2024

Customer Purchase Behavior in a Supermarket

Capstone Project - Clustering

DIBOSH BARUAH

1. Introduction

Customer behavior in a supermarket context is a critical factor for improving business strategies such as marketing, inventory management, and product placement. By clustering customers based on their purchasing behavior, supermarkets can identify distinct customer segments, tailor product recommendations, optimize promotions, and improve operational efficiency. This project utilizes unsupervised machine learning techniques to analyze and segment supermarket customers, focusing on their order patterns, product choices, and overall shopping behavior.

The goal of this analysis is to cluster customers using purchase-related features and provide actionable insights that businesses can leverage to enhance customer engagement and streamline their operations.

2. Data Collection

The dataset used in this project is derived from a supermarket's transaction records. The data consists of several features that capture customer order behaviors, including:

- **order_id**: Unique identifier for each order.
- **user_id**: Unique identifier for each customer.
- **order_number**: The sequential number of the order placed by a user.
- **order_dow**: The day of the week the order was placed.
- **order_hour_of_day**: The hour of the day when the order was placed.
- **days_since_prior_order**: The number of days since the customer's last order.
- **product_id**: ID of the product purchased.
- **add_to_cart_order**: The position of the product in the shopping cart.
- **reordered**: Indicates if the product was previously ordered.
- **department_id**: The department to which the product belongs.
- **department**: Name of the product department.
- **product_name**: Name of the product.

This dataset provides a comprehensive view of customer purchasing patterns, which can be used for further segmentation and clustering.

3. Data Preprocessing

Before performing clustering, several preprocessing steps were applied to clean and prepare the data:

3.1 Handling Missing Values

- **Missing values in the 'days_since_prior_order' column**: The missing values were filled with the median value of this column to ensure no data is lost.
- **No missing values were identified in other columns** based on the output from data.isnull().sum().

3.2 Feature Engineering

- **Dropping Irrelevant Columns**: The order_id column was dropped as it was not relevant for clustering.
- **Binary Column Conversion**: Any Boolean columns (if present) were converted to integers (0 or 1).

3.3 Scaling

- **MinMax Scaling**: Continuous numerical features (e.g., order_number, days_since_prior_order, etc.) were scaled using MinMax scaling to normalize the data between 0 and 1, ensuring that all features contribute equally during clustering.

4. Clustering and Number of Clusters

4.1 Feature Selection

The selected features for clustering include:

- **order_number**: Indicates how many orders the customer has placed, which is an important feature to measure customer loyalty.
- **days_since_prior_order**: Reflects customer frequency and purchase recency, which are crucial for segmentation.
- **add_to_cart_order**: Represents how often a customer adds products to their cart, providing insight into shopping behavior.
- **reordered**: This feature is essential for understanding repeat purchase behavior.
- **unique_products**: The number of unique products a customer has purchased, which reflects variety in purchasing behavior.

4.2 Clustering Model (DBSCAN)

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm was chosen because it can identify clusters of varying shapes and sizes and can handle noise effectively. The model was applied with the following parameters:

- eps=0.5: This parameter defines the maximum distance between two points for them to be considered as in the same neighborhood.
- min_samples=5: Minimum number of samples in a neighborhood to form a dense region (cluster).

The DBSCAN algorithm assigns each customer to a cluster or labels them as noise (denoted by -1).

4.3 Results of Clustering

After applying DBSCAN, customers were clustered into different segments based on their purchasing patterns. The results showed a range of clusters, with some customers being categorized as noise (those with no clear cluster, labeled as -1).

5. Model Evaluation

5.1 Silhouette Score

To evaluate the quality of the clusters, the **Silhouette Score** was calculated. The silhouette score measures how similar an object is to its own cluster compared to other clusters, with a score close to 1 indicating well-defined clusters. The computed silhouette score for the model was **0.50**, indicating that the clustering performed moderately well but could be improved.

5.2 Cluster Visualization

To visualize the clusters, Principal Component Analysis (PCA) was applied to reduce the data to two dimensions. A scatter plot was created to show the distribution of clusters based on the first two principal components.

6. Results & Insights

6.1 Cluster Characteristics

The clusters identified by DBSCAN reveal distinct customer groups. Some customers are highly loyal (frequent reorders and higher average cart size), while others make less frequent purchases but tend to order larger volumes or unique products.

- **Cluster 0**: Customers who order regularly with a moderate cart size and frequent reorders.
- **Cluster 1**: Customers who order less frequently, often placing large orders in terms of unique product variety.
- **Cluster 2**: Customers who place orders occasionally but tend to reorder products frequently.
- **Cluster 3**: Customers with sporadic ordering behavior and lower variety in product choice.

6.2 Business Insights

The insights drawn from the clustering can help supermarket businesses tailor their strategies:

- **Frequent Reorderers** (Cluster 2) should be targeted with loyalty programs or subscription models to enhance retention.
- **High Variety Shoppers** (Cluster 1) could be presented with personalized promotions based on product categories they tend to explore.
- **Occasional Shoppers** (Cluster 3) might benefit from targeted discounts or offers that encourage repeat purchases.

By understanding these segments, supermarkets can better manage inventory, create targeted marketing campaigns, and optimize product placements to meet the needs of each group.

7. Conclusion

This project successfully demonstrated the use of unsupervised learning (DBSCAN clustering) to segment customers based on their purchase behavior. The resulting clusters provide meaningful insights into customer loyalty, product preferences, and shopping frequency. The use of clustering models like DBSCAN helped identify distinct patterns that would be difficult to capture manually.

Despite the promising results, further optimization of clustering parameters (like `eps` and `min_samples`) and the inclusion of additional features (e.g., seasonal trends, promotions) could improve the segmentation accuracy.

8.Original Code:

```
import sqlite3
import pandas as pd

# Connecting to the database
conn = sqlite3.connect('/Users/diboshbaruah/Desktop/Database.db')
data = pd.read_sql_query('SELECT * FROM Supermarket_data', conn)

print("Dataset successfully loaded...\n")
# Display the first few rows to inspect the data
print("Displaying first few rows of the dataset:\n")
print(data.head())

# Checking data types before conversion
print("\nData types before conversion:")
print(data.dtypes)

# Closing the connection
conn.close()
```

```
Dataset successfully loaded...

Displaying first few rows of the dataset:

   order_id  user_id  order_number  order_dow  order_hour_of_day  \
0   1253241   152060             6          1                 20
1   3058717    44755             2          1                 15
2   2252307   169119            12          4                 15
3    188072   162421             3          4                 11
4   2627597   172693            19          0                 23

   days_since_prior_order  product_id  add_to_cart_order  reordered  \
0                    23.0         115                  2          1
1                    30.0          37                  4          0
2                     9.0         123                 19          0
3                    30.0         117                  5          0
4                     5.0          17                  8          0

   department_id department                   product_name
0              7   beverages  water seltzer sparkling water
1              1      frozen                  ice cream ice
2              4     produce     packaged vegetables fruits
3             19      snacks           nuts seeds dried fruit
4             13      pantry             baking ingredients
```

```
Data types before conversion:
order_id                    int64
user_id                     int64
order_number                int64
order_dow                   int64
order_hour_of_day           int64
days_since_prior_order    float64
product_id                  int64
add_to_cart_order           int64
reordered                   int64
department_id               int64
department                 object
product_name               object
dtype: object
```

```
# Check for missing values in the scaled data
missing_values = data.isnull().sum()

# Display the number of missing values for each column
print("Missing values in each column:\n")
print(missing_values)
```

```
Missing values in each column:

order_id                      0
user_id                       0
order_number                  0
order_dow                     0
order_hour_of_day             0
days_since_prior_order    61437
product_id                    0
add_to_cart_order             0
reordered                     0
department_id                 0
department                    0
product_name                  0
dtype: int64
```

```
# Fill missing values in 'days_since_prior_order' with the median
data['days_since_prior_order'] =
data['days_since_prior_order'].fillna(data['days_since_prior_order'].median()
)

# Check if there are any missing values left
missing_values = data.isnull().sum()
```

```python
# Display the number of missing values for each column
print("\nMissing values in each column after filling:\n")
print(missing_values)
```

```
Missing values in each column after filling:

order_id                  0
user_id                   0
order_number              0
order_dow                 0
order_hour_of_day         0
days_since_prior_order    0
product_id                0
add_to_cart_order         0
reordered                 0
department_id             0
department                0
product_name              0
dtype: int64
```

```python
# Drop the 'order_id' column
data_cleaned = data.drop(columns=['order_id'])

from sklearn.preprocessing import MinMaxScaler

# Convert the boolean columns (one-hot encoded) to integers (0 or 1)
binary_columns = data_cleaned.select_dtypes(include=['bool']).columns
data_cleaned[binary_columns] = data_cleaned[binary_columns].astype(int)

# Identify the numerical columns (which are int or float, but excluding the
ones that were converted to int from bool)
numerical_columns = data_cleaned.select_dtypes(include=['int64',
'float64']).columns

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Apply scaling only to the continuous numerical columns
data_cleaned[numerical_columns] =
scaler.fit_transform(data_cleaned[numerical_columns])

# Aggregate data by customer (user_id)
aggregated_data = data_cleaned.groupby('user_id').agg({
    'order_number': 'max',  # Max order number indicates total orders made
    'days_since_prior_order': 'mean',  # Average time between orders
    'add_to_cart_order': 'mean',  # Average cart size
    'reordered': 'mean',  # Proportion of reorders
}).reset_index()
```

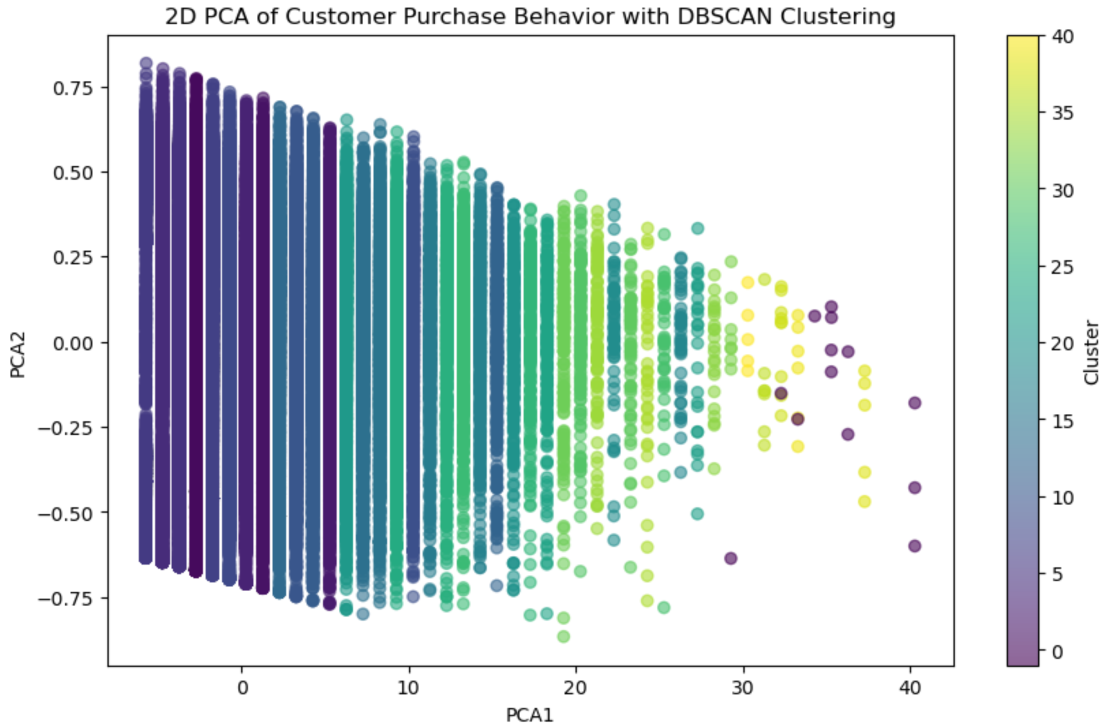
```
# Optionally, add the number of unique products purchased by each customer
aggregated_data['unique_products'] =
data_cleaned.groupby('user_id')['product_id'].nunique().values

# Display the first few rows of the aggregated data
print("\nAggregated data per customer:\n")
print(aggregated_data.head())
```

```
Aggregated data per customer:

    user_id  order_number  days_since_prior_order  add_to_cart_order  \
0  0.000000      0.020202                0.266667           0.014706
1  0.000005      0.101010                0.388889           0.022059
2  0.000024      0.020202                1.000000           0.075630
3  0.000039      0.030303                0.466667           0.106900
4  0.000044      0.040404                1.000000           0.044118

   reordered  unique_products
0   0.285714                4
1   0.333333                8
2   0.785714               12
3   0.230769                8
4   0.375000                7
```

```
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Drop 'user_id' column as it is not needed for clustering
X = aggregated_data.drop(columns=['user_id'])

# Define the DBSCAN model (You can experiment with different values for 'eps'
and 'min_samples')
dbscan = DBSCAN(eps=0.5, min_samples=5)  # Example values for eps and
min_samples

# Fit the model
aggregated_data['cluster'] = dbscan.fit_predict(X)

# View the first few rows with cluster labels
print("\nFirst few rows after DBSCAN clustering:")
print(aggregated_data.head())

# Calculate silhouette score (only for points that are not noise, i.e., label
!= -1)
mask = aggregated_data['cluster'] != -1  # Mask for noise points (label -1)
sil_score = silhouette_score(X[mask], aggregated_data.loc[mask, 'cluster'])
print(f"Silhouette Score: {sil_score}")
```

```python
# Visualize the clusters using PCA for dimensionality reduction to 2D
from sklearn.decomposition import PCA

# Perform PCA to reduce the data to 2D for visualization
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(X)

# Create a DataFrame with PCA components and the assigned clusters
reduced_df = pd.DataFrame(reduced_data, columns=['PCA1', 'PCA2'])
reduced_df['Cluster'] = aggregated_data['cluster']

# Plot the clusters
plt.figure(figsize=(10, 6))
plt.scatter(reduced_df['PCA1'], reduced_df['PCA2'], c=reduced_df['Cluster'],
cmap='viridis', alpha=0.6)
plt.title('2D PCA of Customer Purchase Behavior with DBSCAN Clustering')
plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.colorbar(label='Cluster')
plt.show()
```

```
First few rows after DBSCAN clustering:
    user_id  order_number  days_since_prior_order  add_to_cart_order  \
0  0.000000      0.020202                0.266667           0.014706
1  0.000005      0.101010                0.388889           0.022059
2  0.000024      0.020202                1.000000           0.075630
3  0.000039      0.030303                0.466667           0.106900
4  0.000044      0.040404                1.000000           0.044118

   reordered  unique_products  cluster
0   0.285714                4        0
1   0.333333                8        1
2   0.785714               12        2
3   0.230769                8        1
4   0.375000                7        3


Silhouette Score: 0.5024030725347061
```


2D PCA of Customer Purchase Behavior with DBSCAN Clustering
PCA2
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
PCA1
0
10
20
30
40
Cluster
40
35
30
25
20
15
10
5
0