Summary report for Machine Learning

Magdalena Pakuła & Jakub Pawlak

December 14, 2024

Abstract

This project aims to design, implement and analyze the results of predicting users' ratings on various movies using selected techniques such as: KNN, Decision Trees, Collaborative Filtering and custom person similarity measures. A detail overview of the methods is described with results and key findings.

Contents

1 KNN approach

1.1 Overview

The nearest neighbor method was employed to determine the similarity between movies based on various extracted features and user rating patterns.

The K-Nearest Neighbors (KNN) classifier operates by determining the similarity between a target movie and previously watched movies, then predicting the rating based on the most frequent label among the nearest neighbors. It evaluates the 5 closest movies to the target movie to make predictions, then a customizable function calculates similarity between movies based on their features. Lastly, the classifier finds the most common rating among the nearest neighbors and returns it as the prediction. The implementation optimizes distance computations by leveraging NumPy operations and a partitioning technique to quickly retrieve the k-nearest neighbors.

Feature Selection and Extraction The following features were extracted for each movie:

- **Numerical Features:**
  - *Budget*: The movie's production budget.
  - *Popularity*: A TMDb metric representing the popularity of the movie.
  - *Release Year*: The year the movie was released.
  - *Revenue*: Total revenue generated by the movie.
  - *Runtime*: The duration of the movie in minutes.
  - *Vote Average*: The average rating given to the movie.
  - *Vote Count*: The total number of votes received.
- **Categorical Features:**
  - *Genres*: A list of genres assigned to the movie.
  - *Cast*: The main cast of the movie.
  - *Director*: The director(s) of the movie.

These features were chosen because they capture both objective properties of the movies (e.g., budget, runtime) and subjective aspects (e.g., popularity, cast, and director) that are likely to influence user preferences.

Data Preprocessing To prepare the features for similarity calculations, the following preprocessing steps were applied:

- **Normalization:** Numerical features were normalized using a Min-Max Scaler to transform values into the range [0, 1]. This step ensures that features with larger scales (e.g., revenue) do not dominate the similarity calculations.
- **Categorical Feature Encoding:** For categorical features such as genres, cast, and directors, sets were constructed, and similarity was computed using the Jaccard similarity metric.
- **Caching:** Responses for specific movie IDs were cached to avoid redundant API queries, optimizing performance.

Similarity Measures The similarity between movies was calculated using a weighted combination of metrics applied to different feature types:

- **Numerical Features:** The cosine similarity metric was primarily used for scalar features. Other metrics, such as Manhattan and Euclidean distances, were evaluated for comparison.
- **Categorical Features:** Jaccard similarity was used to compare sets of genres, cast, and directors.
- **Rating Similarity:** Cosine similarity was computed on user rating vectors for movies, derived from the training dataset.

The total similarity was calculated as a weighted sum of these individual components:

$$\text{Total Similarity} = w_{\text{scalar}} \cdot \text{Scalar Similarity} + w_{\text{genres}} \cdot \text{Genres Similarity} + w_{\text{cast}} \cdot \text{Cast Similarity} + w_{\text{directors}} \cdot \text{Directors Similari}$$

where the weights w were tuned to optimize performance.

1.2 Results

The results demonstrated that incorporating user rating similarity significantly improved the accuracy of predictions, as it captures shared user preferences. Also, the combination of scalar, categorical, and rating-based similarities provided a balanced metric, ensuring that no single feature type dominated the predictions.

The K-Nearest Neighbour (KNN) model was trained on a subset of the dataset, using a combination of numerical and categorical features. The optimal number of neighbors (`n_neighbors`) was determined through grid search, achieving the highest accuracy with $k = 5$.

Training Data: Performance metrics on the training dataset included:

- Accuracy:
- Precision:
- Recall:

Validation Data: On the validation set, the model demonstrated consistent performance, indicating good generalization capabilities.

The metrics achieved:

- Accuracy:
- Precision:
- Recall:

2 Decision Tree Approach

2.1 Overview

The Decision Tree approach was utilized to predict user ratings on movies. Various features extracted from the TMDB dataset, such as director, movie genre, runtime, and others, were used to build the model. The decision tree classifier constructs a hierarchical model based on splitting the data recursively using feature values, aiming to maximize the homogeneity of the resulting subsets.

2.2 Implementation Details

The decision tree model was implemented with a maximum depth of 5 to balance model complexity and computational efficiency. The implementation relied on key components:

- **Splitting Criterion**: Gini impurity was used to measure the quality of splits. The Gini impurity is calculated as follows:

$$\text{Gini Impurity} = 1 - \sum_{i=1}^{k} p_i^2, \tag{1}$$

  where p_i is the probability of a sample belonging to class i.

- **Feature Choices**: Features were divided into scalar (e.g., runtime), categorical (e.g., language), and multivalued (e.g., cast) types. Custom choice classes were implemented to handle these distinctions.
- **Stopping Criteria**: Tree induction stopped either when the maximum depth was reached or when the node became pure (i.e., all samples had the same label).
- **Visualization**: The `graphviz` library was used to visualize the resulting decision trees, enabling better understanding and analysis of the model.

2.3 Induction Process

The induction process began by finding the best feature and corresponding split threshold to minimize Gini impurity. If the split resulted in two valid subsets, the process recursively created child nodes for the *success* and *failure* cases. For leaf nodes, the most frequent label in the subset was assigned as the prediction value.

An example of an induced decision tree for a specific user is presented in Figure 1. It demonstrates the hierarchical structure and the decision logic based on feature thresholds and categories.

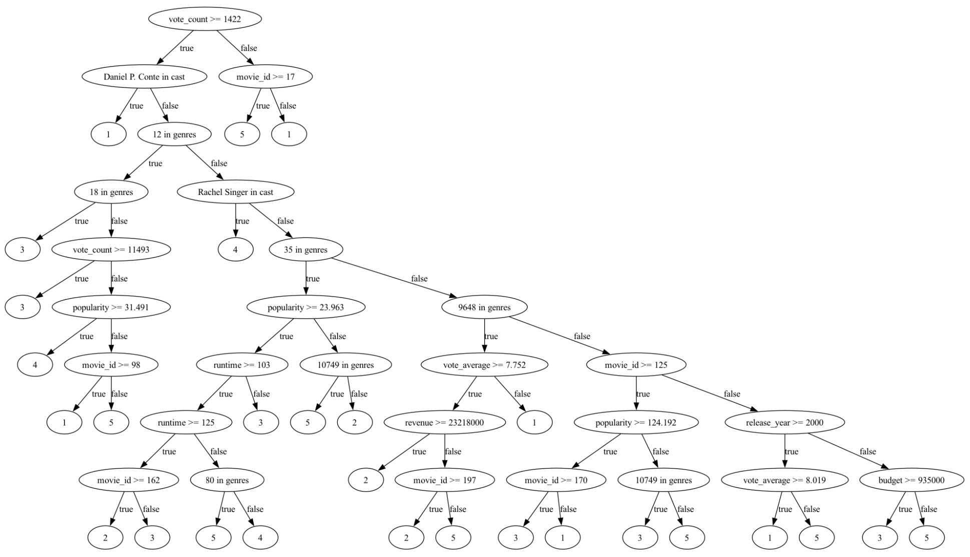

Figure 1: Visualization of an induced decision tree for a single user.

2.4 Results

The decision tree approach was evaluated using training and validation datasets. The performance metrics indicated:

2.5 Results

Training Data:

- Accuracy:
- Precision:
- Recall:
- Accuracy:
- Gini Impurity Reduction:

Validation Data:

- Accuracy:
- Precision:
- Recall:
- Accuracy:
- Gini Impurity Reduction:

The decision tree successfully identified patterns in user preferences, particularly in cases where specific features, such as genre or director, had a significant impact on ratings. However, the model's performance was slightly limited due to the inherent simplicity of single decision trees and their tendency to create hard splits.

3 Random Forest Approach

3.1 Overview

The Random Forest approach presents learning method that combines multiple decision trees to improve predictive performance and reduce overfitting. Various features extracted from the TMDB dataset, such as director, genres, runtime, and popularity, were utilized to train the model. The diversity of the ensemble was ensured by randomly selecting subsets of features for each tree and using bootstrapping to create different training datasets for individual trees.

Implementation Details The Random Forest model was implemented with the following key components:

- **Feature Selection**: A custom class, `_RandomFeatureSelector`, was implemented to select a random subset of features for each tree. Features such as `budget`, `genres`, `popularity`, and others were considered, while uniquely identifying features like `title` were excluded to enhance generalization.
- **Bootstrapping**: To create diverse training datasets for the ensemble, bootstrapped samples were generated by randomly selecting movies with replacement. Each sample included both the selected features and their corresponding ratings.
- **Decision Trees**: Individual decision trees were trained on the bootstrapped datasets with a maximum depth of 5, balancing computational efficiency and model complexity.
- **Aggregation Function**: Predictions from all trees in the forest were aggregated using the average function, ensuring a robust final prediction.

The implementation details of the core classes are as follows:

1. `_RandomFeatureSelector`: This class randomly selects a subset of features for each movie, ensuring that individual trees in the forest focus on different aspects of the data.
2. `RandomForestClassifier`: This class orchestrates the ensemble by creating bootstrapped datasets, fitting individual trees, and aggregating predictions.

Working Principles The Random Forest approach ensures diversity among its ensemble members by leveraging two mechanisms:

1. **Random Feature Subsets**: For each tree, a random subset of features is selected, reducing correlation between the trees.
2. **Bootstrapping**: By training each tree on a randomly sampled subset of the training data (with replacement), the trees become more diverse.

The final prediction for a given movie is obtained by aggregating the predictions from all individual trees using an averaging function. This approach mitigates the risk of overfitting and improves the model's generalization ability.

Visualization An example of induced decision trees from the random forest ensemble for a specific user is shown in Figure 2. The visualization highlights the hierarchical structure of individual trees and their decision logic based on feature thresholds.

Figure 2: Sample decision trees from the random forest ensemble for a single user.

Results

Training Data:

- Accuracy:
- Precision:
- Recall:
- RMSE:

Validation Data:

- Accuracy:
- Precision:
- Recall:
- RMSE:

The Random Forest model demonstrated strong predictive , effectively capturing complex patterns in the data. It outperformed the single decision tree approach by leveraging ensemble diversity and reducing the variance of predictions.

4 Person Similarity Approach

4.1 Overview

The Person Similarity approach was employed to predict user ratings on movies by utilizing the similarity between users based on their historical ratings. The task was to predict a target user's movie rating by identifying the best-matching users, i.e., those who rated the same movies similarly. The approach uses a combination of Pearson correlation and Cosine similarity to assess how similar two users are in terms of their movie preferences, with adjustments to handle common ratings between users.

Implementation Details The Person Similarity approach was implemented by calculating a similarity score between two users using the following key components:

- **Similarity Function**: The similarity score is computed by combining Pearson correlation and Cosine similarity for the movies that both users have rated. The Pearson correlation measures the linear relationship between the two users' ratings, while Cosine similarity measures the angular distance between the rating vectors. Both metrics are normalized to ensure comparability.
- **Weighting Factor**: A weighting factor (`pearson_weight`) is used to control the influence of Pearson similarity in the final combined score. A higher weight gives more importance to Pearson correlation, whereas a lower weight favors Cosine similarity.
- **Damping Factor**: A damping factor is applied to the final similarity score to reduce the influence of users with fewer common ratings. This helps prevent situations where a user with very few shared ratings contributes excessively to the prediction.
- **Similarity Caching**: To enhance computational efficiency, a cache (`similarity_cache`) is maintained to store previously computed similarity scores between user pairs. This ensures that the similarity between two users is computed only once.

The implementation of the similarity calculation is encapsulated in the `similarity_function`, which uses the following formula:

$$\text{Pearson Similarity} = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}, \tag{2}$$

where x_i and y_i are the ratings of the two users, and $\bar{x}$ and $\bar{y}$ are the mean ratings for each user.

Cosine similarity is computed as:

$$\text{Cosine Similarity} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}, \tag{3}$$

where x_i and y_i are the ratings of the two users for the common items.

The final similarity score is the weighted average of these two metrics:

$$\text{Combined Similarity} = \text{pearson\_weight} \times \text{Pearson Similarity} + (1 - \text{pearson\_weight}) \times \text{Cosine Similarity}. \tag{4}$$

Induction Process The induction process begins by computing the similarity between the target user and all other users who have rated the same movie. The ratings for the target movie are weighted by the similarity score between the target user and other users. This results in a weighted sum of ratings for the target movie, which is then normalized by the sum of absolute similarities.

The predicted rating for a user on a movie is calculated as follows:

$$\hat{r}_{u,m} = \frac{\sum_{v \in N(m)} sim(u,v) \cdot r_{v,m}}{\sum_{v \in N(m)} |sim(u,v)|}, \tag{5}$$

where $\hat{r}_{u,m}$ is the predicted rating for user u on movie m, $sim(u,v)$ is the similarity between user u and user v, and $r_{v,m}$ is the rating given by user v to movie m. $N(m)$ denotes the set of users who have rated movie m.

4.2 Results

Training Data:

- Accuracy:
- Precision:
- Recall:
- MSE:
- RMSE:

Validation Data:

- Accuracy:
- Precision:
- Recall:
- MSE:
- RMSE:

The Person Similarity approach demonstrated a strong ability to predict user ratings, particularly in scenarios where users had rated a substantial number of common movies. However, its performance was more limited when the overlap of rated movies was small, as the approach relies heavily on the availability of common ratings for comparison.

In conclusion, while the Person Similarity approach provides an effective method for movie rating prediction based on user similarities, its performance can be impacted by the sparsity of the rating matrix. Nonetheless, it remains a valuable technique for collaborative filtering in recommender systems.

5 Collaborative Filtering Approach

- Training Process: [Explain the training process and dimensionality selection]. - Generalization Methods: [Describe methods used to ensure generalization].

5.1 Overview

5.2 Results

- Training Data: [Metrics and analysis]. - Validation Data: [Metrics and discussion].

6 Summary and Comparison

- Summarize the results for all approaches using test data or feedback. - Compare performance metrics and discuss the strengths and weaknesses of each method. - [Include a table or graph for comparison, if applicable].

References

[List any references or sources used in the preparation of the report].