Food.com Recipes and Interactions

Dataset

This dataset consists of 180K+ recipes and 700K+ recipe reviews covering 18 years of user interactions and uploads on Food.com (formerly GeniusKitchen).

The dataset consists of 4 datasets:

- PP_recipes: This dataset contains information about recipes, including recipe ID, name tokens, ingredient tokens, steps tokens, techniques used, calorie level, and ingredient IDs. There are 178,265 entries in this dataset.

- PP_users: This dataset contains information about users, including user ID, techniques used, items (presumably referring to recipes), number of items, ratings given by users, and number of ratings provided. There are 25,076 entries in this dataset.

- RAW_interactions: This dataset contains raw interaction data between users and recipes, including user ID, recipe ID, date of interaction, rating given by the user, and user's review. There are 1,132,367 entries in this dataset.

- RAW_recipes: This dataset contains information about recipes, including recipe name, ID, duration in minutes, contributor ID, submission date, tags, nutrition information, number of steps, steps description, recipe description, ingredients, and number of ingredients. There are 231,637 entries in this dataset.

These datasets collectively provide a comprehensive view of recipes, user interactions, and user profiles, which can be utilized for various analyses such as recommendation systems, trend analysis, or user behavior modeling in the context of cooking and recipe sharing platforms.

Preprocessing

The **Data** class is a Python implementation aimed at managing and analyzing data related to recipes and user interactions. Below is a detailed description of the functionalities provided by this class:

Data Loading:
The class facilitates the loading of user data and recipe data from CSV files. Upon instantiation, the paths to the CSV files containing user data and recipe data are specified.

Data Filtering:
One of the core functionalities of this class is data filtering. It filters user data based on specific criteria, such as identifying users who have rated more than 5 items and recipes that are rated

by more than 10 users. This step helps in focusing the analysis on significant data points and improving the quality of the insights derived from the data.

Data Splitting:
After filtering the user data, the class further splits the filtered data into training and test datasets. This division is crucial for building predictive models or conducting statistical analysis, as it allows for the evaluation of model performance on unseen data.

Class Methods:
__init__(self): The constructor method initializes the paths to the CSV files and sets default variables.
initialise_user_data(self): This method fetches and loads user data from the specified path.
get_user_data(self): It returns the loaded user data.
filter_user_data(self): This method applies filtering criteria to the user data and splits it into training and test datasets.

The `**collaborative_filtering**` class provides methods for performing collaborative filtering on a user-item interaction dataset. Collaborative filtering is a technique commonly used in recommender systems to make automatic predictions about the interests of a user by collecting preferences from many users.

Features:

- User-Item Matrix Creation: Creates a user-item matrix from the user-item interaction data using the pivot function.
- Normalization: Normalizes the user-item matrix using Mean-Centering or Z-Score normalization techniques.
- Similarity Calculation: Calculates similarity scores between items or users using the cosine similarity metric.
- Recommendation Generation: Generates recommendations for users based on either item-item or user-user collaborative filtering approaches.
- Rating Prediction: Predicts ratings for items using both item-item and user-user collaborative filtering techniques.

Class Methods:

- create_user_item(self, final_user_recipe_data): Creates a user-item matrix from the provided user-recipe interaction data.
- normalize_user_item_mean_centering(self, row): Normalizes the user-item matrix using Mean-Centering normalization.
- normalize_user_item_z_score(self, row): Normalizes the user-item matrix using Z-Score normalization.
- get_item_item_sim(self, norm_user_item): Calculates the item-item similarity matrix using cosine similarity.

- get_user_user_sim(self, norm_user_item): Calculates the user-user similarity matrix using cosine similarity.
- get_similar_items(self, user_id, item_item_similarity, final_user_recipe_data): Retrieves similar items for a given user based on item-item similarity.
- get_similar_users(self, user_user_similarity): Retrieves similar users based on user-user similarity.
- predict_rating_item_item(self, data, user, item, item_item_similarity): Predicts rating for an item using item-item collaborative filtering.
- predict_rating_user_based(self, data, user, item, user_user_similarity): Predicts rating for an item using user-user collaborative filtering.

Normalization Methods

- **Mean Centering:**

  Mean centering involves subtracting the mean (average) of a dataset from each data point within that dataset. This process shifts the distribution so that the mean of the dataset becomes zero. It is particularly useful when dealing with variables with different scales or when analyzing changes in data over time.

  The formula for mean centering a variable x is

  $$x_{centered} = x - \bar{x}$$

- **Z-Score (Standard Score):**

  The z-score, also known as the standard score, measures the number of standard deviations a particular data point is from the mean of the dataset. It standardizes the distribution of data so that it has a mean of zero and a standard deviation of one.

  The formula for calculating the z-score of a variable x is

  $$z = \frac{x - \mu}{\sigma}$$

Model

- User-User
- Item-Item
- Matrix Factorization
- Neural Collaborative Filtering (NCF)

- Alternating Least Squares (ALS)

1. User-Based Collaborative Filtering:

User-based collaborative filtering relies on the idea that users who have similar tastes in the past will have similar tastes in the future. It makes recommendations by finding users with similar preferences to the target user and recommending items that they have liked or interacted with. The steps involved in user-based collaborative filtering are as follows:

Similarity Calculation: Calculate the similarity between users based on their past interactions with items. Cosine similarity is a commonly used metric for this purpose.
Neighborhood Selection: Identify a set of similar users to the target user based on their similarity scores.
Recommendation Generation: Generate recommendations for the target user by aggregating the preferences of the selected similar users.

2. Item-Based Collaborative Filtering:

Item-based collaborative filtering is similar to user-based collaborative filtering, but instead of finding similar users, it identifies similar items based on user interactions. The process involves the following steps:

Item Similarity Calculation: Compute the similarity between items based on the users who have interacted with both items. Cosine similarity is typically used for this purpose.
Neighborhood Selection: Select a set of similar items to the target item based on their similarity scores.

Recommendation Generation: Generate recommendations for the target user by considering items that are similar to those the user has already liked or interacted with.

3. Matrix Factorization (SVD):

Matrix factorization is a technique that decomposes the user-item interaction matrix into two lower-dimensional matrices: one representing users' latent preferences and the other representing items' latent attributes. Singular Value Decomposition (SVD) is a popular matrix factorization method used in collaborative filtering. The steps involved in SVD-based collaborative filtering are:

Matrix Decomposition: Decompose the user-item interaction matrix into three matrices: U (user matrix), Σ (diagonal matrix of singular values), and V^T (item matrix).
Dimensionality Reduction: Retain only the top-k singular values and corresponding columns of U and rows of V^T to reduce dimensionality.

Prediction Generation: Generate predictions for missing or unrated entries in the original matrix by reconstructing it from the reduced matrices.

4.Neural Collaborative Filtering

Neural Collaborative Filtering, is a recommendation system technique that combines traditional collaborative filtering with neural networks to generate personalized recommendations. It leverages deep learning models to capture complex user-item interactions and make accurate predictions.

In the provided ncf_model class, we have implemented an NCF model using TensorFlow/Keras. This model architecture includes embedding layers for users and items, followed by fully connected layers to learn non-linear patterns in user-item interactions. We trained the model using training data, evaluated its performance on a test set, and made predictions for user-item ratings. Overall, the class enables the creation, training, evaluation, and prediction using an NCF model for recommendation tasks.

5.Alternating Least Square

Alternating Least Squares, is a matrix factorization technique commonly used in collaborative filtering-based recommendation systems. It decomposes the user-item interaction matrix into two lower-dimensional matrices representing user and item factors. These factors capture latent features of users and items, allowing for accurate prediction of missing ratings.

In the provided ALS class, we have implemented the ALS algorithm in Python.

Bias Calculation: We computed item biases and user biases based on the average rating and individual ratings. These biases account for global trends in ratings and specific tendencies of users/items to rate higher or lower than average.

Prediction: Using the learned user and item factors along with biases, we made predictions for user-item ratings. These predictions help in recommending items to users based on their preferences and past interactions.

Evaluation

Mean Absolute Error

In the context of recommendation systems, MAE is a commonly used metric to evaluate the accuracy of rating predictions. It measures the average absolute difference between the predicted ratings and the actual ratings provided by users.

The formula for calculating MAE is as follows:

$$MAE = \frac{1}{n} \Sigma_{i=1}^{n} |predicted\ rating - actual\ rating|$$