Movie Recommendation System
User-Based Collaborative Filtering

Under the Guidance of
Professor Gary Davis

Bharath Anand
#ID: 02044023
Graduate Student
Department of Data Science
University of Massachusetts
Dartmouth

Balakrishna Vardhineni
#ID: 02069565
Graduate Student
Department of Data Science
University of Massachusetts
Dartmouth

2. Abstract

This project aims to develop a movie recommendation system using the User-based Collaborative Filtering technique. The system predicts movie recommendations based on users' behavior and past searches. The MovieLens Latest Dataset, which contains 100,000 ratings and 9,000 movies by 600 users, is used to train and test the model. The outcome of this project is a functional movie recommendation system that can provide personalized recommendations to users based on their preferences and history.

3. Introduction and Background

The movie industry has seen exponential growth over the years, and the demand for movies keeps increasing with each passing day. However, with the vast number of movies available, it can be challenging for movie enthusiasts to find films that match their preferences. This is where movie recommendation systems come in. Movie recommendation systems are machine learning-based approaches that use a user's past behavior to filter and predict their possible movie choices. This report will discuss the different filtration strategies for movie recommendation systems, namely content-based and collaborative filtering.

Movie recommendation systems have gained immense popularity in recent times, and they are widely used by streaming platforms such as Netflix, Hulu, and Amazon Prime Video. These platforms use different algorithms and strategies to filter and predict movie preferences based on a user's past behavior. The two most popular filtration strategies used in movie recommendation systems are content-based filtering and collaborative filtering.

3.1 Collaborative Filtering:

Collaborative filtering is a filtering strategy based on the combination of the relevant user's and other users' behaviors. The system compares and contrasts these behaviors for the most optimal results. It's a collaboration of multiple users' film preferences and behaviors.

How it Works:

The core element in this movie recommendation system and the ML algorithm it's built on is the history of all users in the database. Basically, collaborative filtering is based on the interaction of all users in the system with the items, i.e., movies. Thus, every user impacts the final outcome of this ML-based recommendation system, while content-based filtering depends strictly on the data from one user for its modeling.

Collaborative Filtering Algorithms:

Collaborative filtering algorithms are divided into two categories:

User-based collaborative filtering: The idea is to look for similar patterns in movie preferences in the target user and other users in the database.

Item-based collaborative filtering: The basic concept here is to look for similar items (movies) that target users rate or interact with.

3.2 User-Based Filtering:

User-based collaborative filtering is a technique used in movie recommendation systems to suggest movies to users based on their similarity to other users. The main idea is that users who have similar preferences will likely have similar preferences in the future. Here's a high-level overview of how user-based collaborative filtering works:

1. Collect user preferences: Gather data on user preferences, such as movie ratings or viewing history.
2. Calculate similarity between users: Measure the similarity between users using a similarity metric, such as Pearson correlation coefficient or cosine similarity. This step helps identify users with similar tastes.
3. Find the most similar users (neighbors): For a target user, find a set of users who are most similar to them, known as neighbors. The number of neighbors can be a fixed number (k-nearest neighbors) or a threshold-based approach.
4. Generate recommendations: Calculate the predicted rating for each movie that the target user hasn't seen yet, based on the ratings given by their neighbors. The predicted rating can be a weighted average of the neighbors' ratings, where the weights are the similarity scores between the target user and each neighbor.

5. Rank and recommend: Sort the movies by their predicted ratings and recommend the top-N movies to the target user.

In conclusion, movie recommendation systems are crucial for movie enthusiasts who want to find movies that match their preferences. The two most popular filtration strategies used in these systems are content-based filtering and collaborative filtering. While content-based filtering uses only data from one user, collaborative filtering relies on the interaction of all users in the system with the items. As the movie industry keeps growing, movie recommendation systems will play an increasingly vital role in helping movie enthusiasts find the movies they love.

4. Methods:

For this report, we've used the MovieLens Latest Datasets to perform cluster analysis on customer segmentation and the dataset file can be found and downloaded from here. The MovieLens Latest Dataset used in this project is a widely used dataset for developing and testing movie recommendation systems. It contains 100,000 ratings of 9,000 movies given by 600 users. This dataset is relatively small compared to other movie recommendation datasets, but it is still sufficient to build a basic movie recommendation system.

The code implements a User-based Collaborative Filtering approach to make movie recommendations. The strategy is based on the similarity of items, or in this case, movies. The approach uses the cosine similarity metric and the k-nearest neighbors algorithm to find similar movies to a given movie and recommend them.

4.1 Cosine similarity:

In a movie recommendation system that uses user-based collaborative filtering, cosine similarity can be used to find similarities between movies based on the ratings given to them by users. Here's how it works:

1. First, we create a matrix where each row represents a movie and each column represents a user. The cells in the matrix contain the ratings that each user has given to each movie.

2. Next, we calculate the cosine similarity between each pair of movies. This involves treating each movie's ratings as a vector in a multi-dimensional space and then calculating the cosine of the angle between the two vectors. The cosine similarity values range from -1 to 1, with 1 indicating that the two movies are identical in terms of user ratings, and -1 indicating that they are completely dissimilar.

3. Once we have calculated the cosine similarity between each pair of movies, we can use it to predict how much a user would like a movie they haven't seen yet. To do this, we first select a target user and a movie they haven't seen yet. Then, we find the k most similar movies to the target movie based on their cosine similarity values. Finally, we calculate a weighted average of the ratings the target user has given to these k movies, where the weights are the cosine similarity values. This weighted average gives us an estimate of how much the target user would like the movie.

Overall, cosine similarity is helpful for finding similarities between movies in a movie recommendation system and using those similarities to make personalized movie recommendations. It is particularly well-suited for sparse data where many ratings are missing and has been shown to perform well in many real-world recommendation systems.

4.2 Pearson Correlation:

In a movie recommendation system that uses user-based collaborative filtering, Pearson Correlation can be used to find similarities between users based on their movie ratings. Here's how it works:

1. First, we create a matrix where each row represents a user and each column represents a movie. The cells in the matrix contain the ratings that each user has given to each movie.

2. Next, we calculate the Pearson Correlation coefficient between each pair of users. This coefficient measures the linear correlation between two variables, in this case, the movie ratings of two users. The Pearson Correlation coefficient ranges from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.

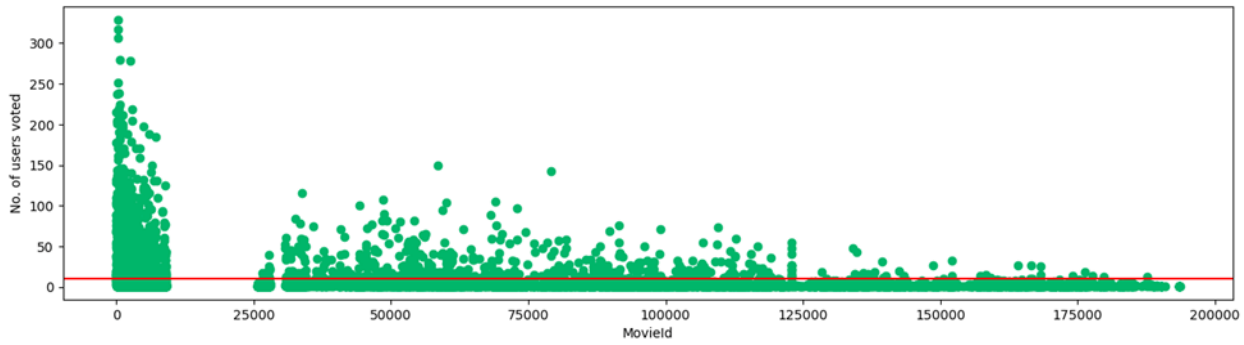
3. Once we have calculated the Pearson Correlation coefficient between each pair of users, we can use it to predict how much a user would like a movie they haven't seen yet. To do this, we first select a target user and a

movie they haven't seen yet. Then, we find the k users who are most similar to the target user based on their Pearson Correlation coefficients. Finally, we calculate a weighted average of the ratings that these k users have given to the movie, where the weights are the Pearson Correlation coefficients. This weighted average gives us an estimate of how much the target user would like the movie.

Overall, Pearson Correlation is useful for finding similarities between users in a movie recommendation system and using those similarities to make personalized movie recommendations.

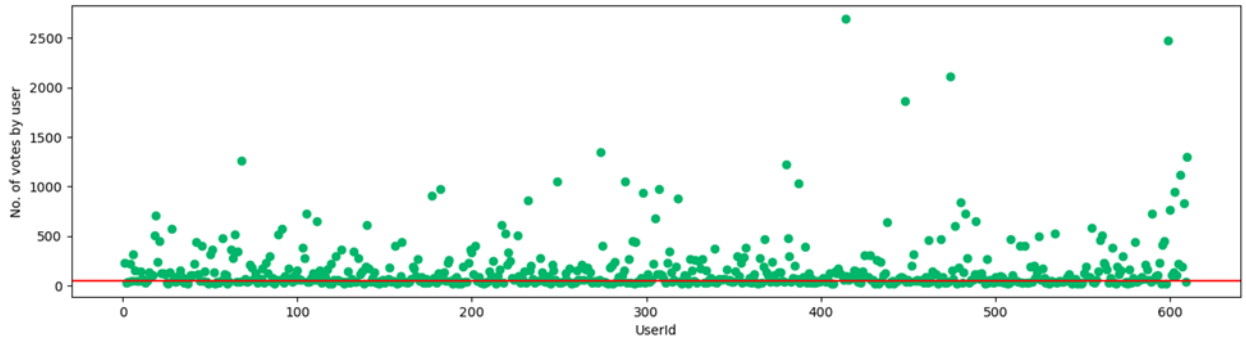
5. Results:

This graph represents, how many users have voted for each and every movie.



This helps us understand how all the movies here are voted and the red line represents the minimum number of votes for each movie.

The graph below helps us understand how many votes each user has voted.



This helps us understand that almost all the users have voted and it can help us recommend the movie much better.

After performing the user-based filtering through a function made for the recommendation. We can use the function to get recommendations as follows:

All the above movies listed are recommendations for the movie Forest Gump:

```
In [12]: print(f"Because you watched {movie_finder(movie_of_interest)}...")
         for r in related:
             recommended_title = get_movie_title(r[0])
             if recommended_title != movie_finder(movie_of_interest):
                 print(recommended_title)
```

```
Because you watched Forrest Gump (1994)...
Shawshank Redemption, The (1994)
Silence of the Lambs, The (1991)
Pulp Fiction (1994)
Schindler's List (1993)
Braveheart (1995)
Jurassic Park (1993)
Seven (a.k.a. Se7en) (1995)
Apollo 13 (1995)
Usual Suspects, The (1995)
```

We can't interpret the results as is since movies are represented by their index. We'll have to loop over the list of recommendations and get the movie title for each movie index.

```
In [18]: for r in recommendations:
             recommended_title = get_movie_title(r[0])
             print(recommended_title)
```

```
Abyss, The (1989)
Princess Bride, The (1987)
Untouchables, The (1987)
Casino (1995)
Heat (1995)
Princess Mononoke (Mononoke-hime) (1997)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Star Trek: First Contact (1996)
Hunt for Red October, The (1990)
Usual Suspects, The (1995)
```

User 95's recommendations consist of action, crime, and thrillers. None of their recommendations are comedies.

6. Discussions & Conclusion:

In this project, we used collaborative filtering with implicit feedback to generate movie recommendations for a user. We started by loading the MovieLens dataset and transforming it into a sparse matrix. We

also created movie title mappers to make it easier to interpret our results. We then built our implicit feedback recommender model using the Alternating Least Squares algorithm. We tested out our model by generating movie recommendations for a given movie and then generating personalized recommendations for a user.

The results of our model seem to be reasonable. The recommended movies are similar in genre to those of interest or the user's previously rated movies. However, one limitation of our model is that it is not able to recommend movies from a different genre than what the user has previously watched. In our example, the user did not receive any comedy recommendations despite having watched comedies in the past.

Overall, collaborative filtering with implicit feedback is a powerful technique for generating personalized recommendations. It works well with large, sparse datasets and can be used in a variety of applications beyond movie recommendations, such as music recommendations or product recommendations.

7. References:

1. How User-Based Collaborative Filtering Work?

Url: How user-based collaborative filtering works? Netflix movie recommendation simulation | Recommendation Systems (medium.com)

2. An Intro to Collaborative Filtering for Movie Recommendation

Url: An Intro to Collaborative Filtering for Movie Recommendation | by Destin Gong | Towards Data Science

3. Movielwns Dataset

Url: https://grouplens.org/datasets/movielens/latest/

4. A Brief Guide to Movie Recommendation Systems Using Machine Learning

Url:https://labelyourdata.com/articles/movie-recommendation-with-machine-learning#_content-based_filtering

Other References:

- Python 3: https://www.python.org/.

- Pandas: https://pandas.pydata.org/.
- NumPy: https://numpy.org/.
- Seaborn: https://seaborn.pydata.org/.
- Matplotlib: https://matplotlib.org/.
- K-Means Clustering: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html.

Appendix

The code can be found in the below-attached appendix:

Collaborative filtering using implicit feedback

Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
from scipy.sparse import csr_matrix

import implicit

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

Loading the Dataset

We are going to load the dataset "MovieLens" which has two of the following files

- ratings.csv
- movies.csv

In [2]:
```
ratings = pd.read_csv('/Users/bharath/Documents/ASm/ratings.csv')
movies = pd.read_csv('/Users/bharath/Documents/ASm/movies.csv')
```

In [3]:
```
ratings.head()
```

Out[3]:

	userId	movieId	rating	timestamp
0	1	1	4.0	964982703
1	1	3	4.0	964981247
2	1	6	4.0	964982224
3	1	47	5.0	964983815
4	1	50	5.0	964982931

Transforming the data

We are going to create a Function that generates a sparse matrix from ratings dataframe.

In [4]:
```
def create_X(df):
    N = df['userId'].nunique()
    M = df['movieId'].nunique()

    user_mapper = dict(zip(np.unique(df["userId"]), list(range(N))))
    movie_mapper = dict(zip(np.unique(df["movieId"]), list(range(M))))

    user_inv_mapper = dict(zip(list(range(N)), np.unique(df["userId"])))
    movie_inv_mapper = dict(zip(list(range(M)), np.unique(df["movieId"])))

    user_index = [user_mapper[i] for i in df['userId']]
    movie_index = [movie_mapper[i] for i in df['movieId']]

    X = csr_matrix((df["rating"], (movie_index, user_index)), shape=(M, N))

    return X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper
```

In [5]:
```
X, user_mapper, movie_mapper, user_inv_mapper, movie_inv_mapper = create_X(r
```

Creating Movie Title Mappers

In [6]:
```
!pip install thefuzz
from thefuzz import fuzz
from thefuzz import process

def movie_finder(title):
    all_titles = movies['title'].tolist()
    closest_match = process.extractOne(title,all_titles)
    return closest_match[0]

movie_title_mapper = dict(zip(movies['title'], movies['movieId']))
movie_title_inv_mapper = dict(zip(movies['movieId'], movies['title']))

def get_movie_index(title):
    fuzzy_title = movie_finder(title)
    movie_id = movie_title_mapper[fuzzy_title]
    movie_idx = movie_mapper[movie_id]
    return movie_idx

def get_movie_title(movie_idx):
    movie_id = movie_inv_mapper[movie_idx]
    title = movie_title_inv_mapper[movie_id]
    return title
```

```
Requirement already satisfied: thefuzz in /Users/bharath/opt/anaconda3/lib/p
ython3.9/site-packages (0.19.0)
```

```
/Users/bharath/opt/anaconda3/lib/python3.9/site-packages/thefuzz/fuzz.py:11:
UserWarning: Using slow pure-python SequenceMatcher. Install python-Levensht
ein to remove this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Leve
nshtein to remove this warning')
```

It's time to test it out! Let's get the movie index of Legally Blonde.

In [7]:
```
get_movie_index('Legally Blonde')
```

Out[7]:
```
3282
```

Let's pass this index value into get_movie_title(). We're expecting Legally Blonde to get returned.

In [8]:
```
get_movie_title(3282)
```

Out[8]:
```
'Legally Blonde (2001)'
```

Building Our Implicit Feedback Recommender Model

In [9]:
```
model = implicit.als.AlternatingLeastSquares(factors=50)
```

```
WARNING:root:Intel MKL BLAS detected. Its highly recommend to set the enviro
nment variable 'export MKL_NUM_THREADS=1' to disable its internal multithrea
ding
```

fitting our model with our user-item matrix.

In [10]:
```
model.fit(X)
```

```
  0%|          | 0/15 [00:00<?, ?it/s]
```

Now, let's test out the model's recommendations. We can use the model's similar_items() method which returns the most relevant movies of a given movie. We can use our helpful get_movie_index() function to get the movie index of the movie that we're interested in.

In [11]:
```
movie_of_interest = 'forrest gump'

movie_index = get_movie_index(movie_of_interest)
related = model.similar_items(movie_index)
related
```

Out[11]:
```
[(314, 1.0),
 (277, 0.92106205),
 (510, 0.85805476),
 (257, 0.85661566),
 (461, 0.76611966),
 (97, 0.7537822),
 (418, 0.7102524),
 (43, 0.6950457),
 (123, 0.6937208),
 (46, 0.6629923)]
```

The output of similar_items() is not user-friendly. We'll need to use our get_movie_title() function to interpret what our results are.

In [12]:
```
print(f"Because you watched {movie_finder(movie_of_interest)}...")
for r in related:
    recommended_title = get_movie_title(r[0])
    if recommended_title != movie_finder(movie_of_interest):
        print(recommended_title)
```

```
Because you watched Forrest Gump (1994)...
Shawshank Redemption, The (1994)
Silence of the Lambs, The (1991)
Pulp Fiction (1994)
Schindler's List (1993)
Braveheart (1995)
Jurassic Park (1993)
Seven (a.k.a. Se7en) (1995)
Apollo 13 (1995)
Usual Suspects, The (1995)
```

Generating User-Item Recommendations

In [13]:
```
user_id = 95
```

In [14]:
```
user_ratings = ratings[ratings['userId']==user_id].merge(movies[['movieId',
user_ratings = user_ratings.sort_values('rating', ascending=False)
print(f"Number of movies rated by user {user_id}: {user_ratings['movieId'].n
```

```
Number of movies rated by user 95: 168
```

User 95 watched 168 movies. Their highest rated movies are below:

In [15]:
```
user_ratings = ratings[ratings['userId']==user_id].merge(movies[['movieId',
user_ratings = user_ratings.sort_values('rating', ascending=False)
top_5 = user_ratings.head()
top_5
```

Out[15]:

	userId	movieId	rating	timestamp	title
24	95	1089	5.0	1048382826	Reservoir Dogs (1992)
34	95	1221	5.0	1043340018	Godfather: Part II, The (1974)
83	95	3019	5.0	1043340112	Drugstore Cowboy (1989)
26	95	1175	5.0	1105400882	Delicatessen (1991)
27	95	1196	5.0	1043340018	Star Wars: Episode V - The Empire Strikes Back...

Their lowest rated movies:

In [16]:
```
bottom_5 = user_ratings[user_ratings['rating']<3].tail()
bottom_5
```

Out[16]:

	userId	movieId	rating	timestamp	title
93	95	3690	2.0	1043339908	Porky's Revenge (1985)
122	95	5283	2.0	1043339957	National Lampoon's Van Wilder (2002)
100	95	4015	2.0	1043339957	Dude, Where's My Car? (2000)
164	95	7373	1.0	1105401093	Hellboy (2004)
109	95	4732	1.0	1043339283	Bubble Boy (2001)

We'll use the recommend() method, which takes in the user index of interest and transposed user-item matrix.

In [17]:
```
X_t = X.T.tocsr()

user_idx = user_mapper[user_id]
recommendations = model.recommend(user_idx, X_t)
recommendations
```

Out[17]:
```
[(855, 1.2092698),
 (898, 0.9775134),
 (1644, 0.9395924),
 (15, 0.89407647),
 (5, 0.80936944),
 (2258, 0.8077935),
 (3633, 0.80654395),
 (1043, 0.80218315),
 (1210, 0.784987),
 (46, 0.76805365)]
```

We can't interpret the results as is since movies are represented by their index. We'll have to loop over the list of recommendations and get the movie title for each movie index.

In [18]:
```
for r in recommendations:
    recommended_title = get_movie_title(r[0])
    print(recommended_title)
```

```
Abyss, The (1989)
Princess Bride, The (1987)
Untouchables, The (1987)
Casino (1995)
Heat (1995)
Princess Mononoke (Mononoke-hime) (1997)
Lord of the Rings: The Fellowship of the Ring, The (2001)
Star Trek: First Contact (1996)
Hunt for Red October, The (1990)
Usual Suspects, The (1995)
```

User 95's recommendations consist of action, crime, and thrillers. None of their recommendations are comedies.