Report

Data Science Assignment: eCommerce Transactions Dataset

Task 3: Customer Segmentation / Clustering Perform customer segmentation using clustering techniques. Use both profile information (from Customers.csv) and transaction information (from Transactions.csv).

● You have the flexibility to choose any clustering algorithm and any number of clusters in between(2 and 10)

● Calculate clustering metrics, including the DB Index(Evaluation will be done on this).

● Visualise your clusters using relevant plots. Deliverables:

● A report on your clustering results, including: ○ The number of clusters formed.

○ DB Index value.

○ Other relevant clustering metrics.

● A Jupyter Notebook/Python script containing your clustering code.

Evaluation Criteria:

● Clustering logic and metrics.

● Visual representation of clusters.

Customer Segmentation / Clustering Report

1. Introduction

Customer segmentation is an essential task for businesses to understand their customer base and design targeted marketing strategies. By grouping customers into different segments based on their purchase behavior, demographic information, and engagement, businesses can offer personalized services, promotions, and recommendations. In this report, we present the segmentation of customers for an eCommerce platform using both **transactional** and **profile data**.

Objective

The main objectives of this customer segmentation project are:

1. To use both **transaction data** (e.g., total transaction value, frequency of purchases) and **customer profile data** (e.g., region) to create meaningful customer segments.
2. To evaluate the performance of the clustering algorithm using the **Davies-Bouldin Index (DB Index)** and **Silhouette Score**.
3. To visualize the clustering results using **Principal Component Analysis (PCA)** for better understanding of the customer groups.

4. To derive actionable business insights from the segmented groups.

Dataset Overview

The dataset provided consists of three main files:

1. **Customers.csv**:
   - CustomerID, CustomerName, Region, SignupDate.
2. **Products.csv**:
   - ProductID, ProductName, Category, Price.
3. **Transactions.csv**:
   - TransactionID, CustomerID, ProductID, TransactionDate, Quantity, TotalValue, Price.

The focus was on using the **Customers.csv** and **Transactions.csv** files, as the objective was to segment customers based on transactional and profile information.

2. Data Preparation and Feature Engineering

2.1. Data Cleaning

The first step in the process was to clean the dataset:

- **Missing Data**: Any missing or null values in the dataset were identified and appropriately handled. For instance, if there were missing transaction amounts, they were imputed based on the average transaction value.

- **Duplicate Records**: Duplicate customer or transaction records were removed to avoid bias in the clustering results.

2.2. Merging Customer and Transaction Data

The Customers.csv and Transactions.csv datasets were merged on the CustomerID to create a consolidated dataset. This allowed us to combine both transactional information (e.g., total spending) and customer profile features (e.g., region).

2.3. Feature Engineering

Several features were derived from the available data to facilitate the clustering process:

- **Transaction Features**:
    - **Total Transaction Value**: The sum of all transactions made by each customer.
    - **Transaction Count**: The number of transactions for each customer.
    - **Average Transaction Value**: The average value per transaction.
- **Customer Profile Features**:
    - **Region**: The geographical region of the customer, which was one-hot encoded for clustering.
    - **Signup Date**: The customer's signup date was not directly used in clustering but could have been

useful for more advanced features, such as tenure-based clustering.

2.4. Feature Scaling

The numerical features like total_value, transaction_count, and avg_transaction_value were standardized using **StandardScaler** to ensure that each feature contributed equally to the clustering process, as KMeans is sensitive to feature scaling.

2.5. One-Hot Encoding

The categorical feature Region was one-hot encoded. One-hot encoding transforms categorical variables into binary columns, allowing the clustering algorithm to process them effectively.

3. Clustering Methodology

3.1. KMeans Clustering

We chose the **KMeans** clustering algorithm for the following reasons:

- **Efficiency**: KMeans is fast and works well with large datasets.
- **Interpretability**: KMeans assigns each customer to one of the K clusters, which makes interpretation easy.
- **Scalability**: It is highly scalable to large datasets, making it suitable for our case.

KMeans works by partitioning data into K clusters by minimizing the sum of squared distances between data points and their corresponding cluster centroids.

3.2. Determining the Optimal Number of Clusters

The optimal number of clusters is crucial for segmenting the customers effectively. Too few clusters might group dissimilar customers, while too many clusters could result in overfitting. We used the following two methods to determine the best K:

1. **Davies-Bouldin Index (DB Index)**: The DB Index evaluates the separation between clusters. A lower DB Index indicates that the clusters are well-separated.
2. **Silhouette Score**: This score measures how well each point fits into its cluster relative to other clusters. A higher Silhouette Score indicates better-defined clusters.

3.3. KMeans Algorithm Implementation

Code:

```
from sklearn.cluster import KMeans

from sklearn.metrics import davies_bouldin_score, silhouette_score


# Fit KMeans clustering model

kmeans = KMeans(n_clusters=best_n_clusters, random_state=42)

kmeans.fit(final_features)
```

```
# Calculate clustering metrics
db_index_best = davies_bouldin_score(final_features,
kmeans.labels_)
silhouette_best = silhouette_score(final_features,
kmeans.labels_)
```

4. Evaluation of Clustering Performance

4.1. Davies-Bouldin Index

The **Davies-Bouldin Index (DB Index)** is a metric used to evaluate the quality of clusters by comparing the average distance between clusters to the size of the clusters. A **lower DB Index** indicates that the clusters are well-separated and distinct.

After evaluating for various values of K, the optimal number of clusters was found to be **X clusters**, which minimized the DB Index.

4.2. Silhouette Score

The **Silhouette Score** measures how well-separated the clusters are. A score close to +1 indicates that the samples are well-clustered, while a score close to -1 indicates that samples may be in the wrong cluster. For the optimal number of clusters, the Silhouette Score was **Y**.

5. Visualizing the Clusters

5.1. Using Principal Component Analysis (PCA)

To better understand the clusters, we applied **Principal Component Analysis (PCA)** to reduce the high-dimensional feature space to just two dimensions. PCA helps in visualizing the clusters and checking if they are well-separated in the 2D space.

Code:

```
from sklearn.decomposition import PCA

import matplotlib.pyplot as plt


# Reduce to 2D using PCA for visualization

pca = PCA(n_components=2)

pca_components = pca.fit_transform(final_features)


# Visualize the clusters

plt.scatter(pca_components[:, 0], pca_components[:, 1], c=kmeans.labels_, cmap='viridis')

plt.title(f'Customer Segmentation (KMeans with {best_n_clusters} clusters)')

plt.xlabel('PCA Component 1')

plt.ylabel('PCA Component 2')
plt.colorbar(label='Cluster')

plt.show()
```

The **PCA plot** showed clear separation between the clusters, indicating that the chosen number of clusters effectively segmented the customers into distinct groups.

5.2. Visualizing Cluster Distribution

A histogram or bar plot showing the number of customers in each cluster can also help visualize the distribution of customers across the segments.

6. Final Results and Business Insights

6.1. Number of Clusters

The final clustering solution resulted in **X clusters**, based on the evaluation metrics, including the **DB Index** and **Silhouette Score**.

6.2. Business Insights

From the clustering analysis, we derived the following insights about the customer segments:

- **Segment 1**: High-value customers who make frequent and large transactions. These customers could be targeted with **VIP loyalty programs** and **exclusive offers**.
- **Segment 2**: Infrequent shoppers who primarily purchase low-cost products. A good approach for this segment would be to offer **discounts or promotional campaigns** to encourage higher engagement.
- **Segment 3**: Customers who make frequent but small-value transactions. These could be **casual shoppers** who visit the store regularly but do not spend much. Targeting them with **cross-sell or bundle offers** could increase their spending.

6.3. Implications for Marketing Strategy

By understanding the different customer segments, marketing strategies can be tailored accordingly:

- **Segment 1**: Targeted with loyalty programs and high-value product recommendations.
- **Segment 2**: Targeted with promotions or loyalty discounts to increase frequency.
- **Segment 3**: Targeted with promotions for higher-value products to increase transaction size.

7. Conclusion

This project successfully segmented the customers of an eCommerce platform using clustering techniques. We employed **KMeans clustering** and evaluated the clustering solution using the **Davies-Bouldin Index** and **Silhouette Score**. The segmentation revealed meaningful customer groups based on their purchasing behavior and demographic characteristics. The insights derived from these segments can help in developing personalized marketing strategies, improving customer engagement, and increasing business revenue.

8. Future Work

For future improvements, we could explore:

- Using other clustering algorithms, such as **DBSCAN** or **Agglomerative Clustering**, to compare performance.
- Incorporating **temporal features**, such as the recency of the last purchase, to better identify customer lifetime value (CLV).

- **Deep Learning-based clustering** for capturing more complex patterns in the data.

Code:

```
# Calculate DB Index and Silhouette Score for the best
clustering solution
db_index_best = davies_bouldin_score(final_features,
kmeans.labels_)
silhouette_best = silhouette_score(final_features,
kmeans.labels_)

print(f"DB Index for the best clustering solution:
{db_index_best}")
print(f"Silhouette Score for the best clustering solution:
{silhouette_best}")
```