ML:Homework4——Group\_Discovery

人工智能91 卢佳源 2191121196

1. 实验目的：

对视频中每一帧的行人轨迹坐标点进行聚类，自动发现行人组。

1. 利用students003数据进行实验；
2. 可视化每一帧的聚类结果，形成视频，用不同颜色标记不同的行人组，每帧至少2组行人；
3. 开放任务，聚类方法不限。
4. 实验环境：

OS：Linux操作系统的ubuntu20.04版本

IDE：VScode,anaconda3(‘base’:conda)

Terminal：gnome-terminal -t $TITLE -x

Language：python(3.8.8 64-bit)

PATH：/home/lujiayuan/ML/Homework3\_Group\_Discovery/Group\_Discovery1.py

DataPath:/home/lujiayuan/ML/Homework3\_Group\_Discovery/TrajectoryData\_students003/TrajectoryData\_students003/students003.txt

1. 实验方法：
2. 实验思路：
3. 读入行人位置、时间和ID信息，并保存在列表中；
4. 将每十帧的行人坐标(x,y)放入一个距离矩阵xx，并将该矩阵传入DBSCAN聚类函数中（DBSCAN函数的实现在调试时用的是sklearn.cluster库自带的DBSCAN函数，也可以自己实现该算法（也在提交代码中），但是自己写的DBSCAN的聚类运行速度太慢，原因见“五——问题1”）；
5. 将DBSCAN函数返回的labels值进行聚类的可视化，用matplotlib画出每一帧的聚类结果，相同颜色代表在这一帧中他们被划分为一类；
6. 调用images\_to\_video()函数将（3）中每一帧的聚类结果连接生成视频。
7. 实验代码：

import numpy as np

import os

import cv2

import random

from collections import defaultdict

from collections import ChainMap

from sklearn.cluster import DBSCAN

import matplotlib.pyplot as plt

import matplotlib

matplotlib.use('TkAgg')

from IPython import embed

def find\_neighbor(X,j,eps):

neighbor=[]

for i in range(X.shape[0]):

oushi=np.sqrt(np.sum(np.square(X[j]-X[i])))

if oushi<=eps:

neighbor.append(i)

return neighbor

def dbscan(X,eps,min\_samples):

neighbor=[]

visited=[]

flag=[x for x in range(len(X))]

cluster=[-1 for i in range(len(X))]

k=-1

while len(flag)>0:

sample=random.sample(flag,1)

flag.remove(sample)

visited.append(sample)

neighbor=find\_neighbor(X,sample,eps)

if len(neighbor)>=min\_samples:

k+=1

cluster[sample]=k

for i in neighbor:

if not i in visited:

flag.remove(i)

visited.append(i)

if cluster[i]==-1:

cluster[i]=k

else:

cluster[sample]=-1

return cluster

def images\_to\_video(path):

img\_array = []

imgList = os.listdir(path)

imgList.sort(key=lambda x: int(x.split('.')[0]))

for count in range(0, len(imgList)):

filename = imgList[count]

img = cv2.imread(path + filename)

if img is None:

print(filename + " is error!")

continue

img\_array.append(img)

height, width, layers = img.shape

size = (width, height)

fps = 5

out = cv2.VideoWriter('GD\_handin.avi', cv2.VideoWriter\_fourcc(\*'DIVX'), fps, size)

for i in range(len(img\_array)):

out.write(img\_array[i])

out.release()

def main():

path = "img22/"

images\_to\_video(path)

datapath='TrajectoryData\_students003/TrajectoryData\_students003/students003.txt'

data=open(datapath,'r',encoding='utf-8',errors='ignore')

line=data.readline()

step=[]

ID=[]

ID1=[]

X1=[]

Y1=[]

while line:

m=line.split()

m=list(map(eval,m))

a=m[0:1]

step.append(a)

b=m[1:2]

ID1.append(b)

c=m[2:3]

X1.append(c)

d=m[3:4]

Y1.append(d)

line=data.readline()

data.close()

step=np.array(step)

ID=np.array(ID1).astype(int)

X1=np.array(X1)

Y1=np.array(Y1)

x=np.hstack((X1,Y1))

k=0

t=0

index=10

for i in range(len(step)):

if int(step[i])==index:

xx=[]

xx=x[k:i]

text=ID[k:i]

db = DBSCAN(eps=1.2, min\_samples=1).fit(xx)

# db = dbscan(xx,1.2,1)

labels=db.labels\_

label=np.array(labels)

core\_samples\_mask = np.zeros\_like(labels, dtype=bool)

core\_samples\_mask[db.core\_sample\_indices\_] = True

n\_clusters\_ = len(set(labels)) - (1 if -1 in labels else 0)

n\_noise\_ = list(labels).count(-1)

print("Estimated number of clusters: %d" % n\_clusters\_)

print("Estimated number of noise points: %d" % n\_noise\_)

unique\_labels = set(labels)

colors = [plt.cm.rainbow(each) for each in np.linspace(0, 1, len(unique\_labels))]

for k, col in zip(unique\_labels, colors):

if k == -1:

col = [0, 0, 0, 1]

class\_member\_mask = labels == k

xy = xx[class\_member\_mask & core\_samples\_mask]

plt.plot(

xy[:, 0],

xy[:, 1],

"o",

markerfacecolor=tuple(col),

markeredgecolor="k",

markersize=14,

)

xy = xx[class\_member\_mask & ~core\_samples\_mask]

plt.plot(

xy[:, 0],

xy[:, 1],

"o",

markerfacecolor=tuple(col),

markeredgecolor="k",

markersize=6,

)

for j in range(len(text)):

plt.annotate(text[j],xy=(xx[j,0],xx[j,1]),xytext=(xx[j,0]+0.1,xx[j,1]+0.1))

plt.savefig('img22/'+str(t)+'.png')

plt.clf()

plt.close()

index+=10

k=i

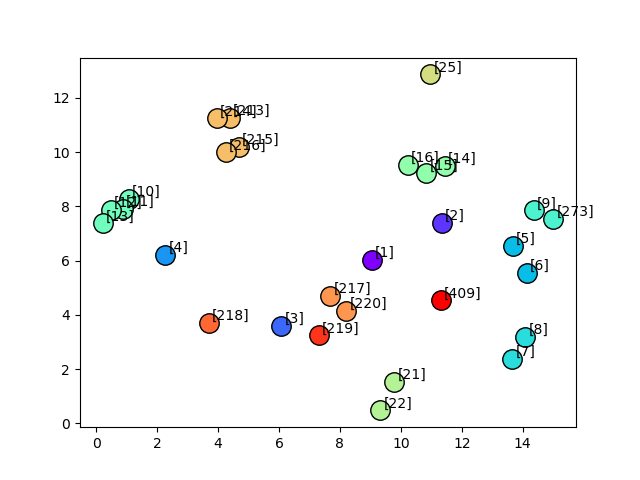
t+=1

main()

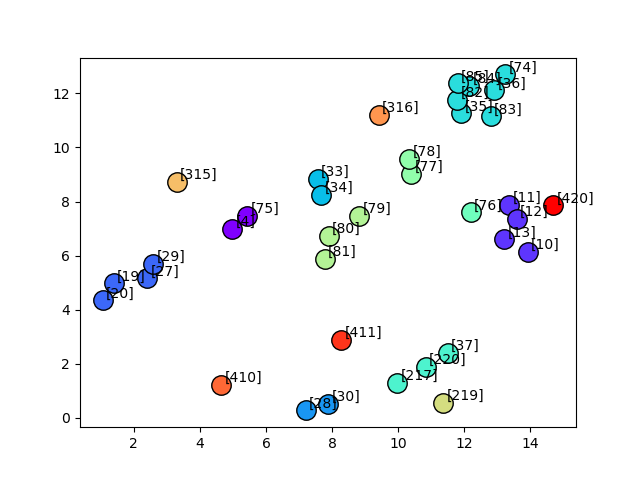
1. 实验结果：
2. 实验结果：
3. 每一帧的聚类结果图片：（所有540帧的聚类结果见附件）

eps=1.2, min\_samples=1

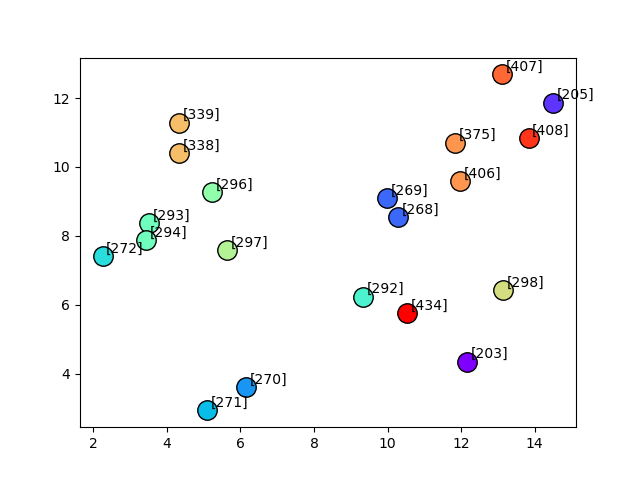
第0帧聚类结果：



第51帧聚类结果：



第539帧聚类结果：

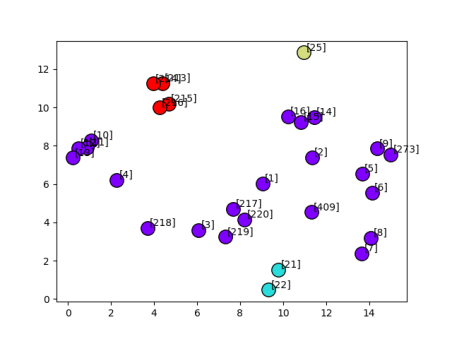
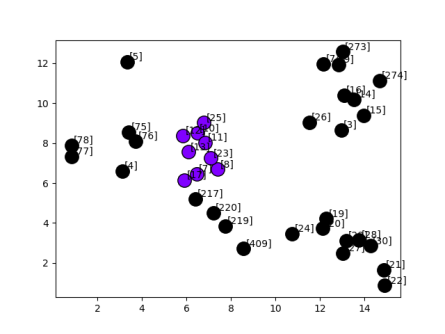


1. 540帧的聚类视频：见附件
2. 实验结果分析：

通过DBSCAN算法进行每一帧的空间上的聚类得到的结果图，可以看出，在核心对象去心邻域的距离阈值设为1.2，样本点成为核心对象所需要的去心邻域的最小样本数阈值设为1时，该算法对于这种空间聚类的效果还是比较好的，不同的人团被标记了不同的颜色，但是当许多不同类别人群聚集在一起时，这种没有记忆的分类方式就会把他们都划分到一类，但是这也的确是算法本身的计算方式存在的问题。

通过改变DBSCAN函数的参数，可以得到不同的聚类结果：随着距离阈值的增大，聚类类别数会减少，即更多的点被划分为一类；随着样本阈值的增大，核心对象数量会减少，更多的点被划分为噪声点（黑色），类别数可能变多。需要不断调整这两个参数，找到一个适于本数据集的参数对。

第0帧聚类结果:eps=3, min\_samples=1 ;第21帧聚类结果：eps=3, min\_samples=1

1. 遇到问题及解决思路：

问题1：sklearn.cluster库中的DBSACN函数对一次聚类的运行速度在毫秒数量级，但是自己写的DBSCAN函数进行一次分类却要花费近半分钟的原因？

解决思路：

sklearn.cluster库中的DBSACN函数使用了Kd—Tree加速索引方法来查询邻域内的点。

最传统的DBSCAN算法（PPT上的算法）是先根据给定邻域参数找出所有核心对象，然后任意选择一个没有类别的核心对象作为种子，找到所有这个核心对象能够密度可达的样本集合，作为一个聚类簇。接着继续选择另一个没有类别的核心对象，寻找密度可达的样本集合，得到另一个聚类簇，直到所有核心对象都有类别为止。计算复杂度为O(n^2)。

sklearn.cluster库中的DBSACN函数在每次查找邻域内样本点时都先建树，然后用Kd—Tree进行最近邻查找（先进行二叉树搜索，随搜索路径找到最近邻的近似点，即与待查询点处于同一个子空间的叶子结点；然后进行回溯搜索路径，判断搜索路径上该结点的其他子结点空间中是否有距查询点更近的样本点，如果有，则要跳到其他子结点空间中去搜索，即将其他子结点加入到搜索路径中，直到搜索路径变空为止。）Kd-Tree的时间复杂度最坏为O(n)，样本点随机分布时为O(logN)，因此使用Kd-Tree进行搜索的DBSCAN的总体计算复杂度为O(NlogN)，比传统方法快很多。

问题2：如何做到相邻帧之间的同一类别的聚类标记颜色尽量不变？

解决思路：

设定时间阈值为2帧，即在相邻2帧中的同一类别颜色尽量不变。

方法1：在散点图分配颜色时，用rainbow代替spectral，这样较小或较大ID号或且在同一个类别时，在组别混合再分开后可以尽量保持原有颜色，使得相邻两帧中同类别的点颜色不会发生剧变。（本次实验代码采取的方法）

方法2：假设第一帧中有set1(A,B,C)，set2(D,E)，set3(F)，第二帧有set1(A,B,C,X)，set2(D)，set3(F)，则将第二帧中的X划进与第一帧的set1相同颜色的类别中，将第二帧的set2中的E剔除，成为与第一帧set2不同的颜色，D和F继续保持颜色。事先设定40种颜色集合，每生成一组set，为其分配一种颜色，直到颜色用完，再循环分配颜色（此时与第一次使用该颜色已经间隔很长时间了，所以同时两组用到该颜色而产生混淆的概率比较低）。

（这种情况颜色改善情况不大）

但是我在试验过程中发现当两个类别相聚再分开时，组别内人员会发生改变，此时再去分配颜色时，必然会出现新增组和融合组，这种情况下再讨论颜色剧变的问题其实意义不大。