Name: Anish Kulkarni Roll No.: 29 Class: D15C AY: 2024-25

Experiment 8

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

- Regression
- Classification
- Clustering
- Decision tree
- Anomaly detection
- Dimensionality Reduction
- Ensemble Methods

Theory:

- **Recommendation types:** Recommendation systems help users discover relevant items by predicting their preferences. The main types include content-based filtering, collaborative filtering, and hybrid methods. Content-based filtering recommends items similar to those the user has liked before, using item features such as category, description, or brand. Collaborative filtering, on the other hand, leverages the preferences of similar users or the patterns in user-item interactions without requiring item-specific data. It can be user-based or item-based, depending on whether it finds similarities between users or between items. Hybrid systems combine both methods to overcome limitations such as data sparsity or the cold-start problem, often resulting in better performance.

- **Recommendation measures:** Evaluating recommendation systems is essential to measure their effectiveness and accuracy. For prediction-based systems, RMSE (Root Mean Square Error) and MAE (Mean Absolute Error) are widely used to quantify the difference between predicted ratings and actual user ratings. Lower values indicate better accuracy. For ranking-based recommendations, metrics like precision, recall, and F1-score assess the system's ability to rank relevant items higher in a list. Additionally, Mean Reciprocal Rank (MRR) and Normalized Discounted Cumulative Gain (NDCG) are used to measure the quality of ranked recommendations. These metrics help determine how well the system delivers useful and personalized results to users.

Performance:

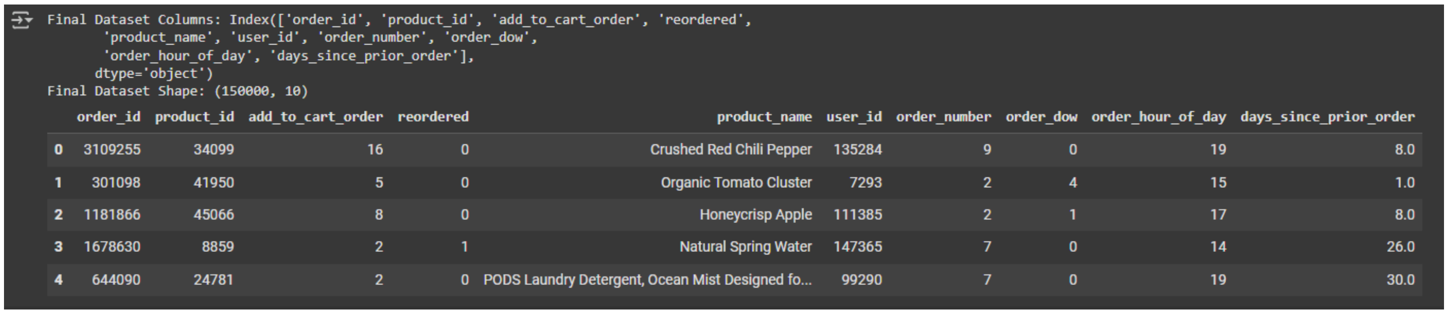
- Prerequisite: The code starts by importing essential libraries for data manipulation (pandas, numpy), visualization (matplotlib.pyplot, seaborn), and modeling (sklearn for K-Means clustering and scaling, and surprise for collaborative filtering using SVD). It then loads six CSV files using pd.read_csv(): orders.csv, order_products__prior.csv, order_products__train.csv, products.csv, aisles.csv, and departments.csv. These datasets contain user order history and product details, which are used to build clustering and recommendation models.

```
Command: 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
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")
```

- Preprocessing and Merging separate csv files into a single cohesive dataset:

```
Command: 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()

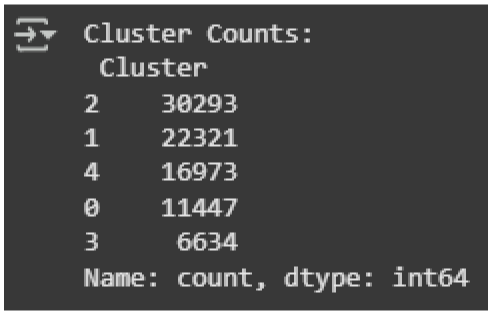


```
Final Dataset Columns: Index(['order_id', 'product_id', 'add_to_cart_order', 'reordered',
       'product_name', 'user_id', 'order_number', 'order_dow',
       'order_hour_of_day', 'days_since_prior_order'],
      dtype='object')
Final Dataset Shape: (150000, 10)
```

	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

The code efficiently preprocesses the Instacart dataset by selecting only essential columns, including product_name, to minimize memory usage while preserving key product details. It merges the products dataset with aisles and departments to enrich each product entry with hierarchical information. The merging of order_products_prior and order_products_train with product details is performed to associate each ordered product with its name and category, enabling meaningful insights and recommendations. Sampling is used to reduce dataset size for faster processing. Finally, merging with the orders dataset adds user-level and temporal context to each transaction, which is crucial for building personalized recommendation systems.

Step 1: Clustering Users Based on Shopping Behavior:
Command: 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())

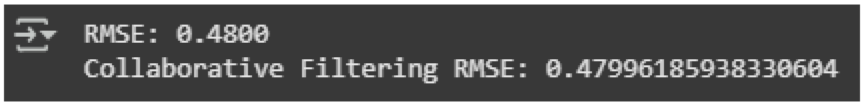


```
Cluster Counts:
 Cluster
2    30293
1    22321
4    16973
0    11447
3     6634
Name: count, dtype: int64
```

This code clusters users based on their shopping behavior to uncover distinct patterns among customer segments. First, it aggregates user-level features like the average number of orders, time between orders, and cart position. Missing values in these features are handled using mean imputation to ensure model reliability. The features are then normalized using StandardScaler to bring all variables to a common scale. Finally, K-Means clustering with 5 clusters is applied to group users with similar purchasing habits, and the distribution of users across clusters is displayed to understand segment sizes. This clustering can later help tailor recommendations to user groups with shared behaviors.

Step 2: Collaborative Filtering Using Matrix Factorization (SVD):

```
Command: 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)
```



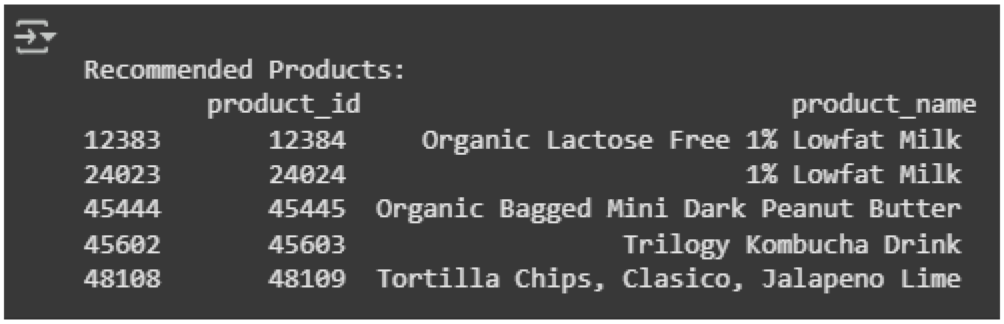
```
RMSE: 0.4800
Collaborative Filtering RMSE: 0.47996185938330604
```

This code implements collaborative filtering to recommend products based on user-product interactions. It begins by preparing a dataset with user_id, product_id, and reordered as the implicit rating, where a reorder indicates user preference. The Surprise library's Reader and Dataset modules convert the DataFrame into a suitable format, followed by an 80-20 train-test split. A Singular Value Decomposition (SVD) model is trained on the training set to learn latent features of users and products. Finally, predictions are made on the test set and evaluated using RMSE (Root Mean Square Error) to measure the model’s accuracy in predicting reorder behavior.

The model achieved a Collaborative Filtering RMSE of 0.47996, which suggests a relatively good level of predictive accuracy. A lower RMSE indicates that the predicted reorder values are close to the actual values, meaning the model is effective at capturing user preferences and recommending products that align with their past shopping behavior.

Step 3: Generating Personalized Product Recommendations:
Command:

```
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)
```



```
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
```

The above function leverages the trained SVD collaborative filtering model to recommend products to a specific user. It begins by identifying all unique products in the dataset and predicts the reorder likelihood for each using the model. The top-N products with the highest predicted scores (by default, 5) are selected and mapped to their respective names using the products DataFrame. This process ensures that the recommendations are both accurate and interpretable.
As shown above, when executed for user_id=1, the model recommended five specific items based on the user's shopping behavior: Organic Lactose Free 1% Lowfat Milk, 1% Lowfat Milk, Organic Bagged Mini Dark Peanut Butter, Trilogy Kombucha Drink, and Tortilla Chips, Clasico, Jalapeno Lime. These results illustrate how collaborative filtering can successfully generate personalized product suggestions that are relevant and likely to be of interest to the user.

Conclusion:

1. In this experiment, we learned how to perform a recommendation system on our dataset using the 'Clustering' machine learning technique.
2. K-Means clustering was effectively applied to group users based on behavioral patterns such as order frequency, recency, and cart activity, allowing us to uncover distinct user segments with similar shopping habits.

3. Collaborative Filtering using Matrix Factorization (SVD) demonstrated strong predictive performance by achieving a low RMSE of 0.47996, indicating the model’s ability to accurately estimate user preferences and reorder likelihood.
4. The recommendation engine successfully generated relevant product suggestions such as Organic Lactose Free 1% Lowfat Milk, Trilogy Kombucha Drink, and Tortilla Chips, reflecting the system's strength in delivering personalized and appealing recommendations.
5. By leveraging user purchase history and clustering, the system enhanced the understanding of user needs, which is crucial for building intelligent recommendation systems that can increase user engagement and satisfaction.
6. In conclusion, the integration of clustering for user segmentation and collaborative filtering for recommendation generation forms a powerful hybrid approach that not only improves accuracy but also helps tailor the shopping experience more effectively.