Movie Recommender System: using Machine Learning

Tanzina Tazreen Meem, Rifat Hossain

Department of Computer Science & Engineering

North South University

Dhaka, Bangladesh, 2021.

Abstract:

This paper discusses movie recommendations and the kinds of recommendation system there is. Recommender systems are one of the most successful and widespread applications of machine learning [5]. technologies in business. A movie recommendation is important in our social life due to its strength in providing enhanced entertainment. Such a system can suggest a set of movies to users based on their interest, previous watching history, previous likes, or the popularities and genre of the movies. A recommendation system is used to suggest items to purchase or to see. They direct users towards those items which can meet their needs by cutting down a large database of information to help users to find the movies of their choices based on the movie experience of other users efficiently and effectively without wasting much time in useless browsing [7]. Here we used MovieLens Dataset for our recommendation system.

Keywords: *Filtering, Recommendation System, Recommender, collaborative filtering, content-based filtering, EDA, dataset, machine learning, model, vector, matrix, Movies, metadata, Ratings, Similarity matrix, MovieLens.*

Introduction:

Recommended systems are one of the most important techniques used to introduce information about user needs, including related services, by analyzing user actions [1, 2]. A recommendation algorithm is a type of machine learning algorithm that is very closely related to real life. It refers to a type of algorithm that does not require users to provide clear needs but models users' interests by analyzing their historical behaviors, to actively recommend products to users that can meet their interests and needs [4]. There are a wide variety of applications for recommendation systems. These have become increasingly popular over the last few years and are now utilized in most online platforms that we use. The content of such platforms varies from movies, music, books, and videos, to friends and stories on social media platforms, to products on e-commerce websites, to people on professional and dating websites, to search results returned on

Google. For example, Facebook can monitor your interaction with various stories on your feed to learn what types of stories appeal to you [5] and to recommend pages to like and people to follow. Amazon uses it to suggest products to customers, YouTube uses it to decide which video to play next on autoplay. Netflix, Hulu, and Hotstar's success revolve around the potency of their recommendations. Netflix even offered a million dollars in 2009 to anyone who could improve its system by 10%. [6]. Recommender systems were first mentioned in a technical report as a "digital bookshelf" in 1990 by Jussi Karlgren at Columbia University, [8] and implemented at scale and worked through in technical reports and publications from 1994 onwards by Jussi Karlgren, then at SICS,[9][10] and research groups led by Pattie Maes at MIT,[11] Will Hill at Bellcore,[12] and Paul Resnick, also at MIT[13][14] whose work with Group Lens was awarded the 2010 ACM Software Systems Award.

Basic Recommender systems can be classified into 3 types:

- **Simple recommenders:** offer generalized recommendations to every user, based on movie popularity and/or genre. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience [6]. They will show a list that is related to one movie by genre and has the most popularity (ratings).
- **Content-based recommenders:** Another common approach when designing recommender systems is content-based filtering [5]. It suggests similar items based on a particular item. This system uses item metadata, such as genre, director, description, actors, etc. for movies, to make these recommendations. The general idea behind these recommender systems is that if a person likes a particular item, he or she will also like an item that is similar to it. And to recommend that, it will make use of the user's past item metadata [6]. Content-based recommenders treat recommendation as a user-specific classification problem and learn a classifier for the user's likes and dislike based on an item's features. In this system, keywords are used to describe the items and a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past, or is examining in the present. It does not rely on a user sign-in mechanism to generate this often temporary profile. In particular, various candidate items are compared with items previously rated by the user, and the best-matching items are recommended. [5]. A good example could be YouTube, where based on your history, it suggests new videos that you could potentially watch [6].
- **Collaborative filtering engines:** Collaborative filtering is a

technique for predicting unknown preferences of people by using already known preferences from many users. It computes similarity on two bases: one is user and the other is the item. It uses the cosine and Pearson correlation similarity approach. The main challenges that Collaborative Filtering deals with are data sparsity, scalability, and cold start problem. CF introduces three main algorithms: memory-based, model-based, and hybrid CF, which are used to combine CF with other recommendation techniques and their power to deal with the challenges [5]. these systems are widely used. as they try to predict the rating or preference that a user would give an item based on past ratings and preferences of other users, Collaborative filters do not require item metadata like their content-based counterparts [6].

There are other types of recommendation systems like **Hybrid recommender systems.** Most recommender systems now use a hybrid approach, combining collaborative filtering, content-based filtering, and other approaches. There is no reason why several different
techniques of the same type could not be hybridized. Hybrid approaches can be implemented in several ways: by making content-based and collaborative-based predictions separately and then combining them; by adding content-based capabilities to a collaborative-based approach (and vice versa); or by unifying the approaches into one model. Several studies empirically compare the performance of the hybrid with the pure collaborative and content-based methods and demonstrated that the hybrid methods can provide more accurate recommendations than pure
approaches. These methods can also be used to overcome some of the common problems in recommender systems such as cold start and the sparsity problem, as well as the knowledge
engineering bottleneck in knowledge-based approaches. Netflix uses Hybrid recommender systems. The website makes recommendations by comparing the watching and searching habits of similar users (i.e., collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering) [7].

Performing Exploratory Data Analysis on Data

Item-Based collaborative filtering Recommender

For this, we will take Movie Lens small dataset and focus on two files, i.e., the movies.csv and ratings.csv.
Movies.csv has three fields namely:
1. Movie Id – It has a unique id for every movie
2. Title – It is the name of the movie
3. Genre – The genre of the movie

```
ratings.head()
```

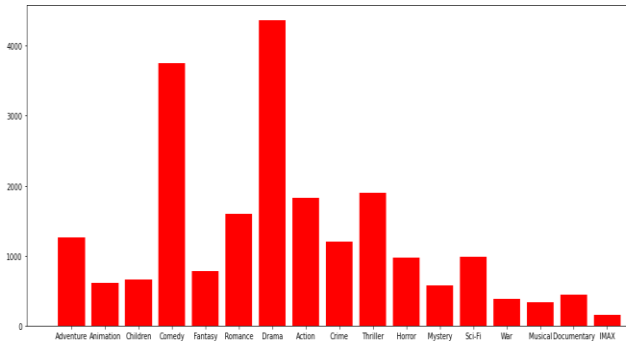
	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

```
movies.head()
```

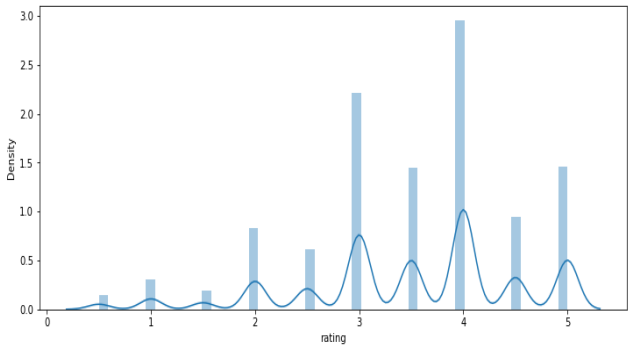
	movieId	title	
0	1	Toy Story (1995)	Adventure|Animation|Children|Co
1	2	Jumanji (1995)	Adventure|Chi
2	3	Grumpier Old Men (1995)	Com
3	4	Waiting to Exhale (1995)	Comedy|Dra
4	5	Father of the Bride Part II (1995)	

```
ratings.describe()
```

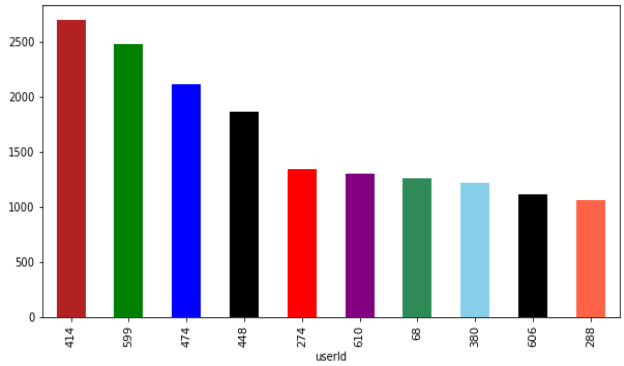
	userId	movieId	rating	ti
count	100836.000000	100836.000000	100836.000000	1.008
mean	326.127564	19435.295718	3.501557	1.205
std	182.618491	35530.987199	1.042529	2.162
min	1.000000	1.000000	0.500000	8.281
25%	177.000000	1199.000000	3.000000	1.019
50%	325.000000	2991.000000	3.500000	1.186
75%	477.000000	8122.000000	4.000000	1.435
max	610.000000	193609.000000	5.000000	1.537



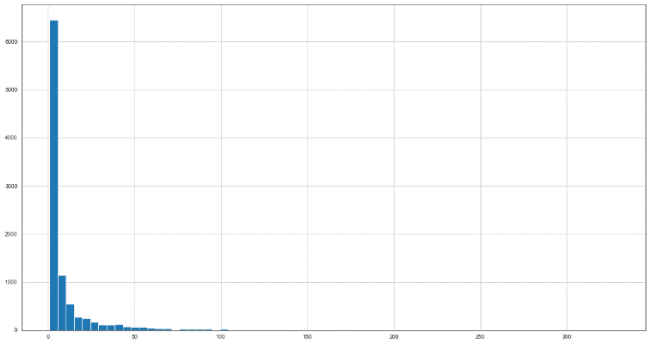
The most popular genres of the movie released



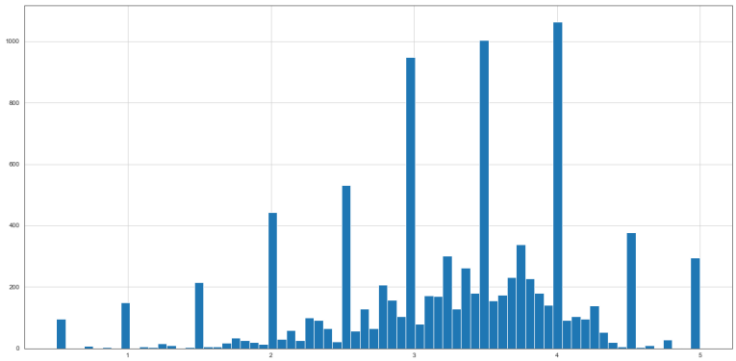
Distribution of users rating



Top 10 users who have rated most of the movies



the graph of the number of ratings



the graph of "rating"

Content-Based Recommender

For this, we will take the Movie Lens dataset and focus on four files, i.e., the movies_metadata.csv, small_links.csv, small_links.csv, credits.csv, keywords.csv.

```
links_small.head()
```

```
0      862
1     8844
2    15602
3    31357
4    11862
Name: tmdbId, dtype: int32
```

```
meta.head()
```

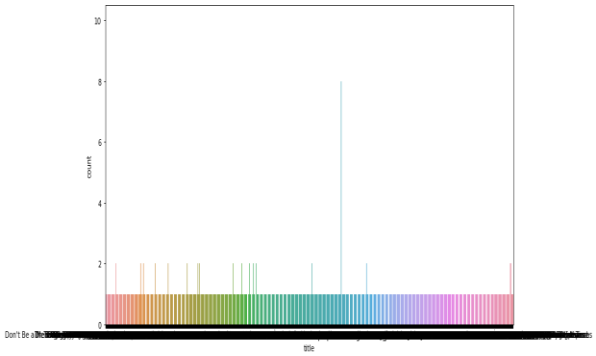
	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[{'id': 16, 'name': 'Animation'}, {'id': 35, '...	http://toystory.disney.com/toy-story	862	tt0114709	
1	False		NaN	65000000	[{'id': 12, 'name': 'Adventure'}, {'id': 14, '...		NaN	8844	tt0113497
		{'id': 119050, 'name':		[{'id': 10749, 'name':				

```
keywords.head()
```

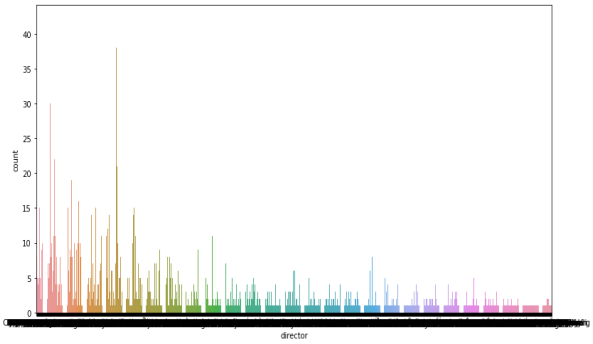
	id	keyv
0	862	[{'id': 931, 'name': 'jealousy'}, {'id': 42
1	8844	[{'id': 10090, 'name': 'board game'}, {'i
2	15602	[{'id': 1495, 'name': 'fishing'}, {'id': 12
3	31357	[{'id': 818, 'name': 'based on novel'},
4	11862	[{'id': 1009, 'name': 'baby'}, {'id': 159

```
credits.head()
```

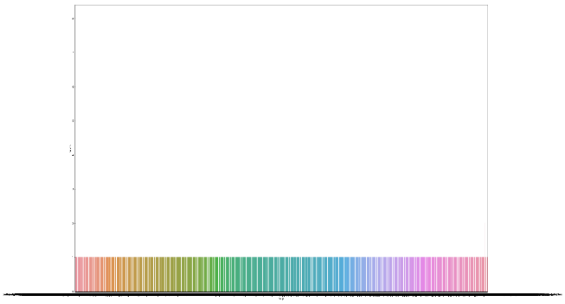
	cast	crew	id
0	[{'cast_id': 14, 'character': 'Woody (voice)',...	[{'credit_id': '52fe4284c3a36847f8024f49', 'de...	862
1	[{'cast_id': 1, 'character': 'Alan Parrish', '...	[{'credit_id': '52fe44bfc3a36847f80a7cd1', 'de...	8844
2	[{'cast_id': 2, 'character': 'Max Goldman', 'c...	[{'credit_id': '52fe466a9251416c75077a89', 'de...	15602
3	[{'cast_id': 1, 'character': "Savannah 'Vannah...	[{'credit_id': '52fe44779251416c91011acb', 'de...	31357
4	[{'cast_id': 1, 'character': 'George Banks', '...	[{'credit_id': '52fe44959251416c75039ed7', 'de...	11862



Barchart for movie “title”



Barchart for movie “Director”



Barchart for the “soup” we will create.

Methodology

Item-Based collaborative filtering Recommender

To implement **item-based collaborative filtering,** KNN is a perfect go-to model and also a very good baseline for recommender system development [16]. So, we used the K-Nearest Neighbor model to get the movie recommending list and for evaluation.

K-Nearest Neighbor: **KNN** is a **non-parametric, lazy** learning method. It uses a database in which the data points are separated into several clusters to make inferences for new samples. KNN does not make any assumptions on the underlying data distribution but relies on **item feature similarity**. When KNN makes an inference about a movie, KNN will calculate the "distance" between the target movie and every other movie in its database, then it ranks its distances and returns the top K nearest neighbor movies as the most similar movie recommendations [16].

First, we need to transform the dataframe of ratings into a proper format that can be consumed by a KNN model. We want the data to be in an m x n array, where m is the number of movies and n is the number of users. To reshape dataframe of ratings, we'll pivot the dataframe to the wide format with movies as rows and users as columns. Then we'll fill the missing observations with 0s since we're going to be performing linear algebra operations (calculating distances between vectors). For more efficient calculation and less memory footprint, we need to transform the values of the dataframe into a **scipy sparse matrix**. our training data has very high dimensionality. KNN's performance will suffer from **curse of dimensionality** if it uses "Euclidean distance" in its objective function. **Euclidean distance** is unhelpful in high dimensions because all vectors are almost equidistant to the search query vector (target movie's features). Instead, we will use **cosine similarity** for the nearest neighbor search. There is also another popular approach to handle nearest neighbor search in high dimensional data. Finally, we can make some movie recommendations for ourselves. [16].

We used the KNN algorithm to compute similarity with cosine distance metric which is very fast and more preferable than the Pearson coefficient. The working principle is very simple. We first check if the movie name input is in the database and if it is we use our recommendation system to find similar movies and sort them based on their similarity distance and output only the top 10 movies with their distances from the input movie. [15].

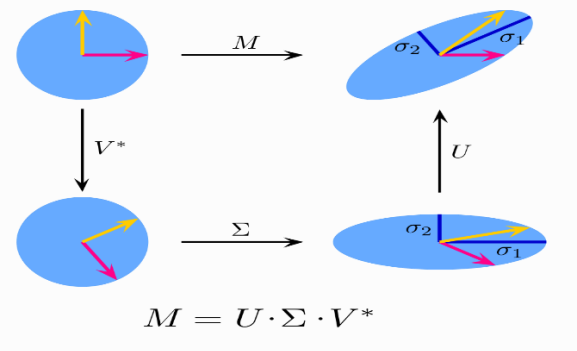
```
get_movie_recommendation('Iron Man')
```

	Title	Distance
1	District 9 (2009)	0.381027
2	Guardians of the Galaxy (2014)	0.379761
3	Kung Fu Panda (2008)	0.375980
4	Batman Begins (2005)	0.372033
5	Watchmen (2009)	0.370870
6	Avatar (2009)	0.326285
7	WALL·E (2008)	0.315061
8	Iron Man 2 (2010)	0.310027
9	Dark Knight, The (2008)	0.308411
10	Avengers, The (2012)	0.294442

Rating prediction

To predict the ratings of the movies, we used the SVD model. And to evaluate (finding RMSE) them we used SVD and SVD++.

SVD: **SVD** is singular value decomposition. We will convert the SVD to k dimensions spaces to make predicted ratings of movies [17].



for the recommendation system, it is a k-rank approximation to the original SVD bellowed:

First you do the SVD: $\underset{n\times m}{X} = \underset{n\times n}{U}\ \overset{n\times m}{\underset{}{\Sigma}}\ \underset{m\times m}{V^T}$, where U and V are rotation matrices, and Σ has the singular values along the diagonal. Then you pick the top k singular values, zero out the rest, and hack off irrelevant rows and columns to make a k-rank approximation to the original: $X \approx \tilde{X} = \underset{n\times k}{\tilde{U}}\ \overset{k\times k}{\underset{}{\tilde{\Sigma}}}\ \underset{k\times m}{\tilde{V}^T}$

Euclidean distance can just be considered as a straight-line distance between two vectors. For two vectors **x** and **y**, we can compute this [17]:

$$EUC(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Though SVD can't predict if there is a NaN value in the matrix. It will make a recommendation for a specific user through the predicted rating [17]. The recommender system works purely based on an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie.

Content-Based Recommender

It builds a system that recommends movies that are similar to a particular movie. The quality of your recommender would be increased with the usage of better metadata and by capturing more of the finer details. we will compute pairwise cosine similarity scores for all movies based on the following metadata: the 3 top actors, the director, related genres, and the movie plot keywords. and recommend movies based on that

similarity score threshold. The keywords, cast, and crew data are not available in our current dataset. so the first step would be to load and merge them into our main DataFrame `metadata` [6].

```
meta.head()
```

	adult	belongs_to_collection	budget	genres	homepage	id	imdb_id	original_lang
0	False	{'id': 10194, 'name': 'Toy Story Collection', ...	30000000	[Animation, Comedy, Family]	http://toystory.disney.com/toy-story	862	tt0114709	
1	False		NaN	65000000	[Adventure, Fantasy, Family]	NaN	8844	tt0113497
2	False	{'id': 119050, 'name': 'Grumpy Old Men Collect...	0	[Romance, Comedy]		NaN	15602	tt0113228

The problem at hand is a Natural Language Processing problem. So, we need to extract some kind of features from the various kinds of features to compute the similarity and/or dissimilarity between them. From our new features, cast, crew, and keywords, we need to extract the three most important actors, the director, and the keywords associated with that movie. After converting the list into a string, we will have to remove the space between words. now we can create a "metadata soup", which is a string that contains all the metadata that we want to feed to the vectorizer (namely actors, director and keywords) [6].

vectors are vectorized representations of words in a document. The vectors carry a semantic meaning with them. For example, man & king will have vector representations close to each other while man & woman would have representation far from each other. we will compute `CountVectorizer()` [6].

Fortunately, scikit-learn gives us a built-in `CountVectorizer().` now we can compute a similarity score. There are several similarity metrics that we can use for this, such as the Manhattan, Euclidean, Pearson, and the cosine similarity scores. we will be using the `cosine similarity` to measure the distance between the embeddings. we use the cosine similarity score since it is independent of magnitude and is relatively easy and fast to calculate [6]. Mathematically, it is defined as follows:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}^\intercal}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} = \frac{\sum_{i=1}^{n} \mathbf{x}_i \cdot \mathbf{y}^\intercal{}_i}{\sqrt{\sum_{i=1}^{n} (\mathbf{x}_i)^2} \sqrt{\sum_{i=1}^{n} (\mathbf{y}_i)^2}}$$

Then we're going to define a function that takes in a movie title as an input and outputs a list of the 10 most similar movies. Firstly, for this, we need a reverse mapping of movie titles and DataFrame indices. we need a mechanism to identify the index of a movie in our `metadata` DataFrame, given its title [6]. Steps of the function which will give us the final recommendation list:

- Get the index of the movie given its title.
- Get the list of cosine similarity scores for that particular movie with all movies. Convert it into a list of tuples where the first element is its position, and the second is the similarity score.
- Sort the aforementioned list of tuples based on the similarity scores; that is, the second element.
- Get the top 10 elements of this list. Ignore the first element as it refers to self (the movie most similar to a particular movie is the movie itself).

- Return the titles corresponding to the indices of the top elements.

Then we can get a result of movies like this:

```
get_recommendations('Iron Man', cosine_sim2)
```

```
7600                        Iron Man 2
1682                              Tron
7764                     TRON: Legacy
7969                     The Avengers
8392                       Iron Man 3
8868          Avengers: Age of Ultron
8872       Captain America: Civil War
2969                          Starman
7122    20,000 Leagues Under the Sea
6359      Zathura: A Space Adventure
Name: title, dtype: object
```

Results

Item-Based collaborative filtering Recommender

```
get_movie_recommendation('Avengers')
```

	Title	Distance
1	Matilda (1996)	0.593354
2	Haunting, The (1999)	0.578944
3	Mercury Rising (1998)	0.568688
4	Speed 2: Cruise Control (1997)	0.567220
5	Reindeer Games (2000)	0.564510
6	Snake Eyes (1998)	0.563339
7	William Shakespeare's A Midsummer Night's Drea...	0.548800
8	Anaconda (1997)	0.507164
9	Edge, The (1997)	0.502303
10	Sphere (1998)	0.464465

Recommendation list using Item-based collaborating with KNN distance. The smaller the distance, the better the result matches with the given movie.

```
get_recommendations_based_on_genres("Father of the Bride Part II (1995)")
```

```
17                      Four Rooms (1995)
18     Ace Ventura: When Nature Calls (1995)
58                       Bio-Dome (1996)
61                         Friday (1995)
79                     Black Sheep (1996)
90                      Mr. Wrong (1996)
92                   Happy Gilmore (1996)
104          Steal Big, Steal Little (1995)
108           Flirting With Disaster (1996)
113                  Down Periscope (1996)
Name: title, dtype: object
```

Calculating top 10 movies to recommend based on the similarity of given movie titles genres.

Evaluating this model with KNN:

```
evaluate_content_based_model()
total = true_count + false_count
print("Hit:"+ str(true_count/total))
print("Fault:" + str(false_count/total))
```

```
Hit:0.8776739889139807
Fault:0.1223260110860193
```

As we can see, we have almost 88% hit with the recommended list. And the number of faults is really low (12%). So, we can say, our model is good.

Collaborative Filtering to **predict ratings**

We split the dataset into 2 parts: training and testing. Then calculated their RMSE, MAE individually using SVD on the test set by splitting them into 5 folds.

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

                  Fold 1  Fold 2  Fold 3  Fold 4  Fold 5  Mean    Std
RMSE (testset)    0.8749  0.8759  0.8767  0.8715  0.8742  0.8746  0.0018
MAE (testset)     0.6716  0.6742  0.6739  0.6692  0.6713  0.6720  0.0018
Fit time          3.50    3.51    3.53    3.51    3.51    3.51    0.01
Test time         0.19    0.10    0.17    0.10    0.17    0.15    0.04

{'test_rmse': array([0.87494171, 0.87591108, 0.87666968, 0.87149773, 0.87417575]),
 'test_mae': array([0.67159744, 0.67417337, 0.6739036 , 0.66922294, 0.6713353 ]),
 'fit_time': (3.500293731689453,
  3.510716438293457,
  3.527679443359375,
  3.5141196250915527,
  3.5056395530700684),
 'test_time': (0.1939084529876709,
  0.09976625442504883,
  0.16556048393249512,
  0.10076546669006348,
  0.16608190536499023)}
```

Then we will make a prediction.

```
svd.predict(77, 7153)
```

```
Prediction(uid=77, iid=7153, r_ui=None, est=4.52091366665309915, details={'was_impossible': False
```

For a movie with ID 77, we get an estimated prediction of 4.52. One startling feature of this recommender system is that it doesn't care what the movie is (or what it contains). It works purely based on an assigned movie ID and tries to predict ratings based on how the other users have predicted the movie.

Then we will again calculate RMSE on the test set using SVD++.

```
print("SVDpp : Test Set")
accuracy.rmse(test_pred, verbose=True)
```

```
SVDpp : Test Set
RMSE: 0.9410

0.9409712053501393
```

Content-Based Recommender

```
get_recommendations('The Dark Knight Rises', cosine_sim2)
```

```
6981         The Dark Knight
6218           Batman Begins
6623             The Prestige
467         Romeo Is Bleeding
5904          State of Grace
6645             Harsh Times
7561             Harry Brown
4021    The Long Good Friday
217         The Glass Shield
2272             In Too Deep
Name: title, dtype: object
```

```
get_recommendations('Iron Man', cosine_sim2)
```

```
7600                         Iron Man 2
1682                               Tron
7764                       TRON: Legacy
7969                       The Avengers
8392                         Iron Man 3
8868          Avengers: Age of Ultron
8872       Captain America: Civil War
2969                            Starman
7122     20,000 Leagues Under the Sea
6359       Zathura: A Space Adventure
Name: title, dtype: object
```

Conclusion and future works:

Watching movies is one of the most popular entertainments in modern society. Nowadays people watch movies a lot. It can happen anytime and everywhere—work or home or in their cars. As there is a huge demand for movies, so a lot of movies are being made too. Throughout 2019, 7,547 most popular English-language movies were released. To save time and effort in searching from this huge supply of movies for a good movie that suits our taste, this movie recommendation system can be used. Even though recommendation or prediction is not 100% accurate, machine learning algorithms recommendations are becoming pretty close [5].

Our next plan is to make a personalized hybrid recommendation system using deep learning. So, each user can have their own recommendation list according to their taste. We already build our own movie-watching website. We can implement our personalized recommendation there to affect the popularity of our website incrementally.

Reference:

[1] Jeong WH, Kim SJ, Park DS, Kwak J (2013) Performance improvement of a movie recommendation system based on personal propensity and secure collaborative filtering. J Inf Process Syst 9:157–172

[2] Viana P, Pinto JP (2017) A collaborative approach for semantic time-based video annotation using gamification. Hum Cent Comput Inf Sci 7:13

[3] Vilakone, P., Park, DS., Xinchang, K. *et al.* An Efficient movie recommendation algorithm based on improved *k*-clique. *Hum. Cent. Comput. Inf. Sci.* **8,** 38 (2018). https://doi.org/10.1186/s13673-018-0161-6

[4] Jingdong Liu, Won-Ho Choi, Jun Liu, "Personalized Movie Recommendation Method Based on Deep Learning", Mathematical Problems in Engineering, vol. 2021, Article ID 6694237, 12 pages, 2021. https://doi.org/10.1155/2021/6694237

[5] A. Surendran, A. K. Yadav, A. Kumar, "Movie Recommendation System using Machine Learning Algorithms", International Research Journal of Engineering and Technology (IRJET), Volume: 07, Issue: 04, Apr 2020

[6] A. Sharma, "Beginner Tutorial: Recommender Systems in Python", Datacamp, May 2020. https://www.datacamp.com/community/tutorials/recommender-systems-python?utm_source=adwords_ppc&utm_campaignid=1455363063&utm_adgroupid=65083631748&utm_device=c&utm_keyword=&utm_matchtype=b&utm_network=g&utm_adpostion=&utm_creative=332602034358&utm_targetid=aud-299261629574:dsa-473406569915&utm_loc_interest_ms=&utm_loc_physical_ms=9074038&gclid=Cj0KCQjw9_mDBhCGARIsAN3PaFPE0NnhlGtTDHbcBOQHljjwoC2FjuyIJiWBHHl2ZAi4CL9e7aoQoVQaAoRWEALw_wcB

[7] A. Kashyap, B. Sunita, S. Srivastava, Aishwarya, A. J. Shah, "A Movie Recommender System: MOVREC using Machine Learning Techniques", Volume 10, Issue No.6, 2020.

[8] Karlgren, Jussi. 1990. "An Algebra for Recommendations." Syslab Working Paper 179 (1990).

[9] Karlgren, Jussi. "Newsgroup Clustering Based On User Behavior-Recommendation Algebra." SICS Research Report (1994).

[10] Karlgren, Jussi (October 2017)A digital bookshelf: original work on recommender systems". Retrieved 27 October 2017.

[11] Shardanand, Upendra, and Pattie Maes. "Social information filtering: algorithms for automating "word of mouth"." In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 210-217. ACM Press/Addison-Wesley Publishing Co., 1995.

[12] Hill, Will, Larry Stead, Mark Rosenstein, and George Furnas. "Recommending and evaluating choices in a virtual community of use." In Proceedings of the SIGCHI conference on Human factors in computing systems, pp. 194-201. ACM Press/Addison-Wesley Publishing Co., 1995.

[13] Resnick, Paul, Neophytos Iacovou, Mitesh Suchak, Peter Bergström, and John Riedl. "GroupLens: an open architecture for collaborative filtering of netnews." In Proceedings of the 1994 ACM conference on Computer supported cooperative work, pp. 175-186. ACM, 1994.

[14] Resnick, Paul, and Hal R. Varian. "Recommender systems." Communications of the ACM 40, no. 3 (1997): 56-58.

[15] https://www.analyticsvidhya.com/blog/2020/11/create-your-own-movie-movie-recommendation-system/

[16] https://towardsdatascience.com/prototyping-a-recommender-system-step-by-step-part-1-knn-item-based-collaborative-filtering-637969614ea#:~:text=KNN%20is%20a%20non%2Dparametric%2C%20lazy%20learning%20method.&text=When%20KNN%20makes%20inference%20about,the%20most%20similar%20movie%20recommendations.

[17] https://towardsdatascience.com/predict-ratings-with-svd-in-collaborative-filtering-recommendation-system-733aaa768b14

[18] https://www.kaggle.com/paramarthasengupta/movies-recommendation-tool-approaching-patterns