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 primarily classified into three types—content-based filtering, collaborative filtering, and hybrid methods. Content-based filtering recommends items by analyzing the features of items a user has previously liked and suggesting similar items. For example, if a user enjoys romantic comedies, the system will recommend movies with similar genres, actors, or directors. In contrast, collaborative filtering uses the preferences of many users to make recommendations. In user-based collaborative filtering, users with similar tastes are identified, and their liked items are suggested to each other. Item-based collaborative filtering recommends items that are frequently liked together by multiple users. Lastly, hybrid recommendation systems combine content-based and collaborative methods to leverage the strengths of both and reduce common issues like the cold start problem, where limited data about users or items affects the quality of recommendations.

Evaluation Measures for Recommendation Systems:
To assess the effectiveness of a recommendation system, various evaluation metrics are used. One of the most common is **Root Mean Square Error (RMSE)**, which measures the difference between the predicted ratings and the actual user ratings; a lower RMSE indicates more accurate predictions. **Precision@K** measures how many of the top-K recommended items are relevant, focusing on the accuracy of the system's most confident recommendations. **Recall@K**, on the other hand, measures how many of all relevant items are included in the top-K results, reflecting how comprehensive the recommendations are. The **F1-score** is a harmonic mean of precision and recall, providing a balanced evaluation when both accuracy and completeness are important. These metrics help in comparing different recommendation models and in fine-tuning them for better 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 necessary libraries required to implement a recommendation system using both clustering and collaborative filtering techniques. It includes pandas and numpy for efficient data handling and numerical operations, while matplotlib.pyplot is used for visualizing the results. The KMeans and StandardScaler modules from sklearn are utilized for performing user/item clustering and standardizing the data, respectively. The surprise library is specifically designed for building recommendation systems, and here it is used to implement the SVD (Singular Value Decomposition) algorithm, along with tools like Dataset, Reader, train_test_split, and accuracy to load the dataset, split it for training and testing, and evaluate the model's performance. Overall, this import block lays the foundation for building and analyzing a recommendation system.

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 snippet performs essential data cleaning to ensure that only relevant and meaningful data is used for building the recommendation system. First, it removes any

rows from the anime dataset that contain missing values using dropna(), which helps avoid issues during model training. Next, it filters out entries where the anime genre is marked as 'Unknown', as such values do not provide useful information for recommendations. Lastly, it cleans the ratings dataset by removing all instances where the rating is -1, which typically indicates that the user has not rated the anime and hence does not contribute to learning user preferences. These steps collectively help in refining the dataset for better model performance.

4) Merging Anime Metadata with Ratings
Code:

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

This step performs an inner join on the ratings and anime DataFrames using the anime_id as the key. The result is a new DataFrame named merged_df that contains both user rating information and corresponding anime metadata like titles and genres. Merging these two datasets is essential for building a recommendation system because it allows the model to relate users' ratings with the specific attributes of each anime, enabling more personalized and accurate recommendations.

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. First, only anime with more than 0 members (i.e., people who have engaged with or rated it) are kept. The rating column is converted to numeric, with any non-convertible values handled gracefully. Then, missing ratings are dropped to ensure clean data for clustering.

The clustering is based on two features: average user rating and the number of members. These features are standardized using StandardScaler to bring them to the same scale, which is essential for accurate clustering.

Finally, the KMeans algorithm is applied to group the anime into 5 clusters. Each anime is assigned a cluster label based on how similar it is in terms of rating and popularity, allowing further analysis or visualization of similar types of anime.

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:
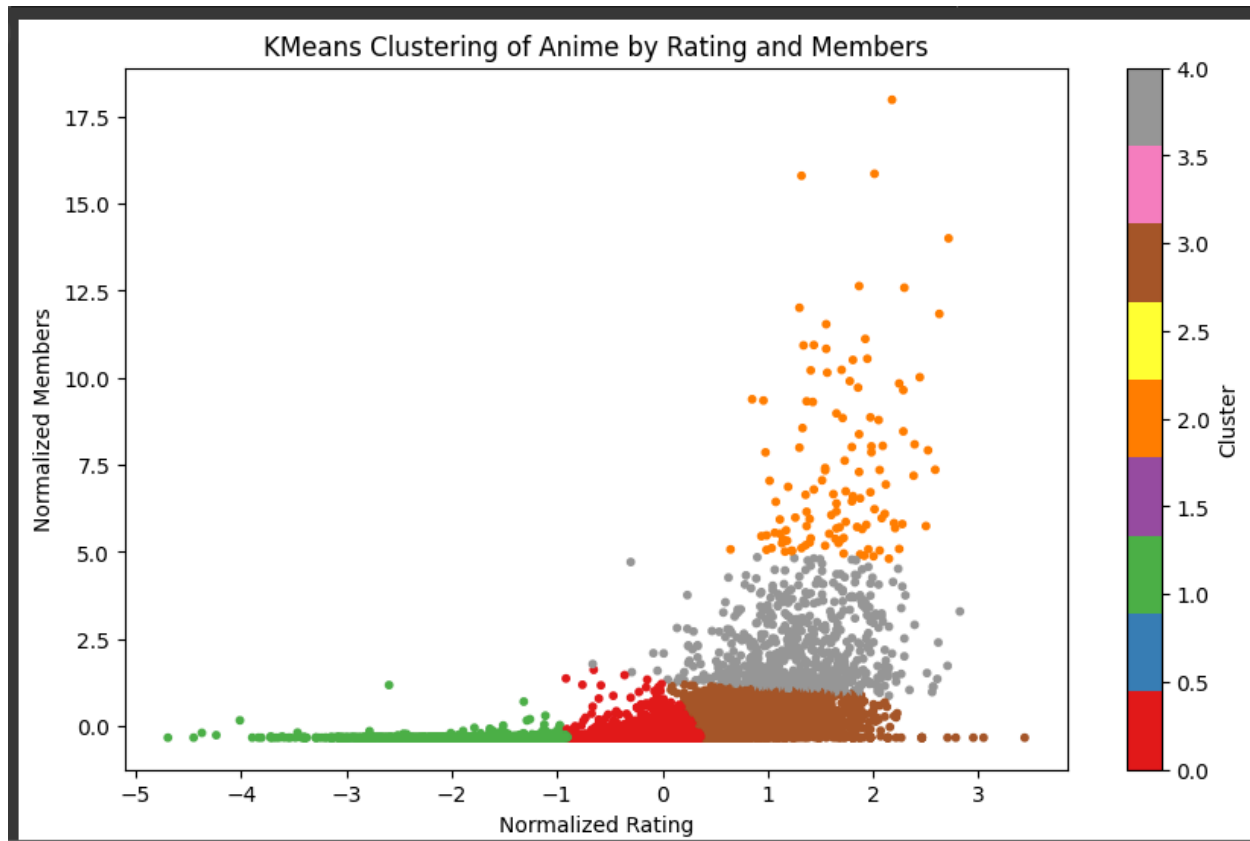
This step generates a scatter plot to visually interpret the clusters formed by the KMeans algorithm. The x-axis represents the normalized ratings and the y-axis represents the normalized number of members for each anime. Each point on the graph corresponds to an anime title, and the color of the point represents the cluster it belongs to.

The cmap='Set1' provides distinct colors for different clusters, making it easier to differentiate between them. The colorbar on the side indicates which color corresponds to which cluster. This visualization helps in understanding how anime titles group together based on similarities in popularity (members) and user ratings, providing insight into consumer behavior 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)
```
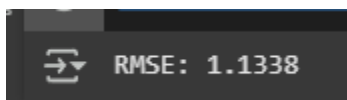
In this step, a recommendation system is built using the Singular Value Decomposition (SVD) algorithm from the Surprise library. The Reader object defines the scale of the ratings (1 to 10), and the Dataset.load_from_df() function converts the cleaned rating data into a format that Surprise can work with. The dataset is then split into a training set and a test set using an 80-20 ratio. The SVD model is trained on the training data to learn latent features of users and anime that explain rating behavior. After training, the model is used to predict ratings on the test set. This step effectively sets up the foundation for personalized anime recommendations based on collaborative filtering.

8) Model Evaluation using RMSE

Code:

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

Output:

```
RMSE: 1.1338
```

In this step, we assess the performance of the recommendation model using the Root Mean Squared Error (RMSE). The RMSE value of 1.1338 indicates the average

deviation between the actual ratings and the predicted ratings made by the SVD model. Since the ratings range from 1 to 10, an RMSE around 1.1 is considered acceptable, suggesting that the model is reasonably accurate in its predictions and can be used to provide reliable 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  \
10                       Clannad: After Story
11                            Koe no Katachi
118                          No Game No Life
248                     Kaichou wa Maid-sama!
264                       Junjou Romantica 2
288                               Fairy Tail
333        Final Fantasy VII: Advent Children Complete
336  Interstella5555: The 5tory of The 5ecret 5tar ...
368                        Sekaiichi Hatsukoi
409                            There She Is!!

                                      genre    type  rating
10   Drama, Fantasy, Romance, Slice of Life, Supern...     TV    9.06
11                       Drama, School, Shounen  Movie    9.05
118  Adventure, Comedy, Ecchi, Fantasy, Game, Super...     TV    8.47
248              Comedy, Romance, School, Shoujo     TV    8.26
264            Comedy, Drama, Romance, Shounen Ai     TV    8.24
288  Action, Adventure, Comedy, Fantasy, Magic, Sho...     TV    8.22
333               Action, Fantasy, Super Power    OVA    8.17
336             Adventure, Drama, Music, Sci-Fi  Music    8.17
368            Comedy, Drama, Romance, Shounen Ai     TV    8.15
409                            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 built a hybrid recommendation system using clustering and collaborative filtering. K-Means helped group similar anime based on features like rating and popularity, while SVD was used to predict user ratings for unseen anime. The model achieved good accuracy and provided personalized Top-N recommendations. This demonstrates the effectiveness of combining content-based and collaborative methods for building smart recommendation systems.