“Customer Behavior Analysis in E-Commerce: A Comprehensive Exploration”

CLASSICAL ALGORITHM IMPLEMENTATION

Team Members:

Chikkeri Chinmaya – 211IT017

Umesh – 211IT073

Vishwa Mohan Reddy G – 211IT082

Vismay P – 211IT083

1) **k-means algo**
   “The k-means Algorithm: A Comprehensive Survey and Performance Evaluation” by Mohiuddin Ahmed, Mohiuddin Ahmed and Syed Mohammed Shamsul Islam (2020)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# Load customer data
customer_data = pd.read_csv('customer_data.csv')

# Select relevant features for segmentation (One can customize this)
selected_features = ['Feature1', 'Feature2', 'Feature3']
X = customer_data[selected_features]

# Standardize the data to have mean=0 and standard deviation=1
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Determine the optimal number of clusters using the Elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300,
n_init=10, random_state=0)
    kmeans.fit(X_scaled)
    wcss.append(kmeans.inertia_)

# Plot the Elbow method graph
plt.figure(figsize=(8, 6))
plt.plot(range(1, 11), wcss)
```

```python
plt.title('Elbow Method for Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS (Within-Cluster Sum of Squares)')
plt.show()

# Based on the Elbow method, choose the optimal number of clusters
optimal_k = 3  # Adjust this based on the plot

# Perform K-Means clustering with the selected number of clusters
kmeans = KMeans(n_clusters=optimal_k, init='k-means++', max_iter=300,
n_init=10, random_state=0)
kmeans.fit(X_scaled)

# Add the cluster labels to the original customer data
customer_data['Cluster'] = kmeans.labels_

# Analyze the segments
segmented_data = customer_data.groupby('Cluster').mean()

# Print the summary statistics of each cluster
print(segmented_data)

# Visualize the clusters (for 2D or 3D data)
if len(selected_features) == 2:
    plt.figure(figsize=(8, 6))
    for cluster in range(optimal_k):
        plt.scatter(X_scaled[kmeans.labels_ == cluster][:, 0],
X_scaled[kmeans.labels_ == cluster][:, 1], label=f'Cluster {cluster}')
    plt.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], s=300, c='red', label='Centroids')
    plt.title('Customer Segmentation')
    plt.xlabel('Feature1')
    plt.ylabel('Feature2')
    plt.legend()
    plt.show()
elif len(selected_features) == 3:
    from mpl_toolkits.mplot3d import Axes3D
    fig = plt.figure(figsize=(10, 8))
    ax = fig.add_subplot(111, projection='3d')
    for cluster in range(optimal_k):
        ax.scatter(X_scaled[kmeans.labels_ == cluster][:, 0],
X_scaled[kmeans.labels_ == cluster][:, 1], X_scaled[kmeans.labels_ ==
cluster][:, 2], label=f'Cluster {cluster}')
    ax.scatter(kmeans.cluster_centers_[:, 0],
kmeans.cluster_centers_[:, 1], kmeans.cluster_centers_[:, 2], s=300,
c='red', label='Centroids')
    ax.set_title('Customer Segmentation')
    ax.set_xlabel('Feature1')
```

```
    ax.set_ylabel('Feature2')
    ax.set_zlabel('Feature3')
    ax.legend()
    plt.show()
```

In this code, we load customer data, select relevant features for segmentation, st andardize the data, and determine the optimal number of clusters using the Elbo w method. After performing KMeans clustering, we add cluster labels to the ori ginal data and analyze the segments. Finally, we visualize the clusters if the dat a is 2D or 3D

2) Behavioural analysis

"Foundations of Consumer Behaviour Analysis" by Gordon R. Foxall (2001)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# Load customer transaction data (sample data)
data = pd.read_csv('customer_transactions.csv')

# Assuming the dataset has columns like 'CustomerID',
'TransactionDate', 'AmountSpent', 'ProductCategory', etc.

# Calculate total spending per customer
total_spending = data.groupby('CustomerID')['AmountSpent'].sum()

# Calculate the frequency of transactions per customer
transaction_frequency = data.groupby('CustomerID').size()

# Calculate the average transaction amount per customer
average_transaction_amount =
data.groupby('CustomerID')['AmountSpent'].mean()

# Calculate the recency of transactions (assuming a reference date)
reference_date = pd.to_datetime('2023-01-01')
data['TransactionDate'] = pd.to_datetime(data['TransactionDate'])
recency = data.groupby('CustomerID')['TransactionDate'].max()
recency = (reference_date - recency).dt.days  # Recency in days

# Create a customer behavior dataframe
customer_behavior = pd.DataFrame({
    'TotalSpending': total_spending,
    'TransactionFrequency': transaction_frequency,
    'AverageTransactionAmount': average_transaction_amount,
    'Recency': recency
})

# Visualize customer behavior
plt.figure(figsize=(12, 8))

# Plot Total Spending vs. Transaction Frequency
plt.subplot(2, 2, 1)
plt.scatter(customer_behavior['TotalSpending'],
customer_behavior['TransactionFrequency'])
plt.title('Total Spending vs. Transaction Frequency')
plt.xlabel('Total Spending')
plt.ylabel('Transaction Frequency')
```

```python
# Plot Total Spending vs. Average Transaction Amount
plt.subplot(2, 2, 2)
plt.scatter(customer_behavior['TotalSpending'],
customer_behavior['AverageTransactionAmount'])
plt.title('Total Spending vs. Average Transaction Amount')
plt.xlabel('Total Spending')
plt.ylabel('Average Transaction Amount')

# Plot Total Spending vs. Recency
plt.subplot(2, 2, 3)
plt.scatter(customer_behavior['TotalSpending'],
customer_behavior['Recency'])
plt.title('Total Spending vs. Recency')
plt.xlabel('Total Spending')
plt.ylabel('Recency (days)')

# Plot Transaction Frequency vs. Recency
plt.subplot(2, 2, 4)
plt.scatter(customer_behavior['TransactionFrequency'],
customer_behavior['Recency'])
plt.title('Transaction Frequency vs. Recency')
plt.xlabel('Transaction Frequency')
plt.ylabel('Recency (days)')

plt.tight_layout()
plt.show()
```

In this code, we calculate customer behavior metrics such as total spending, transaction frequency, average transaction amount, and recency. Then, we visualize the relationships between these metrics. The code plots four scatter plots to visualize how total spending correlates with other behavior metrics.

3) **Rfm analysis (recency, frequency, monetary)**

"Customer Segmentation Based On Recency Frequency Monetary Model: A Case Study in E-Retailing" by KABASAKAL (2020)

```
import pandas as pd
import datetime as dt

# Load customer transaction data (sample data)
data = pd.read_csv('customer_transactions.csv')

# Assuming the dataset has a 'CustomerID', 'TransactionDate', and
'Amount' columns

# Convert 'TransactionDate' to datetime
data['TransactionDate'] = pd.to_datetime(data['TransactionDate'])

# Calculate the current date
current_date = max(data['TransactionDate'])

# Calculate Recency, Frequency, and Monetary Value for each customer
rfm_data = data.groupby('CustomerID').agg({
    'TransactionDate': lambda x: (current_date - x.max()).days,  #
Recency
    'TransactionID': 'count',  # Frequency
    'Amount': 'sum'  # Monetary Value
})

# Rename the columns
rfm_data.rename(columns={
    'TransactionDate': 'Recency',
    'TransactionID': 'Frequency',
    'Amount': 'Monetary'
}, inplace=True)

# Print the RFM data
print(rfm_data)

# Define RFM score quartiles
quantiles = rfm_data.quantile(q=[0.25, 0.5, 0.75])

# Function to create RFM segments
def create_rfm_segments(data, quantiles):
    r_segment, f_segment, m_segment = '', '', ''

    if data['Recency'] <= quantiles['Recency'][0.25]:
        r_segment = '4'
```

```
    elif data['Recency'] <= quantiles['Recency'][0.5]:
        r_segment = '3'
    elif data['Recency'] <= quantiles['Recency'][0.75]:
        r_segment = '2'
    else:
        r_segment = '1'

    if data['Frequency'] <= quantiles['Frequency'][0.25]:
        f_segment = '1'
    elif data['Frequency'] <= quantiles['Frequency'][0.5]:
        f_segment = '2'
    elif data['Frequency'] <= quantiles['Frequency'][0.75]:
        f_segment = '3'
    else:
        f_segment = '4'

    if data['Monetary'] <= quantiles['Monetary'][0.25]:
        m_segment = '1'
    elif data['Monetary'] <= quantiles['Monetary'][0.5]:
        m_segment = '2'
    elif data['Monetary'] <= quantiles['Monetary'][0.75]:
        m_segment = '3'
    else:
        m_segment = '4'

    return r_segment + f_segment + m_segment

# Apply the create_rfm_segments function to create RFM segments
rfm_data['RFM_Segment'] = rfm_data.apply(create_rfm_segments,
args=(quantiles,), axis=1)

# Calculate RFM score
rfm_data['RFM_Score'] = rfm_data['Recency'].astype(str) +
rfm_data['Frequency'].astype(str) + rfm_data['Monetary'].astype(str)

# Print the RFM segments and scores
print(rfm_data[['RFM_Segment', 'RFM_Score']])
```

In this code, we first load the customer transaction data, calculate Recency (ho w recently a customer made a purchase), Frequency (how often a customer mak es a purchase), and Monetary Value (how much a customer spends), and then c reate RFM segments and scores. Onecan use these segments and scores for custo mer segmentation and targeted marketing.

4) Customer lifetime value analysis

"Investigating Two Customer Lifetime Value Models from Segmentation Perspective" by Abdulkadir Hiziroglu and Serkan Sengul (2012)

```
import pandas as pd
from lifetimes import BetaGeoFitter, GammaGammaFitter
import matplotlib.pyplot as plt

# Load your transaction data into a Pandas DataFrame
data = pd.read_csv('customer_data.csv')

# Data Preprocessing
data['date'] = pd.to_datetime(data['date'])  # Convert the date column
to a datetime format
summary = pd.pivot_table(data, values='monetary_value',
index='customer_id', aggfunc=['count', 'mean', 'sum'])
summary.columns = ['frequency', 'monetary_mean', 'monetary_sum']

# Split your data into calibration and holdout periods
calibration_period_end = '2022-01-01'
holdout_period_start = '2022-01-02'
calibration_data = data[data['date'] <= calibration_period_end]
holdout_data = data[(data['date'] > calibration_period_end) &
(data['date'] >= holdout_period_start)]

# Initialize and fit the BG/NBD model
bgf = BetaGeoFitter(penalizer_coef=0.0)
bgf.fit(calibration_data['frequency'], calibration_data['recency'],
calibration_data['T'])

# Calculate the expected number of future purchases for a given
customer
summary['predicted_purchases'] = bgf.predict(holdout_data['frequency'],
holdout_data['recency'], holdout_data['T'])

# Initialize and fit the Gamma-Gamma model
ggf = GammaGammaFitter(penalizer_coef=0.0)
ggf.fit(summary['frequency'], summary['monetary_mean'])

# Calculate conditional expected average profit
summary['predicted_monetary_value'] =
ggf.conditional_expected_average_profit(summary['frequency'],
summary['monetary_mean'])

# Calculate CLV for each customer
summary['CLV'] = summary['predicted_purchases'] *
summary['predicted_monetary_value']
```

```
# Sort customers by CLV to identify high-value customers
sorted_summary = summary.sort_values(by='CLV', ascending=False)

# Print the top N customers by CLV
N = 10
top_customers = sorted_summary.head(N)
print("Top {} Customers by CLV:".format(N))
print(top_customers)

# Visualize the distribution of CLV
plt.figure(figsize=(10, 5))
plt.hist(summary['CLV'], bins=30, edgecolor='k')
plt.title('CLV Distribution')
plt.xlabel('Customer Lifetime Value')
plt.ylabel('Number of Customers')
plt.show()

# Export the results to a CSV file or any other desired format
top_customers.to_csv('top_customers_clv.csv', index=False)
```

The code loads your transaction data and preprocesses it to calculate frequency, monetary mean, and monetary sum for each customer. It splits the data into calibration and holdout periods for model training and validation.

The BG/NBD model is trained using the calibration data to predict future purchases in the holdout period. The Gamma-Gamma model is trained using summary statistics to estimate conditional expected average profit. CLV is calculated for each customer based on these predictions. The code visualizes the CLV distribution using a histogram. The top N customers by CLV are printed and can be exported to a CSV file.