Personalized Movie Recommendation Chatbot

Jiajia Liang
University of Virginia
jl9pg@virginia.edu

Jie Fan
University of Virginia
jf6wt@virginia.edu

Linyang Du
University of Virginia
ld2sf@virginia.edu

Abstract

The movie recommendation system is one of the most commonly used products in our daily life. As we are browsing on Netflix or Disney+, there are movie recommendation modules that recommend movies based on some matching and retrieval algorithms. In this project, we will focus on the movie recommending problem. Specifically, we create a system that recommends personalized movies using content-based filtering techniques and collaborative filtering techniques. The work utilizes various techniques we discussed in the search engine pipeline, including data representation, query matching, and ranking model.

1 Introduction

Recommendation plays a key part in information retrieval applications. A recommendation system can help a user avoid an overload of information by understanding the user's need and recommending a suitable answer. The use of recommendation systems exists everywhere and in many industries. Online stores like Amazon and eBay use a recommendation system to help the user find products based on their query and clicked items. YouTube, an online video-sharing platform, utilizes a recommendation system to give users the choice to approach videos that are relevant to their interests. Besides, the recommendation system also is a useful tool in search engines, social media, restaurants, etc.

In our work, we focus on the movie recommendation domain. We want to utilize the techniques we learn in class to create a web application that is able to make a personalized recommendation to users, based on the information of user's preferences and user's history movie rating.

We start with the most straightforward idea of using content-based filtering to match the user's keyword with the movie's information in our database. The system will ask the user to provide keywords for various aspects of the movie, such as actors, directors, genres, and languages. The system will then return the user with a list of movies ranked by the cosine similarity between the user's keyword and the movie vectors.

One personal experience we have is that in the content-based filtering, the keyword we used are generally broad, and lots of movies can be matched with our keywords. However, different users can have different preferences and movie tastes. So we further use user-based collaborative filtering to add personalize output. We make our code public on Github. [1]

[1] https://github.com/szellen/S21-IR-MovieChatbot

The data we use in the project is the movies dataset, which is able to download from Kaggle. The dataset contains the metadata for 45,000 movies released on or before July 2017. In addition, the dataset contains 26 million ratings from 270,000 users for all movies.

The rest of the paper is organized as the following. In section 2, we will provide some related works in this area. In section 3, we will present the overall architecture of our system. Section 4 and section 5 gives implementation details about content-based filtering and collaborative filtering. The web application workflow is presented in section 6, and three case studies are presented and analyzed in section 7. Lastly, we will briefly talk about the limitation and future work.

2 Related Work

2.1 Content-based Filtering

A content-based filtering system selects items based on the correlation between the content of the items. It has been used in many proposed recommendation systems such as PRES[11]. Van et al. introduce a system that makes recommendations by comparing a user profile with the content of each document in the collection. Other works that utilized content-based filtering to make recommendation includes [8], [9], [12] and many more.

2.2 Collaborative Filtering

Collaborative filtering (CF) is a technique used by recommendation systems that are capable of making predictions on the interest of users by collecting information from other users in this system. Adomavicius and Tuzhilin [1] categorized CF algorithms available into content-based, collaborative, and hybrid and summarized possible extensions. Su and Khoshgoftaar [10] concentrated more on CF methods, including memory-based, model-based, and hybrid methods. A more recent experimental comparison of CF algorithms [5] compares user-based CF, item-based CF, SVD, and several other model-based methods.

2.3 Hybrid Recommendation System

While two traditional content-based and collaborative filtering have their advantages, they also have certain disadvantages, some of which can be solved by combining both techniques to improve the quality of the recommendation. The resulting system is known as a hybrid recommendation system[3]. For example, Basilico and Hofmann proposed an online algorithm JRank[2] that systematically integrates all available information such as past user-item ratings as well

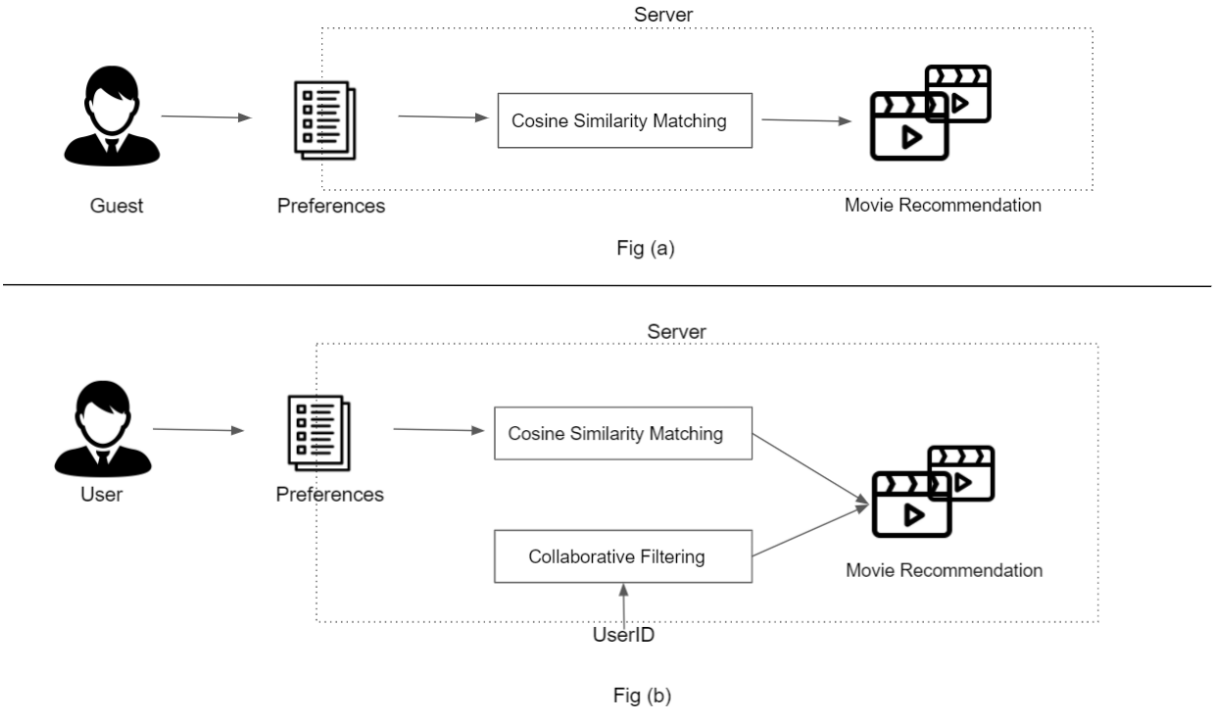

Figure 1. The architecture of the system: a) Content-based Recommendation using a Cosine-Similarity Algorithm for guest user, b) Content-based Recommendation with user-based collaborative filtering for a logged-in user.

as attributes of items or users to learn a prediction function. As an extension of such work, Karatzoglou[6] proposed a collaborative filtering method that allows for flexible and generic integration of contextual information, and de Canois et al. [3] introduces a unifying technique utilizing Bayesian networks.

3 System Architecture

3.1 Overview

The design of our system is shown in figure 1. Our system has two searching modes, namely (1) Guest user, and (2) Logged-in user. For a guest user, we simply use the content-based approach and returning the most relevant movie by comparing the user's preferences and the movie's data. For a logged-in user, we can take advantage of previous ratings given by the users to create a user profile with respect to other users in our system. Next, we can use the user model to fine grain the ranking given by the content-based recommendation. Specifically, for those top relevant movies, movies with the highest predicted score will be ranked the highest in the return recommendation.

The system runs locally, and we use the Flask web framework to create a web application for visualization purposes.

4 Content-based Filtering

Content-based filtering use accumulated information about the user to make recommendations. Our general idea to make recommendation on movie is that if a person likes a particular item, he or she will also like an item that is similar to it.

We use *movies_metadata.csv* data which which contains features for movies including posters, backdrops, budget, genre, revenue, release dates, languages, production countries, and companies. We select come of them to combine with the movie basic information to create a detailed representation of each movie. The features we select include genre, director, description, actors, languages.

CountVectorizer. We will first use CountVectorizer to get a vector for every movie in our datasets. CountVectorizer is a great tool provided by the scikit-learn library in Python and it is frequently used in natural language process research. It can be used to transform a given text into a vector on the basis of the frequency (count) of each word that occurs in the entire text. This is helpful when dealing with multiple texts, and it is capable of converting each word in each text into vectors (for using in further analysis). And after users answer questions in our chatbot, we will get another vector that is based on the input for users' preference on movies. And the features we used to calculate the vector are genre, director, description, actors, and languages for movies.

Cosine Similarity. Then we will calculate the cosine similarity between the vector we get from the queries implemented in our chatbot and the vectors of movies in our datasets. It can measure the similarity between these two vectors. And it measures the cosine of the angle between two vectors projected in a multi-dimensional space. The reason we choose to use cosine similarity as our measurement is that we don't have to worry about the size difference in two vectors we want to compare. The way to calculate the cosine similarity is as following:

$$cos(x, y) = \frac{x \cdot y^T}{||x|| \cdot ||y||} = \frac{\sum_{i=1}^{n} x_i \cdot y_i^T}{\sqrt{\sum_{i=1}^{n}(x_i)^2}\sqrt{\sum_{i=1}^{n}(y_i)^2}}$$

where:

- x is the vector of features for the user's movie preference
- y is the vector of features for every movie in the datasets

5 Collaborative Filtering

Suppose you are a user of our system, while content-based recommendation gives a list of movies that are relevant to your queries, it lacks information about how you, as an individual has different tastes, likes, or dislikes about each movie. The main idea of collaborative filtering is to recommend movies that other users that are similar to you liked. Based on the similarity between you and other users, and the score that other users give for each movie, the collaborative filtering module is able to predict your rating on each relevant movie. Then it will return you with the movie ranked by the predicted ratings. In other words, the module takes *userID* and a list of movies needed to be predicted as input, and return the movie list re-ranked by the predicted score. With such a user model, we are able to add personalized recommendations for each user.

5.1 Collaborative Filtering Algorithm

We formulate our problem as the following:

Problem 1. ***Predicted Rating*** *Given a user a, and movie m, output predicted rating* $pred(a, m)$ *of user a for movie m.*

$$pred(a, m) = \frac{\sum_{b \in N} sim(a, b) * (r_{b,m})}{\sum_{b \in N} sim(a, b)}$$

where:

- $sim(a, b)$ is the similarity between user a and user b.
- M is the set of common rated movies by user a and user b.
- $r_{b,m}$ is the rating of movie m by user b.

5.1.1 Similarity computation. To calculate $pred(a, m)$, we need to calculate $sim(a, b)$ for similarity between user a and some other user b. We evaluate three different similarity functions and pick the one that works the best for our problem. Similarity functions are summarized below:

- Euclidean distance

$$sim(a, b) = \sqrt{\sum_{m \in M}(r_{a,m} - r_{b,m})^2}$$

- Pearson Correlation

$$sim(a, b) = \frac{\sum_{m \in M}(r_{a,m} - \bar{r_a})(r_{b,m} - \bar{r_b})}{\sqrt{\sum_{m \in M}(r_{a,m} - \bar{r_a})^2}\sqrt{\sum_{m \in M}(r_{b,m} - \bar{r_b})^2}}$$

- Cosine Distance 1

$$sim(a, b) = \frac{\vec{a}\vec{b}}{|\vec{a}| * |\vec{b}|}$$

To evaluate which similarity function works the best for our system, we use *rating-small.csv* files, which is a subset of the complete rating file and contains 44994 ratings. We divide them into train and test data using 80%-20% split. The training set has 35731 ratings and the testing set has 9263 ratings. We use RMSE as our evaluation metric, where

$$RMSE = \sqrt{(\frac{\sum(\hat{y} - y)^2}{n})}$$

The performance for three similarity metrics is shown in table 1. Based on the result, we will use Pearson Correlation as our similarity function to calculate $pred(a, m)$.

Similarity Function	RMSE
Euclidean distance	1.0985
Pearson Correlation	1.0364
Cosine distance	1.0483

Table 1

5.2 Data

In this section, we describe the data preprocessing steps we perform and the justification behind such a process. In addition, we also provide some basic statistics for the data used in the collaborative filtering module.

We use the *rating.csv* data to calculate the user profile. The original data contained ratings given by 270, 896 users on 45115 movies, and the data has total 2, 6024, 289 ratings. Considered the data is extremely large, we only use the first 10, 000 users in our project. After trimming down the rating, the rating file now contains 978, 527 rating entry, rated by 10, 000 unique users and rated for 20, 208 unique movies.

Similar to Zipf's law in natural language, we also observe a similar inversely proportional pattern between the user's rating frequency and the user's rank in the frequency table. Figure 1 shows the total number of movies that a user has rated on, the corresponding ranking by the number of movies that the user has rated on. As we can see, the number of movies that a user rated on drop dramatically, and the majority of the users rated less than 10 movies.

The implications of this are that the set M which is the commonly rated movies rated by two users, will be 0 most of the time, and the similarity matrix is an extremely sparse metrics. Therefore, we are wasting lots of time on creating a user's profile but not learning any similarity between the user and other users in our system. In other words, if a user only rated 10 movies, it is very unlikely that other users have watched and rated the same movie. So we cannot determine the similarity score between this user with other users in our system.

In order to balance the efficiency and effectiveness, we decide to further trim the users and the rating data. We keep all the users who have rated more than 100 movies. In total, we have 2374 unique users rated on 19, 930 unique movies, and 755, 536 ratings entry.

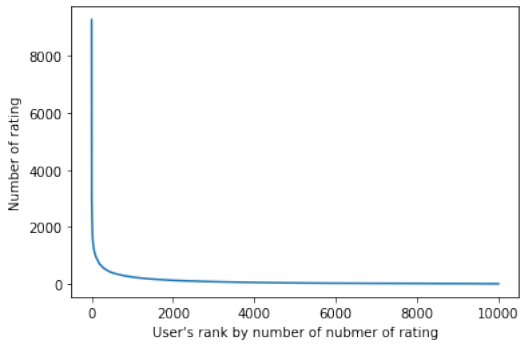

Figure 2. User's rating frequency vs user's rank in the frequency table

6 Web app-Putting Everything Together

In this section, we present the major components of our web application. In addition, we describe how each step correlated with the techniques we describe in previous sections.

6.1 Login

In Figure 3 below, we can see a screenshot of our login page. From this page, we can either log in as a guest or a user that is registered with us before. When logged in as a guest user, we (the movie recommendation chatbot) do not have the guest's past rating on other movies, so we can't calculate a score to rank movies using collaborative filtering. We can only calculate cosine similarity and return a list of all relevant results using content-based recommendations. If logged in as a past user, the system computed the similarity metrics $sim(a, b)$ between user a and all other user b. The calculation takes around 30 seconds as we have more than 2000 users in the system. The metric can be used through the search session until the user logs out of the system.

6.2 Preference Elicitation

In Figure 4, we have the preference elicitation page which the user sees once he/she successfully logged into our system. The recommendation chatbot will start by asking which genre of the movie a user prefers. The user can reply with multiple keywords separated by commas. Then the system will ask the user about which actor(s) and director(s) the user loves. After answering these three questions, our recommendation system will start calculating a list of recommendation results for the user. The web page automatically jumps into the result display page after the calculation is completed.

6.3 Results Display

The result page of our system can be seen in Figure 5. The list of a movie recommended by our system takes up the majority of the space below. The movies are ranked from most relevant to least relevant from top to bottom and from left to right. From each movie, we can see an image of the movie, its name, the reason why it was recommended, its genres, and a predicted score from collaborative filtering. Since some movies in our database do not have enough ratings to compute a predicted score, we prioritize those movies with the predicted score over those that do not. If the number of a movie with the predicted score is less than 8, we append the movie with the highest similarity score but not already in the lists. Those appended movies will be labeled as "Most Relevant" movies on our results page. For those movie labels with "Highest Predicted Score", this movie is ranked using the collaborative filtering technique we describe in section 5.

The user can start a new search, or switch to another user, which the system will need to re-compute the similarity metrics for the new *userID*.

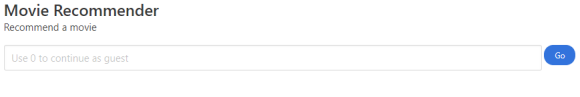

Figure 3. Login page

Figure 4. Preference elicitation page

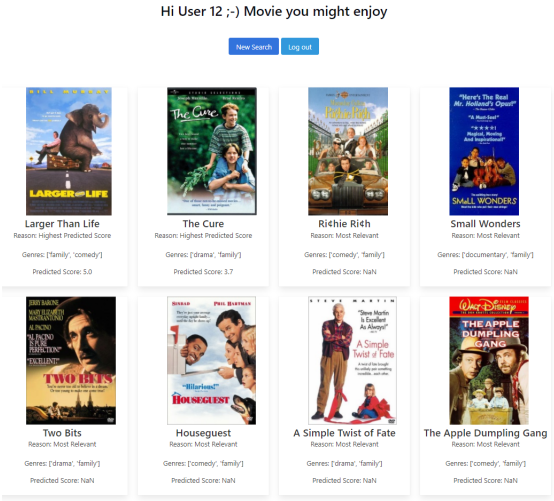

Figure 5. Result page

7 Case Studies

To evaluate our content plus collaborative filtering-based recommendation chatbot, we will conduct several case studies on the system that we have built. There will be three case studies: guest user using system with content-based recommendations, user 1 using system with collaborative filtering, and user 11 also using collaborative filtering. The guest user demonstrates a base case where our system can't use a user's past rating to do collaborative filtering and can only do content-based recommendations. The 2 other collaborative filterings will be run and compared to see if the recommended movies are truly personalized.

7.1 Case 1

In the first case study, we want to see if the cosine similarity content-based recommendation system performs per specification of our cosine similarity design. The cosine similarity approach will try to recommend movies that match any of the keyword queries the user listed on the preference page. This means the result shown from a guest user query will be a list of movies that either satisfy one or more of genre, actor, or director. The order of the movies should be based on cosine similarity angle (from more relevant to less relevant). We will test the guest interaction using this query below.

$$query = \{genre : love, actor : TomHanks, director : NoraEphron\}$$

Figure 6 and Figure 7 is the result obtained after submitting the query as a guest user. As we have discussed, the recommended list of movies is suggested using cosine similarity measurements so each movie in the suggested list matches at least one of the queries we have given. Sleepless in Seattle, Apollo 13, Philadelphia, and many other movies in this list are starred by Tom Hanks. One thing to point out here is "Sleepless in Seattle" is a movie that matches all three queries. Our recommendation bot successfully recognizes Sleepless in Seattle has the most matching criteria (higher cosine similarity) therefore put it as the top movie it recommended.

An anomaly here is the movie "Ladybird Ladybird" because it does not match any of the query keywords we provided. We think it is in the list because its genre is close to love. Also, we see that the movie recommendations do not prioritize any single query and the recommendation does not know whether the user wants to see love story or wants to see Tom Hanks. We want to use the system with collaborative filtering to obtain a more personalized and more accurate recommendation.

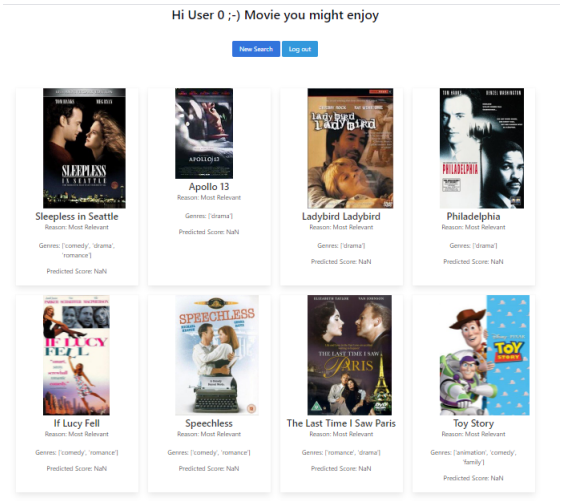

Figure 6. Case 1 result part 1

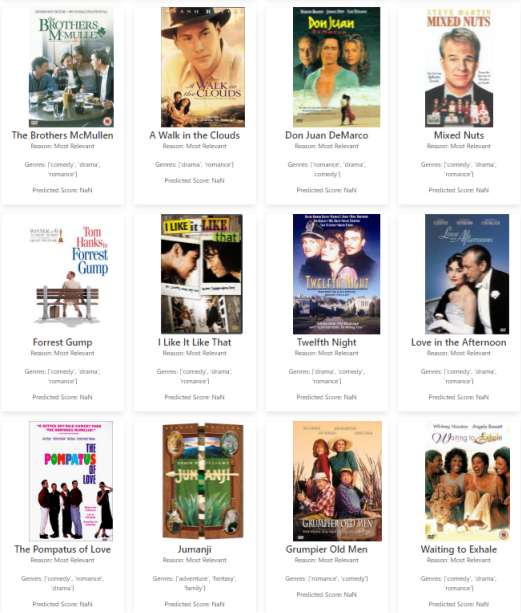

Figure 7. Case 1 result part 2

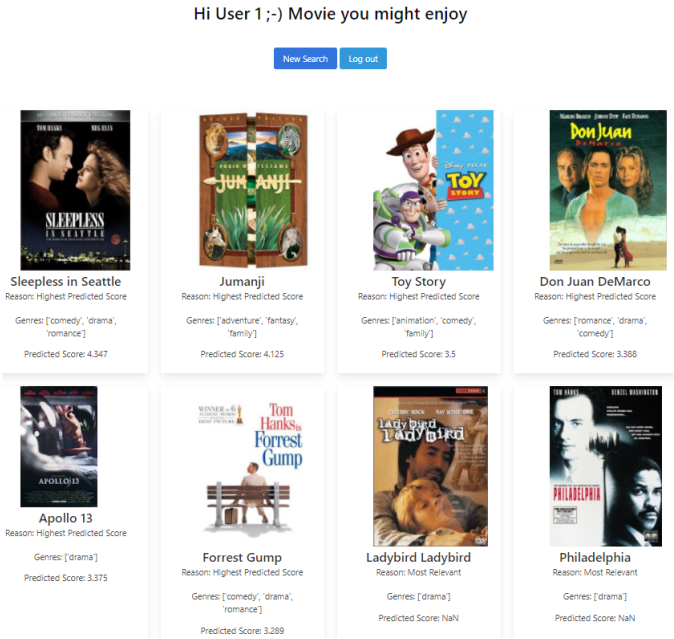

Figure 8. Case 2 result

7.2 Case 2

In the second study, we want to see when given the same query as above, whether the collaborative filtering technique can provide a more personalized review for a registered user. We plan on using user 1 stored in our system. User 1 has 30 past ratings, all on different movies. Therefore, we expect our collaborative filtering algorithm will suggest a more personalized list of recommendations for user 1. We expect "Sleepless in Seattle" will still rank high in the personalized ranking. We also want to see movies ordered by the similarity score between their past rating and the movie in the list. Our input query is the same as the one used in case 1.

$$query = \{genre : love, actor : TomHanks, director : NoraEphron\}$$

In Figure 8, we see the result returned for User 1. First, the list of movie recommended for user 1 is different compared to the one recommended for guest users. Secondly, "Sleepless in Seattle" still ranked first in our list, and has the highest similarity score. Thirdly, the movie rankings are more explainable. The movies ranked on the top level have related genres and have actors/directors mentioned in the query as well. "Sleepless in Seattle", "Toy Story", "Don Juan DeMarco" all relate to genre "love, friendship, romance". These points have proven our collaborative filtering gives a more personalized and useful recommendation.

One anomaly to point out for this recommendation is "Jumanji" does match any keyword we gave, but is ranked second highest with a score close to "Sleepless in Seattle". One explanation for this behavior is Jumanji is distantly related to "love" and the similarity algorithm might recognize other users who have similar tastes have liked this movie.

7.3 Case 3

The third case study is to demonstrate collaborative filtering techniques affect different users' recommendations when provided the same query. We would like to use User 11 because he/she has rated over 150 different movies, therefore its similarity score for movies must be considerably different compared to User 1's. We should expect "Sleepless in Seattle" to rank very high just like the previous 2 studies. The query used is the same as before.

The result can be seen in Figure 9. We can see that most of the movies suggested were the same and stayed in order. However, we also observed Apollo 13 has a higher score than Don Juan DeMarco for User 11. This observation demonstrates our collaborative filtering algorithm is personalizing based on the user's past rating on other movies. Apollo 13 is ranked higher can be explained if the user has rated "drama" movies highly in the past and similar users have rated Apollo 13 highly. Therefore, our engine thinks Apollo 13 is more similar to user 11's taste and recommends it more for user 11.

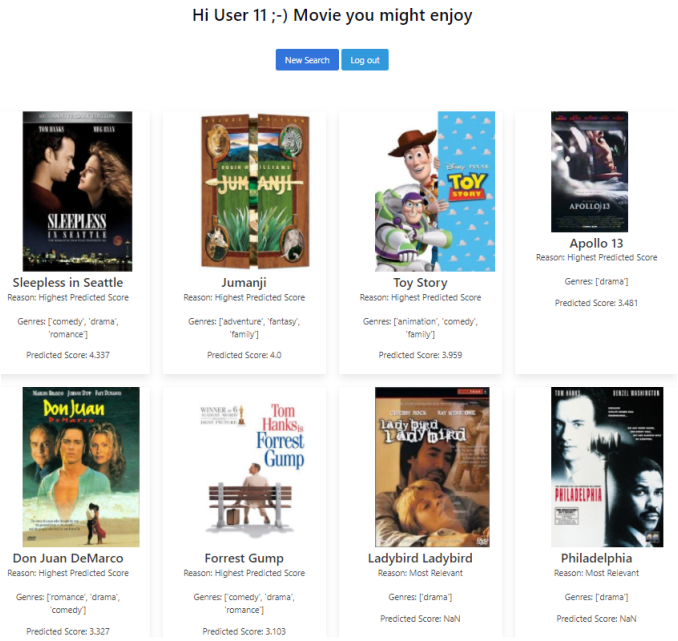

Figure 9. Case 3 result

8 Limitation and Future Work

Our recommendation system can provide a personalized recommendation to registered users. However, this system can be improved in many aspects that we are going to discuss in this section.

First of all, the current system runs on localhost only. In the future, we would like to deploy this system on a public server so the general public can use it to obtain their personalized result. To make this system public, we also need to create authentication for new users to create an account and allow all users to rate more movies.

Another limitation for our system is the user can only enter keywords instead of natural language sentences. Also, each answer can only consist of one type of query. We would like to improve this problem by using natural language processing techniques.

Although this recommendation system has novelty in using collaborative filtering, it is very similar to popular movie recommendation systems in IMDB or Movielens. We would like to incorporate concepts such as Knowledge Graph in explainable conversational recommendation systems to build a more personalized user profile while the user is searching for movies. This way, our recommendation result can be based on what the user has searched and also on what the user has rated. This technique will require more data and model training compared to our current design but produces a more personalized result.

9 Conclusion

In conclusion, we have designed and built a collaborative filtering recommendation that provides personalized recommendations of movies based on past user ratings and user query keywords. We have demonstrated our system can provide useful, personalized recommendations by conducting three case studies of our system. We have learned a lot from this project and got to apply concepts such as cosine similarity. We have encountered lots of difficulties while implementing the base code for our project, but we are delighted our system demonstrates expected behaviors.

10 Miscellaneous

In our original proposal, we were intended to build an explainable conversational recommendation system. The plan was to expand on the conversation recommendation system proposed by Li et al [7] and add explainability using a knowledge graph. However, while we were trying to re-construct the system that Li et al proposed using the code and data they provided, we ran into various problems that prevent us to continue with our original plan. Specifically, the original work merges the ReDial Data[7] and Movielens dataset[4], which the merging steps required a huge amount of RAM resources that most of us are not able to complete. In addition, we also had trouble compiling and run their recommender model using the code provided.

Therefore, we made a difficult decision to give up our original plan and create a personalized movie recommendation chatbot instead. The lesson we learn from this is that, although at the early proposal stage we spend a lot of time making sure we have the data available and the plan is manageable, we should've run the work before consider using it as our foundation.

References

[1] G. Adomavicius and A. Tuzhilin. 2005. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *IEEE Transactions on Knowledge and Data Engineering* 17, 6 (2005), 734–749. https://doi.org/10.1109/TKDE.2005.99

[2] Justin Basilico and Thomas Hofmann. 2004. Unifying collaborative and content-based filtering. In *Proceedings of the twenty-first international conference on Machine learning*. 9.

[3] Luis M De Campos, Juan M Fernández-Luna, Juan F Huete, and Miguel A Rueda-Morales. 2010. Combining content-based and collaborative recommendations: A hybrid approach based on Bayesian networks. *International journal of approximate reasoning* 51, 7 (2010), 785–799.

[4] F. Maxwell Harper and Joseph A. Konstan. 2015. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems* 5, 4 (Dec. 2015). https://doi.org/10.1145/2827872 Publisher Copyright: © 2015 ACM. Copyright: Copyright 2016 Elsevier B.V., All rights reserved.

[5] Zan Huang, Daniel Zeng, and Hsinchun Chen. 2007. A Comparison of Collaborative-Filtering Recommendation Algorithms for E-Commerce. *IEEE Intelligent Systems* 22, 5 (Sept. 2007), 68–78.

[6] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. 2010. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*. 79–86.

[7] Raymond Li, Samira Kahou, Hannes Schulz, Vincent Michalski, Laurent Charlin, and Chris Pal. 2019. Towards Deep Conversational Recommendations. arXiv:1812.07617 [cs.LG]

[8] Peretz Shoval, Veronica Maidel, and Bracha Shapira. 2008. An ontology-content-based filtering method. (2008).

[9] Jieun Son and Seoung Bum Kim. 2017. Content-based filtering for recommendation systems using multiattribute networks. *Expert Systems with Applications* 89 (2017), 404–412.

[10] Xiaoyuan Su and Taghi M. Khoshgoftaar. 2009. A Survey of Collaborative Filtering Techniques. *Adv. in Artif. Intell.* 2009, Article 4 (Jan. 2009), 1 pages. https://doi.org/10.1155/2009/421425

[11] Robin Van Meteren and Maarten Van Someren. 2000. Using content-based filtering for recommendation. In *Proceedings of the machine learning in the new information age: MLnet/ECML2000 workshop*, Vol. 30. 47–56.

[12] Chun Zeng, Chun-Xiao Xing, and Li-zhu Zhou. 2003. A personalized search algorithm by using content-based filtering. *Journal of Software* 14, 5 (2003), 999–1004.