Datamining Homework: Clustering

นาย ดรันภพ เป็งคำตา

580610642

รูปแบบการทดลอง

เป็นการทดลองการ การทำงานของ clustering algorithm กับ dataset ที่กำหนดไว้เพื่อเป็นการทดลอง เพื่อหาว่า การทำ clustering นั้น เหมาะสมกับ dataset ดังกล่าว หรือไม่ และทำการทดสอบด้วยค่า purity ซ่ง เป็นค่าบอกถึงความ บริสุทธิ์โดยรวมของคลัสเตอร์

ในการทดลองนี้ จะใช้ algorithm K-mean clustering โดยจะทดลองของเปลี่ยนค่า K เรื่อย ๆ จาก 2 ไปจนถึง 20 เพื่อตรวจสอบว่า ค่า k และ purity จะสัมพันธ์กันมากน้อยเพียงใด

รูปแบบของดาต้าเซทที่นำมาทดสอบ

เป็นดาต้าเซทชื่อ **Student performance dataset** เป็นดาต้าเซทของข้อมูลต่าง ๆ ของนักศึกษา เช่น โรงเรียน, อาชีพของบิดา มารดา, เพศ, อายุ และอื่น ๆ เป็นจำนวน 33 attributes โดยเป้าหมายของดาต้าเซทนี้ คือ การทำการประเมินผลของคะแนนสอบภาคเรียนสุดท้ายของนักศึกษาแต่ละคน โดยข้อมูลของดาต้าเซทนี้ ประกอบด้วย ข้อมูลลักษณะต่าง ๆดังนี้ เป็นattributeที่เป็นไปได้ 2 ค่า (binary), เป็นจานวนเต็มในช่วง ๆ หนึ่ง (numeric) และ เป็น ข้อความต่าง ๆ กันจำนวนหนึ่ง (nominal)

การ Preprocess ข้อมูล

เนื่องจากข้อมูลที่ได้มา พบว่าเป็นข้อมูลที่ไม่มี missing value จึงไม่จำเป็นที่จะต้องจัดการกับปัญหานี้ แต่จะมีการทำ normalize ข้อมูลทั้งหมดให้อยู่อยู่ในช่วง [0, 1] อย่างเท่าเทียมกัน ยกเว้นเว้น attribute ‘G3’ เนื่องจากเป็น class label ซึ่งจะไม่ได้ใช้ในการทำ clustering ครั้งนี้ และ เนื่องจากพบว่า attribute ทั้งหมด มีข้อมูลอยู่ สามแบบ จังได้ออกแบบ function สาหรับการ normalize ไว้เพื่อความสะดวกในการใช้งาน

- Function สำหรับ normalize ข้อมูล

```
def normalize(input):
    input['school'] = input['school'].apply(lambda x:binary('GP', x))
    input['sex'] = input['sex'].apply(lambda x:binary('F', x))
    input['age'] = input['age'].apply(lambda x:scaling(15, 22, x))
    input['address'] = input['address'].apply(lambda x:binary('U', x))
    input['famsize'] = input['famsize'].apply(lambda x:binary('LE3', x))
    input['Pstatus'] = input['Pstatus'].apply(lambda x:binary('T', x))
    input['Medu'] = input['Medu'].apply(lambda x:scaling(0, 4, x))
    input['Fedu'] = input['Fedu'].apply(lambda x:scaling(0, 4, x))
    input['Mjob'] = input['Mjob'].apply(lambda x:nominal(['teacher', 'health',
'services', 'at_home', 'other'], x))
    input['Fjob'] = input['Fjob'].apply(lambda x:nominal(['teacher', 'health',
'services', 'at_home', 'other'], x))
    input['reason'] = input['reason'].apply(lambda x:nominal(['home',
'reputation', 'course', 'other'], x))
    input['guardian'] = input['guardian'].apply(lambda x:nominal(['mother',
'father', 'other'], x))
    input['traveltime'] = input['traveltime'].apply(lambda x:scaling(1, 4, x))
    input['studytime'] = input['studytime'].apply(lambda x:scaling(1, 4, x))
    input['failures'] = abs(input['failures'].apply(lambda x:scaling(1, 4, x)))
    input['schoolsup'] = input['schoolsup'].apply(lambda x:binary('yes', x))
    input['famsup'] = input['famsup'].apply(lambda x:binary('yes', x))
    input['paid'] = input['paid'].apply(lambda x:binary('yes', x))
    input['activities'] = input['activities'].apply(lambda x:binary('yes', x))
    input['nursery'] = input['nursery'].apply(lambda x:binary('yes', x))
    input['higher'] = input['higher'].apply(lambda x:binary('yes', x))
    input['internet'] = input['internet'].apply(lambda x:binary('yes', x))
    input['romantic'] = input['romantic'].apply(lambda x:binary('yes', x))
    input['famrel'] = input['famrel'].apply(lambda x:scaling(1, 5, x))
    input['freetime'] = input['freetime'].apply(lambda x:scaling(1, 5, x))
    input['goout'] = input['goout'].apply(lambda x:scaling(1, 5, x))
    input['Dalc'] = input['Dalc'].apply(lambda x:scaling(1, 5, x))
    input['Walc'] = input['Walc'].apply(lambda x:scaling(1, 5, x))
    input['health'] = input['health'].apply(lambda x:scaling(1, 5, x))
    input['absences'] = input['absences'].apply(lambda x:scaling(0, 93, x))
    input['G1'] = input['G1'].apply(lambda x:scaling(0, 20, x))
    input['G2'] = input['G2'].apply(lambda x:scaling(0, 20, x))
```

- **Function สำหรับ normalize ข้อมูลที่เป็นช่วงค่า (numeric)**

```
def scaling(min, max, x):
    return (x - min)/(max - min)
```

- **Function สำหรับ normalize ข้อมูลที่เป็นสองค่า (binary)**

```
def binary(a, x):
    return 1 if x == a else 0.5
```

- **Function สำหรับ normalize ข้อมูลที่เป็นชุดของข้อความ (nominal)**

```
def nominal(p_list, x):
    for n, i in enumerate(p_list):
        if i == x:
            return scaling(0, len(p_list)-1, n)
```

เนื่องจากในกระบวนการของ K-mean cluster จะมีการใช้ geometric mean ซึ่งเป็นค่ารูทของผลคูณต่าง ๆ ซึ่ง หากค่าใดค่าหนึ่งมีค่า เป็น **0** จะทำให้การทำงานในส่วนนี้ผิดเพี้ยนไป ดังนั้นในการทำการ normalize จะพยายามลีกเลี่ยงให้เกิดค่า 0 ขึ้น

จำนวนรอบของการทดลอง

จะทำการทดสอบการทำงานของ K-mean เพื่อหาค่า purity ข้อแต่ละค่า k โดยจะเริ่มจาก ค่า 1 ไปจนถึง 20 เนื่องจากผลจากดาต้าเซทนี้มีคลาสคำตอบทั้งหมด 21 คลาส คือ 0 – 20 เพื่อตรวจสอบว่า หากทำการแบ่งคลัสเตอร์ เป็นจำนวนเท่ากับคลาส จะได้ผลอย่างไร

เพื่อความแม่นยำที่มากขึ้น ในการหาค่า purity ของแต่ละ k จะมีการทำซ้ำในแต่ละ k 3 รอบ และนำมาหาค่าเฉลี่ยเพื่อเป็นคำตอบของแต่ละค่า k

โค้ดส่วนการทำงานหลัก

```
import input
import k_mean as km

if __name__ == '__main__':
    data = input.loadFile()             #   dataset loading
    input.normalize(data)               #   normalize dataset
    for i in range(2,21):               #   Loop k from 2 - 20
        k = i
        avg_r = 3                       #   set repeating step
        sum_p = 0
        for j in range(1, avg_r+1):     #   loop for avr_r round
            final_centroidz, final_labels = km.kmeans(data.values, k)
                                        #   get the final centroid points -
                                        #   and final set of clusterd data

            p = km.getPurity(final_labels)
                                        #   calculate the purity
            sum_p += p
            print('%sK = %2d (%d) Purity %f'% (''.ljust(5), i, j, p))
        print('K = %2d Average Purity %f\n'% (i, sum_p/avg_r))
                                        #   show final purity for each k
```

ดังโค้ดที่ได้แสดงไป ในการทำงานหลักจะมีการดึงไฟล์ 2 ไฟล์มาใช้ ไฟล์ **input** รวบรวมคำสั่งเกี่ยวกับการอ่านไฟล์ และการ normalize ข้อมูล ส่วนไฟล์ **k_mean** จะเป็นการทำงานของ algorithm k_mean clustering

สามารถปรับรอบหาค่าเฉลี่ยได้ด้วยการแก้ไขค่า avg_r และ ฟังค์ชั่น kmeans() คือฟังค์ชั่นหลักในการทำงานนี้ ซึ่งจะคืนค่า จุด centroids ที่ปรับแล้ว และ ชุดค่าที่ผ่านการ labels มา

โค้ดการของไฟล์ k_mean

```
import numpy as np

def kmeans(dataSet, k):

    centroids = randomCentroids(dataSet.shape[1], k)

    iterations = 0
    diff = 9999
    o_centroids = None
    epsilon = 0.001
    max_iterations = 150

    while stopCondition(diff, iterations, epsilon, max_iterations):
        o_centroids = centroids
        iterations += 1

        labels = getLabels(dataSet, centroids)

        centroids = findMean(labels)

        diff = sum([distant(centroids[i], o_centroids[i]) for i in
range(len(centroids))])
        # print('This diff :',diff,'i :', iterations)

    return centroids, labels
```

นี่คือฟังค์ชั่นหลักของ k-mean clustering โดยมีหลักการทำงานคร่าวดังนี้

1. สุ่มค่าแต่ละจุด centroids มา k ตัว
2. เข้าสู่ลูปการทำงาน โดยมีเงื่อนไขในการหลุดคือ ค่าความต่างรวมของ centroids ปัจจุบัน และ ของ iteration ก่อนหน้ามีค่าต่างกันน้อยกว่าค่า **epsilon** หรือ จำนวน iterations มากกว่า **max_iteration**
3. จัดกลุ่มของข้อมูลต่างๆ โดยคิดตามระยะห่างจากจุด centroid ของที่ใกล้ที่สุด
4. หาก กลุ่มของ centroid ใดม่มีสมาชิกอยู่เลย ให้ทำการสุ่มค่าใส่centroid นี้ใหม่
5. ทำการหาค่า **Geometric mean** ของแต่ละสมาชิดในแต่ละ centroid โดยกำหนดให้ค่า centroid ใหม่คือค่าฉลี่ยที่ได้มาจากกระบวนการนี้
6. หาค่าความต่างรวมของทุก centroids เทียบกับ centroid ตัวเก่า
7. กลับไปทำซ้ำที่ข้อ 2
8. เมื่ออกจากลูปการทำงานได้แล้ว ผลลัพที่ได้ คือ centroids ณ ขณะนั้น

```
def randomCentroids(attr, k):
    res = []
    for i in range(k):
        tmp = []
        for j in range(attr):
            tmp.append(np.random.rand())
        res.append(tmp)
    return res
```

ฟังชั่น สุ่มค่าให้กับ centroids ในตอนเริ่มต้น

```
def distant(a, b):
    return np.sqrt(sum((a[:-1] - b[:-1]) ** 2))
```

ฟังก์ชั่นหา distant ของจุดสองจุด *โดยจะไม่นำค่า class label มายุ่งเกี่ยวในการคำนวนด้วย

```
def stopCondition(dif_c, iterations, epsilon=0.1, max_it = 1000):
    return dif_c > epsilon and iterations < max_it
```

เงื่อนในการออกลูปการทำงาน

```
def getLabels(data, centroids):
    res = [[] for i in range(len(centroids))]
    for i in data:
        min = [1000,-1]
        for nj, j in enumerate(centroids):
            dis = distant(i,j)
            if dis < min[0]:
                min[0] = dis
                min[1] = nj
        res[min[1]].append(i)
    return np.asarray(res)
```

ฟังก์ชั่น จำแนกกลุ่มของข้อมูลตามจุด centroids

```
def findMean(labels, s_attr=33):
    res = []
    for i in labels:
        l = len(i)
        tmp = [np.power(a, 1./l) for a in [np.prod(x) for x in zip(*i)]]
        if tmp == []:
            tmp = list(np.random.rand(s_attr))
        res.append(tmp)
    return np.asarray(res)
```

ฟังก์ชั่น การหาค่า **Geometric mean** และกำหนด centroids ใหม่ โดยจะเห็นว่าหากไม่มีสมาชิดภายใน centroid จุดนั้นจะถูกสุ่ม ขึ้นมาใหม่

```
def getPurity(labels, k=21):
    s = 0
    ip = 0
    for i in labels:
        s += len(i)
        max = [0 for x in range(k)]
        for nj, j in enumerate(i):
            max[int(j[-1:][0])] += 1
        ip += max[np.argmax(max)]
    return ip / s
```

$$\text{purity}(\Omega, \mathbb{C}) = \frac{1}{N} \sum_{k} \max_{j} |\omega_k \cap c_j|$$

ฟังก์การคำนวณหาค่า purity ของ cluster โดยทำงานตามสมการอ้างอิงด้านบน

โค้ดของไฟล์ input

```
import numpy as np
import pandas as pd
import os
import math
input_file = {1:'student-mat.csv', 2:'student-por.csv'}

def loadFile(file_number=1):
    if not file_number in input_file:
        file_number = 1

    fpath = os.path.join('dataset', input_file[file_number])
    return pd.read_csv(fpath, sep=';', header=0, na_values='?')
```

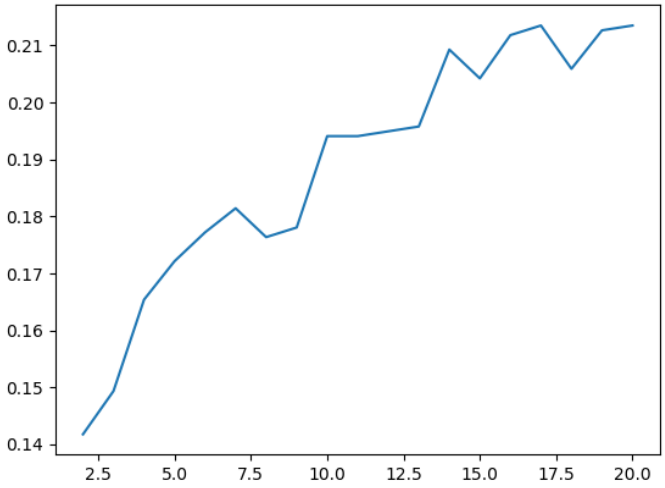
ทำการ ดึงไลบรารี่ที่จำเป็น และนี่คือฟังก์ชั่นของการโหลดข้อมูล ดาต้าเซท

ผลลัพธ์จากการทำงาน

K	Round	Purity
2	1	0.141772
	2	0.141772
	3	0.141772
	Average	**0.141772**
3	1	0.159494
	2	0.154430
	3	0.146835
	Average	**0.153586**
4	1	0.156962
	2	0.172152
	3	0.156962
	Average	**0.162025**
5	1	0.192405
	2	0.179747
	3	0.174684
	Average	**0.182278**
6	1	0.16557
	2	0.184810
	3	0.182278
	Average	**0.177215**
7	1	0.162025
	2	0.189873
	3	0.172152
	Average	**0.174684**
8	1	0.187342
	2	0.182278
	3	0.174684
	Average	**0.181435**
9	1	0.174684
	2	0.164557
	3	0.184810
	Average	**0.174684**
10	1	0.194937
	2	0.189873
	3	0.200000
	Average	**0.194937**
11	1	0.182278
	2	0.187342
	3	0.184810
	Average	**0.184810**

K	round	Purity
12	1	0.169620
	2	0.192405
	3	0.192405
	Average	**0.184810**
13	1	0.179747
	2	0.192405
	3	0.192405
	Average	**0.188186**
14	1	0.197468
	2	0.197468
	3	0.189873
	Average	**0.194937**
15	1	0.215190
	2	0.197468
	3	0.18973
	Average	**0.200844**
16	1	0.215190
	2	0.210127
	3	0.212658
	Average	**0.212658**
17	1	0.225316
	2	0.217722
	3	0.205063
	Average	**0.216034**
18	1	0.225316
	2	0.207595
	3	0.225316
	Average	**0.219409**
19	1	0.215190
	2	0.215190
	3	0.217722
	Average	**0.216034**
20	1	0.210127
	2	0.220253
	3	0.217722
	Average	**0.216034**

สรุปผลการทดลอง



จากผลการทดลอง จะเห็นได้ว่า ค่า purity จะเพิ่มขึ้นตามค่า k ซึ่งค่อนข้างเป็นไปตามสมมุติฐาน แต่ว่าจุดที่ผู้ทดลองสนในคือ หากเป็นที่ k = class แล้วค่าที่ได้จะเป็นอย่างไร ซึ่งผลปรากฏว่า ค่าไม่ได้มากหรือน้อยจนเป็นที่สนใจแต่อย่างใด จึงสรุปผลการทดลองได้ว่า

ชุดข้อมูลที่นำมาทำการทดสอบ ไม่เหมาะกับการนำมาทำ clustering เพื่อใช้ในการจำแนก class ของผลลัพธ์ อย่างน้อยกับการใช้ k-mean algorithm เพราะผลจากค่า purity ไม่ได้โดดเด่น อยากที่คาดหวังไว้