Data Quester’s Movie Recommendation System

Introduction

In today’s entertainment landscape, movies shape our perspectives and emotions. However, with an overwhelming variety of films across platforms, languages, and genres, choosing what to watch can be challenging for users and costly for streaming platforms. Platforms risk losing users and revenue if viewers struggle to find enjoyable content. This underscores the value of intelligent movie recommendation systems in improving user satisfaction and driving business success.

Machine Learning-based Recommender Systems address these challenges by offering personalized movie suggestions tailored to individual preferences. These systems analyze user profiles (e.g., age, gender), viewing history, and movie attributes (e.g., genre, director, ratings) while factoring in platform interactions like clicks and watch duration. The result is a dynamic and intuitive experience, akin to a friend suggesting movies you’ll love.

Recommendation Systems

Collaborative

Collaborative filtering is a machine learning model that utilizes other people’s feedback that have **similar user behaviors** to provide recommendations to you. In our case, it is taking in other user’s ratings and feedback on movies in our dataset, which is also provided in our datasets, and providing recommendations based on that feedback from users that have similar behavior to whoever is using our model.

Content-Based

On the other hand, content-based filtering is focused on the **item itself**, and in our model, it is focused on the movies and their **features** (i.e. genre, actors, release year, etc.). A content-based filtering model will recommend other movies based on such features rather than similar users' streaming behavior.

Data Sets

To evaluate our approach, we utilized the MovieLens dataset, starting with its small version, which includes 610 unique users and 9,724 unique movies, to develop and test our models. For those interested in further exploration, the system can also be applied to the larger dataset containing 200,948 users and 87,585 unique movies, enabling broader insights and scalability.

Tools and Technology Used:

Make sure to have these tools and libraries installed for the various models to function correctly.

- Python (jupyter notebook)
- MongoDB Atlas
- Pandas
- PyMongo
- Scikit-learn (sklearn)
- Fuzzywuzzy
- Pprint (pretty print)
- Numpy
- Pathlib
- Matplotlib
- Datetime
- Tensorflow
- Collections (using counter)
- Itertools

How Our Models Work:

For [KNN recommender]

The model employs **collaborative filtering** to generate personalized recommendations by analyzing patterns of user interactions with movies.

To prepare the data for this approach, the dataset was transformed into a utility matrix where rows represent users and columns correspond to movie titles. This matrix serves as the foundation for identifying similar users or items. Four key mapping dictionaries were developed to streamline data handling:

- **Rating Mapper**: Links movie ratings to their respective row indices.
- **Movie Mapper**: Maps movie IDs to their corresponding column indices.
- **Ratings INV Mapper**: Reverses row indices back to ratings.
- **Movie INV Mapper**: Converts column indices back to movie IDs.

The utility matrix is implemented as a `scipy.sparse.csr_matrix`, optimized for efficient storage and computation with sparse data. To enhance the model's reliability, noise was reduced by including only users with at least 1,000 ratings and movies with at least 200 ratings, making the matrix denser and more representative of meaningful patterns.

Once prepared, the model was trained and evaluated by calculating the Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) for both the training and test datasets, ensuring its predictive accuracy. When provided with a movie ID, the model identifies similar movies based on cosine similarity in the user-item space, delivering tailored recommendations to the user.

For [Cosine Reccomender with Rating Predictor]
This model uses **content-based filtering** to recommend movies based on potential users' interests. We used cosine similarity scores to calculate how similar each movie is based on genre. The cosine similarity matrix was able to calculate approximately 9700 dimensions within the movie genres. We used the Python package fuzzywuzzy – which uses string-matching algorithms to find the closest title input – to build our function movie_finder. Movie_finder gives a list of titles that match the title input by the user and the movie IDs for this movie. We then use the calculated cosine similarity scores to check which movies are most similar to the input movie. The user can then select how many recommendations they want based on the input movie and receive that number of genre-based recommendations.

After that, we utilized an SVD model from sklearn to predict the user's rating before they watch a movie, based on others' and their previous ratings (**collaborative recommendation model**). Truncated Singular Value Decomposition (Truncated SVD) is a dimensionality reduction technique that is particularly useful for large, sparse datasets, like in our recommendation system. Truncated SVD is a variant of Singular Value Decomposition (SVD) that reduces the number of dimensions in the data while preserving as much information as possible. To prepare the data for our model, we created a user-item matrix with the rating information for each user and movie. Missing data was replaced with the mean score for that movie. The model was trained and evaluated by calculating the Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) for both the training and test datasets, ensuring its predictive accuracy. The trained model is reconstructed with the previously flattened components to return dimensionality. The original_id_finder function helps users get the movie ID for one of the recommended movies. The my_rating function provides the predicted rating for a specific user and movie based on the trained svd model; users can use it to predict how a similar user would rate a movie they were recommended above.

For [DBSCAN Clustering and Neural-network model]
This model is an unsupervised learning method that groups points based on similar properties. DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise. Each density of points creates a cluster if a neighborhood's radius contains the minimum number of points. DBSCAN is good for irregular data, like real-life data. The DBSCAN takes in an eps (epsilon), min_samples (minimum points), and metric parameters. For the eps, we experimented between 0.1 and 1. The Epsilon is the distance between two points to be considered neighbors. The distance must be lower or equal to the epsilon. The minimum point is the smallest number of data points that creates a neighborhood. For the metric we experimented between 'l1' and 'l2'. L1 regularization is a lasso regularization that inhibits overfitting. L2 is a ridge regularization that avoids overfitting.

For our movie data set, we wanted to figure out how users would rate a certain genre of movies based on their rating of other genres. To test, we separated "Thriller" genre movies from the rest of the movies. We then grouped the other movies into clusters based on the DBSCAN. Then, we fit our data into this model, labeled our data based on the cluster they were in, and added the average rating of thriller movies for that user. We then split and tested our data using a Keras neural network and evaluated our model based on the R-squared and MAE (Mean absolute error).

For [franchise SVC]
For this section, we created dataframes for films from the same franchise. Then, we used the SVC to test and train the model for each franchise. This will predict the ratings of future films within that franchise. A Support Vector Classifier (SVC) is a type of Support Vector Machine (SVM) used for classification tasks. SVMs are powerful supervised learning algorithms used for both classification and regression. They are particularly effective in high-dimensional spaces and are versatile due to their ability to use different kernel functions.

Analysis of Model Performance:

[KNN recommender]
Pros

- Simple to implement, works well with sparse data, and provides highly personalized, interpretable recommendations.
- Requires less training phase, making it quick to set up and suitable for small-scale systems.

Cons

- Some of the recommendations are less relevant – if looking at movie features – due the to nature of the recommendation (ratings-based).
- Suffers from the cold start problem and cannot learn latent features or patterns like advanced models.

[Cosine Reccomender with Rating Predictor]
Pros

- More relevant recommendation list as it is based on genres (relative to solely rating-based)
- A simpler design for recommendations, which allows for predicted ratings using SVD
- The rating predictor model has good MAE and RSME scores.

Cons

- Memory-intensive (i.e. cannot run larger versions of a dataset on the local device; recommend AWS or similar platform to run full dataset)
- Difficulty evaluating the recommender model mathematically due to it being based on cosine similarity scores, which is using a way to evaluate. Our team manually searched the recommended movies to gauge whether the movies were relevant to the movies we tested the model with.

[DBSCAN clustering]
Pros

- Can cluster arbitrary shapes/Good for real-life data.
- Number of clusters does not need to be specified in advance.
- DBSCAN can identify outliers and leave them out of clusters that they do not belong in.

Cons

- Can be sensitive to parameter choice (eps and minimum points)
- Difficult clustering with high density data.
- Not guaranteed to find all clusters within the data.

[Franchise SVC]

Pros

- This would be beneficial to production companies who are interested in making future films in a series. This test would also interest fans who are anticipating a new film in their favorite series.

Cons

- Because there are usually only 4-6 films per series, this significantly reduces the dataset. As a result, the predictions have lower accuracy ratings. Furthermore, ratings are not necessarily indicative of box office success, seeing that some franchise films with low average ratings generated a lot of revenue.

Conclusions

In our recommender system, we utilized two models to enhance user experience: **K-Nearest Neighbors (KNN)** for collaborative filtering and **Truncated Singular Value Decomposition (SVD)** for latent factor analysis. Through evaluation, we observed that the Truncated SVD model demonstrated a lower RMSE, indicating higher predictive accuracy. However, while the KNN model effectively recommends movies based on a given movie ID, the SVD model currently focuses on predicting movie ratings based on user ID and movie ID inputs. Expanding the SVD model to generate direct recommendations, similar to the KNN approach, would further enhance its utility.

To address the cold-start problem, where limited user or movie information is available, we implemented a cosine similarity metric based on genre features. This ensures that our system can still provide recommendations when user or item data is sparse.

The combination of these models provides a robust framework for personalized recommendations, but future work will focus on integrating the strengths of both models—precision from SVD and user-centric recommendations from KNN—while improving the handling of sparse data scenarios.

The DBSCAN was hard to manipulate the clustering. Having an eps score of 0.5 would produce a lot of clusters, meanwhile an eps = 1 would only produce 2 clusters with the majority of the data in 1 cluster. The model was able to produce good MAE scores around 0.25. This means that on average the difference between the actual rating and the predicted rating was 0.25. The

best R-squared produced from this model was over 0.59. This is not a great score as we would like this number to be as close to 1 as possible. 0.59 means that over half of the data was used to find the dependent or target variable.
The clustering and modeling can be manipulated in many ways to improve results. With the DBSCAN, we can try different eps scores, minimum samples, and different metric combinations to find the best clustering model. We tried using a grid search to get the best eps/min_samples combination, but a lot of the results did not improve the model we created. For the neural network, we can experiment with model layers and units per layer. We noticed that defining different activations (relu, sigmoid) hurt the model more. Also, another way to enhance the data may be to clean and process the data in a better manner. The data set is very large and could have incorrect information that may hurt the clustering and modeling. Also, we could try to single out a different genre rather than 'Thriller' and see if the clustering gets different results.

The SVC Franchise recommender is a start to our attempt to help users or stakeholders try to predict potential franchise reception by our user base.

Credits

Dataset:

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19. https://doi.org/10.1145/2827872.