Machine Learning HW

Team Member:
0516017 李柏毅
0516059 劉嘉豪
0516306 尤健羽
0516319 傅信瑀
0516322 朱蝶

1. What environments the members are using:

Use Jupyter as the environment

2. K-means code and result:

*Feature : X , Y Target : pitch_type

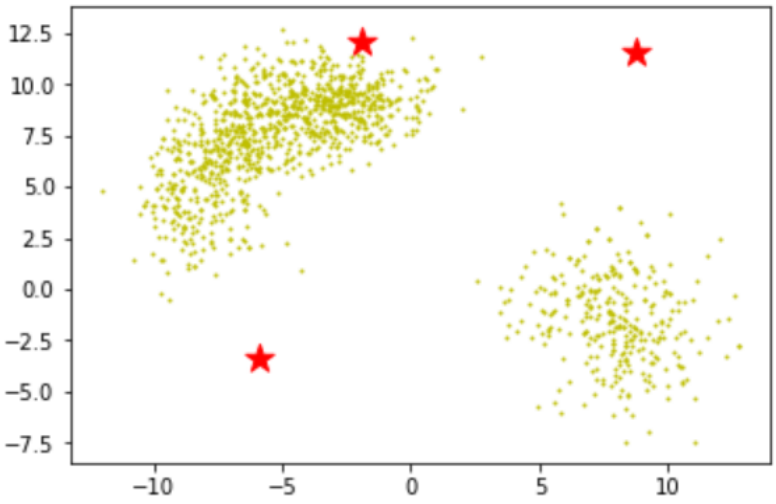
```
%matplotlib inline
from copy import deepcopy
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import random
```

```
#read data
source = pd.read_csv('data_noah.csv')
data = source[['x','y','pitch_type']]
data = data.replace("CH","0")
data = data.replace("FF","1")
data = data.replace("CU","2")
```

```
#select feature
f1 = data['x'].values
f2 = data['y'].values
X_ori = data.values
X = X_ori[:,0:2]
X = X.astype(float)
target = X_ori[:,2]
target = target.astype(float)
k=3
```

```
#choose three random points
x_cen = [];
for i in range (3):
    x_cen.append(random.uniform(np.min(X[:,0]), np.max(X[:,0])))
y_cen = []
for i in range (3):
    y_cen.append(random.uniform(np.min(X[:,1]), np.max(X[:,1])))
print("x : ", x_cen)
print("y : ", y_cen)
C = np.dstack((x_cen,y_cen))
C = C[0]
plt.scatter(f1, f2, c='y', s=1)
plt.scatter(x_cen, y_cen, marker='*', s=200, c='r')
C_arg = np.argsort(C[:,0])
C = C[C_arg]
```

```
x :  [8.772388334549944, -1.92753183025558, -5.864938578235353]
y :  [11.593149728113907, 12.039029544109336, -3.421305046088942]
```

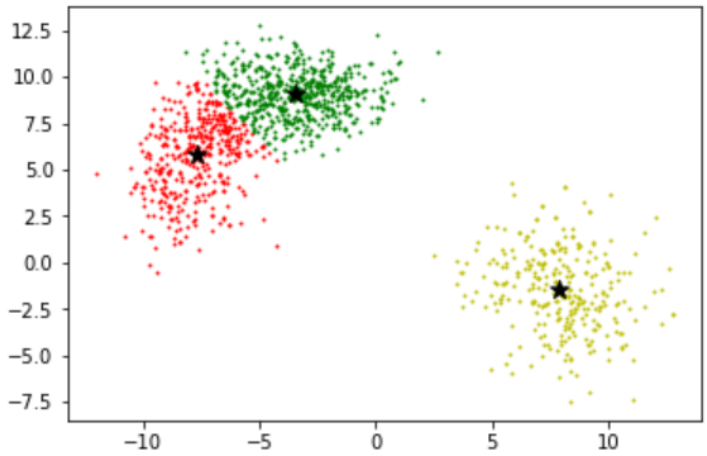


```
#repeat calculating the means of each cluster until it converges
pitch_type = np.zeros(len(X))
C_origin = np.zeros(C.shape)
f=0
while np.linalg.norm(C - C_origin, None) !=0:
    f=f+1
    cost=0
    for i in range(len(X)):
        dis = np.linalg.norm(X[i] - C, axis=1)
        pitch_type[i] = np.where(dis==np.min(dis))[0][0]
    C_origin = C.copy()
    for i in range(k):
        points = [X[j] for j in range(len(X)) if pitch_type[j] == i]
        C[i] = np.mean(points, axis=0)
    C_arg = np.argsort(C[:,0])
    C = C[C_arg]
```

```
####cost function
    for i in range(k):
        points2 = np.array([X[j] for j in range(len(X)) if pitch_type[j] == i])
        x_distance = points2[:,0] - C[i,0]
        x_distance_power = np.power(x_distance,2)
        y_distance = points2[:,1] - C[i,1]
        y_distance_power = np.power(y_distance,2)
        distance = np.sum(x_distance_power) + np.sum(y_distance_power)
        costi = distance / points2[:,0].size
        cost += costi
    print('%2d' %f," cost = ",round(cost,6))
```

```
#draw the plot
colors = ['r', 'g', 'y']
fig, ax = plt.subplots()
for i in range(k):
        points = np.array([X[j] for j in range(len(X)) if pitch_type[j] == i])
        ax.scatter(points[:, 0], points[:, 1], s=1, c=colors[i])
ax.scatter(C[:, 0], C[:, 1], marker='*', s=100, c='black')
```

```
<matplotlib.collections.PathCollection at 0x17e5cd80588>
```



3. Cost function and accuracy:

$$COST = \sum_{i=1}^{10} \frac{1}{|C_i|} \sum_{x \in C_i} \|x - C_i\|^2$$

Calculate every distance between every point and their own center, and sum them which are in the same cluster, then divide the cluster's size.

```
####cost function
    for i in range(k):
        points2 = np.array([X[j] for j in range(len(X)) if pitch_type[j] == i])
        x_distance = points2[:,0] - C[i,0]
        x_distance_power = np.power(x_distance,2)
        y_distance = points2[:,1] - C[i,1]
        y_distance_power = np.power(y_distance,2)
        distance = np.sum(x_distance_power) + np.sum(y_distance_power)
        costi = distance / points2[:,0].size
        cost += costi
    print('%2d' %f," cost = ",round(cost,6))
```

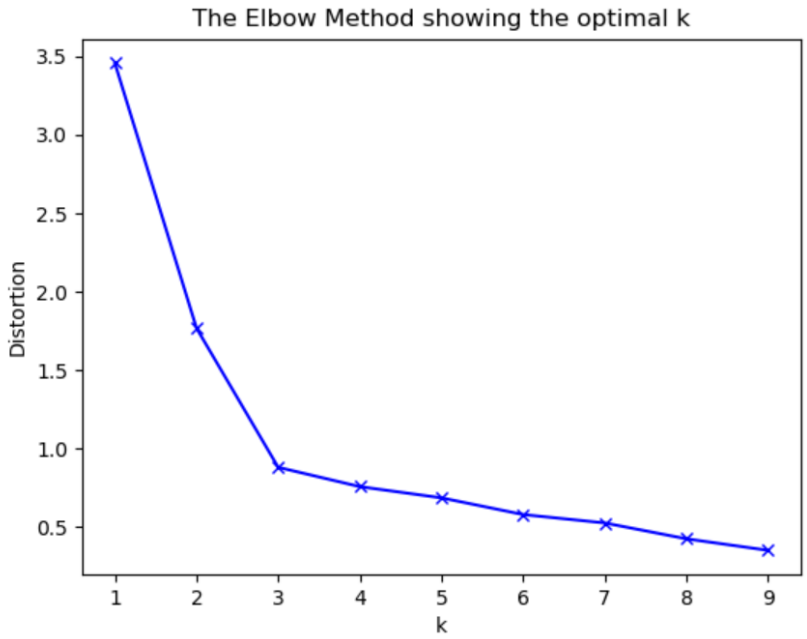
```
 1  cost =  217.450046
 2  cost =  111.628087
 3  cost =  19.12643
 4  cost =  18.692223
 5  cost =  18.593864
 6  cost =  18.521015
 7  cost =  18.515944
 8  cost =  18.516584
 9  cost =  18.510021
10  cost =  18.512433
11  cost =  18.512433
```

```
#accuracy: check the difference between predicted result and target
wrong=0
for i in range(len(target)):
    if target[i]!=pitch_type[i]:
        wrong = wrong+1
accuracy = 1-(wrong/len(target))
print("accuracy =", accuracy)
```

```
accuracy = 0.8001514004542014
```

4. Why choose k=3:

The technique to determine k called the elbow method.



We can see that the improvement will decline, at some point rapidly, creating the elbow shape. That point is the optimal value, and it is 3.

5. Another two attributes to partition:

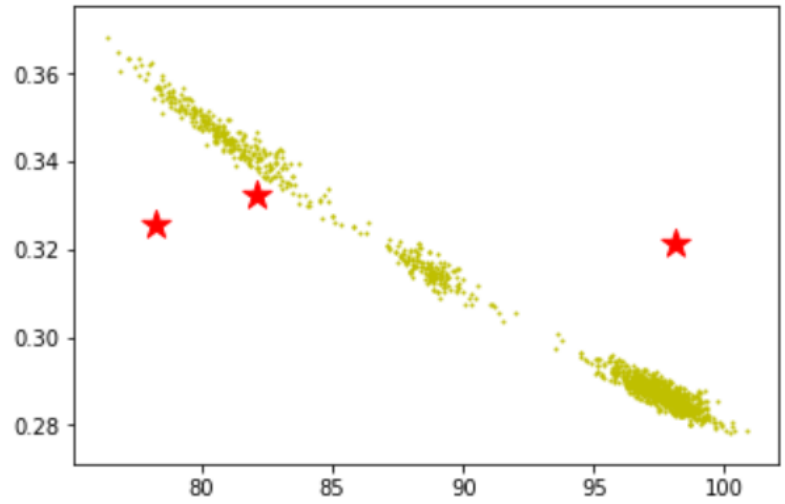
*Feature: ftime, speed Target: pitch_type

Only screenshot the code differ from above

```
#select features
f1 = data['speed'].values
f2 = data['ftime'].values
X_ori = data.values
X = X_ori[:,0:2]
X = X.astype(float)
target = X_ori[:,2]
target = target.astype(float)
k=3
```

```
#choose three random points
x_cen = [];
for i in range (3):
    x_cen.append(random.uniform(np.min(X[:,0]), np.max(X[:,0])))
y_cen = []
for i in range (3):
    y_cen.append(random.uniform(np.min(X[:,1]), np.max(X[:,1])))
print("x : ", x_cen)
print("y : ", y_cen)
C = np.dstack((x_cen,y_cen))
C = C[0]
plt.scatter(f1, f2, c='y', s=1)
plt.scatter(x_cen, y_cen, marker='*', s=200, c='r')
C_arg = np.argsort(C[:,0])
C = C[C_arg]
```

```
x :  [98.13424189440241, 82.1190603775276, 78.25840281449148]
y :  [0.3211879839606597, 0.33247955238350235, 0.32581194351288767]
```



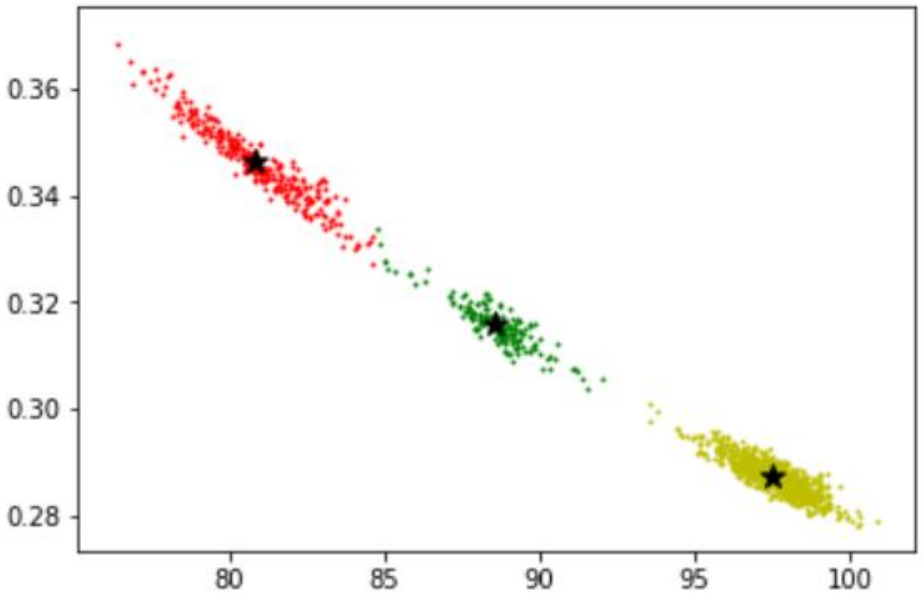
*Cost function result and accuracy:

```
1  cost =  14.530882
2  cost =  10.554894
3  cost =  6.294323
4  cost =  5.223014
5  cost =  5.089737
6  cost =  5.047699
7  cost =  5.047699
```

```
wrong=0
for i in range(len(target)):
    if target[i]!=pitch_type[i]:
        wrong = wrong+1
accuracy = 1-(wrong/len(target))
print("accuracy =", accuracy)
```

```
accuracy = 0.9947009841029523
```

*Result:

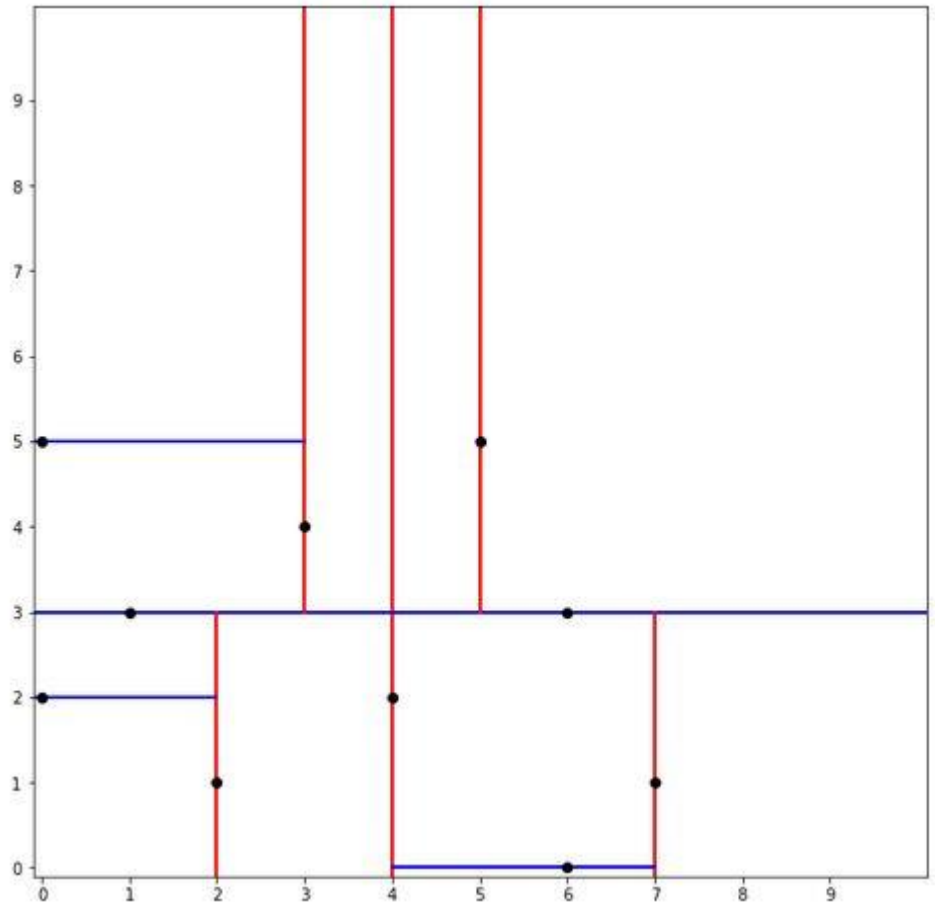


6. KD-tree code

Divide into two parts, and do it recursively, we use median to be divided point.

```
class Node(namedtuple('Node', 'location left_child right_child')):
    def __repr__(self):
        return pformat(tuple(self))
def kdtree(point_list, depth=0):
    try:
        k = len(point_list[0])
    except IndexError:
        return None
    axis = depth % k
    point_list.sort(key=itemgetter(axis))
    median = len(point_list) // 2
    return Node(
        location=point_list[median],
        left_child=kdtree(point_list[:median], depth + 1),
        right_child=kdtree(point_list[median + 1:], depth + 1)
    )
```

7. Result of KD-tree



END