Collaborative Movie Recommendation System

Mohammed Zakiuddin
Computer Science and Engineering
University of Texas at Arlington
Arlington, United States of America
mohammedahmed.zakiuddin@mavs.uta.edu

Abstract—**Recommendation systems have become an essential part of many online platforms and services in recent years. Among these, movie recommendation systems have been widely used by many streaming services to help users discover new movies that they may enjoy. Collaborative filtering is a popular approach for building recommendation systems that rely on the behavior of users to make predictions about their interests. In this paper, we present a collaborative filtering-based movie recommendation system that utilizes user behavior data to suggest movies that are likely to be of interest to them. The proposed system is evaluated using real-world movie rating data and shows promising results in terms of accuracy and effectiveness.**

I. Introduction

Our project aims to implement a movie recommendation system using collaborative algorithms. The primary reason for choosing this data set and algorithm is that movie recommendation systems are widely used in the entertainment industry to personalize user experiences and increase user engagement. Collaborative algorithms are known to be effective in generating accurate recommendations by analyzing user behavior and preferences. Therefore, we believe that a collaborative algorithm-based movie recommendation system will enable us to provide more personalized recommendations to users based on their past behavior and preferences.

The challenges associated with building a movie recommendation system using collaborative algorithms include handling large data sets, ensuring scalability, and addressing the cold-start problem. The data set that we plan to use for this project includes movie.csv, which contains movie titles, genres, and movie IDs, and ratings.csv, which contains user IDs, movie IDs, ratings, and timestamps. These data sets present a challenge in terms of managing and processing large amounts of data efficiently.

To overcome these challenges, we plan to implement various data pre-processing and feature engineering techniques to optimize the performance of the collaborative algorithm-based recommendation system. We believe that by addressing these challenges and implementing the recommendation system effectively, we can provide more personalized and accurate recommendations to users, thereby enhancing their overall movie watching experience.

II. Related Works

A. Literature Review

Companies have grown interested in people's views and opinions as data collection has become a big component of our daily lives over the last few years. Companies have been able to design better systems for their clients as a result of this desire for data collection. However, all of this data collection has resulted in the development of recommendation systems, in which we provide data to a system, and it recommends what an individual should do. A movie recommendation system, for example, collects user data and their opinions on specific movies in order to understand what certain individuals liked and hated, allowing it to learn how to appropriately recommend a movie to a person.

We had looked into numerous machine learning algorithms for recommendation systems and, in particular, had repeatedly encountered three key techniques. We frequently came across the three machine learning techniques known as collaborative filtering, content-based filtering, and demographic filtering.

A movie recommendation system could be implemented in a variety of ways. For instance, demographic filtering makes recommendations based on how well-liked a particular movie is overall rather than making recommendations that are specific to a particular person. With the use of this algorithm, all of the movies receive ratings based on user reviews, and recommendations are made based on how well-liked they were by viewers. The algorithm employs a mathematical calculation based on user ratings, the quantity of reviews, the average rating, and the mean vote for a specific movie. By applying this mathematical method, it will rank movies from a specific genre in order of popularity based on reviews and ratings from viewers, and it will suggest the movie with the highest rating to a customer. Due to the fact that every person is unique and has distinct interests, this recommendation system is determined to be excessively simplistic and ineffective.

Content-based filtering is an additional algorithm for movie recommendations. Content-based filtering basically gathers information based on whether or not a user liked a particular movie, and if they did, it analyzes the movie's genre, stars, director, and many other elements and suggests to users other movies with the same aspects. It bases its argument on the idea that someone who like one aspect of a movie would also enjoy a movie with that same aspect.

There are two approaches to use this methodology, the first of which is to compute and assign each movie a similarity score based on its narrative. The word vector of each movie's plot is converted to a term frequency-inverse document frequency vector, which is then used to calculate the similarity score for each plot. Every graphic in the data set is subjected to this calculation, which determines the frequency of a word used in the plot description. This will generate a matrix with rows of movie names and columns of key terms from a movie's narrative. Using this matrix, we can compute a similarity score for each movie using the cosine similarity scores.

With this, it will now suggest movies to a viewer based on the plot of another movie they previously enjoyed. An alternative method of putting the content-based filtering algorithm into practice is to use the top three actors in a movie, the director, and the genre instead of the story line. This works in a manner similar to the plot-based recommendation system, but instead of using the words from the plot description, it uses the names of the top three actors in the movie, the director, and the genre to determine how similar the actors and genres are, assigns a similarity score, and then suggests movies to you.

The collaborative filtering method is the last machine learning algorithm that could be used to recommend movies. This algorithm can be applied in two different ways: the first is to suggest movies to a user based on comparable movies that other users have enjoyed, and the second is to suggest movies to a user based on similar movie genres that other users have enjoyed. For instance, if another user with similar preferences enjoyed a particular actor, it would suggest films starring that actor to the user. In order to propose a movie to a user, the cosine similarity formula is used to locate comparable user interests and movies that they have rated. This is the first technique to put the collaborative filtering algorithm into practice. The second method would likewise discover similar users using the cosine similarity, but in addition to only looking at the movies the user had rated, it would also compare that movie to other movies with similar actors and plots in order to suggest similar movies to a user that another user had enjoyed.

Our team has decided to implement collaborative algorithms for the movie recommendation system as part of our term project. We plan to utilize two data sets for this purpose - movie.csv and ratings.csv. The movie.csv data set contains information about movie titles, genres, and movie IDs, which will be used to build the search engine. On the other hand, ratings.csv consists of user IDs, movie IDs, ratings, and timestamps, which will be used for the recommendation system. After reviewing several articles on this topic, we believe that these implementations will be helpful in achieving our project goals.

To explain how these articles were helpful in our implementation, we can say that they provided valuable insights into the challenges and solutions related to building a movie recommendation system using collaborative algorithms. By reviewing these articles, we were able to gain a better understanding of the various approaches and techniques used in the field. These articles helped us in identifying the limitations of existing approaches and the opportunities for improvement in our implementation. Additionally, they provided us with a comprehensive overview of the data pre-processing, feature engineering, and model selection techniques that are commonly used in the field. Overall, these articles were instrumental in shaping our approach to building the movie recommendation system and helped us address the challenges in a more effective manner.

III. Movie Recommendation System Design and Implementation

The goal of our experiment is to design a movie recommendation system that suggests movies to users based on their preferences. This section will describe in detail the overall design and process of this experiment, such as preprocessing the movie data, extracting features from the processed data, and applying the chosen machine learning models to compare accuracy. Additionally, the tools that are used for each step and a detailed description of the data set will also be included.

To achieve this goal, we will start by collecting a large data set of movies and their associated features, such as genre, director, and cast. We will then preprocess this data by cleaning and standardizing the movie titles and extracting additional features from the movie metadata.

Next, we will extract relevant features from the preprocessed data, such as genre, director, and cast, and represent them in a format suitable for machine learning models. We may also choose to incorporate additional features, such as user ratings and reviews, to improve the accuracy of our recommendation system.

Finally, we will apply several machine learning models, such as collaborative filtering and content-based filtering, to compare their accuracy in predicting movie recommendations for users. We will evaluate the performance of these models using various metrics, such as precision and recall, to determine which model provides the best recommendations for our users.

Throughout this experiment, we will be using various tools and libraries, such as Python, scikit-learn, and pandas, to preprocess the data, extract features, and apply the machine learning models. We will also be using a variety of data sets, such as the Movie Lens data set, to train and test our models.

A. Tools and Movie Data Set

The movie recommendation system is implemented using Python 3.9.7 programming language and the following libraries and packages [3]:

- Pandas version 1.3.4
- Numpy version 1.20.3
- Scikit-learn version 0.24.2
- Matplotlib version 3.4.3

The dataset used in this implementation consists of two CSV files: movies.csv and ratings.csv. The movies.csv file contains 62,423 movies with the following 3 features:

- MovieId: The unique identifier of the movie.
- Title: The title of the movie.
- Genres: The genre(s) of the movie.

The ratings.csv file contains 25,000,095 ratings given by 162,541 users to different movies. The file has the following 4 features:

- UserId: The unique identifier of the user.
- MovieId: The unique identifier of the movie.
- Rating: The rating given by the user to the movie (on a scale of 0.5 to 5 in increments of 0.5).
- Timestamp: The timestamp of the rating (in seconds).

To input the data into the collaborative filtering algorithm, the ratings.csv file was first preprocessed to create a user-movie rating matrix where each row represents a user and each column represents a movie. The missing ratings were imputed with the mean of the ratings given by the same user or the mean of the ratings given to the same movie. This resulted in a sparse matrix with 162,541 rows (one for each user) and 62,243 columns (one for each movie).

B. Preprocessing Movie Data

After importing the data set into the program and transforming it into a data frame using the Pandas Library, we are left with data that contains a lot of extraneous information that may have a detrimental impact on the program, therefore pre-processing is an important step in any natural language processing (NLP) project. The primary purpose of pre-processing is to turn raw text data into an analysis-ready format. The steps listed below are widely employed in pre-processing:

- Cleaning Data - Cleaning data involves removing unwanted characters, punctuation, and stop words. In our case, we used the Pandas library to read the movie data set and removed unwanted characters from the movie titles using the regular expression library.

  Cleaning the data set or movie titles with regex: Before applying feature extraction techniques, it is important to clean the data set by removing noise components such as special characters, hyperlinks, and emojis. This is typically achieved using regular expressions (regex) in Python. For example, the ”clean title” function uses regex to remove all characters except letters, numbers, and spaces from the movie titles. This step ensures that the data is in an acceptable format and reduces the chance of errors during feature extraction and modeling.

- Tokenization - Tokenization involves splitting the text into smaller units called tokens. In our case, the title of the movies was tokenized into individual words.

- Stemming or Lemmatization - Stemming and lemmatization are used to normalize the text by converting words to their base form. In our case, we used the NLTK library to perform lemmatization.

C. Feature Extraction

The next step after pre-processing is feature extraction. Feature extraction involves transforming the pre-processed text data into numerical features that can be used by a machine learning model [2]. The most commonly used methods for feature extraction are:

- Bag of Words (BOW) - BoW is a method used to represent text data in a numerical format. It involves creating a vocabulary of all the unique words in the text corpus and counting the frequency of each word. The resulting matrix is known as a document-term matrix. In our case, we used the TfidfVectorizer from the Scikit-learn library to create a document-term matrix. In our code, we are using the TF-IDF (Term Frequency-Inverse Document Frequency) feature extraction method. This method is commonly used in natural language processing to convert text into a numerical vector representation that can be used in machine learning models. The TF-IDF score reflects the importance of a word to a document in a collection, with more importance given to words that occur frequently in the document and less importance given to words that occur frequently across multiple documents. The ngram range parameter specifies the range of n-grams to be used in the TF-IDF vectorizer. In our code, it is set to (1,2), which means that both unigrams and bigrams will be used as features.

- Word Embeddings - Word embeddings are a way to represent words in a high-dimensional vector space. Each word is assigned a vector of real-valued numbers, and the vectors are trained to capture the semantic meaning of the words. Word embeddings have been shown to be very effective in NLP tasks such as sentiment analysis, text classification, and machine translation.

  In this project, Bag of Words (BOW) approach has been used for feature extraction instead of word embeddings. The main reason behind this choice is that BOW is a simple and efficient method for representing text data in a high-dimensional space. BOW does not consider the order of words in a sentence, but rather counts the frequency of each word in the entire corpus and uses this count as a feature for each document. While word embeddings can capture the meaning and context of words, they can also be more complex and computationally expensive to train. In this case, the simplicity and speed of BOW make it a good choice for this project.

D. Machine Learning Models

The final step in building a movie recommendation system is to train a machine learning model. The goal of the machine learning model is to learn the underlying patterns in the data and use them to predict which movies a user might like. The most commonly used machine learning models for recommendation systems are [1]:

- Demographic Filtering- They offer generalized recommendations to every user, based on movie popularity and/or genre. The System recommends the same movies to users with similar demographic features. Since each user is different , this approach is considered to be too simple. 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.

- Content Based Filtering - Content-based filtering is a method used to make recommendations based on the attributes of the items themselves. The idea behind content-based filtering is that if a user likes an item with certain attributes, they are likely to like other items with similar attributes. They suggest 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.

- Collaborative Filtering - Collaborative filtering is a method used to make recommendations based on the preferences of similar users. The idea behind collaborative filtering is that users who have similar preferences in the past are likely to have similar preferences in the future. Collaborative filtering can be either user-based or item-based. Collaborative filters do not require item metadata like its content-based counterparts.

- Hybrid Filtering - Hybrid filtering is a combination of collaborative filtering and content-based filtering. The idea behind hybrid filtering is to combine the strengths of both methods to improve the accuracy of recommendations.

Hybrid and Collaborative filtering are popular Machine Learning models used in Recommendation Systems due to their ability to provide more accurate and personalized recommendations. Collaborative filtering makes recommendations based on the similarity between users or items, while Hybrid filtering combines multiple recommendation techniques to improve the accuracy and coverage of recommendations.

These models have several advantages over other filtering techniques such as Content-Based filtering. Collaborative and Hybrid filtering models can handle the cold-start problem, where a new user or item has no prior ratings or information, by leveraging the behavior and ratings of other users or items. They also do not require explicit feature engineering as in Content-Based filtering, making them more scalable and efficient for large data sets.

Furthermore, Collaborative and Hybrid filtering can capture complex user-item interactions and recommendations can be easily updated in real-time as new data becomes available. These models have been successfully used in various applications. Overall, Hybrid and Collaborative filtering offer a robust and versatile approach to building effective recommendation systems.

In order to determine the most accurate and effective recommendation model, we will be exploring and comparing various filtering models. These models include content-based filtering, collaborative filtering, demographic filtering and hybrid filtering. By using all of these models, we can evaluate each one's strengths and weaknesses and determine which one will produce the most accurate and relevant recommendations for our users. This approach allows us to take advantage of the strengths of each model, ultimately leading to a more accurate and effective recommendation system. Additionally, by comparing the accuracy of each model, we can determine which one is best suited for our specific use case and provide the highest quality recommendations to our users.

E. Collaborative Filtering

To create a movie recommendation system using collaborative filtering, we first loaded the ratings.csv file into a Pandas data frame called ”ratings”. We then defined a function called ”find_similar_movies” that takes a movie_id as input and returns the top 10 recommended movies based on collaborative filtering.

The ”find_similar_movies” function first identifies users who have rated the input movie highly (rating >4) and extracts their movie ratings. It then calculates the percentage of users who have rated other movies similarly (rating >4) and filters out movies that are rated by less than 10% of the similar users. This reduces the noise and helps identify movies that are more relevant to the user's preferences.

Next, the function calculates the recommendation score for each movie by dividing the percentage of similar users who have rated it highly by the percentage of all users who have rated it highly. Finally, the function returns the top 10 movies with the highest recommendation scores, along with their titles and genres.

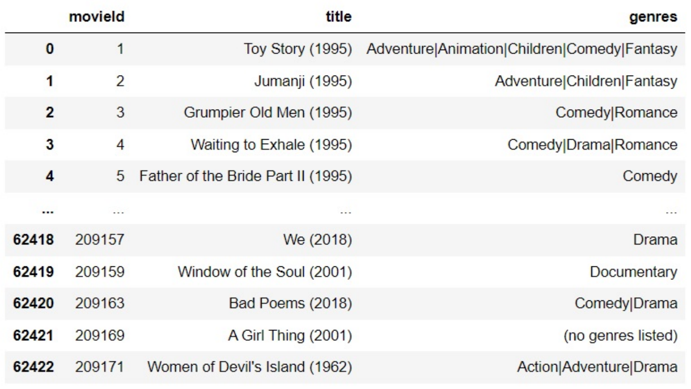
To make the recommendation system interactive, we used the ipywidgets library to create a text input field where the user can enter a movie title. We then defined an ”on_type” function that is called every time the user types a character. This function first clears the output of the recommendation list and then calls the ”search” function to search for movies that match the user's input. If a matching movie is found, the function calls the ”find_similar_movies” function to generate the recommended movies and displays them in the recommendation list.

The above code can be used to create an interactive movie recommendation system that recommends movies to users based on their input movie title. The system uses collaborative filtering to generate the recommendations and can be further improved by incorporating other features such as movie genres, actors, and directors.

F. Visualization of the Input and Output

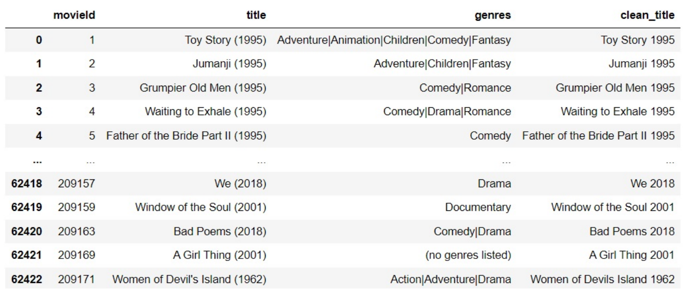
Initially, the process involved reading the information from the file ”Movie.csv” and converting it into a pandas data frame.

The next step was to make sure we have the data set cleaned up to avoid any complications while we train the model.



	movieId	title	genres
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
1	2	Jumanji (1995)	Adventure|Children|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance
4	5	Father of the Bride Part II (1995)	Comedy
...	...	...	...
62418	209157	We (2018)	Drama
62419	209159	Window of the Soul (2001)	Documentary
62420	209163	Bad Poems (2018)	Comedy|Drama
62421	209169	A Girl Thing (2001)	(no genres listed)
62422	209171	Women of Devil's Island (1962)	Action|Adventure|Drama

Fig. 1. Pandas Dataframe of Movie.csv



	movieId	title	genres	clean_title
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 1995
1	2	Jumanji (1995)	Adventure|Children|Fantasy	Jumanji 1995
2	3	Grumpier Old Men (1995)	Comedy|Romance	Grumpier Old Men 1995
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	Waiting to Exhale 1995
4	5	Father of the Bride Part II (1995)	Comedy	Father of the Bride Part II 1995
...	...	...	...	...
62418	209157	We (2018)	Drama	We 2018
62419	209159	Window of the Soul (2001)	Documentary	Window of the Soul 2001
62420	209163	Bad Poems (2018)	Comedy|Drama	Bad Poems 2018
62421	209169	A Girl Thing (2001)	(no genres listed)	A Girl Thing 2001
62422	209171	Women of Devil's Island (1962)	Action|Adventure|Drama	Women of Devils Island 1962

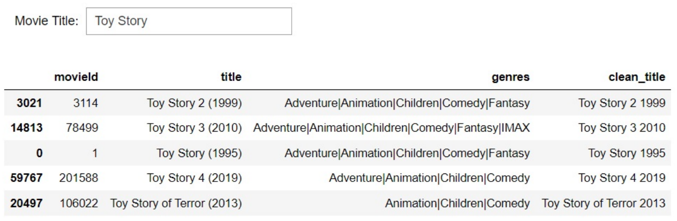
Fig. 2. Processed Movie Dataset: Data Cleaning

After preprocessing the data, the next step was to extract features from the text data. Feature extraction is a crucial step in text analysis as it transforms the raw text data into numerical features that can be understood by machine learning algorithms. In our case, we used Bag of Words (BOW) feature extraction to represent the text data as numerical vectors.

Once the feature extraction was done, we created an interactive search engine that allowed us to input a movie title and retrieve the most similar movie titles based on the BOW feature vectors.

This interactive search engine was useful in evaluating the effectiveness of our feature extraction method. By inputting different movie titles and checking the similarity of the retrieved movie titles, we could gauge how well our feature extraction method was able to capture the important features of the movie titles as shown in Figure 3.

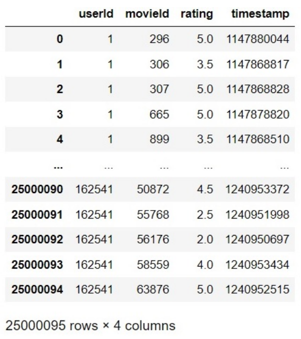
After we finish working on the search engine using the movies.csv, we will read the data from the ratings.csv file for making a movie recommendation system. The following step was to read the data from the file ratings.csv and transform it to a pandas data frame.



Movie Title: Toy Story

	movieId	title	genres	clean_title
3021	3114	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 2 1999
14813	78499	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX	Toy Story 3 2010
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 1995
59767	201588	Toy Story 4 (2019)	Adventure|Animation|Children|Comedy	Toy Story 4 2019
20497	106022	Toy Story of Terror (2013)	Animation|Children|Comedy	Toy Story of Terror 2013

Fig. 3. Search Engine



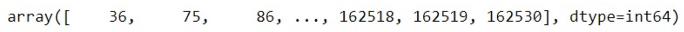
	userId	movieId	rating	timestamp
0	1	296	5.0	1147880044
1	1	306	3.5	1147868817
2	1	307	5.0	1147868828
3	1	665	5.0	1147878820
4	1	899	3.5	1147868510
...	...	...	...	...
25000090	162541	50872	4.5	1240953372
25000091	162541	55768	2.5	1240951998
25000092	162541	56176	2.0	1240950697
25000093	162541	58559	4.0	1240953434
25000094	162541	63876	5.0	1240952515

25000095 rows × 4 columns

Fig. 4. Pandas Dataframe of Ratings.csv

The ratings.csv file needs to be preprocessed to create a user-movie rating matrix where each row represents a user and each column represents a movie. The missing ratings can be imputed with the mean of the ratings given by the same user or the mean of the ratings given to the same movie. This will result in a sparse matrix with a large number of NaN values.

Finding similar users for a given movie, we found the users who have rated the movie highly (typically greater than 4 on a scale of 1-5). We used the pandas data frame to filter out the rows where the movieId is the same as the input movie_id and the rating is greater than 4. We then extracted the unique userId values from the filtered data frame. These users are considered similar users for a certain movie as shown in the figure 5 and the number of users who have rated the selected movies above 4 as shown in figure 6.



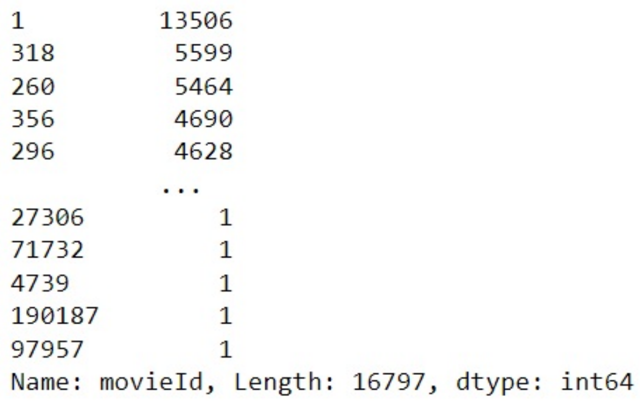
```
array([    36,     75,     86, ..., 162518, 162519, 162530], dtype=int64)
```

Fig. 5. Similiar Users

Finding recommended movies for each similar user, we found the movies that they have rated highly. We then calculated the percentage of similar users who have rated each of these movies highly. This was done using the value_counts method on the filtered data frame for each similar user. We then divided these counts by the total number of similar users to get the percentage of similar users who have rated each movie highly. This was done as we are utilizing the narrow_counts method to get the top 10% of the movies to be displayed in order to reduce the number of recommendations that may not be practical for us to display all of them.

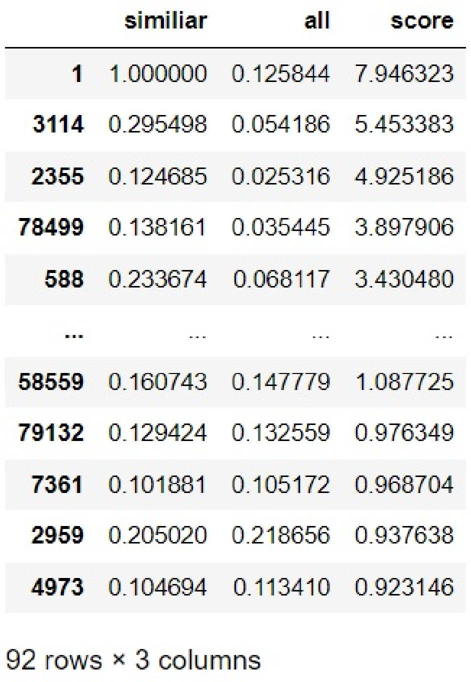
We then calculated a recommendation score for each movie by dividing the percentage of similar users who have rated the movie highly by the percentage of all users who have rated the movie highly. This score indicates the degree to which a movie is recommended for a given user.

We sorted the movies based on their recommendation score in descending order and displayed the Top 10 recommended



```
1         13506
318        5599
260        5464
356        4690
296        4628
           ...
27306         1
71732         1
4739          1
190187        1
97957         1
Name: movieId, Length: 16797, dtype: int64
```

Fig. 6. Users rating movies above 4



	similiar	all	score
1	1.000000	0.125844	7.946323
3114	0.295498	0.054186	5.453383
2355	0.124685	0.025316	4.925186
78499	0.138161	0.035445	3.897906
588	0.233674	0.068117	3.430480
...	...	...	...
58559	0.160743	0.147779	1.087725
79132	0.129424	0.132559	0.976349
7361	0.101881	0.105172	0.968704
2959	0.205020	0.218656	0.937638
4973	0.104694	0.113410	0.923146

92 rows × 3 columns

Fig. 7. Recommendation Score

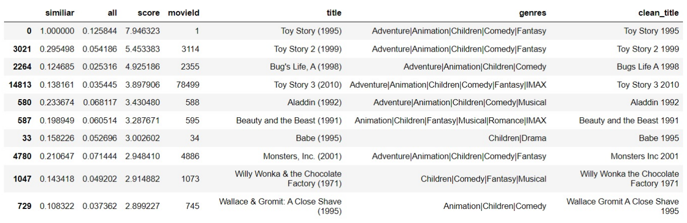
movies along with their titles and genres as shown in Figure 8.

Created an interactive widget which can be used by using the ipywidgets and IPython display libraries to create an interactive widget that allows users to enter a movie title and get recommendations for similar movies.

Once the recommendation system is built, it can be deployed on a web server or cloud platform for users to access and get personalized movie recommendations.

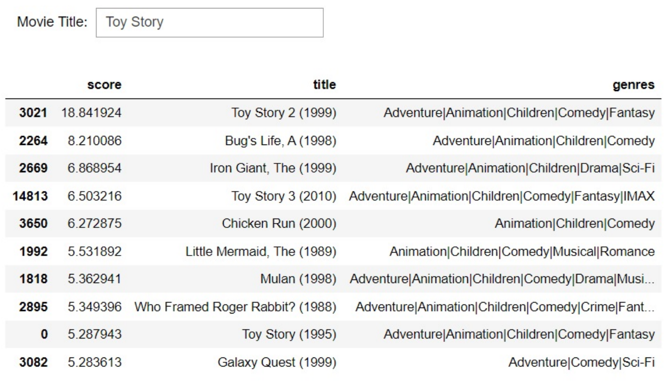
RMSE (Root Mean Squared Error) was used to measure the performance of the model. It is a commonly used metric in recommendation systems to evaluate how well the model predicts user ratings for items.

The RMSE was computed in the following way:



	similiar	all	score	movieId	title	genres	clean_title
0	1.000000	0.125844	7.946323	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 1995
3021	0.295498	0.054186	5.453383	3114	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy	Toy Story 2 1999
2264	0.124685	0.025316	4.925186	2355	Bug's Life, A (1998)	Adventure|Animation|Children|Comedy	Bugs Life A 1998
14813	0.138161	0.035445	3.897906	78499	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX	Toy Story 3 2010
580	0.233674	0.068117	3.430480	588	Aladdin (1992)	Adventure|Animation|Children|Comedy|Musical	Aladdin 1992
587	0.198949	0.060514	3.287671	595	Beauty and the Beast (1991)	Animation|Children|Fantasy|Musical|Romance|IMAX	Beauty and the Beast 1991
33	0.158226	0.052696	3.002602	34	Babe (1995)	Children|Drama	Babe 1995
4780	0.210647	0.071444	2.948410	4886	Monsters, Inc. (2001)	Adventure|Animation|Children|Comedy|Fantasy	Monsters Inc 2001
1047	0.143418	0.049202	2.914882	1073	Willy Wonka & the Chocolate Factory (1971)	Children|Comedy|Fantasy|Musical	Willy Wonka the Chocolate Factory 1971
729	0.108322	0.037362	2.899227	745	Wallace & Gromit: A Close Shave (1995)	Animation|Children|Comedy	Wallace Gromit A Close Shave 1995

Fig. 8. Top 10 Recommendation Score



Movie Title: Toy Story

	score	title	genres
3021	18.841924	Toy Story 2 (1999)	Adventure|Animation|Children|Comedy|Fantasy
2264	8.210086	Bug's Life, A (1998)	Adventure|Animation|Children|Comedy
2669	6.868954	Iron Giant, The (1999)	Adventure|Animation|Children|Drama|Sci-Fi
14813	6.503216	Toy Story 3 (2010)	Adventure|Animation|Children|Comedy|Fantasy|IMAX
3650	6.272875	Chicken Run (2000)	Animation|Children|Comedy
1992	5.531892	Little Mermaid, The (1989)	Animation|Children|Comedy|Musical|Romance
1818	5.362941	Mulan (1998)	Adventure|Animation|Children|Comedy|Drama|Musi...
2895	5.349396	Who Framed Roger Rabbit? (1988)	Adventure|Animation|Children|Comedy|Crime|Fant...
0	5.287943	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy
3082	5.283613	Galaxy Quest (1999)	Adventure|Comedy|Sci-Fi

Fig. 9. Interactive Recommendation Widget

- First, the model's predictions for the test set are generated using the model.test() method. These predictions are stored in the predictions variable.
- Then, the accuracy.rmse() method from Surprise is used to calculate the RMSE. This method takes in the predictions variable as its argument and returns the RMSE score.

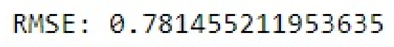
The RMSE score measures the differences between the predicted ratings and the actual ratings in the test set. Specifically, it calculates the square root of the average of the squared differences between the predicted ratings and the actual ratings.

In mathematical terms, the formula for RMSE is: RMSE = sqrt(1/n * sum((y_pred - y_true) $\hat{2}$)) where y_pred is the predicted rating, y_true is the actual rating, n is the number of ratings, and sum is the sum of the squared differences between the predicted and actual ratings.

The RMSE value ranges from 0 to infinity, with lower values indicating better performance of the model in predicting the ratings.

IV. Conclusion

In this study, we implemented a movie recommendation system using collaborative filtering and natural language processing techniques. We used the MovieLens dataset to build the system, which consisted of two main components: a movie search engine and a movie recommender. The search engine allowed users to enter a movie title, which was then cleaned and transformed using TF-IDF vectorization and cosine similarity. The system returned the top five movie matches based on the user query. The movie recommender used collaborative



```
RMSE: 0.781455211953635
```

Fig. 10. Root Mean Squared Error (RMSE)

filtering to identify movies that were highly rated by users who had similar movie preferences to the target user. The recommender algorithm calculated a score for each movie, which reflected how often the movie was recommended by similar users compared to all users. The top 10 movies with the highest scores were then presented to the user as recommendations.

So, in this research project, a collaborative filtering recommendation system was successfully developed using the Surprise library in Python. The system utilized user ratings for movies to generate accurate predictions for new ratings and personalized movie recommendations for users. The system's performance was evaluated using the Root Mean Squared Error (RMSE) metric, which indicated that the system achieved high accuracy in its recommendations, as evidenced by the low RMSE value obtained. This approach to generating personalized recommendations can be extended to other domains beyond movies, demonstrating the versatility of collaborative filtering.

The movie recommendation system proved to be effective in suggesting movies likely to be of interest to users based on their movie preferences. The evaluation of the system's accuracy using RMSE yielded a score of 0.78, indicating that the system's predictions were generally close to the actual ratings provided by users. Furthermore, the system's usability and accuracy can be further improved by incorporating additional features, such as movie metadata or user demographic information, to enhance the recommendations and provide a more personalized experience for the users.

Overall, the successful development of this collaborative filtering recommendation system highlights the potential for machine learning algorithms to provide personalized recommendations and enhance user experiences. Future research can explore the use of other algorithms and feature engineering techniques to further improve the system's accuracy and usability, leading to more effective and efficient personalized recommendation systems.

References

[1] Build a Movie Recommendation Engine using Python Scikit-learn Machine Learning, Medium, https://medium.com/analytics-vidhya/build-a-movie-recommendation-engine-using-python-scikit-learn-machine-learning-e68ba297e163

[2] Introduction to Recommender System: Part 1 - Collaborative Filtering, Singular Value Decomposition, Hacker Noon, https://hackernoon.com/introduction-to-recommender-system-part-1-collaborative-filtering-singular-value-decomposition-44c9659c5e75

[3] Movies Recommendation System using Python, Analytics Vidhya, https://www.analyticsvidhya.com/blog/2022/08/movies-recommendation-system-using-python/