Hybrid Recommendation System on Netflix Prize Data

Team 7

José Luis Garza, 1725240
Rahul Joshi, 1727600
Marvin Ostermaier, 1724032
Chien Sheng Liu, 1725115
Wafaa AbuObidAlla, 1619985

Abstract

In this era of flourishing information, how to reduce the opportunity cost when we make choices has become one of the most significant topics for discussion in the contemporary era. In this project, we mainly want to recommend movies that are most relevant to users' habits or preferences. We use a variety of recommendation systems based on user data, reviews, movie characteristics, and other features to obtain and Users are used to related movies.

1 Introduction

Recommending Items to customers is one of the major challenges for many companies around the world. Streaming Platforms have a particularly wide range of different customers, what leads to an even bigger challenge. In our report, we will discuss how we analyzed the Netflix-Prize[1] and MovieLens[2] datasets provided on Kaggle.com. Our aim is to use the past user interactions to predict movie rankings provided in the dataset.

For this purpose we are using different algorithms, comparing their performance based on certain evaluations and building the best model to predict the user ratings. To find the best model we mainly focused on different Collaborative, Content based filtering methods. A combination of both approaches is used in the end to build our final Hybrid-Recommender System.

[1]https://www.kaggle.com/netflix-inc/netflix-prize-data
[2]https://www.kaggle.com/grouplens/movielens-20m-dataset

2 Data Preparation

2.1 Data Acquisition

2.1.1 Netflix Prize Data

We are performing our experiments on Netflix Prize Data - Netflix held the Netflix Prize open competition for the best algorithm to predict user ratings for films. This dataset was used in that competition. We acquired a dataset that contained 4 text files in the following format :
MovieId:
CustomerID, Rating, Date
CustomerID, Rating, Date
CustomerID, Rating, Date
MovieId2:
CustomerID,Rating,Date
Some conversion was carried out to convert these text files and the above-mentioned format into a CSV file that can be easily accessed.

2.1.2 MovieLens Dataset

As the titles in Netflix Prize Data don't have important attributes such as description, director, cast, we decide to merge another dataset called The Movies Dataset which could help us in the content-based recommendation. This dataset contains metadata for all 45,000 movies listed in the Full MovieLens Dataset. Data points include cast, crew, plot keywords, budget, revenue, posters, release dates, languages, production companies, countries, TMDB vote counts, and vote averages.

2.2 Data Integration

To integrate these two datasets we try to find which are the attributes that could be used for similarity calculation. Looking into the dataset we find that we can only use title and release year

Dataset (D)	#Entities	#Attributes	List of attributes
Netflix Titles	17,770	4	movie_id, title, release year, genre
Netflix Ratings	100,000,000	4	user_id, movie_id, rating, rating_date
External Movie Dataset	45,466	24	adult, belongs_to_collection, budget, genres, homepage, id, imdb_id, original_language, original_title, overview, popularity, poster_path, production_companies, production_countries, release_date, revenue, runtime, spoken_languages, status, tagline, title, video, vote_average, vote_count

Table 1: Schema and basic profile of each dataset

as our attributes for processing. The following procedure and tools were used to integrate Netflix and MovieLens Dataset:-

1. Altova Mapforce: Using MapForce we created converted each dataset into an XML file with a pre-defined target schema. The attributes present in both transformed datasets are id, title, and release year. This dataset acts like an input to the next process we follow in the JAVA environment.

2 . WIntr Framework: The WInte.r framework (Lehmberg, 2015) provides methods for end-to-end data integration. The framework implements well-known methods for data preprocessing, schema matching, identity resolution, data fusion, and result evaluation. The methods are designed to be easily customizable by exchanging pre-defined building blocks, such as blockers, matching rules, similarity functions, and conflict resolution functions. We are listing some key takeaways from our Data integration process in the following points:-

• **Identity Resolution**:
Gold standard had 11 examples where 6 are positive and 5 are negative examples.release_year used as Blocking key which helped in reducing comparison between movies with reduction ratio of 98%. Title was used as the main attribute for comparison with a cut-off of 0.64. We used n-gram Jaccard tokenizer to calculate similarities between movies where n=3. 8215 correspondences were generated in a file which can be used for Data Fusion Purpose. Testing resultant correspondences against gold standard, we got a Precision, Recall, and F1 score of 1.

Group Size	Frequency
2	8008
3	58

Table 2: Group Size Distribution of 8066 elements

• **Data Fusion**
Our Fused Gold standard consisted of 5 examples. After running the code for Data Fusion on result of Identity Resolution, we got a fused XML file which contains the results of all the correspondences. We needed to fix the groups with a size 3 in the fused XML and analyze whether there are any duplicates in the Netflix Prize Data. We noticed that several titles had the keyword 'Bonus Material' which was causing duplicates to occur.We decided to manually fixed the 3 grouped sized titles by removing the title id with 'Bonus material'.

On Evaluating the results against gold standard, we got Attribute-specific Accuracy of 1 and 1 for Release Year and Title respectively. Overall, We can conclude that we were able to match 8066 elements between these two data set. We also update the ratings dataset by removing user and movie ratings from the full ratings data which could not be matched.

2.3 Data Exploration

In this subsection, we conduct Exploratory Data Analysis (EDA). Totally 26 different genres are in the data set.

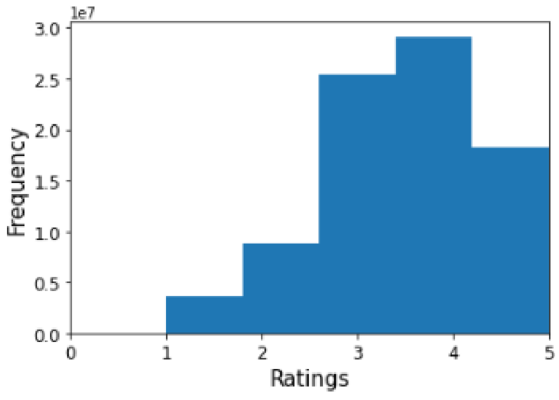


Figure 1: Rating distribution among users

The highest rating given by the user is 4 followed by 3. From the Figure(2) we can see that the graph has right skewed distribution which would result in very sparse matrix. So we need to remove some users and movies in order to decrease the sparseness, so that our calculations will be efficient.

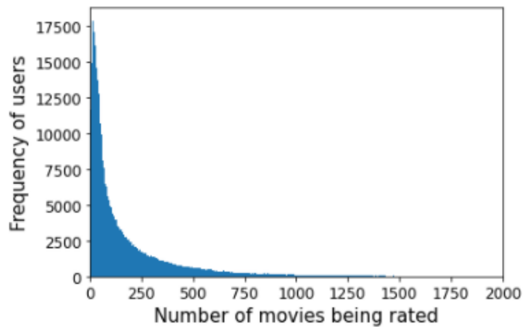


Figure 2: Distribution of the #users & #movies rated per user

2.4 Data Preprocessing

2.4.1 For Collaborative Filtering

For the data preprocessing of the CF approach, we removed users who rated movies less than 100 and more than 1750 movies and removed movies which is rated by less than 100 users.

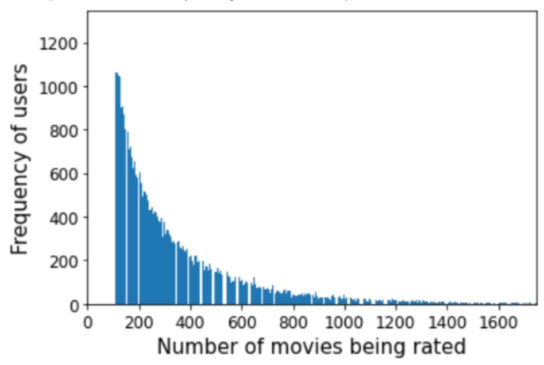


Figure 3: Distribution of the #users & #movies rated per user after preprocessing

2.4.2 For Content based Recommendations

For this purpose we created a column out of Genres, title and overview. To get our data in the right shape, we used different types of preprocessing. such as lemmatization, stop word removal and keeping only the alphanumeric values. We use attributes genre, overview and title and create a new column with the prefix '_metadata' for each attribute. This way embedding could be calculated separately and weights could be applied.

2.4.3 Truncate data for Hyperparameter Tuning

We have a large dataset of 100 million ratings which can be difficult to process if we are trying to train and test out different variants of model, so we only use 1 text file from the Netflix Prize data which consists of around 25 million ratings of 2043 titles by 467,350 users. After analyzing the rating distribution among users and movies we decide to remove users that have rated less than 100 movies and remove movies that have been rated by less than 100 users.

This results in 1998 titles and 57357 users with preserved rate of movies and users as 97.80% and 12.27% respectively. Then, we create a train test split of the truncated data with test_size of 25% and random_state of 42. It results in 3,199,119 record of training data and 1,066,374 record of test data.

2.5 Data Splitting

The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model. It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for our predictive modeling problem. We create a train test split of the preprocessed full ratings data with test_size of 25% and random_state of 42.

It results in 54,469,341 records of training data and 18,156,448 records of test data where there are 7891 titles and 215850 users with preserved rate of movies and users as 97.87% and 45.02% respectively.

3 Algorithms

3.1 Baseline Model

In order to provide context for the evaluation of more advanced recommendation models, We begin with constructing a very simple baseline model, where : Rating by User for Movie = Avg Rating of the movie. When Evaluating the model, we get a RMSE score of 0.9904. Although this could be hard to beat we try another variant of simple baseline model, where Rating by User for Movie = (Avg Rating of Movie + Avg Rating of User)/2.

When evaluating this model, we get a RMSE score of 0.9441, which is far better than our first approach and would be quite difficult to beat. So we choose our second model as our baseline model

3.2 Collaborative Filtering

Collaborative filtering is a technique that can filter out items that a user might like on the basis of reactions by similar users.

It works by searching a large group of people and finding a smaller set of users with tastes similar to a particular user. It looks at the items they like and combines them to create a ranked list of suggestions. There are many ways to decide which users are similar and combine their choices to create a list of recommendations. We are gonna play around with the Surprise Library and use prediction algorithms suchs KNNWithMeans, SVD, NMF, Co-Clustering, We will be using the truncated data as full data would take a lot of time to find out the best model among them. For finding out the best variant of the model we use **5 Fold** cross-validation on training data.

3.2.1 KNNWithMeans

The KNNWithMeans algorithm comes from the basic KNN algorithm which computes the class label of a record by taking a majority vote between the k nearest neighboring records in the dataset. For our collaborative filtering approach, the KNNWithMeans make sense as it considers the mean rating of each user. To get best parameters of our algorithm see, we use hypermeter parameters tuning with different K, as well we use Item based approach to compute the similarities between items and for the similarity measure, we use Cosine and Pearson Similarity.

By using hyper-parameter tuning previously mention, we found out that the best k parameter setting for our given data set was to use a k value of 30 and using Cosine Similarity as a similarity measure. This resulted in a RSME score of 0.863156 and MAE of 0.674094.

3.2.2 Singular Value Decomposition (SVD)

SVD is a matrix factorisation technique, which reduces the number of features of a dataset by reducing the space dimension from N-dimension to K-dimension (where K¡N). It uses a matrix structure where each row represents a user, and each column represents an item. The elements of this matrix are the ratings that are given to items by users. We are going to use the famous SVD algorithm, as popularized by Simon Funk[3] during the Netflix Prize. The prediction rating is set as:

$$\hat{r}_{ui} = \mu + b_u + b_i + q_i^T p_u \tag{1}$$

If user u is unknown, then the bias bu and the factors pu are assumed to be zero. The same applies for item i with bi and qi. To estimate all the unknown, we minimize the following regularized squared error:

$$\sum_{r_{ui} \epsilon R_{train}} (r_{ui} - \hat{r}_{ui})^2 + \lambda(b_i^2 + b_u^2 + ||q_i||^2 + ||p_u||^2) \tag{2}$$

The minimization is performed by a very straightforward stochastic gradient descent:

$$\begin{aligned} b_u &\leftarrow b_u + \gamma(e_{ui} - \lambda b_u) \\ b_i &\leftarrow b_i + \gamma(e_{ui} - \lambda b_i) \\ p_u &\leftarrow p_u + \gamma(e_{ui} \cdot q_i - \lambda p_u) \\ q_i &\leftarrow q_i + \gamma(e_{ui} \cdot p_u - \lambda q_i) \end{aligned} \tag{3}$$

where $e_{ui} = r_{ui} - \hat{r}_{ui}$.

3.2.3 Non Negative Matrix Factorization (NMF)

A collaborative filtering algorithm based on Non-negative Matrix Factorization. This algorithm is very similar to SVD. The prediction r_{ui} is set as:

$$\hat{r}_{ui} = q_i^T p_u \tag{4}$$

where user and item factors are kept positive. Our implementation follows that suggested in (Luo et al., 2014). Both are direct applications of NMF for dense matrices.The optimization procedure is a (regularized) stochastic gradient descent with a specific choice of step size that ensures non-negativity of factors. At each step of the SGD procedure, the factors f or user u and item i are updated as follows:

$$\begin{aligned} p_{uf} &\leftarrow p_{uf} \cdot \frac{\sum_{i \epsilon I_u} q_{if} \cdot r_{ui}}{\sum_{i \epsilon I_u} q_{if} \cdot \hat{r} + \lambda_u |I_u| p_{uf}} \\ q_{if} &\leftarrow q_{if} \cdot \frac{\sum_{u \epsilon U_i} p_{uf} \cdot r_{ui}}{\sum_{u \epsilon U_i} p_{uf} \cdot \hat{r}_{ui} + \lambda_i |U_i| q_{if}} \end{aligned} \tag{5}$$

where λ_u and λ_i are regularization parameters.

[3]https://sifter.org/ simon/journal/20061211.html

3.2.4 Co-Clustering

This is a straightforward implementation of (Thomas George, 2005). Basically, users and items are assigned some clusters Cu, Ci, and some co-clusters Cui. Clusters are assigned using a straightforward optimization method, much like k-means. The prediction $\hat{r}_{ui}$ is set as:

$$\hat{r}_{ui} = \overline{C_{ui}} + (\mu_u - \overline{C_u}) + (\mu_i - \overline{C_i}) \qquad (6)$$

where $\overline{C_{ui}}$ is the average rating of co-cluster $C_{ui}, \overline{C_u}$ is the average rating of u's cluster, and $\overline{C_i}$ is the average rating of i's cluster. If the user is unknown, the prediction is $\hat{r}_{ui} = \mu i$. If the item is unknown, the prediction is $\hat{r}_{ui} = \mu_U$. If both user and the item are unknown, the prediction is $\hat{r}_{ui} = \mu$.

3.3 Content Based Recommendations

To get a accurate Recommendation-System we are using a Content based approach to predict movies based on the top rated content by a user in the past. For this purpose we collected additional information for each movie to enrich the original Netflix-prize dataset. The additional features we are considering for our Content based filtering-Model were Genres, Language and Overview.

3.3.1 BERT Embeddings

BERT(Devlin et al., 2018) is a deep learning model that has given state-of-the-art results on a wide variety of natural language processing tasks. It stands for Bidirectional Encoder Representations for Transformers. It has been pre-trained on Wikipedia and BooksCorpus and requires task-specific fine-tuning. For each sentences it gives out a vector with dimensions of 768.

To measure the similarity between movies based on the Bert embedding, we used cosine similarity between given movies metadata which is preprocessed already by applying lemmatization, stop word removal and keeping only the alphanumeric values.

To know which user likes a specific content, we got all movie rated by the user, sort it in a descending order and select one movie from the top 5 movie randomly and show similar movies to the user based on embedding comparison of the top movie and all the other inputs. We also explore weighted approach which gives far better results than just combining together all the metadata into one column and calculating the vector embedding.

3.4 Evaluate best Model

We are gonna use the RMSE to evaluate our best model on truncated training data. Root mean squared error (RMSE) is the square root of the mean of the square of all of the error. The use of RMSE is very common, and it is considered an excellent general-purpose error metric for numerical predictions.where N is the number of observations available for analysis given by:

$$RMSE = \sqrt{\frac{\sum_{i=i}^{N} (Predicted_i - Actual_i)^2}{N}} \qquad (7)$$

After running 5 Fold cross validation of truncated training data on different variants of all models we get these result as show in the tables below.

We choose the two best models according to RMSE : KNNWithMeans and SVD. Furthermore, We perform hyper-parameter tuning on SVD to get the best out of SVD algorithm which is explained in the next sub-section.

Algorithm KNNWithMeans item-based	RMSE	MAE
k=30, sim=cosine	**0.863156**	**0.674094**
(k=40, sim=cosine,	0.864992	0.676136
k=50, sim=cosine	0.867302	0.678379
k=40, sim=pearson	0.869570	0.678852
k=50, sim=pearson	0.869728	0.679470
k=30, sim=pearson	0.871423	0.679999

Table 3: Cross validation results of KNNWithMeans

Algorithm	RMSE	MAE
SVD(n_factors=10)	0.867617	0.680086
SVD(n_factors=15)	0.866663	0.679213
SVD(n_factors=20)	**0.866285**	**0.678765**
SVD(n_factors=50)	0.867448	0.679721
SVD(n_factors=100)	0.869149	0.680600

Table 4: Cross validation results of SVD

Algorithm	RMSE	MAE
NMF(n_factors=10)	0.920045	0.731944
NMF(n_factors=15)	**0.904520**	**0.709816**
NMF(n_factors=30)	0.934385	0.720568
NMF(n_factors=50)	0.989072 8	0.759516

Table 5: Cross validation results of NMF

Algorithm variant	RMSE	MAE
n_cltr_u=5,n_cltr_i=5	**0.913118**	**0.717387**
n_cltr_u=15,n_cltr_i=15	0.929670	0.730353
n_cltr_u=30,n_cltr_i=30	0.9398118	0.737375

Table 6: Cross validation results of CoClustering

Algorithm	RMSE	MAE
KNNWith Means (k=30, sim=cosine, item-based)	**0.863156**	**0.674094**
SVD (n_factors=20)	**0.866285**	**0.678765**
NMF (n_factors=15)	0.904520	0.709816
CoClustering n_cltr_u=5, n_cltr_i=5	0.913118	0.717387

Table 7: Best Cross validated models

3.4.1 Hyperparameter Tuning of SVD

Surprise leverages scikit-learn's GridSearchCV() methods and in just a few lines returns a set of tuned parameters bypassing the manual, trial-and-error process of adjusting hyperparameters. GridSearchCV() calculates a score for each combination of hyperparameters on a k-fold cross validated dataset and returns the set of parameters that minimises the mean score across folds. Both the number of folds and the score can be selected by the user— we use **3 folds** and RMSE accuracy score. These are the following parameters that we decide to perform GridSearch on :

n_epochs: 20,30,50,
lr_all: 0.005,0.002,0.01
reg_all: [0.02,0.1],
n_factors: [20,30,40]

We add 30, 40 as nfactors to gridsearch because the lowest value of RMSE was with 20 factors, and nfactor with 50 was the next variant with lowest RMSE so we need to check in between them.

After running the gridsearch, we get the following results :
• Best params by RMSE - **n_epochs**: 50, **lr_all**: 0.01, **reg_all**: 0.1, **n_factors**: 40
• Best params by MAE - **n_epochs**: 50, **lr_all**: 0.002, **reg_all**: 0.02, **n_factors**: 20

We choose the model with best RMSE = 0.8694 for training it with full ratings train data.

3.4.2 Evaluating SVD

We test the best model of SVD by training the model with full truncated data and fit test data onto it. We get a RMSE score of 0.8624 by which we can assume that SVD is working quite well.

4 Evaluation and Result

4.1 Evaluate Best Models on Training Data

4.1.1 KNNWithMeans

After running the hyper-parameter tuning, we use our training and testing data to evaluate the performance of the model and we got a RMSE score of 0.85 using the following parameters:
• Number of K = 30.
• Similarity Measure: Cosine Similarity .
• Using the Item Based Approach

4.1.2 SVD

After getting the best parameters for SVD by running cross-validation and hyper parameter tuning, we train the model on full training data and fit test data onto it.
• No. of Training data: 54,469,341.
• No. of Test data: 18,156,448.
• Parameters used to train the model:
n_epochs: 50
lr_all: 0.01,
reg_all: 0.1,
n_factors: 40
• RMSE score on full test data = 0.8658

4.1.3 BERT

After creating embedding for all the metadata attributes separately, we examined results based on different weights. After further analyzation of different results we came to the conclusion that the similarity between items is best when using weights of 0.35,0.2,0.45 for genre, overview and title respectively. Prediction results are presented in the Prediction Results section.

4.2 Build Hybrid Model

Hybrid recommender systems combine two or more recommendation techniques to gain better performance with fewer of the drawbacks of any individual one. Most commonly, collaborative filtering is combined with some other technique in an attempt to avoid the ramp-up problem.

A weighted hybrid recommender is one in which the score of a recommended item is computed from the results of all of the available recommendation techniques present in the system. For example, the simplest combined hybrid would be a linear combination of recommendation scores. We read the saved test data predictions by users from KNNWithMeans and SVD and try to perform hybrid recommendation with different weights.

Weight SVD	Weight KNN	RMSE Score
0.80	0.20	0.8590
0.75	0.25	0.8577
0.7	0.3	0.8565
0.65	0.35	0.8555
0.6	0.4	0.8545
0.55	0.45	0.8538
0.5	0.5	0.8532
0.45	0.55	0.8527
0.4	0.6)	0.8523
0.35	0.65	0.8522
0.3	**0.7**	**0.8521**
0.25	0.75	0.8522
0.20	0.80	0.8524

Table 8: Demonstration of weight's effect on RMSE score

As we can see from the Table 8., RMSE score for the weights 0.3 and 0.7 is greater than other combinations so we select these weights to predict the user rating for a movie.

$$hybrid_{ui} = 0.3 \times (svd_{ui}) + 0.7 \times (KNN_{ui})$$

4.3 Evaluate and Compare Hybrid Model

In addition to RMSE scores, we try to explore more evaluation metrics such as :

a) Precision@K: Precision means: ”of all examples I predicted to be TRUE, how many were actually TRUE?”. It is simply Precision evaluated only up to the kth prediction, i.e.:

$$Precesion\ @\ k = \frac{truepositives\ @\ k}{(truepositives\ @\ k) + (falsepositives\ @\ k)} \tag{8}$$

b) Recall@K: It means: ”of all examples that were actually TRUE, how many I predicted to be TRUE?”. Recall@K is simply Recall evaluated only up to the kth prediction, i.e.:

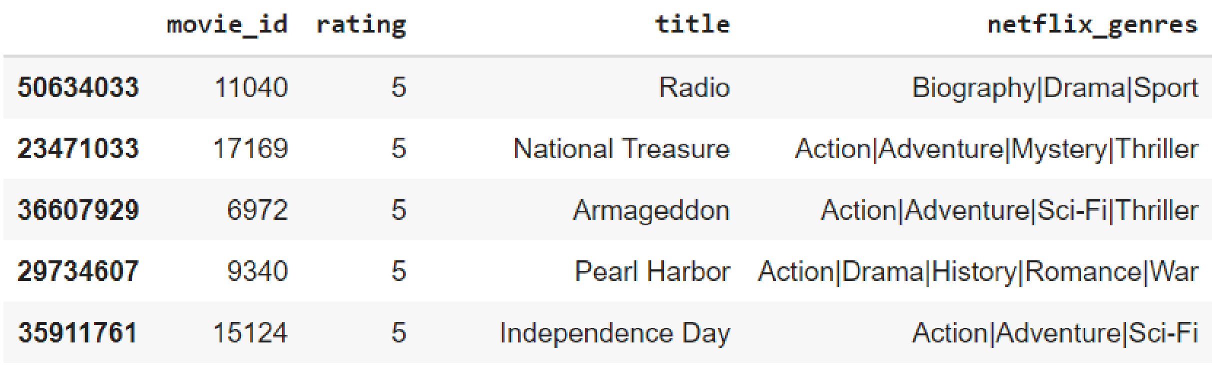
$$Recall\ @\ k = \frac{number\ of\ true\ labels\ captured}{number\ of\ true\ labels} \tag{9}$$

Metric	Rating Threshold	Baseline	KNN
RMSE	-	0.9441	0.8549
Precision@5	3.5	0.6193	0.8597
Precision@5	4	0.1765	0.8046
Precision@10	3.5	0.6193	**0.8315**
Precision@10	4	0.1765	0.7907
Recall@5	3.5	0.0597	0.1356
Recall@5	4	0.0171	0.1043
Recall@10	3.5	0.1194	0.2493
Recall@10	4	0.0302	0.1748

Table 9: Metric scores for different models (Part 1)

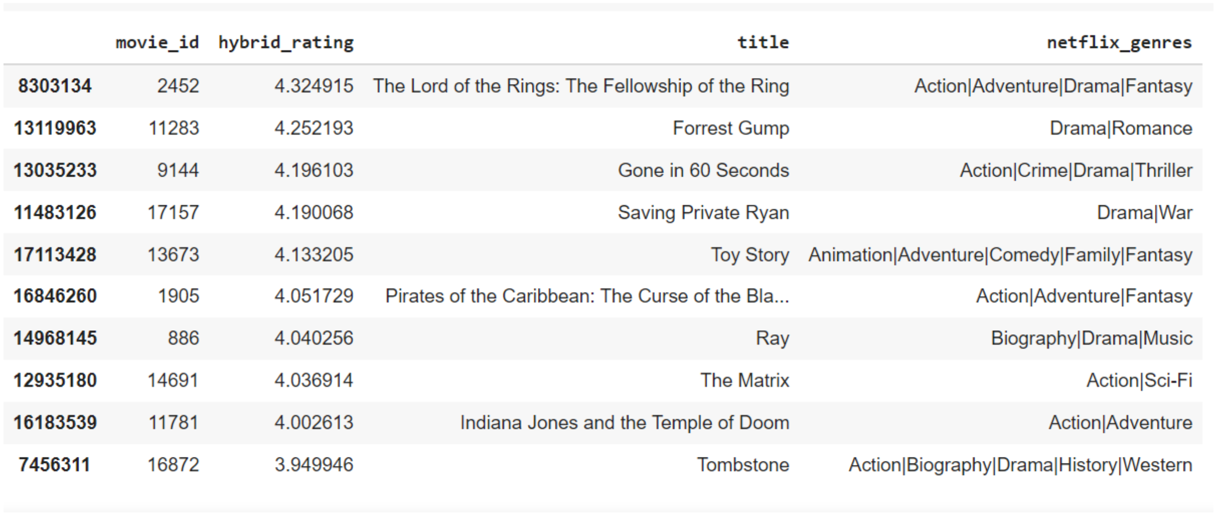
Metric	Rating Threshold	SVD	Hybrid
RMSE	-	0.8658	**0.8521**
Precision@5	3.5	0.8460	0.8606
Precision@5	4	0.7565	0.7943
Precision@10	3.5	0.8180	0.8327
Precision@10	4	0.7459	0.7820
Recall@5	3.5	0.1336	0.1357
Recall@5	4	0.0931	0.1010
Recall@10	3.5	0.2451	0.2493
Recall@10	4	0.1529	0.1679

Table 10: Metric scores for different models (Part 2)



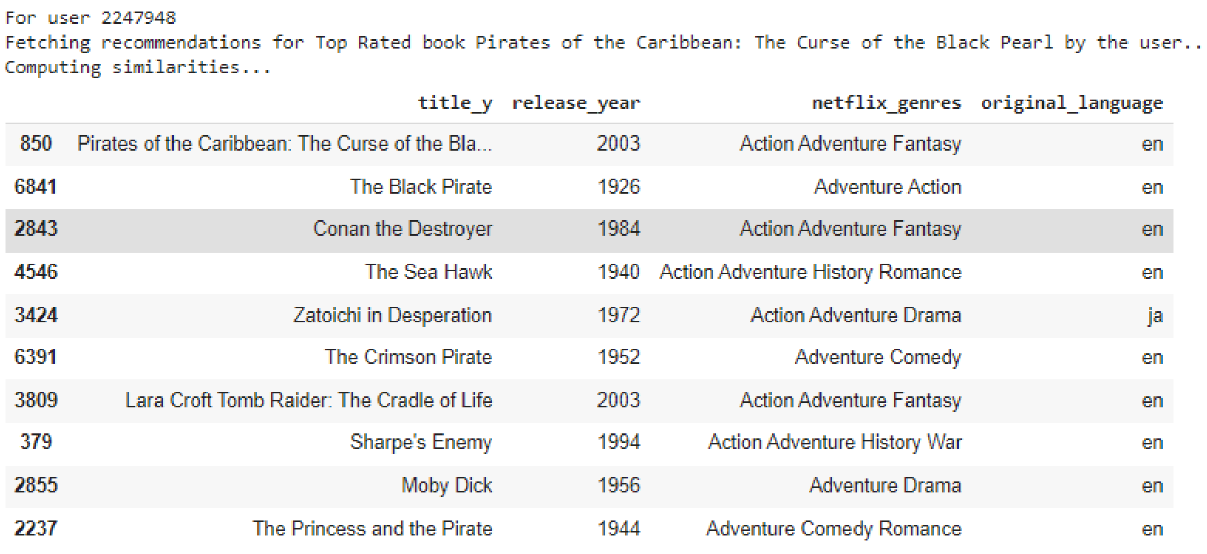
	movie_id	rating	title	netflix_genres
50634033	11040	5	Radio	Biography|Drama|Sport
23471033	17169	5	National Treasure	Action|Adventure|Mystery|Thriller
36607929	6972	5	Armageddon	Action|Adventure|Sci-Fi|Thriller
29734607	9340	5	Pearl Harbor	Action|Drama|History|Romance|War
35911761	15124	5	Independence Day	Action|Adventure|Sci-Fi

Figure 4: Top rated movies by User 653392



	movie_id	hybrid_rating	title	netflix_genres
8303134	2452	4.324915	The Lord of the Rings: The Fellowship of the Ring	Action|Adventure|Drama|Fantasy
13119963	11283	4.252193	Forrest Gump	Drama|Romance
13035233	9144	4.196103	Gone in 60 Seconds	Action|Crime|Drama|Thriller
11483126	17157	4.190068	Saving Private Ryan	Drama|War
17113428	13673	4.133205	Toy Story	Animation|Adventure|Comedy|Family|Fantasy
16846260	1905	4.051729	Pirates of the Caribbean: The Curse of the Bla...	Action|Adventure|Fantasy
14968145	886	4.040256	Ray	Biography|Drama|Music
12935180	14691	4.036914	The Matrix	Action|Sci-Fi
16183539	11781	4.002613	Indiana Jones and the Temple of Doom	Action|Adventure
7456311	16872	3.949946	Tombstone	Action|Biography|Drama|History|Western

Figure 5: Movies recommended for User 653392



```
For user 2247948
Fetching recommendations for Top Rated book Pirates of the Caribbean: The Curse of the Black Pearl by the user..
Computing similarities...
```

	title_y	release_year	netflix_genres	original_language
850	Pirates of the Caribbean: The Curse of the Bla...	2003	Action Adventure Fantasy	en
6841	The Black Pirate	1926	Adventure Action	en
2843	Conan the Destroyer	1984	Action Adventure Fantasy	en
4546	The Sea Hawk	1940	Action Adventure History Romance	en
3424	Zatoichi in Desperation	1972	Action Adventure Drama	ja
6391	The Crimson Pirate	1952	Adventure Comedy	en
3809	Lara Croft Tomb Raider: The Cradle of Life	2003	Action Adventure Fantasy	en
379	Sharpe's Enemy	1994	Action Adventure History War	en
2855	Moby Dick	1956	Adventure Drama	en
2237	The Princess and the Pirate	1944	Adventure Comedy Romance	en

Figure 6: Results from BERT Content Based Recommendation

4.4 Prediction Results

1. Recommendations based on user's rating preference using Hybrid Recommender in Figure (4, 5).

2. Recommendation of Similar Movies to the top rated movies by the user in Figure(6).

5 Conclusion

The project has been very enriching, as we finally were able to put into practice the theory learned through the course. Having to come up a functioning and effective pipeline that would work properly and that would be able to handle larger amounts of data, means that we use our analytic and technical skills as a team.

Despite the challenging, after so much researching and testing, we are certain that the lessons gathered will be useful in the near future.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Brinkmann A. Bizer C. Lehmberg, O. 2015. Winte. r - a web data integration framework. ISWC: 2017.

Xin Luo, Mengchu Zhou, Yunni Xia, and Qingsheng Zhu. 2014. An efficient non-negative matrix-factorization-based approach to collaborative filtering for recommender systems. *IEEE Transactions on Industrial Informatics*, 10(2):1273–1284.

Srujana Merugu Thomas George. 2005. A scalable collaborative filtering framework based on co-clustering.