ML Final Assignment- Movie recommendation algorithms

Introduction

Due to the vast amount of data available on the internet, it has become nearly impossible to navigate without assistance. Recommendation engines have provided a solution to this problem by suggesting relevant items to users. These engines have become essential in various industries, including online video streaming, as they help users make informed decisions amidst the sea of data. There are two types of recommendation systems: collaborative and content-based methods. Collaborative engines rely on past user interactions to predict future behavior, which are stored and used to generate predictions. Collaborative filtering algorithms can be memory-based or model-based, with the latter assuming a generative model to explain interactions. While these algorithms do not require additional user information and can develop a better model with repeated interactions, they struggle to predict the behavior of new users, known as the "cold start problem."

Content-based filtering involves using additional information about users or items to make recommendations. By incorporating factors such as the user's age or gender, or the category and director of a movie, the model attempts to build a more accurate understanding of user-item interactions. Unlike collaborative filtering, content-based methods can avoid the "cold start problem" by requiring only a basic user profile. However, in this discussion, we will not be exploring content-based filtering and will instead focus on comparing different collaborative filtering algorithms.

Online video providers, such as Netflix and Amazon, rely heavily on efficient movie recommendation systems to suggest relevant movies to their users. This is important to maintain customer engagement and interest in the service, as the sheer number of choices available can be overwhelming. In this paper, we attempt to determine the best approach to movie recommendations by building different recommendation systems using small movielens data provided by GroupLens Research. The models we investigate include Singular Value Decomposition (SVD), K-Nearest Neighbors, Co-Clustering and a Neural Network Approach. By training these models on 100,000 ratings data points from the MovieLens dataset, we aim to identify what makes a strong recommendation model and how to achieve it.

My goal is to recommend movies to users that are worth their time and effort, in order to provide the best movie-watching experience possible. This project is intended to help me learn about recommendation algorithms that could be useful in my future role as a business analyst.

Collaborative filtering algorithms

Collaborative filtering is widely used in recommendation systems that leverages the patterns and preferences of users to make predictions or recommendations for other users. In the context of movie recommendation engines, collaborative filtering works by analyzing the ratings and preferences of users for movies, and then using this information to recommend movies to users who have not yet watched them.

There are two main types of collaborative filtering:

1. User-based collaborative filtering: This approach looks for users who have similar preferences to a given user and recommends movies that those similar users have rated highly. The idea is that if two users have similar preferences for movies, then a movie that one user enjoyed will likely also be enjoyed by the other user.

2. Item-based collaborative filtering: This approach looks for movies that are similar to those that a given user has already rated highly and recommends those similar movies. The idea is that if a user enjoyed a particular movie, then they are likely to enjoy other movies that are similar to it.

Here is an example of how collaborative filtering can be used in a movie recommendation engine:

Suppose we have a dataset of movie ratings by users, where each row represents a user and each column represents a movie, and the values in the cells represent the rating given by the user for that movie. We want to recommend new movies to a user based on their previous ratings.

First, we can use user-based collaborative filtering to identify users who have similar preferences to the given user. This can be done by calculating the similarity between each pair of users based on their ratings for the same movies, using a similarity metric such as cosine similarity.

Once we have identified similar users, we can recommend movies that those users have rated highly but the given user has not yet watched. For example, if User A has watched and rated highly movies X, Y, and Z, and User B has also watched and rated highly movies X and Y, but not Z, then we can recommend movie Z to User A based on the fact that User B, who has similar preferences to User A, enjoyed it.

Alternatively, we can use item-based collaborative filtering to identify movies that are similar to those that the given user has already rated highly. This can be done by calculating the similarity between each pair of movies based on the ratings given by all users, again using a similarity metric such as cosine similarity.

Once we have identified similar movies, we can recommend those similar movies to the given user. For example, if User A has watched and rated highly movies X, Y, and Z, and movies X and Y are similar to movie W based on the ratings given by all users, then we can recommend movie W to User A based on the fact that it is similar to movies that User A has already enjoyed.

Building Models

Using KNN

The k-NN model is trained using the cosine similarity metric and the brute force algorithm. The cosine similarity between two users is a measure of the similarity between their movie ratings, and the brute force algorithm simply computes the distances between all pairs of users.

The cosine similarity between two users u and v is computed as follows:

$$\cos(u,v) = (u \,.\, v) / (||u||\ ||v||)$$

where u . v is the dot product of the movie ratings of u and v, and ||u|| and ||v|| are the Euclidean norms of the movie ratings of u and v, respectively.

The dot product (u . v) is computed as the sum of the element-wise product of the movie ratings of u and v:

$$u \,.\, v = \mathrm{sum}(ui * vi) \text{ for all } i$$

where ui is the rating of movie i by user u, and vi is the rating of movie i by user v.

The Euclidean norm of a vector x is computed as the square root of the sum of the squares of its elements:

$$||x|| = \mathrm{sqrt}(\mathrm{sum}(xi\hat{}2)) \text{ for all } i$$

where xi is the i-th element of vector x.

Once the cosine similarity between all pairs of users is computed, the k-NN algorithm finds the k users who are most similar to the target user. The algorithm then recommends movies that those k users liked by computing the average rating for each movie among those k users.

Singular value decomposition

Secondly, we have also used the SVD (Singular Value Decomposition) algorithm to build a movie recommendation engine using collaborative filtering. The SVD algorithm is a matrix factorization method that decomposes a user-movie rating matrix R into three matrices:

- U (m x k) represents the user latent factors, where m is the number of users and k is the number of latent factors chosen by the algorithm.
- S (k x k) is a diagonal matrix containing the singular values of R.
- V (n x k) represents the movie latent factors, where n is the number of movies.

The predicted rating of user u for movie i can then be calculated as the dot product of the corresponding row in U and column in V, which gives the user-movie rating matrix R_hat: R_hat = U x S x V^T where V^T is the transpose of V.

The SVD algorithm is trained by minimizing the sum of the squared errors between the actual ratings R and the predicted ratings R_hat. This is done using gradient descent with regularization to prevent overfitting.

Once the SVD algorithm is trained on the user-movie rating matrix, it can be used to make movie recommendations for a given user. For example, the code used in our model makes movie recommendations for user 1 by predicting the ratings for all the movies the user has not rated using the trained SVD algorithm, and then selecting the top 10 movies with the highest predicted ratings.

Co-clustering technique

Co-clustering was also incorporated for building movie recommendation engines. In this technique, both the users and the items are clustered simultaneously. The idea is to group together similar users and similar items, and then use this information to make predictions for a specific user-item combination.

For instance, consider a dataset consisting of n users and m items. We represent the user-item rating matrix as R of size n x m, where each element R_ij represents the rating given by user i to item j. In co-clustering, both the rows and columns of the matrix R are clustered into r and c clusters, respectively. Let U be a matrix of size n x r, where each row corresponds to a user and each column corresponds to a cluster. Similarly, let V be a matrix of size m x c, where each row corresponds to an item and each column corresponds to a cluster.

The co-clustering algorithm tries to find the optimal U and V matrices that minimize the following objective function:

Min (||R - U * S * V^T||^2_F)

where S is a diagonal matrix of size r x c, and ||.||^2_F denotes the Frobenius norm. The diagonal elements of S represent the strengths of the clusters, and are used to adjust the contribution of each cluster to the final prediction.

Once the co-clustering model is trained, we can use it to make recommendations for a given user. To do this, we first find the movies that the user has not rated, and then predict the ratings for these movies using the learned co-clustering model. We then recommend the top-rated movies to the user.

Building a neural network

The neural network model uses two inputs, the encoded user IDs and movie titles, and one output, the predicted rating. The model learns to predict the ratings based on the patterns in the training data. The training process involves adjusting the weights of the model based on the MSE loss function, which measures the difference between the predicted ratings and the actual ratings in the training set. The early stopping callback is used to stop the training process early if the validation loss does not improve for five consecutive epochs.

This neural network model is a type of regression model that predicts a continuous value, in this case, the rating.The model consists of three layers, an input layer, a hidden layer, and an output layer. The input layer has two nodes, representing the user ID and the movie ID. The hidden layer has 64 nodes, and the activation function used is ReLU (Rectified Linear Unit). The dropout layer is added after the first hidden layer to prevent overfitting, which drops out 50% of the randomly selected nodes. The second hidden layer has 32 nodes with a ReLU activation function, followed by another dropout layer. The output layer consists of a single node, which is used to output the predicted rating.The neural network model is trained using the mean squared error (MSE) loss function and the Adam optimizer with a learning rate of 0.001. The early stopping callback is used to stop training the model when the validation loss stops improving.

Results and conclusion

Algorithm	RMSE Scores
KNN	0.45
Co-clustering	0.95
SVD	0.87
Neural Networks	0.99

Based on the RMSE scores obtained, the KNN algorithm appears to be the best performing algorithm for movie recommendation using collaborative filtering, with an RMSE score of 0.45.However, KNN was predicting movie ratings of zero, which does not really make sense,It's possible that KNN was overfitting, Increasing the number of nearest neighbors could not still help reduce overfitting and improve the performance of the model on new data. Additionally, using techniques such as cross-validation to tune the hyperparameters of the model might help to prevent overfitting.

The SVD algorithm also performed relatively well with an RMSE score of 0.87, while the Co-clustering algorithm had a higher RMSE score of 0.95. The neural network algorithm had the highest RMSE score of 0.99, indicating that it may not be the most effective algorithm for this type of recommendation system.However, in future, I can explore the applications of neural collaborative filtering algorithms for user recommendation systems

References:

Vijay Kotu, Bala Deshpande, “Recommendation Engines.” Data Science (Second Edition). Morgan Kaufmann. 2019. Pages 343-394.
https://doi.org/10.1016/B978-0-12-814761-0.00011-3.

“MovieLens.” GroupLens. 2021. https://grouplens.org/datasets/movielens/.

JJ. “MAE and RMSE: Which Metric is Better?”
https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better-e60ac3bde13d.

“Co-clustering.” ML Wiki. http://mlwiki.org/index.php/Co-Clustering.
16) “Collaborative Filtering: Co-clustering.” Data Science Made Simpler.

“Recommendation Systems.” Google.
https://developers.google.com/machine-learning/recommendation/overview/candidate-generation.