Abstract

This project aims to implement a recommendation system on movies, to output movies similar to the input movie and to predict the ratings that a user will give to an unseen movie. This is made possible by implementing an item-item collaborative filtering recommender system, where predictions are made based on past movie-user ratings and similar movies. This recommender system is run by performing a supervised machine learning algorithm, k-Nearest Neighbour (KNN) and evaluated by testing its root-mean-square deviation (RMSD) and recall coverage. As a result, the item-item collaborative filtering movie recommender system performed with a low RMSD and a high recall coverage. Our findings suggest that the movie recommendation model we have built works well. We found out that the system suggests very similar movies based on the user's information. Consequently, not only is it just a movie recommendation system, but it's also convenient for a movie producer to plan a new movie. The experiment results obtained on the MovieLens dataset stipulate that the planned approach could give high performance concerning reliability, potency and deliver correct personalised show recommendations compared with existing methods.

Table of Contents

1 Introduction

There are more than thousands of movies that are available out there, plus many more to come. Since everyone has different taste in movies, it can be challenging for the user to find the movies that they like or dislike. It is important to find a method for it to suggest the movies based on what the user wants and filter out the one that is irrelevant to the user.

Movie recommendation systems offer a way to help the users in classifying users with the same interest. In this paper, we will focus on the movie recommendation system and demonstrate how we build a simple movie recommendation system to predict which movies a user may like.

Collaborative Filtering and Content-based Filtering are the most popular ones for recommendation systems. They are used for filtering items through the sentiments of other people. First, it initially gathers the movie ratings given by people and then recommends movies to the target users with similar tastes and interest.

This report aims to address the objective of being able to analyse and predict which movies a user may like and the rate the user would give to a movie based on the opinions of the users' group; this will be demonstrated using the MovieLens dataset. This objective will be achieved by analysing the relationship between movies and ratings, and through the use of collaborative filtering, it will find similar users based on the user ratings.

The rest of this technical report is planned as follows: In Section 2 we will discuss several approaches for a movie recommender system. In Section 3 we define the dataset. In Section 4, we will provide justifications of the proposed method and how it was implemented in Python. In Section 5, we show our experimental results. In Section 6, we conclude the paper with what can be improved for future work.

2 Related Work

To develop our movie recommender system, multiple approaches were considered. These approaches were based on lectures from the Social and Network Information Analysis subject and existing movie recommender systems. In this section, we will be comparing multiple potential approaches.

2.1 Movie Recommendation System by Collaborative Filtering

Recommendation system, which was one of the most successful information management systems, was introduced by the Tapestry project in 1992. It helps users to filter useless information when handling booming information and provides personalised suggestions. The recommendation system has a wide application among e-commerce and multimedia websites.

Collaborative filtering (CF), which is based on nearest-neighbour, is an effective mechanism in movie recommendation systems. This technique stems from the assumption that individuals that have similar preferences in the past, will also have similar preferences in the future. Essentially, the process of CF involves predicting an individual's interest in a particular item by associating it to like-minded people's interest in that item (Herlocker, Konstan, & Riedl, 2000). The "Like-minded" users (called "neighbours") are derived from a rating database which records evaluation values to movies. For example, as shown in the figure below, User 1 and 2 are considered to have similar taste in movies based on the high ratings they have given for Movie A and B. While User 2 has watched and liked Movie C and User 1 has not, CF allows us to predict that User 1 will also enjoy Movie C. Prediction of a missing rating given by a target user can be inferred by the weighted similarity of his/her neighbourhood. Hence, we can confidently recommend Movie C to User 1.

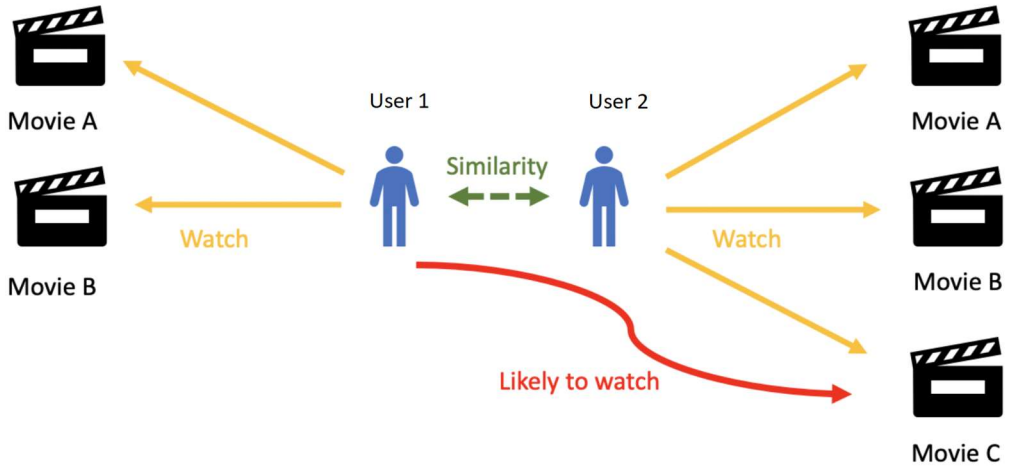

Figure 1: Visual representation of Collaborative Filtering (Leban, 2020).

CF can be implemented either using an item-item based or user-user based approach. The difference between these two approaches is such that movie recommendation for user-user CF is provided based on the users that have similar movie ratings, while movie recommendation for item-item CF is provided based on the similarity between different movies (Bostrom & Filipsson, 2017).

2.2 Movie Recommendation System by Content-Based Filtering

On the other hand, Content-Based Filtering (CBF) is also another common technique utilized in recommender systems. CBF involves making item recommendations or predictions to an individual based on his/her feedback on similar items (Google Developers, n.d). The similarity between items is determined from the item's features or attributes (Son & Kim, 2017). The features or attributes of an item are defined as the descriptions of the item. This can be in the form of tags, genres, or titles. For example, as shown in the figure below, given that Movie A and B are similar based on its movie genre, horror. If the user enjoys watching horror movies, then CBF allows us to recommend other horror movies (Movie B) to the user in which the user might also enjoy.

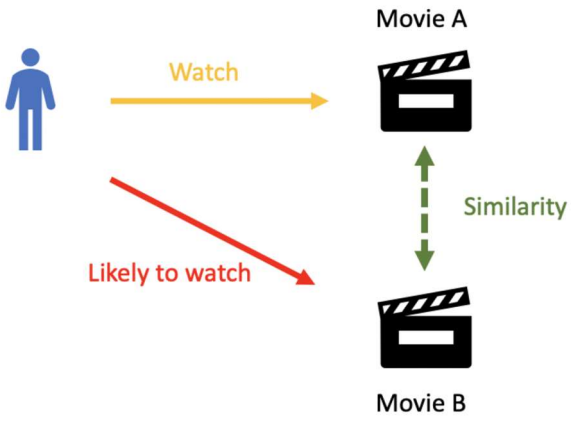

Figure 2: Visual representation of Content-Based (Leban, 2020).

2.3 Movie Recommendation System by Cluster-Based Collaborative Filtering

To deal with scalability problems in movie recommendation, clustering is widely used and provides a comparable accuracy. Clustering aims to partition objects into clusters to minimise the distance between objects within a cluster to identify similar objects. Clustering-based CF prebuilds an offline clustering model to improve k-NN performance. Clustering is able to group users into clusters based on their rating similarity to look for “like-minded” neighbours. The clustering process works offline to build the model. When reaching a target user, the online module assigns a cluster with the largest similarity weight to the user. The prediction rating of a specific item is computed based on the members in the same cluster.

CF with clustering can provide accurate personal recommendation, if the clustering technique is appropriate and the dataset is good.

2.4 PCA-GAKM Based Collaborative Filtering

To improve movie prediction accuracy, (Wang et al., 2014) also proposed a hybrid cluster-based model which uses both online and offline modules to make movie recommendations. Since traditional CF can lead to poor recommendation because of sparseness, a two-phase offline clustering module is suggested to deal with sparsity issues. The first phase is to use the PCA technique to concentrate feature information into lower and less dense spaces; the second phase is to develop an effective GA-KM algorithm based on transformed user space.

3 Dataset

In this section, we will further explain the dataset that were given for the movie recommendation. And thus further breaking down the importance of knowing the dataset before picking our possible solutions.

3.1 MovieLens 20M

The dataset for this study is to build a movie recommendation framework using the Movielens dataset. The data set consists of six variables derived from over 20 million observations. The users in the dataset were chosen at random for inclusion. With no demographic details, each participant had a unique id and rated at least 20 different movies. Thus, by using these parameters, we guarantee that the movie's scores are not skewed. The dataset includes the following feature:

- userId: An integer value between 1 and 138000 assigned to each person.
- movieId: A unique movie is given a numeric value ranging from 1 to 27278.
- Ratings: A numerical value given by each user for a specific movie ranging from 1 - 5 with 0.5 increments with a total of 20,000,263 movie ratings.
- Genres: a character variable assigned to a movie that has a designated genre to which the movie belongs, which has 17 distinct genres.
- Timestamp: Date and Time at which the movie was reviewed which were between 9th January 1995 to 31st March 2015.

3.2 Pre-Processing Dataset

Before we can use the dataset in our assigned software notebook, we must first import it into our applications. We attempted to import the MovieLens 20M original data into Google Colab, but it took way too long, and we did not use the entire dataset due to time constraints. We must import similar libraries in order for Google Colab to work properly. As a result, we whittled down the initial dataset to the first 750 users and 750 movies. Due to the redundancy of the CF methods, some changes were made to remove the tags and relevance from the initial dataset during the pre-processing step. Additional enhancements have been developed, such as the removal of genres from the Movie dataset to eliminate redundant attributes.

	userId	movieId	rating
0	1	2	3.5
1	1	29	3.5
2	1	32	3.5
3	1	47	3.5
4	1	50	3.5
...	...	...	...
25498	750	719	3.0
25499	750	725	3.0
25500	750	733	3.0
25501	750	736	3.0
25502	750	748	3.0

25503 rows × 3 columns

Table 1: User-movie ratings dataset.

	movieId	title
0	1	Toy Story (1995)
1	2	Jumanji (1995)
2	3	Grumpier Old Men (1995)
3	4	Waiting to Exhale (1995)
4	5	Father of the Bride Part II (1995)
...	...	...

Table 2: Movie dataset.

As a result, to complete the pre-processing, we used the train-test-data algorithm to separate the training and test data. We would use 80/20 from the assignment criterion, which means 80 percent test data and 20 percent training data. The code snippet 1 (*refer to Appendix*) shows how the dataset was split in our program

4 Methods

4.1 Justification of Selected Approach

Despite the fact that PCA-GAKM Based Collaborative Filtering and Cluster-Based Collaborative Filtering are technically superior, our group lacks the knowledge, expertise, and experience to implement these approaches. As a result, we will only consider the basic Content-Based Filtering (CBF) and Collaborative Filtering (CF) approach for our movie recommender system. Our team aims to improve our knowledge on recommender systems by implementing either a CBF or CF so that we can further venture and attempt to implement the PCA-GAKM or Cluster-Based approach in the future.

With two approaches for the movie recommender system in mind, our team has chosen to implement the CF approach. This decision is supported by several factors. Firstly, the CBF approach mostly lacks features that CF is capable of, whereby CBF suffers from over-specialization (Melo, Nogueira, & Guliato, 2015). Users will only receive movie recommendations that are similar to those already defined in their profiles (Isinkaye, Folajimi, & B.A.Ojokoh, 2015). This creates no opportunities for the users to explore other movie genres (Educative, n.d.).

Moreover, the recommendation performance of CBF approaches greatly depends on the availability of a rich domain knowledge of items (Isinkaye, Folajimi, & B.A.Ojokoh, 2015). The greater the domain knowledge, the better the recommendation. However, reflecting from the evaluation of our dataset, MovieLens, we discovered insufficient domain knowledge (only genres and tags) of the movies to make meaningful and accurate recommendations. Given that our dataset provided more attributes such as cast, director, and language may deem CBF as a suitable approach. Besides, these attributes are subject to errors, whereby the attributes falsely represent the movies. This contributes further to the inaccuracy of movie recommendations. CF does not require any domain knowledge of movies to make good movie recommendations as it only needs movie ratings/interactions by users.

However, we also discovered weaknesses with the CF approach. CF suffers from what is known as a ‘cold-start’ problem (Milankovich, 2015). ‘Cold-start’ means that some movies will get minimal chances of being recommended which occurs with new movies being listed that have no ratings or movies with minimal frequency of ratings available by users (Melo, Nogueira, & Guliato, 2015). This presents a popularity bias for movies (Abdollahpouri, Mansoury, Burke, & Mobasher, 2019). Even though a movie is good but has no interaction with or ratings by users, CF cannot recommend the movie. ‘Cold-start’ can also be applied to users, whereby users get poor recommendations. This occurs with new users that have not rated any movies or users that do not rate enough movies (Melo,

Nogueira, & Guliato, 2015), since CF relies heavily on users' past movie interaction/rating history to make personalized recommendations.

Furthermore, CF suffers from data sparsity (Grčar, Mladenić, Fortuna, & Grobelnik, 2006). CF recommender systems typically store users' data in a user-item matrix (Schickel-Zuber & Faltings, 2005). Where in our context, the matrix will store all ratings ever performed by the user onto each movie. A user commonly does not rate or interact with every movie. Therefore, we can imagine that the matrix will be missing many values, especially when the system scales up and there are a large number of users and movies. This makes it difficult to calculate the similarity between users or movies. Hence, the data sparsity problem seen in CF will contribute negatively to the reliability and accuracy of movie recommendations (Grčar, Mladenić, Fortuna, & Grobelnik, 2006).

Thankfully, these limitations of CF can be mitigated in the implementation of CF in the Python program, further discussed and demonstrated in the code implementation.A summary of the pros and cons of CF and CBF is summarized in the table below:

	Pros	**Cons**
Collaborative Filtering	● Does not require rich domain knowledge of items. ● Only requires history of user's interaction/ratings on items to provide recommendations. ● Capable of recommending new items outside the user's interest.	● Does not work well with new user/items or user/items with minimal interactions 'cold-start'. ● Popularity bias. ● Data sparsity problem.
Content-Based Filtering	● Does not suffer from data sparsity. ● Does not suffer from 'cold-start' problems.	● Over-specialization. ● Requires rich domain knowledge to provide reliable recommendations. ● Defining features for items is difficult.

Further approaches to consider when deciding to implement a CF-based movie recommender system. This includes implementing either a user-user CF or item-item CF. Item-item CF provides an advantage over user-user CF. User's preferences are likely to change with time (Xiao, Shi, Zheng, Wang, & Hsu, 2018), whereas movies do not change.

So, once the user preferences changes, the entire recommender model needs to be recomputed. Therefore, we have decided to select an item-item CF approach for our recommender system.

4.2 Item-item Collaborative Filtering Implementation

Our item-item CF movie recommender system will be implemented in Python code. Based on our team members' lack of experience in Python coding and recommender systems, we have chosen to code our movie recommender system with reference to an existing recommender system source code and make improvements where appropriate, instead of coding one from scratch. Our goal for this approach is to test the performance of item-item CF movie recommender system and fully understand the workings behind a recommender system by the end of this project. Thus, our movie recommender system will be based on the following source code:

1. https://github.com/So-ham/Movie-Recommendation-System (GitHub Repository, 2020)
2. https://github.com/aliabbas101/WennovationAcademy (GitHub Repository, 2020)

To implement the item-item CF movie recommender system. The first step in the process after pre-processing the dataset is to convert the dataset into a user-item matrix as required by CF. The code snippet 2 (*refer to Appendix*) shows the conversion process where the rows represent the movies while the column represents the users and the data stored in the matrix are the users' ratings on each movie. '0.0' means the user has not rated that particular movie.

userId	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
movieId																				
1	0.0	0.0	0.0	0.0	0.0	5.0	0.0	0.0	0.0	4.0	4.5	4.0	4.0	4.5	0.0	3.0	0.0	0.0	5.0	0.0
2	3.5	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
3	0.0	4.0	0.0	0.0	0.0	3.0	3.0	5.0	0.0	0.0	0.0	3.0	0.0	0.0	0.0	0.0	0.0	0.0	4.0	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
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	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 3: User-item matrix representation of the MovieLens dataset.

As mentioned before, CF suffers from data sparsity and 'cold-start'. The code snippet 4 (*refer to Appendix*) shows the code to mitigate the data sparsity problem where the matrix is compressed. This allows a more efficient row slicing (SciPy.org, n.d.). The code snippet 3 (*refer to Appendix*) shows the process to mitigate the 'cold-start' problem. To qualify for

the recommendation process, a user must rate at least 10 movies and a movie must be rated by 10 users.

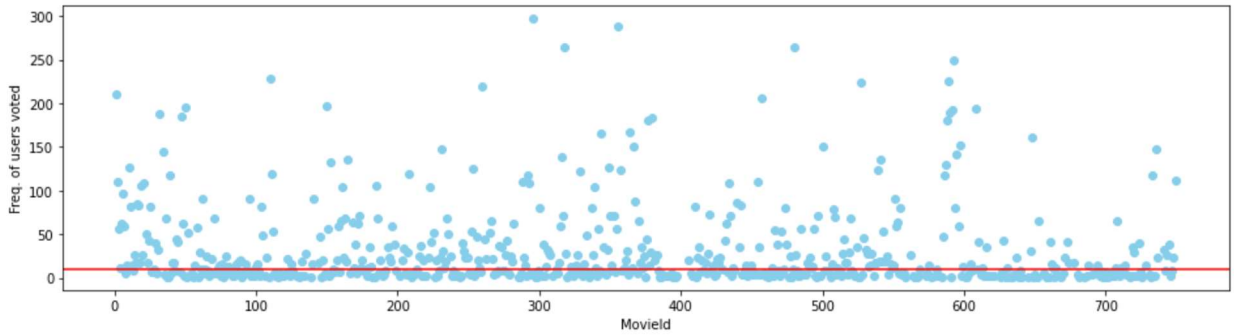

Figure 4: Dot plot of frequency of ratings for each movie.

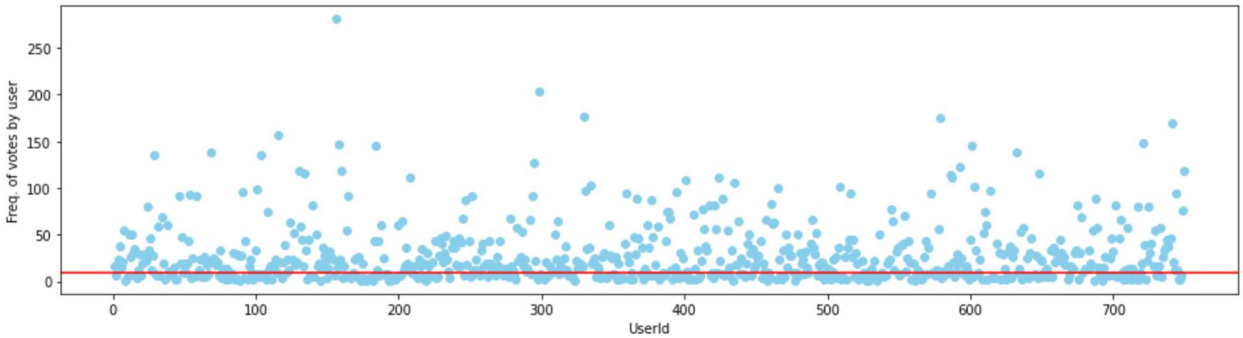

Figure 5: Dot plot of number of movies rated by each user.

To implement the item-item CF movie recommender system, the k-Nearest Neighbour (KNN) is used. KNN allows the system to detect clusters of similar movies (Li, 2017). The training data is used to train the KNN model. The code snippet 5 (*refer to Appendix*) encapsulates the procedure of the system to recommend similar movies to the input movie. The function `get_movie_recommendation` requires two inputs, a movie id and number of movies to recommend. The function will output a dataframe containing all similar movies and its cosine similarity score to the input movie. For example, in the figure below, 10 movies similar to Toy Story (1995) were recommended by the system.

```
1 get_movie_recommendation(1, 10)
```

	Title	Distance
1	Lion King, The (1994)	0.560898
2	Pulp Fiction (1994)	0.555260
3	Fargo (1996)	0.555196
4	Aladdin (1992)	0.538427
5	Terminator 2: Judgment Day (1991)	0.529548
6	Star Wars: Episode IV - A New Hope (1977)	0.518971
7	Forrest Gump (1994)	0.516579
8	Mission: Impossible (1996)	0.513758
9	Twelve Monkeys (a.k.a. 12 Monkeys) (1995)	0.506230
10	Jurassic Park (1993)	0.495030

Figure 6: Example of similar movie to movie id: 1, Toy Story (1995) and each movie's cosine similarity.

After that, we implemented the rating prediction function for a user on an unseen movie. The rating prediction is calculated based on the formula below.

$$r_{xi} = \frac{\sum_{y \in N} s_{xy} \cdot r_{yi}}{\sum_{y \in N} s_{xy}} \quad \text{(weighted average)}$$

Formula: Weighted average of similar movie ratings

The code snippet 6 (*refer to Appendix*) encapsulates the process of the system to predict a movie rating for a given user. The `predict_ratings` function takes the user id, movie id, and number of movies to recommend as input. With the movie id and number of movies to recommend inputs, it will call the `get_movie_recommendation function` to obtain similar movies. The `predict_ratings` function will then calculate the weighted average ratings of all movies that the user has watched from the recommended list.

5 Results

Our movie recommender system was able to predict movie ratings for a user on unseen data. To illustrate, the figure 6 below shows the movie recommender system predicting random samples from the test data. From observation, it was able to make decent predictions close to truth rating.

```
userId     149          userId     142          userId     648
movieId    589          movieId    165          movieId     16
Name: 3363, dtype: int64 Name: 4045, dtype: int64 Name: 694, dtype: int64
Predicted rating: 3.0   Predicted rating: 3.23  Predicted rating: 4.25
Truth: 3.0              Truth: 3.0              Truth: 4.0
```

Figure 7: Movie rating predictions for three random samples from the test data.

To evaluate the movie recommender system, the root-mean-square deviation (RMSD) and recall metrics were used. The RMSD is a commonly used metric to evaluate the performance of recommender systems based on user's explicit movie ratings. It works by measuring the average of the difference/error between all predicted movie rating and the true moving rating (Řehořek, 2016). However, according to Řehořek (2016), the data of movie ratings by users is almost non-existent. Therefore, we need a metric that will be able to evaluate the movie recommender system from the user's implicit ratings. Implicit ratings can be in the form of a user liking, viewing, or clicking on a movie (Bruijn, 2019). The recall metrics aim to achieve this evaluation on user's implicit ratings.

$$\mathrm{RMSD} = \sqrt{\frac{\sum_{t=1}^{n}(\hat{y}_t - y_t)^2}{n}}$$

Formula 2: RMSD

$$R = \frac{T_p}{T_p + F_n}$$

Formula 3: Recall score

The evaluation based on RMSD is straight-forward, where for each user-movie pair in the unseen data, it is passed into the `predict_ratings` function to obtain the movie rating prediction by the user. The error of predicted ratings against truth ratings for each user-movie pair is obtained and the RMSD is calculated as shown in formula 2.

To evaluate the movie recommender system based on implicit ratings of users, we firstly have to translate the explicit ratings into implicit ratings (binary). Our team has decided to approach this by defining that a user-movie rating >= 3 (arbitrary) means that the user liked (1) the movie while a user-movie rating < 3 means the user disliked (0) the movie. The code snippet 7 illustrates the process of the program translating this event where the translation was done on the test data and predicted ratings array.

As a result, our movie recommender system performed with an RMSD of and a recall value of 0.958 and 0.836 respectively. From observation, the results suggest a decent movie recommender system as there is not much deviation between the predicted ratings and truth ratings, and a high recall score is also obtained. The system also does not perform 'too well' and suggests the model is not over-fitted.

6 Conclusion

Using the dataset given, this report synthesised the awareness of approaches considered to create a movie recommendation framework. This report begins with a brief overview of the project and how we as a group will build a movie recommendation system. Our team was able to successfully implement a movie recommender system based on the item-item collaborative filtering approach. Multiple approaches were considered, content-based filtering and collaborative filtering including item-item based and user-user based. The selection for a collaborative filtering approach was factored by the dataset used and the several advantages of collaborative filtering over content-based filtering. To summarize, a collaborative filtering approach would not require a rich movie domain knowledge and only needs movie-user ratings to provide recommendations. Since, the dataset we used does not possess sufficient movie domain knowledge and heavily contains movie-user ratings, the collaborative filtering approach was more appropriate. Although the collaborative filtering approach suffers from data sparsity and 'cold-start' problems, we were able to mitigate this problem through our movie recommender system program. As a result, our movie recommender system performed well with an RMSD of and a recall value of 0.958 and 0.836 respectively.

With that, our team was able to greatly improve our understanding behind recommender systems by implementing a basic recommender system. Although collaborative filtering was the only approach implemented in this project, it would be recommended that content-based filtering was also implemented so that we can compare and further validate the quality and performance of our movie recommender system. It would also be recommended that our team proceed to expand and explore other better methods/approaches used in recommender systems. Firstly, it would be recommended in future works such that our team attempts to implement the hybrid approach (CF and CBF) in the movie recommender system. The hybrid approach allows employing elements of both CF and CBF approaches to maximize performance (Makadia, 2020). Other than that, our team can also attempt to implement the model-based CF as the collaborative filtering is not only limited to item based or user based. Model-based CF is capable of handling data sparsity and scalability issues seen in traditional CF approaches (Aditya, Budi, & Munajat, 2016). PCA-GAKM Based Collaborative Filtering and Cluster-Based Collaborative Filtering approaches can also be attempted.

7 Reference List

Abdollahpouri, H., Mansoury, M., Burke, R., & Mobasher, B. (2019). The Unfairness of Popularity Bias in Recommendation. arXiv:1907.13286v3 [cs.IR].

Aditya, P. H., Budi, I., & Munajat, Q. (2016). A comparative analysis of memory-based and model-based collaborative filtering on the implementation of recommender system for E-commerce in Indonesia: A case study PT X. *2016 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, (pp. 303-308). doi:0.1109/ICACSIS.2016.7872755

Bostrom, P., & Filipsson, M. (2017). Comparison of User Based and Item Based Collaborative Filtering Recommendation Services. Retrieved May 13, 2021, from https://www.diva-portal.org/smash/get/diva2:1111865/FULLTEXT01.pdf

Bruijn, L. d. (2019). *Implementing a Collaborative Filtering Movie Recommender from Implicit Feedback*. Retrieved May 19, 2021, from Medium: https://medium.com/analytics-vidhya/implementation-of-a-movies-recommender-from-implicit-feedback-6a810de173ac#:~:text=and%20the%20like.-,Implicit%20and%20explicit%20feedback,implied%20in%20the%20user%20actions

Educative. (n.d.). What is content-based Filtering? Retrieved May 12, 2021, from Educative: https://www.educative.io/edpresso/what-is-content-based-filtering

GitHub Repository. (2020). Movie-Recommendation-System. GitHub Repository. Retrieved from https://github.com/So-ham/Movie-Recommendation-System

GitHub Repository. (2020). Recommendation System using K-Nearest Neighbors. GitHub Repository. Retrieved from https://github.com/aliabbas101/WennovationAcademy

Google Developers. (n.d). Recommendation Systems. Retrieved May 12, 2021, from Google Developers: https://developers.google.com/machine-learning/recommendation/collaborative/basics

Google Developers. (n.d). Recommendation Systems. Retrieved May 12, 2021, from Google Developers: https://developers.google.com/machine-learning/recommendation/content-based/basics#:~:text=Content%2Dbased%20filtering%20uses%20item,for%20the%20Google%20Play%20store.

Grčar, M., Mladenić, D., Fortuna, B., & Grobelnik, M. (2006). Data Sparsity Issues in the Collaborative Filtering Framework. Proceedings of the 7th international conference on Knowledge Discovery on the Web: advances in Web Mining and Web Usage Analysis, (pp. 58-76). doi:10.1007/11891321_4

Herlocker, J. L., Konstan, J. A., & Riedl, J. (2000). Explaining collaborative filtering recommendations. Proceedings of the 2000 ACM conference on Computer supported cooperative work (pp. 241-250). NY, USA: Association for Computing Machinery. doi:10.1145/358916.358995

Isinkaye, F., Folajimi, Y., & B.A.Ojokoh. (2015). Recommendation systems: Principles, methods and evaluation. Egyptian Informatics Journal, 16(3), 261-273. doi:https://doi.org/10.1016/j.eij.2015.06.005

Leban, J. (2020). Essentials of recommendation engines: content-based and collaborative filtering. Retrieved May 12, 2021, from Towards Data Science: https://towardsdatascience.com/essentials-of-recommendation-engines-content-based-and-collaborative-filtering-31521c964922

Li, S. (2017). *How Did We Build Book Recommender Systems in An Hour Part 2 — k Nearest Neighbors and Matrix Factorization*. Retrieved May 19, 2021, from Towards Data Science: https://towardsdatascience.com/how-did-we-build-book-recommender-systems-in-an-hour-part-2-k-nearest-neighbors-and-matrix-c04b3c2ef55c

Makadia, M. (2020). *Explained – Working and Advantages of a Recommendation Engine*. Retrieved May 23, 2021, from Business 2 Community: https://www.business2community.com/business-intelligence/explained-working-and-advantages-of-a-recommendation-engine-02344556

Melo, E. V., Nogueira, E. A., & Guliato, D. (2015). Content-Based Filtering Enhanced by Human Visual Attention Applied to Clothing Recommendation. IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI) (pp. 644-651). IEEE. doi:10.1109/ICTAI.2015.98

Milankovich, M. (2015). The Cold Start Problem for Recommender Systems. Retrieved May 21, 2021, from Medium: https://medium.com/yusp/the-cold-start-problem-for-recommender-systems-89a76505a7

Řehořek, T. (2016). *Evaluating Recommender Systems: Choosing the best one for your business*. Retrieved May 19, 2021, from Medium: https://medium.com/recombee-blog/evaluating-recommender-systems-choosing-the-best-one-for-your-business-c688ab781a35

Schickel-Zuber, V., & Faltings, B. (2005). Overcoming Incomplete User Models in Recommendation Systems Via an Ontology. Proceedings of the 7th international conference on Knowledge Discovery on the Web: advances in Web Mining and Web Usage Analysis, 4198, pp. 39-57. doi:10.1007/11891321_3

SciPy.org. (n.d.). *scipy.sparse.csr_matrix*. Retrieved May 19, 2021, from SciPy.org: https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csr_matrix.html

Son, J., & Kim, S. B. (2017). Content-based filtering for recommendation systems using multiattribute networks. Expert Systems with Applications, 89, 404-412. doi:https://doi.org/10.1016/j.eswa.2017.08.008

Xiao, Y., Shi, J., Zheng, W., Wang, H., & Hsu, C.-H. (2018). Enhancing Collaborative Filtering by User-User Covariance Matrix. Mathematical Problems in Engineering. doi:https://doi.org/10.1155/2018/9740402

Wang, Z., Yu, X., Feng, N., & Wang, Z. (2014). An improved collaborative movie recommendation system using computational intelligence. *Journal Of Visual Languages & Computing*, *25*(6), 667-675. https://doi.org/10.1016/j.jvlc.2014.09.011

8 Appendix

The group's item-item CF Movie Recommender System Code on Google Colab: https://colab.research.google.com/drive/1jZjORiBKamqnSXCFdVKi47W-y8TJLUxg?usp=sharing

```
train_df, test_df = train_test_split(user_ratings_df, test_size=0.2)
train_df = train_df.reset_index(drop = True)
test_df = test_df.reset_index(drop = True)
```

Code snippet 1: Train-Test data split - 80/20

```
user_item_matrix =
user_ratings_df.pivot(index='movieId',columns='userId',values='rating').fillna(
0)
```

Code snippet 2: User-item matrix conversion.

```
user_item_matrix = user_item_matrix.loc[freq_user_voted[freq_user_voted >=
10].index,:]
user_item_matrix = user_item_matrix.loc[:,freq_movies_voted[freq_movies_voted
>= 10].index]
```

Code snippet 3: 'cold-start' mitigation.

```
csr_data = csr_matrix(user_item_matrix.values)
```

Code snippet 4: 'data sparsity' mitigation.

```
knn = NearestNeighbors(metric='cosine', algorithm='brute', n_neighbors=30,
n_jobs=-1)
knn.fit(train_df_matrix_csr)

def get_movie_recommendation(movie_id, n_movies_to_recomend):
    movie_idx = train_df_matrix[train_df_matrix['movieId'] ==
movie_id].index[0]

    distances , indices =
knn.kneighbors(train_df_matrix_csr[movie_idx],n_neighbors=n_movies_to_recomend+
1)
    rec_movie_indices =
sorted(list(zip(indices.squeeze().tolist(),distances.squeeze().tolist())),\
                        key=lambda x: x[1])[:0:-1]
```

```
    recommend_frame = []

    for val in rec_movie_indices:
        movie_idx = train_df_matrix.iloc[val[0]]['movieId']
        idx = movies_df[movies_df['movieId'] == movie_idx].index

recommend_frame.append({'Title':movies_df.iloc[idx]['title'].values[0],'Distanc
e':val[1]})
    df = pd.DataFrame(recommend_frame,index=range(1,n_movies_to_recomend+1))
    return df
```

Code snippet 5: Function to recommend n movies from an input movie.

```
def predict_ratings(userid, movieid, n_movie_recommend):
    total = 0
    total_weight = 0
    df = get_movie_recommendation(movieid, n_movie_recommend)
    for i in range(len(df)):
        movie_title = df.iloc[i]['Title']
        movie_id = movies_df.loc[movies_df['title'] ==
movie_title].iloc[0]['movieId']
        cos_sim = df.iloc[i]['Distance']
        match_df = train_df.loc[train_df['userId'] == userid]
        match_df = match_df.loc[match_df['movieId'] == movie_id]
        if (match_df.empty):
            continue
        else:
            movie_rating = match_df.iloc[0]['rating']
            total = total + (cos_sim * movie_rating)
            total_weight = total_weight + cos_sim

    if (total_weight == 0):
        pred_rate = 0
    else:
        pred_rate = (total / total_weight)
    return round(pred_rate, 2)
```

Code snippet 6: Function to predict a user's rating on a movie.

```
likes = (test_df['rating'] >= 3).to_list()
test_df['likes'] = likes
test_df['likes'] = test_df['likes'].astype(int)

for index, item in enumerate(predicted_array):
      if item >= 3:
```

```
        predicted_array[index] = 1
      else:
        predicted_array[index] = 0
```

Code snippet 7: Explicit ratings translation into implicit ratings.