Table of Contents

1.0 Project Background

1.1 Background of the problem domain

With the continuous development of information technology, people are faced with more and more choices in daily life, such as movies, music, books, and other entertainment fields. To help users better find content that meets their interests, and preferences, recommendation systems have emerged as the times require[1]. The recommendation system provides users with personalized recommendation content by analyzing users' historical behaviors, interests and preferences, thereby improving user experience and platform stickiness.

Collaborative filtering is a commonly used method in recommendation systems, collaborative filtering techniques encompass two key methods: memory-based filtering and model-based filtering. memory-based filtering methods rely on historical behavioral data of users or objects and make recommendations by directly utilizing these data. However, the effect of memory methods can be limited when the data is sparse[2]. In contrast, the model-based filtering method can better deal with sparsity and cold start problems by building a mathematical model to capture the complex relationship between users and items.

Therefore, collaborative filtering combining memory-based and model-based becomes an attractive method. By combining memory and model, this method overcomes its limitations and improves the performance of the recommendation system. In the movie recommendation scene, the user's historical movie ratings can be used as part of memory, and the model can learn the user's preferences for different genres, actors, directors, and other factors, to provide more personalized recommendations.

1.2 Description of the problem

The core issue of this study is identifying the most accurate approach for recommending movies. Using collaborative filtering techniques, such as memory-based and model-based filtering, the study aims to assess the efficacy of each method in predicting user preferences accurately. Finding the most precise method is crucial for enhancing recommendation systems and boosting user satisfaction and engagement with movie watching.

1.3 Objective

In the paper, I will use recommendation systems applied to movie ratings data, which can help people make the right decisions when they want to watch movies of the same genre or don't know what to watch. Then this interactive system recommends movies to the user based on factors such as rating, genre, etc.
The core objective of my thesis revolves around the implementation of memory-based collaborative filtering techniques for recommendation systems. This encompasses the utilization of user-based, item-based, and content-based approaches. Through these methods, I seek to recommend products to users based on the preferences of similar users, predict analogous items based on users' ratings, and harness content features to enhance the recommendation process.
By studying the application of collaborative filtering based on memory and model in movie recommendation, we can further improve the accuracy of the recommendation system and user satisfaction, improve user trust in the recommendation system, and promote the development of the movie industry.

2.0 Data Understanding & Integration

2.1 Data description

The Movies Dataset and Ratings Dataset for this paper come from MovieLens (http://www.movielens.org/) and Grouplens (https://grouplens.org/) respectively. The ratings

dataset contains approximately 3,908,657 anonymous ratings of 68,044 movies made by 6,724 MovieLens users who have logged in to the website over 12 times from 2019 to 2020 and rated over 20 movies since their registration. The movie dataset is comprised of 62423 movie names, and genres, and uses `movieId` to connect with the ratings dataset.

Table1: Ratings dataset description

Features	Data type	Description
userId	int64	the anonymized unique id for each active user, indexed from 1 to 6724.
movieId	int64	the id of the movie that the user (corresponding to userId) rated.
rating	float64	the rating (from 0.5 to 5 stars) provided by the user.
tstamp	object	when the user rated the movie.

Table2: Movies dataset description

Features	Data type	Description
movieId	int64	the id of the movie
title	object	Title of the movie
genres	object	Genres of the movie

2.2 Exploration Data Analysis

EDA is a must-do step in the data analysis process in order to explore the structure and laws of data. Its main work includes: cleaning the data, describing the data (describing statistics, charts), viewing the distribution of data, comparing the relationship between data, cultivating intuition, and summarizing the data.

In this part, I will use some common methods to explore and analyze data, the purpose of EDA is to get a better understanding of the dataset and provide the basis for subsequent analysis.

(1) Loading the dataset is the first step at the beginning of any analysis, Fig1. , Fig 2. below are the loading data parts of this experiment

```
ratings = pd.read_csv('ratings.csv')
ratings.head()
```

	userId	movieId	rating	tstamp
0	206	4803	4.0	2003-04-07 13:52:01
1	5073	72731	4.0	2020-02-19 16:07:53
2	4739	91653	4.0	2020-12-28 15:35:58
3	535	3005	3.0	2008-12-26 05:38:11
4	465	4776	3.0	2008-08-13 20:22:36

Fig1. Ratings Example Data

We can see from Fig1. that the ratings.csv file contains the `userId`, `movieId`, `ratings`, and timestamp attributes. Each row in the dataset corresponds to one rating, the `userId` column contains the ID of the user who left the rating, the `movieId` column contains the Id of the movie, the rating column contains the rating left by the user, ratings can have values between 1 and 5. Finally, the timestamp refers to the time at which the user left the rating.

```
movies = pd.read_csv('movies.csv')
movies.head()
```

	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

Fig2. Movies Example Data

From the Fig2. we can see that contains the `movieId`, `title`, and `genres` attributes. Each movie title corresponds to a unique Id, then we can use the `movieId` to connect the ratings dataset.

```
movie_data = pd.merge(ratings, movies, on='movieId')
movie_data.head()
```

	userId	movieId	rating	title	genres
0	206	4803	4.0	Play Misty for Me (1971)	[Drama, Thriller]
1	201	4803	3.0	Play Misty for Me (1971)	[Drama, Thriller]
2	3698	4803	3.0	Play Misty for Me (1971)	[Drama, Thriller]
3	104	4803	1.0	Play Misty for Me (1971)	[Drama, Thriller]
4	12	4803	4.0	Play Misty for Me (1971)	[Drama, Thriller]

Fig3. Merge Example Data

So now we get our desired information in a single data frame, through the `merge () ` function and use the `movieId` to connect, as shown in Fig3.

(2) There is no missing value in our datasets, and also don't have the duplicated data. As shown in Fig4.

```
# check if has missing value
print(ratings.isnull().sum())
print(movies.isnull().sum())
```

```
userId     0
movieId    0
rating     0
tstamp     0
dtype: int64
movieId    0
title      0
genres     0
dtype: int64
```

```
# check if has duplicated data
print(ratings[ratings.duplicated()])
print(movies[movies.duplicated()])
```

```
Empty DataFrame
Columns: [userId, movieId, rating, tstamp]
Index: []
Empty DataFrame
Columns: [movieId, title, genres]
Index: []
```

Fig4. Check the missing value and duplicated data

(3) In the ratings dataset, the `rating` mean value is close to 3.4 and the std approximately is 1.02, with `rating` values between 0.5 and 5 as the data description writes.

```
ratings.describe().T
```

	count	mean	std	min	25%	50%	75%	max
userId	3908657.0	2434.060851	1950.445219	1.0	727.0	1973.0	3858.0	6724.0
movieId	3908657.0	62535.529829	67051.572745	1.0	3478.0	45447.0	106002.0	270592.0
rating	3908657.0	3.419320	1.022044	0.5	3.0	3.5	4.0	5.0

Fig5. Descriptive Statistics

(4) Now let's take a look at the average rating of each movie, we can group the dataset by the title of the movie and then calculate the mean of the rating for each movie. Before we group the dataset, we need to filter which movies are rated less than 50 times, because if only a single user has given a movie five stars, then the above stars will mislead the analysis.

```
print('# movies: %d' % len(movie_data))
movie_ratings_count = movie_data.groupby('movieId').size()
# fiter ratings_counts more than 50
popular_movies = movie_ratings_count[movie_ratings_count >= 50].index
filtered_movie_data = movie_data[movie_data['movieId'].isin(popular_movies)]
print('# At least 50 movie ratingst: %d' % len(filtered_movie_data))
```

```
# movies: 3756004
# At least 50 movie ratingst: 3409871
```

Fig6. Filter Movies

```
# create a new dataframe to show the title, rating, rating_counts
ratings_mean = pd.DataFrame(filtered_movie_data.groupby('title')['rating'].mean())
ratings_mean['rating_counts'] = pd.DataFrame(filtered_movie_data.groupby('title')['rating'].count())
ratings_mean['rating_sum'] = pd.DataFrame(filtered_movie_data.groupby('title')['rating'].sum())
ratings_mean.sort_values(by='rating', ascending=False).head()
```

title	rating	rating_counts	rating_sum
Planet Earth II (2016)	4.472727	660	2952.0
Planet Earth (2006)	4.415816	980	4327.5
Band of Brothers (2001)	4.384065	979	4292.0
Human Condition III, The (Ningen no joken III) (1961)	4.372549	51	223.0
Shawshank Redemption, The (1994)	4.350253	4945	21512.0

Fig7. New Statistical Data Frame

From Fig7. We can see the average ratings are sorted, the count of movies, and the sum of ratings. The above list supports our point that good movies normally receive higher ratings. Now we know that both the average rating per movie and the number of ratings per movie are important attributes.

(5) Plot a histogram for average ratings,

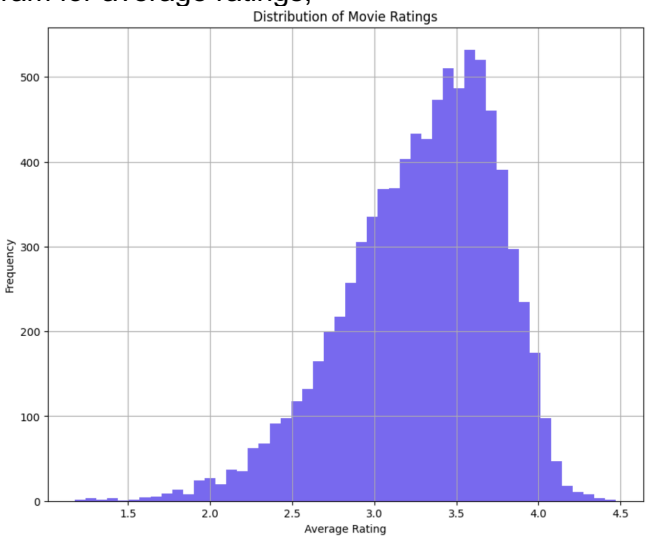


Fig8. Distribution of Movie Ratings

We can see from Fig8. That most ratings are in the range of 2.5 to 4, and it is evident that the data has a weak normal distribution with a mean of around 3.4, have a few outliers in the data as well.

(6) Usually, movies with a higher number of ratings have a high average rating as well since a good movie is normally well-known and a well-known movie is watched by a large number of people, and thus usually has a higher rating. As shown in Fig9. That plot average ratings against the number of ratings.

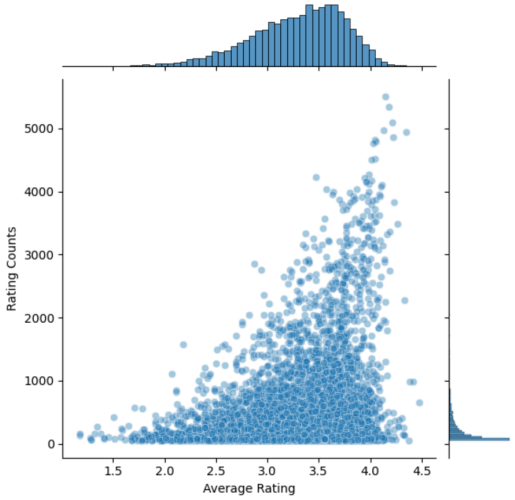


Fig9. Relationship between Average Rating and Rating Counts

The graph shows that, in general, movies with higher average ratings actually have a greater number of ratings, compared with movies that have lower average ratings.

(7) Fig10. Shows the top 10 highest-rated movies. This graph is sketched between the Movie Name and the Sum of Ratings by users online. The main purpose of this graph is to give an insight into which movie is the most loved across the globe.

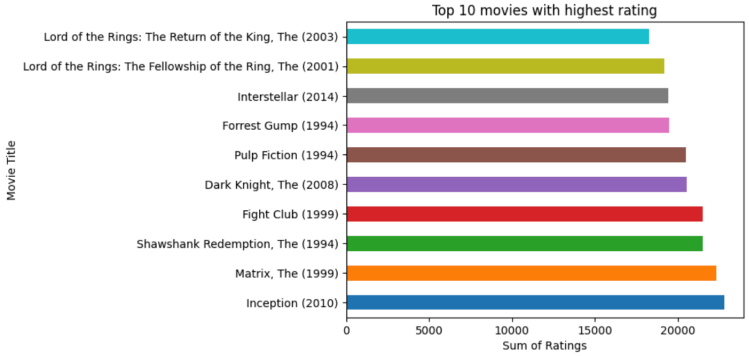


Fig10. Top 10 Highest Rated Movies

(8) Fig11. shows the top 10 Movies with the Highest Number of Ratings. This graph is sketched between the Movie Name and the Number of Reviews online. The main purpose of this graph is to give an insight into which movie is the most reviewed across the globe.

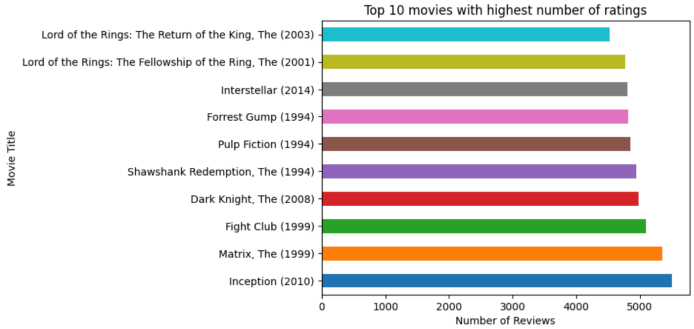


Fig11. Movies with the Highest Number of Ratings

(9) In our dataset, the ratings span from 0.5 to 5. To better analyze this data, we'll divide the ratings into five intervals: [0-1], [1-2], [2-3], [3-4], and [4-5]. This will allow us to visualize the proportion of ratings falling within each interval, providing insights into how users rate movies across different levels of satisfaction.

```
rating_group
1–2      13272
2–3     482329
3–4    2650244
4–5     264026
Name: rating_counts, dtype: int64
```

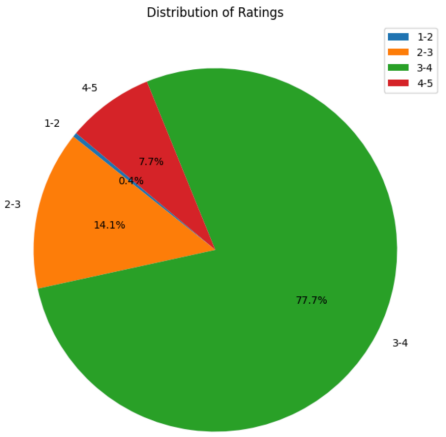
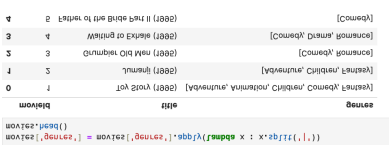


Fig12. Rating Proportion Distribution

(10) The more frequently a word appears in the text, the larger and more prominent it appears in the word cloud. From Fig14. We can see that the "War", "Musical", "Romance", and "IMAX" words are bigger than others. Before we create word clouds, we need to preprocess the data, as shown in Fig13. All genres become unique after processing.

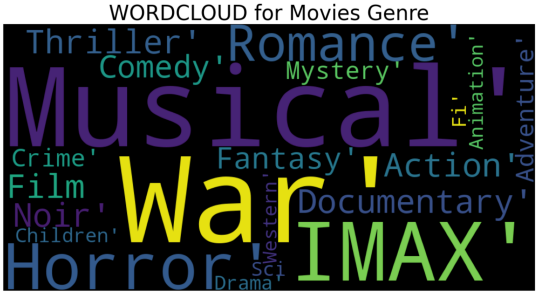


```
from wordcloud import WordCloud

# used for wordcloud graph
genres=list(set([genre for genre_list in movies['genres'] for genre in genre_list]))
# remove '(no genres listed)'
genres.remove('(no genres listed)')
movie_titles = [title[:-7] for title in movies['title']]
```

Fig13. Preprocess Data for Word cloud

WORDCLOUD for Movies Genre



WORDCLOUD for Movies title

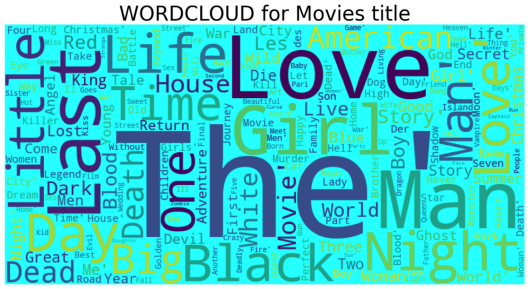


Fig14. Data for Movie Genre and Movie Title in Word clouds

3.0 Recommender Systems

A recommender system is a data science application that is used to predict or offer personalized products to customers based on their past purchases or browsing history and so on. It is based on the similarity based on the entities or users that previously rated those entities[6]. There are various recommendation systems techniques, as shown in Fig15.

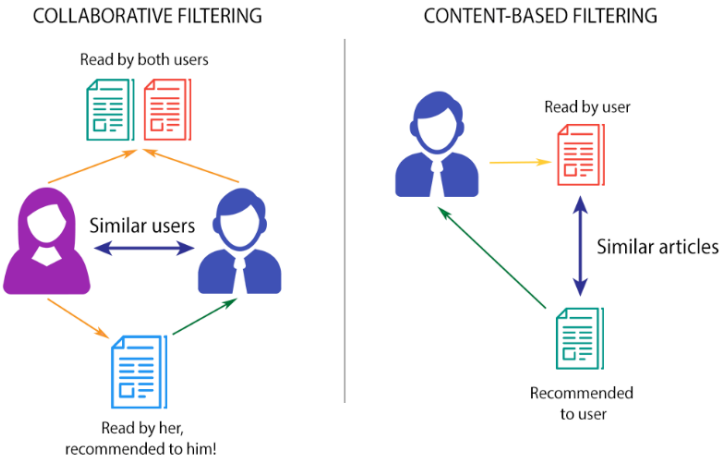


Fig15. Collaborative Filtering vs Content-based Filtering Techniques

3.1 Collaborative-filtering technique

Memory-based collaborative filtering and model-based collaborative filtering are two well-known approaches developed for recommendation systems. The main idea behind these approaches is to use other users' preferences and tastes to recommend new items to a user. The usual procedure is to find similar users (or items) to recommend new items which were liked by those users, and which presumably will also be liked by the user being recommended.
In this part, I will use 3 different approaches for collaborative filtering, each following their theories and mathematical formulas.

3.1.1 Memory-based filtering technique

The foundation of memory-based collaborative filtering, also known as neighborhood-based or user-item filtering, is the presumption that users who have historically displayed similar preferences will continue to do so in the future. This method takes into account the ratings of things or people that are close by, making the computation of item ratings simple.

Cosine similarity is a dot product of unit vectors.
Cosine similarity

$$similarity(A,B) = \frac{A \cdot B}{\|A\| * \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

User-based collaborative filtering
In this method, products are recommended to a user based on the fact that the products have been liked by users similar to the user.

```
user_similarity = cosine_similarity(user_data)
user_similarity[np.isnan(user_similarity)] = 0
print(user_similarity)
print('- '*10)
print(user_similarity.shape)
```

```
[[1.         0.2157621  0.12812291 ... 0.07759321 0.08962526 0.08148425]
 [0.2157621  1.         0.14705552 ... 0.06963207 0.1085028  0.11635261]
 [0.12812291 0.14705552 1.         ... 0.01904439 0.11938205 0.1145834 ]
 ...
 [0.07759321 0.06963207 0.01904439 ... 1.         0.10686266 0.06424515]
 [0.08962526 0.1085028  0.11938205 ... 0.10686266 1.         0.17617765]
 [0.08148425 0.11635261 0.1145834  ... 0.06424515 0.17617765 1.        ]]
- - - - - - - - - -
(6723, 6723)
```

```
# predict the movies'rating
user_predicted_ratings = np.dot(user_similarity, user_data)
print(user_predicted_ratings)
print(user_predicted_ratings.shape)
```

```
[[1166.45814134  572.28731801   94.49329617 ...   53.88288515
    30.63771285  245.67841067]
 [1407.81190424  708.7586942   103.28418416 ...   57.97951873
    32.59551856  278.75105886]
 [1286.84252583  628.28996906   87.01563524 ...   57.3760476
    26.03830111  253.22338033]
 ...
 [1065.97140269  505.2645594    53.95031522 ...   55.73851503
    18.334961    220.44951503]
 [1414.97011614  661.47031665   78.55089658 ...   71.00539287
    25.32802418  282.63505164]
 [1846.57853904  906.09325942   97.14037112 ...   81.51875908
    43.47644593  396.79730083]]
(6723, 8511)
```

```
user_final_ratings = np.multiply(user_predicted_ratings, dummy_train)
```

Fig16. User-based Codes

Item-based collaborative filtering
This method identifies and predicts similar items based on users' previous ratings.

```
item_similarity = cosine_similarity(movie_features)
item_similarity[np.isnan(item_similarity)] = 0
print(item_similarity)
print('- '*10)
print(item_similarity.shape)
```

```
[[1.         0.40389708 0.18043234 ... 0.1213453  0.08908057 0.22813075]
 [0.40389708 1.         0.14491844 ... 0.09372162 0.08293381 0.20881718]
 [0.18043234 0.14491844 1.         ... 0.05160476 0.05245923 0.06460393]
 ...
 [0.1213453  0.09372162 0.05160476 ... 1.         0.03625098 0.12030302]
 [0.08908057 0.08293381 0.05245923 ... 0.03625098 1.         0.08391244]
 [0.22813075 0.20881718 0.06460393 ... 0.12030302 0.08391244 1.        ]]
- - - - - - - - - -
(8511, 8511)
```

```
item_predicted_ratings = np.dot(movie_features.T, item_similarity)
print(item_predicted_ratings)
print(item_predicted_ratings.shape)
```

```
[[519.57797375 407.20600277 340.88058504 ... 143.09047535 133.14575585
  248.98614261]
 [262.3506296  219.55837844 148.58960396 ...  62.55333921  61.33191305
  120.55276964]
 [211.87352459 172.32989565 112.31748153 ...  54.509494    41.06718228
   95.31746188]
 ...
 [ 46.57714659  36.80397829  16.74500531 ...  13.36581744   7.58602756
   22.02572658]
 [134.53166414 103.91088678  53.64997792 ...  38.30660053  21.65832097
   60.54704978]
 [219.61143062 180.46270993  80.38577367 ...  54.53296937  47.39436894
  108.28991436]]
(6723, 8511)
```

```
item_final_ratings = np.multiply(item_predicted_ratings, dummy_train)
```

Fig17.Item-based Codes

3.1.2 Model-based filtering technique

Collaborative Modeling Filters use a statistical or machine learning model to find and take advantage of hidden links and patterns in the data rather than a predefined set of rules. Using training data from previous user-item interactions, these models are then used to estimate users' preferences for unseen objects.

Matrix Factorization via Singular Value Decomposition
Matrix factorization is an approximation of a matrix into a product of matrices[4]. In this paper, we use one of the Eigenvalue-based decomposition methods, named Singular Value Decomposition (SVD) which factorizes any matrix with any dimension as 3 parts USV.

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

In this case, A is the user ratings matrix, U is the user "features" matrix, Σ is the diagonal matrix of singular values (essentially weights), and V^T is the movie "features" matrix. U and V^T are orthogonal and represent different things. U represents how much users "like" each feature and V^T represents how relevant each feature is to each movie. As shown in Fig18.

```
R_sparse = scipy.sparse.csr_matrix(R_demean)
U, sigma, Vt = svds(R_sparse, k=50)
sigma = np.diag(sigma)
```

```
all_user_predicted_ratings = np.dot(np.dot(U, sigma), Vt) + user_ratings_mean.reshape(-1, 1)
preds_df = pd.DataFrame(all_user_predicted_ratings, columns = user_data.columns)
```

```
def recommend_movies(predictions_df, userId, movies_df, original_ratings_df,num_recommendations=5):
    user_row_number = userId -1
    sorted_user_predictions = predictions_df.iloc[user_row_number].sort_values(ascending=False)
    # Get the user's data and merge in the movie information.
    # user_full = df[df.userId == (userId)].sort_values(['rating'], ascending=False)
    user_data = original_ratings_df[original_ratings_df.userId == (userId)]
    user_full = (user_data.merge(movies_df, how = 'left', left_on = 'movieId', right_on = 'movieId')
                 .sort_values(['rating'], ascending=False))
    print('Recommending UserId={0} the highest {1} predicted ratings movies not already rated.'
          .format(userId, num_recommendations))
    # Recommend the highest predicted rating movies that the user hasn't seen yet.
    recommendations = (movies_df[~movies_df['movieId'].isin(user_full['movieId'])].
         merge(pd.DataFrame(sorted_user_predictions).reset_index(), how = 'left',left_on = 'movieId',right_on = 'movieId').
         rename(columns = {user_row_number: 'Predictions'}).
         sort_values('Predictions', ascending = False).iloc[:num_recommendations, :-1])

    return user_full, recommendations
```

Fig18. Model-based Code

3.2 Content-based filtering technique

In the part, where use TF-IDF to do this approach, TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents, which is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents[5].

To put it in more formal mathematical terms, the TF-IDF score for the word t in document d from the document set D is calculated as follows:

$$tf\ idf(t,d,D) = tf(t,d)\ .idf(t,D)$$

Where:

$$tf(t,d) = \log(1 + freq(t,d))$$

$$idf(t,D) = \log\left(\frac{N}{count(d \in D: t \in d)}\right)$$

idf function ensures lower weight to more common words and higher rate to rarer words.

```
genre_popularity = (movies_.genres.str.split('|')
                    .explode()
                    .value_counts()
                    .sort_values(ascending=False))
genre_popularity.head(10)
```

```
genres
Drama                25606
Comedy               16870
Thriller              8654
Romance               7719
Action                7348
Horror                5989
Documentary           5605
Crime                 5319
(no genres listed)    5062
Adventure             4145
Name: count, dtype: int64
```

```
tfidf_genres = TfidfVectorizer(analyzer=lambda s: (c for i in range(1,4)
                                                   for c in combinations(s.split('|'), r=i)))

tfidf_matrix = tfidf_genres.fit_transform(movies_['genres'])

print(tfidf_matrix.shape)
```

```
(62423, 897)
```

```
cosine_sim_movies = cosine_similarity(tfidf_matrix)

cosine_sim_df = pd.DataFrame(cosine_sim_movies, index=movies_['title'], columns=movies_['title'])
print('Shape:', cosine_sim_df.shape)
cosine_sim_df.sample(5, axis=1).round(2)
```

Shape: (62423, 62423)

title	Cairo (1963)	Zotz! (1962)	The Pace That Kills (The Cocaine Fiends) (1935)	Misérables, Les (1998)	Brief Encounter (1974)
title					
Toy Story (1995)	0.00	0.13	0.00	0.00	0.00
Jumanji (1995)	0.00	0.09	0.00	0.00	0.00
Grumpier Old Men (1995)	0.00	0.07	0.00	0.08	0.00
Waiting to Exhale (1995)	0.07	0.04	0.21	0.12	0.21
Father of the Bride Part II (1995)	0.00	0.17	0.00	0.00	0.00
...	...	...	...	...	...
We (2018)	0.34	0.00	1.00	0.08	1.00
Window of the Soul (2001)	0.00	0.00	0.00	0.00	0.00
Bad Poems (2018)	0.14	0.08	0.40	0.03	0.40
A Girl Thing (2001)	0.00	0.00	0.00	0.00	0.00
Women of Devil's Island (1962)	0.06	0.00	0.17	0.01	0.17

62423 rows × 5 columns

Fig19. Content-based Codes

The genres can be used to provide a reasonably good content-based recommendation, but before that, we need to analyze some important aspects. Most popular genres will be a relevant aspect to take into account when building the content-based recommender. We want to understand which genres really are relevant when it comes to defining a user's taste. Based on the movie genres, there will be a common approach TF-IDF which captures the important genres of each movie by giving a higher weight to the less frequent genres.

4.0 Discussion & Analysis

In this section, I will analyze the results of each approach as well as their respective advantages, disadvantages, and performance to better select the best approach.

4.1 Results of Collaborative Filtering Methods

4.1.1 Analysis of Memory-based Filtering Results

User-based

```
# for instance
# recommend top5 movies to userId=10
user_recommend_movies = user_final_ratings.iloc[10].sort_values(ascending = False).head()
top_movies = movies[movies['movieId'].isin(user_recommend_movies.index)][['movieId', 'title']]
print('Top 5 movie recommendations for User 10:')

for movie_id, movie_title in zip(top_movies['movieId'], top_movies['title']):
    print(f"Movie ID: {movie_id}, Title: {movie_title}")
```

```
Top 5 movie recommendations for User 10:
Movie ID: 2959, Title: Fight Club (1999)
Movie ID: 4993, Title: Lord of the Rings: The Fellowship of the Ring, The (2001)
Movie ID: 7153, Title: Lord of the Rings: The Return of the King, The (2003)
Movie ID: 60069, Title: WALL·E (2008)
Movie ID: 79132, Title: Inception (2010)
```

Fig20. The Result of User-based

In this case, we use an example in which user ID=10, then use a user-based filtering technique to recommend top5 movies to the user, which does not include movies that the user has already seen.

Item-based

```
# for instance
# recommend top5 movies to userId=10
item_recommend_movies = item_final_ratings.iloc[10].sort_values(ascending = False).head()
top_movies_item = movies[movies['movieId'].isin(item_recommend_movies.index)][['movieId', 'title']]
print('Top 5 movie recommendations for User 10:')

for movie_id, movie_title in zip(top_movies_item['movieId'], top_movies_item['title']):
    print(f"Movie ID: {movie_id}, Title: {movie_title}")
```

```
Top 5 movie recommendations for User 10:
Movie ID: 1210, Title: Star Wars: Episode VI - Return of the Jedi (1983)
Movie ID: 2716, Title: Ghostbusters (a.k.a. Ghost Busters) (1984)
Movie ID: 4993, Title: Lord of the Rings: The Fellowship of the Ring, The (2001)
Movie ID: 6377, Title: Finding Nemo (2003)
Movie ID: 33794, Title: Batman Begins (2005)
```

Fig21. The Result of Item-based

Compared with the User-based one, this is an Item-based analysis. The examples are all the same user ID = 10, but the recommended movies will be different.

4.1.2 Analysis of Model-based Filtering Results

```
already_rated, predictions = recommend_movies(preds_df,10,movies,ratings)
predictions
```

```
Recommending UserId=10 the highest 5 predicted ratings movies not already rated.
```

	movieId	title	genres
8813	60069	WALL·E (2008)	[Adventure, Animation, Children, Romance, Sci-Fi]
11355	80906	Inside Job (2010)	[Documentary]
7503	45186	Mission: Impossible III (2006)	[Action, Adventure, Thriller]
1351	2474	Color of Money, The (1986)	[Drama]
1682	3037	Little Big Man (1970)	[Western]

Fig22. The Result of Model-based

In the method, we use the model-based filtering technique to recommend movies to users, the same user ID=10, but we can see from Fig22. Most films are different from those

recommended by the first two methods, this movie "WALL · E (2008)" is the one with the highest prediction scores for recommended user ID = 10.
though we didn't actually use the genre of the movie as a feature, the truncated matrix factorization features "picked up" on the underlying tastes and preferences of the user, and then the system recommended some film noirs, crime, drama, and war movies - all of which were genres of some of this user's top-rated movies.

4.2 Analysis of Content-based Filtering Results

```
def get_recommendations_based_on_genres(movie_title, cosine_sim_movies=cosine_sim_movies):
    '''
    Calculates top 10 movies to recommend based on given movie titles genres
    '''
    # Get the index of the movie that matches the title
    idx_movie = movies.loc[movies['title'].isin([movie_title])]
    idx_movie = idx_movie.index
    sim_scores_movies = list(enumerate(cosine_sim_movies[idx_movie][0]))
    sim_scores_movies = sorted(sim_scores_movies, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores_movies = sim_scores_movies[1:11]
    movie_indices = [i[0] for i in sim_scores_movies]
    return pd.DataFrame(movies[['title','genres']].iloc[movie_indices])
```

```
get_recommendations_based_on_genres('Pulp Fiction (1994)')
```

	title	genres
600	Fargo (1996)	[Comedy, Crime, Drama, Thriller]
1011	Freeway (1996)	[Comedy, Crime, Drama, Thriller]
3173	Man Bites Dog (C'est arrivé près de chez vous)...	[Comedy, Crime, Drama, Thriller]
4138	Beautiful Creatures (2000)	[Comedy, Crime, Drama, Thriller]
5891	Confessions of a Dangerous Mind (2002)	[Comedy, Crime, Drama, Thriller]
6363	Hard Word, The (2002)	[Comedy, Crime, Drama, Thriller]
6582	Party Monster (2003)	[Comedy, Crime, Drama, Thriller]
9115	Freeway II: Confessions of a Trickbaby (1999)	[Comedy, Crime, Drama, Thriller]
10561	Cul-de-sac (1966)	[Comedy, Crime, Drama, Thriller]
11434	Body Count (1998)	[Comedy, Crime, Drama, Thriller]

Fig23. The Result of Content-based

As we can see from Fig23, the test movie is named A, and then the 10 highest-scoring movies are recommended, all of which are based on genre recommendations.

4.3 Advantages and Disadvantages

Advantages
The advantage of memory-based collaborative filtering is easy to handle medium-sized datasets and intuitive, easy to understand for transparency, as they are based on direct user-project interaction. Furthermore, it can offer serendipitous recommendations, as users may discover previously unknown but intriguing content through shared relationships with others.
Using a model-based filtering technique is good at solving big and sparse datasets as they learn underlying patterns without making direct comparisons of users or things, and this method is more flexible, can handle a variety of data, and has a certain mitigation effect on cold start.
Using a content-based filtering technique has several advantages, it doesn't rely on data from other users, avoids cold-start and sparsity issues, and also can provide explanations for recommended items by listing content features that caused an item to be recommended (in this case, movie genres).

Disadvantages
One of the limitations of memory-based collaborative filtering is that it becomes more challenging to identify neighbors as the data set grows and user-item interaction frequency declines. The other is a cold start problem which there is no historical data to understand the

preferences of new users. For complex data sets and features, the key features cannot be captured well, which will result in bad prediction results.
The drawbacks of the model-based approach are that the complexity of the data set will cause the model to become very complex and time-consuming. The training model may overfit and other problems, high accuracy is possible with model-based filtering, but there will be a black box case because the interior of the model is not easy to understand.
Using a content-based filtering technique is hard to find appropriate features and is unable to exploit the quality judgments of other users.

4.4 Comparative Performance Analysis

In this part, I'm going to analyze the performance of these approaches, below are three visuals illustrating comparisons: one displaying RMSE, another MAE, and the last showcasing execution time across different recommendation methods.

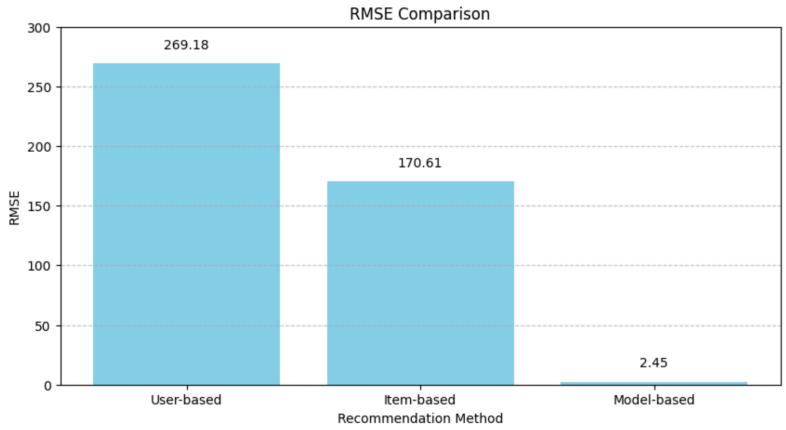


Fig24. RMSE Comparison

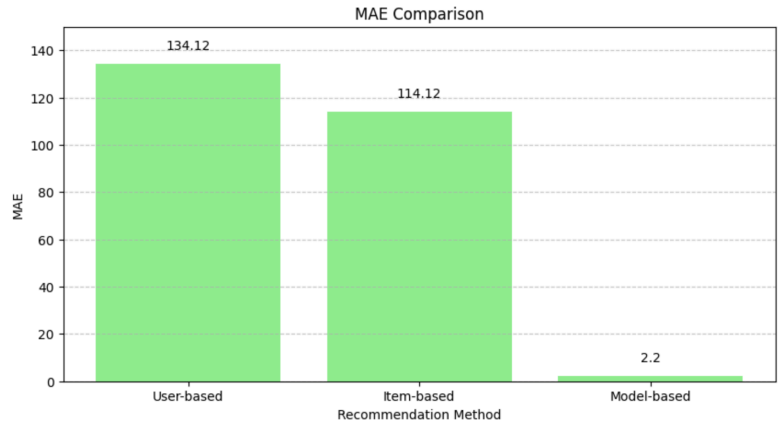


Fig25. MAE Comparison

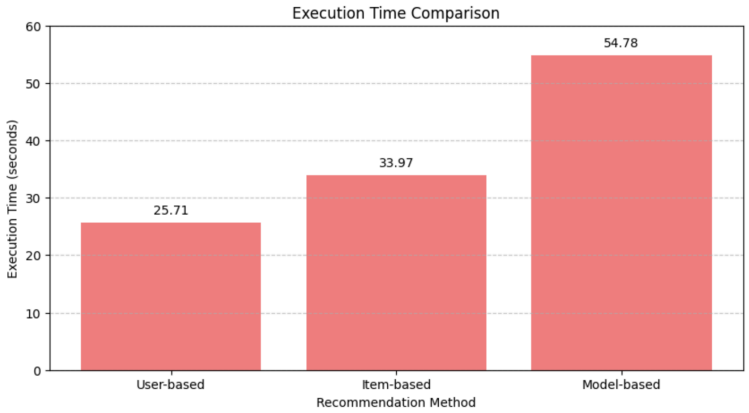


Fig26. Execution Time Comparison

From Fig24., Fig25 we can see that clearly demonstrate the superior performance of the model-based approach compared to the others, comparing user-based and item-based approaches, item-based performance is slightly better from Fig24 and Fig25, but both are memory-based approaches, so there is a big difference between them. From Figure 26, it's evident that the model-based approach requires more time for execution compared to the user-based and item-based approaches, which are relatively faster.

In a shut, I prefer to use a model-based approach due to its superior accuracy demonstrated by lower RMSE and MAE values. Despite its longer execution time and potentially less interpretable inner workings, the model-based method's ability to leverage complex algorithms often leads to more effective and personalized recommendations, aligning well with my system goals.

5.0 Conclusion

This experiment employs the collaborative filtering method based on users, items, models, and content along with the score data of the movie provided by users to enhance the accuracy and personalization of the movie recommendation system.

This is an overview of the study:

User-based collaborative filtering: In the experiment, the user-based collaborative filtering method is applied by examining user similarities. Utilizing user history movie ratings, the algorithm identifies user groups with comparable rating trends and suggests films that these comparable users enjoy to the intended audience. According to experimental findings, user-based collaborative filtering can sometimes significantly increase the recommendation's accuracy.

Item-based collaborative filtering: The item-based collaborative filtering approach is also used in this experiment. Users are able to suggest other films that are comparable to their previous favorite films by comparing the similarities between them. This approach can increase the universality of advice while overcoming any individual disparities in the user-based strategy. According to experimental findings, item-based collaborative filtering performs better when making recommendations for certain users.

Model-based collaborative filtering: To further improve the performance of the recommendation system, the model-based collaborative filtering method is introduced in this experiment. By establishing a mathematical model to capture the complex relationship between the user and the movie, the method can better deal with data sparsity and cold start problems. Experimental results show that model-based collaborative filtering can provide more accurate and personalized recommendations in many cases.

The content-based filtering method uses movie genre information and other attributes to tailor recommendations for each user. By analyzing a movie's characteristics and finding similarities, the system can suggest relevant movies that match the user's past preferences. This approach avoids the cold-start problem of new items by using feature extraction to ensure recommendations are highly relevant.

In the future, recommendation systems will likely merge collaborative and content-based filtering to provide even more personalized suggestions. With advances in machine learning, especially deep learning, they'll be able to identify subtle user preferences, leading to even better recommendations.

Reference

[1] Fayyaz, Z., Ebrahimian, M., Nawara, D., Ibrahim, A., & Kashef, R. (2020). Recommendation systems: algorithms, challenges, metrics, and business opportunities. Applied Sciences, 10(21), 7748. https://doi.org/10.3390/app10217748.

[2] Sun, Ruixuan & Kong, Ruoyan & Jin, Qiao & Konstan, Joseph. (2023). Less Can Be More: Exploring Population Rating Dispositions with Partitioned Models in Recommender Systems. 291-295. 10.1145/3563359.3597390.

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

[4] Liú, D., & Li, H. (2022). A matrix decomposition model based on feature factors in movie recommendation system. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2206.05654.

[5] Reddy, S. K., Nalluri, S., Kunisetti, S., Sharmila, A., & Venkatesh, B. (2018). Content-Based Movie Recommendation System using genre correlation. In Smart innovation, systems and technologies (pp. 391–397). https://doi.org/10.1007/978-981-13-1927-3_42.

[6] Gupta, M., Thakkar, A., Aashish, Gupta, V., & Rathore, D. P. S. (2020). Movie recommender System using collaborative filtering. 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC). https://doi.org/10.1109/icesc48915.2020.9155879.