Project report-Spherical K-Means Pattern Discovery in Textures

NAME: RISHI RAJ SINGH
AI AND ML B-4

Problem Statement:
Generate a dummy dataset using Scikit-Learn having high dimensionality (number of features >10) and total 4 classes. For this dataset, first implement K-Means clustering and then use the clusters for classification purpose. Now using the same dataset, implement spherical clustering and then check accuracy for classification. Notice the change in accuracy. You may also plot the obtained clusters from both the methods using t-SNE plots or by projecting data into two dimensions using PCA.

Prerequisites:
- Software: Python 3

Tools:

- Pandas
- Numpy
- Matplotlib
- Sklearn

The classical k-means method of clustering minimizes the sum of squared distances between cluster centres and cluster members. The intuition is that the radial distance from the Cluster- Centre should be similar for all elements of that cluster. The spherical k-means algorithm, however, is equivalent to the k-means algorithm with cosine similarity, a popular method for clustering high-dimensional data. The idea is to set the centre of each cluster such that it makes the angle between components both uniform and minimal.

Implementation:

- Load all required libraries And Dataset

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,f1_score,plot_confusion_matrix
```

In [2]:
```
X, y= make_blobs(n_samples=500, centers=4, n_features=11,random_state=0,cluster_std=0.60)
```

In [3]:
```
X.shape
```

Out[3]: (500, 11)

In [4]:
```
y
```

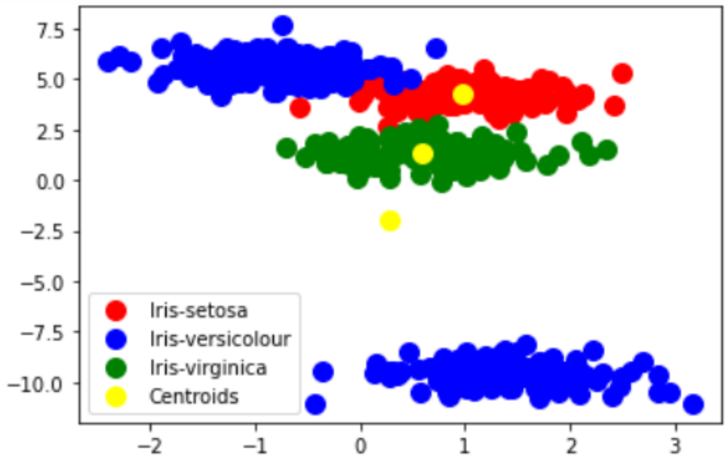
Applying K-Means Classifier:

In [5]:
```
#applying kmeans-classifier
kmeans = KMeans(n_clusters=3,init = 'k-means++',max_iter = 100, n_init = 10,random_state = 0)
```

In [7]:
```
#Predicting the cluster for our data
y_kmeans = kmeans.fit_predict(X)
print(kmeans.cluster_centers_)
#Visualising the clusters
X = np.array(X)
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label = 'Iris-versicolour')
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:,1], s = 100, c = 'yellow', label = 'Centroids')

plt.legend()
```



- Classifier Report

In [8]:
```
print(classification_report(y,y_kmeans))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       125
           1       0.00      0.00      0.00       125
           2       0.00      0.00      0.00       125
           3       0.00      0.00      0.00       125

    accuracy                           0.25       500
   macro avg       0.25      0.25      0.25       500
weighted avg       0.25      0.25      0.25       500
```

```
C:\Users\dell\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1308: UndefinedMetricWarning:
```

- Initialize Centriod

In [11]:
```python
#initializing centriod
```

In [12]:
```python
def init_centroids(K, data_arr, label_arr):
    mean_cent = []
    size_cent = []
    cluster_cent = [[] for i in range(K)]

    for i in range(len(data_arr)):
        for k in range(K):
            if label_arr[i]==k:
                data_pt = np.ravel(data_arr[i,:])/np.linalg.norm(np.ravel(data_arr[i,:]))
                cluster_cent[k].append(data_pt)
    for k in range(K):
        cluster_mat = np.matrix(cluster_cent[k])
        pointNum = cluster_mat.shape[0]
        mean_k = np.mean(cluster_mat, axis=0)
        mean_k = np.ravel(mean_k)/np.linalg.norm(np.ravel(mean_k))
        mean_cent.append(mean_k)
        size_cent.append(pointNum)
    return mean_cent, size_cent
```

- Updating labels and Centriods

In [14]:
```python
def label_update(prev_mean, data_arr, label_arr):
    for i in range(len(data_arr)):
        sim_pt = []
        for k in range(K):
            sim = similarity(data_arr[i], prev_mean[k])
            sim_pt.append(sim)
        sim_arr = np.array(sim_pt)
        new_label = np.argmax(sim_arr)
        label_arr[i] = new_label
    return label_arr
```

In [15]:
```python
#updatingclusters
```

In [16]:
```python
def update_centroids(K, prev_mean, prev_size, data_arr, label_arr):
    cluster_pts = [[] for k in range(K)]

    for i in range(data_arr.shape[0]):
        for k in range(K):
            if label_arr[i]==k:
                data_pt = np.ravel(data_arr[i,:])/np.linalg.norm(np.ravel(data_arr[i,:]))
                cluster_pts[k].append(data_pt)
    for k in range(K):
        print(len(cluster_pts[k]))
        if len(cluster_pts[k])!=0:
            cluster_mat = np.matrix(cluster_pts[k])
            pointNum = cluster_mat.shape[0]
            mean_k = np.mean(cluster_mat, axis=0)
            mean_k = np.ravel(mean_k)/np.linalg.norm(np.ravel(mean_k))
            prev_mean[k] = mean_k
            prev_size[k] = pointNum
        new_mean = prev_mean
        new_size = prev_size
    return new_mean, new_size
```

- Calling And Applying Spherical K-Means

In [17]:

```
#calling and applying spherical-kmeans
```

In [20]:

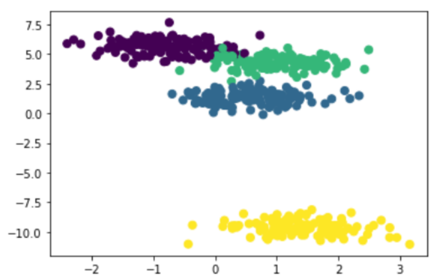
```
def SphericalKMeans(data_arr, label_arr, maxIter):
    prev_mean, prev_size = init_centroids(K, data_arr, label_arr)
    print(prev_size)
    for iter in range(maxIter):
        new_label = label_update(prev_mean, data_arr, label_arr)
        new_mean, new_size = update_centroids(K, prev_mean, prev_size, data_arr, label_arr)
        label_arr = new_label
        prev_mean = new_mean
        prev_size = new_size
        print(f"Iteration: {iter} is completed!")
    return new_mean, new_size,label_arr
```

- Visualizing Spectral

In [26]:

```
from sklearn.cluster import SpectralClustering
model = SpectralClustering(n_clusters=4, affinity='nearest_neighbors',
                           assign_labels='kmeans')
labels = model.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=labels,s=50, cmap='viridis');
```

```
C:\Users\dell\anaconda3\lib\site-packages\sklearn\manifold\_spectral_embedding.py:260: UserWarning: Graph is not fully conne
cted, spectral embedding may not work as expected.
  warnings.warn(
```



- Classification Report

In [28]:
```python
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,f1_score,plot_confusion_matrix
```

In [29]:
```python
print(confusion_matrix(y,labels))
```

```
[[  0   0 125   0]
 [  0 125   0   0]
 [125   0   0   0]
 [  0   0   0 125]]
```

In [30]:
```python
print(classification_report(y,labels))
```

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       125
           1       1.00      1.00      1.00       125
           2       0.00      0.00      0.00       125
           3       1.00      1.00      1.00       125

    accuracy                           0.50       500
   macro avg       0.50      0.50      0.50       500
weighted avg       0.50      0.50      0.50       500
```