Machine Learning I: Fractal 2

Programming Assignment 1

Submitted by

Debonil Ghosh (M21AIE225)

Question 1.

Implement the k-means and spectral clustering algorithms for clustering the points given in the datasets: http://cs.joensuu.fi/sipu/datasets/jain.txt. Plot the obtained results. In order to evaluate the performance of these algorithms, find the percentage of points for which the estimated cluster label is correct. Report the accuracy of both the algorithm. The ground truth clustering is given as the third column of the given text file. [15 Marks]

Solution:

Kmeans Algorithm:

Kmeans algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster**.**

The way kmeans algorithm works is as follows:

1. Specify number of clusters K.
2. Initialize centroids by first shuffling the dataset and then randomly selecting K data points for the centroids without replacement.
3. Keep iterating until there is no change to the centroids. i.e assignment of data points to clusters isn't changing.

➢ Compute the sum of the squared distance between data points and all centroids.
➢ Assign each data point to the closest cluster (centroid).
➢ Compute the centroids for the clusters by taking the average of the all data points that belong to each cluster.

The approach kmeans follows to solve the problem is called Expectation-Maximization. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.

Spectral Clustering Algorithm:

Clustering is one of the most widely used techniques for exploratory data analysis. Its goal is to divide the data points into several groups such that points in the same group are similar and points in different groups are dissimilar to each other. It can be solved efficiently by standard linear algebra, and very often outperforms traditional algorithms such as the k-means algorithm.

To perform a spectral clustering, we need 3 main steps:

1. Create a similarity graph between our N objects to cluster.
2. Compute the first k eigenvectors of its Laplacian matrix to define a feature vector for each object.
3. Run k-means on these features to separate objects into k classes.

Implementation:

- KMeans Cluster From Scratch

In [ ]:

```
import numpy.linalg as la


class KMeans:
    def __init__(self, n_clusters=2, tollerance=0.001, max_iter=10):
        self.k = n_clusters
        self.tollerance = tollerance
        self.max_iter = max_iter

    def fit_predict(self, data):

        self.centroids = {}

        for i in range(self.k):
            self.centroids[i] = data[i]

        for i in range(self.max_iter):
            self.classifications = {}

            for i in range(self.k):
                self.classifications[i] = []

            for featureIndex, featureset in enumerate(data):
                distances = [la.norm(featureset-self.centroids[centroid])
                             for centroid in self.centroids]
                classification = distances.index(min(distances))
                self.classifications[classification].append(featureIndex)

            prev_centroids = dict(self.centroids)

            for classification in self.classifications:
                self.centroids[classification] = np.average(
                    data[self.classifications[classification]], axis=0)

            optimized = True

            for c in self.centroids:
                centroid_shift = np.sum(
                    (self.centroids[c]-
prev_centroids[c])/prev_centroids[c]*100.0)
                if centroid_shift > self.tollerance:
                    optimized = False

            if optimized:
                break

        predictions = np.empty([len(data)])
        for classification in self.classifications:
            predictions[self.classifications[classification]] =
classification
        return predictions
```

- Spectral Clustering from scratch

In [ ]:

```
def spectral_clustering(X, k):

    # create weighted adjacency matrix
    W = nearest_neighbor_graph(X)

    # create unnormalized graph Laplacian matrix
    L = compute_laplacian(W)

    # create projection matrix with first k eigenvectors of L
    E = get_eigvecs(L, k)

    # return clusters using k-means on rows of projection matrix
    f =  KMeans(n_clusters=k).fit_predict(E)# k_means_clustering(E,k)
    return np.ndarray.tolist(f)
```

- Data load

In [ ]:

```
data = pd.read_csv('data-ques-1/jain.txt', sep='\t', names=['X', 'Y',
'Class'])
x = data.drop('Class', axis=1)
data['Class'] = data['Class']-1
data.head()
```

Out[ ]:

	X	Y	Class
0	0.85	17.45	1
1	0.75	15.60	1
2	3.30	15.45	1
3	5.25	14.20	1
4	4.90	15.65	1

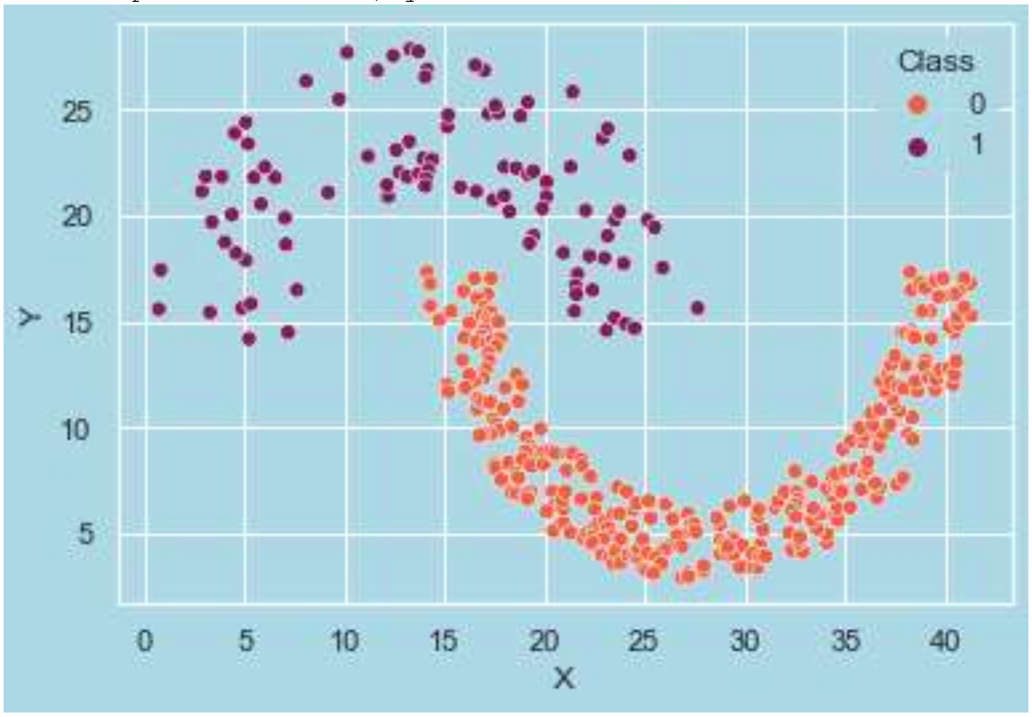
- Visualize Acutal Data

In [ ]:

```
sns.scatterplot(data=data, x='X', y='Y', hue='Class',  palette='rocket_r')
```

Out[ ]:

```
<AxesSubplot:xlabel='X', ylabel='Y'>
```



Applied KMeans Clustering

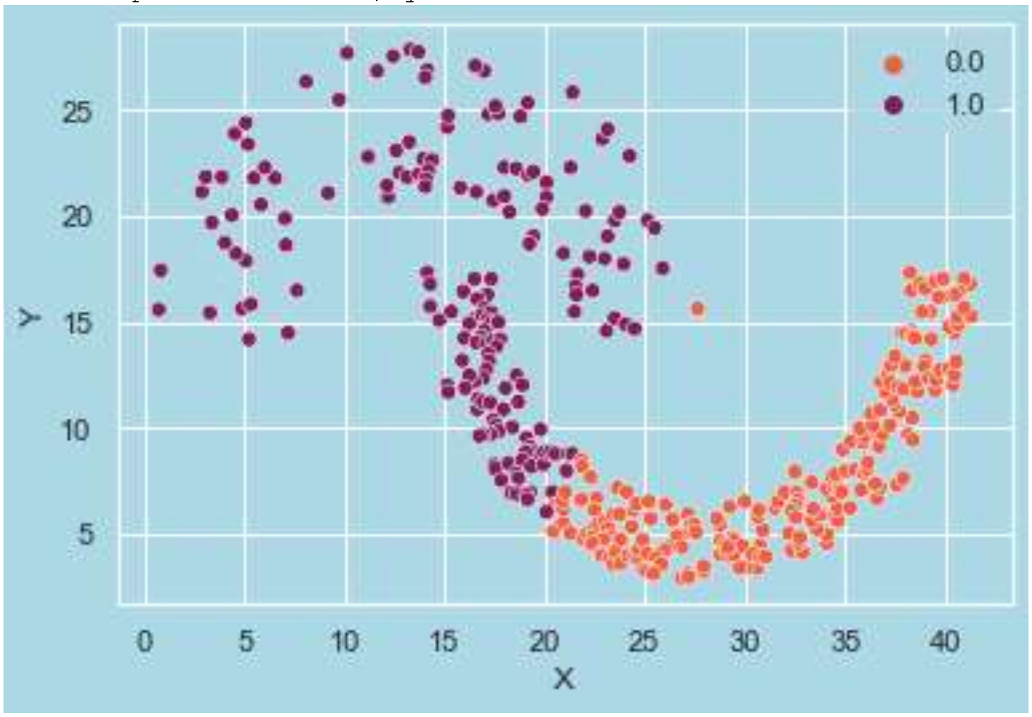
In [ ]:

```
#from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2)
pred =kmeans.fit_predict(x.values)
pred=(pred+1)%2
```

```
#pred =k_means_clustering(x.values,2)

sns.scatterplot(data=data, x='X', y='Y', hue=pred,  palette='rocket_r')
```

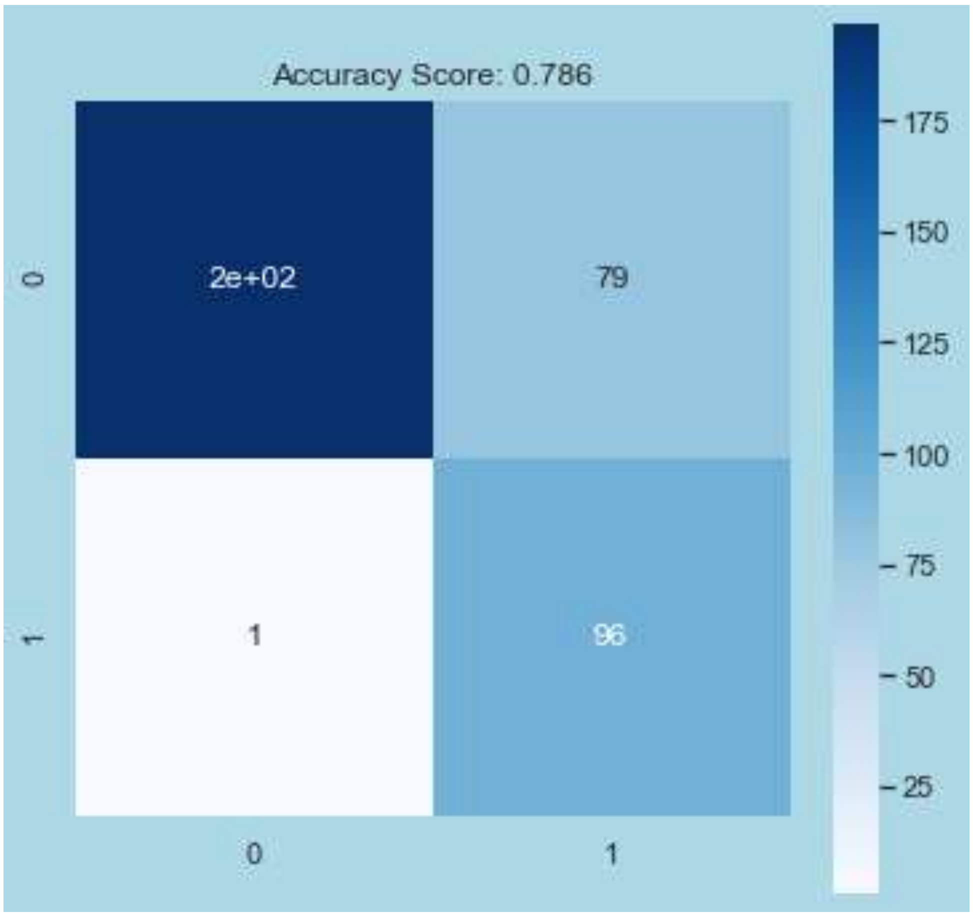
Out[ ]:

```
<AxesSubplot:xlabel='X', ylabel='Y'>
```



In [ ]:

```
confusionMatrixAndAccuracyReport(data['Class'], pred)
```



```
Overall Accuracy Score: 0.786
Classwise Accuracy Score: [0.78552279 0.78552279]
```

Applied SpectralClustering

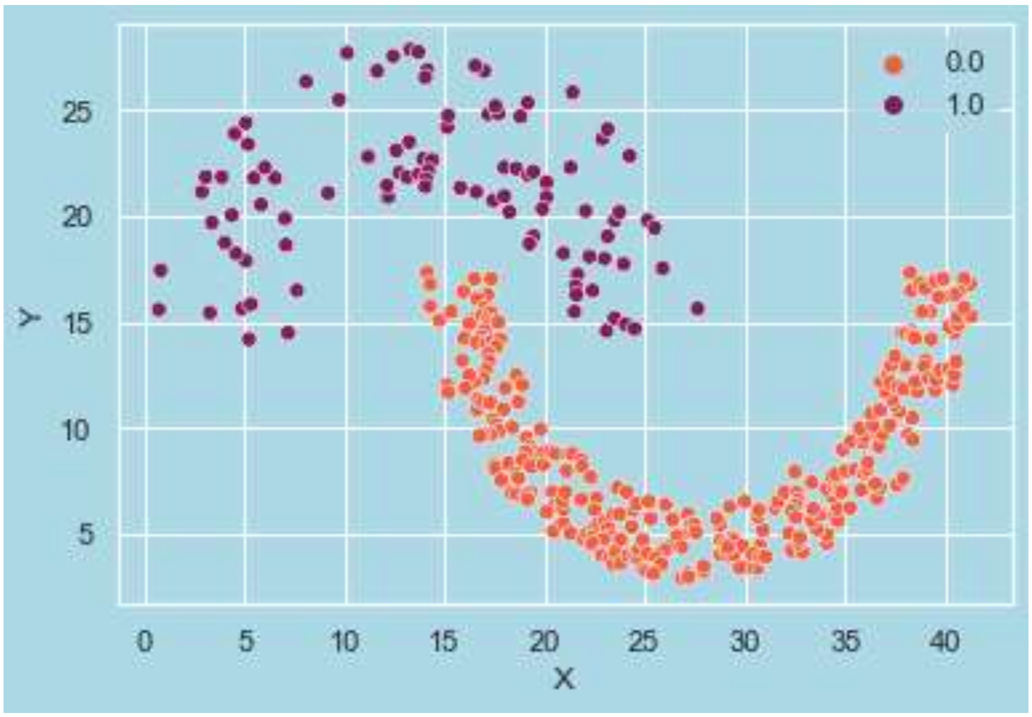
In [ ]:

```
#from sklearn.cluster import SpectralClustering

#model = SpectralClustering(n_clusters=2)

pred = spectral_clustering(x.values, 2)
print(np.unique(pred))
sns.scatterplot(data=data, x='X', y='Y', hue=pred,  palette='rocket_r')
```

```
[0. 1.]
```

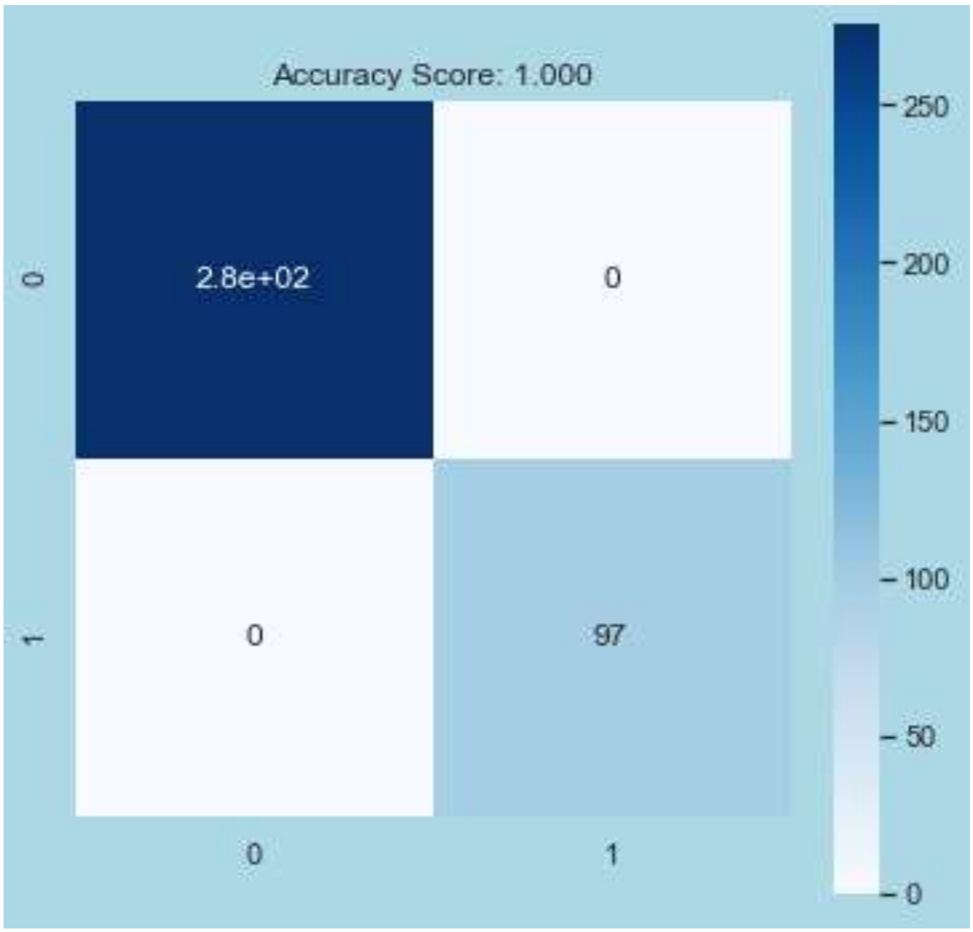
Out[ ]:

```
<AxesSubplot:xlabel='X', ylabel='Y'>
```



In [ ]:

```
confusionMatrixAndAccuracyReport(data['Class'], pred)
```



```
Overall Accuracy Score: 1.000
Classwise Accuracy Score: [1. 1.]
```

Question 2.

Implement the Principal Component Analysis algorithm for reducing the dimensionality of the points given in the datasets: https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris. data. Each point of this dataset is a 4-dimensional vector (d = 4) given in the first column of the datafile. Reduce the dimensionality to 2 (k = 2). This dataset contains 3 clusters. Ground-truth cluster IDs are given as the fifth column of the data file. In order to evaluate the performance of the PCA algorithm, perform clustering (in 3 clusters) before and after dimensionality reduction using the Spectral Clustering algorithm and then find the percentage of points for which the estimated cluster label is correct. Report the accuracy of the Spectral Clustering algorithm before and after the dimensionality reduction. Report the reconstruction error for k = 1, 2, 3. [15 Marks]

Solution:

Principal Component Analysis:

Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components.

1. Normalize the data

Standardize the data before performing PCA. This will ensure that each feature has a mean = 0 and variance = 1.

$$Z = \frac{x - \mu}{\sigma}$$

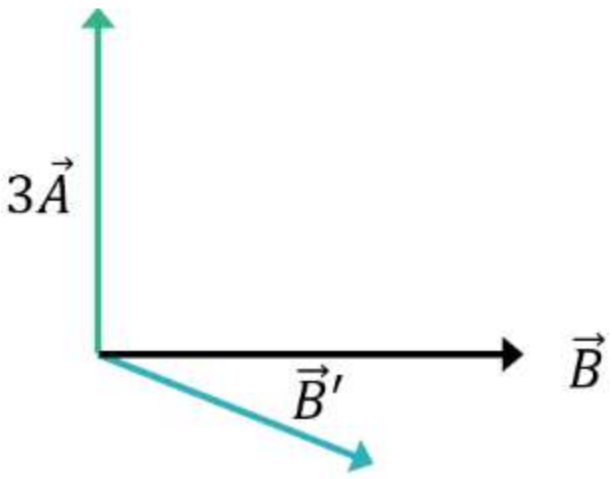
2. Build the covariance matrix

Construct a square matrix to express the correlation between two or more features in a multidimensional dataset.

$$\begin{bmatrix} \mathrm{Var}(x) & \mathrm{Cov}(x,y) & \cdots & \mathrm{Cov}(x,m) \\ \mathrm{Cov}(x,y) & \ddots & & \vdots \\ \mathrm{Cov}(z,x) & & \ddots & \vdots \\ \vdots & & & \\ \mathrm{Cov}(n,x) & \cdots & \cdots & \mathrm{Var}(n) \end{bmatrix}$$

3. Find the Eigenvectors and Eigenvalues

Calculate the eigenvectors/unit vectors and eigenvalues. Eigenvalues are scalars by which we multiply the eigenvector of the covariance matrix.



4. **Sort the eigenvectors in highest to lowest order and select the number of principal components.**

Implementation:

- PCA Decomposition from scratch

In [ ]:

```
import numpy.linalg as la


class PCA:
    def __init__(self, n_components=2):
        self.n_components = n_components

    def fit_transform(self, X_data):
       # centering data
        self.X_mean = np.mean(X_data, axis=0)
        x_centered = X_data - self.X_mean
        # calculating covariance matrix
        x_cov = np.cov(x_centered.T)
        # eigendecomposition
        eigenvals, eigenvecs = la.eig(x_cov)
        # sorting
        i = np.argsort(eigenvals)[::-1]
        self.eigenvecs = eigenvecs[:, i]
        eigenvals = eigenvals[i]
        # retaining the eigenvectors for first n PCs
        self.X_evecs_n = self.eigenvecs[:, :self.n_components]

        return np.dot(X_data - self.X_mean, self.X_evecs_n)

    def inverse_transform(self, data):
        return np.dot(data, self.X_evecs_n.T)+self.X_mean
```

In [ ]:

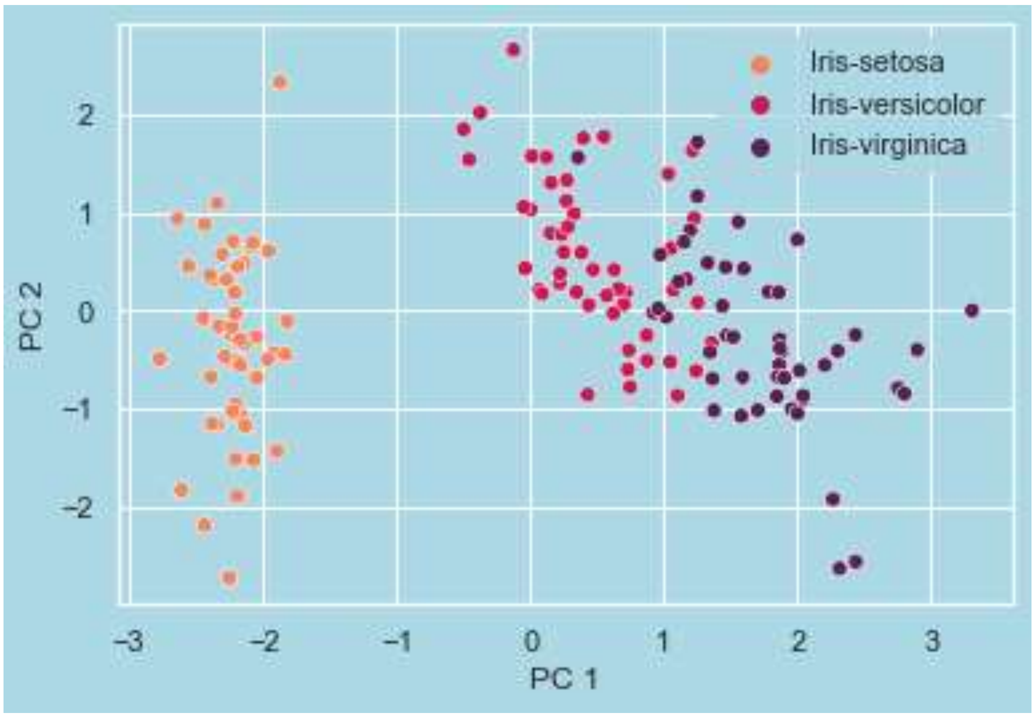
```
#from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X)
principalDf = pd.DataFrame(data=principalComponents, columns=['PC 1', 'PC
2'])
```

In [ ]:

```
sns.scatterplot(data=principalDf, x='PC 1',
                y='PC 2', hue=Y,  palette='rocket_r')
```

Out[ ]:

```
<AxesSubplot:xlabel='PC 1', ylabel='PC 2'>
```

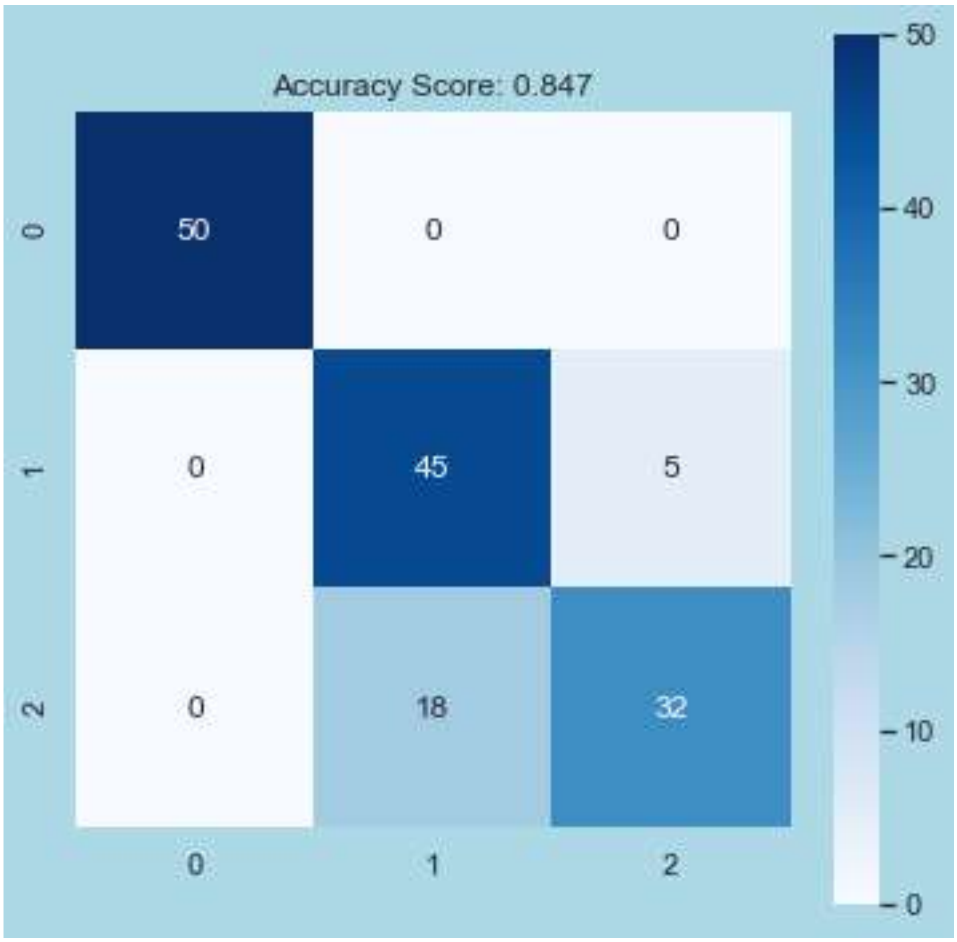


In [ ]:

```
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=3)
```

In [ ]:

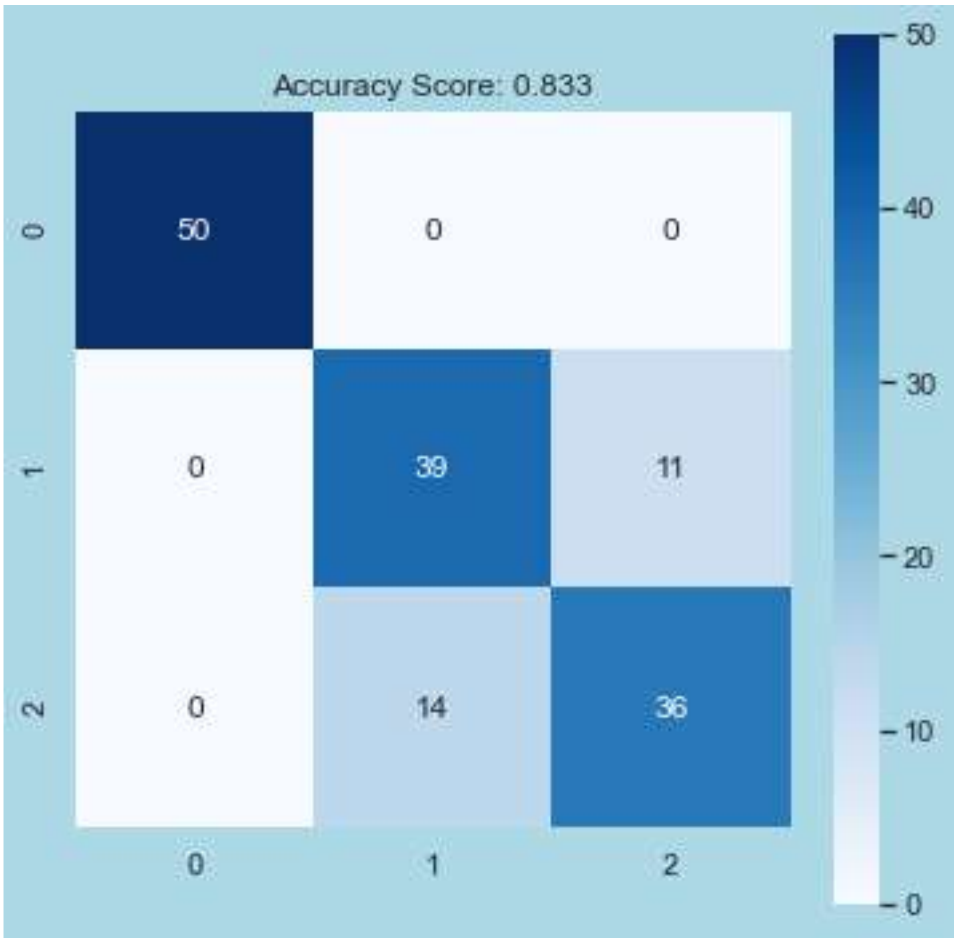
```
pred = model.fit_predict(X)

confusionMatrixAndAccuracyReport(Y_bin, pred)
```



```
Overall Accuracy Score: 0.847
Classwise Accuracy Score: [1.         0.84666667 0.84666667]
```

In [ ]:

```
predPca = model.fit_predict(principalDf)

confusionMatrixAndAccuracyReport(Y_bin, predPca)
```



```
Overall Accuracy Score: 0.833
Classwise Accuracy Score: [1.         0.83333333 0.83333333]
```

In [ ]:

```
def reconstructionError(X_train, X_projected):
    return np.round(np.sum((X_train - X_projected) ** 2, axis=1).mean(), 3)
```

In [ ]:

```
for k in range(3):
    pca_k = PCA(n_components=k)
    pc_x_train = pca_k.fit_transform(X)
    pc_x_projected = pca_k.inverse_transform(pc_x_train)
    print(
        f'The reconstruction error for k = {k+1} is ::
{reconstructionError(X,pc_x_projected)}')
```

```
The reconstruction error for k = 1 is :: 4.0
The reconstruction error for k = 2 is :: 1.089
The reconstruction error for k = 3 is :: 0.168
```