University of Piraeus
Department of Digital Systems

Data Mining and Predictive Analytics

Project Title	Movie suggestions by utilizing a hybrid recommendation approach
Name:	Panagiotakopoulos Georgios
Registration No	me2030
Supervisor	Maria Chalkidi
Deadline	18/02/2021

Contents

Abstract

Movies are one of the most popular forms of entertainment and they are available in large numbers. Consequently, people need assistance in deciding what movie to watch. As a result, movie providers must deal with the problem of recommending to their users/customers movies that will satisfy them. Due to the highly competitive video streaming market, streaming platforms aim to create recommendation systems that fulfill their customer needs. This report aims to evaluate a few existing recommendation approaches and present a hybrid recommendation system. Content-based filtering makes recommendations of items based on the content of the items. Collaborative-based filtering such as user-based and item-based are two of the most popular techniques and were a part of our evaluation along with matrix factorization approaches such as NMF, SVD and SVD++. The results showed that the best approach was the combination of a content-based model and a matrix factorization using SVD++. The main characteristic of SVD++ is that it introduces the implicit feedback information based on classic SVD. To accomplish that it adds factor vector for each item in order to describe the characteristics of the item, regardless of whether it has been evaluated. Our final hybrid recommendation system managed to recommend the top predicted movies that the user has not yet rated. The system was highly accurate and showed diversity on its genre variety. Other advantages of this approach beyond the accuracy and diversity are the scalability of the overall system due to the dimensionality reduction, the nullifying of the cold start issue due to the SVD++, the ability to recommend to users with unpopular tastes due to the content-based layer. The movie recommendation systems are a constant effort for improvement due to the highly competitive video streaming market, the constantly changing environment and the demand of the consumers for good recommendations in order to stay loyal and engaged.

1. Introduction

1.1. Problem Statement

One of the most common types of entertainment is watching movies. According to statista [1], 800 new movies were released in 2019, in the USA and Canada. People across the globe are watching movies either at home through streaming platforms and dvd or at cinema.

Since the number of available movies is enormous, people need assistance in deciding what movie to watch. As a result, movie providers must deal with the problem of recommending to their users/customers movies that will satisfy them. Most movie providers such as subscription-based platforms have developed rating systems. Rating systems offer to the subscribers the opportunity to evaluate and rate previously watched movies. Moreover, they could check ratings, comments and reactions of other subscribers for movies that they have not yet seen.

However due to the highly competitive video streaming market, streaming platforms aim to create recommendation systems that fulfill their customer needs. Matching consumers with the most appropriate products is key to enhancing user satisfaction and loyalty. Therefore, more and more video streaming providers have shown interest recommender systems, which spot and analyze patterns of user interest in products to provide personalized recommendations that suit a user's preferences.

Nowadays data science is used to develop recommendation systems with accurate predictions. Machine learning and data mining techniques are used, in order to optimize the current recommendation systems and solve the mentioned problem.

1.2. Objectives

This report aims to evaluate a few existing recommendation approaches and present a hybrid recommendation system, that proposes movies to the user which he has not yet seen. The main objective is to recommend a list of top-rated movies factoring historical data of the user and of other users with similar characteristics. The predicted ratings of the user should be approximate enough so that the recommended list ensures the user's satisfaction.

2. Related work

Many solutions have been proposed in order to solve the movie recommendation problem. Main goals of the solutions are to minimize the response time of the queries and to optimize the prediction algorithms. Content platform and production company Netflix, Inc. [2] held a competition on 2006 [3] for the best optimization of their current recommendation system. BellKor's Pragmatic Chaos [4] won by proposing a solution that mainly utilizes feature engineering. A combination of a matrix factorization model, Restricted Boltzmann Machines (RBM) and a blending algorithm that is based on gradient boosted decision trees (GBDT) ensured the best proposed solution.

Website paperswithcode [5] hosts a competition for the best solution on MovieLens 1M Dataset [6]. The main winning criteria is the minimize of the Root Mean Square Error (RMSE). Currently Steffen Rendle, Li Zhang, Yehuda Koren [7] win the competition by proposing a Bayesian timeSVD++ flipped solution with a RMSE of 0.818. The paper A Neural Autoregressive Approach to Collaborative Filtering [8] focuses on a neural autoregressive architecture for collaborative filtering (CF) tasks (RMSE of 0.829) that combines RBM and the neural Autoregressive Distribution Estimator (NADE). On report Graph Convolutional Matrix Completion [9] they approach the recommendation issue by considering matrix completion for recommender systems from the point of view of link prediction on graphs. Movie rating could be visualized through a graph (user-item) with labeled edges that denote observed ratings. Their presented model reached high benchmarks in terms of collaborative filtering with an RMSE of 0.832. A study at University of Lille [10] considered a hybrid approach that combines matrix factorization [MF} techniques and autoencoders that are based on the theory of neural networks. Firstly, they used top of the field matrix factorization techniques to create the ratings matrix and then they optimized the autoencoders to handle incomplete data. Finally, they designed a system that could handle with accuracy any relevant external information. Their method reached an RMSE of 0.8321.

Based on the previously mentioned papers we could assume that there is great variety of methods and techniques that could be used to design and possibly improve a recommendation system. Techniques such as content-based filtering, collaborative filtering, reduction-based, graphs, Bayesian models, neural networks and deep learning in general could be used to achieve the desired results. Also, it is highlighted that the importance of having a system able to satisfy and keep loyal a client is so essential, that it is still a subject of study and analysis. Further implementations and improvements are inserted each year and the optimization of recommendation systems is a constant work in progress.

3. Description of the dataset

The dataset that will be analyzed is MovieLens 25M Dataset [11]. More specifically the dataset contains 25.000.095 movie ratings, one million tag applications which are related to 62.423 movies and 162.000 unique users. The dataset is stable and was last updated in December 2019. There are no null values contained among movies and ratings. A further analysis of the dataset using Python reveals the following:

- Ratings have a range of 0.5 to 5 and their distribution is depicted on diagram 1. It is shown that most ratings vary from 3-5 with 4 being the most common rating.

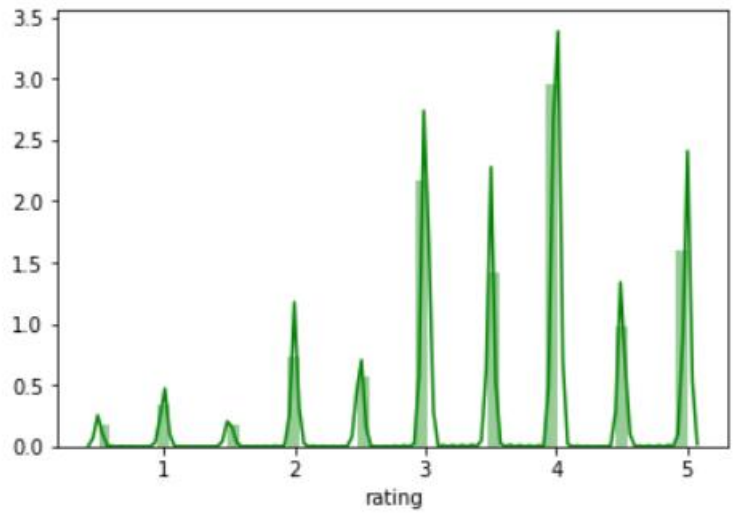


Diagram 1. Rating Distribution.

- Movies belong to various genres or a combination of them. Diagram 2 depicts the distribution of genres with Drama and Comedy being the most popular.

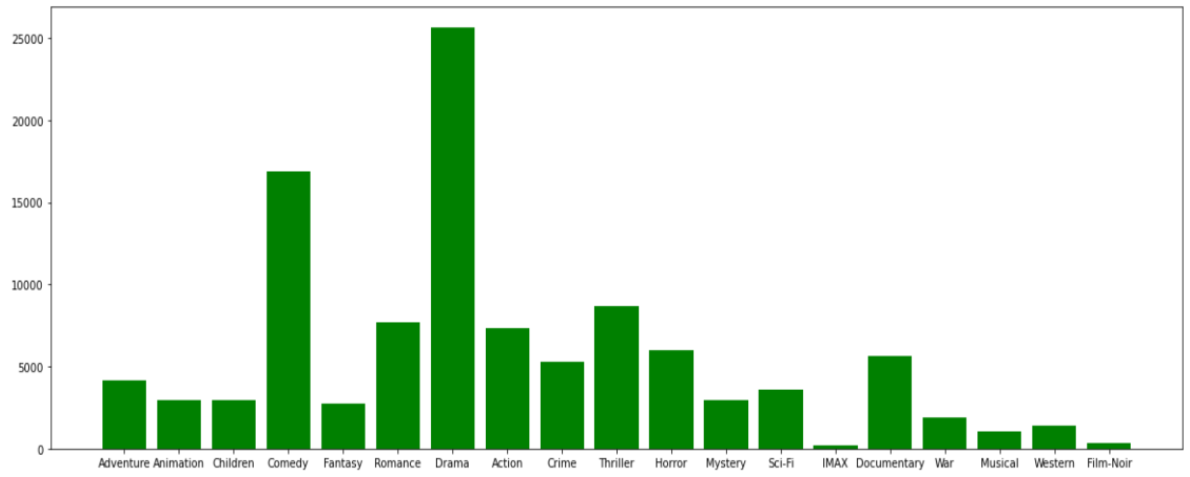


Diagram 2. Genres distribution.

- There are 5062 movies that are not assigned to a genre.
- None of the registered users has given less than 20 ratings.
- The 20 worst rated movies filtered by popularity are depicted on diagram 3.

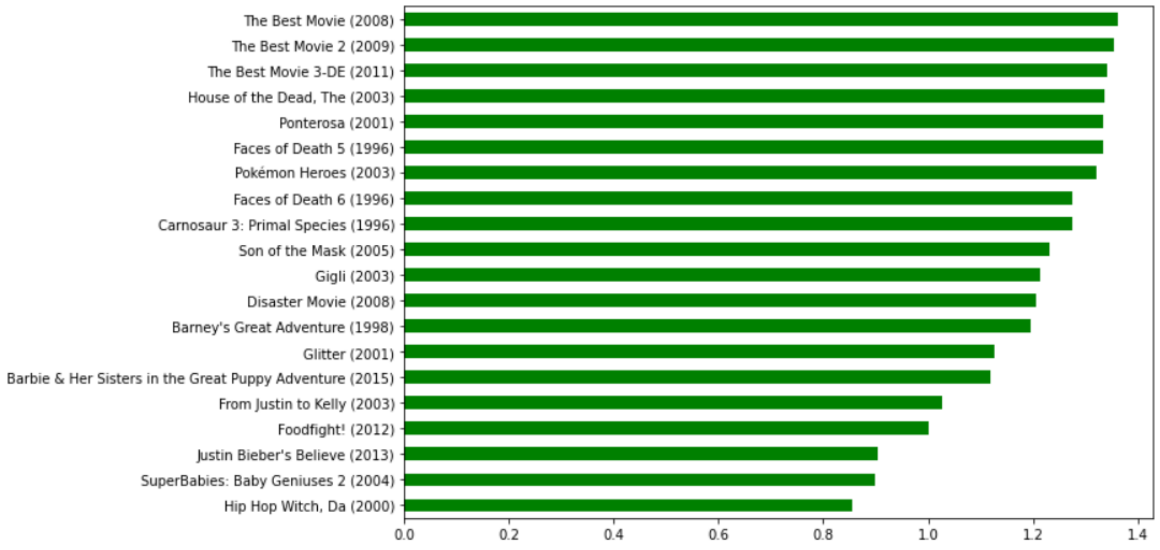


Diagram 3. Worst rated movies filtered by popularity.

- The top 20 rated movies filtered by popularity are depicted on diagram 4.

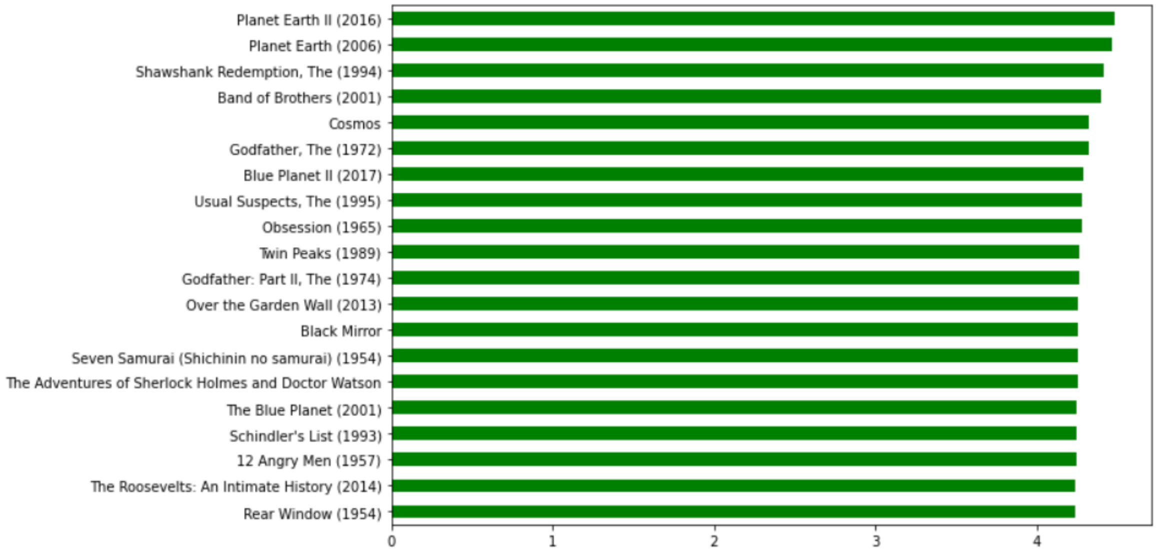


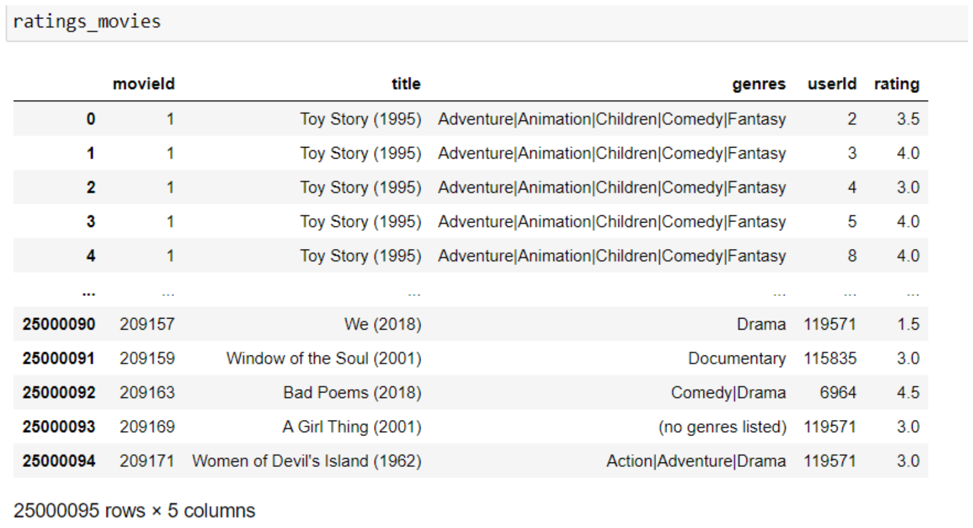
Diagram 4. Top rated movies filtered by popularity.

4. Methodology

4.1. Preprocessing

Beyond the first layer analysis that described on Section 3, it was necessary to do further preprocessing on the data in order to implement machine learning techniques. Particularly using the library Pandas [12], we did the following tasks:

- Creation of a dataframe called ratings_*movies* that contains joined data of movies and ratings.
- Key data needed for the project were movie id, movie title, movie genre, user id and rating of each user for each movie.
- We dropped the movies that were not listed on a genre.
- We dropped the column timestamp since it is not necessary on our next steps. The dataframe is shown on image 1.



```
ratings_movies
```

	movieId	title	genres	userId	rating
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	2	3.5
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	3	4.0
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	4	3.0
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	5	4.0
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	8	4.0
...	...	...	...	...	...
25000090	209157	We (2018)	Drama	119571	1.5
25000091	209159	Window of the Soul (2001)	Documentary	115835	3.0
25000092	209163	Bad Poems (2018)	Comedy|Drama	6964	4.5
25000093	209169	A Girl Thing (2001)	(no genres listed)	119571	3.0
25000094	209171	Women of Devil's Island (1962)	Action|Adventure|Drama	119571	3.0

25000095 rows × 5 columns

Image 1. Ratings_movies dataframe

One major issue that we had to overcome was the insufficient computing power to handle a dataset of that size. Most of the techniques that were used to design the recommendation system require lot of processing power especially in terms of memory. As a result, a decision was made to proceed with a sample of the 25M dataset. In order to create the new dataset, we filtered the 1001 best users in terms of movies watched and reviewed. In order to have a more stable dataset the first user was dropped because of the huge size of movies rated. The total number of ratings was 2.174.488 which was still too large to analyze. Finally, we took a sample of that and the final dataset contained 1000 users, 23518 movies and 217.449 ratings.

4.2. Content based filtering

Content-based filtering makes recommendations of items based on the content of the items. A comparison between the content of the items and the profile of a specific user leads to recommending items. Typically, words are the main descriptor of the item's content. In the context of movie recommendation system, we decided that genre is the term that suits better the content based filtering. To analyze with the content based filtering technique we need to implement two important concepts. Particularly, we need to compute a TF-IDF matrix and calculate the cosine similarity.

Term frequency commonly known as TF is the frequency of a word in a document [13]. It gives higher weight when a word occurs more frequently. Below is shown a normalized version of the TF type.

$$\text{Tf(t)}\frac{term\ (t)\ frequency\ in\ a\ document}{total\ terms\ in\ a\ document}$$

Inverse document frequency commonly known as IDF is the inverse of the document frequency among the whole corpus of documents. Basically, it gives higher weight to rare terms in documents thus signifying their importance.

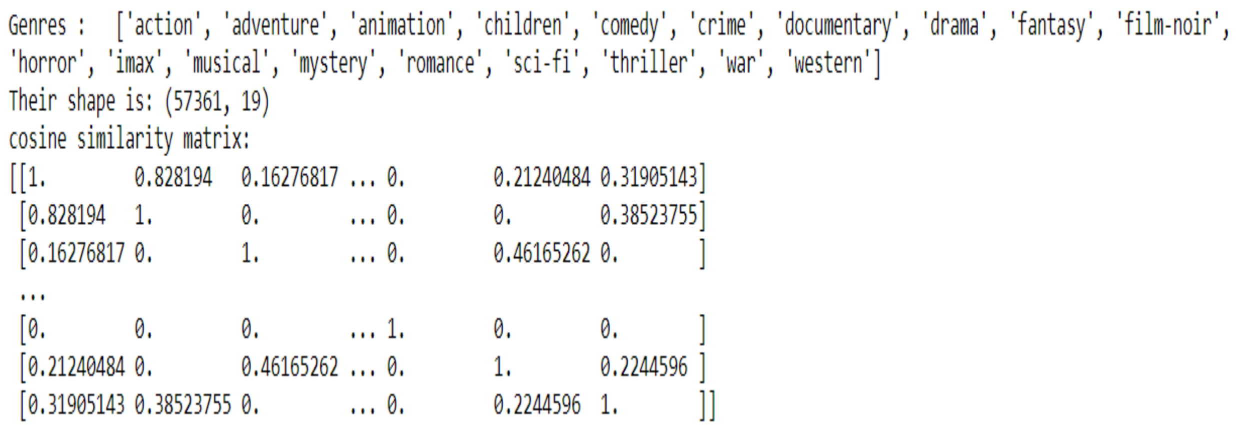
$$\text{Idtf(t)}=\log_{10}\frac{total\ number\ of\ documents}{number\ of\ documents\ with\ term\ (t)}$$

After calculating the TF-IDF scores we need to determine which items (movies) are closer to each other. For that we compute cosine similarity using the library scikit-learn [14].

Cosine similarity is the ratio between the dot product and the product of the magnitudes of two vectors A and B.

$$\text{Similarity}=\cos(\theta)=\frac{AB}{\|A\|\|B\|}$$

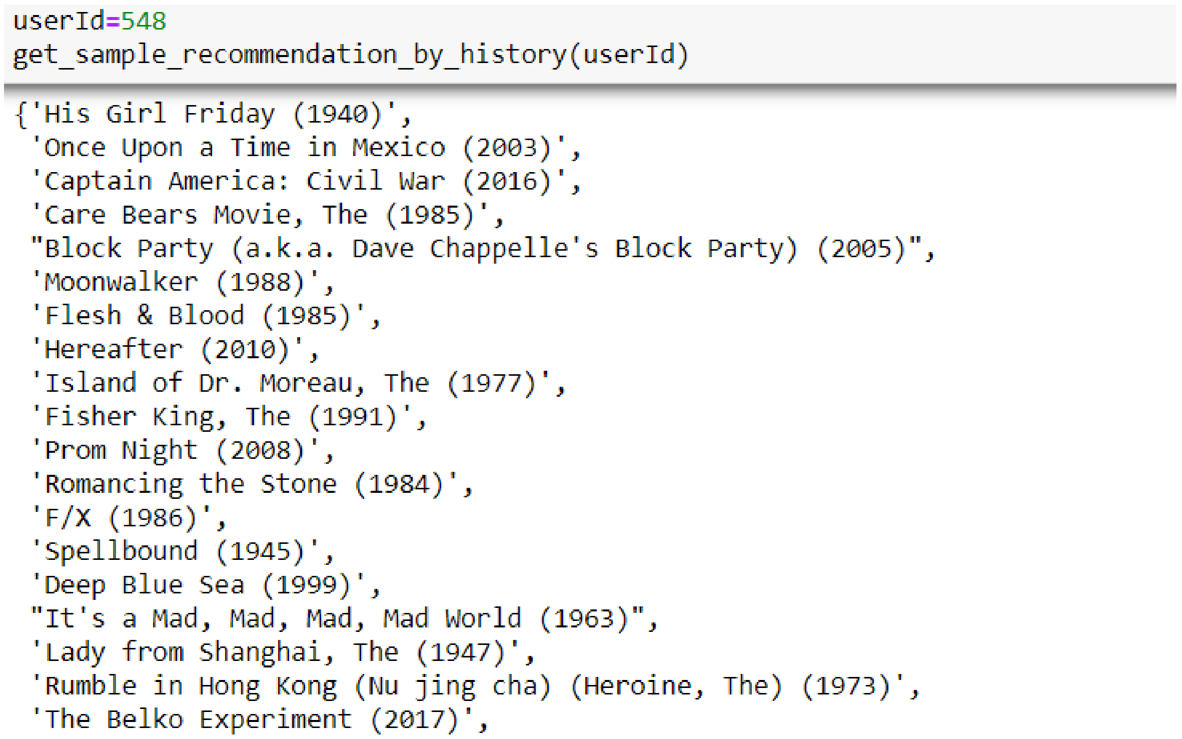
The cosine similarity matrix is shown on image 2. The closer the number is to 1 means the 2 vectors are identical. If the value is 0 it means that the 2 vectors are orthogonal.



```
Genres :  ['action', 'adventure', 'animation', 'children', 'comedy', 'crime', 'documentary', 'drama', 'fantasy', 'film-noir',
'horror', 'imax', 'musical', 'mystery', 'romance', 'sci-fi', 'thriller', 'war', 'western']
Their shape is: (57361, 19)
cosine similarity matrix:
[[1.         0.828194   0.16276817 ... 0.         0.21240484 0.31905143]
 [0.828194   1.         0.         ... 0.         0.         0.38523755]
 [0.16276817 0.         1.         ... 0.         0.46165262 0.        ]
 ...
 [0.         0.         0.         ... 1.         0.         0.        ]
 [0.21240484 0.         0.46165262 ... 0.         1.         0.2244596 ]
 [0.31905143 0.38523755 0.         ... 0.         0.2244596  1.        ]]
```

Image 2. TF-IDF genres and cosine similarity matrix

For a given user the system recommends a list of movies with genres similar to the ones he had seen before (image 3).



```
userId=548
get_sample_recommendation_by_history(userId)
```

```
{'His Girl Friday (1940)',
 'Once Upon a Time in Mexico (2003)',
 'Captain America: Civil War (2016)',
 'Care Bears Movie, The (1985)',
 "Block Party (a.k.a. Dave Chappelle's Block Party) (2005)",
 'Moonwalker (1988)',
 'Flesh & Blood (1985)',
 'Hereafter (2010)',
 'Island of Dr. Moreau, The (1977)',
 'Fisher King, The (1991)',
 'Prom Night (2008)',
 'Romancing the Stone (1984)',
 'F/X (1986)',
 'Spellbound (1945)',
 'Deep Blue Sea (1999)',
 "It's a Mad, Mad, Mad, Mad World (1963)",
 'Lady from Shanghai, The (1947)',
 'Rumble in Hong Kong (Nu jing cha) (Heroine, The) (1973)',
 'The Belko Experiment (2017)',
```

Image 3. List of recommended movies for user with Id 548 based on content based filtering

For user with id 548 the system found 1322 movies that he has not yet seen to recommend. Even though the system achieved in finding many films we should not ignore the limitations of content based recommenders. The fact that these types of recommenders fail to capture inter-dependencies or complex tendencies, highlights the importance of a hybrid recommendation system.

4.3. Item based collaborative filtering

Item-item filtering is one of the two memory based collaborative approaches. Its simple definition is 'users who enjoyed these items also enjoyed the following items'. Basically, it takes an item as input, find users that like that item and identifies other items that these users or similar ones like.

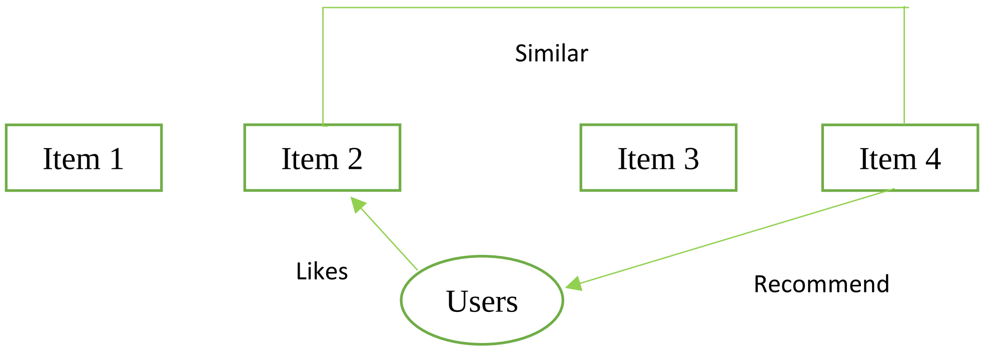


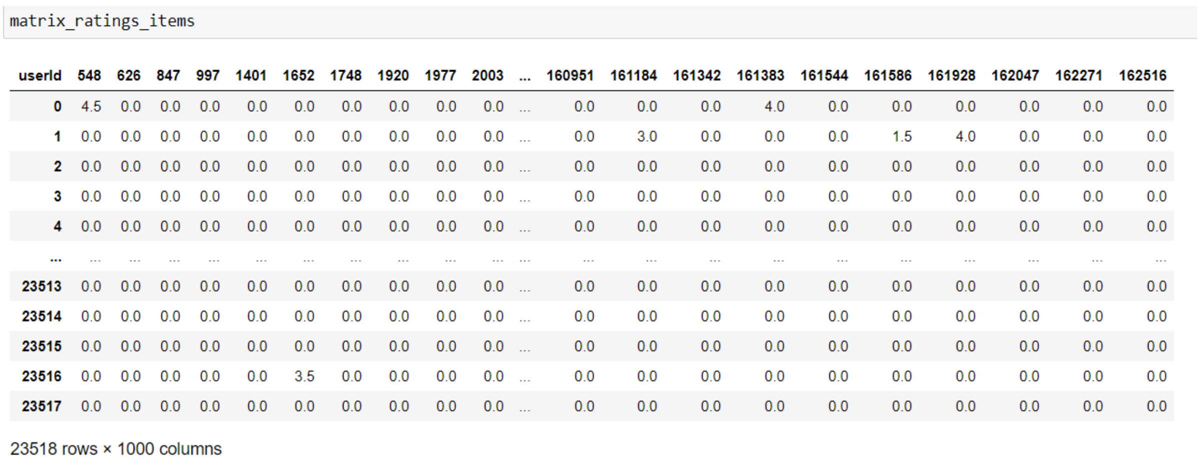
Diagram 5. Item based collaborative system

The core advantages of this approach are:

- Less resource consuming than user-user type.
- Better predictions for a new user.

The main disadvantage is that the simpler computation results in reduction of diversity. As a result, the recommendations are kind of obvious and not that differential.

For the item based collaborative filtering we need to create first a matrix with item-user ratings. The referred matrix is shown on image 4.



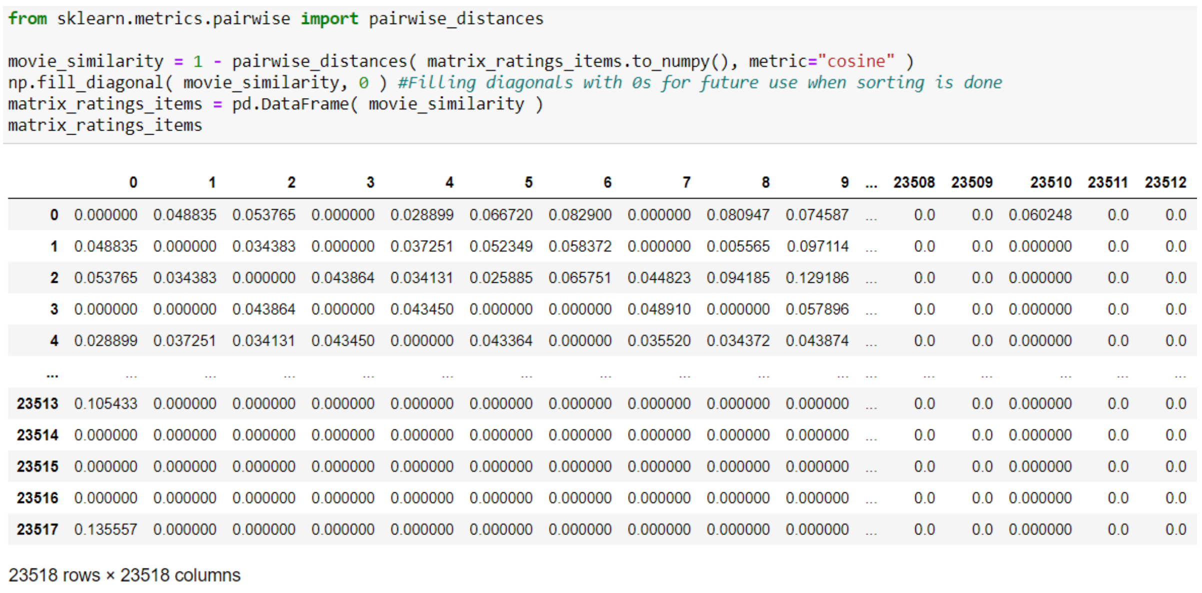
```
matrix_ratings_items
```

userId	548	626	847	997	1401	1652	1748	1920	1977	2003	...	160951	161184	161342	161383	161544	161586	161928	162047	162271	162516
0	4.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	4.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	3.0	0.0	0.0	0.0	1.5	4.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23513	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23514	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23515	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23516	0.0	0.0	0.0	0.0	0.0	3.5	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
23517	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

23518 rows × 1000 columns

Image 4. Matrix for item-item filtering.

For the next step we calculated the cosine similarity to find similar items with similar ratings for the dataset users. We used the pairwise distances function from Scikit-learn library [14] to effectively calculate the cosine similarity (image 5). The similarity values between items in Item-Item Collaborative Filtering are measured by observing all the users who have rated both items [15].



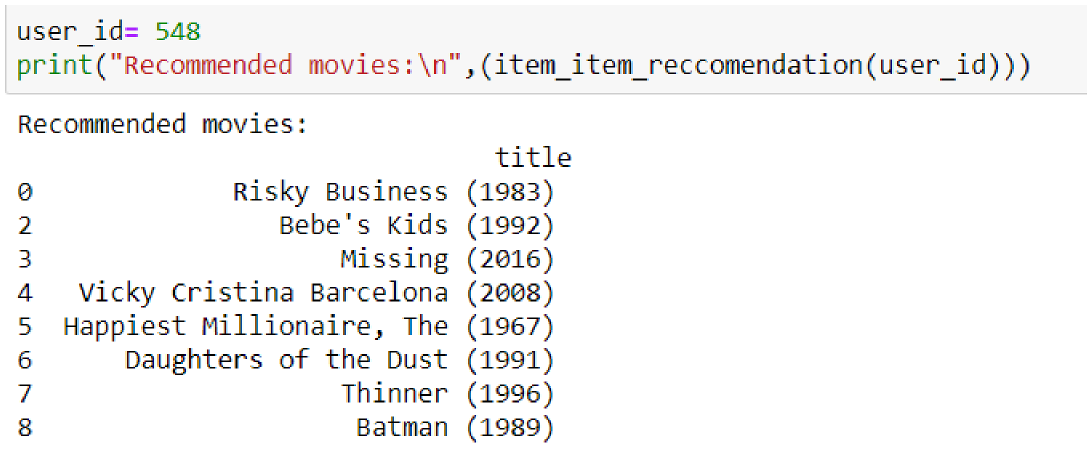
```
from sklearn.metrics.pairwise import pairwise_distances

movie_similarity = 1 - pairwise_distances( matrix_ratings_items.to_numpy(), metric="cosine" )
np.fill_diagonal( movie_similarity, 0 ) #Filling diagonals with 0s for future use when sorting is done
matrix_ratings_items = pd.DataFrame( movie_similarity )
matrix_ratings_items
```

	0	1	2	3	4	5	6	7	8	9	...	23508	23509	23510	23511	23512
0	0.000000	0.048835	0.053765	0.000000	0.028899	0.066720	0.082900	0.000000	0.080947	0.074587	...	0.0	0.0	0.060248	0.0	0.0
1	0.048835	0.000000	0.034383	0.000000	0.037251	0.052349	0.058372	0.000000	0.005565	0.097114	...	0.0	0.0	0.000000	0.0	0.0
2	0.053765	0.034383	0.000000	0.043864	0.034131	0.025885	0.065751	0.044823	0.094185	0.129186	...	0.0	0.0	0.000000	0.0	0.0
3	0.000000	0.000000	0.043864	0.000000	0.043450	0.000000	0.000000	0.048910	0.000000	0.057896	...	0.0	0.0	0.000000	0.0	0.0
4	0.028899	0.037251	0.034131	0.043450	0.000000	0.043364	0.000000	0.035520	0.034372	0.043874	...	0.0	0.0	0.000000	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
23513	0.105433	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.000000	0.0	0.0
23514	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.000000	0.0	0.0
23515	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.000000	0.0	0.0
23516	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.000000	0.0	0.0
23517	0.135557	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.0	0.0	0.000000	0.0	0.0

23518 rows × 23518 columns

Image 5. Matrix with cosine similarity calculated.

On the last step of this method, we filtered the dataset for top ratings (range of ratings between 4-5) and searched for a similarity above 0.25. The results were not that promising since the recommendation system managed to propose only 8 movies for our test user with id 548.



```
user_id= 548
print("Recommended movies:\n",(item_item_reccomendation(user_id)))
```

```
Recommended movies:
                                title
0              Risky Business (1983)
2                 Bebe's Kids (1992)
3                     Missing (2016)
4    Vicky Cristina Barcelona (2008)
5   Happiest Millionaire, The (1967)
6       Daughters of the Dust (1991)
7                     Thinner (1996)
8                      Batman (1989)
```

Image 6. Recommended movies based on item-item filtering.

For the evaluation of this approach, we used the Surprise library [16] to calculate the Root Mean Square Error and the Mean Absolute Error.

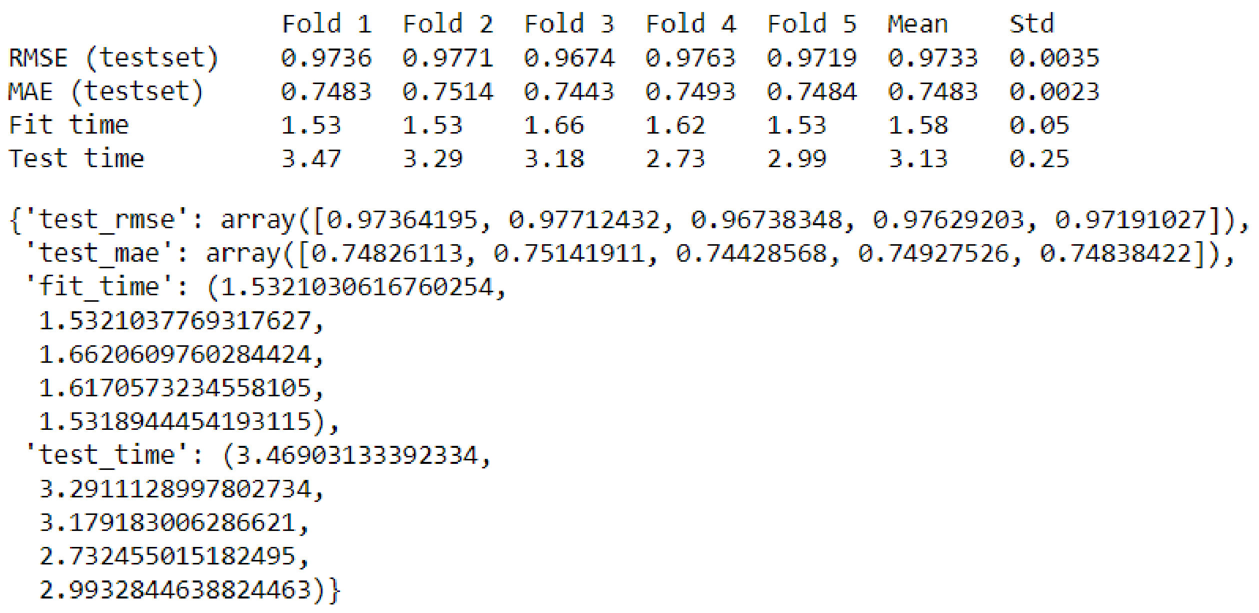
Root Mean Square Error (RMSE) [17] puts more emphasis on larger absolute error and when RMSE is low than accuracy of recommender system is better.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(P_{(u,i)} - R_{(u,i)})^2}{N}}$$

Mean Absolute Error (MAE) is calculated by deducting the actual rating R(u, i) from the predicted rating P(u, i) for user u on item i , R(u, i) with N being the total number of ratings. The lower the MAE the more accurate the recommendation system.

$$MAE = \frac{1}{N}\sum_{(u,i)} |P_{(u,i)} - R_{(u,i)}|$$

RMSE and MAE results using cross validation are the ones shown on image 7. The RMSE score is 0.9733 +- 0.0035 and the MAE score is 0.7483 +- 0.023.



```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9736  0.9771  0.9674  0.9763  0.9719  0.9733  0.0035
MAE (testset)     0.7483  0.7514  0.7443  0.7493  0.7484  0.7483  0.0023
Fit time          1.53    1.53    1.66    1.62    1.53    1.58    0.05
Test time         3.47    3.29    3.18    2.73    2.99    3.13    0.25

{'test_rmse': array([0.97364195, 0.97712432, 0.96738348, 0.97629203, 0.97191027]),
 'test_mae': array([0.74826113, 0.75141911, 0.74428568, 0.74927526, 0.74838422]),
 'fit_time': (1.5321030616760254,
  1.5321037769317627,
  1.6620609760284424,
  1.6170573234558105,
  1.5318944454193115),
 'test_time': (3.46903133392334,
  3.2911128997802734,
  3.179183006286621,
  2.732455015182495,
  2.9932844638824463)}
```

Image 7. RMSE and MAE scores for item-item filtering.

4.4. User based collaborative filtering

User based collaborative filtering is the remaining memory based filtering. Its simple definition is ‘users similar to you enjoyed these movies’. Basically, it takes a specific user, find users that are similar regarding their ratings and recommend items (movies) that those similar users liked.

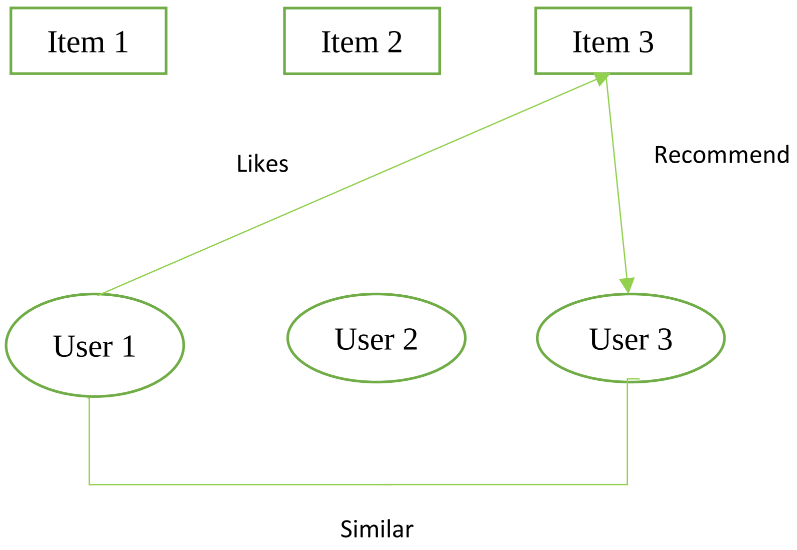
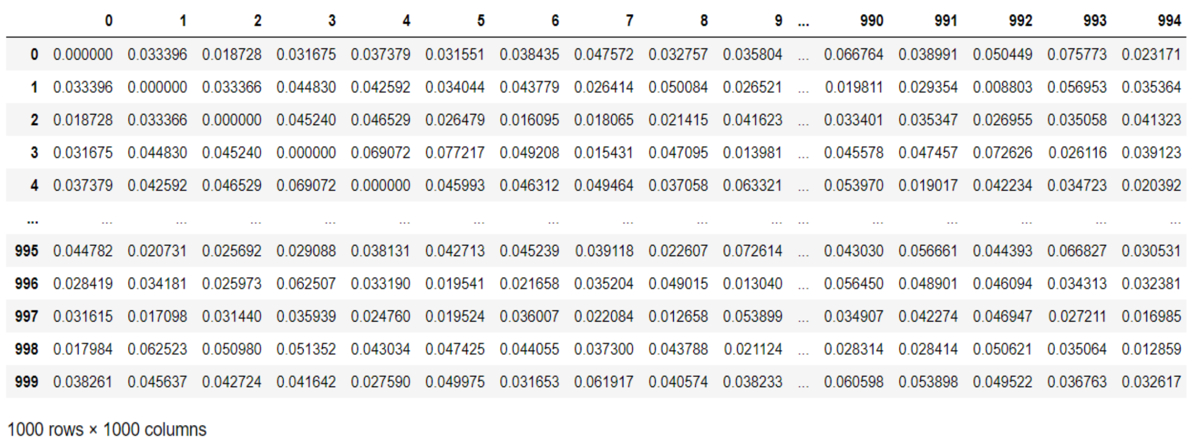


Diagram 6. User based collaborative filtering system.

This algorithm is pretty accurate but consumes a lot of time and resources. The reason is that it makes a computation for every pair of users. The main advantage of this algorithm is that the implementation is usually easy. However, it has two core disadvantages:

- Sparsity issue since generally there are not many users who rate items.
- The cold start issue which regards new users. Specifically, the system does not have information about their preferences in order to make recommendations [18].

For the user based collaborative filtering we need to create first a matrix with user-item ratings. The referred matrix is shown on image 7.
On the next step we calculate the cosine similarity matrix to find similar users based on their ratings.

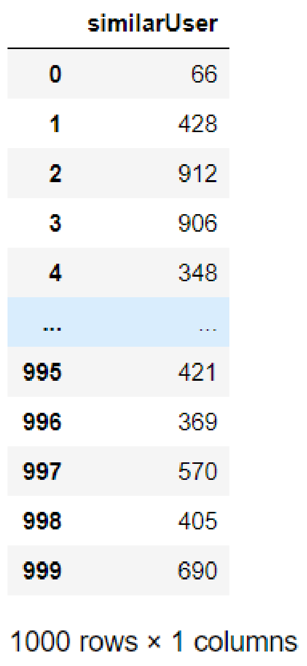


	0	1	2	3	4	5	6	7	8	9	...	990	991	992	993	994
0	0.000000	0.033396	0.018728	0.031675	0.037379	0.031551	0.038435	0.047572	0.032757	0.035804	...	0.066764	0.038991	0.050449	0.075773	0.023171
1	0.033396	0.000000	0.033366	0.044830	0.042592	0.034044	0.043779	0.026414	0.050084	0.026521	...	0.019811	0.029354	0.008803	0.056953	0.035364
2	0.018728	0.033366	0.000000	0.045240	0.046529	0.026479	0.016095	0.018065	0.021415	0.041623	...	0.033401	0.035347	0.026955	0.035058	0.041323
3	0.031675	0.044830	0.045240	0.000000	0.069072	0.077217	0.049208	0.015431	0.047095	0.013981	...	0.045578	0.047457	0.072626	0.026116	0.039123
4	0.037379	0.042592	0.046529	0.069072	0.000000	0.045993	0.046312	0.049464	0.037058	0.063321	...	0.053970	0.019017	0.042234	0.034723	0.020392
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
995	0.044782	0.020731	0.025692	0.029088	0.038131	0.042713	0.045239	0.039118	0.022607	0.072614	...	0.043030	0.056661	0.044393	0.066827	0.030531
996	0.028419	0.034181	0.025973	0.062507	0.033190	0.019541	0.021658	0.035204	0.049015	0.013040	...	0.056450	0.048901	0.046094	0.034313	0.032381
997	0.031615	0.017098	0.031440	0.035939	0.024760	0.019524	0.036007	0.022084	0.012658	0.053899	...	0.034907	0.042274	0.046947	0.027211	0.016985
998	0.017984	0.062523	0.050980	0.051352	0.043034	0.047425	0.044055	0.037300	0.043788	0.021124	...	0.028314	0.028414	0.050621	0.035064	0.012859
999	0.038261	0.045637	0.042724	0.041642	0.027590	0.049975	0.031653	0.061917	0.040574	0.038233	...	0.060598	0.053898	0.049522	0.036763	0.032617

1000 rows × 1000 columns

Image 8. Similarity matrix for user-user.

We need to spot the best pair for every similar user which is shown on image 9.

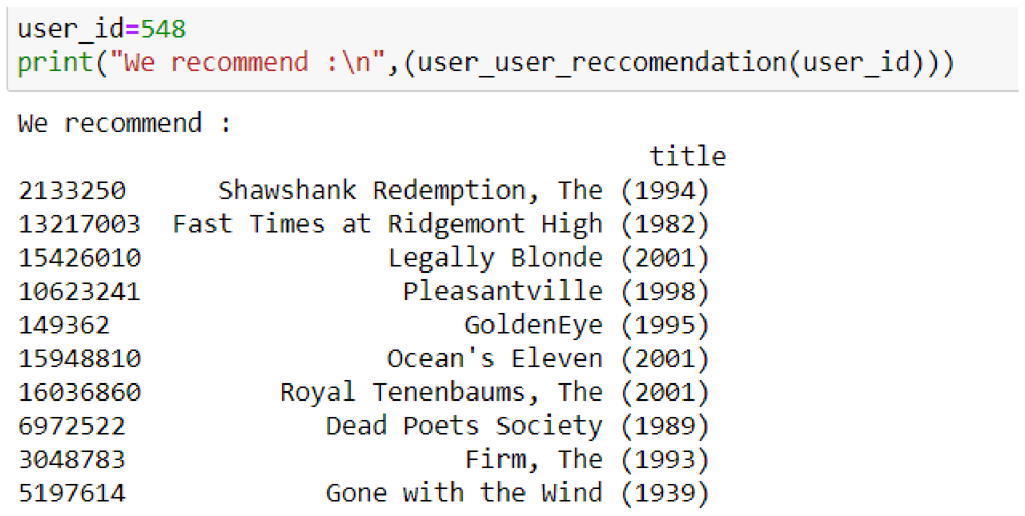


	similarUser
0	66
1	428
2	912
3	906
4	348
...	...
995	421
996	369
997	570
998	405
999	690

1000 rows × 1 columns

Image 9. Pairs of similar users.

On the last step the system recommends items that the user has not yet rated from the ones of its similar user. E.g. for user with id 548 the algorithm recommends the following movies:

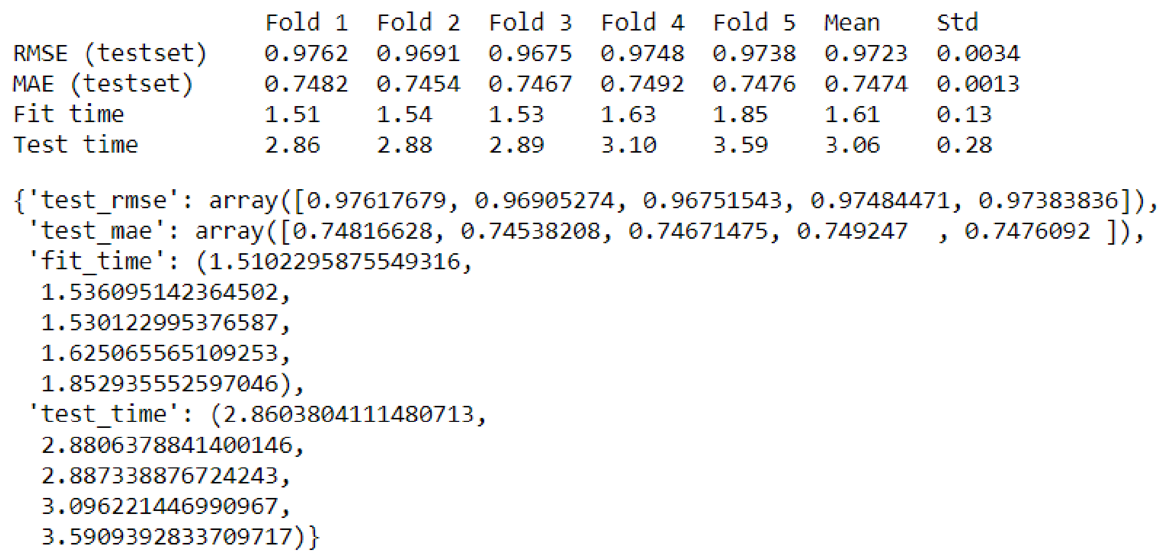


```
user_id=548
print("We recommend :\n",(user_user_reccomendation(user_id)))
```

```
We recommend :
                                           title
2133250      Shawshank Redemption, The (1994)
13217003  Fast Times at Ridgemont High (1982)
15426010                Legally Blonde (2001)
10623241                 Pleasantville (1998)
149362                       GoldenEye (1995)
15948810                Ocean's Eleven (2001)
16036860         Royal Tenenbaums, The (2001)
6972522             Dead Poets Society (1989)
3048783                     Firm, The (1993)
5197614             Gone with the Wind (1939)
```

Image 10. Recommended movies based on user-based filtering.

The user based system has marginally better results than the item based system in terms of their RMSE score which is 0.9723+- 0.0034 and MAE score which is 0.7474 +- 0.0013 (image 11).



```
                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9762  0.9691  0.9675  0.9748  0.9738  0.9723  0.0034
MAE (testset)     0.7482  0.7454  0.7467  0.7492  0.7476  0.7474  0.0013
Fit time          1.51    1.54    1.53    1.63    1.85    1.61    0.13
Test time         2.86    2.88    2.89    3.10    3.59    3.06    0.28

{'test_rmse': array([0.97617679, 0.96905274, 0.96751543, 0.97484471, 0.97383836]),
 'test_mae': array([0.74816628, 0.74538208, 0.74671475, 0.749247  , 0.7476092 ]),
 'fit_time': (1.5102295875549316,
  1.536095142364502,
  1.530122995376587,
  1.625065565109253,
  1.852935552597046),
 'test_time': (2.8603804111480713,
  2.8806378841400146,
  2.887338876724243,
  3.096221446990967,
  3.5909392833709717)}
```

Image 11. RMSE and MAE scores for user-user filtering.

4.5. Evaluation of content based and collaborative filtering.

In the previous sections we tested two different methods of filtering to design recommendation systems. Those methods were content based filtering and collaborative based filtering, which we evaluated from 2 different perspectives. Those perspectives were item based collaborative filtering and user based collaborative filtering. On table 1 are depicted the pros and cons of the 2 methods.

Methods	Content Based filtering	Collaborative filtering	
Approaches		Item based	User based
Pros	No cold start	Works for any kind of item	Easy to implement
	Recommendation of new movies	Less resource consuming than user based	Context independent
	Ability to recommend to users with unpopular tastes	Statistically more precise than content based filtering and user based filtering	Statistically more precise than content based filtering
	Variety of representations	Less dynamic model	The model can help users discover new interests
	System transparency		
	On our system needs less resources		
Cons	Over-specialization	Sparsity issue	Sparsity issue
	Statistically less precise than other methods such us collaborative	Tends to recommend obvious items	Cold start issue
		Not very diverse recommendations	Scalability
			New item issue

Table 1. Pros and cons of recommendation approaches

The pros and cons table depicts the theoretical pros and cons of the mentioned approaches. On our analysis however there are some observations regarding these pros and cons, that should be mentioned.

1. Although statistically item based filtering is more precise than user based, on our analysis it is marginally worse. The main reason is the fact that users are significantly less than total movies (1000 users for 23518 movies) thus nullifying the core advantage of this approach. In a real dynamic environment (e.g. online retail store, streaming platform), users are usually more by a large amount compared to the items. This makes the item-item filtering more effective and less resource and time consuming than the

user-user filtering. In this situation the 2 approaches have fairly similar results with a small difference of 0.001 in RMSE score which is in the range of std margins. In terms of time responds the approaches also have fairly close results.

Approach	RMSE SCORE	STD	MAE	STD	Difference RMSE	Difference MAE
Item-item	0,9733	0,0035	0,7483	0,0023		
User-user	0,9723	0,0034	0,7474	0,0013	-0,001	-0,0009

Table 2. Comparison between i item and user based filtering.

Approach	Avg Fit time	Avg Test time	Dif Fit time	Dif Test time	Dif Total time
Item-item	1,664404345	2,827637005			
User-user	1,610889769	3,06310358	-0,053514576	0,235466576	0,181952

Table 3. Fit and test times of item and user based filtering.

2. Content based filtering algorithm needed less resources to run compared to collaborative based filtering. Content based algorithm recommended items while running on the full 25million dataset. In contrast collaborative based algorithms in order to run on the full dataset demanded more than our available processing power.
3. We consider important disadvantages of collaborative filtering the cold start issue and the sparsity problem.

4.6. Hybrid model

Based on the results that were presented on section 4.5 we decided to design a hybrid recommendation system which combines different approaches. Our goal is to overcome or mitigate some of the previously mentioned issues. We could relate that the most common hybrid approaches that have been a subject of research so far are the following:

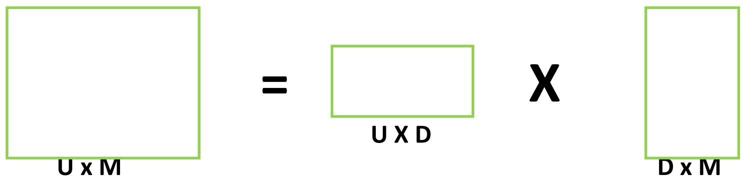
- Combination of separate recommenders. Essentially the recommendation system is built with the implementation of separate content based algorithms and collaborative based algorithms. On that basis there are two paths of designing strategy. The first one is using a metric that evaluates which algorithm fits better at that moment of inquiry. The second is using either a linear combination of ratings or a voting scheme to combine the outputs of the separate recommenders into one final one.

- The addition of content based characteristics to collaborative models. Essentially these systems use traditional collaborative techniques while keeping some content profiles for each user [19]. Then the similarity is calculated for these profiles instead of the commonly rated items. This approach dampens in some extend the sparsity issues that commonly met in collaborative methods.
- The addition of collaborative characteristics to content based models. This hybrid approach is used more frequently and is implementing dimensionality reduction methods to content based models. Essentially it creates with dimension reduction a collaborative view of content based user profiles, thus significantly improving the accuracy of a solely content based system.

On our analysis we decided to proceed with a hybrid system that makes use of matrix factorization based algorithms to dimensionally reduce our initially content based system. Matrix factorization based algorithms belong to the broader range of Model-based Collaborative filtering algorithms which provide item recommendation by first developing a model of user ratings. Matrix factorization is widely used because it can deal better with scalability and sparsity than Memory-based Collaborative Filtering. The purpose of these approaches is to learn the latent preferences of users and the latent attributes of items from known ratings to then predict the unknown ratings. The unknown prediction is usually accomplished by calculating the dot product of the latent features of users and items. To summarize, MF learn features that describe the characteristics of ratings and try to predict unknown ratings. For our analysis we evaluated three different matrix factorization techniques which were Non-Negative Matrix Factorization, Singular Value Decomposition and modified Singular Value Decomposition known as SVD++.

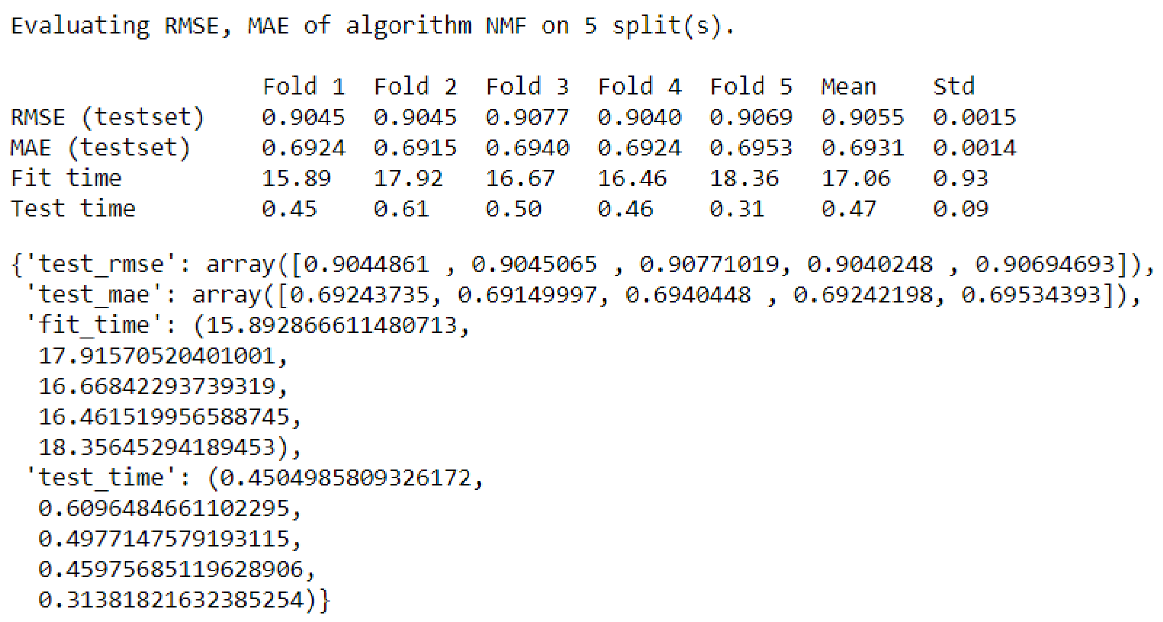
4.6.1. NMF: Non-Negative Matrix Factorization

This group of algorithms took the name of Non-Negative Matrix Factorization commonly known as NMF from Lee and Seung [20] because it contains only positive values. In our analysis ratings are never negative and NMF rates the non-rated as 0. NMF uses only the observed or rated items and creates two individual matrixes and weights to apply to those matrixes to recreate the original data. So for U= Users, M= Movies and D=Dimension of feature vector:



Its main advantage is its ability to automatically extract sparse and easily interpretable factors.

On our analysis we evaluated using cross validation the NMF algorithm and had a RMSE score of 0.9055 +- 0.0015 and a MAE score of 0.6931 +- 0.0014.



```
Evaluating RMSE, MAE of algorithm NMF on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.9045  0.9045  0.9077  0.9040  0.9069  0.9055  0.0015
MAE (testset)     0.6924  0.6915  0.6940  0.6924  0.6953  0.6931  0.0014
Fit time          15.89   17.92   16.67   16.46   18.36   17.06   0.93
Test time         0.45    0.61    0.50    0.46    0.31    0.47    0.09

{'test_rmse': array([0.9044861 , 0.9045065 , 0.90771019, 0.9040248 , 0.90694693]),
 'test_mae': array([0.69243735, 0.69149997, 0.6940448 , 0.69242198, 0.69534393]),
 'fit_time': (15.892866611480713,
  17.91570520401001,
  16.66842293739319,
  16.461519956588745,
  18.35645294189453),
 'test_time': (0.4504985809326172,
  0.6096484661102295,
  0.4977147579193115,
  0.45975685119628906,
  0.31381821632385254)}
```

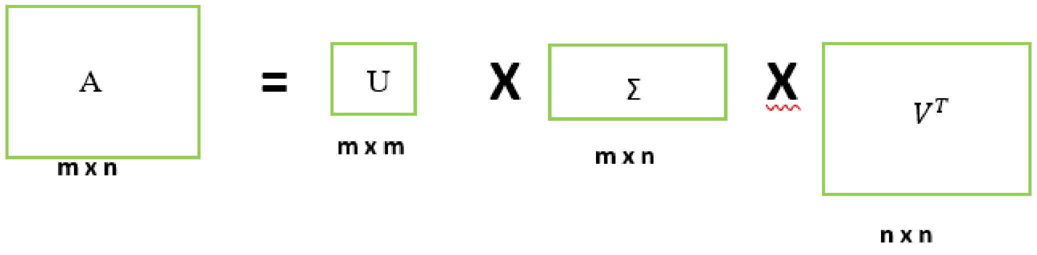
Image 12. RMSE and MAE scores of NMF

4.6.2. SVD: Singular Value Decomposition

Singular Value Decomposition commonly known as SVD is one of the most popular dimensionality reduction techniques in machine learning. It reduces the number of features of a dataset by reducing the space dimension from N-dimension to M dimension (N>M). In the sector of recommendation systems SVD is a matrix factorization collaborative technique. Given a matrix with rows as users, columns as items (each row a user and each column an item) and ratings as elements we use SVD to factorize it to 3 new matrices.

$$A = U\Sigma V^T$$

A is user's ratings matrix.
U is left singular Orthogonal vector (user feature vector)
Σ is the diagonal matrix of singular values (essentially weights/strength of each concept)
V^T is the right singular vectors (movie features matrix)
n>m



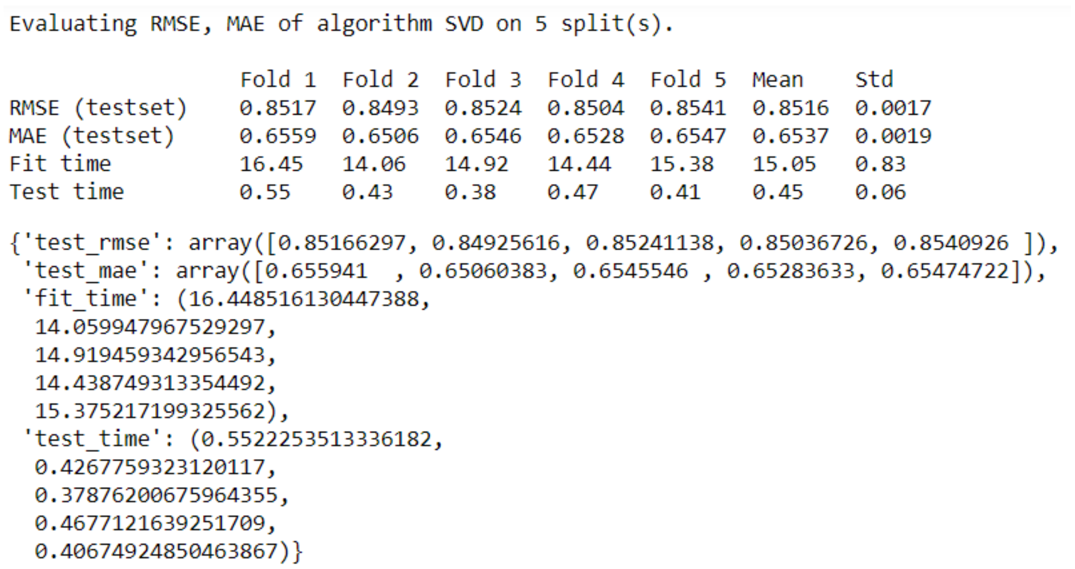
The decomposition and factorization are accomplished by extracting the latent factors. So, a mapping occurs which maps every user and item in a r-dimensional space and show the relationships between users and items. Essentially the input matrix is factorized in two low rank ones with the one representing users and the other items. The missing ratings are then predicted from the inner product of these two factor matrices [21].

The formula to calculate the predicting rating r_{ui} is:

$$\tilde{r}_{ui} = \mu + b_u + b_i + q_i^T \cdot p_u,$$

where μ is the overall average rating, p is user factor, q the item factor, b_u and b_i indicate the bias of user and item, respectively. The bias contributes a major role in the formula since it reflects the rating more accurately and objectively.

Using cross validation, the SVD approach had a RMSE score of 0.8516 +- 0.0017 and a MAE score of 0.6537 +- 0.0019.



```
Evaluating RMSE, MAE of algorithm SVD on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8517  0.8493  0.8524  0.8504  0.8541  0.8516  0.0017
MAE (testset)     0.6559  0.6506  0.6546  0.6528  0.6547  0.6537  0.0019
Fit time          16.45   14.06   14.92   14.44   15.38   15.05   0.83
Test time         0.55    0.43    0.38    0.47    0.41    0.45    0.06

{'test_rmse': array([0.85166297, 0.84925616, 0.85241138, 0.85036726, 0.8540926 ]),
 'test_mae': array([0.655941  , 0.65060383, 0.6545546 , 0.65283633, 0.65474722]),
 'fit_time': (16.448516130447388,
  14.059947967529297,
  14.919459342956543,
  14.438749313354492,
  15.375217199325562),
 'test_time': (0.5522253513336182,
  0.4267759323120117,
  0.37876200675964355,
  0.4677121639251709,
  0.40674924850463867)}
```

Image 13. RMSE and MAE scores of SVD

4.6.3. SVD++ : Modified Singular Value Decomposition

The final approach that we tested was a modified version of the SVD called SVD++. It is based on the theoretical basis of regular SVD with the addition of an important feature. In reality a user will leave some implicit feedback information, such as historical data. These could be rating or browsing data or even the rating operation which in a way reflects the degree of a user's preference for each latent factor. Therefore, the SVD++ model introduces the implicit feedback information based on SVD. To accomplish that it adds factor vector (y_j) for each item in order to describe the characteristics of the item, regardless of whether it has been evaluated. This impacts the user's factor matrix improving the bias. Thus, the predictive rating of the SVD++ model is:

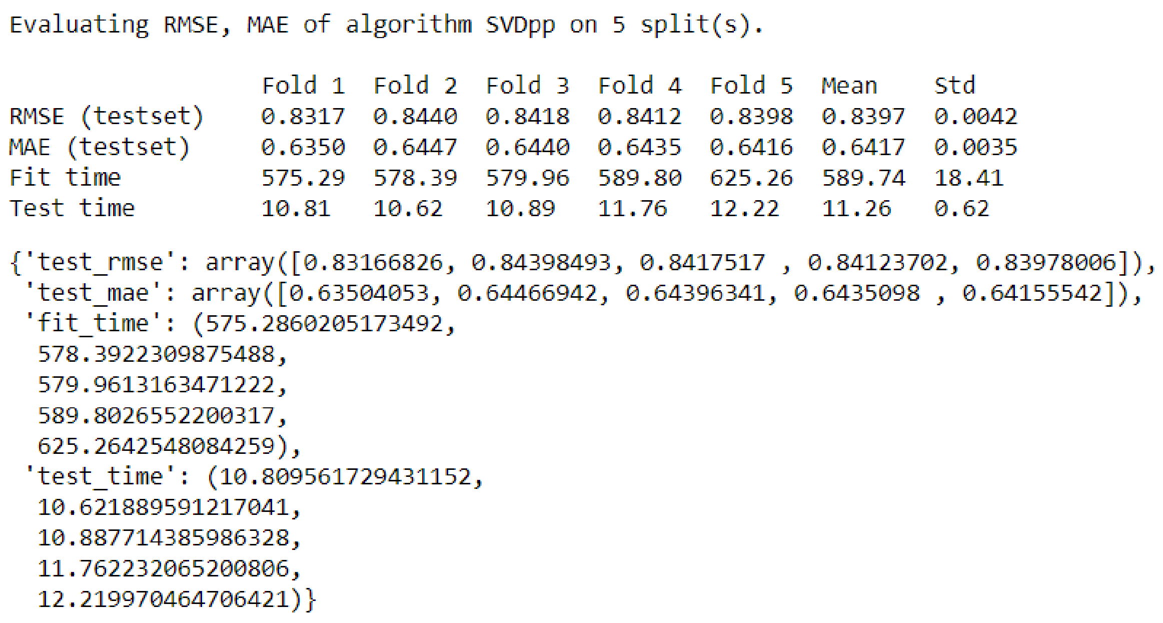
$$\tilde{r}_{ui} = \mu + b_u + b_i + q_i^T \cdot \left(p_u + |R(u)|^{-1/2} \sum_{j \in R(u)} y_j \right)$$

R(u) represents the number of items that user (u) has rated.

For optimal P and Q we could calculate the regularized (for λ being the regularization parameter) squared error with the following formula.

$$\min_{P,Q} \sum_{r_{ui} \in R} \left[r_{ui} - \mu - b_u - b_i - q_i^T \cdot \left(p_u + |R(u)|^{-1/2} \sum_{j \in R(u)} y_j \right)^2 + \lambda \left(b_u^2 + b_i^2 + \|p_u\|^2 + \|q_i\|^2 \right) \right],$$

The bias optimization of SVD++ helps negate the cold start effect.
Using cross validation, the SVD++ approach had a RMSE score of 0.8397 +- 0.0042 and a MAE score of 0.6417 +- 0.0035.



```
Evaluating RMSE, MAE of algorithm SVDpp on 5 split(s).

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8317  0.8440  0.8418  0.8412  0.8398  0.8397  0.0042
MAE (testset)     0.6350  0.6447  0.6440  0.6435  0.6416  0.6417  0.0035
Fit time          575.29  578.39  579.96  589.80  625.26  589.74  18.41
Test time         10.81   10.62   10.89   11.76   12.22   11.26   0.62

{'test_rmse': array([0.83166826, 0.84398493, 0.8417517 , 0.84123702, 0.83978006]),
 'test_mae': array([0.63504053, 0.64466942, 0.64396341, 0.6435098 , 0.64155542]),
 'fit_time': (575.2860205173492,
  578.3922309875488,
  579.9613163471222,
  589.8026552200317,
  625.2642548084259),
 'test_time': (10.809561729431152,
  10.621889591217041,
  10.887714385986328,
  11.762232065200806,
  12.219970464706421)}
```

Image 14. RMSE and MAE scores of SVD++

4.7. Evaluation of the different approaches

On our report we implement five different approaches of recommenders to find the most accurate for our recommendation system. Our main metric of evaluation is the RMSE score while taking in consideration the MAE score and their fit and test time. The five approaches for evaluation are:

- Item-based collaborative filtering.
- User-based collaborative filtering.
- Content-based + matrix factorization with NMF.
- Content-based + matrix factorization with SVD.
- Content-based + matrix factorization with SVD++.

The most accurate is the hybrid recommendations system that combines content-based filtering and dimension reduction with SVD++. Its RMSE score is 0.8397 +-0.0042 which is 0,0119 (1,40%) better than the classic SVD. The results are very encouraging in accordance with the global standards so far. On the following table the approaches are placed sorted in descending order and presenting RMSE and MAE difference.

A/A	Approaches	RMSE SCORE	STD	MAE	STD	Difference RMSE	Difference MAE
1	Item-item	0,9733	0,0035	0,7483	0,0023		
2	User-user	0,9723	0,0034	0,7474	0,0013	-0,001	-0,0009
3	Con+NMF	0,9055	0,0015	0,6931	0,0014	-0,0668	-0,0543
4	Con+SVD	0,8516	0,0017	0,6537	0,0019	-0,0539	-0,0394
5	con+SVD++	0,8397	0,0042	0,6417	0,0035	-0,0119	-0,012

Table 6. RMSE SCORES AND MAE COMPARISONS

It is worth mentioning that there is a significant improvement of 6,87% (0,0668) between the user-based filtering and the closest hybrid recommender which is the combination of content and NMF. The difference of 13,64 % between the SVD++ approach and the user-based filtering highlights even further the correct decision of proceeding with a hybrid system.

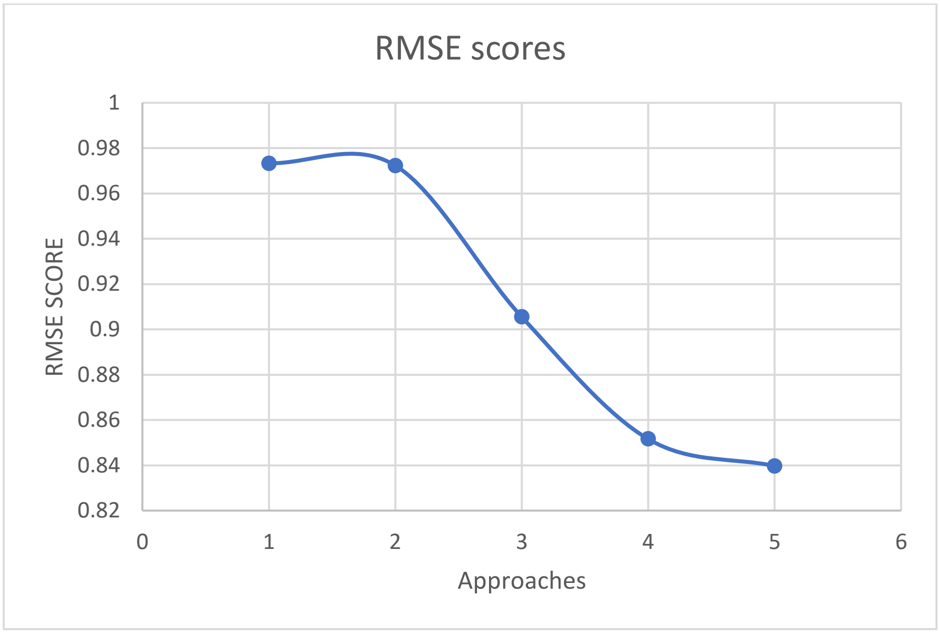


Diagram 7. RMSE-approaches

Time wise the results are showing that the approach of content-based filtering and SVD++ is way more consuming that the rest. One table 7 we could see that SVD++ is almost 34 times slower at fitting into the model compared to classic SVD. The reason is the computation of the new factor vector for each item that SVD++ does. We notice that for such a huge difference of fitting and testing time the RMSE score is not showing huge improvement (only 1,4%).

A/A	Approaches	Avg Fit time	Avg Test time	Dif Fit time	Dif Test time	Dif Total time
1	Item-item	1,664404345	2,827637005			
2	User-user	1,610889769	3,06310358	-0,053514576	0,235466576	0,181952
3	Con+NMF	17,05899353	0,466287374	15,44810376	-2,596816206	12,85128756
4	Con+SVD	15,04837799	0,446444941	-2,01061554	-0,019842434	-2,030457973
5	con+SVD++	589,7412956	11,26027365	574,6929176	10,81382871	585,5067463

Table 7. Time distribution of approached methods

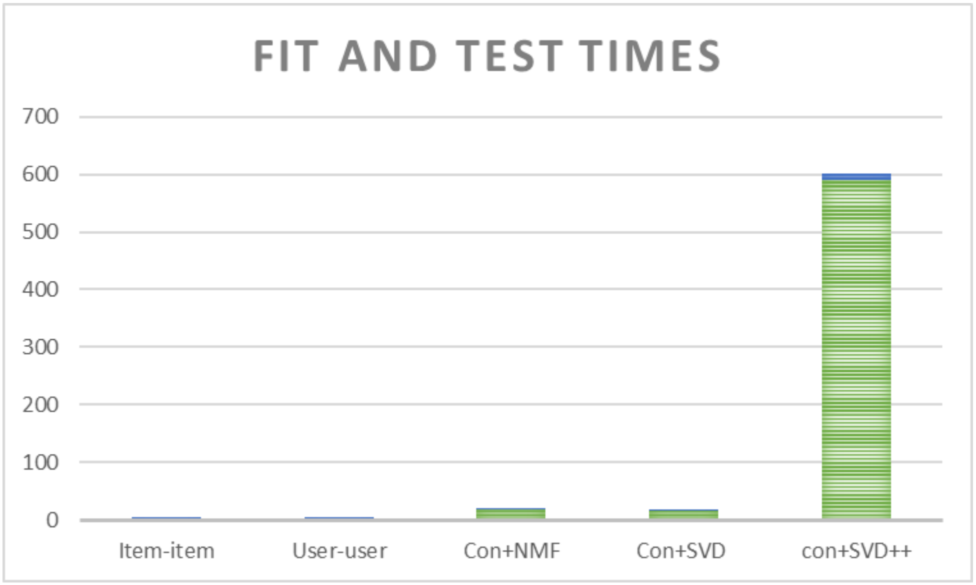


Diagram 8. Fit and test times.

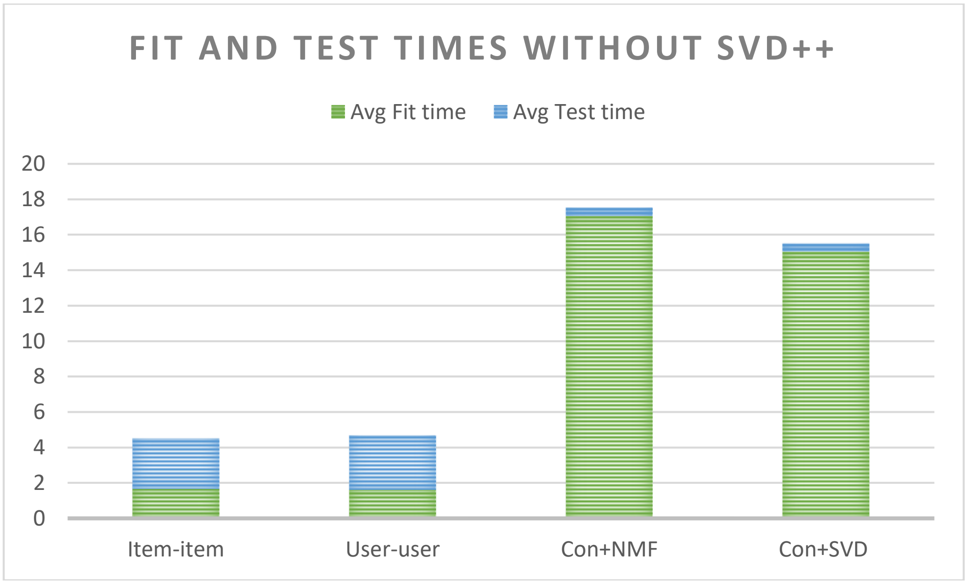


Diagram 8. Fit and test times without SVD++

From the diagram 8 we notice that hybrid models need about 10 times more fitting time than the simple collaborative methods. Also, we could conclude that hybrid models need more time to fit than for testing in contrast to the simple collaborative methods who take around the same time for fitting and testing.

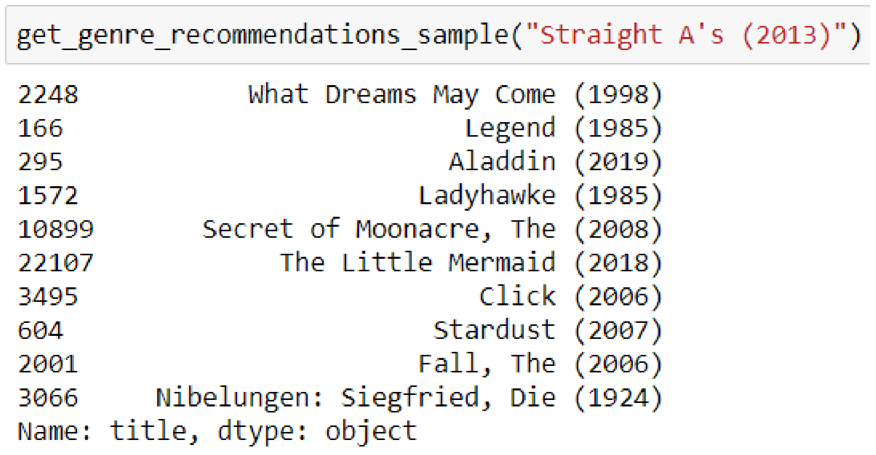
4.8. Final hybrid system

The evaluation process showed that the hybrid approach that combines content-based filtering and factorization with SVD++ is the most accurate. So, we designed the recommendation system based on this approach. The steps were the following:

1. Loading the dataset Movielens 25M.
2. Create the sample dataset of 1000 users and 23518 movies.
3. Generate the utility matrix.
4. Use Content-based filtering on the utility matrix picking movies genre as a feature.
5. Use SVD++ to achieve model based collaborative filtering.
6. Create the hybrid recommendation system.
7. Provide the top recommendations to the user.

The steps 1 to 4 have already been presented on sections 4.1 and 4.2 showing the results of taking genre as a feature for content based filtering.

Specifically, for user with id 548 we could find the top 10 similar movies based on a movie he had already rated.

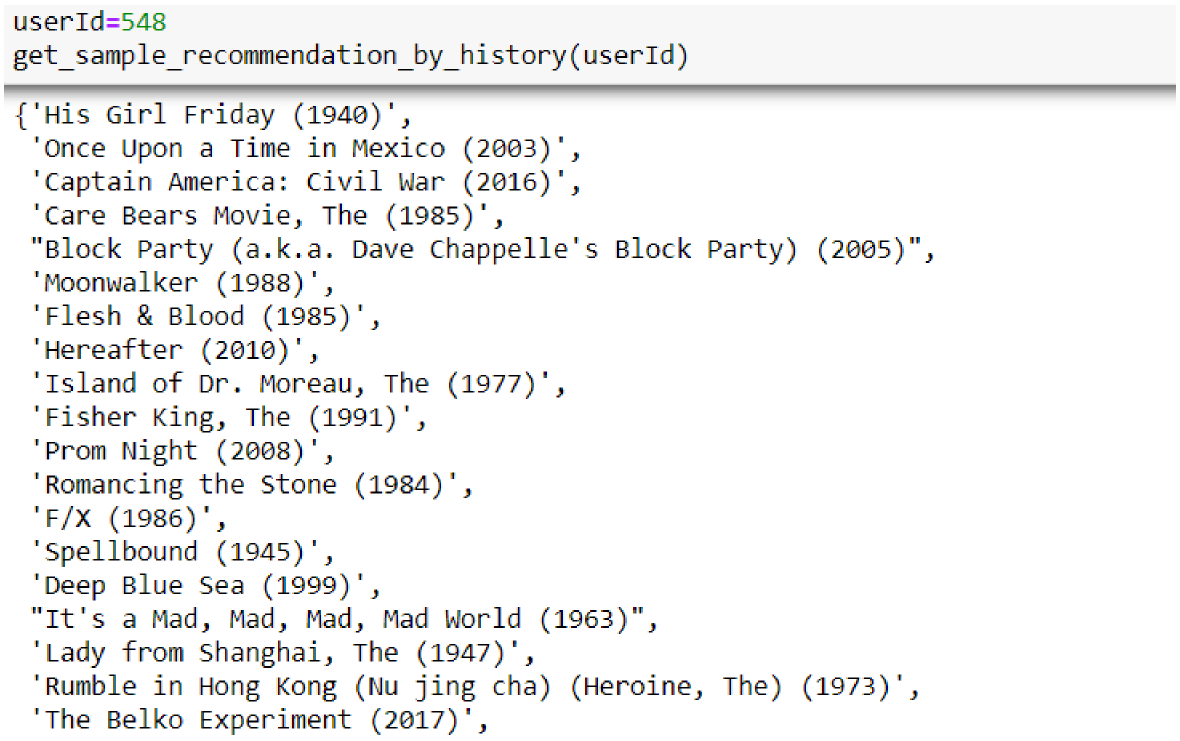


```
get_genre_recommendations_sample("Straight A's (2013)")
```

```
2248            What Dreams May Come (1998)
166                           Legend (1985)
295                          Aladdin (2019)
1572                      Ladyhawke (1985)
10899        Secret of Moonacre, The (2008)
22107             The Little Mermaid (2018)
3495                           Click (2006)
604                         Stardust (2007)
2001                       Fall, The (2006)
3066      Nibelungen: Siegfried, Die (1924)
Name: title, dtype: object
```

Image 15. List of recommended movies most similar to the one he had already rated.

And a full list of similar movies he has not yet watched based on all the movies he had already rated.

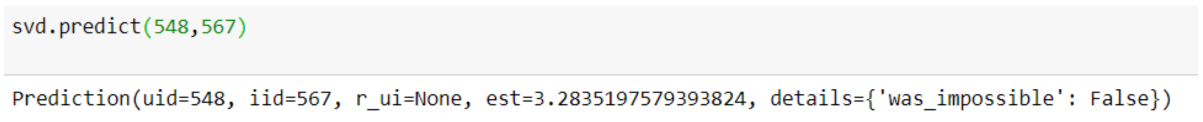


```
userId=548
get_sample_recommendation_by_history(userId)
```

```
{'His Girl Friday (1940)',
 'Once Upon a Time in Mexico (2003)',
 'Captain America: Civil War (2016)',
 'Care Bears Movie, The (1985)',
 "Block Party (a.k.a. Dave Chappelle's Block Party) (2005)",
 'Moonwalker (1988)',
 'Flesh & Blood (1985)',
 'Hereafter (2010)',
 'Island of Dr. Moreau, The (1977)',
 'Fisher King, The (1991)',
 'Prom Night (2008)',
 'Romancing the Stone (1984)',
 'F/X (1986)',
 'Spellbound (1945)',
 'Deep Blue Sea (1999)',
 "It's a Mad, Mad, Mad, Mad World (1963)",
 'Lady from Shanghai, The (1947)',
 'Rumble in Hong Kong (Nu jing cha) (Heroine, The) (1973)',
 'The Belko Experiment (2017)',
```

Image 16. List of recommended movies for user with Id 548 based on content based filtering.

After implementing the SVD++ we could predict for every unrated movie a rating for every user on our system. On image 17 we check for our usual user with id 548 for the movie with id 567 the predicted rating is 3.284

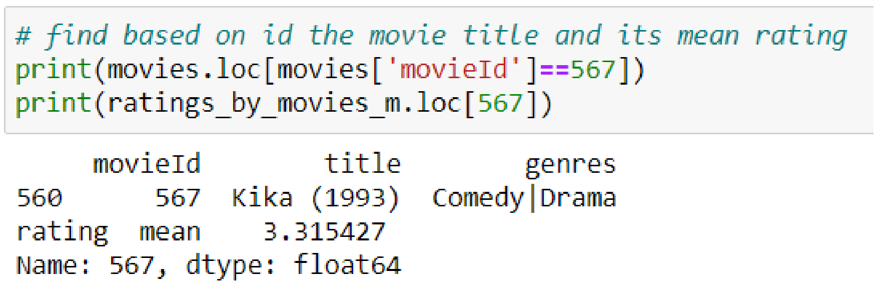


```
svd.predict(548,567)
```

```
Prediction(uid=548, iid=567, r_ui=None, est=3.2835197579393824, details={'was_impossible': False})
```

Image 17. Predicting movie rating based on SVD++.

The movie with id 567 has an actual mean rating of 3.315 which shows that the hybrid system prediction had an accuracy of 93,51% on this particular situation.

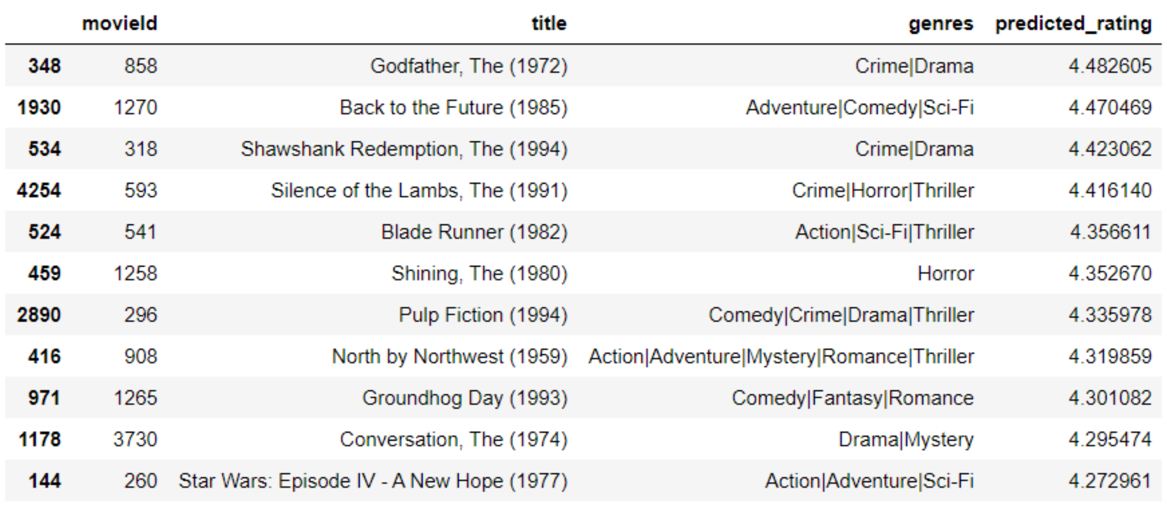


```
# find based on id the movie title and its mean rating
print(movies.loc[movies['movieId']==567])
print(ratings_by_movies_m.loc[567])
```

```
     movieId         title         genres
560      567  Kika (1993)  Comedy|Drama
rating  mean     3.315427
Name: 567, dtype: float64
```

Image 18. Finding mean rating by searching with movie id.

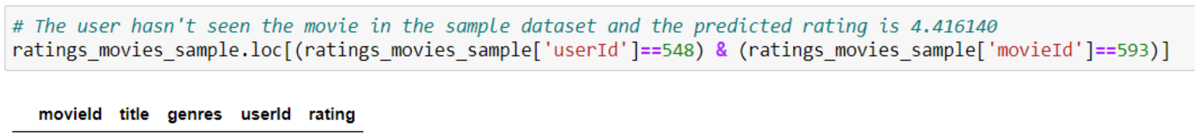
On the final step we want the system to recommend the top predicted movies for each user. On our example user with id 548 the system recommended the following movies:



	movieId	title	genres	predicted_rating
348	858	Godfather, The (1972)	Crime|Drama	4.482605
1930	1270	Back to the Future (1985)	Adventure|Comedy|Sci-Fi	4.470469
534	318	Shawshank Redemption, The (1994)	Crime|Drama	4.423062
4254	593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller	4.416140
524	541	Blade Runner (1982)	Action|Sci-Fi|Thriller	4.356611
459	1258	Shining, The (1980)	Horror	4.352670
2890	296	Pulp Fiction (1994)	Comedy|Crime|Drama|Thriller	4.335978
416	908	North by Northwest (1959)	Action|Adventure|Mystery|Romance|Thriller	4.319859
971	1265	Groundhog Day (1993)	Comedy|Fantasy|Romance	4.301082
1178	3730	Conversation, The (1974)	Drama|Mystery	4.295474
144	260	Star Wars: Episode IV - A New Hope (1977)	Action|Adventure|Sci-Fi	4.272961

Image 19. Top recommendations for user with id 548.

A very interesting observation that needs to be mentioned is that the movie with id 593 and title 'Silence of the Lambs (1991)' happened to actually be rated by user with id 548 on the original 25 million dataset. Of course, in the sample dataset that we created does not exist. Image 20 shows that on the recommendation system this specific rating does not exist.

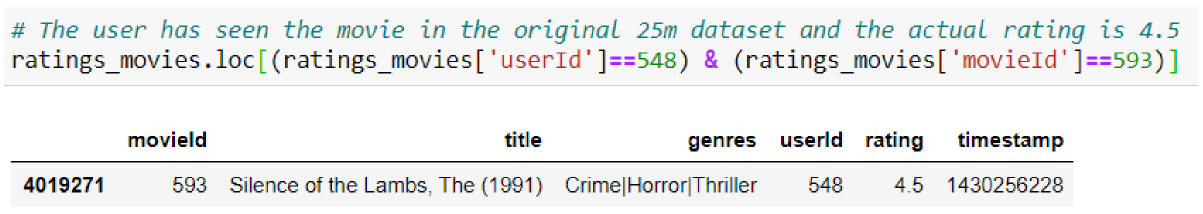


```
# The user hasn't seen the movie in the sample dataset and the predicted rating is 4.416140
ratings_movies_sample.loc[(ratings_movies_sample['userId']==548) & (ratings_movies_sample['movieId']==593)]
```

movieId	title	genres	userId	rating

Image 20. Check if movie with id 593 has been rated by user with id 548

On image 21 we observed that the movie has been rated by user with id 548 on the full dataset.



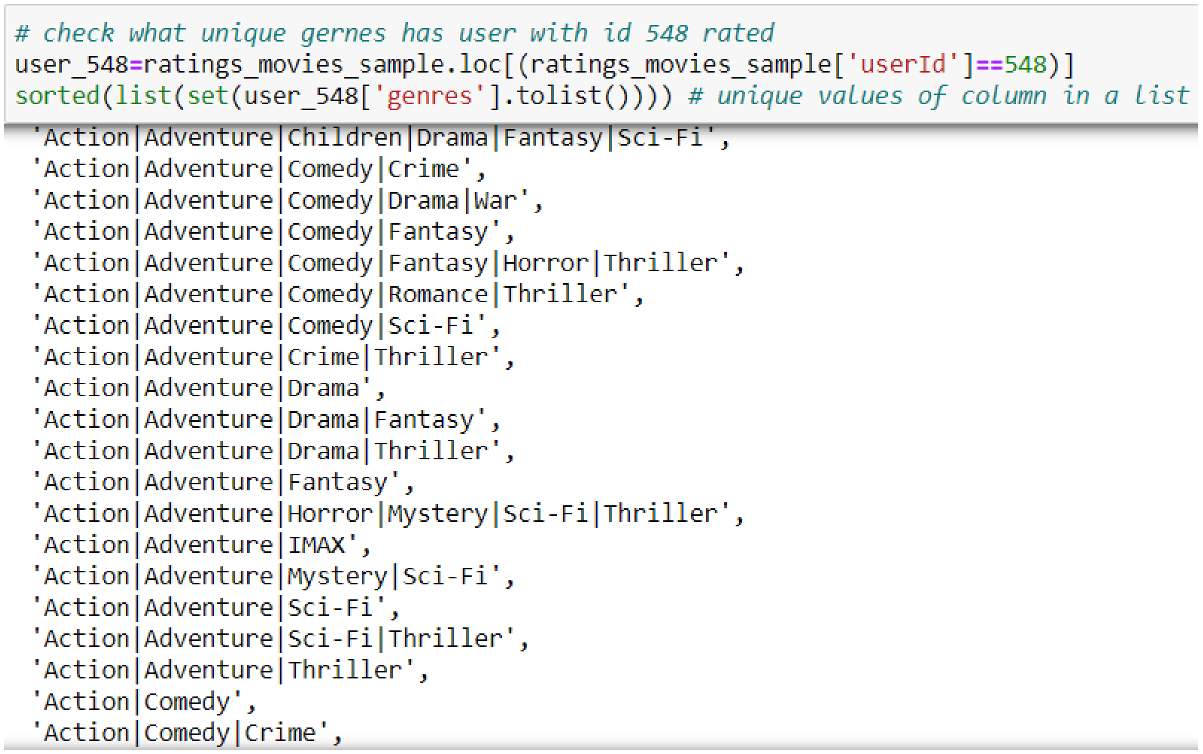
```
# The user has seen the movie in the original 25m dataset and the actual rating is 4.5
ratings_movies.loc[(ratings_movies['userId']==548) & (ratings_movies['movieId']==593)]
```

	movieId	title	genres	userId	rating	timestamp
4019271	593	Silence of the Lambs, The (1991)	Crime|Horror|Thriller	548	4.5	1430256228

Image 21. Check if movie with id 593 has been rated by user with id 548 on the full dataset.

Again, we noticed how accurate the recommendation system is, predicting a rating of 4.416 while the actual rating is 4.5. There is an accuracy of 98,13%.

On terms of diversity the system achieved very good results since it recommended movies with different genres and similar to the ones the user has already rated (image 22).



```
# check what unique gernes has user with id 548 rated
user_548=ratings_movies_sample.loc[(ratings_movies_sample['userId']==548)]
sorted(list(set(user_548['genres'].tolist()))) # unique values of column in a list
```

```
 'Action|Adventure|Children|Drama|Fantasy|Sci-Fi',
 'Action|Adventure|Comedy|Crime',
 'Action|Adventure|Comedy|Drama|War',
 'Action|Adventure|Comedy|Fantasy',
 'Action|Adventure|Comedy|Fantasy|Horror|Thriller',
 'Action|Adventure|Comedy|Romance|Thriller',
 'Action|Adventure|Comedy|Sci-Fi',
 'Action|Adventure|Crime|Thriller',
 'Action|Adventure|Drama',
 'Action|Adventure|Drama|Fantasy',
 'Action|Adventure|Drama|Thriller',
 'Action|Adventure|Fantasy',
 'Action|Adventure|Horror|Mystery|Sci-Fi|Thriller',
 'Action|Adventure|IMAX',
 'Action|Adventure|Mystery|Sci-Fi',
 'Action|Adventure|Sci-Fi',
 'Action|Adventure|Sci-Fi|Thriller',
 'Action|Adventure|Thriller',
 'Action|Comedy',
 'Action|Comedy|Crime',
```

Image 21. Check genres of movies that user with id 548 has rated.

The recommended movies are highly rated in terms of mean rating on the dataset which verifies the validity of the recommendation approach and the design of the hybrid system.

5. Conclusions

In this study we set an objective to design a recommendation system that will be able to efficiently propose movies that a user will enjoy. By evaluating a series of algorithms and techniques we decided to design a hybrid recommendation system that combines a content based-filtering using genres as a feature and a matrix factorization technique by decomposing the user-item matrix with SVD++. The RMSE score was 0,8397 which is very encouraging. Based on the final recommendations the factorized matrix seemed to identify the underlying tastes and preferences of the user. Given the fact that there is a dimensionality reduction technique implemented on the model there is a good scalability of the overall system. In conclusion the objective was met with good results and the predicted ratings of the recommended movies were highly accurate.

6. References

1. *Statista site: https://www.statista.com/statistics/187122/movie-releases-in-north-america-since-2001/*
2. *Netflix site: https://www.netflix.com/gr/*
3. *Netflix Prize Wikipedia site: https://en.wikipedia.org/wiki/Netflix_Prize*
4. *Yehuda Koren. The BellKor Solution to the Netflix Grand Prize.*
5. *Paperswithcode movielens competition: https://paperswithcode.com/sota/collaborative-filtering-on-movielens-1m*
6. *MovieLens 1M Dataset: https://grouplens.org/datasets/movielens/1m/*
7. *Steffen Rendle,Li Zhang,Yehuda Koren. On the Difficulty of Evaluating Baselines: A Study on Recommender Systems.*
8. *Yin Zheng, Bangsheng Tang, Wenkui Ding, Hanning Zhou. A Neural Autoregressive Approach to Collaborative Filtering.*
9. *Rianne van den Berg, Thomas N. Kipf, Max Welling. Graph Convolutional Matrix Completion.*
10. *Florian Strub, Jeremie Mary, Romaric Gaudel. Hybrid Recommender System based on Autoencoders.*
11. *MovieLens 25M Dataset: https://grouplens.org/datasets/movielens/25m/*
12. *Pandas library: https://pandas.pydata.org/*
13. *Beginners Guide to learn about Content Based Recommender Engines: https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/*
14. *Scikit-learn site: https://scikit-learn.org/stable/*
15. *Agnes Johannsdottir. Implementing your Own Recommender Systems in Python.*
16. *Surprise library. http://surpriselib.com/*
17. *D. Parra and S. Sahebi, "Recommender systems: Sources of knowledge and evaluation metrics", Springer,Berlin, Heidelberg, 149-175, 2013. DOI: http://dx.doi.org/10.1007/978-3-642-33326-2 7*
18. *Blerina Lika, Kostas Kolomvatsos, Stathes Hadjiefthymiades. Facing the cold start problem in recommender systems.*
19. *Aanchal Singh, Abinash Mohapatra, Anisha Pradhan. A HYBRID MOVIES RECOMMENDATION SYSTEM.*
20. *Non-negative_matrix_factorization Wikipedia site. https://en.wikipedia.org/wiki/Non-negative_matrix_factorization*
21. *Zhengzheng Xian, Qiliang Li, Gai Li, Lei Li. New Collaborative Filtering Algorithms Based on SVD++ and Differential Privacy*