Practical Assessment 2: Clustering random data.

Introduction

In this assignment, you'll explore artificial datasets for machine learning tasks and clustering methods. It is highly recommended that you use Google Colab or Jupyter Notebook to complete this assignment. Jupyter will require more installation steps if you are not already familiar with the environment.

Learning Objectives

- Understand the concept of artificial data generation.
- Become familiar with using Google Colab/Jupyter Notebook for data manipulation.
- Gain practical experience with libraries like NumPy and scikit-learn.
- Explore data visualization with Matplotlib.

Steps

1. **Import Necessary Libraries:**

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

2. **Define Dataset Parameters (Hint: Experiment with different values!):**

```
# Number of observations in each class
N1 = 200
N2 = 200

# Distance between class centers
d = 1.0

# Dispersion (standard deviation)
sigma1 = 0.5
sigma2 = 0.5
```

You can adjust `N1`, `N2`, `d`, `sigma1`, and `sigma2` to observe how these parameters affect the generated data and the resulting clusters. For example, increasing `d` will separate the clusters further in the feature space, while decreasing `sigma1` and `sigma2` will tighten the clusters around their centres.

3. **Generate Class Center Locations:**

```
# Class center 1 coordinates
x1_center = d / 2 * np.array([-1, 1])

# Class center 2 coordinates
x2_center = d / 2 * np.array([1, 1])
```

4. **Create Feature Matrix:**

```
# Combine class center locations
X_T = np.vstack((x1_center, x2_center))

# Add noise (random variations) to each feature
noise1 = np.random.normal(scale=sigma1, size=(N1, 2))
noise2 = np.random.normal(scale=sigma2, size=(N2, 2))

# Add noise to class centres to create data points
X1 = X_T[0, :] + noise1
X2 = X_T[1, :] + noise2

# Combine data points from both classes
X = np.concatenate((X1, X2), axis=0)
```

5. **Generate Class Labels:**

```
# Create class labels (0 for class 1, 1 for class 2)
y = np.array([0] * N1 + [1] * N2)
```

6. **Visualize and Cluster the data:**

This code snippet plots the generated data points, allowing you to visually inspect the separation between the two classes. Play around with the parameters in step 2 and observe how the plot changes.

```
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title("Generated Dataset")
plt.show()
```

Use scikit-learn's `KMeans` algorithm to cluster the data and compare the results with the actual class labels.

```
# Define the number of clusters (should be 2 for this dataset)
n_clusters = 2

# Create a KMeans object with the specified number of clusters
kmeans = KMeans(n_clusters=n_clusters)

# Fit the KMeans model to the data
kmeans.fit(X)

# Get the cluster labels assigned by KMeans
kmeans_labels = kmeans.labels_
```

After obtaining the KMeans labels, we align them with the true class labels by assigning each cluster label to the majority class in that cluster. Then we calculate the accuracy by comparing the aligned labels with the true class labels.

```
# Align KMeans labels with true class labels
aligned_labels = np.zeros_like(kmeans_labels)
for cluster in range(n_clusters):
    mask = (kmeans_labels == cluster)
    aligned_labels[mask] = np.argmax(np.bincount(y[mask]))
# Calculate accuracy
accuracy = np.sum(aligned_labels == y) / len(y)
print("Accuracy:", accuracy)
```

Visualise the clustered data and evaluate your results. (**! No more hints !)**

Report:
Finally, write a report that will be submitted as a .pdf containing screenshots of the written code as well as the output. Add personal identifiers to each part of your code. Please include full desktop screenshots and remember that any form of plagiarism will be strictly punished.