Assignment 9

K-Means Plus Plus Algorithm

K-Means Plus Plus Algorithm คือ Algorithm ที่ช่วยให้มั่นใจในการเริ่มต้นจุด Centroids พูดง่าย ๆว่าเหมือน K-Means Algorithm แต่ใช้ Algorithm พิเศษช่วยในการหาค่าของ Centroids ตอนแรกเริ่ม

ลำดับวิธีขั้นตอน

- 1.สุ่มเลือก Centroid จาก data points
- 2.คำนวณหาระยะทางของจุดใดๆ บน data points กับ centroid ที่ใกล้ที่สุด
- 3.เลือกจุดที่มีระยะทางสูงสุดจากจุดนั้น กับ centroid ที่ใกล้ที่สุด
- 4.ทำข้อ 2 กับ ข้อ 3 จนกว่าจะได้ centroids ครบทุกจุด

```
di=[]
                        dindIntializer Algorithm:
Input: Number of Centroids
Output:Collection of centroids
Centroids<--{}
C_1 <-- random first centroid among data points
For C_i in range 1 to K:
        di<--[] #List of distance of each point with nearest centroid
        for i in range number of centroids:
                all min d ist ance<--[]
                for c in centroids:
                        d=distance(c,datapoints[i])
                        all min distance.appenmin=find_min(all min distance)
                di.append(min)
        next_c <-- ntroid=datapoints[index of maximun element in di]
        Centroi <-- s[C_i]=next_centr <-- id
```

```
def initial K mean plus(self,K):
  centroids={}
 C 1=random.sample(self.points,1)
  centroids[0]=C 1[0]
  for C i in range(1,K):
   di=[]
    for i in range(self.size):
      k=i
      d=sys.maxsize
      for c in centroids:
        v=centroids[c]
        u=self.points[k]
        dist=(u[0]-v[0])**2+(u[1]-v[1])**2
        d=min(d,dist)
      di.append(d)
    next C=self.points[di.index(max(di))]
   centroids[C_i]=next_C
   di=[]
  return centroids
```

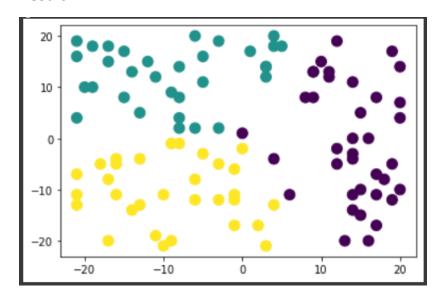
```
import numpy as np
import random
import matplotlib.pyplot as plt
import matplotlib
import sys
#Euclidean Distance
class K_means():
    def __init__(self,K,n):
        self.size=n
        self.points=self.R(n)
        #self.points=[[1,1],[1.5,2.0],[3,4],[5,7],[3.5,5],[4.5,5],[3.5,4.5]]
        #self.centroids=self.ran_k(K)
        self.centroids=self.initial_K_mean_plus(K)
        #self.centroids={0:[1,1],1:[1.5,2],2:[3,4]}
        self.clusters={key: [] for key in range(K)}
```

```
def clustering(self):
 A=self.clusters
  self.clusters={key: [] for key in range(self.K)}
  for i in range(self.size):
   B={}
   k=i
   for j in self.centroids:
     u=self.points[k]
     v=self.centroids[j]
     x=(u[0]-v[0])**2+(u[1]-v[1])**2
     B[j]=x
   min_key=min(B.keys(), key=(lambda k: B[k]))
   self.clusters[min key].append(self.points[i])
 #print(f"After={self.clusters}")
def update_cent(self,iter):
 for j in range(iter):
   self.clustering()
   for i in self.clusters:
        A=self.centroids[i]
        coor=np.sum(self.clusters[i], axis=0)/len(self.clusters[i])
        self.centroids[i]=coor
```

```
B=K_means(3,100)
x=[B.points[i][0] for i in range(100)]
y=[B.points[i][1] for i in range(100)]
print(B.points)
print(len(x))
print(len(y))
z=[]
```

```
B.update_cent(2000)
    C=B.clusters[0]+B.clusters[1]+B.clusters[2]
    print(B.clusters[0])
    print(B.clusters[1])
    print(B.clusters[2])
    print(C)
    a=[C[i][0] for i in range(len(C))]
    b=[C[i][1] for i in range(len(C))]
    print(a)
    print(b)
len(B.clusters)
    print(B.clusters)
    z_1=[1 for i in range(len(B.clusters[0]))]
    print(len(z_1))
    z_2=[2 for i in range(len(B.clusters[1]))]
    z 3=[3 for i in range(len(B.clusters[2]))]
    print(len(z_2))
    print(len(z_3))
    z=z_1+z_2+z_3
    print(z)
    plt.scatter(a,b,s=100,c=z)
    plt.show()
```

Result



Panot Srinakkarin 645020096-3