Technique Assignment 5: Clustering

Cogs Fall 2020

Due: Thursday December 3 11:59pm

100 points total

Xing Hong A15867895

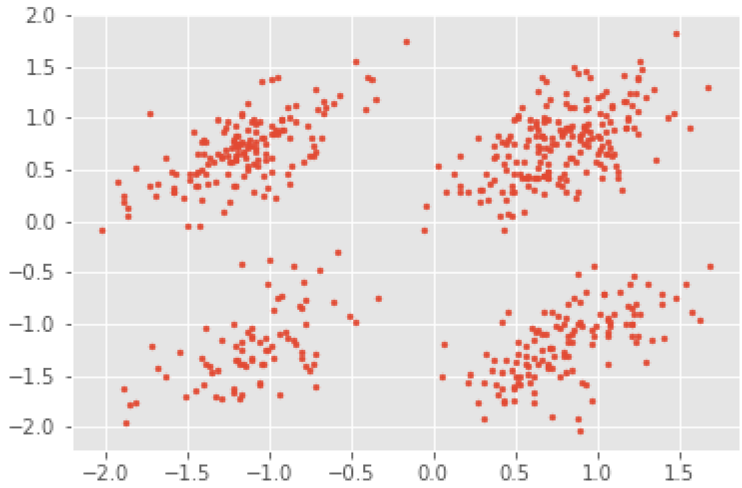
In [1]:
```
%matplotlib inline
import numpy as np
from scipy.io import loadmat
from matplotlib import pyplot as plt

plt.style.use('ggplot')
```

In [2]:
```
## Load the data to be clustered (X)
## Load the set of priors, p1-p3

data = loadmat('cluster_data.mat')
X = data['kmeandata']
print(X.shape)
```

```
(560, 2)
```

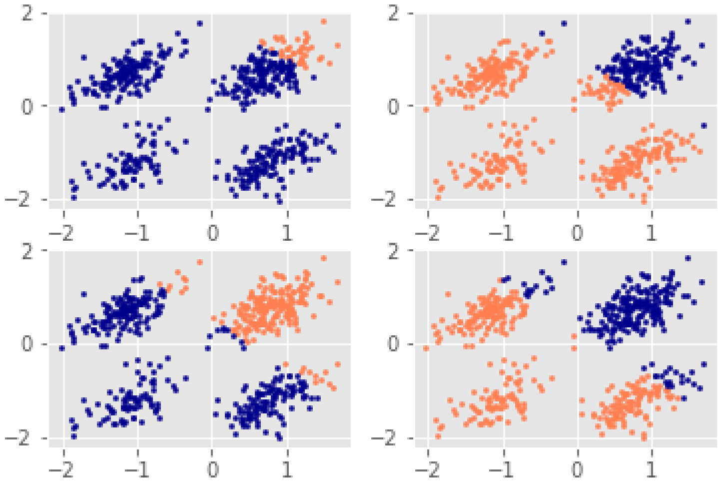
In [3]:
```
## Plot the data with no cluster labels
plt.scatter(X[:, 0], X[:, 1], s=8)
```

Out[3]: <matplotlib.collections.PathCollection at 0x7ff027b3d100>



In [4]:
```
k = 2

# Assign 2 clusters (1st initialization)
C = [[-2, -2], [4, 4]]

for itr in range(4):  # set number of k-mean iterations
    # initialize distance and cluster membership
    cluster_ind = np.zeros(len(X))
    distance = np.zeros((len(X), k))

    # find distance of every point to each centroid, and cluster membership
```

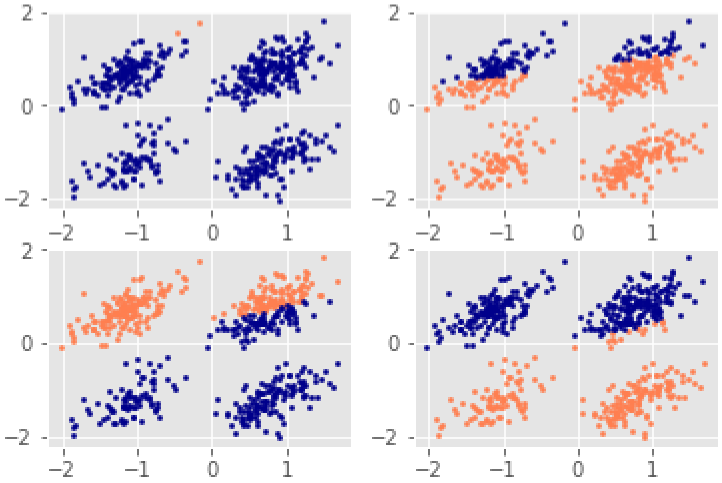
```
    d1 = np.sqrt(pow(X[:, 0] - C[0][0], 2) + pow(X[:, 1] - C[0][1], 2))
    d2 = np.sqrt(pow(X[:, 0] - C[1][0], 2) + pow(X[:, 1] - C[1][1], 2))
    distance[:, 0] = d1
    distance[:, 1] = d2
    cluster_ind = np.where(distance[:, 0] < distance[:, 1], 1, 0)

    # update cluster centroids
    for i in range(k):
        C[i] = np.array([
            np.mean(X[np.where(cluster_ind == i), 0]),
            np.mean(X[np.where(cluster_ind == i), 1])
        ])

    # Plot each iteration of k-means to show the first 4
    # Points should be assigned a color based on which cluster they belong to

    plt.subplot(2, 2, itr + 1)
    plt.scatter(X[np.where(cluster_ind == 0), 0],
                X[np.where(cluster_ind == 0), 1],
                s=5,
                c='coral')
    plt.scatter(X[np.where(cluster_ind == 1), 0],
                X[np.where(cluster_ind == 1), 1],
                s=5,
                c='darkblue')
```



In [5]:

```
k = 2

# Assign 2 clusters (2nd initialization)
C = [[0, -1], [-1, 4]]

# Use code from above to generate similar graphs, but with new cluster initia
for itr in range(4):
    cluster_ind = np.zeros(len(X))
    distance = np.zeros((len(X), k))

    d1 = np.sqrt(pow(X[:, 0] - C[0][0], 2) + pow(X[:, 1] - C[0][1], 2))
    d2 = np.sqrt(pow(X[:, 0] - C[1][0], 2) + pow(X[:, 1] - C[1][1], 2))
    distance[:, 0] = d1
    distance[:, 1] = d2
    cluster_ind = np.where(distance[:, 0] < distance[:, 1], 1, 0)

    for i in range(k):
        C[i] = np.array([
            np.mean(X[np.where(cluster_ind == i), 0]),
            np.mean(X[np.where(cluster_ind == i), 1])
        ])
```

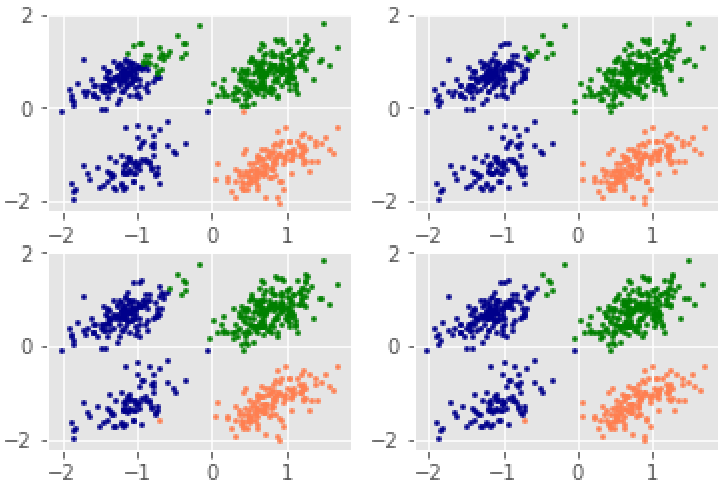
```
    plt.subplot(2, 2, itr + 1)
    plt.scatter(X[np.where(cluster_ind == 0), 0],
                X[np.where(cluster_ind == 0), 1],
                s=5,
                c='coral')
    plt.scatter(X[np.where(cluster_ind == 1), 0],
                X[np.where(cluster_ind == 1), 1],
                s=5,
                c='darkblue')
```



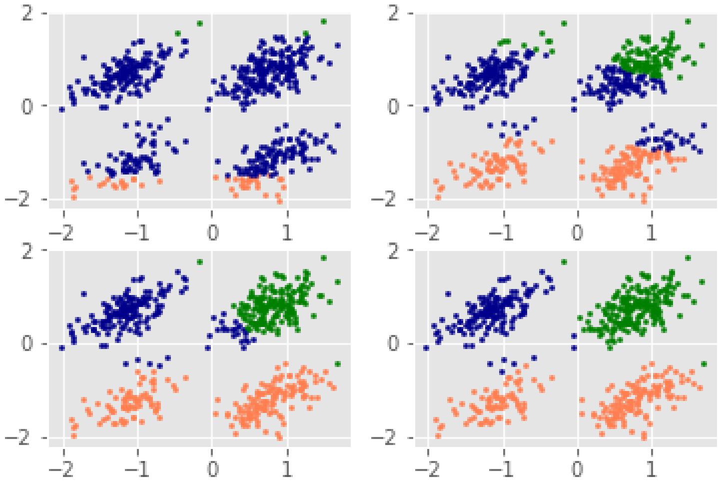
In [6]:

```
k = 3

# Assign 3 clusters (1st initialization)
C = [[2, -2], [-2, -2], [2, 2]]

# Use code from above
for itr in range(4):
    cluster_ind = np.zeros(len(X))
    distance = np.zeros((len(X), k))

    d1 = np.sqrt(pow(X[:, 0] - C[0][0], 2) + pow(X[:, 1] - C[0][1], 2))
    d2 = np.sqrt(pow(X[:, 0] - C[1][0], 2) + pow(X[:, 1] - C[1][1], 2))
    d3 = np.sqrt(pow(X[:, 0] - C[2][0], 2) + pow(X[:, 1] - C[2][1], 2))
    distance[:, 0] = d1
    distance[:, 1] = d2
    distance[:, 2] = d3

    for i in range(len(X)):
        cluster_ind[i] = np.where(distance[i, :] == np.min(distance[i, :]))[0

    for i in range(k):
        C[i] = np.array([
            np.mean(X[np.where(cluster_ind == i), 0]),
            np.mean(X[np.where(cluster_ind == i), 1])
        ])

    plt.subplot(2, 2, itr + 1)
    plt.scatter(X[np.where(cluster_ind == 0), 0],
                X[np.where(cluster_ind == 0), 1],
                s=5,
                c='coral')
    plt.scatter(X[np.where(cluster_ind == 1), 0],
                X[np.where(cluster_ind == 1), 1],
                s=5,
                c='darkblue')
    plt.scatter(X[np.where(cluster_ind == 2), 0],
                X[np.where(cluster_ind == 2), 1],
```

```
                s=5,
                c='g')
```

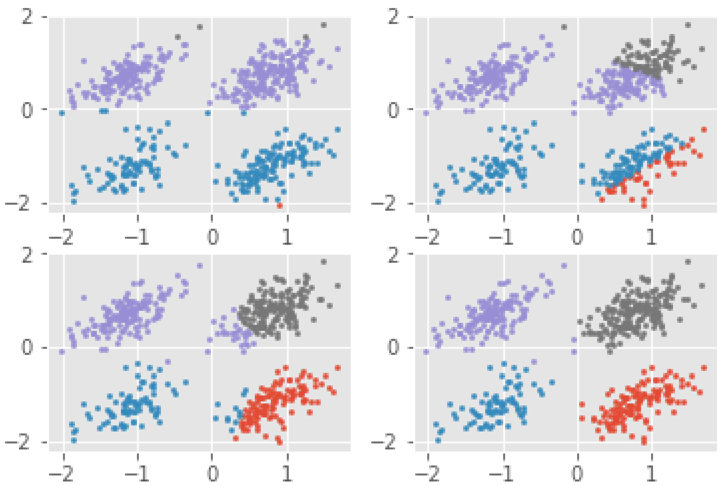


In [7]:
```
k = 3

# Assign 3 clusters (2nd initialization)
C = [[0, -3], [0, 0], [0, 3]]

for itr in range(4):
    cluster_ind = np.zeros(len(X))
    distance = np.zeros((len(X), k))

    d1 = np.sqrt(pow(X[:, 0] - C[0][0], 2) + pow(X[:, 1] - C[0][1], 2))
    d2 = np.sqrt(pow(X[:, 0] - C[1][0], 2) + pow(X[:, 1] - C[1][1], 2))
    d3 = np.sqrt(pow(X[:, 0] - C[2][0], 2) + pow(X[:, 1] - C[2][1], 2))
    distance[:, 0] = d1
    distance[:, 1] = d2
    distance[:, 2] = d3

    for i in range(len(X)):
        cluster_ind[i] = np.where(distance[i, :] == np.min(distance[i, :]))[0

    for i in range(k):
        C[i] = np.array([
            np.mean(X[np.where(cluster_ind == i), 0]),
            np.mean(X[np.where(cluster_ind == i), 1])
        ])

    plt.subplot(2, 2, itr + 1)
    plt.scatter(X[np.where(cluster_ind == 0), 0],
                X[np.where(cluster_ind == 0), 1],
                s=5,
                c='coral')
    plt.scatter(X[np.where(cluster_ind == 1), 0],
                X[np.where(cluster_ind == 1), 1],
                s=5,
                c='darkblue')
    plt.scatter(X[np.where(cluster_ind == 2), 0],
                X[np.where(cluster_ind == 2), 1],
                s=5,
                c='g')
```



In [8]:

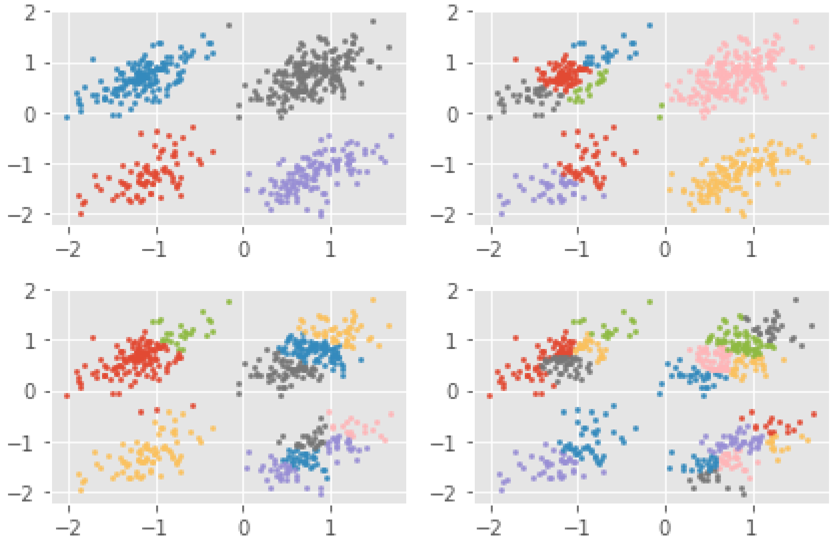
```
k = 4

# Assign 4 clusters
C = [[0, -3], [0, -1], [0, 1], [0, 2]]

for itr in range(4):
    cluster_ind = np.zeros(len(X))
    distance = np.zeros((len(X), k))

    for i in range(k):
        distance[:, i] = np.sqrt(
            pow(X[:, 0] - C[i][0], 2) + pow(X[:, 1] - C[i][1], 2))

    for i in range(len(X)):
        cluster_ind[i] = np.where(distance[i, :] == np.min(distance[i, :]))[0

    for i in range(k):
        C[i] = np.array([
            np.mean(X[np.where(cluster_ind == i), 0]),
            np.mean(X[np.where(cluster_ind == i), 1])
        ])

    plt.subplot(2, 2, itr + 1)
    for i in range(k):
        plt.scatter(
            X[np.where(cluster_ind == i), 0],
            X[np.where(cluster_ind == i), 1],
            s=5,
        )
```



4 . (10points) Which value of k will produce the best result? How can you tell?

From the graph, my eye can see 4 distinct clusters, which in this case, k=4 will produce the best result even though higher k values will produce lower error. We can plot out the error or sum of distance squares within cluster, and see where the turning point appears (where SSE drops the most rapid under the specific k).

In [9]:
```
k = [4, 8, 12, 20]

for kvalue in enumerate(k):
    # Assign k clusters as randomly selected points from the dataset
    index = np.random.choice(np.arange(X.shape[0]),
                             size=kvalue[1],
                             replace=False)
    # C = X[np.random.randint(X.shape[0], size=kvalue[1])]
    C = X[index]
    for itr in range(200):
        cluster_ind = np.zeros(len(X))
        distance = np.zeros((len(X), kvalue[1]))

        for i in range(kvalue[1]):
            distance[:, i] = np.sqrt(
                pow(X[:, 0] - C[i][0], 2) + pow(X[:, 1] - C[i][1], 2))

        for i in range(len(X)):
            cluster_ind[i] = np.where(
                distance[i, :] == np.min(distance[i, :]))[0]

        for i in range(kvalue[1]):
            C[i] = np.array([
                np.mean(X[np.where(cluster_ind == i), 0]),
                np.mean(X[np.where(cluster_ind == i), 1])
            ])

    plt.subplot(2, 2, kvalue[0] + 1)
    for i in range(kvalue[1]):
        plt.scatter(
            X[np.where(cluster_ind == i), 0],
            X[np.where(cluster_ind == i), 1],
            s=5,
        )
        plt.tight_layout()
```



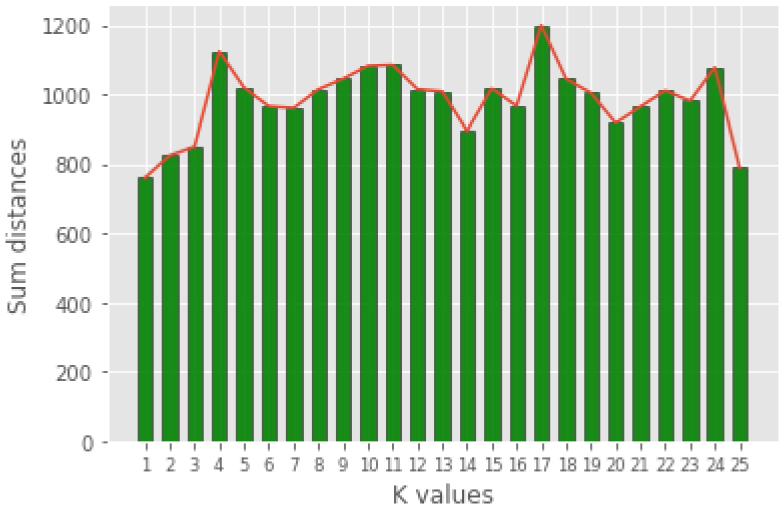
In [10]:
```
k = list(range(1, 26))
distances = np.zeros(len(k))

for kvalue in k:
```

```
    index = np.random.choice(np.arange(X.shape[0]), size=kvalue, replace=Fals
    C = X[index]
    for itr in range(500):
        cluster_ind = np.zeros(len(X))
        distance = np.zeros((len(X), kvalue))

        for i in range(kvalue):
            distance[:, i] = np.sqrt(
                pow(X[:, 0] - C[i][0], 2) + pow(X[:, 1] - C[i][1], 2))

        for i in range(len(X)):
            cluster_ind[i] = np.where(
                distance[i, :] == np.min(distance[i, :]))[0][0]

        for i in range(kvalue):
            C[i] = np.array([
                np.mean(X[np.where(cluster_ind == i), 0]),
                np.mean(X[np.where(cluster_ind == i), 1])
            ])

    for i in range(len(X)):
        distances[kvalue - 1] += distance[i, kvalue - 1]
```

In [11]:

```
plt.xlabel('K values')
plt.ylabel('Sum distances')
rn = [
    '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14'
    '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25'
]
x = np.arange(len(rn))
xticks1 = list(rn)
plt.xticks(x, xticks1, size='small')
plt.bar(x,
        height=distances,
        facecolor="green",
        edgecolor="black",
        width=0.6,
        align='center',
        alpha=0.9)
plt.plot(distances)
plt.tight_layout
```

Out[11]: <function matplotlib.pyplot.tight_layout(*, pad=1.08, h_pad=None, w_pad=None, rect=None)>



a. From the graph above, I think 6 represents the data best, since it is the first turning point of total distances in an significant degree while it is relatively small in K value. As for the second turning point, which is 11, may cause overfittinng. While the K increases, most lower points of error is higher than the one of k=6, therefore k=6 can well represents the data without overfitting.

b. The jumping of distances may caused by the initial centers, since K-means is unstable under noises and initial, it may perform badly with a local optimal value based on the initial. More clusters can lead to a smaller total distances while some of them can have an smaller distance. However, these models may cause overfitting.

In [13]:

```
k = list(range(1, 26))
distances = np.zeros(len(k))
distances_2 = np.ones(25)
distances_2 *= 3000

for kvalue in k:
    for i in range(5):
        index = np.random.choice(np.arange(X.shape[0]),
                                 size=kvalue,
                                 replace=False)
        C = X[index]
        for itr in range(200):
            cluster_ind = np.zeros(len(X))
            distance = np.zeros((len(X), kvalue))

            for i in range(kvalue):
                distance[:, i] = np.sqrt(
                    pow(X[:, 0] - C[i][0], 2) + pow(X[:, 1] - C[i][1], 2))

            for i in range(len(X)):
                cluster_ind[i] = np.where(
                    distance[i, :] == np.min(distance[i, :]))[0][0]

            for i in range(kvalue):
                C[i] = np.array([
                    np.mean(X[np.where(cluster_ind == i), 0]),
                    np.mean(X[np.where(cluster_ind == i), 1])
                ])

        for i in range(len(X)):
            distances[kvalue - 1] += distance[i, kvalue - 1]
        if distances[kvalue - 1] < distances_2[kvalue - 1]:
            distances_2[kvalue - 1] = distances[kvalue - 1]
```

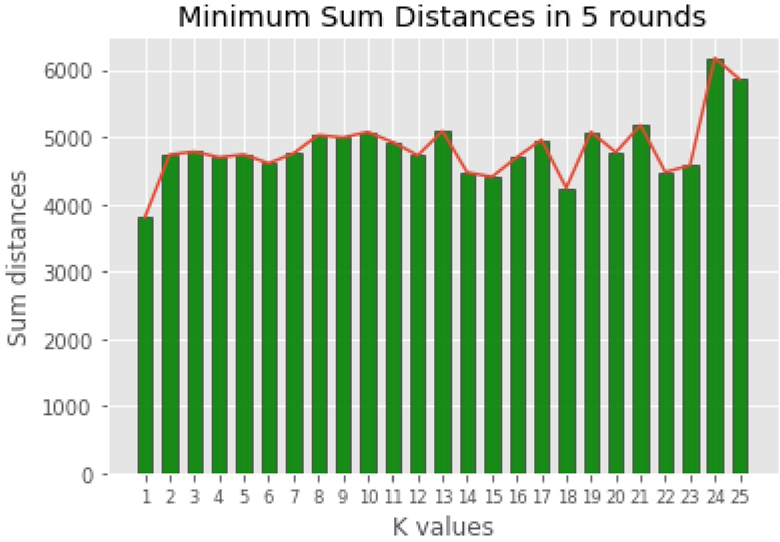
In [14]:

```
plt.title('Minimum Sum Distances in 5 rounds')
plt.xlabel('K values')
plt.ylabel('Sum distances')
rn = [
    '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14'
    '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25'
]
x = np.arange(len(rn))
xticks1 = list(rn)
plt.xticks(x, xticks1, size='small')
plt.bar(x,
        height=distances,
        facecolor="green",
        edgecolor="black",
        width=0.6,
        align='center',
```

```
        alpha=0.9)
plt.plot(distances)
plt.tight_layout
```

Out[14]:

```
<function matplotlib.pyplot.tight_layout(*, pad=1.08, h_pad=None, w_pad=None, rect=None)>
```



In [ ]: