Q1: 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.

In [1]:

```
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score
from sklearn.datasets import make_blobs, load_digits
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.cluster import KMeans
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure
```

In [2]:

```
class KMeans_Test:

    def __init__(self, k=2, tolerance=0.002, max_iter=100):
        """
        :param k: the value of k clusters
        :param tolerance: represents how much centroid is going to move. It w
        :param max_iter: times we want to run the testcase
        :param centroid: the dictionary reprents the centroid

        Init operation for the class. Init basic values required for the clas
        Copying the code from sklearn
        """
        self.k = k
        self.tolerance = tolerance
        self.max_iter = max_iter
        self.centroids = {}
        self.classifications = {}


    def fit(self, _data):
        """
        Performs the actual fit function as reprented by KMeans library
        :return:
        """

        for eachK in range(0, self.k):
            self.centroids[eachK] = _data[eachK]

        while True:
            each_iter = 0
```

```
            isOptimized = True

            # Clearing out classifications after each iteration because each
            # 0,1 index but value will change every time.
            # Classification will cleaned everytime
            self.classifications = {}

            for eachK in range(0, self.k):
                self.classifications[eachK] = []

            # Adding feature to an centroid
            for eachF in range(0,len(_data)):
                feature = _data[eachF]
                distances = []
                for centroid in self.centroids:
                    distances.append(np.linalg.norm(feature - self.centroids[
                min_distance = min(distances)
                self.classifications[distances.index(min_distance)].append(fe

            # Comparing the centroids so that we can verify how much they are
            # the tolerance value
            temp_centroids = dict(self.centroids)

            for classification in self.classifications:
                # Taking average value of the classifications and computing c
                self.centroids[classification] = np.average(self.classificati

            for centroid in self.centroids:
                original_centroid = temp_centroids[centroid]
                current_centroid = self.centroids[centroid]
                centroid_space = current_centroid - original_centroid
                if np.sum((centroid_space / original_centroid) * 100.0) > sel
                    isOptimized = False


            if each_iter == self.max_iter or isOptimized:
                break

            each_iter += 1


    def predict(self, _data):
        distances = []
        for centroid in self.centroids:
            distances.append(np.linalg.norm(_data - self.centroids[centroid])
        return distances.index(min(distances))

    def plot_data(self):
        colors = 10 * ['#1f77b4', '#ff7f0e']
        for centroid in self.centroids:
            plt.scatter(self.centroids[centroid][0], self.centroids[centroid]
                        marker = ".",
                        color="k",
                        s=150,
                        linewidths=5)

        for classification in self.classifications:
            color = colors[classification]
            for feature in self.classifications[classification]:
                plt.scatter(feature[0],
                            feature[1],
                            marker="x",
                            color=color,
                            s=150,
                            linewidths=5)
```

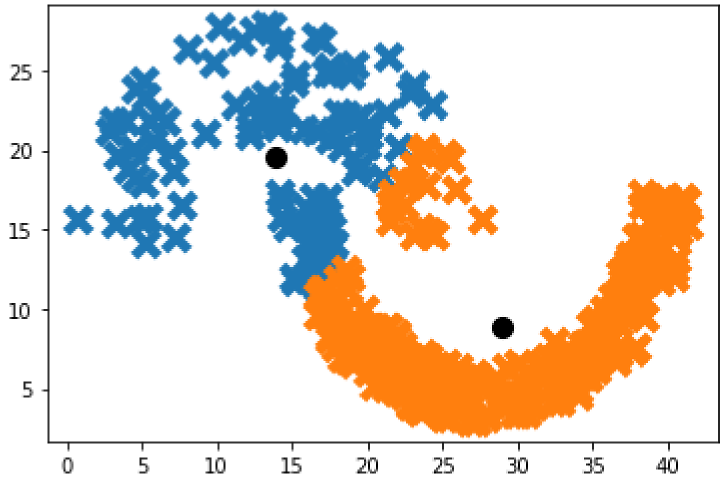
```
    def compare(self, _data):
        predicted = []
        actual = _data[:, 2]

        for each in _data:
            predicted_op = self.predict(each)
            predicted.append(predicted_op)

        centroids = np.zeros([4], dtype = int)
        for each_actual, each_predicted in zip(actual, predicted):
            if each_actual == 2 and each_predicted == 0:
                centroids[0] = centroids[0] + 1
            elif each_actual == 1 and each_predicted == 1:
                centroids[1] = centroids[1] + 1
            elif each_actual == 2 and each_predicted == 1:
                centroids[2] = centroids[2] + 1
            elif each_actual == 1 and each_predicted == 0:
                centroids[3] = centroids[3] + 1

        print("Accuracy = ",100 * (centroids[0] + centroids[1])/np.sum(centro
```

In [3]:

```
# Reading input provided in the file
def read_input():
    jain_csv = pd.read_csv('jain.txt', sep='\t')
    jain_arr = (np.array(jain_csv))
    return jain_arr
```

In [4]:

```
data = read_input()
model = KMeans_Test(2)
model.fit(data)
model.plot_data()
```



In [5]:

```
model.compare(data)
```

```
Accuracy =  84.13978494623656
```

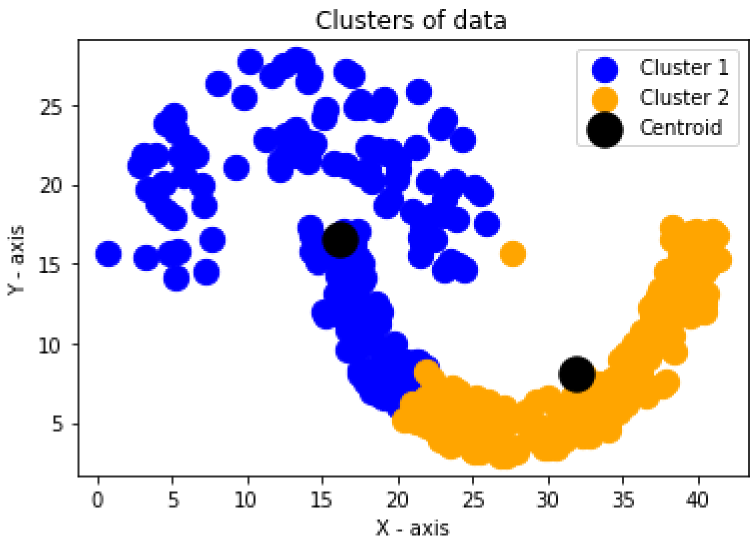
In [6]:

```
kmeans = KMeans(n_clusters=2, init='k-means++', random_state= 42)
y_predict= kmeans.fit_predict(data)
```

In [7]:

```
plt.scatter(data[y_predict == 0, 0], data[y_predict == 0, 1], s = 150, c = 'b
plt.scatter(data[y_predict == 1, 0], data[y_predict == 1, 1], s = 150, c = 'o
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =
```

```
plt.title('Clusters of data')
plt.xlabel('X - axis')
plt.ylabel('Y - axis')
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

Refrences: https://www.youtube.com/watch?v=H4JSN_99kig https://www.youtube.com/watch?v=HRoeYblYhkg https://www.javatpoint.com/k-means-clustering-algorithm-in-machine-learning

In [ ]: