Movie Recommendation System Report

1. Introduction

Movie recommendation systems are an advanced trend in the current digitized world. Reading the local TV guides, renting CDs and DVDs, watching tapes or filmstrip projectors are endangered or even extinct. All movie libraries are transformed to online platforms. As “**“Desire drives destiny”**, these online movie streaming services want the users to watch more movies. In layman terms, advertising is the popular way to make people know about the movies. But advertising is a generic method which doesn't serve to the taste of each and every individual. Here the advanced approach called recommender systems is playing a vital role. In our case, “Movie recommendation systems” predicts the user’s preferences based on their past choice and preferences and also evaluates the similarities of movies. Accumulating these personalized recommendations are given to each and every user to give a smooth experience for users. Movie recommendation systems use three different approaches namely, User-based, Item-based and Random-walk.

1.1 User Based Recommendation

The technique known as "User-Based Collaborative Filtering" works by using ratings from other users who share the target user's tastes to forecast what the user might like. For example, if User A and User B watched Movie 1 and movie 2 and User A watched Movie 3, then Movie 3 is also recommended for User B. Here the similarities between users are evaluated considering the rating given for movies from a User - Movie matrix. Similarities can be calculated with metrics like Cosine similarity or Pearson correlation. With the most similar users (Number of similar users depends on the given scenario), their ratings are aggregated and top movies will be recommended.

1.2 Item Based Recommendation

Item-based recommendation is a type of collaborative filtering for recommender systems that uses ratings of objects to determine how similar they are to one another. For example, if User A uses Item 1 and Item 2. If Item 3 is similar to both the used items, then Item 3 is recommended to the user. Here the similarities between movies are evaluated considering the rating given for movies from a User - Movie matrix. Similarities can be calculated with metrics like Cosine similarity or Pearson correlation. Verify which movies received high ratings from the target user. Look for other things that

are similar to those. Sorting by the highest similarity and those items will be recommended.

1.3 Random-walk Based Recommendation

A Random-Walk Based Recommendation is a technique that makes recommendations for movies by mimicking how a user could "wander" across a network of links, such as a bipartite user-item graph or a social graph. Here both the users and movies act as nodes. Build a graph with users and movies as vertices and edges will be drawn from user to movie if that movie is rated by that user with weight as their rating. Begin walking from a user, and do a random walk for a given number of steps. Based on the number of times a movie is visited, top visited movies are recommended.

2. Dataset Description

One of the most popular datasets for developing and evaluating recommendation systems, particularly collaborative filtering algorithms, is the MovieLens 100K dataset. It contains **100,000 ratings** from **943 users** on **1,682 movies**, collected by the GroupLens research group. The minimum rating is 1 (worst) and maximum rating is 5 (best).

2.1 User Features

user_id	Unique identifier for each user (from 1 to 943)
age	Age of the user
gender	Male (M) or Female (F)
occupation	User's occupation (e.g., student, engineer)
zip_code	U.S. ZIP code

2.2 Movie Features

movie_id	Unique identifier for each movie (from 1 to 1682)
title	Movie title with release year (e.g., *Toy Story (1995)*

release_date	When the movie was released
genre	19 binary genre flags (e.g., Action, Comedy, Drama)

2.3 Ratings Features

user_id	ID of the user who rated
movie_id	ID of the rated movie
rating	An integer from 1 (worst) to 5 (best)
timestamp	When the rating was recorded (UNIX time)

2.4 Pre- Processing Steps

Three datasets were considered for users, movies and data (ratings). These raw data are converted into dataframes with column names. The ratings dataset is read from "u.data" using tab ('\t') as a separator and column names ("user_id", "movie_id", "rating" and "timestamp"). The movies dataset is read from "u.item" using '|' as a separator, use columns (0, 1, 2), encoding ("latin-1") and name the columns (movie_id, title, and release_date). The users dataset is read from "u.user" using '|' as a separator, use columns (0, 1, 2, 3) and name the columns (user_id, age, gender, and occupation). And for better understanding, these data frames can be saved as .csv. After this pre-processing steps are performed:

3. Methodology

3.1 User Based Collaborative Filtering

- Find the cosine similarity between users using cosine_similarity functions and also considering Nan as zeroes.
- Both rows and columns will be users and all the diagonal values will be 1, as similarities between the same items will be one. Cosine similarity always gives similarity between 0 and 1. So, users with similarity greater than 0 are considered as similar.

3.2 Item Based Collaborative Filtering

- Do transpose of the user movie matrix, where items turn to rows and users turn to columns.
- Find the cosine similarity between movies using cosine_similarity functions and also considering Nan as zeroes.
- Both rows and columns will be movies and all the diagonal values will be 1, as similarities between the same items will be one. Cosine similarity always gives similarity between 0 and 1. So, movies with similarity greater than 0 are considered as similar.

3.3 Random - walk Based Pixie Algorithm

- All the graph based approaches perform well for large datasets. The Movie Lens dataset which we took is also quite large enough.
- Pixie performs well when the graph is sparse but is well - connected. Here most users haven't rated most movies but there are enough overlaps between users and movies to connect everyone through a few hops.
- It also helps to find indirect relationships.

4. Implementation Details

IDE - Jupyter Notebook
Language - Python3
Libraries - Pandas, Numpy, cosine_similarity, random

4.1 Understanding the dataset & Formatting

- It is a better practice to visualise the shape of the dataset before starting the implementation. So, the first 10 rows of all the three datasets are visualised.
- Datasets are converted to dataframes using pandas and saved as .csv files.
    - Data (ratings) dataset is separated using delimited '\t'. With that, columns are separated as 'user_id', 'movie_id', 'rating' and 'timestamp'.
    - Movies dataset is separated using delimiter '|'. With that only three columns are considered for forming data frames and those are: 'movie_id', 'title', 'release_date'. Encoding is explicitly given as latin-1 to handle special or non - ASCII characters.
    - Users dataset is separated using delimiter '|'. With that only four columns are considered for forming data frames and those are: 'user_id', 'age', 'gender', 'occupation'.

4.2 Data Pre - Processing

- Timestamps in rating are converted into readable date time format.
- Check for missing values in the dataframe. There are no missing values in users and ratings. In movies there is a missing value in release and it is trivial for our prediction. So, there is no need for dropping any missing values.
- There are no duplicates in all the three data frames.
- So there are no rows removed from any dataframe and it remains as it is.

4.3 User Based Filtering

- Create a user-movie matrix with users as rows and movies as columns using pivot function.
- Fill the non rated movies (Nan) with 0.
- Find the user-user similarity using Cosine - similarity and convert it to a 2d data frame with both rows and columns as users.
- Receive the user_id from the user.
- If the user_id is less than 0 or greater than 943 throw error or execute the below.
- Creates a Boolean mask (a matrix of True and False) from user_movie_matrix, where, ‘True’ means the user has rated the movie (i.e., the value is not zero) and ‘False’ means the user has not rated the movie (i.e., the value is zero).
- For the desired user, get the entire row from the user_movie_matrix.
- Change the same user’s similarity as 0 as we need to exclude the similarity of the same user.
- Keep similarity scores only for users who rated a given movie. Zeros out the similarity if a user didn’t rate that movie. This is done by multiplying the similarity row with the masked matrix. (True*similarity score = similarity score, Fasle*similarity score = 0.0, True*similarity score = 0 = 0.0).
- Multiply the above non_zero_mask with the user_movie matrix. A matrix where each rating position is replaced by the similarity of that user, only if the user rated that movie. Otherwise, it remains 0.
- Sum all the ratings of the movie and divide with the sum of similarities of all users (considered only when the user has rated the movie), which is the weighted average.
- Sort the predicted ratings of movies in descending order and take the top N (default = 5) ratings.
- With the sorted movie_ids get their corresponding movie name (title) from the movie data frame and display it.

4.2 Movie Based Filtering

- Take the transpose of user - movie matrix, where movies are rows and users are columns.
- Fill the non rated movies (Nan) with 0.
- Find the movie-movie similarity using Cosine - similarity and convert it to a 2d data frame with both rows and columns as movies.
- Receive the movie name from the user.
- Get the movie_id with the inputted movie name by using string matching.
- If any movie_id can't be retrieved, then throw an error as invalid movie name.
- From the movie-movie similarity matrix, retrieve the movie_id row which we inputted as the movie name.
- Then convert it into a separate data frame to view as a .csv file - Optional.
- Exclude the similarity of the inputted movie.
- Map or merge the movie_id with their corresponding titles using merge functions.
- Sort in descending order and retrieve and print the top num (default = 5) movies.

4.3 Adjacency list Formation

- In graph theory, graphs are represented with several data structures like adjacency list, adjacency matrix and edge list. Of this the most space and time efficient way is the adjacency list.
- Merge title in movie data frame to ratings and ignore error when merge is executed multiple times.
- Initialize a graph (bi-partite graph with users as one type and movies as another type) as a dictionary in python.
    - Connect movies and users when the user has rated that movie
    - As it is a bipartite graph, both movies and users must act as nodes and it is undirected. To implement a weighted pixie algorithm, weight must be considered while forming a graph.
    - Example: graph ={
      User 1 : {Movie A : rating, Movie B : rating},
      User 2 : {Movie C : rating, Movie B : rating},
      Movie A : {User 1 : rating},
      Movie B : {User 1 : rating, User 2 : rating},
      Movie C : {User 2 : rating}
      }

4.4 Weighted Pixie Algorithm - User Based:

- Execute the random walk for the number of times the walk length is inputted.

    - As it is a user based approach, start the walk from the user and get the connected movies from the user. Among the connected movies, get the maximum rating and if there is only one movie with the maximum rating (say 4), move to that movie. If there are more than one movie with rating 4 (highest), randomly move to any of the highest ranked movies.
    - If the current standing node is a movie (we are having movie name and user id and hence if the current node is a string then it must be a movie) then add to dictionary and increment count, if the movie is already visited just increment the visit count of that movie.
    - Repeat the loop with the randomly found (highest rated) node as the current node until the walk length is completed.
- Once the walk is completed, sort the movies in the visited dictionary based on visit count in descending order. Get only the number of movies to be recommended from the array and convert to data frame and display.

5. Results and Evaluation

Top 5 movies for User 1 with User - user collaborative filtering

```
Enter the user id :  1
| Ranking | Movie Name                          |
|---------|-------------------------------------|
| 1       | Star Wars (1977)                    |
| 2       | Return of the Jedi (1983)           |
| 3       | Fargo (1996)                        |
| 4       | Raiders of the Lost Ark (1981)      |
| 5       | Silence of the Lambs, The (1991)    |
```

Top 5 similar movies for “Jurassic Park (1993)” based on Item - item collaborative filtering.

```
Enter the movie name:  Jurassic Park (1993)
| Ranking | Movie Name                                |
|---------|-------------------------------------------|
| 1       | Top Gun (1986)                            |
| 2       | Speed (1994)                              |
| 3       | Raiders of the Lost Ark (1981)            |
| 4       | Empire Strikes Back, The (1980)           |
| 5       | Indiana Jones and the Last Crusade (1989) |
```

Top 5 movies recommended for user based on weighted - pixie algorithm

```
Enter user id :  1
Enter the walk length :  12
Enter the number of top movies to be listed :  5
 Ranking                        Movie Name
       1           Terminator, The (1984)
       2                 Net, The (1995)
       3 Shawshank Redemption, The (1994)
       4               Stand by Me (1986)
       5                    Gandhi (1982)
```

5.1 Accuracy - User Based collaborative filtering

- RMSE: 1.0146 (On average, your predicted ratings differ from the actual ratings by about 1.01 stars.)
- MAE : 0.8060 (On average, the absolute difference between your predicted and actual ratings is about 0.81 stars.)
- The RMSE is being greater than 1 will be due to the below reasons

Data sparsity: Predictions are frequently based on scant data because not all users have reviewed every film.
No adjustment of hyperparameters: Similarity-based forecasts are not restricted to the "top-k most similar users," which could include neighbors who are noisy or weak.
No baseline correction: User bias is not taken into account (some users consistently provide high ratings, while others give negative ratings).
Cold start effect: Predictions are less accurate when a user or film shows up in the test set but receives low ratings in the training set.
Simple averaging: This method can increase accuracy by weighting users based on similarity without taking into account user mean ratings or standard deviation.

5.2 Accuracy - Item Based collaborative filtering

- RMSE: 1.0128
- MAE : 0.8067
- The above explanation in user based collaborative filtering holds for this as well.

5.3 Accuracy - Weighted Pixie Algorithm

Implementation of the weighted Pixie algorithm, precision, recall, and hit rate are calculated based on how often the recommended movies (determined by random walk visit counts) match the actual movies a user rated in the test set. Since the recommendations are driven by visit frequency rather than actual rating values or user preferences, the algorithm may miss relevant movies, leading to low precision and

recall. Additionally, because the walk is stochastic and the graph is sparse, many relevant items might not be visited frequently enough to appear in the top-K list, resulting in lower hit rates as well.

Note: To get the accuracy results here we need to execute the algorithm for all users for at least 1000 which is computationally expensive. And if we consider only 100 test users the accuracy results will be misleading.

5.3 Limitations

Scalability: As the number of users or items increases, algorithms such as item-item and user-user collaborative filtering may become computationally costly, particularly when pairwise similarity calculations and matrix operations are involved.
Cold Start Issue: Because these algorithms mostly rely on past data, they won't provide insightful recommendations for new users or films with little to no interaction data.
Data Sparsity: Most users rate very few films in real-world rating matrices, which reduces the usefulness of similarity-based techniques and may result in erroneous predictions.
Pixie Randomness: Because the Pixie algorithm relies on random walks, the outcomes may vary. It might not adequately represent each user's interests if customization tuning or weighting based on prior preferences are not used.

Lack of Contextual Awareness: In real-world situations, context—such as time, place, or mood—can be extremely important. However, the guidelines fail to take this into account. 6.

6. Conclusion

6.1 Key Takeaways

- Based on identifying people who share similar preferences and suggesting films they enjoyed.
- Discovered related films to suggest to viewers based on item-to-item collaborative filtering.
- A graph-based method, modeled after Pinterest's recommender system, that use random walks across a bipartite graph (users - movies).
- Used explicit ratings from the MovieLens 100K dataset.
- Created a bipartite user and movie graph.
- To mimic user behavior, go around this graph using weighted, tie-broken random walks. The most popular films were suggested.

6.2 Potential Improvements

A number of improvements could be investigated to make the existing recommendation system better. By utilizing both user-item interactions and movie metadata such as genres, directors, or tags, a hybrid model that blends collaborative filtering with content-based techniques may provide more individualized results. The model can gain a better understanding of user trends and preferences by including features like timestamps, movie popularity, and user demographics. Complex patterns can be more successfully captured by utilizing model-based approaches like matrix factorization or neural collaborative filtering rather than memory-based ones like User-User or Item-Item similarity. By learning improved representations of users and items, graph-based advancements such as GNNs or sophisticated random walk methods can improve algorithms like Pixie. Additionally, large-scale recommendations might be more efficient if calculation were accelerated through parallel processing or approximation search. Lastly, a more comprehensive understanding of suggestion quality would be possible by improving evaluation using metrics like NDCG or diversity and originality checks.

6.3 Real world applications

There are numerous real-world uses for the recommendation techniques employed in this research, such as the Pixie algorithm, item-item collaborative filtering, and user-user collaborative filtering. These algorithms aid in content personalization on streaming services like Netflix and Spotify by making movie or music recommendations based on user activity or similarities to other users. Similar strategies are employed by e-commerce sites like Amazon to make product recommendations based on past

purchases and item similarity. Social media sites use graph-based techniques, such as Pixie, to navigate through interaction networks and recommend friends, groups, or content. Based on patient profiles and comparable medical histories, these techniques might suggest experts or treatment regimens in the healthcare industry. Collaborative filtering can also be used in education to provide tailored course or material recommendations based on students' learning preferences.