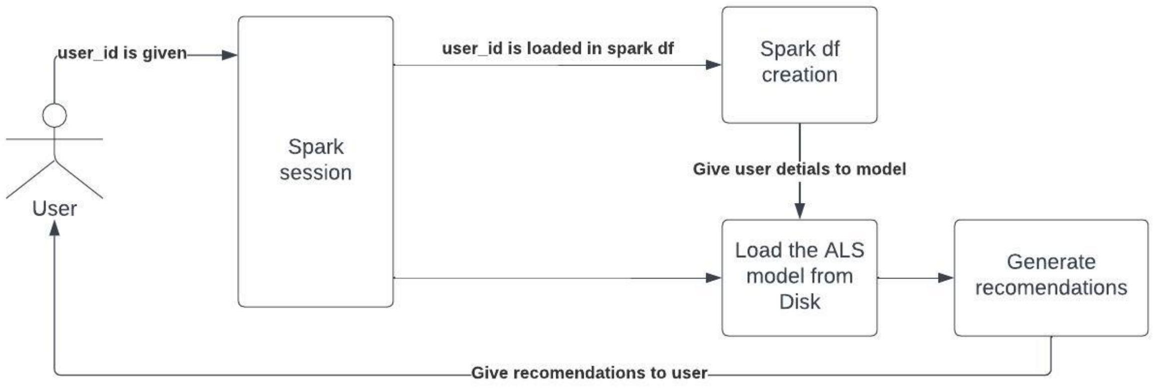
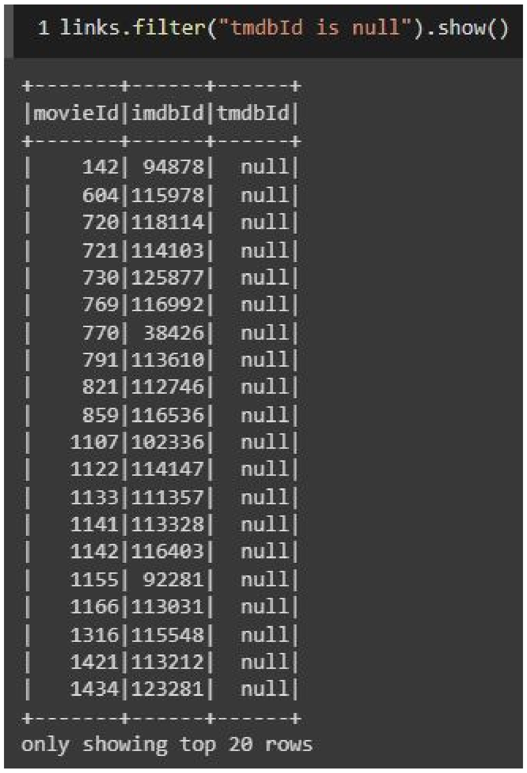

user_id is given
Spark session
user_id is loaded in spark df
Spark df creation
Give user detials to model
Load the ALS model from Disk
Generate recomendations
User
Give recomendations to user

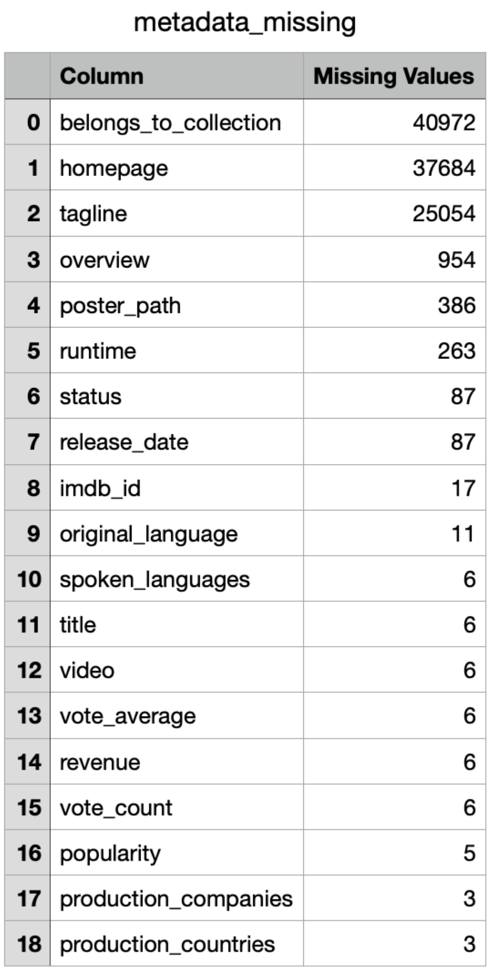
Figure 2. Usecase diagram for the proposed system



```
links.filter("tmdbId is null").show()

+-------+------+------+
|movieId|imdbId|tmdbId|
+-------+------+------+
|    142| 94878|  null|
|    604|115978|  null|
|    720|118114|  null|
|    721|114103|  null|
|    730|125877|  null|
|    769|116992|  null|
|    770| 38426|  null|
|    791|113610|  null|
|    821|112746|  null|
|    859|116536|  null|
|   1107|102336|  null|
|   1122|114147|  null|
|   1133|111357|  null|
|   1141|113328|  null|
|   1142|116403|  null|
|   1155| 92281|  null|
|   1166|113031|  null|
|   1316|115548|  null|
|   1421|113212|  null|
|   1434|123281|  null|
+-------+------+------+
only showing top 20 rows
```

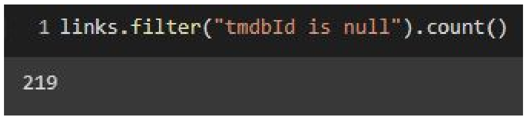
Figure 3. Rows with missing values in tmdbId column (Links file)



metadata_missing

	Column	Missing Values
0	belongs_to_collection	40972
1	homepage	37684
2	tagline	25054
3	overview	954
4	poster_path	386
5	runtime	263
6	status	87
7	release_date	87
8	imdb_id	17
9	original_language	11
10	spoken_languages	6
11	title	6
12	video	6
13	vote_average	6
14	revenue	6
15	vote_count	6
16	popularity	5
17	production_companies	3
18	production_countries	3

Figure 5. Count of missing values for metadata file



```
links.filter("tmdbId is null").count()

219
```

Figure 4. Count of missing values in tmdbId coloumn (Links file)

4.2.3 *Special Characters:*

Text data often contains special characters that can adversely impact our analysis. We decided to remove these special characters from our text data to prevent them from disrupting our analysis and to ensure that we obtained more accurate results.

4.2.4 *Stop words, lemmatization and Stemming:*

For the creation of WordCloud during EDA, we used nltk for removing stop words and converting each word into its *root* form for maximum overlap.

4.3 Data Filtering and Combining

After performing pre-processing on the data, we want to merge the data from the three different data files. To combine the data, we selected columns from the movies-metadata data file (title, overview, release date, and IMDB-id), the rating data file (user-id, movie-id, ratings, and timestamps), and the links data file (movie-id and IMDB-id) using the select command in Spark.

We then combined all of the above using joins. For this, first, we merged movies-metadata.csv and links.csv using the common column - IMDB-id with an inner join operation. Then, we merged the resulting dataset with ratings.csv using the common column movie-id with another inner join operation. We used the resulting dataframe for all further operations.

The final merged dataset contains columns for user-id, movie-id, title, rating, timestamps, and overview. Each user has a unique id, and each movie has a unique id. The title column contains the title of the movie, and the rating column contains how much the user rated that movie on a scale of 1-5. The timestamps column contains when a particular user rated the movie, and the overview column contains the summary of the movie. This information is summarized in Table 2.

Feature	Description
User id	Each user has an unique id
Movie id	Each movie had an unique id
Title	Contains the title of the movie.
Rating	On a scale of 1-5, how much each user has rated each movie
Timestamps	When did a paticular user rate the movie
Overview	Contains the summary of the movie.

Table 2. Final dataframe coloums

4.4 Exploratory data analysis

4.4.1 *Rating Distribution:*

Consider figure 6. Here, we plot the frequency at which a particular rating is given across all movies. We can also see the same statistic represented in a different way in figure 7 where we see how much of the total percentage of ratings go into each rating. An interesting observation is that the fractional ratings (0.5, 1.5, 2.5, 3.5, 4.5) are generally smaller in number compared to the whole number ratings. We can also observe that a considerable number of movies are rated 4.0, followed by 3.0. Surprisingly, more movies are rated 5 than 4.5 as mentioned earlier.

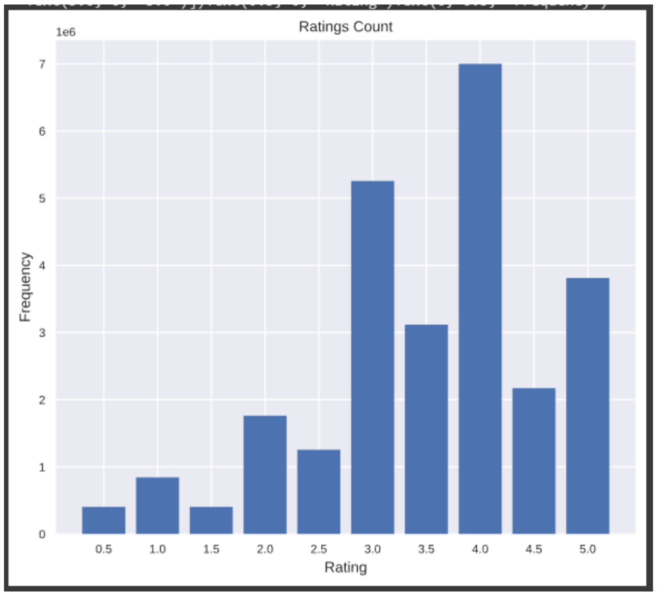


Figure 6. Rating Frequency

4.4.2 *Movie Release Distribution:*

Consider Figure 8. This displays the number of movies released per year, indicating that 2014 witnessed the highest number of movie releases. The number of movies released increased dramatically from the late 1980s and remains extremely high after 2005. In 2014, we can say that on an average, five movies were released per day, which is an outrageous number!. But with the rise of OTT platforms, it is somewhat explainable that such a high number of movies are being released worldwide.

4.4.3 *Analysis of the top 1000 movies in terms of Average Rating:*

Our focus was on identifying commonalities among the top 1000 movies. While there are many ways of defining a top movie (movie with most number of ratings, movie with

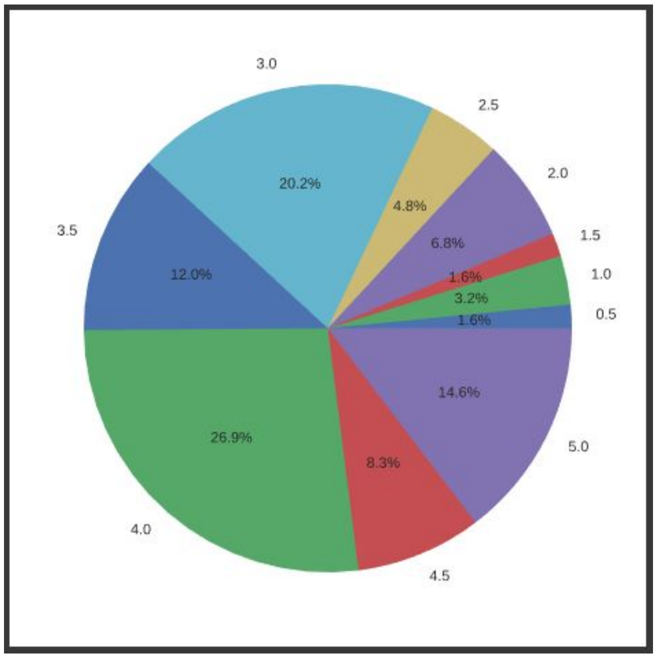


Figure 7. Rating frequency in terms of percentages

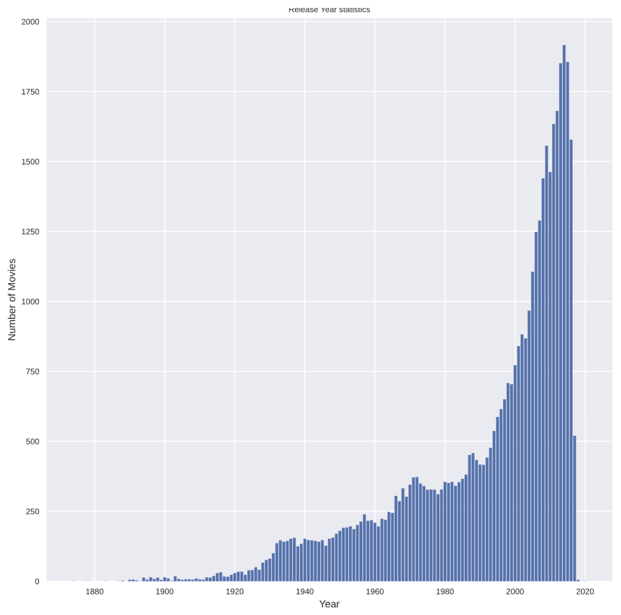


Figure 8. Release years of movies in the dataset

the highest average rating, movie with the highest budgest, movie with highest box-office earnings etc.), We defined the top movies based on having the highest average ratings. We put a restriction of at least 100 votes to remove outliers as there are many movies with a single or a few votes with very high rating.

Figure 9 illustrates the most common words found in the overviews of these top 1000 movies. This analysis can provide valuable information on the types and genres of movies that tend to be considered as 'top'.

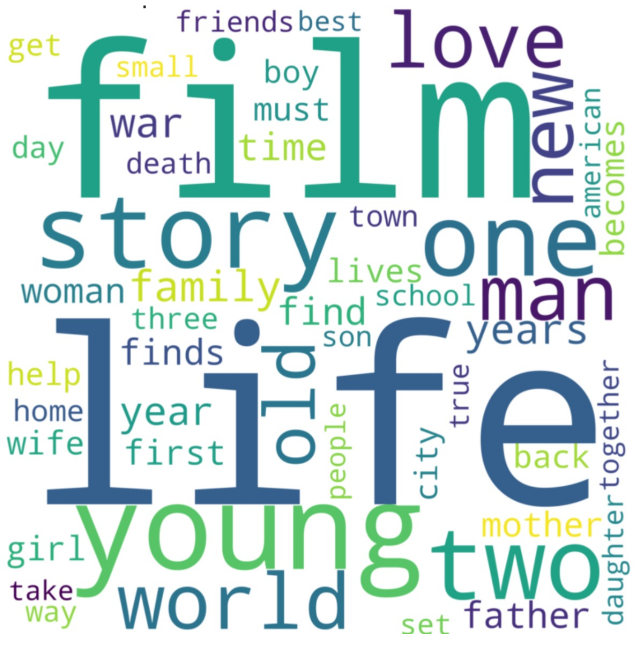


Figure 9. Word cloud of 1000 movies

4.4.4 *Analysis on one user's watch history:*

For analysis purposes, user ID 45811, with the highest number of votes, was selected. Figure 10 illustrates the word cloud (top 50 words) of all the movies that were watched and rated by this user. This shows that the user prefers movies with themes like family, life, love, and other similar topics. Word cloud analysis provides insights into the types of movies that the user is likely to enjoy based on the frequency of occurrence of these words in the overviews of all the movies they have previously watched.

5 Final Project Milestones

5.1 Recomendation model

The movie recommendation system that we are building aims to suggest movies that match user preferences based on their ratings data. Here, we are using collaborative fil-

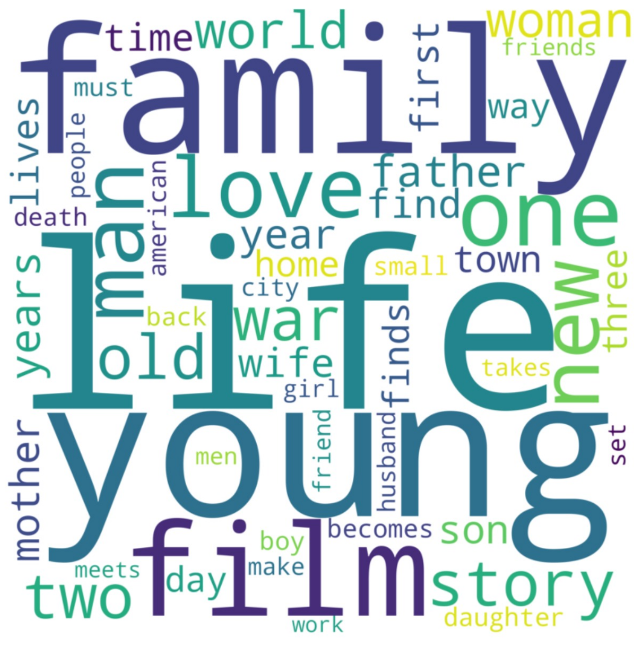


Figure 10. Word cloud of the user 45811

tering to get movie recommendations. In particular, we are using the Alternating Least Squares (ALS) matrix factorization algorithm that is available within the Spark's Machine Learning Library - MLLib.

ALS (Alternating Least Squares) is a collaborative filtering algorithm that uses matrix factorization technique to factorize a user-item interaction matrix into two lower-dimensional matrices, one for users and the other for items. These resulting low-dimensional representations of users and items can be used to make personalized recommendations to users based on their past interactions with items and the preferences of similar users. It can even be used to recommend the users best suited for a particular item (movie). ALS has been shown to be effective in many real-world recommendation scenarios and is widely used in industry.

5.2 Splitting the data

As we have mentioned before, we have a *Small* data split and a *Full* data split. Within each of them, we have done a 80 : 20 split into training and testing sets.

5.3 Training the model

From our final dataset, we have used the movie id column (item), user id column (user), and rating column (user-item interaction) to train our model.

The training happens in two phases. In phase 1, we first train the ALS model on a smaller subset of the data (*Small* split). This smaller dataset was used to ensure that the model was behaving as expected and that it was giving good recommendations. We also used the *Small* split to perform extensive analysis on the model performance with respect to hyper parameters.

We identify the best set of hyperparameters from Phase 1. After that, in Phase 2, we train the ALS model on the *Full* split. After the model is trained, we obtain predictions on the test dataset and we analyze the model performance. We also perform some sanity checks and testing to ensure that the model is performing as expected.

5.4 Hyperparameter tuning

The performance and quality of recommendations of the ALS model will be affected by the hyperparameters used to train the model. According to the Spark ALS documentation, there are three main hyperparameters – **rank**, **regParam**, and **maxIters**. The table 3 shows a description of the hyperparameters. The default parameters generally work well. Hence, we opted to perform a grid search of the hyperparameters around the default values to find the optimal combination of parameters.

As we vary the hyperparameters, we observe how the $RMSE$, and MAE changes to identify the optimal combination of hyperparameters.

Parameter	Description	Default
Rank	Number of latent factors. Larger number mean more intrinsic factors are considered in the factorization modeling.	10
RegParam	Regularization parameter. Value needs to be selected empirically to avoid overfitting.	0.1
MaxIters	Maximum number of iterations Higher the number, the better better the model converges to the optimal point.	10

Table 3. Hyperparameter description

We found from our sweep of the hyperparameters that $regParam = 0.1$ worked best (irrespective of the value of $rand$ and $maxIters$). Hence, we fixed the $regParam$ as 0.1 and visualized the effect on $RMSE$ and MAE with respect to $rank$ and $maxIters$ through a heatmap.

From the figures 12 and 11, we see that the $RMSE$ and MAE are the lowest at $rank = 5$, $maxIter = 15$, and $regParam = 0.1$. While this occurs in the left corner of the heatmap, it was not recommended to decrease the rank further and we stopped there. We have selected these hyperparameters as the optimal set for training our ALS model

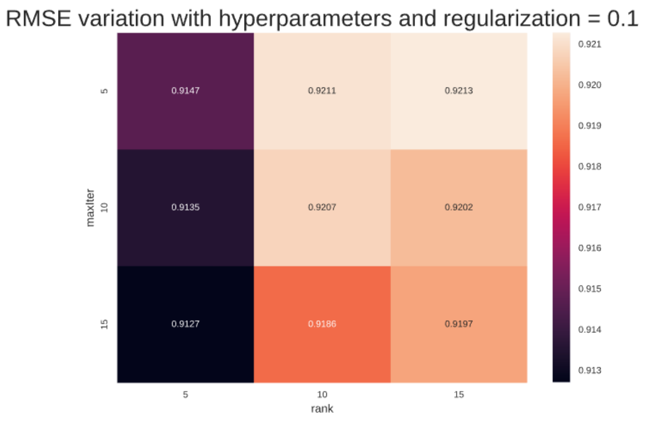


Figure 11. RMSE heatmap

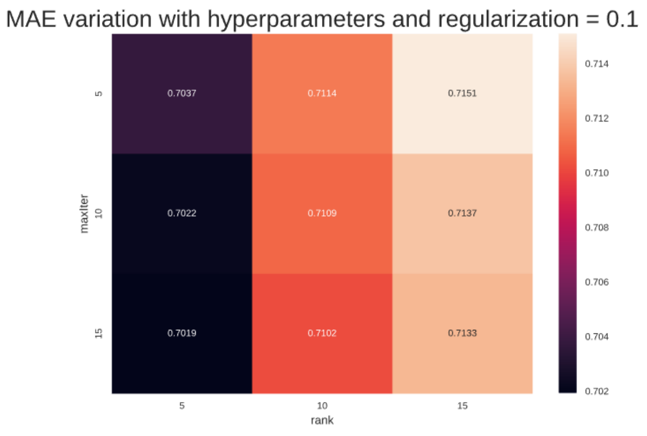


Figure 12. MAE heatmap

on both the *Small* and *Large* splits.

5.5 Testing

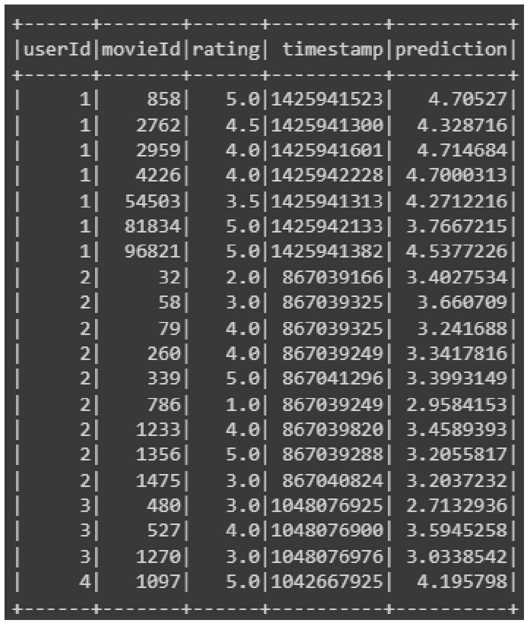

```
+------+-------+------+----------+----------+
|userId|movieId|rating| timestamp|prediction|
+------+-------+------+----------+----------+
|     1|    858|   5.0|1425941523|   4.70527|
|     1|   2762|   4.5|1425941300|  4.328716|
|     1|   2959|   4.0|1425941601|  4.714684|
|     1|   4226|   4.0|1425942228| 4.7000313|
|     1|  54503|   3.5|1425941313| 4.2712216|
|     1|  81834|   5.0|1425942133| 3.7667215|
|     1|  96821|   5.0|1425941382| 4.5377226|
|     2|     32|   2.0| 867039166| 3.4027534|
|     2|     58|   3.0| 867039325|  3.660709|
|     2|     79|   4.0| 867039325|  3.241688|
|     2|    260|   4.0| 867039249| 3.3417816|
|     2|    339|   5.0| 867041296| 3.3993149|
|     2|    786|   1.0| 867039249| 2.9584153|
|     2|   1233|   4.0| 867039820| 3.4589393|
|     2|   1356|   5.0| 867039288| 3.2055817|
|     2|   1475|   3.0| 867040824| 3.2037232|
|     3|    480|   3.0|1048076925| 2.7132936|
|     3|    527|   4.0|1048076900| 3.5945258|
|     3|   1270|   3.0|1048076976| 3.0338542|
|     4|   1097|   5.0|1042667925|  4.195798|
+------+-------+------+----------+----------+
```

Figure 13. Test set predictions of the model

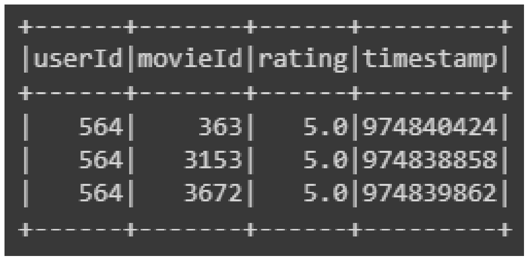

```
+------+-------+------+---------+
|userId|movieId|rating|timestamp|
+------+-------+------+---------+
|   564|    363|   5.0|974840424|
|   564|   3153|   5.0|974838858|
|   564|   3672|   5.0|974839862|
+------+-------+------+---------+
```

Figure 14. Ratings of user 564 on top recommended movies

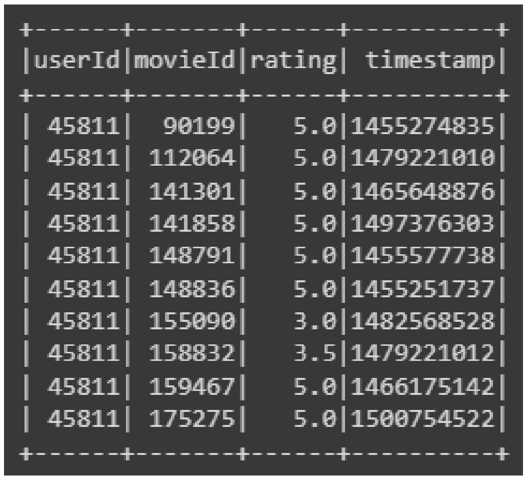
We perform sanity checks to ensure that the model is recommending relevant items (movies) to users. To do this, we perform three tests.

5.5.1 *Compare test predictions against their labels*

After the model has been trained on the train data, we take the predictions of the model on the testing data where an input of $userId$ and $movieId$ is given and the model tries to predict the rating. We can see the first 20 predictions of the test set in the figure 13 and we observe that the prediction column is very close to the actual ratings column.

5.5.2 *Compare recommendations against known data*

In case of both the *small* ratings split and *full* ratings split, we selected the $userId$ with the highest movie watch history



```
+------+-------+------+----------+
|userId|movieId|rating| timestamp|
+------+-------+------+----------+
| 45811|  90199|   5.0|1455274835|
| 45811| 112064|   5.0|1479221010|
| 45811| 141301|   5.0|1465648876|
| 45811| 141858|   5.0|1497376303|
| 45811| 148791|   5.0|1455577738|
| 45811| 148836|   5.0|1455251737|
| 45811| 155090|   3.0|1482568528|
| 45811| 158832|   3.5|1479221012|
| 45811| 159467|   5.0|1466175142|
| 45811| 175275|   5.0|1500754522|
+------+-------+------+----------+
```

Figure 15. Ratings of user 45811 on top recommended movies

(number of ratings). Then, we obtain 20 movie recommendations from the model for those $userId$. Once we have these recommendations ($movieIds$), we check our ratings dataset for a rating assigned to one of these recommended $movieIds$ by the $userId$.

We can see from figure 14 and 15 that most of the top recommendations have a high rating.

5.5.3 Compare movie overview with User's wordcloud

In case of $userId = 45811$, we further analyze the movies recommended. We choose three examples - one good prediction, one bad prediction and one unknown prediction as seen below:

Good prediction: Taraneh is a model 15-year-old Iranian girl, studious and filial, who supports her ailing grandmother with a job at a photo shop and visits her father (who has been imprisoned for reasons never made clear in the film) bearing gifts of cigarettes and magazines. But when Amir, a young man from a well-off family, sets his sights on Taraneh and courts her with an intensity that borders on stalking, her well-ordered life spirals into chaos.

Bad prediction: Roja lives in a Tamil village, and her sister is about to marry a man from the city, who decides to marry Roja instead and gets a job assignment in Kashmir, where some militants decide to kidnap them.

Unknown prediction: 30-year old Kertu has lived under her father's power her whole life. Because of her gentle nature, she is thought by locals to be a little simple-minded. The young woman makes her first timid attempt to change something in her life - she send a postcard to Villu, a handsome but degenerate village drunk.

When we compare these examples with the word cloud of user 45811 as shown in 10, we see that there is a lot of similarity among them. The word cloud has a large emphasis on words like young, life, love, family, film, story, girl/woman, boy/man, son, father, mother, daughter, and war. These 3 movie recommendations also fall along criteria of young people, love, family, chaos/action, woman and man. Thus, we can say that the recommendations were relevant as there is a large overlap between the word cloud of the user and the synopsis of the movies recommended.

5.6 Experimental Results

After we identified the best hyperparameters using grid search, we trained an ALS model on the two train datasets obtained from the small and full ratings datasets. After the training was completed, we got predictions of the model on the corresponding test datasets and computed various performance metrics as listed in the tables 4 and 5.

Performance Metric	Score
RMSE	0.9148
MAE	0.7035
R2	0.2428
Explained variance score	0.2645
Precision@10	0.8904
Recall@10	0.5770
NDCG@10	0.9981
Mean average precision	0.5770

Table 4. Performance Metrics on the small ratings Test Set

Performance Metric	Score
RMSE	0.8246
MAE	0.6357
R2	0.4006
Explained variance score	0.4083
Precision@10	0.6967
Recall@10	0.7317
NDCG@10	0.9999
Mean average precision	0.7317

Table 5. Performance Metrics on the full ratings Test Set

6 SOFTWARE AND HARDWARE USED

We have opted to use **Spark** as our primary software for this project and have installed the necessary pyspark software. Spark is a well-suited tool for processing large datasets quickly and efficiently.

For our exploratory data analysis, we utilized several software packages, including NLTK, wordcloud, and matplotlib. NLTK was used for pre-processing text while wordcloud was employed to generate and produce visualizations of the word clouds. Finally, matplotlib was utilized to plot different graphs.

In addition to these packages, we also installed numpy and pandas to support our analysis. We were able to complete all of our analysis using Google Colab and did not require any specialized hardware.