INDIAN INSTITUTE OF TECHNOLOGY, KHARAGPUR

BIG DATA ANALYSIS COURSE PROJECT

MOVIE RECOMMENDATION SYSTEM

Nidhi Goyal
20MA20072

Gunjan Agarwal
20MA20069

Pranshul Gupta
20MA20044

Mohit Agarwal
20MA20075

Mohd. Arsalan
20MA20034

Nabhya Khoria
20MA20035

Yash Kunwar
20MA20065

Rajiv Harlalka
20MA20073

Karan Ralhan
20MA20025

Akshat Jain
20MA20005

Supervised by
Prof. Bibhas Adhikari

11 April, 2023

1 Problem Statement and Abstract

In recent years, the amount of available data on movies has grown exponentially, and this presents an opportunity to develop more accurate and effective movie recommendation systems. However, the sheer volume of data can make it challenging to identify the most relevant features and patterns in the data. Additionally, traditional recommendation systems often struggle to provide **personalized** recommendations that reflect the unique preferences of individual users. This project aims to address these challenges by leveraging **big data analysis** techniques to develop a movie recommendation system that provides more accurate and personalized recommendations.

The project will explore various machine learning algorithms to identify the most effective approach for generating relevant and engaging recommendations for users. The purpose of this project is to develop a movie recommendation system using big data analysis techniques combining various methods that have been implemented in production by various companies. The project will leverage data from multiple sources, including user ratings and movie metadata, to create a personalized recommendation system. The project will also incorporate natural language processing (NLP) techniques to analyse user reviews and identify key features of movies that are most relevant to individual users. The project will involve implementing and evaluating **several** machine learning algorithms to identify the most effective method for generating accurate and relevant movie recommendations.

2 Data

2.1 Data Collection

In this project, our objective was to perform a content-based analysis of movies using big data techniques. To achieve this, we first attempted to scrape data from the TMDB (The Movie Database) website. However, the data we obtained was not useful as it was not in a structured format and lacked the required information.

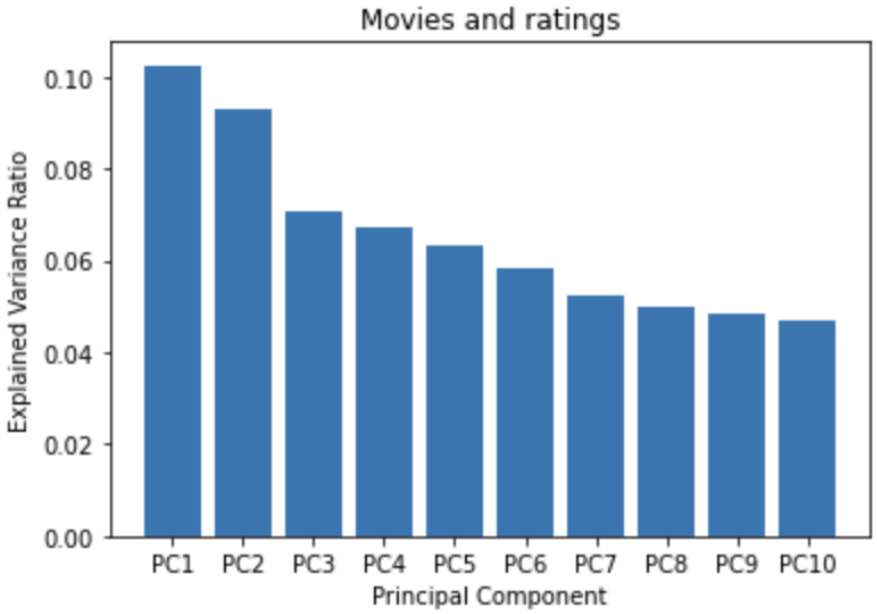
We then decided to use the Mindlens 100K dataset for our analysis. This dataset is available from the University of Minnesota and is considered trustworthy and reliable. It contains information about movies and demographic data for users, which is necessary for our content-based analysis.

The Mindlens 100K dataset contains 100,000 ratings from 943 users on 1682 movies. Each user has rated at least 20 movies, and the dataset provides information on each movie, including the title, cast, release date, and genre. This dataset allowed us to perform a detailed analysis of movie preferences based on user demographics and movie attributes.

2.2 Data Analysis

We began our analysis by exploring the dataset and visualizing the relationships between different variables. We found that there was a strong correlation between movie genres and user preferences.

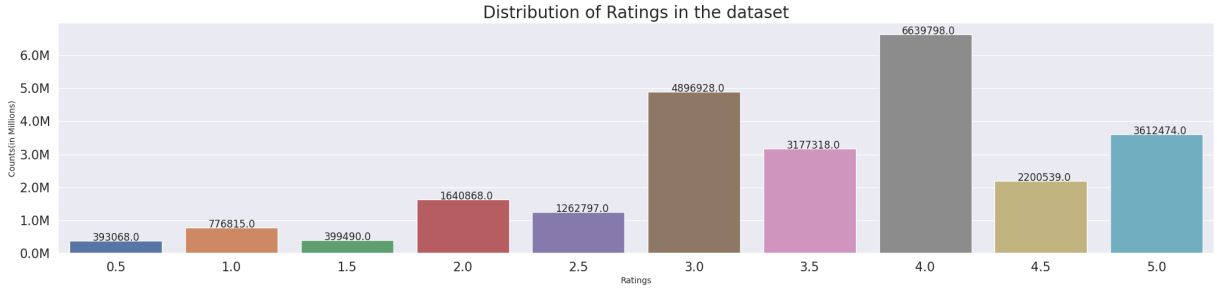
We took the genre column from the movies list dataset and applied one hot encoding to include the genres in the dataset. Afterward, we added a rating column to each movie, which represents the mean rating of all users who have rated that movie.

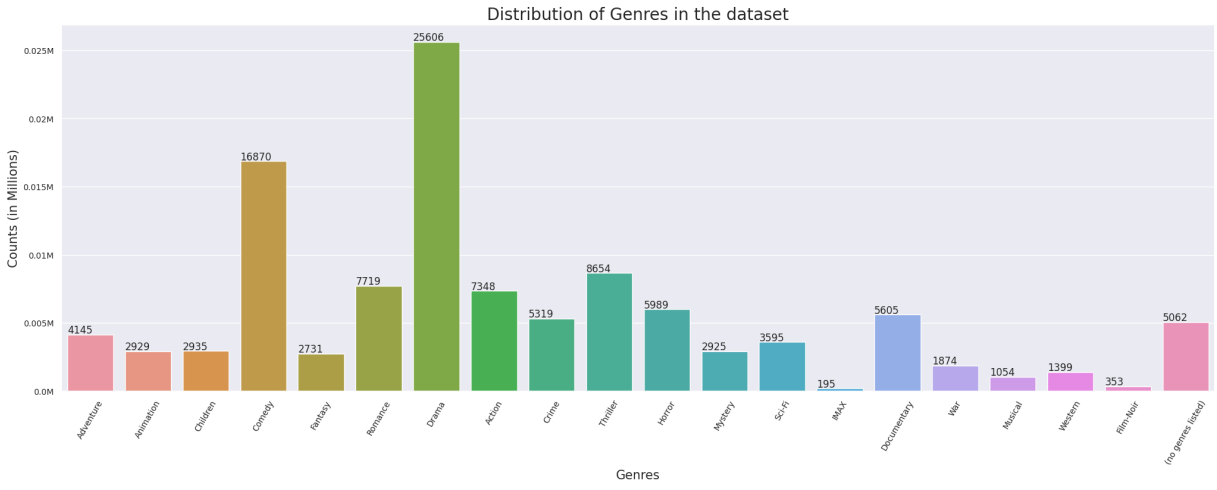


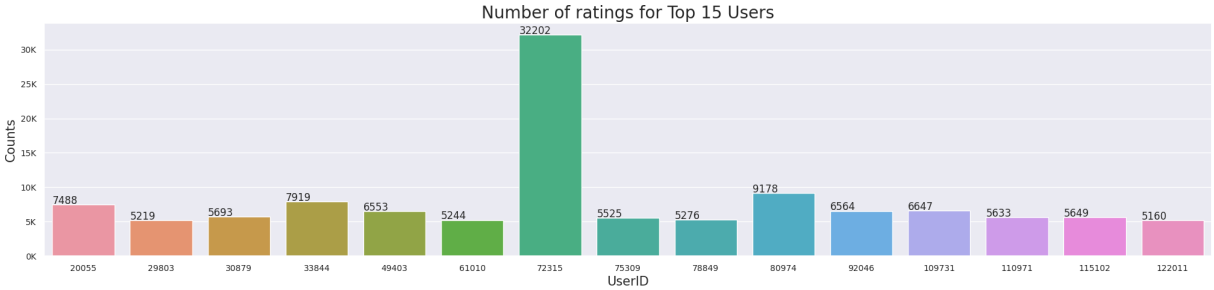
Principal Component Analysis is a popular technique for reducing the dimensionality of a dataset. Based on the analysis results, it is evident that the variance ratio does not vary significantly with the decreasing number of principal components. Therefore, reducing the dimensionality of the data may not be possible.

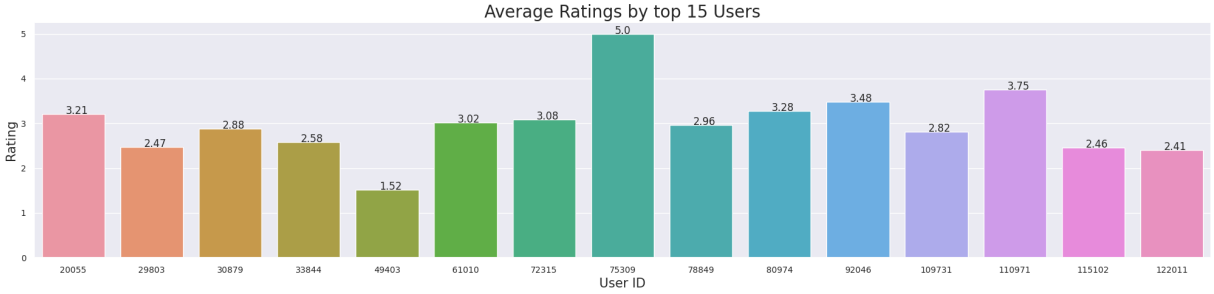
MDS (Multidimensional Scaling) is a technique used for visualizing and analyzing the similarity of data points based on their pairwise distances. However, MDS may not be the most appropriate technique for finding similarities. We first used a cosine similarity algorithm to calculate the similarity between movies based on their attributes. We then used this similarity matrix to recommend similar movies to users based on their previous ratings.

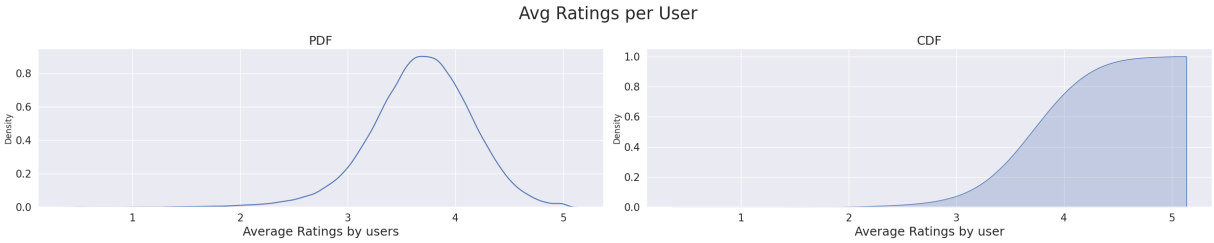
2.3 Graphs

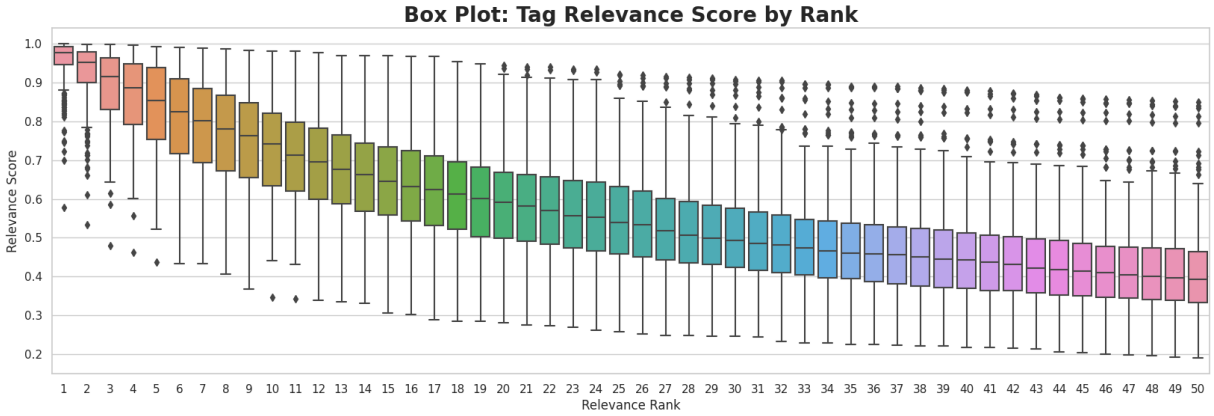


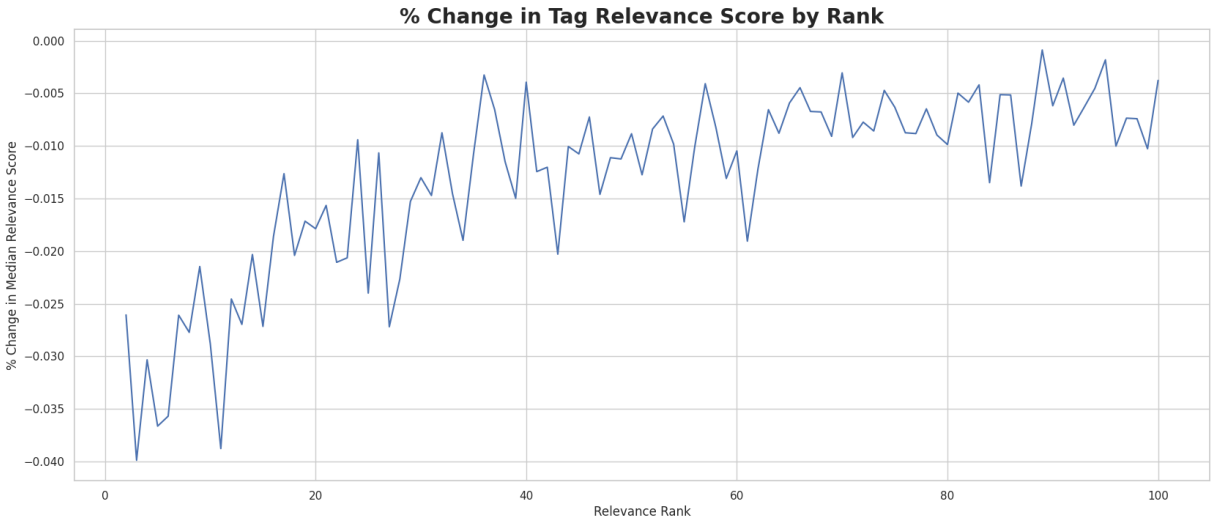



Number of ratings for Top 15 Users
Counts
UserID
20055
29803
30879
33844
49403
61010
72315
75309
78849
80974
92046
109731
110971
115102
122011
7488
5219
5693
7919
6553
5244
32202
5525
5276
9178
6564
6647
5633
5649
5160


Average Ratings by top 15 Users
Rating
User ID
20055
29803
30879
33844
49403
61010
72315
75309
78849
80974
92046
109731
110971
115102
122011
3.21
2.47
2.88
2.58
1.52
3.02
3.08
5.0
2.96
3.28
3.48
2.82
3.75
2.46
2.41


Avg Ratings per User
PDF
Density
Average Ratings by users
CDF
Density
Average Ratings by user


Box Plot: Tag Relevance Score by Rank
Relevance Score
Relevance Rank


% Change in Tag Relevance Score by Rank
% Change in Median Relevance Score
Relevance Rank

3 Methods

3.1 Method 1:

Collaborative methods for recommender systems are techniques that solely rely on past user-item interactions to produce new recommendations. These interactions are stored in a matrix known as the "user-item interactions matrix".

To make a recommendation for a given user, first, we represent each user with a vector of their interactions with different items. Then, we can use the k-nearest-neighbors algorithm to find similar users based on the user-item interactions matrix.

In the item-item method, we find items similar to the ones that the user has already interacted with positively. Two items are considered to be similar if most of the users that interacted with them did so similarly.

After finding similar users, we can recommend movies watched by those users that have the potential to be liked by the target user. However, recommending all such movies may not be entirely accurate. Therefore, we use the Doc2vec model in natural language processing to come up with a subset of movies to be recommended.

The Doc2vec model uses a neural network approach to create vector representations of variable-length pieces of text like sentences, paragraphs, or documents. These vector representations capture the semantics or meaning of the input texts, meaning that texts that are similar in meaning or context will be closer to each other in vector space.

In this model, Doc2vec creates vector representations from the top 100 tags of each movie, decided based on the relevance score of the tag given for a particular movie, after cleaning the tags and tokenizing them. To train the model, we create a Doc2vec object and initialize it with different hyperparameters such as epochs, vector size, and min count. These hyperparameters are optimized through a process called hyperparameter tuning. Finally, we build a vocabulary, which is essentially a dictionary that contains the occurrence counts of all unique words in the training corpus. The model is then trained on the inferred vectors, and the top n recommendations are returned using the model.docvecs.most_similar function, where n is specified as one of the parameters in the function. To implement this recommendation system, we first load movie data from the dataset, which contains information about movies, users, ratings, genres, and tags. Then, we create a sparse user-item interactions matrix with users as rows, movies as columns, and ratings as values. Next, we choose a target user to recommend movies to and apply the k-nearest-neighbors (KNN) algorithm to find similar users to the target user based on the user-item interactions matrix. We then get a subset of movies watched by the similar users. To recommend similar movies, we choose a target movie and apply the Doc2Vec model in NLP to find the embedding vectors for all movies from the tags in the subset. We fetch the top 100 tags of the subset of movies based on relevance score, tokenize the tags by removing non-alphanumeric elements and stopwords, represent each movie in the subset as a tagged document, and use the Doc2Vec model to generate embedding vectors for each movie in the subset.

Finally, we return the top-n most similar movies as recommendations for the target user, where the value of n depends on the number of recommendations we want to make.

Algorithm 1 Movie Recommendation System

Require: $D \leftarrow$ movie dataset ▷ Contains information about movies, users, ratings, genres, and tags
Require: $R \leftarrow$ sparse user-item interactions matrix ▷ Users as rows, movies as columns, and ratings as values
Require: $u \leftarrow$ target user ▷ User to recommend movies to
Require: $k \leftarrow$ number of similar users ▷ Hyperparameter for KNN algorithm
Require: $m \leftarrow$ target movie ▷ Movie to recommend similar movies for

procedure GENERATEREC(D, R, u, k, m)
  $S_u \leftarrow \text{KNN}(R, u, k)$ ▷ Find similar users to u based on R
  $M_S \leftarrow \text{GetSubsetMovies}(D, S_u)$ ▷ Get subset of movies watched by similar users
  $T_m \leftarrow \text{GetTopTags}(M_S, m)$ ▷ Get top tags of m based on relevance score in M_S
  $M_T \leftarrow \text{GetTaggedDocs}(M_S, T_m)$ ▷ Represent movies in M_S as tagged documents
  $E_T \leftarrow \text{Doc2Vec}(M_T)$ ▷ Generate embedding vectors for each movie in M_S
  $R_m \leftarrow \text{GetTopN}(E_T, m, n)$ ▷ Get top n most similar movies as recommendations for m
  return R_m ▷ Return movie recommendations for u

3.2 Method 2:

Weighted Random Walk Sampling for Multi-Relational Recommendation
The breadth of the dataset presents recommender systems with two challenges:

- The issue of efficiently integrating a vast range of data into a recommendation framework,
- The issue of handling a large number of potential recommendation tasks due to the great diversity of things provided.

Solution: The process of expanding a meta-path starts at a starting node and proceeds through all routes that adhere to the meta-path in order to produce a collection of destination nodes. Extensive meta-paths (those that connect beyond the immediate neighbours of users and target objects) are excellent at establishing relations on which to base suggestions, according to recent research on multi-relational recommendation in heterogeneous networks.

The creation of a relation based on paths through the network is the aim of meta-path generation. The extended **user-movie-genre-movie-user meta-path**, for instance, enables the system to start with a certain user and identify other users who have viewed movies with the same genres as the user's movies. The end result of this meta-path expansion procedure is a set of destination nodes (in this case, users) that are weighted according to how many of the enlarged meta-paths go through that node.

However, each of these models makes the assumption that every relation in the network has a homogeneous preference. Nonetheless, weighted edges are common in many networks and they often contain crucial information. For instance, if users submit explicit rating values, these encapsulate helpful information about preferences, therefore it is preferable to pay attention to them.

To rank items and provide recommendation models, Christoffel et al. developed a random walk sampling approach to determine the transition probability in a random walk model. An unweighted bipartite graph that reflects the binary relations between users and things is the foundation of this paradigm. Based on this strategy, we provide a technique that makes use of biased random walk sampling to create meta-paths in a heterogeneous network. This approach has the benefit of increasing meta-path generating efficiency and enabling user rating sensitivity.

Instead of taking all potential pathways, a random-walk variant of this algorithm selects edges from the subsequent relation in the meta-path at random. This is more effective than creating all possible paths, and the number of samples can be selected to be high enough

to give a reliable approximation of the results of a full expansion. In this work, we take a more detailed look at networks with just one "ratingędge going from user to item. In other words, the first connection from a user is believed to be to an item, with more rated products being more favoured, and is considered to have a weight that symbolises the user rating. In circumstances where consumers' quantitative preferences may be gathered, this construction is frequently used.

Consider a fraction of movie dataset in which the user provides rating value for each movie in range of $1-5$. For example, user "Bob" rated movie "Whiplash" 2 which indicates that this user did not like the movie. A meta−path expansion following user−movie−genre−movie usually takes this edge and adds all "Drama" movies to the list of "Bob" preferences ignoring the rating value. However, the reason that "Bob" rated this movie 2 may be that he does not enjoy the "Drama" genre. Therefore, following a meta-path Bob → Whiplash → Drama → $*$ can be a poor user preference projection for this user. On the other hand, "Alice" rated movie "X Men" as 5, which represents the highest interest in that movie. Therefore, a meta-path following Alice → Xmen → "IanMcKellen" → $*$ can be a stronger representation of user preferences based on an actor of movies.

The proposed algorithm is not limited to a single weighted edge and can be extended for any number of weighted edges.

Algorithm 2 Random Walk

Require: $i \leftarrow [u]$ ▷ Initialize path with starting node:user
Require: $m \leftarrow$ metapath ▷ Queue of edge types
1: **procedure** GENERATE(l, m)
2: **if** m!={} **then**
3: $me \leftarrow \text{pop}(m)$ ▷ Next edge type
4: $n \leftarrow l[1]$ ▷ Current Edge
5: $E \leftarrow \text{GetEdges}(n, me)$ ▷ Get Edges of type me
6: **if** me = user-item **then**
7: $< n, j, v > \leftarrow \text{Random}(E)$ ▷ weighted
8: **else**
9: $< n, j, v > \leftarrow \text{Random}(E)$ ▷ uniform
10: $\text{push}(j, l)$ ▷ Add node j to path
11: $\text{MPgenerate}(l, m)$
12: **else return** l

w is the edge weight

For example, in a movie dataset, consider the goal of generating expansions of the metapath user − movie − genre − movie, for user u. The algorithm would proceed as follows:

1. The random walker starts from u. Is the next edge weighted? Yes, user − movie is a weighted edge, and there are three such edges e_1 (w_1 =5),$e2$ (w_2=1), and $e3$ (w_3=3).

2. Random function returns a weighted edge user − movie in the way that the probability of selecting that edge is proportional to w, where w is the edge weight. In our example, e_1 would be chosen with probability 87%, e_3 with probability 12%, and e_2 with probability 1%.

3. Random walker moves from u to a movie m based on the selected edge in step 2.

4. Random walker is at the movie m. The next edge type in the meta−path is movie − genre. The algorithm checks if the next edge is weighted or not.

5. The next edge movie − genre is not weighted. Therefore, random function returns a random edge from movie m and genres that m belongs to.

6. Random walker moves to genre g based on the selected edge in step 4.

7. Considering the target meta-path, the next edge is movie − genre. The movie − genre an unweighted edge, therefore random function is again used, returning a random edge that leads to a movie m.

8. Random walker moves to movie m.

Although all the meta-paths generated in this paper started from a user node and an initial weighted edge, this algorithm is sufficiently flexible to handle meta-paths where the a weighted edge appears at any point in the path. For example, if an unweighted social relation of the user was available in the movie dataset, such as user − friend, a metapath user − friend − movie − actor − movie can be generated following this algorithm. The application of the algorithm to paths with multiple weighted edges is something we hope to explore in future work.

3.3 Method 3:

In the realm of movie recommendations, user ratings hold significant importance as they reflect the users' preferences and tastes. To leverage this valuable information, we have employed the concept of Random Walk, which is a widely used algorithm for personalized recommendation systems. The algorithm models the user's behavior as a random walk through a network of connected movies, where each movie is a node and the connections represent similarities between them. By traversing this network, the algorithm can identify movies that are similar to those that the user has already enjoyed and provide recommendations based on those similarities.

In this code, we are using a random walk algorithm to generate movie recommendations for a given user. The first step is to compute the movie similarity matrix using cosine similarity. To do this, we create a pivot table movie_ratings_df where each row represents a user and each column represents a movie, with the values being the rating given by that user to that movie. We then fill in the missing values with 0 using the fillna(0) method.

The proposed method involves computing the movie similarity matrix using cosine similarity on the user ratings data. The movie similarity matrix serves as a measure of similarity between any two movies in the dataset. This is done by comparing the rating patterns of the movies across all users. Next, we use the cosine_similarity function from scikit-learn's metrics module to compute the cosine similarity between each pair of movies. This generates a matrix movie_similarity_matrix where each element (i, j) represents the cosine similarity between movies i and j.

After computing the movie similarity matrix, we define the random_walk function. This function takes in a user, the movie similarity matrix, and two optional parameters -alpha and max_iterations.

We then define the random walk function, which takes the user as input and generates the top-recommended movies for that user. The function initializes the scores for all movies to zero and sets the starting movie to be the one with the highest rating by the user. The random walk is performed by iteratively updating the scores for all movies based on their similarity to the starting movie, until convergence. The final scores are then used to generate the top-recommended movies.

The scores variable is initialized as an array of zeros with a length equal to the number of movies in the movie_similarity_matrix. We then determine the starting movie for the random walk. This is done by finding the movie with the highest rating given by the user, using the idxmax() method on the user's row in movie_ratings_df. We then use get_loc() on

the movie_ratings_df.columns attribute to find the index of the starting movie in the scores array, and set its score to 1.0.

The random walk itself is performed in a loop, with each iteration computing a new set of scores based on the previous scores and the movie similarity matrix. The dot() method is used to compute the dot product between the movie_similarity_matrix and the scores array, and the result is multiplied by the damping factor-alpha. We then add the damping factor times the starting score vector, (1 -alpha) * scores, to the dot product to compute the new scores. We set the score of the starting movie to 0 in each iteration so that we don't recommend it again.

Finally, we sort the scores array in descending order and use the argsort() method to obtain the indices of the top recommended movies. We use these indices to look up the corresponding movie Ids in movie_ratings_df.columns and return the top-recommended movies.

We then generate recommendations for a given user by calling the random_walk function with the user id and the movie similarity matrix. The recommended movies are printed to the console.

In our experiments, we observed that this method has a significant advantage over other traditional recommendation techniques such as collaborative filtering, as it does not require explicit user-item interactions. Instead, it relies solely on user ratings, which are more readily available and easier to obtain.

Overall, the random walk method based on user ratings is a promising approach for movie recommendations, with great potential for further research and development.
The algorithm of the above mentioned method is as follows:

Algorithm 3 Movie Recommendation System

Require: movie_ratings_df, alpha, max_iterations
Ensure: top-recommended movies for the user

1: **procedure** RANDOM_WALK($user$, movie_similarity_matrix, $alpha$ = 0.85, max_iterations=100)
2: Initialize the scores for all movies: $scores$ = array of zeros with shape (number of movies)
3: Set the starting movie to be the one with the highest rating by the user: user_ratings = row of movie_ratings_df corresponding to the user, starting_movie = movie with the highest rating in user_ratings, starting_movie_index = index of starting_movie in movie_ratings_df.columns, scores[starting_movie_index] = 1.0
4: **for** i in range(max_iterations) **do**
5: new_scores = $\alpha *$ dot product of movie_similarity_matrix and $scores$ + $(1-\alpha)$ * $scores$
6: new_scores[starting_movie_index] = 0
7: $scores$ = new_scores
8: **return** the top-recommended movies
9: Create a pivot table movie_ratings_df and fill in the missing values with 0 using the $fillna(0)$ method.
10: Generate movie_similarity_matrix using the cosine_similarity function from scikit-learn's $metrics$ module. =0

4 Results and Discussion

4.1 Method-1

This description outlines a recommendation system that uses collaborative filtering and natural language processing (NLP) methods to suggest movies to a target user. The system first applies k-nearest neighbors (KNN) to find similar users based on past user-item interactions. Then, using the item-item method, the system recommends movies that are similar to ones the user has positively interacted with. To filter the recommendations further, the system employs the Doc2Vec model in NLP to generate vector representations of the movies based on their tags. The model's most similar function returns the top-n most similar movies as recommendations for the target user. Overall, the system leverages user interactions and movie tags to suggest movies that are similar to those the user has enjoyed in the past.

The result shows top-70 movies for user with userid=1

```
In [33]: ratings[ratings.userId==1]
Out[33]:
    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
..     ...      ...     ...         ...
65       1    27193     3.0  1147879774
66       1    27266     4.5  1147879365
67       1    27721     3.0  1147869115
68       1    31956     3.5  1147877610
69       1    32591     5.0  1147879538

[70 rows x 4 columns]
```

The result shows information regarding target movie with movieId=1704

```
In [32]: movies[movies.movieId==1704]
Out[32]:
      movieId                   title         genres
1640     1704  Good Will Hunting (1997)  Drama|Romance
```

The following are the recommendation for the above two details.

```
2795    Jagged Edge (1985)
Name: title, dtype: object
2501    Monty Python's And Now for Something Completel...
Name: title, dtype: object
5897    Millennium Actress (Sennen joyû) (2001)
Name: title, dtype: object
2496    Tomb of Ligeia, The (1965)
Name: title, dtype: object
5747    Knights of the Round Table (1953)
Name: title, dtype: object
Series([], Name: title, dtype: object)
774    Crows and Sparrows (Wuya yu maque) (1949)
Name: title, dtype: object
Series([], Name: title, dtype: object)
5587    Wiz, The (1978)
Name: title, dtype: object
Series([], Name: title, dtype: object)
977    Monty Python's Life of Brian (1979)
Name: title, dtype: object
5973    Crimewave (1985)
Name: title, dtype: object
1522    Midnight in the Garden of Good and Evil (1997)
Name: title, dtype: object
4054    High Spirits (1988)
Name: title, dtype: object
3548    Lost Souls (2000)
Name: title, dtype: object
1511    Amistad (1997)
Name: title, dtype: object
Series([], Name: title, dtype: object)
2517    Christmas Story, A (1983)
Name: title, dtype: object
```

```
5870    Handmaid's Tale, The (1990)
Name: title, dtype: object
Series([], Name: title, dtype: object)
2448    Golden Child, The (1986)
Name: title, dtype: object
6773    Totally F***ed Up (1993)
Name: title, dtype: object
3455    Affair of Love, An (Liaison pornographique, Un...
Name: title, dtype: object
3549    Billy Jack (1971)
Name: title, dtype: object
193    To Wong Foo, Thanks for Everything! Julie Newm...
Name: title, dtype: object
1508    Man Who Knew Too Little, The (1997)
Name: title, dtype: object
2923    Wayne's World (1992)
Name: title, dtype: object
6150    Quick Change (1990)
Name: title, dtype: object
2287    Baby Geniuses (1999)
Name: title, dtype: object
6261    Lord of the Rings: The Return of the King, The...
Name: title, dtype: object
```

4.2 Method-2

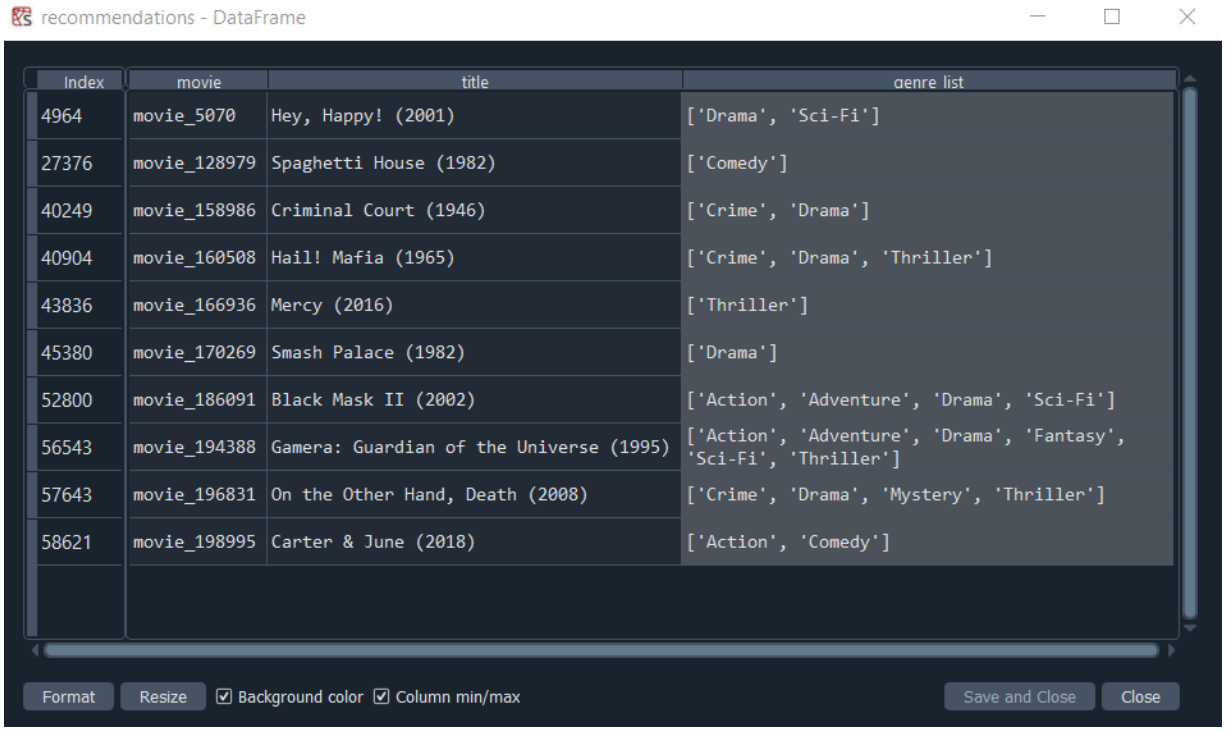
Weighted Random Walk Sampling is a graph-based recommendation method that utilizes the relationships between items and users to provide personalized recommendations. It works by creating a weighted graph of the items and users, where the weights represent the strength of the relationships between them. The algorithm then performs random walks on the graph, starting from the user's known items and following the edges of the graph to discover related items that the user may be interested in. The weights of the edges are used to bias the random walk towards more relevant items, resulting in personalized recommendations that are tailored to the user's preferences.

The result shows top-112 movies for user with userid=130048

```
In [241]: ratings[ratings.userId == 130048]
Out[241]:
          userId  movieId  rating  timestamp         user       movie
19999935  130048       29     5.0  981338154  user_130048    movie_29
19999936  130048       32     5.0  981338285  user_130048    movie_32
19999937  130048       47     5.0  981339934  user_130048    movie_47
19999938  130048       50     4.0  981340581  user_130048    movie_50
19999939  130048      111     5.0  981340602  user_130048   movie_111
...          ...      ...     ...        ...          ...         ...
20000042  130048     3793     4.0  981338285  user_130048  movie_3793
20000043  130048     3863     3.0  981338285  user_130048  movie_3863
20000044  130048     3889     1.0  981339197  user_130048  movie_3889
20000045  130048     3959     4.0  981338002  user_130048  movie_3959
20000046  130048     4079     4.0  981338063  user_130048  movie_4079

[112 rows x 6 columns]
```

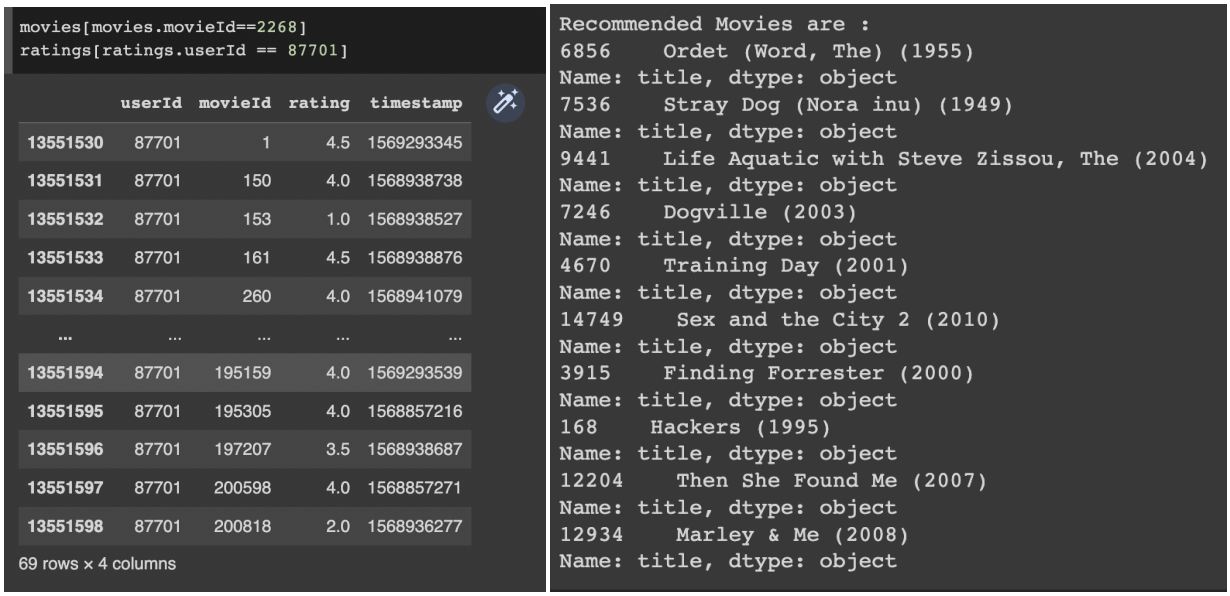
The following are the recommendation.



4.3 Method-3

It uses a random walk algorithm for generating personalized movie recommendations based on user ratings. The algorithm uses cosine similarity to compute a movie similarity matrix and performs a random walk through this matrix to identify similar movies and provide recommendations. The function takes in a user, the movie similarity matrix, and two optional parameters, alpha, and max_iterations. The approach has advantages over traditional colla-

borative filtering techniques as it relies solely on user ratings, which are more readily available and easier to obtain. This method is a promising approach for movie recommendations with the potential for further research and development.



```
movies[movies.movieId==2268]
ratings[ratings.userId == 87701]
```

	userId	movieId	rating	timestamp
13551530	87701	1	4.5	1569293345
13551531	87701	150	4.0	1568938738
13551532	87701	153	1.0	1568938527
13551533	87701	161	4.5	1568938876
13551534	87701	260	4.0	1568941079
...	...	...	...	...
13551594	87701	195159	4.0	1569293539
13551595	87701	195305	4.0	1568857216
13551596	87701	197207	3.5	1568938687
13551597	87701	200598	4.0	1568857271
13551598	87701	200818	2.0	1568936277

69 rows × 4 columns

```
Recommended Movies are :
6856    Ordet (Word, The) (1955)
Name: title, dtype: object
7536    Stray Dog (Nora inu) (1949)
Name: title, dtype: object
9441    Life Aquatic with Steve Zissou, The (2004)
Name: title, dtype: object
7246    Dogville (2003)
Name: title, dtype: object
4670    Training Day (2001)
Name: title, dtype: object
14749    Sex and the City 2 (2010)
Name: title, dtype: object
3915    Finding Forrester (2000)
Name: title, dtype: object
168    Hackers (1995)
Name: title, dtype: object
12204    Then She Found Me (2007)
Name: title, dtype: object
12934    Marley & Me (2008)
Name: title, dtype: object
```

Top-112 movies for userid=87701(left) Information for target movie movieId=2329()

5 References

1. Argerich, L., Pujol, J. M. (2017). A random walk algorithm for personalized movie recommendations. In *Proceedings of the 2017 Symposium on Student Research and Creative Inquiry* (pp. 9-14). DePaul University. https://scds.cdm.depaul.edu/wp-content/uploads/2017/05/SOCRS_2017_paper_3.pdf

2. Random Walk Algorithms for Movie Recommendation. (2011, October 31). Cornell University INFO 2040 Blog. https://blogs.cornell.edu/info2040/2011/10/31/random-walk-algorithms-for-movie-recommendation/

3. Singh, A. (2019, June 6). Detecting Document Similarity with Doc2Vec. Towards Data Science. https://towardsdatascience.com/detecting-document-similarity-with-doc2vec

4. Build a Recommendation Engine With Collaborative Filtering.(n.d.).Real Python. https://realpython.com/build-recommendation-engine-collaborative-filtering/

5. KNN Algorithm In Machine Learning | KNN Algorithm Using Python | K Nearest Neighbor. https://www.youtube.com/watch?v=4HKqjENq9OU

6. A gentle introduction to Doc2Vec. https://medium.com/wisio/a-gentle-introduction-to-doc

7. Prof. Bibhas Adhikari Big Data Analysis Course http://www.facweb.iitkgp.ac.in/~bibhas/BDA23.html

8. Christoffela, F., Paudela, B., Newellb, B., Bernstein, A.(2015). Blockbusters and Wallflowers: Accurate, Diverse, and Scalable Recommendations with Random Walks, RecSys 2015 (pp.163-170) https://dl.acm.org/doi/pdf/10.1145/2792838.2800180