Report - Recommendation System

04/28/2024

1. Preprocessing:

1.1. Importing Libraries:

Firstly we need to install the scikit-surprise library and import necessary modules for collaborative filtering and recommendation systems. It utilizes Surprise for matrix factorization-based recommendation, including SVD (Singular Value Decomposition), and sklearn for preprocessing and nearest neighbor-based recommendation. Additionally, it imports modules for data visualization. This code sets the stage for building recommendation systems, preprocessing data, and evaluating models through cross-validation and parameter tuning.

1.2. Reading the Dataset:

Then we read movie and user data from CSV files into Pandas DataFrames, specifying column names and delimiters as needed. The movie data includes attributes like title, genre, and release date, while the user data includes information such as age, gender, occupation, and zip code. Additionally, it reads multiple rating datasets (training and testing sets) for collaborative filtering. Each rating dataset contains columns for user ID, item (movie) ID, rating, and timestamp. These data are essential for building and evaluating recommendation systems based on user preferences and interactions with movies.

1.3. Combined split train and test dataset:

We concatenate training data from five datasets df1, df3, df5, df7, and df9 (uk.base) into one DataFrame 'df', and testing data from five other datasets df2, df4, df6, df8, and df10 (uk.test) into another DataFrame 'df_test', effectively splitting the data into an 80-20 train-test ratio for model training and evaluation, where k is user 1 to 5.

1.4. Merging the features:

We merge the 'df' DataFrame containing training data with the 'movies' DataFrame based on the 'item_id' and 'movie_id' columns, incorporating movie attributes. Then, it merges the result with the 'users' DataFrame based on the 'user_id' column, adding user attributes to the training data.

1.5. Adding new features:

We first drop unnecessary columns ('timestamp', 'movie_id', 'IMDb_URL', 'video_release_date') from the DataFrame 'df'. Then, it removes rows with missing values. It converts the 'release_date' column to datetime format, extracting the day of the week and storing it in a new column 'day_of_week'. Finally, it drops the original 'release_date' column, effectively preprocessing the data by removing unnecessary information, handling missing values, and extracting relevant temporal features for further analysis or modeling.

1.6. Scaling features:

For scaling, we utilize LabelEncoder from scikit-learn to transform categorical variables ('gender', 'occupation', 'zip_code') into numerical values. It assigns unique numerical labels to each category within each column, enabling machine learning algorithms to process categorical data effectively by converting them into a numerical format.

2. Evaluation metrics:

2.1. Evaluating method - custom:

The code is performing model evaluation for collaborative filtering using Singular Value Decomposition (SVD) with the Surprise library. It imports evaluation metrics from Surprise and scikit-learn. KFold cross-validation with 5 splits is utilized for robust evaluation. Within the loop, each fold's training set is used to fit the SVD model, and predictions are made on the test set. A threshold of 3.5 is applied to convert predicted ratings to binary values (1 for predicted ratings >= 3.5, 0 otherwise). Accuracy is calculated by comparing true ratings with binary predictions, while RMSE (Root Mean Squared Error) is computed between true ratings and predicted ratings. For each fold, accuracy and RMSE are appended to corresponding lists. Finally, the mean and standard deviation of accuracy and RMSE across folds are calculated and printed as evaluation results. This process assesses the model's ability to correctly predict user preferences (accuracy) and the precision of predicted ratings compared to actual ratings (RMSE). It's essential for understanding the model's performance and its potential application in recommendation systems.

2.2. Evaluating with cross_validate:

The `cross_validate` function evaluates the performance of the SVD model using the Surprise library. It takes the model (`svd`), test data (`data_test`), evaluation measures (in this case, 'RMSE' and 'MAE' for Root Mean Squared Error and Mean Absolute Error), and the number of cross-validation folds (5 in this case). During evaluation, the test data is split into 5 folds, with each fold serving as a test set while the remaining data is used for training. The model is trained and tested on each fold, and performance metrics (RMSE and MAE) are computed. Finally, the average RMSE and MAE across all folds are returned as evaluation results.

3. Helper Functions:

3.1. Collaborative filtering for movie recommendations:

The function `generate_recommendation` generates movie recommendations for a given user based on collaborative filtering using a trained model. It first identifies all unique movie IDs in the ratings dataframe (`ratings_df`) and the movie IDs that the user has already rated. Then, it computes the set difference to obtain movie IDs that the user has not yet rated. For each of these unrated movies, a test set is

constructed with a fixed rating (e.g., 4, indicating a high likelihood of preference). The model then predicts ratings for these movies using the `model.test()` method. Predicted ratings are extracted from the predictions. The function prints the top `n_items` recommendations for the user, sorted by predicted rating percentile. It identifies the indices of the highest predicted ratings using argsort, and then retrieves the corresponding movie titles from the movies dataframe (`movies_df`). Finally, it prints the movie titles along with their predicted rating percentiles. This function provides a straightforward way to generate personalized recommendations for users based on their historical ratings and the collaborative filtering model's predictions.

3.2. Item-based filtering for movie recommendations:

The function, `get_movie_recommendation`, aims to provide movie recommendations based on a given movie's title. Initially, the input movie name is transformed using a LabelEncoder (`movie_label`) to obtain its numerical representation. Then, it retrieves the corresponding item ID from the dataset (`final_dataset`). If the movie exists in the dataset, its index is determined. Using the k-nearest neighbors (KNN) algorithm (`knn`), the function finds the most similar movies to the input movie based on their feature vectors (here, cosine similarity is likely used). It retrieves the indices and distances of the nearest neighbors. The function then constructs a DataFrame (`recommend_frame`) containing the recommended movies along with their distances from the input movie. It iterates over the recommended movie indices, retrieves their titles using the inverse transformation of the LabelEncoder, and appends them to the DataFrame. Finally, the function returns the DataFrame containing the recommended movies along with their distances. If the input movie is not found in the dataset, it returns a message indicating no movies were found. Overall, this function efficiently generates movie recommendations based on the similarity of movies' features, facilitating personalized recommendations for users based on their preferred movies.

4. Recommendation Systems:

4.1. Collaborative Filtering:

4.1.1. Description:

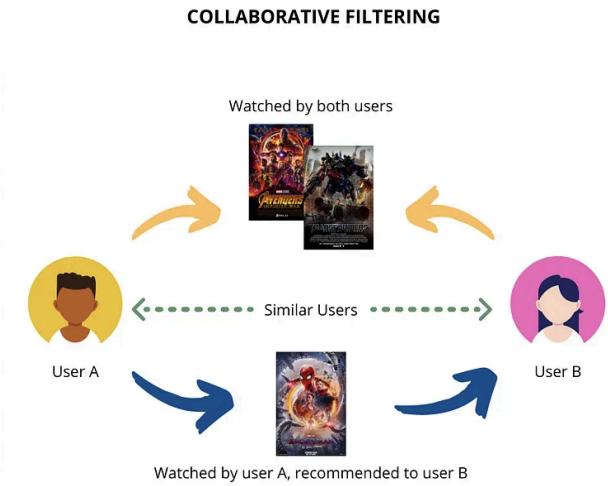
Collaborative Filtering (CF) is a widely used technique in recommendation systems that leverages the collective wisdom of users' interactions to make personalized recommendations. The algorithm works by identifying similarities between users or items based on their historical interactions, such as ratings or purchases. There are two main types of CF: user-based and item-based. In user-based CF, similarities between users are calculated based on their shared interactions with items. Similarly, in item-based CF, similarities between items are determined based on the users who interacted with them. Once similarities are computed, predictions for user-item interactions can be made by aggregating ratings or preferences from similar users or items. This allows the system to recommend items to a user based on the preferences of users with similar tastes or the characteristics of items similar to those the user has liked. Overall, collaborative filtering enables recommendation systems to provide personalized suggestions by harnessing the collective behavior of users. Calculate the similarity between users based on their past interactions. A common similarity measure is cosine similarity, which computes the cosine of the angle between two user vectors in the rating space. For users u and v, the cosine similarity is given by:

$$sim(u,v) = \frac{\sum_{i \in I_u \cap I_v} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I_u} r_{ui}^2} \cdot \sqrt{\sum_{i \in I_v} r_{vi}^2}}$$

Where I_u and I_v are the sets of items rated by users u and v respectively, and r_{ui} represents the rating of user u for item i.

4.1.2. Design of the prediction model:

The prediction model utilized in this scenario is based on Singular Value Decomposition (SVD), a matrix factorization technique commonly employed in recommendation systems. SVD decomposes the user-item interaction matrix into three matrices: a user matrix, a feature matrix, and an item matrix. These matrices represent latent factors that capture underlying patterns in the data. During the training phase, the SVD model is fitted to the training data using gradient descent or similar optimization algorithms. The model learns the optimal values of latent factors to minimize the difference between predicted ratings and actual ratings in the training set. To optimize the model's hyperparameters, a grid search is conducted over a predefined parameter grid, considering factors such as the number of latent factors (n_factors) and the number of epochs (n_epochs). Cross-validation is employed to evaluate the model's performance using metrics like Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). The best-performing model configuration, determined through grid search, is then utilized to train the SVD model on the entire training dataset. Finally, the trained model is evaluated on the test dataset to assess its generalization performance and suitability for making accurate predictions in real-world scenarios.



4.1.3. Hyperparameters tuning:

Parameter/technique tuning is the process of optimizing the hyperparameters or configurations of a machine learning model to improve its performance. In the provided code, a grid search technique is employed to find the best combination of hyperparameters for the Singular Value Decomposition (SVD) model used in collaborative filtering. Hyperparameters like the number of latent factors and the number of epochs are systematically varied over a predefined grid of values. Cross-validation is then used to evaluate each combination's performance and identify the optimal configuration that minimizes metrics such as Root Mean Squared Error (RMSE) or Mean Absolute Error (MAE). This process ensures the model's parameters are fine-tuned to achieve the best possible performance on unseen data.

4.1.4. Result evaluation:

The evaluation of the collaborative filtering model using Singular Value Decomposition (SVD) involved several steps, including hyperparameter tuning, model training, and performance assessment. Initially, the SVD model was trained using default hyperparameters (e.g., 10 epochs), and its performance was evaluated using cross-validation with 5 folds. The mean Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE) across folds were 0.5979 and 0.4678, respectively. The fit time for training was approximately 4.74 seconds, with a test time of 1.18 seconds. Subsequently, hyperparameter tuning was conducted using GridSearchCV to optimize the number of latent factors and epochs. The best configuration, with 10 latent factors and 10 epochs, achieved an RMSE of 0.8333. Using this optimized configuration, the model was trained on the full training dataset and evaluated on the test dataset. The RMSE and MAE obtained were 0.9452 and 0.7483, respectively, with reduced computational time compared to the initial training phase. Finally, the model's performance was further evaluated using K-Fold cross-validation on the test dataset. The mean accuracy was 0.0098 with a standard deviation of 0.0006, while the mean RMSE was 0.9448 with a standard deviation of 0.0046.

Overall, the collaborative filtering model demonstrated reasonable performance in predicting user ratings and provided insights into its generalization capabilities and computational efficiency.

4.1.5. Analysis:

The function takes a trained model, a user ID, a dataframe containing user ratings, a dataframe of movie information, and the number of items to recommend as input. It generates personalized movie recommendations for the specified user based on their past interactions and the model's predictions. First, it identifies the unique movie IDs in the dataset and extracts the IDs of movies that the user has not yet rated. Then, it constructs a test set containing the user ID, unrated movie IDs, and a placeholder rating (e.g., 4). The model is used to predict ratings for these unrated movies. The function then prints the top recommended movies for the user, sorted by predicted rating percentile. It retrieves the movie titles from the movies dataframe based on the predicted movie IDs and prints them along with their predicted ratings. This recommendation function provides a practical way to offer personalized movie suggestions to users based on their historical preferences and the collaborative filtering model's predictions. It enhances user experience by facilitating the discovery of relevant and appealing movie choices.

4.1.6. Coherence and Clarity:

The Singular Value Decomposition (SVD) algorithm is a powerful technique used in recommendation systems to uncover latent factors within user-item interaction matrices. By decomposing the matrix into three submatrices, SVD captures underlying patterns in the data, facilitating accurate predictions of user preferences. This algorithm offers robustness in handling sparse and high-dimensional data, making it suitable for large-scale recommendation tasks. With its simplicity and effectiveness, SVD enhances recommendation system performance by providing personalized suggestions that align closely with user preferences, thereby improving user satisfaction and engagement.

4.2. Content-Based Filtering:

4.2.1. Description:

Content-based filtering is a recommendation system approach that suggests items to users based on the similarity between items' features and users' preferences. In item-based content-based filtering, items are represented by a set of features or attributes. The algorithm works by computing the similarity between items based on these features. First, each item is represented as a feature vector in a high-dimensional space, where each dimension corresponds to a feature or attribute. Then, similarity between items is calculated using similarity measures such as cosine similarity or Euclidean distance. Items with similar feature vectors are considered to be more alike and are recommended to users who have shown interest in similar items in the past. Recommendations are made by identifying items that are most similar to items that the user has

interacted with or rated positively. By focusing on item attributes and users' preferences, content-based filtering offers personalized recommendations that align closely with users' tastes and preferences.

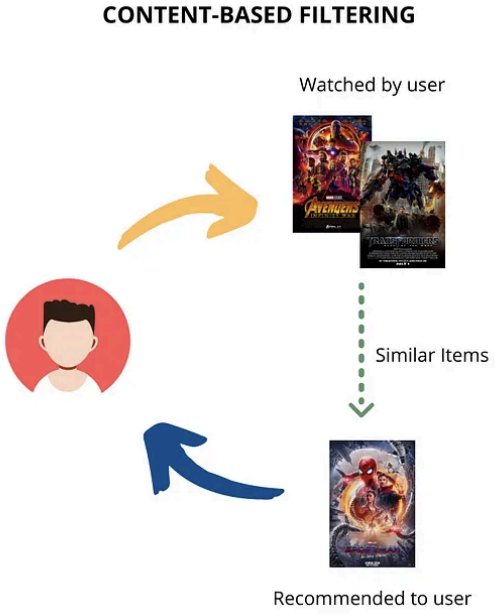
The similarity between items i and j is computed using a similarity measure such as cosine similarity:

$$sim(u, v) = \frac{V_i . V_j}{||V_i|| \cdot ||V_j||}$$

Where $\cdot$ denotes the dot product and $||V_i||$ denotes the Euclidean norm.

4.2.2. Design of the prediction model:

The prediction model employed in this scenario combines elements of data preprocessing, dimensionality reduction, and similarity-based recommendation. First, the dataset is filtered to include only items with a sufficient number of user votes and users who have provided a substantial number of ratings, ensuring data quality. Next, the movie titles are encoded using LabelEncoder to convert them into numerical representations. The feature matrix is constructed using a sparse matrix format (csr_matrix), which efficiently represents the dataset's high-dimensional feature space. A nearest neighbors model (KNN) is then trained on this feature matrix, using cosine similarity as the distance metric. This model enables efficient similarity search and retrieval of similar items based on their feature vectors. Overall, this prediction model integrates data preprocessing, encoding, sparse matrix representation, and similarity-based recommendation to generate personalized item recommendations for users based on their historical interactions and item similarities.



4.2.3. Hyperparameters tuning:

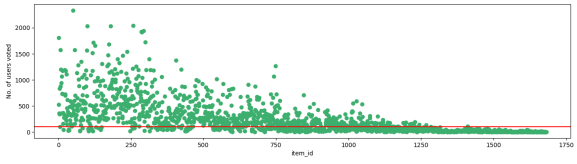
Parameter/technique tuning involves optimizing the hyperparameters and configuration of the K-Nearest Neighbors (KNN) model to enhance its performance. In the provided code, tuning may involve selecting the appropriate number of neighbors (n_neighbors), the distance metric (e.g., cosine similarity), and the algorithm (e.g., brute force or KD-tree). Techniques such as grid search or randomized search can be employed to systematically explore different parameter combinations and select the optimal configuration based on performance metrics like accuracy or precision. Tuning ensures that the KNN model effectively captures the underlying structure of the data and provides accurate and reliable recommendations to users.

4.2.4. Result evaluation:

The result evaluation of the movie recommendation function involves presenting the top recommended movies based on their similarity to the input movie. The function takes as input a movie name and the number of movies to recommend. Firstly, it transforms the movie name into its numerical representation using LabelEncoder. Then, it retrieves the index of the input movie in the dataset and calculates the distances to its nearest neighbors using the KNN model. The function prints the recommended movies along with their distances, sorted by decreasing similarity. Each recommended movie is displayed alongside its distance from the input movie. This evaluation provides users with a list of movies that are most similar to the input movie, allowing them to discover related films that align with their preferences. The distances serve as a measure of similarity, with smaller distances indicating greater similarity between movies.

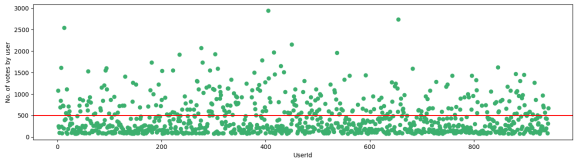
4.2.5. Analysis:

This code visualizes the distribution of user votes for each movie and the number of movies voted by each user. It creates a scatter plot where the x-axis represents the item IDs (movies) and the y-axis represents the count of users who voted for each movie. The red horizontal line indicates a threshold of 100 votes per movie. This visualization helps identify movies with significant user engagement and potential outliers. Understanding this distribution can guide decisions related to data filtering or model training in recommendation systems.



The filters users who have voted on more than 100 items, creating a final dataset containing only those users. It then visualizes the number of votes by each user using a scatter plot, with user IDs on the x-axis and the corresponding number of votes on the y-axis. The horizontal red line indicates a threshold of 500 votes. This visualization helps identify users who have provided a significant number of

ratings, potentially indicating active users or those with a strong engagement with the system.



4.2.6. Coherence and Clarity:

The K-Nearest Neighbors (KNN) algorithm efficiently identifies similar items by measuring distances in a high-dimensional feature space. Through cosine similarity, it calculates the proximity between items, aiding in personalized recommendation generation. By tuning parameters like the number of neighbors and distance metric, KNN adapts to varying datasets and user preferences. Leveraging sparse matrix representation and nearest neighbor search, it efficiently retrieves relevant recommendations. This algorithm's intuitive approach and adaptability contribute to its effectiveness in providing tailored suggestions and enhancing user experience in recommendation systems.

5. Movie recommendations:

5.1. Collaborative Filtering:

These recommendations for user 2 are generated based on collaborative filtering, leveraging the interactions of similar users to make personalized suggestions. The algorithm identifies movies that user 2 has not yet rated but is highly rated by users with similar preferences. The top 5 recommendations reflect popular and highly acclaimed movies across various genres and eras, such as "Schindler's List," "Casablanca," and "Shawshank Redemption." These suggestions are likely to resonate with user 2's tastes and preferences, enhancing their movie-watching experience by providing a diverse selection of high-quality films that align with their interests.

```
[28]  1  user_id = 2#int(input('enter the user id(1-5): '))
      2  n_items = 5#int(input('enter the number of movie suggestions: '))
      3  generate_recommendation(svd, user_id, df_test, movies, n_items)

Top 5 item recommendations for user 2:

1. "Schindler's List (1993)", Rating percentile: 4.687843813403272
2. "Casablanca (1942)", Rating percentile: 4.660213540831488
3. "Wrong Trousers, The (1993)", Rating percentile: 4.659215036646986
4. "Shawshank Redemption, The (1994)", Rating percentile: 4.623367341881574
5. "12 Angry Men (1957)", Rating percentile: 4.604021779096695
```

5.2. Content-Based Filtering:

In this content-based recommendation, movies similar to "Toy Story (1995)" are suggested based on their feature similarity. The output presents the top 5 recommendations alongside their distances from the input movie, reflecting their level of similarity. The suggested movies include "Twelve Monkeys (1995)" and "Usual Suspects, The (1995)," indicating that they share similar attributes or themes with "Toy Story." These recommendations are tailored to user preferences based on the content of the input movie, offering a selection of films that are likely to appeal to individuals interested in animated, adventure, or mystery genres, enhancing their movie-watching experience.

```
[38]  1  get_movie_recommendation('Toy Story (1995)', 5)

      movie_title: "Toy Story (1995)", Distance:"0.0"
      movie_title: "Twelve Monkeys (1995)", Distance:"6.148219760648832e-05"
      movie_title: "Twelve Monkeys (1995)", Distance:"6.148219760648832e-05"
      movie_title: "Usual Suspects, The (1995)", Distance:"0.00027185227882431384"
      movie_title: "Usual Suspects, The (1995)", Distance:"0.00027185227882431384"
```