# Introduction to Machine Learning Assignment #2

Team ID: 15
Team member: 0416004 郭羽喬 0416208 黃士軒 0416025 呂翊愷 0416022 楊旻學

## 1. What environments the members are using

NCHC-TWGC Machine Learning Cloud service with V100 GPU

## 2. K-means code

```
cluster_center = random.sample(noah_data, 3)
```

First, we randomly choose 3 data as cluster centroids.

```
#found cluster for each data
for row in noah_data:
    min = 999999
    index = 0
    cluster_index = -1
    for i in range(k_cluster):
        center = cluster_center[i]
        square_sum = 0.0
        for j in range(attr_num):
            dist = float(row[j]) - float(center[j])
            square_sum += dist*dist
        hypot =  math.sqrt(square_sum)
        if(hypot< min):
            min = hypot
            cluster_index = i
```

To find the nearest centroids of each point, we calculate the Euclidean distance between point and cluster center. The distance is calculated by the square root of the sum of the square of the difference of two points. If the distance is the minimum between point and cluster center, we assign the point to the cluster.

```
# define the center by calculate center of each axis in each cluster
for cluster_id in range(k_cluster):
    axis_sum = [0.0,0.0]
    for single_data in cluster[cluster_id]:
        for attr_id in range(attr_num):
            axis_sum[attr_id] += float(single_data[attr_id])
    for attr_id in range(attr_num):
        new_cluster_center[cluster_id][attr_id] =float(axis_sum[attr_id]/len(cluster[cluster_id]))
```

For each cluster, we calculate the mean of all points assigned to the cluster, and the mean will be seemed as the new cluster center coordinates. We repeat the previous two steps, which are respectively assigning point to cluster and updating new center for each cluster, until none of the cluster assignments change. So far, we have the final cluster found by K-means.

Besides, we also implement the version of K = 5, which has 5 cluster, to test the performance between different k.

## 3. Cost function and accuracy

```
cluster[cluster_index].append(row)
if row[len(row)-1] == 'FF' :
    cluster_cnt[cluster_index][0]+=1
elif row[len(row)-1] == 'CU' :
    cluster_cnt[cluster_index][1]+=1
elif row[len(row)-1] == 'CH' :
    cluster_cnt[cluster_index][2]+=1
```

```
# plot the scatter
fig = plt.figure(figsize=(8,6))
purity = [0.0,0.0,0.0]

for cluster_id in range(k_cluster):
    purity[cluster_id] = max(cluster_cnt[cluster_id])/sum(cluster_cnt[cluster_id])
    if purity[cluster_id] > 0.9:
        label = k_cluster_[cluster_cnt[cluster_id].index(max(cluster_cnt[cluster_id]))]
    else:
        label = 'CH'
        purity[cluster_id] = 1 - purity[cluster_id]
    total_label[cluster_id] = label
    plt.scatter(cluster[cluster_id][:,0].astype(np.float32),cluster[cluster_id][:,1].ast

cluster_center = np.array(cluster_center)
plt.scatter(cluster_center[:,0].astype(np.float32),cluster_center[:,1].astype(np.float32
plt.legend()
plt.show()
```

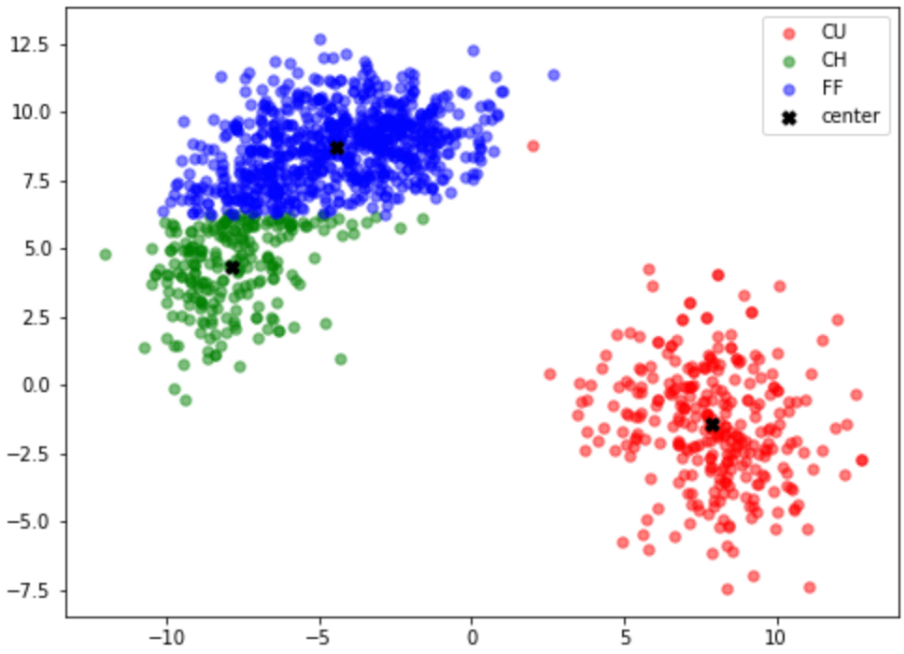
A useful approach to measure our clustering performance is to calculate purity. To compute purity, we count the number of correctly assigned data in each cluster first, and the number will be divided by the number of the total of correctly assigned data. If it has a purity of 1, it means that all of the instances in that cluster are the same label. The higher the purity is, the better K-means performance is. To show the result of K-means clustering, we draw a figure by `matplotlib.pyplot`.

```
#compute accuracy
score = 0
for row in noah_data:
    min = 999999
    index = 0
    cluster_index = -1
    for i in range(k_cluster):
        center = cluster_center[i]
        square_sum = 0.0
        for j in range(attr_num):
            dist = float(row[j]) - float(center[j])
            square_sum += dist*dist
        hypot =  math.sqrt(square_sum)
        if(hypot< min):
            min = hypot
            cluster_index = i
    if row[len(row)-1]==total_label[cluster_index] :
        score+=1

accuracy = score/len(noah_data)
```

In order to compute accuracy, we calculate the distance between each point and 3 cluster centers found by K-means, and we assigned the point to the nearest cluster. If the nearest cluster is same as the pitch type of the original dataset, it indicates that the clustering is accurate. Then, we get the accuracy by dividing the times of accuracy into the number of total data.

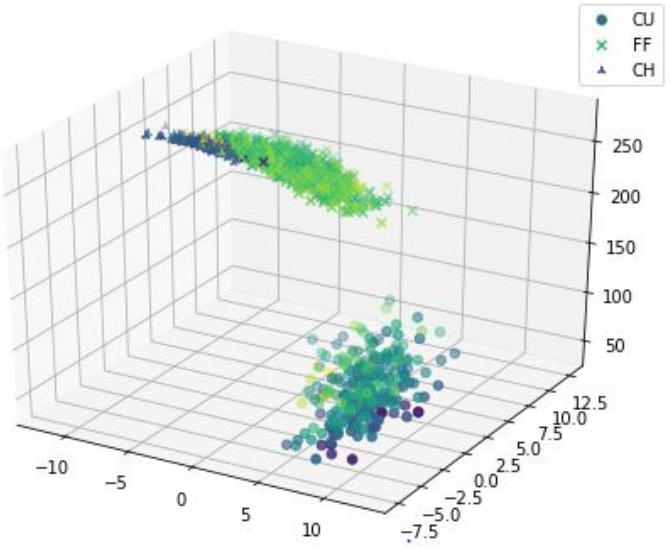
4. The result of K-Means clustering




```
Purity of class CU: 0.9966887417218543
Purity of class CH: 0.6340425531914894
Purity of class FF: 0.9834183673469388
Accuracy: 0.8872066616199848
```

The purity and accuracy is quite high in the result of k = 3. However, we discover that the result varies greatly. Every time we run the code, the result may be very different. We consider that the possible reason is that the initial centroid of the cluster is generated randomly, and it leads to variability of final center of cluster. Therefore, the result of clustering might be different every time, but the result is accurate overall.

## 5. Use another two or more attributes to partition

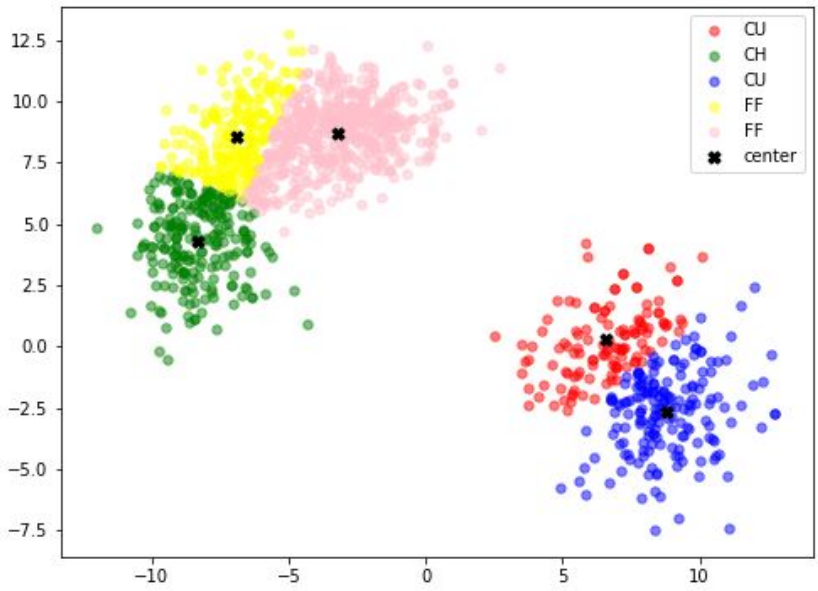



```
Purity of class CU: 1.0
Purity of class FF: 0.9820574162679426
Purity of class CH: 0.7989130434782609
Accuracy: 0.9167297501892505
```

This is the outcome of using four attributes(x, y, speed, spin) to partition. Color represents one of the four attributes. The accuracy shows that using four attributes is better than two attributes.

## 6. The reason of K=3

Because the highest accuracy of K=3 and other Ks are almost the same(about 90%), we think that K=3 is good enough and choose it as the best.




```
Purity of class CU: 1.0
Purity of class CH: 0.7117117117117117
Purity of class CU: 1.0
Purity of class FF: 0.9919028340080972
Purity of class FF: 0.9963702359346642
Accuracy: 0.9409538228614686
```

## 7. Kd-tree code

We use the library provided by Alexey (https://github.com/aabramovrepo) to implement Kd-tree and draw the diagram.

```
import csv
import numpy as np

results = []
with open('./points.txt', newline='') as inputfile:
    dataset = np.array(list(csv.reader(inputfile, delimiter=' '))).astype(int)
#print(dataset)
```

```
# Import KDTree Library
%run ./KDTreeLib/KDTree.ipynb

n = dataset.size  # number of points
min_val = 0       # minimal coordinate value
max_val = 10      # maximal coordinate value
delta = 1

# construct a K-D tree
kd_tree = kdtree(dataset.tolist())
```

```
# Import KDTree Library
%run ./KDTreeLib/KDTree.ipynb
import matplotlib.pyplot as plt

plt.figure("KD Tree", figsize=(10., 10.))
plt.axis( [min_val-delta, max_val+delta, min_val-delta, max_val+delta] )

plt.grid(b=True, which='major', color='0.8', linestyle='--')
plt.xticks([i for i in range(min_val-delta, max_val+delta, 1)])
plt.yticks([i for i in range(min_val-delta, max_val+delta, 1)])

# draw the tree
plot_tree(kd_tree, min_val-delta, max_val+delta, min_val-delta, max_val+delta, None, None)

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

8. The result of Kd-tree