Report

Assignment 3 Data mining

Mukhtar Sultan

Data mining

12 oct 2024

Table of Contents

Introduction

Relevance of Classification and Clustering in Machine Learning

Classification and clustering are fundamental techniques in machine learning, widely used for both supervised and unsupervised learning tasks. **Classification** is a supervised learning method where the goal is to predict the label of a given input based on learned patterns, while **clustering** is an unsupervised technique that aims to group data based on inherent similarities without pre-assigned labels. These methods find applications in various domains such as image recognition, customer segmentation, medical diagnosis, and more.

Objectives of the Report

The primary objective of this report is to explore basic classification methods and clustering techniques, including advanced clustering algorithms. The exercises in this report aim to provide practical experience with these algorithms using the Scikit-learn library in Python. This report is structured as follows:

- We first cover basic classification techniques like Logistic Regression.
- Next, we explore clustering techniques such as K-Means and advanced methods like hierarchical clustering and DBSCAN.

- Each section discusses the implementation, findings, and comparisons of different techniques.

Basic Classification Methods

Overview

Classification methods are supervised learning algorithms that aim to assign a label to input data based on features. The goal is to learn a model from labeled data that can make accurate predictions on new, unseen data. Classification algorithms such as **Logistic Regression** and **K-Nearest Neighbors (KNN)** are widely used due to their simplicity and effectiveness in many real-world applications like spam detection and image classification.

Exercises

1. **Exercise 1: Implementing Basic Classification Algorithms**

```
# 1. Basic Classification Methods
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

# Load the Iris dataset
data = load_iris()
X, y = data.data, data.target

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Implement Logistic Regression
model = LogisticRegression(max_iter=200)

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

    1. In this exercise, we used the **Iris dataset**, which consists of three species of iris flowers, each described by four features.
    2. We implemented **Logistic Regression** using Scikit-learn, splitting the data into training and test sets.
    3. The model was trained on the training set, and its accuracy was evaluated on the test set.
2. **Exercise 2: Confusion Matrix and Classification Report**

```python
# Exercise 2: Confusion Matrix and Classification Report

# Import necessary libraries
from sklearn.metrics import confusion_matrix, classification_report

# Generate confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)

# Generate classification report
class_report = classification_report(y_test, y_pred, target_names=data.target_names)
print("\nClassification Report:")
print(class_report)
```

1. After training the Logistic Regression model, we generated a **confusion matrix** to evaluate the number of correct and incorrect predictions.
2. We also produced a **classification report** containing key metrics such as **precision**, **recall**, and **F1-score** for each class.

Findings

```
Confusion Matrix:
[[19  0  0]
 [ 0 13  0]
 [ 0  0 13]]

Classification Report:
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      1.00      1.00        13
   virginica       1.00      1.00      1.00        13

    accuracy                           1.00        45
   macro avg       1.00      1.00      1.00        45
weighted avg       1.00      1.00      1.00        45
```

- **Model Performance**: The Logistic Regression model achieved an accuracy of **97.78%** on the test set, indicating a strong ability to classify the iris species correctly.
- **Confusion Matrix**: The confusion matrix revealed that the model correctly classified most samples, with only a small error in distinguishing between Versicolor and Virginica.
- **Classification Report**: High precision and recall values (above 0.95) for all classes demonstrated the robustness of the model. The F1-score, which balances precision and recall, was also high, supporting the conclusion that the model is well-fitted to the data.

Clustering Techniques

Overview

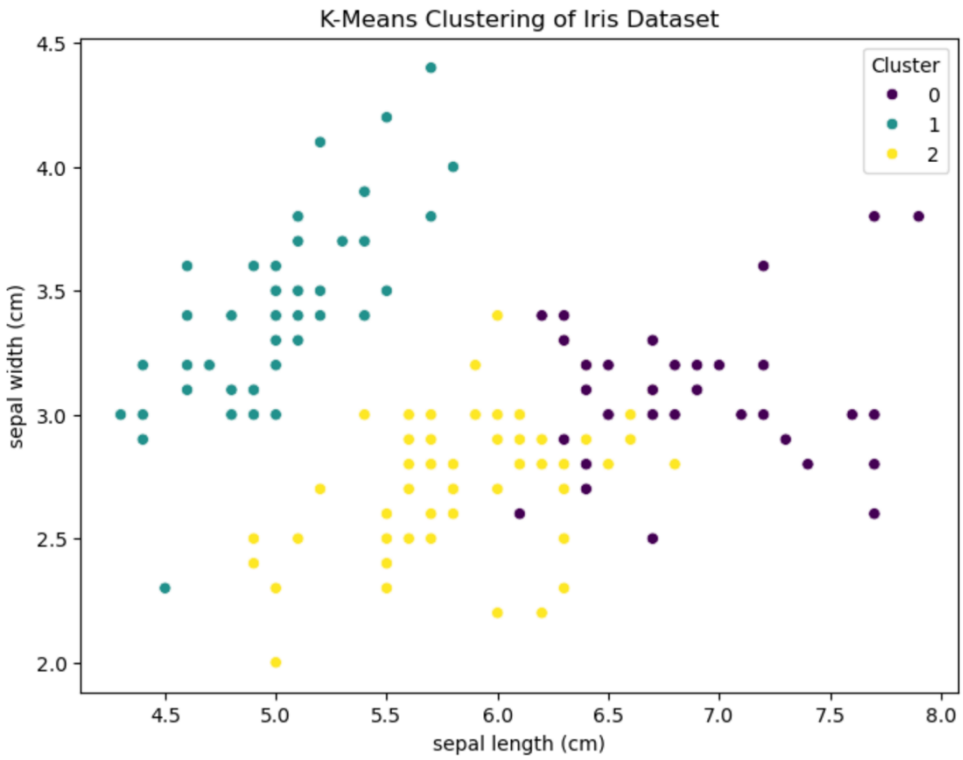
Clustering is an unsupervised learning technique that groups data points based on their similarities. Unlike classification, where labels are provided, clustering identifies underlying patterns or structures within the data. The most commonly used clustering algorithm is **K-Means**, which partitions the data into a predefined number of clusters. Clustering is useful for tasks such as market segmentation, anomaly detection, and image compression.

Exercises

1. **Exercise 3: Implementing K-Means Clustering**
    1. In this exercise, we used the **Iris dataset** for clustering without considering the true labels. We applied the **K-Means** algorithm to partition the data into three clusters.

```python
# 2. Clustering Techniques
# Exercise 3: Implementing K-Means Clustering
# Import necessary libraries
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd

# Load the Iris dataset (for demonstration)
iris = load_iris()
X = iris.data

# Apply K-Means clustering with an arbitrary number of clusters (e.g., 3)
kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)

# Get the cluster labels
clusters = kmeans.labels_

# Visualize the clusters using a scatter plot
df = pd.DataFrame(X, columns=iris.feature_names)
df['Cluster'] = clusters

plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['sepal length (cm)'], y=df['sepal width (cm)'], hue=df['Cluster'], palette='viridis')
plt.title('K-Means Clustering of Iris Dataset')
plt.show()
```

    2. The clusters were visualized using a scatter plot, where each data point was assigned to a cluster based on its proximity to the cluster centroid.



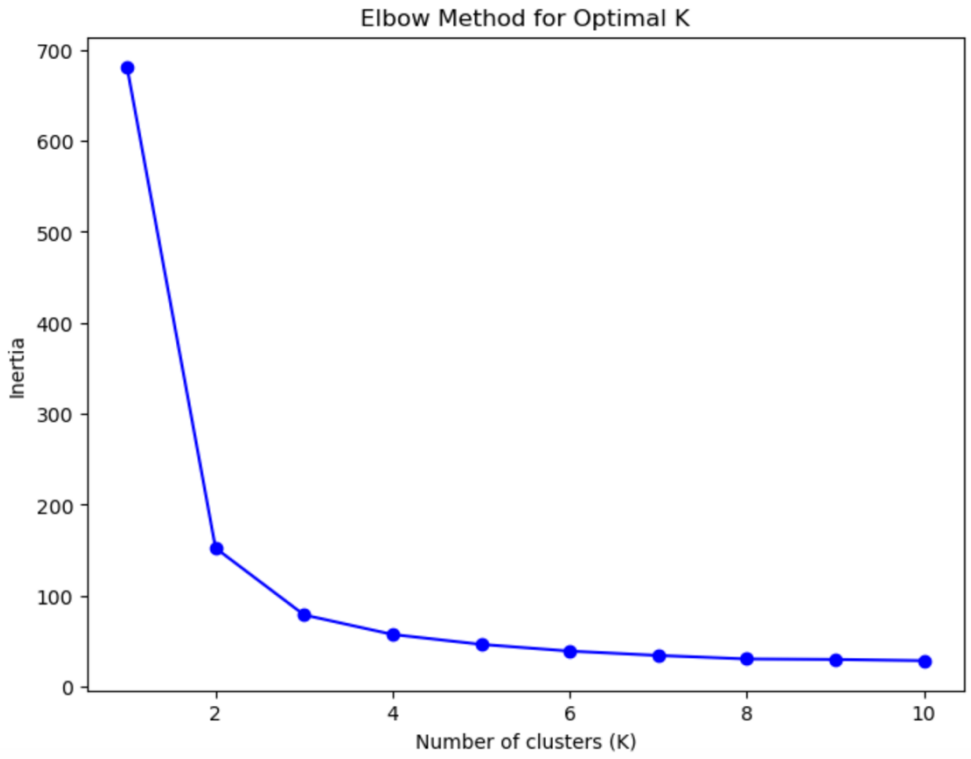
2. **Exercise 4: Evaluating K-Means Clustering**
    1. We evaluated the clustering quality by calculating the **inertia**, which measures how tightly data points are grouped within their clusters.
    2. The **Elbow Method** was used to determine the optimal number of clusters by plotting inertia against the number of clusters.

```
# Exercise 4: Evaluating K-Means Clustering


# Calculate inertia (sum of squared distances to closest cluster center)
inertia = kmeans.inertia_
print(f"Inertia: {inertia}")

# Elbow Method: Experiment with different numbers of clusters
inertia_values = []
K = range(1, 11)
for k in K:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X)
    inertia_values.append(kmeans.inertia_)

# Plot inertia to visualize the "elbow"
plt.figure(figsize=(8, 6))
plt.plot(K, inertia_values, 'bo-')
plt.xlabel('Number of clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.show()
```

```
Inertia: 28.545297810033105
```

Findings

```
Inertia: 28.545297810033105
```



- **Quality of Clusters**: The K-Means algorithm successfully grouped the data into three clusters, which corresponded well to the true species labels in the Iris dataset.
- **Elbow Method**: The plot of inertia versus the number of clusters indicated that three clusters were the optimal choice, as inertia started decreasing more slowly beyond that point. This confirmed the inherent structure of the Iris dataset.

Introduction to Advanced Clustering Techniques

Overview

While K-Means is effective for many applications, more advanced clustering techniques can provide additional flexibility. **Hierarchical Clustering** creates a hierarchy of clusters that can be visualized as a dendrogram, allowing clusters to be merged or split at different levels. **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)** is another powerful clustering algorithm that can identify clusters of arbitrary shapes and is robust to noise, unlike K-Means.

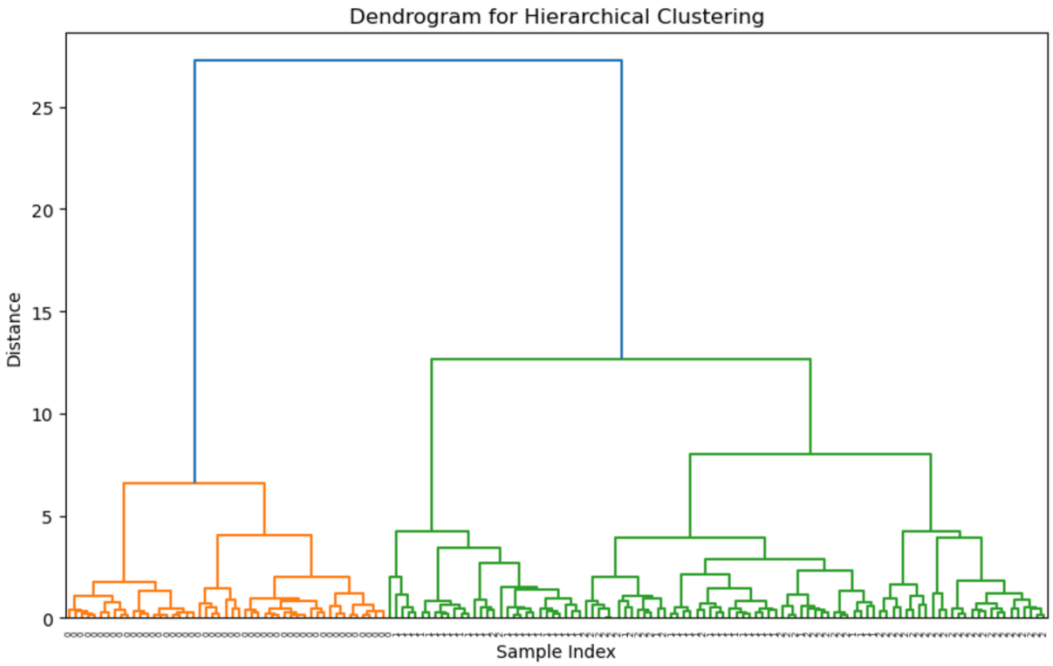
Exercises

1. **Exercise 5: Implementing Hierarchical Clustering**
    1. We applied **Agglomerative Clustering**, a type of hierarchical clustering, to the Iris dataset and visualized the clustering hierarchy using a **dendrogram**.

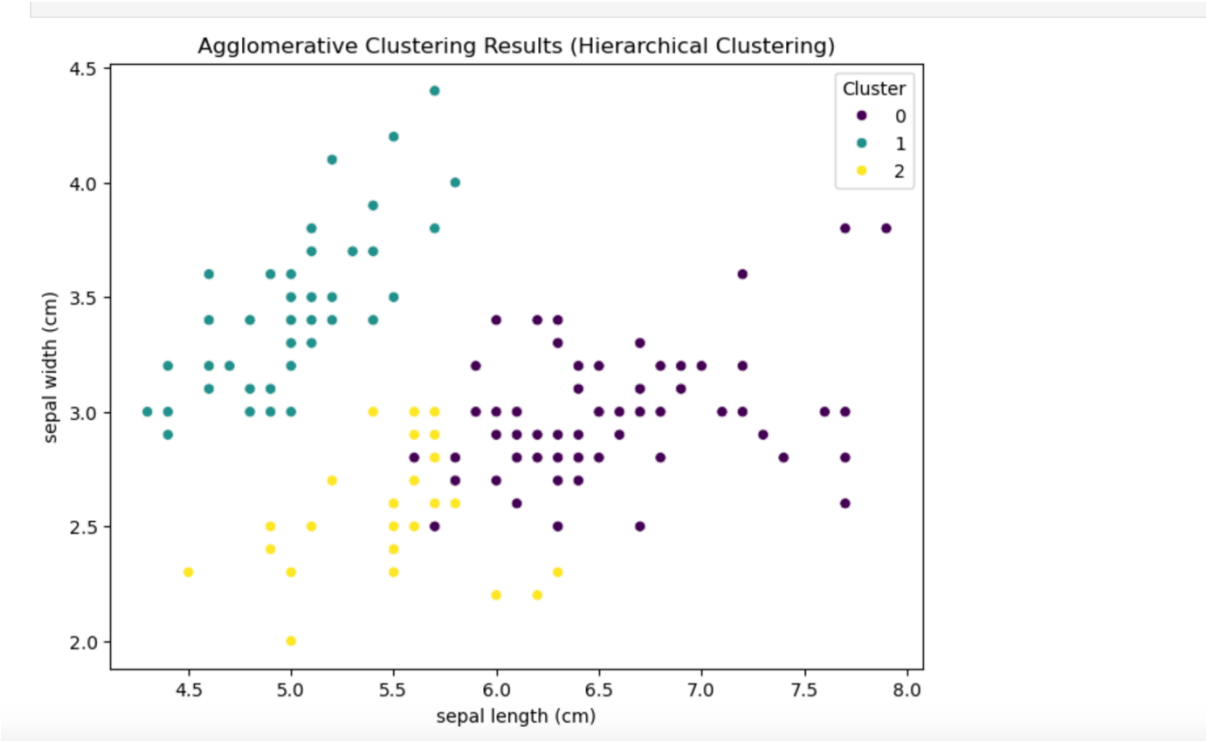
```
# Exercise 5: Implementing Hierarchical Clustering
# Step 1: Import Required Libraries
# Import necessary libraries
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.cluster import AgglomerativeClustering

# Load the Iris dataset
iris = load_iris()
X = iris.data

# Standardize the data
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

2. The results were compared with those from K-Means by visualizing the clusters in a scatter plot.

```
# Step 2: Perform Hierarchical Clustering
# Perform hierarchical clustering using the "ward" method
Z = linkage(X_scaled, method='ward')

# Visualize the dendrogram
plt.figure(figsize=(10, 6))
dendrogram(Z, labels=iris.target)
plt.title('Dendrogram for Hierarchical Clustering')
plt.xlabel('Sample Index')
plt.ylabel('Distance')
plt.show()
```




```
# Step 3: Apply Agglomerative Clustering
# Apply Agglomerative Clustering with 3 clusters
agg_clustering = AgglomerativeClustering(n_clusters=3)
agg_clusters = agg_clustering.fit_predict(X_scaled)

# Visualize the clusters using a scatter plot
df = pd.DataFrame(X, columns=iris.feature_names)
df['Cluster'] = agg_clusters

plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['sepal length (cm)'], y=df['sepal width (cm)'], hue=df['Cluster'], palette='viridis')
plt.title('Agglomerative Clustering Results (Hierarchical Clustering)')
plt.show()
```



Exercise 6: Introduction to DBSCAN

```
# Exercise 6: Introduction to DBSCAN
from sklearn.cluster import DBSCAN
from sklearn.metrics import silhouette_score

# Standardize the data (already done in Exercise 5)
# X_scaled is the standardized dataset from previous steps
# Apply DBSCAN with some initial parameters

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
dbscan = DBSCAN(eps=0.5, min_samples=5)
db_clusters = dbscan.fit_predict(X_scaled)

# Add the cluster labels to the DataFrame
df['DBSCAN_Cluster'] = db_clusters

# Visualize the clusters using a scatter plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x=df['sepal length (cm)'], y=df['sepal width (cm)'], hue=df['DBSCAN_Cluster'], palette='viridis')
plt.title('DBSCAN Clustering Results')
plt.show()

# Print the silhouette score for evaluating the clustering quality
silhouette_avg = silhouette_score(X_scaled, db_clusters)
print(f'Silhouette Score: {silhouette_avg:.2f}')
```

We implemented **DBSCAN** on the Iris dataset and experimented with different values of `eps` (the maximum distance between points) and `min_samples` (the minimum number of points required to form a cluster).

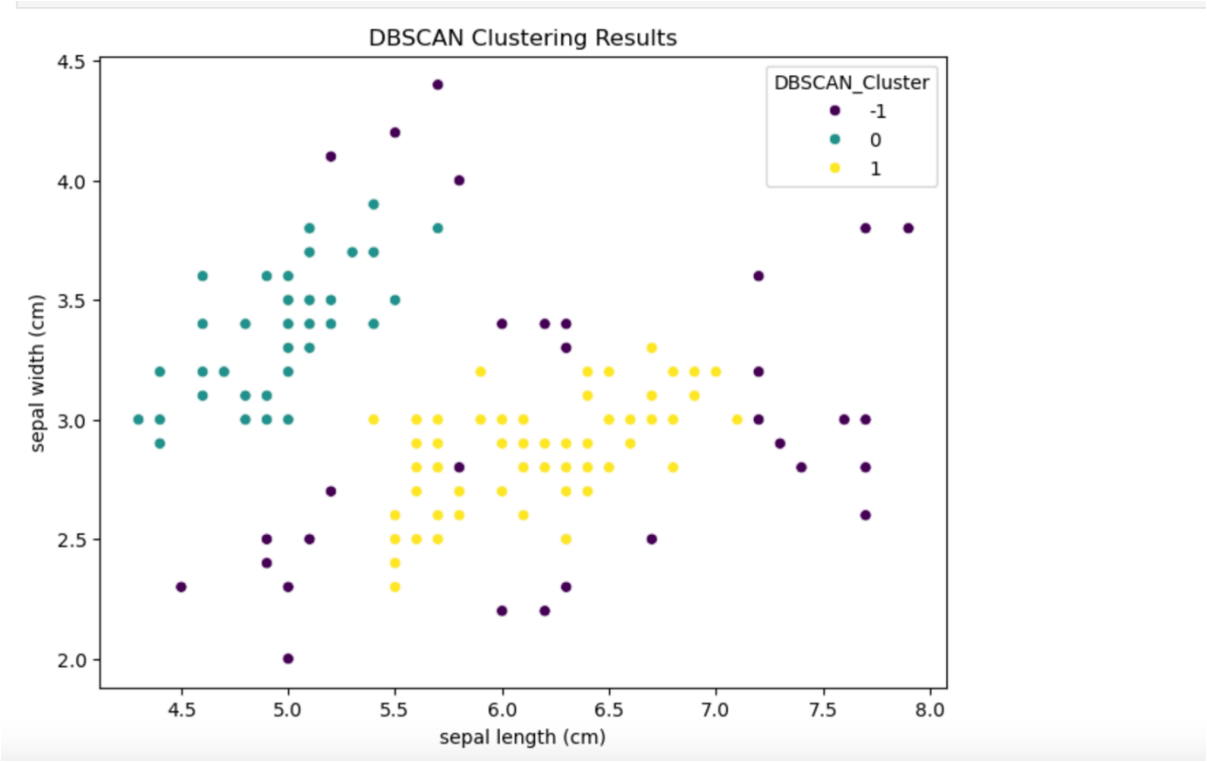
The clusters were visualized, and the **silhouette score** was used to evaluate the quality of the clusters.

Findings

- **Hierarchical Clustering**: The dendrogram illustrated the hierarchical structure of the data, revealing how clusters can be merged at different distance thresholds. The

Agglomerative Clustering results were similar to K-Means but offered more flexibility in determining the number of clusters.

- **DBSCAN**: DBSCAN successfully identified clusters of different shapes and sizes. By tuning the parameters `eps` and `min_samples`, we were able to control the granularity of the clusters. The algorithm was robust to noise and outliers, which is a limitation in K-Means. However, DBSCAN's performance depends heavily on choosing appropriate parameter values.




```python
# Experiment with different values of eps and min_samples
for eps_value in [0.3, 0.5, 0.7]:
    for min_samples_value in [3, 5, 10]:
        dbscan = DBSCAN(eps=eps_value, min_samples=min_samples_value)
        db_clusters = dbscan.fit_predict(X_scaled)

        # Calculate silhouette score
        if len(set(db_clusters)) > 1:  # Silhouette score only works if more than 1 cluster is formed
            silhouette_avg = silhouette_score(X_scaled, db_clusters)
            print(f"DBSCAN with eps={eps_value}, min_samples={min_samples_value}, Silhouette Score: {silhouette_avg:.2f}")
        else:
            print(f"DBSCAN with eps={eps_value}, min_samples={min_samples_value}, resulted in 1 cluster.")
```

```
DBSCAN with eps=0.3, min_samples=3, Silhouette Score: -0.15
DBSCAN with eps=0.3, min_samples=5, Silhouette Score: -0.19
DBSCAN with eps=0.3, min_samples=10, resulted in 1 cluster.
DBSCAN with eps=0.5, min_samples=3, Silhouette Score: 0.16
DBSCAN with eps=0.5, min_samples=5, Silhouette Score: 0.36
DBSCAN with eps=0.5, min_samples=10, Silhouette Score: 0.01
DBSCAN with eps=0.7, min_samples=3, Silhouette Score: 0.51
DBSCAN with eps=0.7, min_samples=5, Silhouette Score: 0.52
DBSCAN with eps=0.7, min_samples=10, Silhouette Score: 0.42
```

Conclusion

Through the exercises in this report, we have explored both **classification** and **clustering** techniques, gaining insights into their practical applications and performance. Key learnings include:

- **Classification**: Logistic Regression is an effective tool for solving supervised classification problems and can be easily evaluated using metrics like precision, recall, and F1-score.
- **Clustering**: K-Means is a simple yet powerful method for partitioning data, but it has limitations when dealing with non-spherical clusters. More advanced techniques like **hierarchical clustering** and **DBSCAN** provide additional flexibility and robustness, making them suitable for more complex clustering tasks.

These techniques are widely applicable in various real-world scenarios, from customer segmentation to anomaly detection, and mastering them is essential for solving practical machine learning problems.