Advanced Market Segmentation Using Deep Clustering

Phase 3: Model Training and Evaluation

3.1 Overview of Model Training and Evaluation

This phase centers on model development for deep clustering and market segmentation. We select appropriate algorithms, train them using our pre-processed data, and rigorously evaluate their performance. Algorithm selection considers the specific requirements of these tasks. Model performance is optimized through hyperparameter tuning, and we utilize a range of metrics to assess predictive capabilities. Cross-validation ensures model generalizability to unseen data.

3.2 Choosing Suitable Algorithms

For the **Advanced Market Segmentation using Deep Clustering** project, the key algorithms are:

1. **Autoencoder (for feature extraction and dimensionality reduction)** – This deep learning model is used to encode customer data into a lower-dimensional latent space.
2. **K-Means Clustering (for customer segmentation)** – After dimensionality reduction, K-Means clustering is applied to group customers into distinct segments.

Source code :

```
# Import necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from sklearn.decomposition import PCA

# Define Autoencoder
input_dim = scaled_features.shape[1]

input_layer = Input(shape=(input_dim,))
encoded = Dense(16, activation='relu')(input_layer)
encoded = Dense(8, activation='relu')(encoded)
encoded = Dense(3, activation='relu')(encoded)  # Reduce to 3D for visualization

decoded = Dense(8, activation='relu')(encoded)
decoded = Dense(16, activation='relu')(decoded)
decoded = Dense(input_dim, activation='linear')(decoded)

# Compile the Autoencoder
autoencoder = Model(input_layer, decoded)
```

```
autoencoder.compile(optimizer='adam', loss='mse')

# Train the Autoencoder
autoencoder.fit(scaled_features, scaled_features, epochs=50, batch_size=32, shuffle=True,
verbose=1)

# Extract the Encoder part
encoder = Model(input_layer, encoded)

# Reduce data to 3D
encoded_features = encoder.predict(scaled_features)
customer_encoded_df = pd.DataFrame(encoded_features, columns=["Feature1", "Feature2",
"Feature3"])
customer_encoded_df.insert(0, "CustomerID", customer_data["CustomerID"])
```

3.3 Model Evaluation Metrics

The performance of the model is evaluated using several metrics that measure clustering quality and the reconstruction accuracy of the autoencoder. These include:

3.2.1 **Silhouette Score** – Measures how similar data points are within their cluster compared to other clusters. A higher score indicates better clustering.

Source code :

```
silhouette_avg = silhouette_score(latent_features, clusters)
print("Silhouette Score:", silhouette_avg)
```

Adjusted Rand Index (ARI) – Measures the similarity between the predicted clusters and ground truth labels, adjusting for chance. ARI values closer to 1 indicate better alignment with true labels.

Source code :

```
from sklearn.metrics import adjusted_rand_score

true_labels = _  # Replace with your actual ground truth labels
ari_score = adjusted_rand_score(true_labels, clusters)
print(f"Adjusted Rand Index: {ari_score:.2f}")
```

Mean Squared Error (MSE) – Measures the difference between the original and reconstructed data, indicating how well the autoencoder captures the data's structure.
Source code :

```
# Predict reconstructed data
reconstructed_data = autoencoder.predict(data_scaled)

# Calculate mean squared error (MSE) between original and reconstructed data
```

```
reconstruction_loss = np.mean(np.square(data_scaled - reconstructed_data))
print(f"Reconstruction Loss: {reconstruction_loss:.4f}")
```

3.4 Cross-Validation

To ensure model generalizability and prevent overfitting, we employ cross-validation. While clustering tasks don't inherently have a validation set, we adapt techniques like K-fold cross-validation. This involves partitioning the data into multiple subsets, training the model on some, and evaluating its performance on the remaining held-out subsets.

Source code:

```
from sklearn.model_selection import KFold
from sklearn.metrics import silhouette_score

# Define KFold cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

silhouette_scores = []

# Perform cross-validation
for train_index, test_index in kf.split(latent_features):
    X_train, X_test = latent_features[train_index], latent_features[test_index]
    y_train, y_test = clusters[train_index], clusters[test_index]

    # Fit KMeans on the training data
    kmeans = KMeans(n_clusters=best_n_clusters, random_state=42)
    kmeans.fit(X_train)

    # Predict clusters on the test set
    clusters_pred = kmeans.predict(X_test)

    # Evaluate clustering quality using Silhouette Score
    score = silhouette_score(X_test, clusters_pred)
    silhouette_scores.append(score)

# Average Silhouette Score across all folds
avg_silhouette_score = np.mean(silhouette_scores)
print(f"Average Silhouette Score from cross-validation: {avg_silhouette_score:.4f}")
```

3.5 Conclusion of Phase 3

In Phase 3, we focus on training a model using an autoencoder for dimensionality reduction, followed by k-means clustering. The k-means algorithm's hyperparameters were optimized using grid search. Performance was evaluated using metrics such as silhouette score, adjusted Rand index, and reconstruction loss. Cross-validation assessed model robustness and generalizability. These evaluation metrics provided insights into both the clustering quality and the autoencoder's effectiveness in representing underlying data patterns.