Introduction to Machine Learning

Program Assignment #2

K-means and Kd-tree

Team 32

0416009 洪冠群
0416090 王湑晟
0416091 林彥岑
0416098 王于哲
0513406 陳凱文

1. What environments the members are using

Python3.6 , Anaconda: Spyder and Jupyter

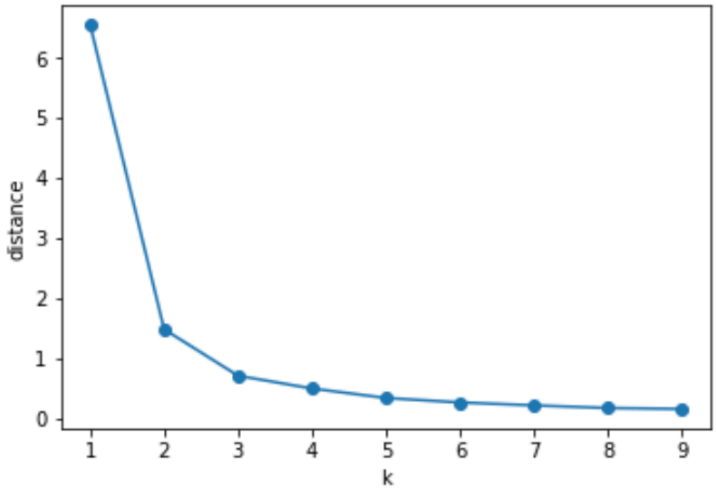
2. K-means code

```
class Kmeans(object):
    def __init__(self,cluster_num,data):
        self.centers = np.zeros((cluster_num,data.shape[1]), dtype=float)
        self.data = data
        self.cluster_num = cluster_num
        self.data_num = data.shape[0]
        self.feature_num = data.shape[1]
        self.clusters = np.zeros(data.shape[0],dtype=int)
        self.distances = np.zeros((data.shape[0],cluster_num), dtype=float)
        self.is_complete = False

    # Initial center by randomly selecting data
    def initialize_centers(self):
        self.centers = np.array([self.data[i]
                                 for i in random.sample(range(0, self.data_num),
                                                        self.cluster_num)])

    # Assignment data to the nearest center
    def assignment_step(self):
        # Calcaulate distance between two data
        def dis(a,b):
            return np.sqrt(np.sum((a-b)**2))

        for index_datum, datum in enumerate(self.data):# For each datum
            for index_center, center in enumerate(self.centers): # calculate distance to every center
                self.distances[index_datum][index_center] = dis(datum,center)
            self.clusters[index_datum] = np.argmin(self.distances[index_datum])# choose the nearest one

            # check the clustering situation
            numbers = np.bincount(self.clusters) # bincount the clustering situation
            # Bad center choose, there is at least one cluster has one member
            if np.count_nonzero(numbers) < self.cluster_num :
                self.initialize_centers()

    # Update new center
    def update_step(self):
        centers_old = deepcopy(self.centers)
        self.centers = np.zeros((self.cluster_num,self.feature_num),
                                dtype=float)
        for index_datum, datum in enumerate(self.data):
            self.centers[self.clusters[index_datum]]+= datum
        numbers = np.bincount(self.clusters)
        numbers.resize(self.cluster_num,1)
        self.centers = self.centers / numbers
        # if this time the center doesn't change, end the clustering.
        error = np.linalg.norm(self.centers - centers_old)
        if error == 0:
            self.is_complete = True

    # Calculate the whole clustering distance
    def distance_to_centroid(self):
        sum = 0
        for index_datum, datum in enumerate(self.data):
            sum += self.distances[index_datum][self.clusters[index_datum]]
        return sum/(self.cluster_num*self.data_num)

    # Do clustering
    def clustering(self):
        while self.is_complete is False:
            self.assignment_step()
            self.update_step()
```

3. How to decide cluster number k = 3 ?

先算出 k = 1 ~ 9 每種分法的到群中心的距離總和平均，再給定一個小數值 threshold(這裡為 0.5)，當 distance 的變化率小於 threshold 時的第一個 cluster number 就是選定的 k。

並沒有一個完美的方法可以決定 k，但因為 k-means 的方法中，增加 cluster 數目一定會讓 distance 下降，所以我們可以選擇當變化率小於某個小的臨界值時的 cluster number 來當作我們的 k。




```
We choose 3 as our clustering number,
because its absolute slope is smaller than 0.5
```

```
def decide_clustering_num(data):
    threshold_slope = 0.5
    cluster_num = None
    distance = np.zeros(9,dtype=float)
    x = [i for i in range(1,10)]
    #print(x)

    for i in range(1,10):
        print(f"Calculating {i} clustering distance...")
        kmeans = Kmeans(i,data)
        kmeans.clustering()
        distance[i-1] = kmeans.distance_to_centroid()
    for i in range(1,9):
        print(abs(distance[i]-distance[i-1]),end=' ')
        if cluster_num is None and abs(distance[i]-distance[i-1]) < abs(threshold_slope):
            cluster_num = i

    plt.plot(x,distance,'-o')
    plt.ylabel('distance')
    plt.xlabel('k')
    plt.show()
    print(f"""We choose {cluster_num} as our clustering number,
because its absolute slope is smaller than {threshold_slope}
""")
    return cluster_num
```

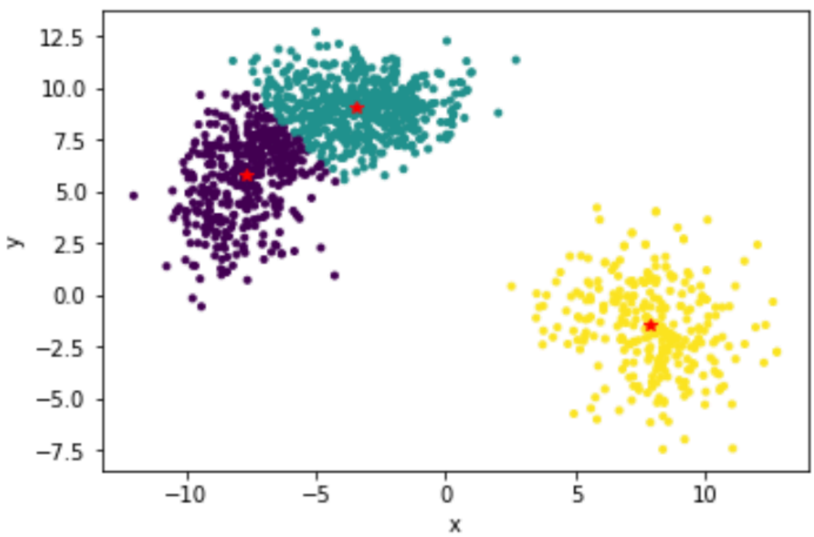
4. Cost function and accuracy

```
K-Means clustering Accuracy:  80.02%
```

當做完分群後需要去判斷分出來的 0,1,2 三群分別為 FF,CH,CU 的哪一個，然後再算出 accuracy，算法就是：(分對的 data/全部 data)。我們的做法是將 0,1,2 的各種情況做排列組合，算出最高的 accuracy 就是答案。

```python
def calculate_accuracy(model,truth):
    truth = [''.join(item) for item in truth] # np.array to list
    permutations = list(itertools.permutations([str(i) for i in range(kmeans.cluster_num)]))
    kmeans_result = np.array([str(i) for i in model])
    result_label = []
    accuracy = 0
    for i in range(len(permutations)):# count all possible permetations
        sum = 0
        # change the content of the string to be coherent
        pitch_type = [item.replace("FF",permutations[i][0]) for item in truth]
        pitch_type = [item.replace("CH",permutations[i][1]) for item in pitch_type]
        pitch_type = [item.replace("CU",permutations[i][2]) for item in pitch_type]
        # compare the list
        for i, j in zip(pitch_type,kmeans_result):
            if i == j:
                sum += 1
        #the best is the result we want
        if sum /len(kmeans_result) > accuracy:
            accuracy = sum /len(kmeans_result)
    print("%.2f%%" % (accuracy*100))
```

5. The result of K-Means clustering



6. Use another two or more attributes to partition

我們這組選了 vx0，以及 speed 這兩個 attributes 去做 K-Means。

```
Add two attribute 'vx0','speed' to do clustering, Accuracy = 100.00%
```

7. Kd-tree code

```python
# ## 2. KD-Tree


import pprint


def build_kd_tree(points, depth = 0 ):
    n = len(points)
    if n <= 0 :
        return None
    axis = (depth + SPILTTING_PLANE) % 2
    sorted_points = sorted(points, key=lambda point:point[axis])
    return {
        'left': build_kd_tree(sorted_points[:n//2],depth+1),
        'root': sorted_points[n//2],
        'right': build_kd_tree(sorted_points[n//2+1:],depth+1),
        'axis': axis
    }

def draw_kd_tree( node, x_min, x_max, y_min, y_max):
    if node is not None:
        plt.plot(node['root'][0],node['root'][1],'k.')
        if node['axis']: # X axis
            plt.plot([x_min,x_max], [node['root'][1],node['root'][1]],color='r',linewidth=0.5)
            draw_kd_tree(node['left'], x_min, x_max,y_min, node['root'][1])
            draw_kd_tree(node['right'], x_min, x_max, node['root'][1], y_max)
        else: # Y axis
            plt.plot([node['root'][0],node['root'][0]],[y_min,y_max],color='b',linewidth=0.5)
            draw_kd_tree(node['left'], x_min,node['root'][0],y_min, y_max)
            draw_kd_tree(node['right'], node['root'][0], x_max, y_min, y_max)
    plt.xlim([x_min, x_max])
    plt.ylim([y_min, y_max])
    plt.xlabel("x")
    plt.ylabel("y")
    plt.title("KD-Tree")

data = np.empty(shape=[0, 2])

with open("./datasets/points.txt") as file:
    for line in file:
        data = np.append(data, [[int(num) for num in line.split(" ")]],axis=0)

coordinate = ['X','Y']
print(f"std of {coordinate[0]}: {np.std(data[:,0])}, std of {coordinate[1]}: {np.std(data[:,1])}")

# Choose a axis with bigger std
SPILTTING_PLANE = 0 if np.std(data[:,0]) > np.std(data[:,1]) else 1

print(f"Choose {coordinate[SPILTTING_PLANE]} as axis-aligned splitting plane")
# Set range of coordinate
x_min = min(data[:,0])-1
y_min = min(data[:,1])-1
x_max = max(data[:,0])+1
y_max = max(data[:,1])+1

kdtree = build_kd_tree(data)
print("Show KD-Tree Structure\n")
pprint.pprint(kdtree)
print("\nShow KD-Tree Picture\n")
draw_kd_tree(kdtree, x_min, x_max, y_min, y_max)
```

8. The result of Kd-tree