

Experiment No 8

Aim: To implement a recommendation system on your dataset using the following machine learning techniques.

Theory:

Types of Recommendation Systems:

Recommendation systems are generally categorized into three main types: content-based filtering, collaborative filtering, and hybrid approaches.

Content-based filtering makes recommendations by analyzing the characteristics of items a user has previously interacted with or liked. For instance, if a user frequently watches romantic comedies, the system might suggest movies with similar genres, actors, or directors.

Collaborative filtering, on the other hand, leverages the preferences of multiple users to generate recommendations. In user-based collaborative filtering, users with similar preferences are identified, and items liked by one user are recommended to others with similar tastes. In item-based collaborative filtering, items that are often liked together by many users are recommended based on their co-occurrence patterns.

Hybrid recommendation systems combine both content-based and collaborative filtering techniques. These systems aim to capitalize on the advantages of each method while addressing common challenges like the cold start problem, which arises when there is insufficient data about new users or items, reducing recommendation accuracy.

Evaluation Measures for Recommendation Systems:

To evaluate the performance of a recommendation system, several metrics are commonly used:

Root Mean Square Error (RMSE) measures the difference between predicted ratings and actual user ratings. A lower RMSE indicates better predictive accuracy.

Precision@K evaluates the proportion of relevant items among the top K recommendations, focusing on the system's ability to recommend high-quality items.

Recall@K measures the proportion of all relevant items that appear in the top K results, reflecting the system's completeness in capturing user preferences.

F1-score is the harmonic mean of precision and recall, offering a balanced assessment when both metrics are critical. These evaluation metrics play a crucial role in comparing different recommendation models and in optimizing them for enhanced performance.

Steps :**1) Import Required Libraries**

Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from surprise import SVD, Dataset, Reader
from surprise.model_selection import train_test_split
from surprise import accuracy
```

This code snippet imports all the essential libraries needed to build a recommendation system that combines clustering and collaborative filtering techniques. Pandas and NumPy are used for efficient data manipulation and numerical computations, while Matplotlib.pyplot supports the visualization of data and results. From the scikit-learn library, KMeans is employed for clustering users or items based on similarities, and StandardScaler is used to normalize the data for better clustering performance. The Surprise library, which is specifically designed for recommendation systems, is utilized to implement the SVD (Singular Value Decomposition) algorithm. It also provides helpful tools such as Dataset, Reader, train_test_split, and accuracy to load and preprocess the dataset, split it into training and testing sets, and evaluate the model's performance. Overall, this collection of imports forms the backbone for developing, analyzing, and visualizing a recommendation system using both clustering and collaborative filtering methods.

2) Load the Dataset

Code:

```
anime = pd.read_csv('/content/drive/MyDrive/anime.csv')
ratings = pd.read_csv('/content/drive/MyDrive/rating.csv')
```

This code loads two CSV files from your Google Drive into pandas DataFrames:

- anime.csv → Contains information about anime shows (like title, genre, type, rating, etc.).
- rating.csv → Contains user ratings for those anime (user ID, anime ID, and the score given).

These two datasets will be used together to build the recommendation system — one for the content metadata and the other for user interaction data.

3) Data Cleaning

Code:

```
anime.dropna(inplace=True)
anime = anime[anime['genre'] != 'Unknown']
ratings = ratings[ratings['rating'] != -1]
```

This code snippet performs essential data cleaning to ensure that only relevant and meaningful data is used to build the recommendation system. It begins by removing rows with missing values from the anime dataset using `dropna()`, which helps prevent errors during model training. It then filters out entries with the genre labeled as 'Unknown', since such entries lack informative value for generating recommendations. Additionally, it cleans the ratings dataset by eliminating records where the rating is -1, as these typically represent unrated entries that do not contribute to understanding user preferences. Together, these cleaning steps refine the dataset and enhance the overall performance of the recommendation model.

4) Merging Anime Metadata with Ratings

Code:

```
merged_df = ratings.merge(anime, on='anime_id')
```

This step performs an inner join between the ratings and anime DataFrames using `anime_id` as the common key. The resulting DataFrame, `merged_df`, combines user rating information with relevant anime metadata such as titles and genres. Merging these datasets is a crucial step in building a recommendation system, as it links user preferences to specific anime attributes. This integration enables the model to generate more personalized and accurate recommendations based on both user behavior and item characteristics.

5) Clustering Anime Based on Popularity and Rating

Code:

```
anime_cluster = anime.copy()
anime_cluster = anime_cluster[anime_cluster['members'] > 0]
anime_cluster['rating'] = pd.to_numeric(anime_cluster['rating'], errors='coerce')
```

```
anime_cluster.dropna(subset=['rating'], inplace=True)
```

```
features = anime_cluster[['rating', 'members']]  
scaler = StandardScaler()  
scaled_features = scaler.fit_transform(features)
```

```
kmeans = KMeans(n_clusters=5, random_state=42, n_init='auto')  
anime_cluster['cluster'] = kmeans.fit_predict(scaled_features)
```

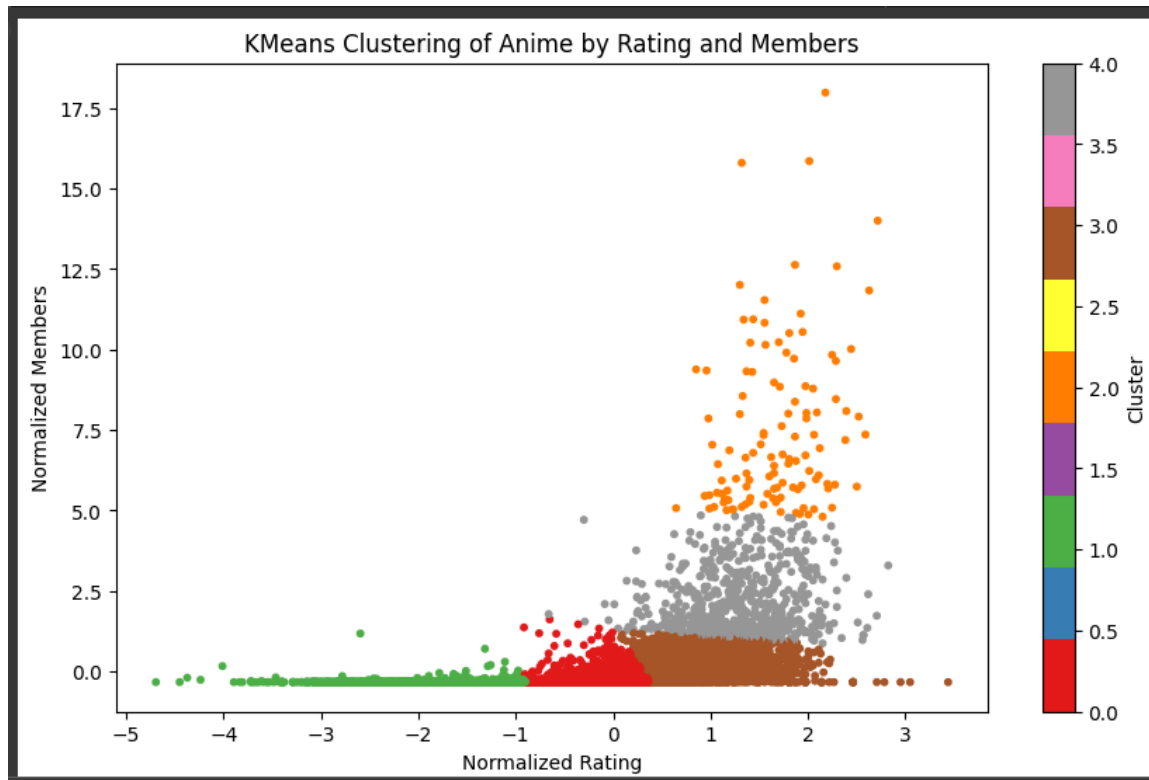
In this step, a copy of the original anime dataset is created to prepare it for clustering. The data is first filtered to include only anime entries with more than zero members, ensuring that only titles with some user engagement are considered. The rating column is then converted to a numeric format, with any non-convertible values handled gracefully. Rows with missing ratings are subsequently removed to maintain clean and reliable input for clustering. The clustering process uses two key features: the average user rating and the number of members. These features are standardized using StandardScaler to ensure they are on the same scale—an essential step for accurate clustering. Finally, the KMeans algorithm is applied to group the anime into five distinct clusters. Each anime is assigned a cluster label based on its similarity in terms of rating and popularity, enabling deeper analysis or visualization of anime with similar characteristics.

6) Visualizing Clusters of Anime

Code:

```
plt.figure(figsize=(10,6))  
plt.scatter(scaled_features[:, 0], scaled_features[:, 1], c=anime_cluster['cluster'], cmap='Set1',  
s=10)  
plt.title('KMeans Clustering of Anime by Rating and Members')  
plt.xlabel('Normalized Rating')  
plt.ylabel('Normalized Members')  
plt.colorbar(label='Cluster')  
plt.show()
```

Output:



This step creates a scatter plot to visually interpret the clusters formed by the KMeans algorithm. In the plot, the x-axis represents the normalized average ratings, while the y-axis shows the normalized number of members for each anime. Each point corresponds to an individual anime title, and its color indicates the cluster to which it has been assigned. The colormap `cmap='Set1'` is used to assign distinct and easily distinguishable colors to each cluster, enhancing visual clarity. A colorbar is included to map each color to its corresponding cluster label. This visualization helps reveal how anime titles are grouped based on similarities in user ratings and popularity, offering valuable insights into audience preferences and content trends.

7) Building and Training the SVD-Based Recommendation Model

Code:

```
reader = Reader(rating_scale=(1, 10))
data = Dataset.load_from_df(ratings[['user_id', 'anime_id', 'rating']], reader)
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)

model = SVD()
model.fit(trainset)
```

```
predictions = model.test(testset)
```

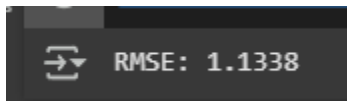
In this step, a recommendation system is constructed using the Singular Value Decomposition (SVD) algorithm from the Surprise library. A Reader object is first defined to specify the rating scale, which ranges from 1 to 10. The `Dataset.load_from_df()` function is then used to convert the cleaned ratings data into a format compatible with the Surprise framework. The dataset is split into training and test sets using an 80-20 ratio to facilitate model evaluation. The SVD model is trained on the training data to learn latent factors that capture underlying patterns in user preferences and anime characteristics. Once trained, the model is used to predict ratings on the test set. This process lays the groundwork for generating personalized anime recommendations through collaborative filtering.

8) Model Evaluation using RMSE

Code:

```
rmse = accuracy.rmse(predictions)
```

Output:



In this step, the performance of the recommendation model is evaluated using the Root Mean Squared Error (RMSE). The obtained RMSE value of 1.1338 reflects the average deviation between the actual user ratings and the predictions made by the SVD model. Given that the rating scale ranges from 1 to 10, an RMSE around 1.1 is considered reasonably acceptable. This indicates that the model is fairly accurate in its predictions and can be relied upon to provide meaningful and personalized anime recommendations to users.

9) Function to Generate Top-N Anime Recommendations for a User

Code:

```
def get_top_n_recommendations(user_id, anime_df, ratings_df, model, n=10):
    all_anime_ids = anime_df['anime_id'].unique()
    rated_anime_ids = ratings_df[ratings_df['user_id'] == user_id]['anime_id'].unique()
    unseen_anime_ids = [aid for aid in all_anime_ids if aid not in rated_anime_ids]
    predictions = [model.predict(user_id, aid) for aid in unseen_anime_ids]
    top_predictions = sorted(predictions, key=lambda x: x.est, reverse=True)[:n]
    top_anime_ids = [pred.iid for pred in top_predictions]
    recommendations = anime_df[anime_df['anime_id'].isin(top_anime_ids)][['name', 'genre',
'type', 'rating']]
    return recommendations
```

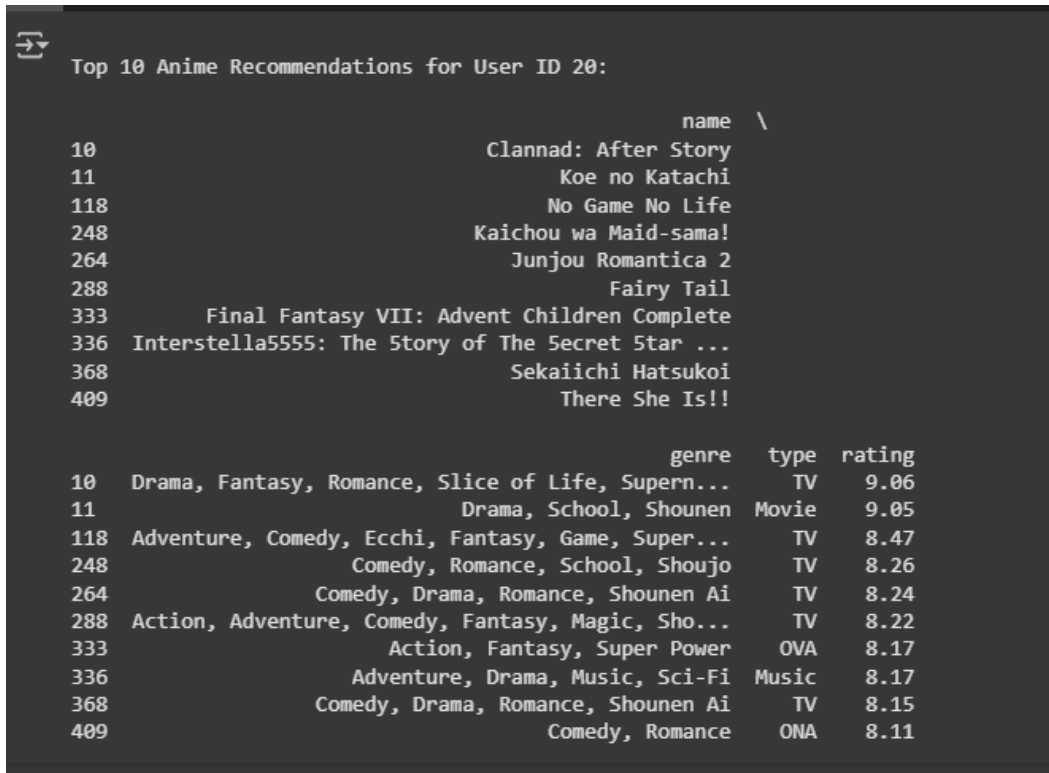
This function is designed to provide personalized anime recommendations for a specific user based on the trained recommendation model. It first retrieves all available anime IDs and filters out the ones the user has already rated. It then uses the model to predict ratings for the unseen anime and selects the top N (default 10) highest-rated predictions. The function finally returns detailed information (name, genre, type, and rating) about these top recommendations. This allows the system to generate relevant suggestions tailored to the user's interests.

10) Displaying Top 10 Anime Recommendations for a Specific User

Code:

```
user_id_sample=20
print(f"\nTop 10 Anime Recommendations for User ID {user_id_sample}:\n")
print(get_top_n_recommendations(user_id_sample, anime, ratings, model))
```

Output:



Top 10 Anime Recommendations for User ID 20:

	name \	genre	type	rating
10	Clannad: After Story	Drama, Fantasy, Romance, Slice of Life, Supern...	TV	9.06
11	Koe no Katachi	Drama, School, Shounen	Movie	9.05
118	No Game No Life	Adventure, Comedy, Ecchi, Fantasy, Game, Super...	TV	8.47
248	Kaichou wa Maid-sama!	Comedy, Romance, School, Shoujo	TV	8.26
264	Junjou Romantica 2	Comedy, Drama, Romance, Shounen Ai	TV	8.24
288	Fairy Tail	Action, Adventure, Comedy, Fantasy, Magic, Sho...	TV	8.22
333	Final Fantasy VII: Advent Children Complete	Action, Fantasy, Super Power	OVA	8.17
336	Interstella5555: The Story of The Secret 5tar ...	Adventure, Drama, Music, Sci-Fi	Music	8.17
368	Sekaiichi Hatsukoi	Comedy, Drama, Romance, Shounen Ai	TV	8.15
409	There She Is!!	Comedy, Romance	ONA	8.11

In this step, the system fetches and prints the top 10 anime recommendations for a user with user_id 20. By calling the get_top_n_recommendations() function and passing in the required data and model, it displays a curated list of anime titles that the user has not yet rated but is likely to enjoy based on the trained SVD model. This output demonstrates how the recommendation

system can be personalized for individual users and showcases the system's practical usage in making intelligent suggestions.

Conclusion:

In this experiment, we developed a hybrid recommendation system by integrating clustering and collaborative filtering techniques. K-Means was employed to group similar anime based on features such as average rating and popularity, enabling content-based insights. Simultaneously, the SVD algorithm was used to predict user ratings for unseen anime, capturing collaborative filtering patterns. The model achieved satisfactory accuracy and was able to generate personalized Top-N recommendations. This highlights the effectiveness of combining content-based and collaborative approaches to build intelligent and user-centric recommendation systems.