08_claret

November 11, 2018

1 Partical Work 08 - Clustering algorithms

- Author: *Romain Claret*
- Due-date: *12.11.2018*

1.1 Exercice 1 Getting the data

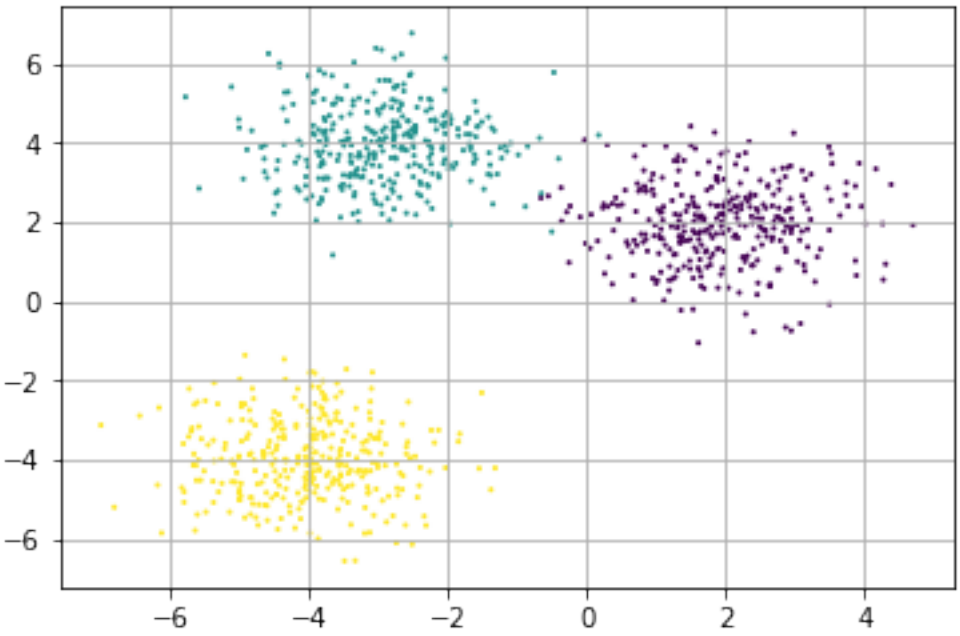
1.1.1 a) Load the two given datasets

```
In [1]: #import pickle
        # 'ascii' codec can't decode byte 0xee in position 6: ordinal not in range(128)
        # X1,label1 = pickle.load(open("dataset_1.pkl","rb"))

        import pandas as pd
        X1,label1 = pd.read_pickle("dataset_1.pkl")
        X2,label2 = pd.read_pickle("dataset_2.pkl")
```

1.1.2 b) Visualize the data using various color for each unique labels

```
In [2]: import numpy as np
        import matplotlib.pyplot as plt
        %matplotlib inline

        def get_plot_colors(label):
            plot_color_label = []
            for i in label:
                if i == 0:
                    plot_color_label.append("blue")
                elif i == 1:
                    plot_color_label.append("orange")
                else:
                    plot_color_label.append("green")
            return plot_color_label

        def plot_clusters(X, colors, size, m="o"):
            plt.scatter([X[i][0] for i in range(len(X))],
                        [X[i][1] for i in range(len(X))],
                        c=colors,
```

```
                    s=size,
                    marker=m)
plot_clusters(X1, label1, 1)


plt.grid(True)
plt.show()
```

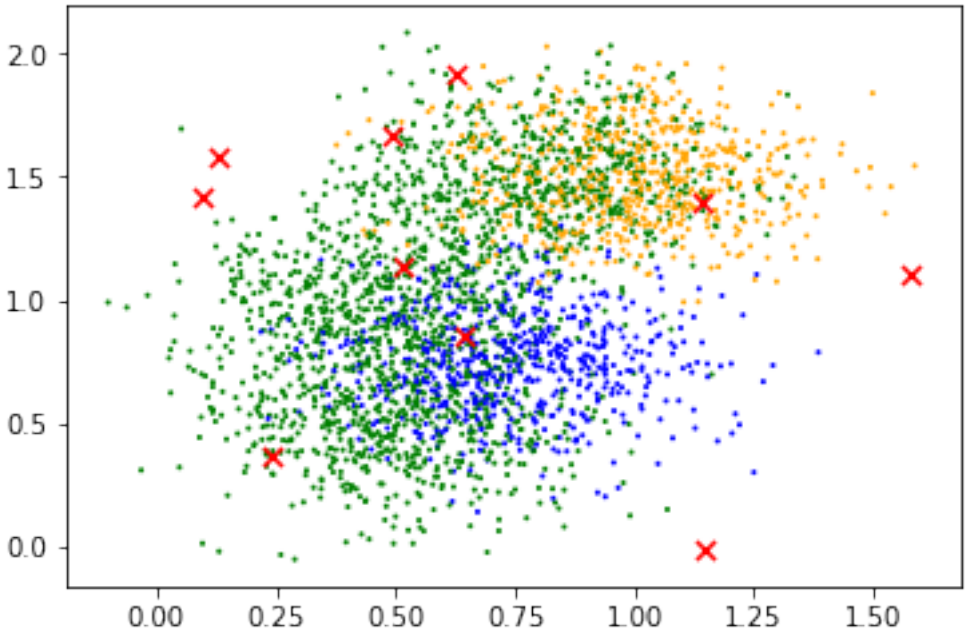


1.2 Exercice 2 The k-means algorithm

1.2.1 a) Initialise the centroids 1 ,2 , ..., K

In [3]:
```
import random

def get_rand_centroids(X, K):
    random_centroids = []

    xmin = np.min([X[i][0] for i in range(len(X))])
    ymin = np.min([X[i][1] for i in range(len(X))])
    xmax = np.max([X[i][0] for i in range(len(X))])
    ymax = np.max([X[i][1] for i in range(len(X))])

    for i in range(K):
        random_centroids.append([random.uniform(xmin, xmax),
                                 random.uniform(ymin, ymax)])
    return random_centroids
```

```
K=10
random_centroids = get_rand_centroids(X2, K)

plot_clusters(X2, get_plot_colors(label2), 1)
plot_clusters(random_centroids, "red", 50, "x")

plt.show()
```



1.2.2 b) Find the closest centroid for each point and reevaluate the centroids

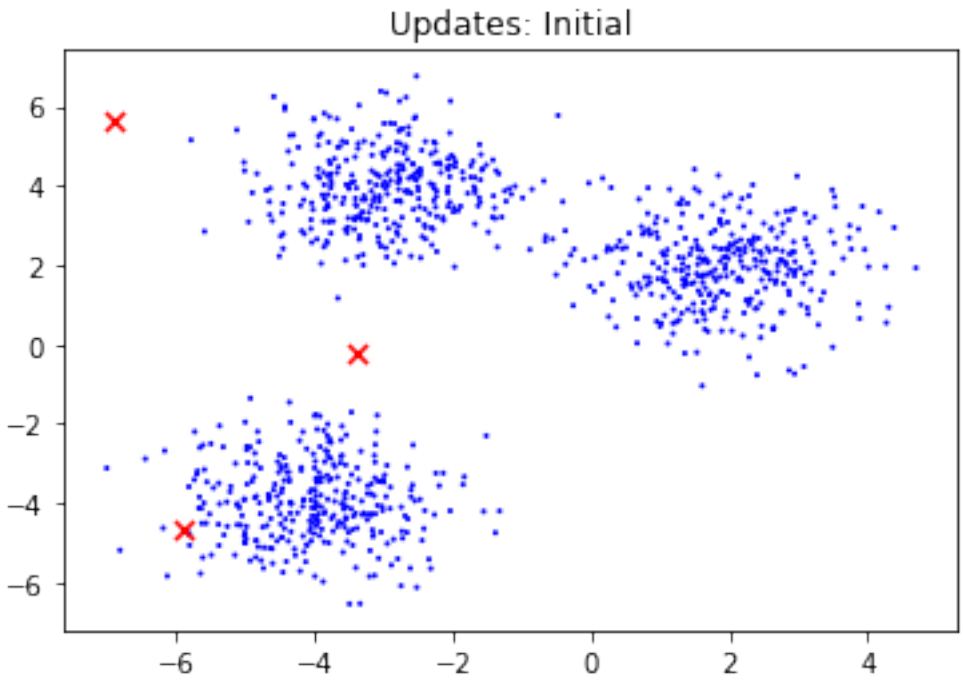
```
In [4]: from scipy.spatial import distance

        def my_kmean(k_X, k_centroids, k_current_predict):
            centroids_updated = 0

            for i in range(len(k_X)):
                current_cluster = 0
                current_dist = distance.euclidean(k_X[i], k_centroids[0])

                for k in range(1, len(k_centroids)):
                    dist = distance.euclidean(k_X[i], k_centroids[k])
                    if dist < current_dist:
                        current_dist = dist
                        current_cluster = k
```

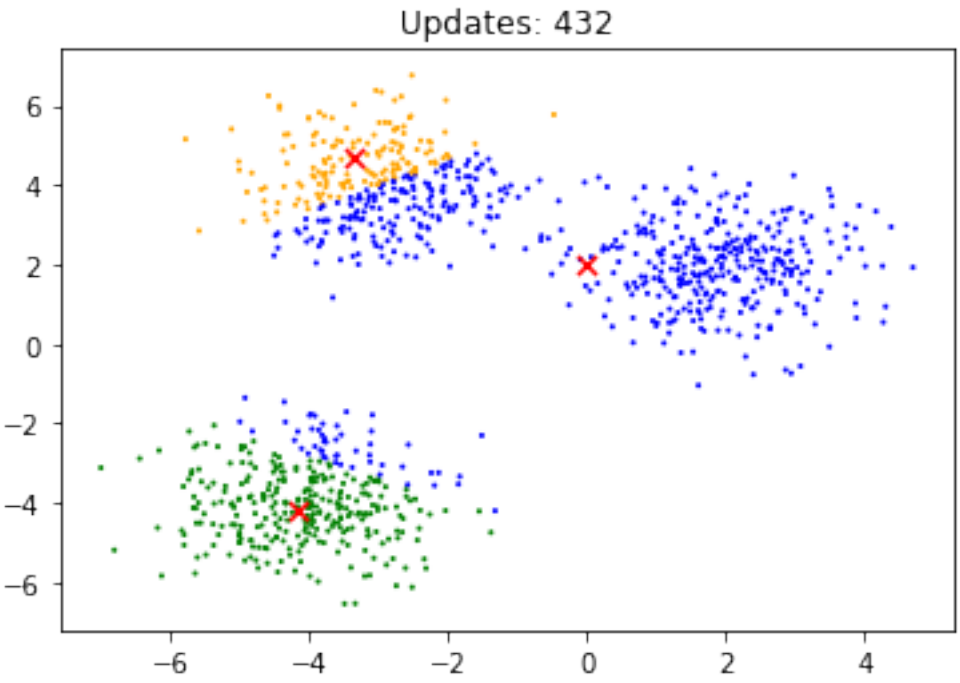
```
            if k_current_predict[i] != current_cluster:
                centroids_updated += 1

            k_current_predict[i] = current_cluster

        for k in range(len(k_centroids)):
            x_move, y_move, count = 0, 0, 0
            for i in range(len(k_current_predict)):
                if k_current_predict[i] == k:
                    x_move += k_X[i][0]
                    y_move += k_X[i][1]
                    count += 1
            if count != 0:
                centroids[k][0] = x_move / count
                centroids[k][1] = y_move / count
        #print(k_centroids)

        return centroids_updated, k_centroids, k_current_predict
```

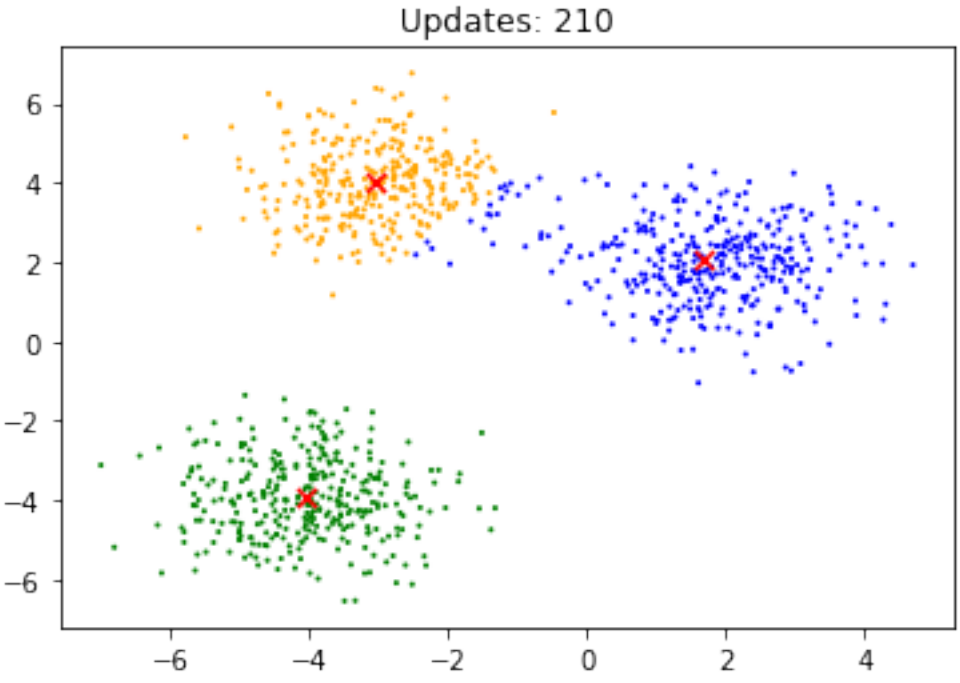
1.2.3 c) Return the centroids and the label predicted.

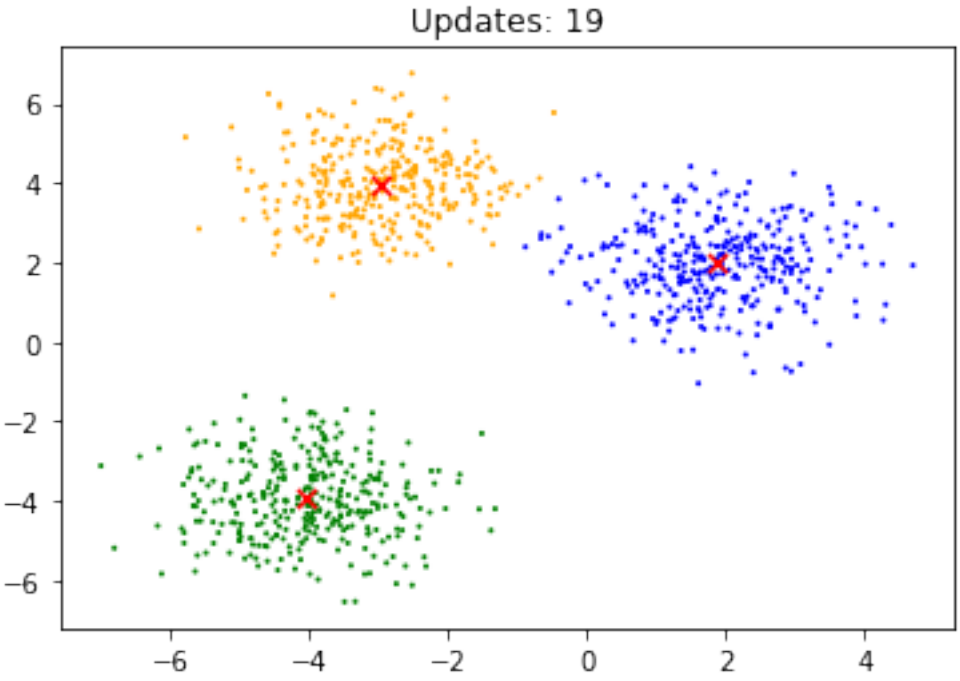
```
In [5]: def plot_update(X, updates, centro, predict):
            plot_clusters(X, get_plot_colors(predict), 1)
            plot_clusters(centro, "red", 50, "x")

            plt.title("Updates: "+ str(updates))
            plt.show()

        current_predict = np.zeros(len(X1))
        centroids = get_rand_centroids(X1, 3)

        updates_hist = []
        predict_hist = []
        centroids_hist = []

        init_predict = current_predict
        init_centroids = centroids

        plot_update(X1, "Initial", init_centroids, init_predict)


        while True:
            updates, centroids, current_predict = my_kmean(X1,
                                                           centroids,
                                                           current_predict)
            updates_hist.append(updates)
            predict_hist.append(current_predict) # this is BS, doesn't append
            centroids_hist.append(centroids) # this is BS, doesn't append
```

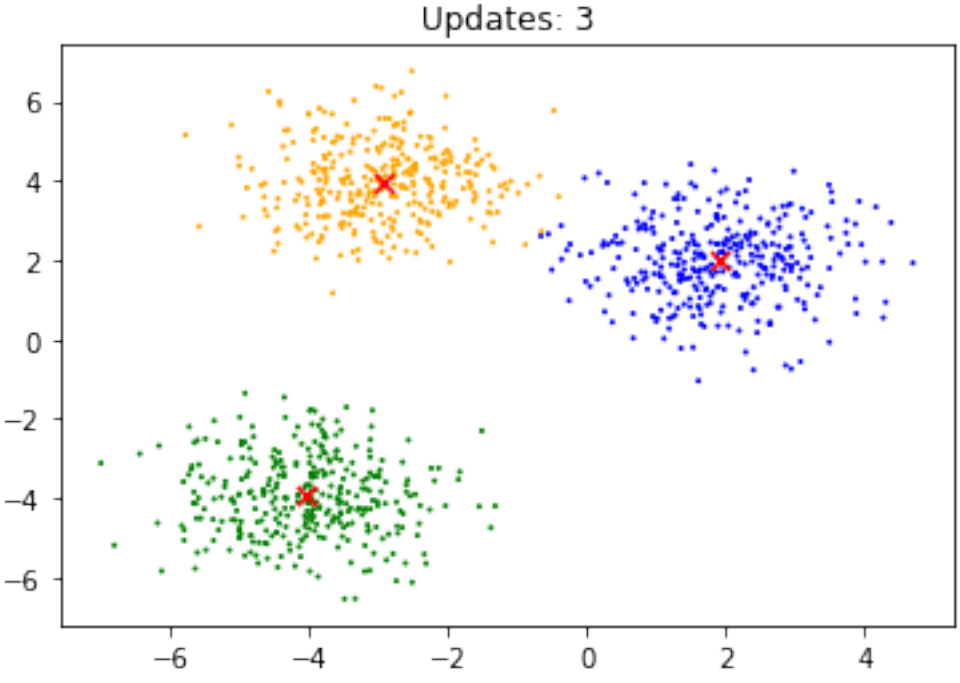
```
    plot_update(X1, updates, centroids, current_predict)


    if updates == 0:
        break
#print(centroids_hist[0]==centroids_hist[1])
```

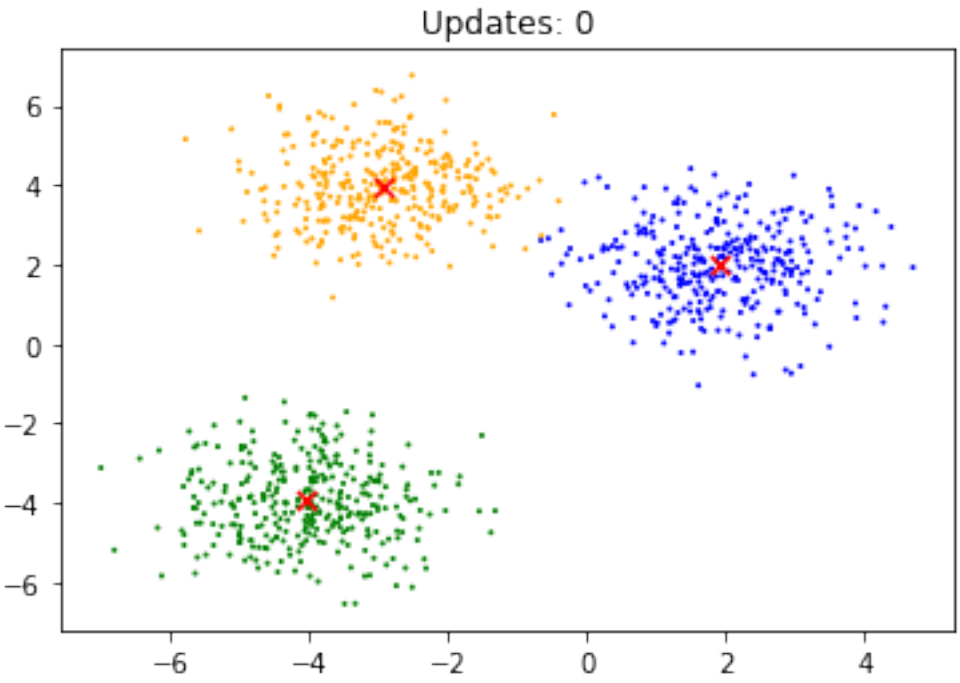



Updates: 432
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4


Updates: 210
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4


Updates: 19
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4

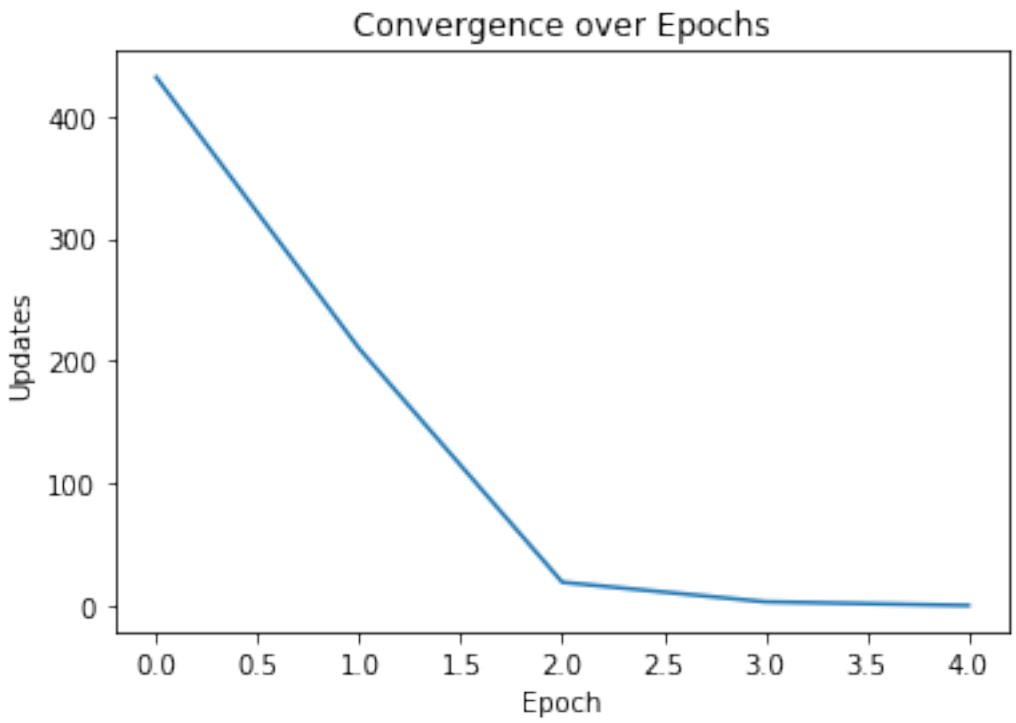

Updates: 3
6
4
2
0
−2
−4
−6
−6
−4
−2
0
2
4



1.3 Exercice 3 Evaluate your model

- Visualize your convergence criteria over the epochs (One epoch is a complete visit of the training set.) using the dataset 1.

```
In [6]: plt.plot(np.arange(0,len(updates_hist)), updates_hist)
        plt.title("Convergence over Epochs")
        plt.xlabel("Epoch")
        plt.ylabel("Updates")
        plt.show()
```



- Visualize the output of your k-means on the dataset 1. THIS IS BS: WHY THE LIST IS NOT APPENDING CORRECTLY? I HAVE TO DO IT IN THE LOOP ABOVE TO MAKE IT WORK...

```
In [7]: #plot_update(X1, "Initial", init_centroids, init_predict)
        #print(centroids_hist[4])
        #for i in range(0, len(centroids_hist)):
            #print(updates_hist[i])
            #print(centroids_hist[i])
            #print(predict_hist[i])
            #plot_update(X1,
            #            updates_hist[i],
            #            centroids_hist[i],
            #            predict_hist[i])
        #plt.show()
```

- Do you experience sensitivity to the initial values of the centroids ? Is your strategy for initialization working well in most cases ?

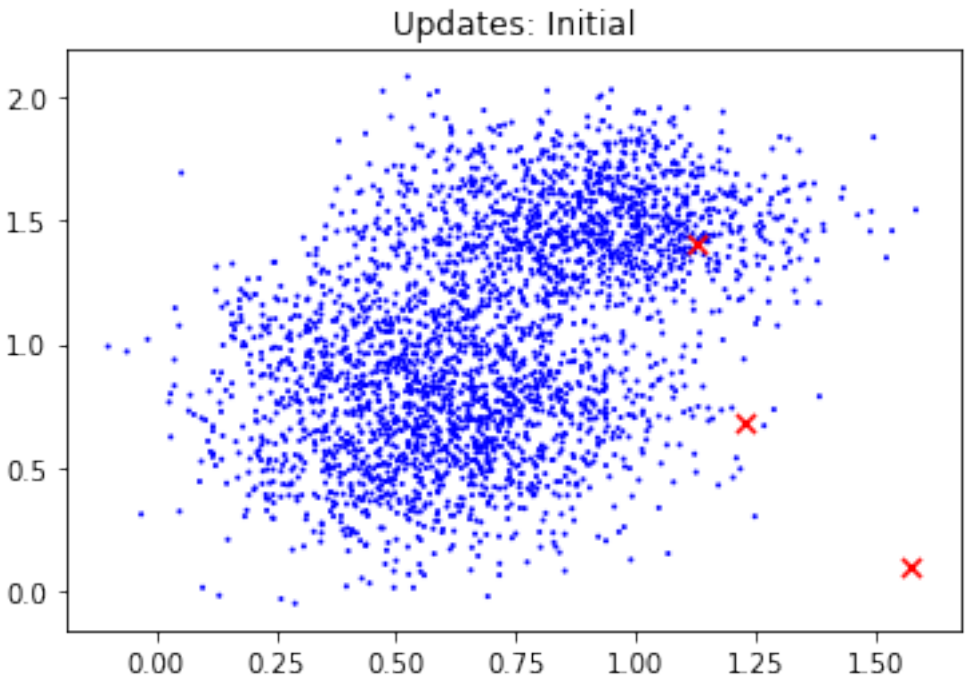
```
- It works in most of the cases on this dataset. The convergence over epochs varies however fr
```

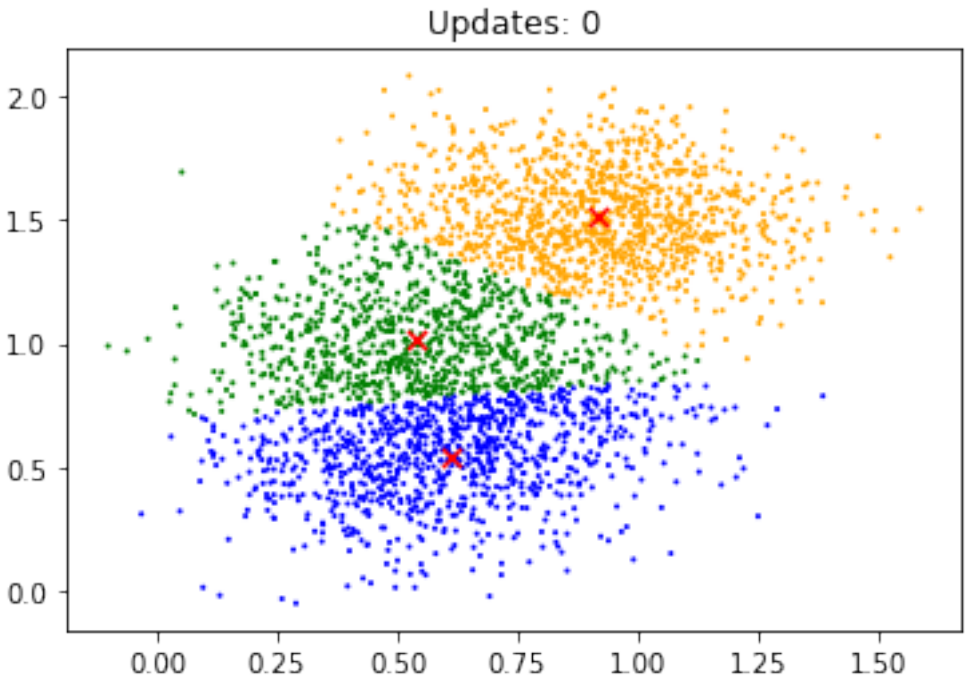
- Document your convergence criteria. Could you think about other convergence criteria ?

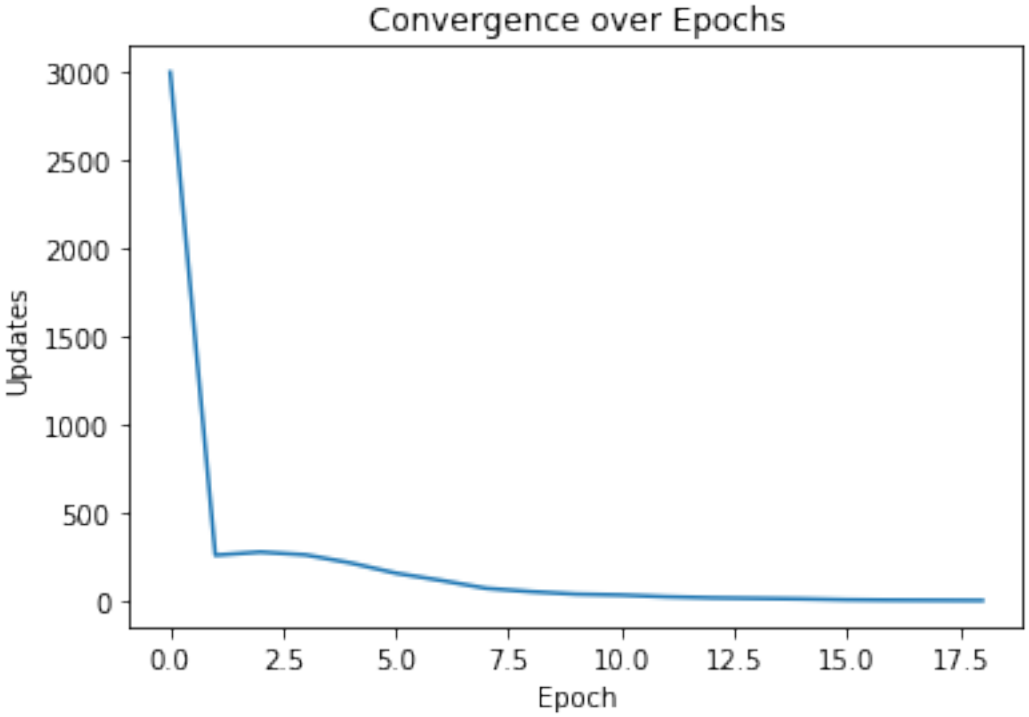
```
- Updates per epochs based on the distance.
```

- Visualize your convergence criteria over time using the dataset 2.

```
In [8]: current_predict = np.zeros(len(X2))
        centroids = get_rand_centroids(X2, 3)
        updates_hist = []
        plot_update(X2, "Initial", centroids, current_predict)


        while True:
            updates, centroids, current_predict = my_kmean(X2,
                                                           centroids,
                                                           current_predict)
            updates_hist.append(updates)
            if updates == 0:
                break
        plot_update(X2, updates, centroids, current_predict)

        plt.plot(np.arange(0,len(updates_hist)), updates_hist)
        plt.title("Convergence over Epochs")
        plt.xlabel("Epoch")
        plt.ylabel("Updates")
        plt.show()
```







- Visualize the output of your k-means on the dataset 2 and comment your results.

- Look above, for the same reason that with the dataset 1. Append on lists are not working, and
- It doesn't look like it finds multiple classes, resulting in a drastic convergence.