Phase 3: Model Training and Evaluation

3.1 Overview of Model Training and Evaluation

In this phase, we focus on selecting suitable algorithms, training the models using the processed data, and evaluating their performance. The goal is to refine the market segmentation process by leveraging deep clustering techniques. Hyperparameter tuning ensures optimal model performance, and various evaluation metrics are used to assess predictive capabilities. Cross-validation is employed to enhance generalization to unseen data.

3.2 Choosing Suitable Algorithms

Building upon the preprocessed data from Phase 2, we employ the following key algorithms:

1. **Autoencoder (for feature extraction and dimensionality reduction)** – The autoencoder processes customer data and encodes it into a lower-dimensional latent space, preserving essential information.
2. **K-Means Clustering (for customer segmentation)** – After feature extraction, K-Means clustering is applied to group customers into distinct segments.

```
import pandas as pd
import pickle as pk
from sklearn.feature_extraction.text import TfidfVectorizer
import streamlit as st
```

Train Autoencoder

```
autoencoder = tf.keras.Sequential([
tf.keras.layers.InputLayer(input_shape=(num_features,)),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(256, activation='relu'),
tf.keras.layers.Dense(128, activation='relu'),
tf.keras.layers.Dense(256, activation='relu'),
tf.keras.layers.Dense(512, activation='relu'),
tf.keras.layers.Dense(num_features, activation='sigmoid')
])

autoencoder.compile(optimizer='adam', loss='mean_squared_error')
history = autoencoder.fit(data_train, data_train,
epochs=50,
batch_size=32,
validation_data=(data_val, data_val))
```

```
# Extract latent eatures
latent_features = autoencoder.predict(data_scaled)

# Apply K-Means clustering
kmeans = KMeans(n_clusters=5, random_state=42)
clusters = kmeans.fit_predict(latent_features)

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

3.3 Hyperparameter Tuning

To enhance performance, hyperparameter tuning is conducted for both the autoencoder and K-Means:

- **Autoencoder:** Learning rate, batch size, and layer configurations are optimized using random search or Bayesian optimization.
- **K-Means:** The optimal number of clusters is determined using a grid search approach.

Source Code for Grid Search (K-Means):

```
from sklearn.metrics import silhouette_score
best_score = -1
best_n_clusters = 0

for n_clusters in range(2, 10):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    clusters = kmeans.fit_predict(latent_features)
    score = silhouette_score(latent_features, clusters)
    if score > best_score:
        best_score = score
        best_n_clusters = n_clusters

print(f"Best Number of Clusters: {best_n_clusters}, Best Silhouette Score: {best_score:.4f}")
```

3.4 Model Evaluation Metrics

The model's performance is assessed using multiple clustering quality and reconstruction accuracy metrics:

1. **Silhouette Score -**The Silhouette Score evaluates clustering quality by measuring the similarity of an object to its own cluster versus other clusters, indicating better-defined clusters in contextual language understanding with transformer models.
2. **Adjusted Rand Index (ARI)** – measures the similarity between the clustering results and the ground truth labels
3. **Mean Squared Error (MSE)** – quantifies the average squared difference between the original and reconstructed data,

Source Code:

```
from sklearn.linear_model import LogisticRegression


# Silhouette Score
silhouette_avg = silhouette_score(data_val, clusters)

22print(f"Silhouette Score: {silhouette_avg:.4f}")


# Adjusted Rand Index
ari_score = adjusted_rand_score(true_labels_val, clusters)

26print(f"Adjusted Rand Index (ARI): {ari_score:.4f}")


# Mean Squared Error
mse = np.mean(np.square(data_val - reconstructed_data))

40print(f"Mean Squared Error (MSE): {mse:.4f}")
```

3.5 Cross-Validation

To ensure model generalizability, **K-Fold Cross-Validation** is implemented:

Source Code:

```
from sklearn.model_selection import KFold


kf = KFold(n_splits=5, shufle=True, random_state=42)

silhouette_scores = []


for train_idx, test_idx in kf.split(latent_features):

    X_train, X_test = latent_features[train_idx], latent_features[test_idx]
```

```
    kmeans = KMeans(n_clusters=best_n_clusters, random_state=42)
    kmeans.fit(X_train)
    clusters_pred = kmeans.predict(X_test)
    silhouette_scores.append(silhouette_score(X_test, clusters_pred))

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

3.5 Cross-Validation

To ensure model generalizability, **K-Fold Cross-Validation** is implemented:

Source Code:

```
from sklearn.model_selection import KFold

kf = KFold(n_splits=5, shufle=True, random_state=42)
silhouette_scores = []

for train_idx, test_idx in kf.split(latent_features):
    X_train, X_test = latent_features[train_idx], latent_features[test_idx]
    kmeans = KMeans(n_clusters=best_n_clusters, random_state=42)
    kmeans.fit(X_train)
    clusters_pred = kmeans.predict(X_test)
    silhouette_scores.append(silhouette_score(X_test, clusters_pred))

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

Conclusion of Phase 3

In this phase, the autoencoder-based dimensionality reduction and K-Means clustering approach were successfully implemented for advanced market segmentation.

- **Hyperparameter tuning** was conducted to optimize model performance.
- **Evaluation metrics** such as silhouette score, ARI, and reconstruction loss provided insights into clustering quality.
- **Cross-validation** ensured that the model generalizes well to new data.

The refined segmentation approach enhances targeted marketing strategies, improving business decision-making and customer engagement.