

Experiment No: 8

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

THEORY:

1] Recommendation System:

Recommendation systems are a subclass of machine learning applications that provide personalized suggestions to users based on their preferences, behavior, and interactions. These systems are widely used in various domains such as e-commerce (Amazon), streaming platforms (Netflix, Spotify), and social media (YouTube, Instagram). The main objective is to enhance user experience and increase engagement by predicting and displaying items that a user is likely to be interested in.

2] Types of Recommendation:

- A. Content-Based Filtering:** This method recommends items based on the **similarity between item attributes and the user's profile or past preferences.** For example, if a user likes action movies, the system suggests other action movies.
- B. Collaborative Filtering:** This approach uses user-item interaction data (like ratings or purchases) to recommend items. It is of two types:
 - 1. **User-based filtering:** Suggests items liked by similar users.
 - 2. **Item-based filtering:** Suggests items similar to what the user has already liked.
- C. Hybrid Systems:** These combine content-based and collaborative filtering to overcome the limitations of individual methods and improve accuracy.

DATASET:

The **Instacart Online Grocery Basket Analysis Dataset** is a real-world dataset containing over **3 million grocery orders** from more than **200,000 users**,

featuring around 50,000 unique products categorized into various aisles and departments. It includes detailed information about user orders (orders.csv), products purchased in prior and training orders (order_products__prior.csv and order_products__train.csv), and product metadata (products.csv, aisles.csv, and departments.csv). This dataset is widely used for building recommendation systems, analyzing customer purchasing behavior, and performing market basket and clustering analysis.

STEPS: To Implement Collaborative Filtering on Chosen dataset.

Step 1: Install Required Libraries

Code:

```
!pip uninstall -y numpy
!pip install numpy==1.26.0
!pip install scikit-surprise
```

Output:

```
Found existing installation: numpy 1.23.5
Uninstalling numpy-1.23.5:
  Successfully uninstalled numpy-1.23.5
Collecting numpy==1.26.0
  Downloading numpy-1.26.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (58 kB)
    58.5/58.5 kB 3.2 MB/s eta 0:00:00
  Downloading numpy-1.26.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.2 MB)
    18.2/18.2 MB 54.2 MB/s eta 0:00:00
Installing collected packages: numpy
Successfully installed numpy-1.26.0
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.11/dist-packages (1.1.4)
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise) (1.4.2)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise) (1.26.0)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-surprise) (1.14.1)
```

This code uninstalls the existing version of the numpy library (1.23.5) and installs a newer version (1.26.0) using pip. It then installs the scikit-surprise library, which is used for building recommendation systems. The output confirms that numpy was successfully upgraded, and scikit-surprise along with its dependencies (joblib, scipy, and numpy) were already installed in the system. This setup is necessary to ensure compatibility for collaborative filtering models using scikit-surprise.

Step 2: Import Libraries

Code:

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

This **code imports libraries needed for data handling** (numpy, pandas), **visualization** (matplotlib, seaborn), **clustering** (KMeans, StandardScaler), and **building a recommendation system using the surprise library with the SVD algorithm**, along with tools for splitting the data and evaluating model accuracy.

Step 3: Load Datasets**Code:**

```
orders = pd.read_csv("orders.csv")
order_products_prior = pd.read_csv("order_products__prior.csv")
order_products_train = pd.read_csv("order_products__train.csv")
products = pd.read_csv("products.csv")
aisles = pd.read_csv("aisles.csv")
departments = pd.read_csv("departments.csv")
```

This **code loads six CSV** files using pandas, each containing different parts of the Instacart dataset: orders (order details), order_products_prior and order_products_train (products in previous and training orders), products (product info), aisles (aisle names), and departments (department names). These datasets are essential for preparing and analyzing user purchase behavior for recommendations.

Step 4: Data Preprocessing and Merging**Code:**

```
import pandas as pd
```

```
products = products[["product_id", "aisle_id", "department_id", "product_name"]]
aisles = aisles[["aisle_id"]]
departments = departments[["department_id"]]

products = products.merge(aisles, on="aisle_id", how="left")
products = products.merge(departments, on="department_id", how="left")

order_products_prior = order_products_prior[["order_id", "product_id",
"add_to_cart_order", "reordered"]]
order_products_train = order_products_train[["order_id", "product_id",
"add_to_cart_order", "reordered"]]

order_products_prior = order_products_prior.merge(products, on="product_id",
how="left")
order_products_train = order_products_train.merge(products, on="product_id",
how="left")

order_products_prior = order_products_prior.sample(n=100000, random_state=42)
order_products_train = order_products_train.sample(n=50000, random_state=42)
all_orders = pd.concat([order_products_prior, order_products_train], ignore_index=True)

final_dataset = all_orders.merge(orders, on="order_id", how="left")

final_dataset.drop(columns=["eval_set", "aisle_id", "department_id"],
inplace=True)

print("Final Dataset Columns:", final_dataset.columns)

print("Final Dataset Shape:", final_dataset.shape)
final_dataset.head()
```

Output:

```
⇒ Final Dataset Columns: Index(['order_id', 'product_id', 'add_to_cart_order', 'reordered',
'product_name', 'user_id', 'order_number', 'order_dow',
'order_hour_of_day', 'days_since_prior_order'],
dtype='object')
Final Dataset Shape: (150000, 10)
```

	order_id	product_id	add_to_cart_order	reordered	product_name	user_id	order_number	order_dow	order_hour_of_day	days_since_prior_order
0	3109255	34099	16	0	Crushed Red Chili Pepper	135284	9	0	19	8.0
1	301098	41950	5	0	Organic Tomato Cluster	7293	2	4	15	1.0
2	1181866	45066	8	0	Honeycrisp Apple	111385	2	1	17	8.0
3	1678630	8859	2	1	Natural Spring Water	147365	7	0	14	26.0
4	644090	24781	2	0	PODS Laundry Detergent, Ocean Mist Designed fo...	99290	7	0	19	30.0

This code prepares the final dataset by selecting only the required columns from the original data to reduce memory usage, merging product details (like product_name) with aisle and department info, and combining product order data from both prior and train datasets. It samples 100,000 rows from prior and 50,000 from train to reduce the size, merges them into one dataset, and finally merges this with the main orders dataset. The **output confirms that the final dataset has 150,000 rows and includes important columns** like user_id, product_name, order_hour_of_day, etc., which are useful for building the recommendation model.

Step 5: User Clustering Based on Behavior

Code:

```
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer

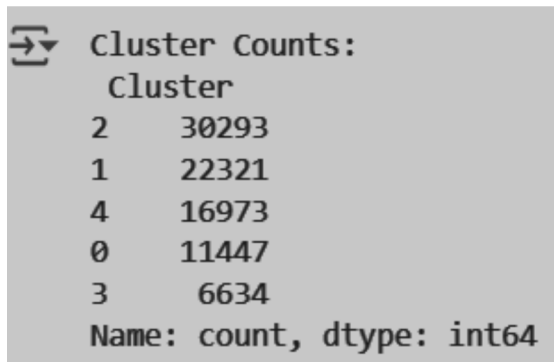
user_data = final_dataset.groupby("user_id").agg({
    "order_number": "mean",
    "days_since_prior_order": "mean",
    "add_to_cart_order": "mean"
}).reset_index()

imputer = SimpleImputer(strategy="mean")
user_data[["order_number", "days_since_prior_order", "add_to_cart_order"]] =
imputer.fit_transform(
    user_data[["order_number", "days_since_prior_order", "add_to_cart_order"]]
)

scaler = StandardScaler()
scaled_features = scaler.fit_transform(user_data[["order_number",
"days_since_prior_order", "add_to_cart_order"]])
```

```
kmeans = KMeans(n_clusters=5, random_state=42, n_init=10)
user_data["Cluster"] = kmeans.fit_predict(scaled_features)

print("Cluster Counts:\n", user_data["Cluster"].value_counts())
```

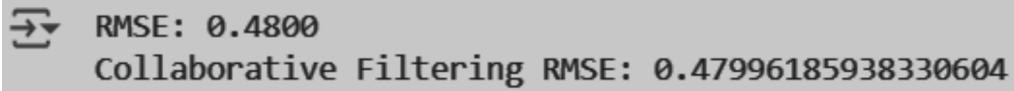
Output:A screenshot of a Jupyter Notebook cell showing the output of a value_counts() operation. The output is a Series with the index 'Cluster' and values representing the count of users in each cluster. The clusters are 2, 1, 4, 0, and 3, with counts 30293, 22321, 16973, 11447, and 6634 respectively. The dtype is int64.

```
Cluster Counts:
Cluster
2      30293
1      22321
4      16973
0      11447
3       6634
Name: count, dtype: int64
```

The **code clusters users into 5 groups** based on their shopping behavior **using KMeans**. It calculates average order stats per user, fills missing values, scales the data, and applies clustering. **The output shows how many users fall into each cluster.**

Step 6: Collaborative Filtering with SVD**Code:**

```
cf_data = final_dataset[["user_id", "product_id", "reordered"]].dropna()
reader = Reader(rating_scale=(0, 1))
data = Dataset.load_from_df(cf_data, reader)
trainset, testset = train_test_split(data, test_size=0.2, random_state=42)
svd = SVD(n_factors=50, random_state=42)
svd.fit(trainset)
predictions = svd.test(testset)
rmse = accuracy.rmse(predictions)
print("Collaborative Filtering RMSE:", rmse)
```

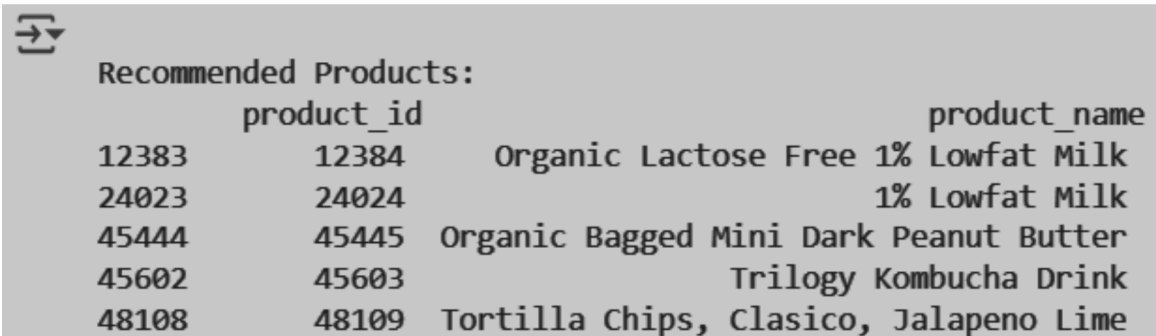
Output:


```
RMSE: 0.4800
Collaborative Filtering RMSE: 0.47996185938330604
```

The code applies collaborative filtering using SVD on user-product reorder data to predict preferences. It splits the data, trains the model, makes predictions, and evaluates performance using RMSE, which came out to around **0.48**, indicating good prediction accuracy. In recommendation systems, RMSE under **0.5** on a 0–1 scale is often seen as **acceptable to good**, depending on the dataset size and sparsity.

Step 7: Recommend Products for a User**Code:**

```
def recommend_products(user_id, num_recommendations=5):
    unique_products = final_dataset["product_id"].unique()
    predicted_ratings = [(product, svd.predict(user_id, product).est) for product in
        unique_products]
    top_products = sorted(predicted_ratings, key=lambda x: x[1],
        reverse=True)[:num_recommendations]
    recommended_product_ids = [prod[0] for prod in top_products]
    recommended_products =
        products[products["product_id"].isin(recommended_product_ids)][["product_id",
            "product_name"]]
    return recommended_products
recommendations = recommend_products(user_id=1)
print("\nRecommended Products:\n", recommendations)
```

Output:


```
Recommended Products:
      product_id      product_name
12383      12384  Organic Lactose Free 1% Lowfat Milk
24023      24024                1% Lowfat Milk
45444      45445  Organic Bagged Mini Dark Peanut Butter
45602      45603                Trilogy Kombucha Drink
48108      48109  Tortilla Chips, Clasico, Jalapeno Lime
```

A product recommendation function was built using the trained SVD model to suggest items a user is likely to reorder. For a given user_id, it predicts ratings for all products, ranks them, and returns the top 5 with the highest estimated scores. For example, for user_id = 1, the recommended products include items like **Organic Lactose Free Milk**, **Peanut Butter**, and **Trilogy Kombucha**, indicating personalized suggestions based on the user's past order patterns and similar user behavior.

CONCLUSION:

In conclusion, **this experiment built a collaborative filtering-based recommendation system using the Instacart dataset.** After merging and processing user-product data, KMeans clustering was applied to understand user behavior, and the SVD algorithm was used to predict preferences. With a good **RMSE of 0.48**, the model effectively generated personalized product suggestions, demonstrating the potential of data-driven recommendations in real-world grocery platforms.

According to my dataset **Clustering** was used to group users based on similar shopping behaviors using the KMeans algorithm. This helps identify user segments for better personalization. Additionally, **Collaborative Filtering** with SVD was applied to recommend products based on user-product interactions. Other methods like Regression or Classification were not suitable, as the recommendation problem focuses on predicting user preferences rather than labeled outputs. Similarly, Decision Trees or Anomaly Detection were not directly applicable since the task is not about decision boundaries or detecting outliers. Thus, **Clustering** combined with **Collaborative Filtering** was most appropriate for capturing user similarity and generating personalized recommendations.