```
import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn import model_selection
from sklearn import metrics
from sklearn import neighbors
from sklearn.model_selection import StratifiedKFold
```

Introduction

The realm of movie entertainment has been continuously expanding, offering viewers an overwhelming selection of film across various genres. In this vast ocean of choices, finding the one that aligns with a viewer's preference can be daunting. A Movie Recommendation System aims to simplify this challenge by suggesting films tailored to individual tastes, leveraging data from past viewing patterns and ratings to predict which content a viewer might find enjoyable, thereby enhancing their viewing experience.

Methodology

Our approach to the Movie Recommendation System is underpinned by Collaborative Filtering using the K-Nearest Neighbors algorithm. Initially, both movies. Finally, the datasets for user and ratings.csv are imported and subsequently merged on the 'movieId' column, giving a combined dataset which maps each user's rating to their respective movie. The dataset is then cleaned by dropping the 'timestamp' column, as it is not relevant to our model's analysis. Exploratory Data Analysis (EDA) is conducted to understand the distribution of ratings and user-item based and user-based Collaborative Filtering. K-get exact, using 10-fold cross-validation, is then executed to evaluate the model's performance, with the accuracy and confusion matrix being the metrics of interest. Additionally, SMD and Jaccard scores are also used as the complete dataset. K-Fold cross-validation, plot(1) is developed to recommend the top 10 unrelated movies for any given user based on their estimated ratings. Finally, the movies recommended for a random user are extracted and displayed.

Data Exploration & Preprocessing

The Dataset are used in the MovieLens dataset. As we can see from below, the movies.csv is consisted of three columns: movieId, title, and genres. The movieId is a unique identifier for each film, the title includes the movie's name, and genres lists the corresponding categories. Starting our data collaborative filtering for our recommendation algorithm, we will only need the movieId, userId and ratings. This is because collaborative filtering is either user-user based or item-item based, focusing on how users rate items and not the specific nature or difference in content similarity between two items, and, such as genre, or actors and directors.

```
movies_df = pd.read_csv('movies.csv')
movies_df.shape, movies_df.sample(3)
```

```
((9742, 3),
      movieId                                      title genres
 8  ...
```

```
ratings_df = pd.read_csv('ratings.csv')
ratings_df.shape, ratings_df.sample(3)
```

```
((100836, 4),
      userId  movieId  rating  timestamp
 8  ...
```

We merge the two data frames on movieId column, and by keeping only the userId, movieId and rating columns, we then separated the dataset we need for using python surprise library for building our recommender.

```
pd.set_option('display.max_columns', 50)
movie_rating = pd.merge(movies_df, ratings_df, on='movieId')
movie_rating.shape, movie_rating.sample(3)
```

```
((100836, 6),
      movieId                                      title genres  userId  rating  timestamp
 8  ...
```

```
movie = movie[['userId', 'title', 'rating']]
data = Dataset.load_from_df(movie[['userId', 'title', 'rating']], reader)
```

Results

The chosen algorithm for our recommendation system is the Centered KNN. It operates by computing either the cosine similarity or the Pearson correlation between each pair of entities (users or items) and then predicts ratings by centering the data (subtracting mean ratings) as a user's expected rouse rating. To optimize the model's performance, we employed hyperparameter tuning through a 10-fold cross-validation. During this process, the Root Mean Squared Error (RMSE) serves as the primary metric to assess the efficacy of various hyperparameter combinations. Following this grid search, an optimal model yielded an RMSE of 0.9864, which we believe represents a satisfactory level of accuracy.

```
sim_options = {
    "name": "cosine",
    "user_based": True,
    "min_support": [3, 4, 6, 7, 8],
    "shrinkage": [0, 100, 500]
}
param_grid = {"sim_options": sim_options}

gs = CrossValidation(NormalPredictor, param_grid, measures=["rmse", "mae"], cv=10)

gs.fit(data)

print("Best RMSE score")
print(gs.best_score["rmse"])
print(gs.best_params["rmse"])
```

```
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mse similarity matrix...
Done computing similarity matrix.
Computing the mae similarity matrix...
Done computing similarity matrix.
Computing the mae similarity matrix...
Done computing similarity matrix.
Computing the mae
```