Movie Recommendation System

1 Introduction

Movies hold universal appeal, connecting people of all backgrounds. Despite this unity, our individual movie preferences remain distinct, ranging from specific genres like thrillers, romance, or sci-fi to focusing on favorite actors and directors. Our aim for this project is to suggest movies tailored to individual tastes thereby enhancing the movie-watching experience. In this regard, we develop the **Movie Recommendation System**.

1.1 Recommendation System

A Recommendation System is a filtration program whose prime goal is to predict the *rating* or *preference* of a user towards a domain-specific item. In this project, the domain-specific item is movie. Therefore, the main focus of the movie recommendation system is to filter and predict only those movies which a user potentially would like to watch.

1.2 Types of Recommendation Systems

Most of the recommendation systems are based on the following approaches :-

1. Content-based filtering

Content-based filtering is a recommendation strategy that suggests items similar to those a user has previously liked. It calculates similarity between the user's preferences and item attributes. However, content-based filtering limits exposure to different products, preventing users from exploring a variety of items.

2. Collaborative filtering

Collaborative filtering is a recommendation strategy that considers the user's behavior and compares it with other users in the database. It uses the history of all users to influence the recommendation algorithm. Unlike content-based filtering, collaborative filtering relies on the interactions of multiple users with items to generate suggestions and doesn't solely depend on one user's data for modeling.

There are 2 ways in which collaborative filtering algorithms are implemented :-

a) User-based Collaborative filtering

User-based collaborative filtering aims to recommend items to a user based on the preferences of similar users in the database. It involves computing similarity scores between users, and then suggesting items that one user hasn't encountered yet but similar users have liked.

For example, if user A likes *Batman Begins*, *Justice League*, and *The Avengers*, and user B likes *Batman Begins*, *Justice League*, and *Thor*, they share similar interests in the superhero genre. Therefore, there is a high likelihood that user A would enjoy *Thor*, and user B would like *The Avengers*.

b) Content-based Collaborative Filtering

Item-based Collaborative Filtering focuses on finding similar movies instead of similar users to recommend to a user based on their past preferences. It identifies pairs of movies rated/liked by the same users, measures their similarity across all users who rated both, and then suggests similar movies based on the similarity scores.

For example, when comparing movies A and B, we analyze the ratings given by users who rated both movies. If these ratings show high similarity, it indicates that A and B are similar movies. Thus, if someone liked A, they should be recommended B, and vice versa.

2 Our Approach

For this project, we are using **Content based Collaborative filtering**. So basically, the user feeds the name of a movie that he likes (Eg. *Godfather*) and the recommendation system would suggest 10 other movies that are similar to the given movie which the user may also like. This recommendation system makes use of the **K-Nearest Neighbour algorithm** and we have used an open source database named **MovieLens-small** for this project.

2.1 K-Nearest Neighbour Algorithm

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. In our case, KNN algorithm is utilized to find out the most similar movies to the given movie. This similarity between two data samples is computed using different metrics such as **Cosine Similarity** or **Euclidean distance**.

2.2 Dataset Preprocessing

We have used **MovieLens-small** dataset for training the movie recommendation system. However, this dataset has two problems that prevent us from using this dataset in the current state.

1. Missing Value

Not every user rates all the movies present in the dataset resulting in several missing values in the dataset. These missing values impede the KNN Algorithm from determining the most similar movies. In order to resolve this issue, we impute all the missing values with 0.

2. Sparsity

Real-world ratings are usually very sparse and data points are mostly collected from very popular movies and highly engaged users. However, movies that were rated by a small number of users are unreliable to be included in the training of KNN Algorithm. Similarly, users who have rated only a handful of movies should also not be taken into account. In order to remove this noise from the dataset, we apply the following filters on the dataset :-

1. To qualify a movie, a minimum of 10 users should have rated a movie
2. To qualify a user, a minimum of 50 movies should have rated by the user

Inspite of this, the dataset still has some amount of sparsity. To reduce the sparsity we use the `csr_matrix` function from the `scipy` library that creates a sparse matrix of compressed sparse row format.

2.3 Recommendation System

Now that all things are in place, we complete our recommendation system. The user provides the title of a movie as input to the recommendation system. We first check if that movie is present in the database and if it is then we use the KNN Algorithm to find similar movies and sort them based on their similarity distance and output only the top 10 movies.