Machine Learning Program Assignment #2

Team ID: 8

Team members:

0516037 朱俐瑄 0516212 吳子涵 0516218 軒轅照雯 0516239 王盈筑

1. What environments the members are using:

朱俐瑄:

Environment: Linux (Ubuntu)

Programming Language: Python 2.7

王盈筑:

Environment: Windows10

Programming Language: Python 3.7

吳子涵:

Environment: macOS

Programming Language: Python 3

軒轅照雯:

Environment: macOS

Programming Language: Python 2.7

2. K-means code:

```
get_ipython().run_line_magic('matplotlib', 'inline')

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from collections import Counter
from copy import deepcopy

df=pd.read_csv('data_noah.csv')
# Change "pitch_type" data to number 0-2
#0:CH,1:CU,2:FF
df["pitch_type"] = pd.Categorical(df["pitch_type"])
df["pitch_type"] = df["pitch_type"].cat.codes
#print(df["pitch_type"])
data=df.values[:,13:15]
category=df.values[:,15]
# Number of clusters
k=3
# Number of training data
#The shape attribute for numpy arrays returns the dimensions of the array.
#If Y has n rows and m columns, then Y.shape is (n,m). So Y.shape[0] is n.
n=data.shape[0]
# Number of features in the data
c=data.shape[1]
# Generate random centers, use sigma and mean to ensure it represent the whole data
mean = np.mean(data, axis = 0)
std = np.std(data, axis = 0, dtype = np.float64)
centers = np.random.randn(k,c)*std + mean
# Plot the data and the random generated centers
plt.figure()
#orange0:CH, blue1:CU, red2:FF
colors=['orange', 'blue', 'red']
```

```
################

clusters_old = deepcopy(category)
clusters_new = np.zeros(n)
distances = np.zeros((n,k))
data = data.astype(float)
centers = centers.astype(float)
centers_new = deepcopy(centers) # store new centers

error=n

# When the estimate of the center stays the same after an update, exit loop
while error!=0:
    # Calculate the distance to every center
    for i in range(k):
        distances[:,i] = np.linalg.norm(data - centers_new[i], axis=1)
    # Assign all training data to the closest center
    clusters_new = np.argmin(distances, axis = 1)
    # update the center
    for i in range(k):
        centers_new[i] = np.mean(data[clusters_new == i], axis=0)
    error=0
    for i in range(n):
        if clusters_new[i]!=clusters_old[i]:
            error=error+1

    clusters_old=deepcopy(clusters_new)
```

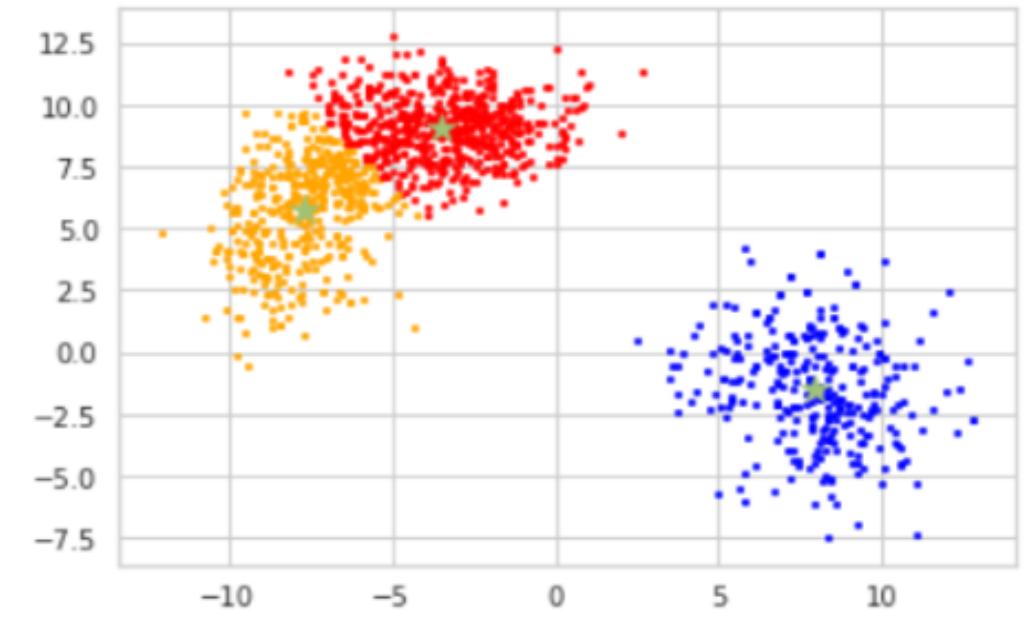
3. Cost function and accuracy

```
################cost function#################

#orange0:CH(xmin), blue1:CU(xmax ), red2:FF(ymax)
x_y_max=np.argmax(centers_new, axis = 0)   #x_y_max[0]=xmax,x_y_max[1]=ymax
x_min=np.argmin(centers_new, axis = 0)   #xmin[0]
temp0=(clusters_new==x_y_max[0]) #index of 右下cluster=cu
temp1=(clusters_new==x_y_max[1])  #index of 中高cluster=ff
temp2=(clusters_new==x_min[0])     #index of 左中cluster=ch
clusters_new[temp0]=1
clusters_new[temp1]=2
clusters_new[temp2]=0
match=0
for i in range(n):
    if clusters_new[i]==category[i]:
        match=match+1
print('accuracy:', float(match)/float(n))
```

```
('accuracy:', 0.8001514004542014)
```

4. The result of K-Means clustering:

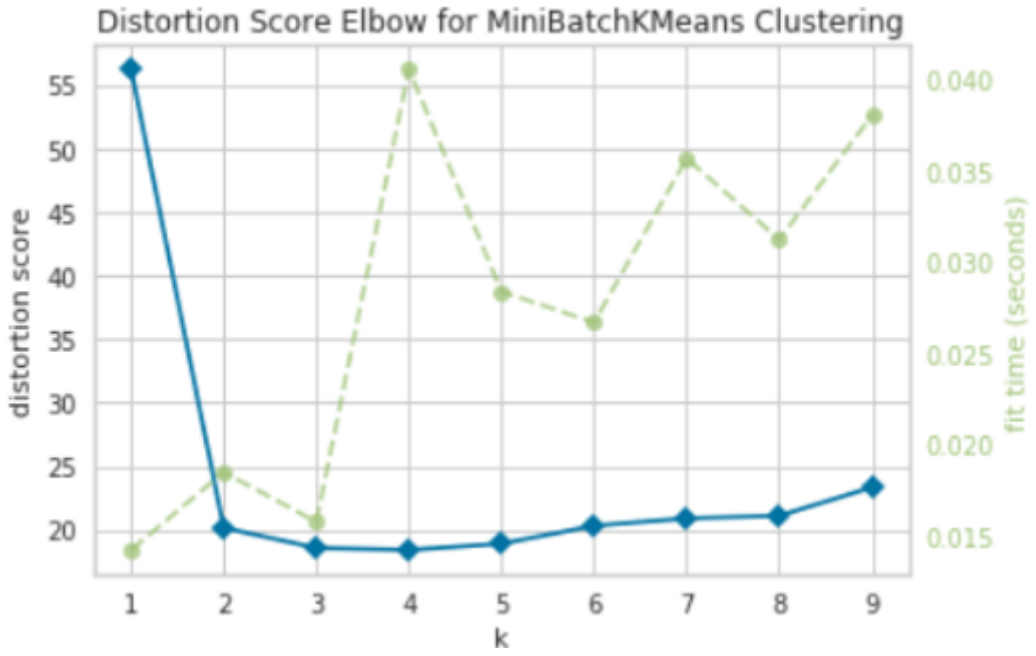
```
################plot#################

plt.figure()
for i in range(n):
    plt.scatter(data[i, 0], data[i,1], s=7, color = colors[int(clusters_new[i])])
plt.scatter(centers_new[:,0], centers_new[:,1], marker='*', c='g', s=150)
plt.show()
```



5. Try to explain why k = 3 is the best:

We use the elbow method to select the optimal number of clusters (k) for K-means clustering.The elbow method runs k-means clustering on the dataset for a range of values for k (in our case from 1~10), and then for each value of k, it computes an average score for all clusters (in our case, the distortion_score is computed, which is the sum of square distances from each point to its assigned center). When these overall metrics for each model are plotted, it is possible to visually determine the best value for k. We select the k while d(distortion score)/d(k) is less than a small number, from the elbow method figure we plotted, we can see that k=3 is the best number of clusters for K-means clustering in our case.

```
##########################################

from sklearn.cluster import MiniBatchKMeans
from yellowbrick.cluster import KElbowVisualizer
plt.figure()
# Instantiate the clustering model and visualizer
visualizer = KElbowVisualizer(MiniBatchKMeans(), k=(1,10))
visualizer.fit(data)
visualizer.poof() # Draw/show/poof the data
plt.show()
```



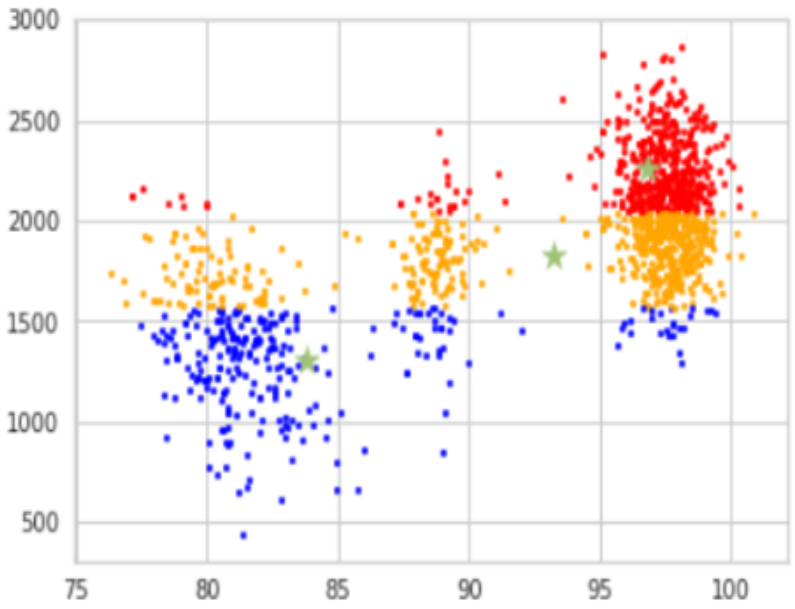
6. Use another two or more attributes to partition

```
######use speed and tm_spin to partition#######

data1=df.values[:,44] #speed
data1_ = data1[:, np.newaxis] #將data1的資料轉成2D模式
data2=df.values[:,49] #tm_spin
data2_ = data2[:, np.newaxis]
data_=np.concatenate((data1_, data2_), axis=1)
#print(data_)
k_=3
# Number of training data
n_=data_.shape[0]
# Number of features in the data
c_=data_.shape[1]
mean_ = np.mean(data_, axis = 0)
std_ = np.std(data_, axis = 0, dtype = np.float64)
centers_ = np.random.randn(k_,c_)*std_ + mean_
```

```
###########
data_ = data_.astype(float)
centers_ = centers_.astype(float)
centers_new_ = deepcopy(centers_)
clusters_old_ = deepcopy(category)
clusters_new_ = np.zeros(n_)
distances_ = np.zeros((n_,k_))

error_ =n_
while error_ !=0:
    for i in range(k_):
        distances_[:,i] = np.linalg.norm(data_ - centers_new_[i], axis=1)
    # Assign all training data to the closest center
    clusters_new_ = np.argmin(distances_, axis = 1)
    # Calculate mean for every cluster and update the center
    for i in range(k_):
        centers_new_[i] = np.mean(data_[clusters_new_ == i], axis=0)
    error_=0
    for i in range(n_):
        if clusters_new_[i]!=clusters_old_[i]:
            error_=error_+1
    clusters_old_=deepcopy(clusters_new_)
```

```
###############cost function##############

#orange0:CH(xmid), blue1:CU(xmin ), red2:FF(xmax)
x_y_max=np.argmax(centers_new, axis = 0)   #x_y_max[0]=xmax
x_min=np.argmin(centers_new, axis = 0)   #xmin[0]
temp0=(clusters_new==x_y_max[0]) #index of 右cluster=ff
temp1=(clusters_new==x_min[0])     #index of 左cluster=cu
#temp2=(clusters_new!=x_y_max[0] & clusters_new!=x_min[0])  #index of 中高cluster=ch
temp2=(temp0 | temp1)
for i in range(n):
    temp2[i]=~temp2[i]

temp2=np.array(temp2)
clusters_new[temp0]=2
clusters_new[temp1]=1
clusters_new[temp2]=0
match=0
for i in range(n):
    if clusters_new_[i]==category[i]:
        match=match+1
print('accuracy:', float(match)/float(n))

##################plot##################
plt.figure()
for i in range(n_):
    plt.scatter(data_[i, 0], data_[i,1], s=7, color = colors[int(clusters_new_[i])])
plt.scatter(centers_new_[:,0], centers_new_[:,1], marker='*', c='g', s=150)
plt.show()
```

```
('accuracy:', 0.5844057532172596)
```



7. Kd-tree code

```
####################kdtree#########################
from collections import namedtuple
from operator import itemgetter
from pprint import pformat
import matplotlib.pyplot as plt
import numpy as np

get_ipython().run_line_magic('matplotlib', 'inline')
class Node(namedtuple('Node', 'location left_child right_child')):

    def __repr__(self):
       return pformat(tuple(self))

def kdtree(point_list, depth=0):
    global var

    try:
        k = len(point_list[0])
    except IndexError:
        return None

    axis = (depth + var) % 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)
    )
```

```
def plot_tree(tree, min_x, max_x, min_y, max_y, prev_node, branch, depth=0):
    global var
    line_width = 0.5
    cur_node = tree.location
    left_branch = tree.left_child
    right_branch = tree.right_child

    k = len(cur_node)
    axis = (depth + var) % k

    if axis == 0:
        if branch is not None and prev_node is not None:
            if branch:
                max_y = prev_node[1]
            else:
                min_y = prev_node[1]

        plt.plot([cur_node[0],cur_node[0]], [min_y,max_y], linestyle='-', color='red', linewidth=line_width)


    elif axis == 1:
        if branch is not None and prev_node is not None:
            if branch:
                max_x = prev_node[0]
            else:
                min_x = prev_node[0]

        plt.plot([min_x,max_x], [cur_node[1],cur_node[1]], linestyle='-', color='blue', linewidth=line_width)

    plt.plot(cur_node[0], cur_node[1], 'k.')

    if left_branch is not None:
        plot_tree(left_branch, min_x, max_x, min_y, max_y, cur_node, True, depth+1)

    if right_branch is not None:
        plot_tree(right_branch, min_x, max_x, min_y, max_y, cur_node, False, depth+1)
```

```
f = open('points.txt','r')
p=[]

for line in f:
    line=line.strip()
    p.append(list(map(int,line.split(' '))))

x = np.asarray([p[i][0] for i in range(len(p))])
y = np.asarray([p[i][1] for i in range(len(p))])

var = 0 if np.std(x) > np.std(y) else 1

tree = kdtree(p)
x_min = p[0][0]
y_min = p[0][1]
x_max = p[0][0]
y_max = p[0][1]

for i in range(len(p)):
    if p[i][0] < x_min :
        x_min = p[i][0]
    if p[i][1] < y_min :
        y_min = p[i][1]
    if p[i][0] > x_max :
        x_max = p[i][0]
    if p[i][1] > y_max :
        y_max = p[i][1]

delta=1
plt.axis( [x_min-delta, x_max+delta, y_min-delta, y_max+delta] )

#plt.grid(b=True, which='major', color='0.75', linestyle='--')
plt.xticks([i for i in range(x_min-delta, x_max+delta, 1)])
plt.yticks([i for i in range(y_min-delta, y_max+delta, 1)])

# draw the tree
plot_tree(tree, x_min-delta, x_max+delta, y_min-delta, y_max+delta, None, None)

plt.title('KD Tree')
plt.show()
plt.close()
```

8. The result of Kd-tree