part1

April 10, 2021

1 Machine Learning Project 1 (Part 1)

1.1 UNICAMP

1.2 Dataset 1

Load Datasets

[126]:
```
import pandas as pd
import numpy as np

df = pd.read_table("cluster.dat",header=None, sep="\s+")
datos = pd.DataFrame(df)

datos
```

[126]:

	0	1
0	1555.0	28.65
1	1490.0	27.55
2	1445.0	28.35
3	1415.0	28.80
4	1375.0	28.05
..	…	…
568	3420.0	27.95
569	3465.0	26.85
570	3525.0	26.00
571	3570.0	26.15
572	3440.0	25.60

[573 rows x 2 columns]

Transformamos el dataset a un array de nummpy

[127]:
```
datos=datos.to_numpy()
datos
```

[127]:
```
array([[1555.  ,   28.65],
       [1490.  ,   27.55],
       [1445.  ,   28.35],
```

```
       …,
       [3525.  ,   26.  ],
       [3570.  ,   26.15],
       [3440.  ,   25.6 ]])
```

Shuffle data

[128]:
```
np.random.shuffle(datos)
datos
```

[128]:
```
array([[3465.  ,   26.85],
       [1310.  ,   21.3 ],
       [1800.  ,    8.55],
       …,
       [2350.  ,    5.05],
       [1555.  ,    8.2 ],
       [ 535.  ,   19.05]])
```

1.2.1 Split Dataset

Split data in train and test: - 90% train - 10% test

[129]:
```
datos_train = datos[:int(0.9*len(datos))]
datos_test = datos[int(0.9*len(datos)):]

print("Train", datos_train.shape)
print("Test", datos_test.shape)
```

```
Train (515, 2)
Test (58, 2)
```

[130]:
```
x_train = datos_train[:,0]
y_train = datos_train[:,1]
```

1.2.2 Normalization

Min max Scaler

[131]:
```
def normalization(x,y,x_max, x_min , y_max, y_min):
    print("Normalizating using parameters: x", x_max,"\t",x_min)
    print("Normalizating using parameters: y", y_max,"\t",y_min)
    x=(x-x_min)/(x_max-x_min)
    y=(y-y_min)/(y_max-y_min)
    return x,y

x_max = x_train.max()
x_min = x_train.min()
y_max = y_train.max()
```

```
y_min = y_train.min()
```

[132]:
```
x_n_train, y_n_train = normalization(x_train,y_train,x_max,x_min,y_max,y_min)
print("x_train",x_n_train[:3] ,"....")
print("y_train",y_n_train[:3] ,"....")
```

```
Normalizating using parameters: x 3635.0         335.0
Normalizating using parameters: y 29.15          1.95
x_train [0.94848485 0.29545455 0.44393939] …
y_train [0.91544118 0.71139706 0.24264706] …
```

1.2.3 K-Means Function

[133]:
```
import matplotlib.pyplot as plt
def kmeans(k,x,y):


    # Find min and max values
    x_min=np.min(x)
    x_max=np.max(x)
    y_min=np.min(y)
    y_max=np.max(y)

    # print("Min Max X values",x_min,x_max)
    # print("Min Max Y values",y_min,y_max)

    # Select K Random Points
    punto_x=np.random.uniform(low=x_min,high=x_max,size=k)
    punto_y=np.random.uniform(low=y_min,high=y_max,size=k)


    # print("puntos X Random",punto_x)
    # print("punto Y Random",punto_y)

    old_distortion = np.inf
    for it in range(20):
        # Se creará una matriz de distancias
        matriz_distancias = np.zeros((len(x),k))
        # print (matrix.shape)

        for k_i in range(k):

            # Cálculo de Distancia Euclideana
            x_h=(x-punto_x[k_i])**2
            y_h=(y-punto_y[k_i])**2
            dist= np.sqrt(x_h+y_h)
            matriz_distancias[:,k_i]=dist
```

```
# Encontrar la mínima distancia y actualizar la pertenencia en la
# matriz máscara

min=np.argmin(matriz_distancias,axis=1)
matriz_clusters=np.zeros(matriz_distancias.shape)
i=0
for min_i in min:
    matriz_clusters[i][min_i]=1
    i+=1

# Transformar en matriz columna para multiplicar
x_r=x.reshape(-1,1)
y_r=y.reshape(-1,1)

# Calcular la suma de los valores clusters
matriz_clusters_x=x_r*matriz_clusters
matriz_clusters_y=y_r*matriz_clusters

total=matriz_clusters.sum(axis=0)

total_x=matriz_clusters_x.sum(axis=0)
total_y=matriz_clusters_y.sum(axis=0)

# Actualizar puntos con la media total_x
# print("total",total_x, "\n" , total_y, "\n",  total)
punto_x=(total_x/ (total+0.0000000001) )
punto_y=(total_y/ (total+0.0000000001) )

# Lista de clusters ( 0, 1 , 2) Para colorear
clusters = np.argmax(matriz_clusters,axis=1)

centroides_x = np.matmul(matriz_clusters,punto_x)
centroides_y = np.matmul(matriz_clusters,punto_y)


distortion = ( ( x - centroides_x )**2 + ( y - centroides_y )**2 ).sum()




# Distortion No varía
if ( abs( old_distortion - distortion ) <= 0.001 ): break
else: old_distortion = distortion
```

```
    print ( "\n \n DISTORTION =",distortion, "ITERATION=", it)
    plt.scatter(x,y,marker="o",c=clusters)
    plt.scatter(punto_x,punto_y,marker="*",c="red")
    plt.show()

    return punto_x,punto_y,matriz_clusters,distortion
```

[134]:
```
def RMSSTD(x,y,punto_x,punto_y,matriz_clusters,k):

    centroides_x = np.matmul(matriz_clusters,punto_x)
    centroides_y = np.matmul(matriz_clusters,punto_y)


    distortion = ( ( x - centroides_x )**2 + ( y - centroides_y )**2 ).sum()


    number_attributes = 2 * ( len(x) - k )

    return np.sqrt(distortion/number_attributes)
```

1.2.4 Metodo Codo

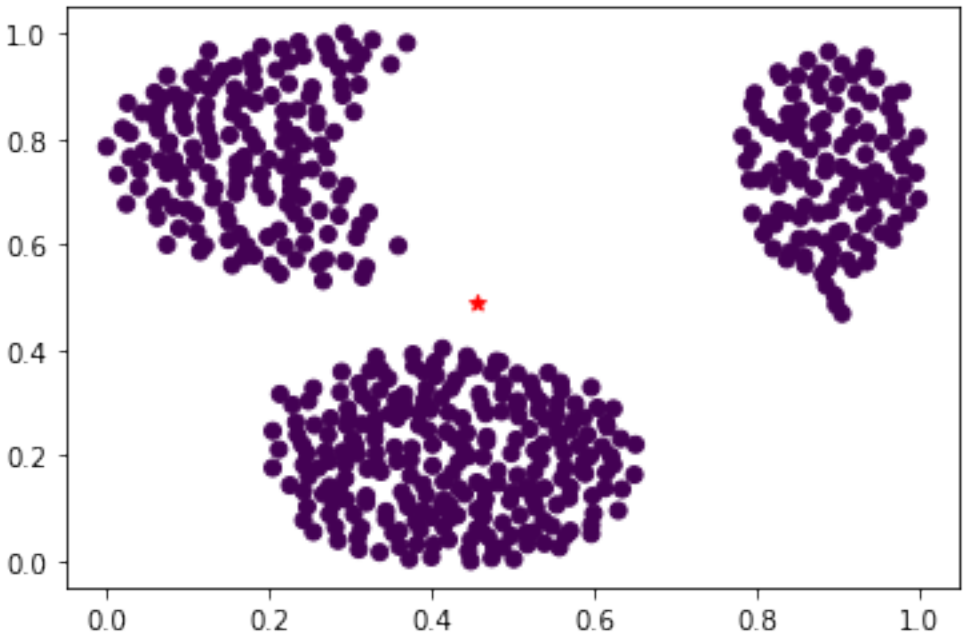
$$5 = 3 + 25$$

$$RMSTDE = \sum_{i=0}^{n} \sqrt{2}$$

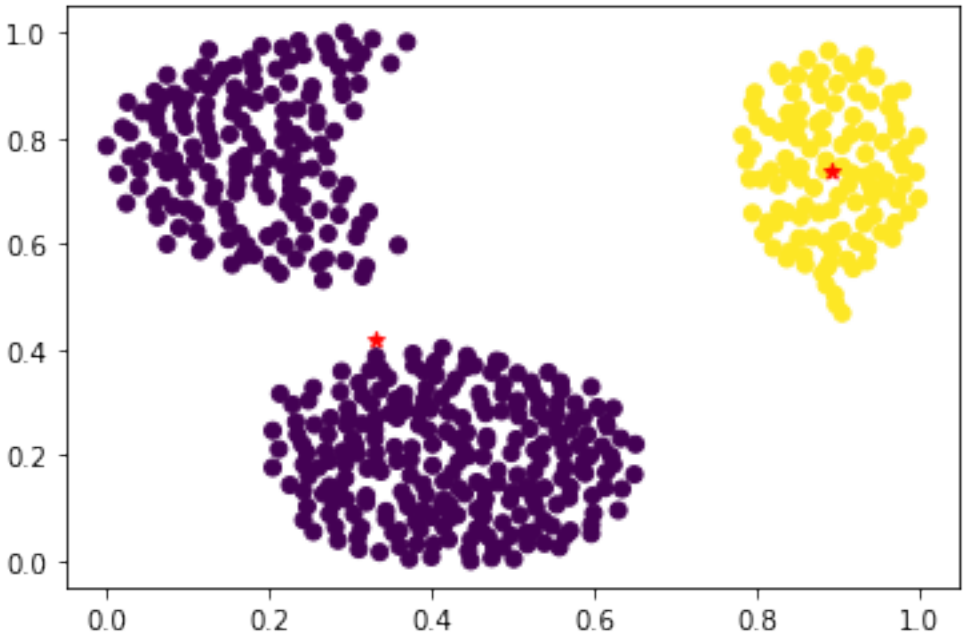
[135]:
```
def metodo_codo(k_max ):
    list_distortions = []
    list_rmmstds = []
    for k in range(1,k_max+1):
        centroide_x , centroide_y , matriz_clusters , distortion =␣
↪kmeans(k,x_n_train,y_n_train)
        rmmstd = RMSSTD(x_n_train,y_n_train,centroide_x,centroide_y,␣
↪matriz_clusters, k)

        list_distortions.append(distortion)
        list_rmmstds.append(rmmstd)

    print("Disttortion")
    plt.plot(range(1,k_max+1),list_distortions,marker="*")
    plt.show()
```

```
    print("Root Mean Squared Standard Deviation")
    plt.plot(range(1,k_max+1),list_rmmstds,marker="o")
    plt.show()



metodo_codo(10)
```

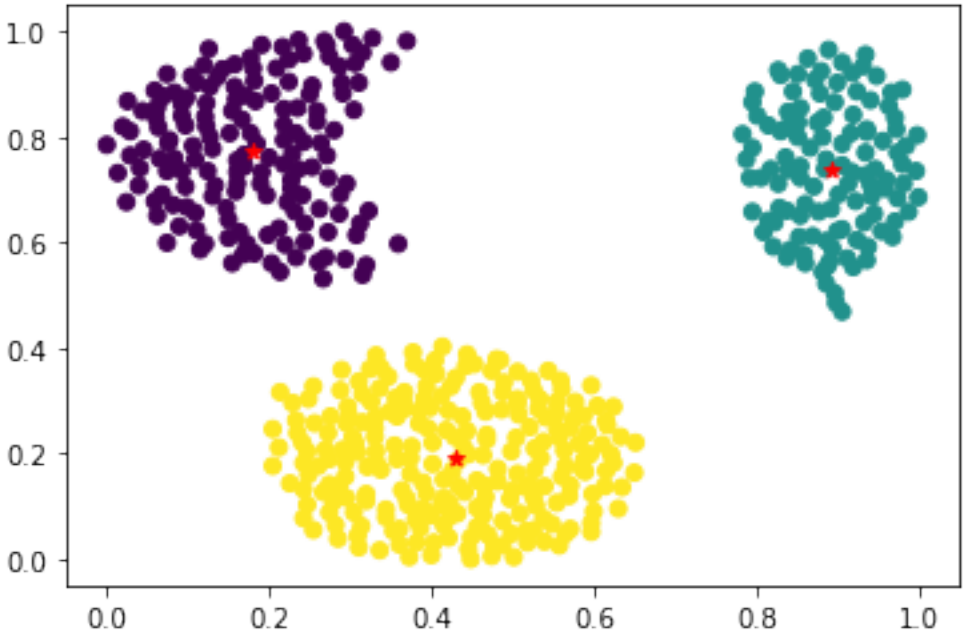
```
DISTORTION = 86.98974058604522 ITERATION= 1
```



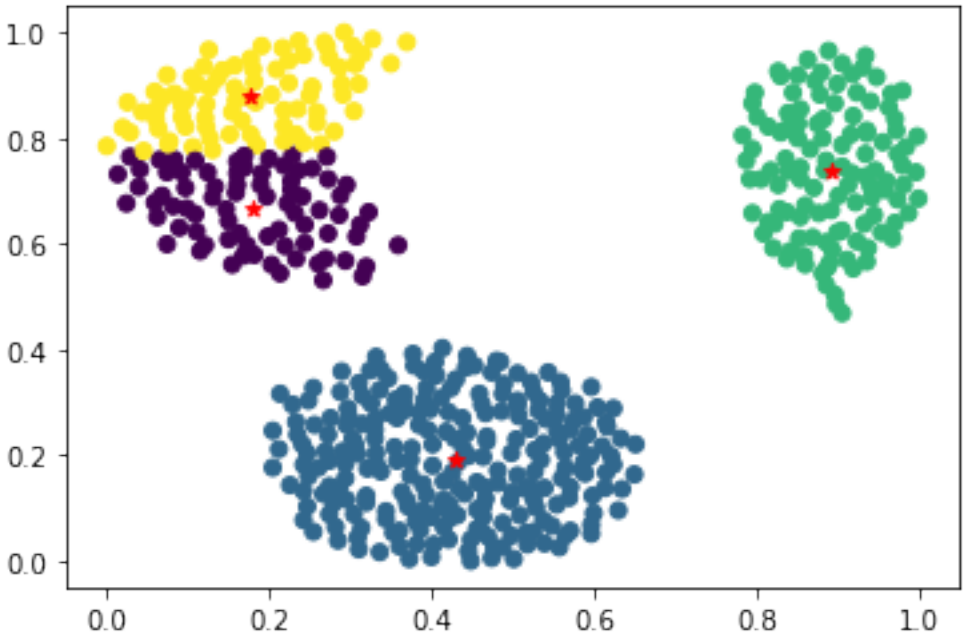

```
DISTORTION = 49.57985268880977 ITERATION= 2
```



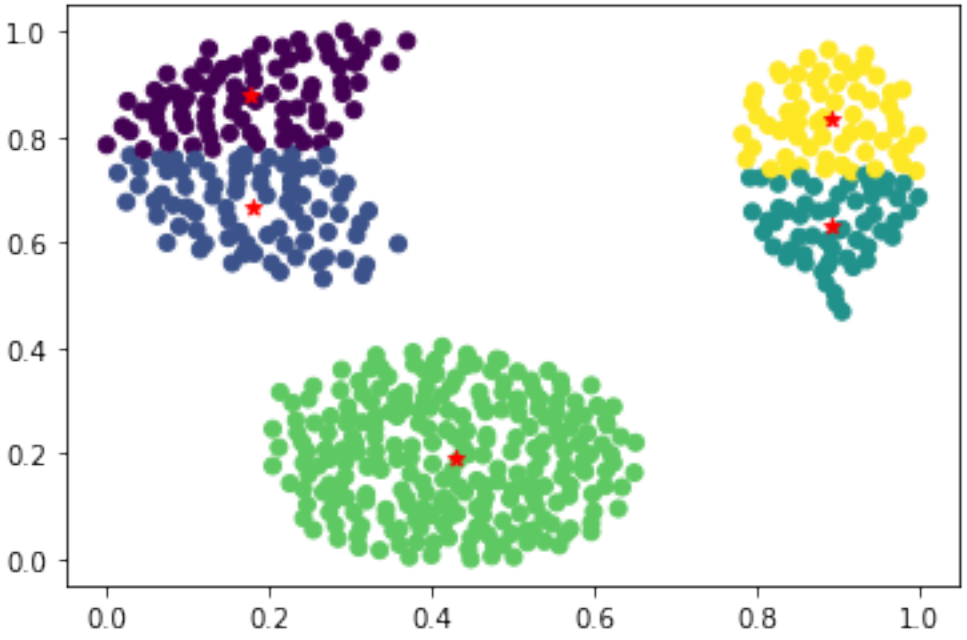
DISTORTION = 11.40346737015686 ITERATION= 3



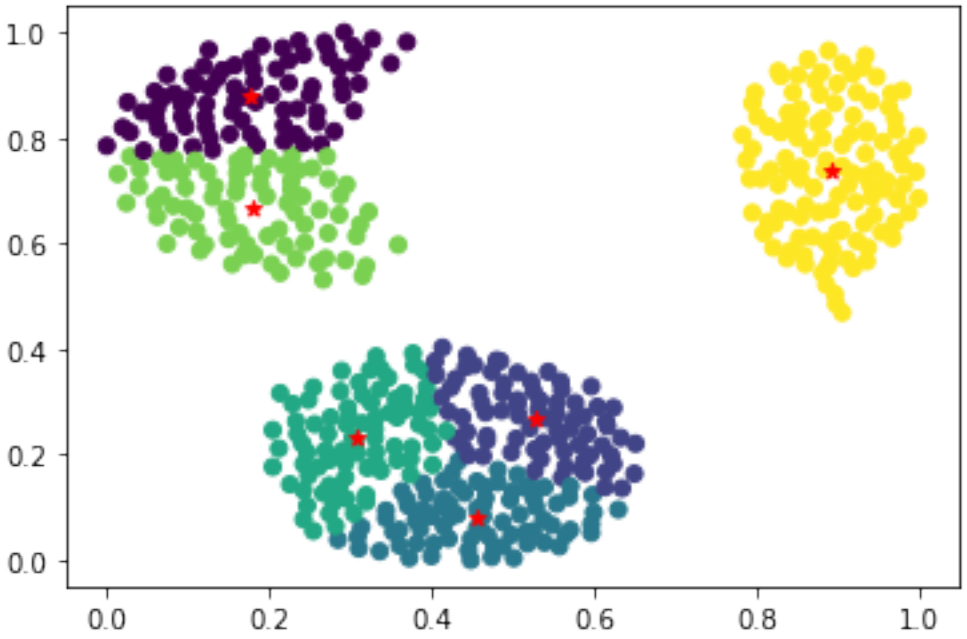
DISTORTION = 9.619651866176516 ITERATION= 8



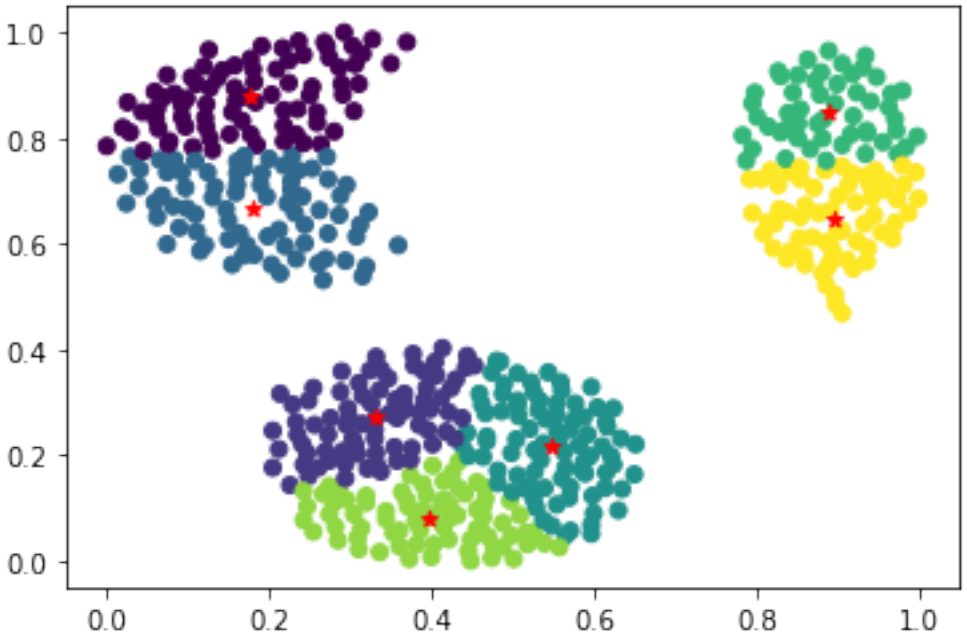
DISTORTION = 8.43251256368104 ITERATION= 9



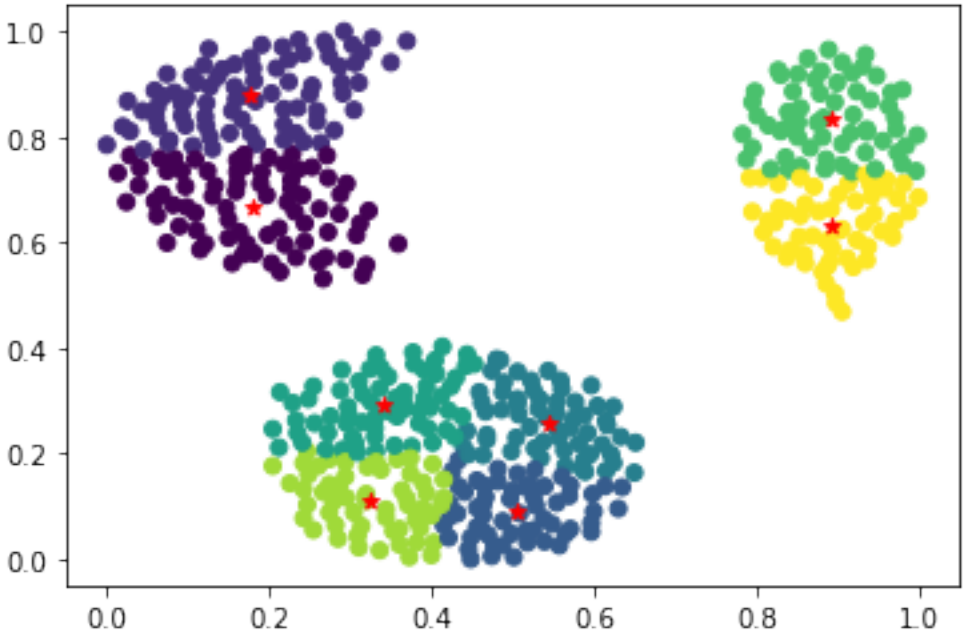
DISTORTION = 6.078351121025027 ITERATION= 6



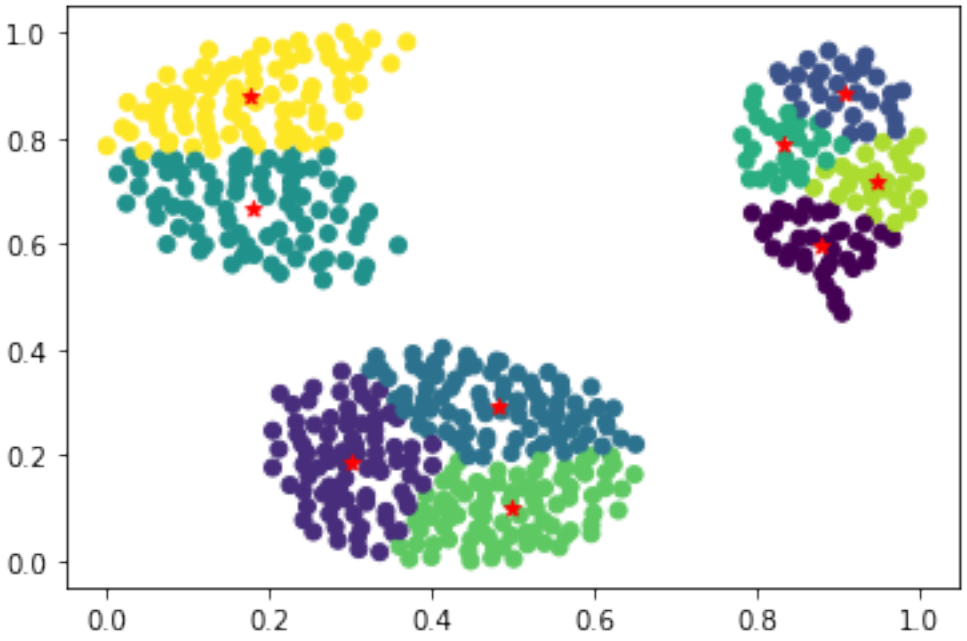
DISTORTION = 4.8948750208396845 ITERATION= 9



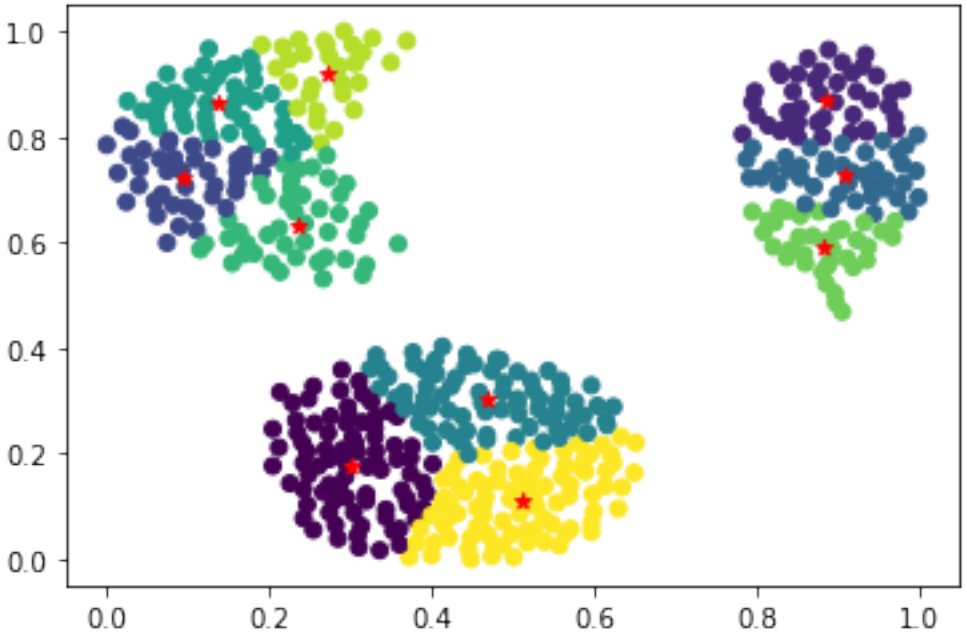
DISTORTION = 4.268918544009868 ITERATION= 9



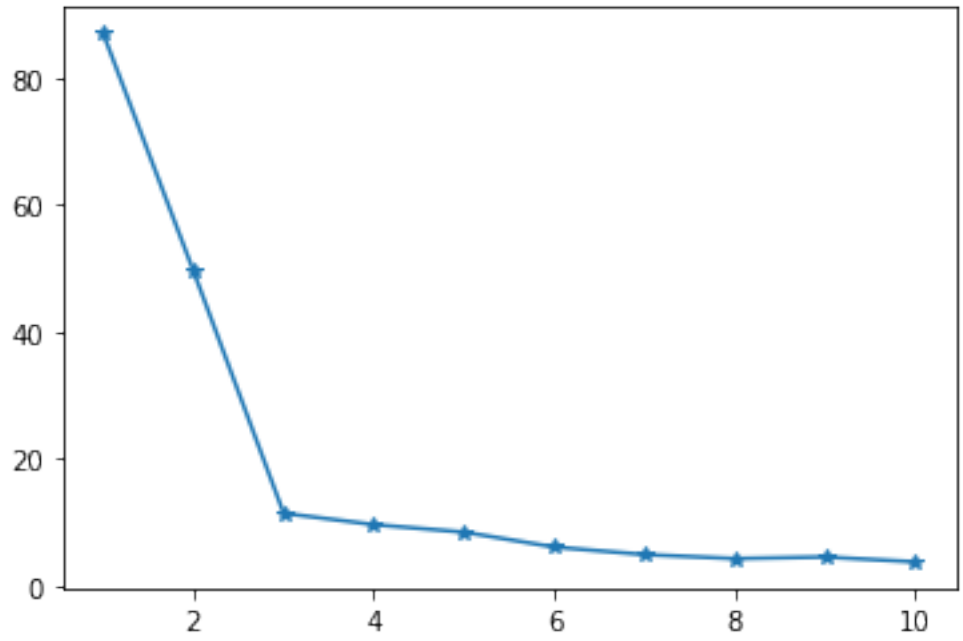
DISTORTION = 4.520507605758721 ITERATION= 11



DISTORTION = 3.7488075630124174 ITERATION= 16



Disttortion



Root Mean Squared Standard Deviation