**西安交通大学实验报告**

**课程名称**： 人工智能的现代方法Ⅱ-机器学习

**实验名称**： 基于DBSCAN的行人组聚类

**姓 名**： 陈梓轩

**班 级**： 人工智能002

**学 号**： 2204411669

**学 院**： 人工智能学院

**实验日期**： 2022年 11 月 30 日

# 一、实验目的

本次实验利用students003数据进行实验，目的为对视频中每一帧的行人轨迹坐标点进行聚类，自动发现行人组并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。

本次实验采用的聚类算法为DBSCAN算法，并根据相邻帧的联系优化算法结果。

# 二、实验环境

## 1、实验硬件设备列表

CPU：i7-10875H CPU @ 2.30GHz；

GPU：NVIDIA GeForce RTX2060；

## 2、实验软件配置列表

编程语言：Python 3.9，

编辑器：PyCharm Community Edition 2022.2.2

# 三、实验步骤与结果分析

## 1.DBSCAN算法原理

密度聚类亦称“基于密度的聚类”(density-based clustering)，此类算法假设聚类结构能通过样本分布的紧密程度确定。通常情形下，密度聚类算法从样本密度的角度来考察样本之间的可连接性，并基于可连接样本不断扩展聚类簇以获得最终的聚类结果。

DBSCAN (Density-Based Spatial Clustering of Application with Noise)是一种著名的密度聚类算法。该算法将具有足够密度的区域划分为簇，并在具有噪声的空间数据库中发现任意形状的簇，它将簇定义为密度相连的点的最大集合。

DBSCAN算法基本流程如下所示：

输入：样本集D=()，邻域参数(ϵ, MinPts), 样本距离度量方式

输出：簇划分C。

步骤：

**（1）**初始化核心对象集合Ω=, 初始化聚类簇数k=0，初始化未访问样本集合Γ = D,  簇划分C = ；

**（2）** 对于j=1,2,...m, 按下面的步骤找出所有的核心对象：

**（a）** 通过距离度量方式，找到样本的ϵ-邻域子样本集；

**（b）** 如果子样本集样本个数满足||≥MinPts，将样本加入核心对象样本集合：Ω=Ω∪{}；

**（3）**如果核心对象集合Ω=，则算法结束，否则转入步骤4；

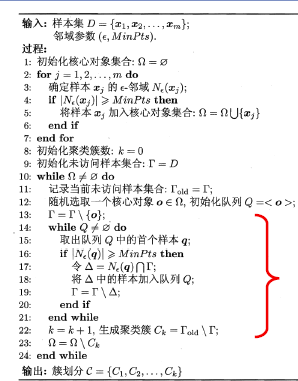
**（4）**在核心对象集合Ω中，随机选择一个核心对象o，初始化当前簇核心对象队列={o}, 初始化类别序号k=k+1，初始化当前簇样本集合, 更新未访问样本集合Γ=Γ−{o};

**（5）**如果当前簇核心对象队列=，则当前聚类簇生成完毕, 更新簇划分,更新核心对象集合Ω=Ω−， 转入步骤3。否则更新核心对象集合Ω=Ω−；

**（6）**在当前簇核心对象队列中取出一个核心对象o′,通过邻域距离阈值ϵ找出所有的ϵ-邻域子样本集，令Δ=∩Γ, 更新当前簇样本集合=∪Δ, 更新未访问样本集合Γ=Γ−Δ,  更新=∪(Δ∩Ω)−o′，转入步骤5；

输出结果为： 簇划分。

相应伪代码如下图所示：



**图1 DBSCAN伪代码**

本次实验需要实现在每一帧中对行人进行分组且行人的标签信息未知，因此可以采用无监督学习中的DBSCAN算法进行行人聚类。由于DBSCAN是根据当前帧样本距离进行聚类，忽略了时间维度下行人的运动轨迹方向，因此可能出现不同行人组聚成一簇的错分现象；同时，行人轨迹重叠还可能导致多个行人组聚类成一个行人组的错误。为优化DBSCAN的聚类结果，本实验还通过考虑当前帧前后k帧的聚类结果，取频数最高的类别来更新当前帧的聚类结果。最终，实验将聚类结果可视化后保存为图片作为视频的帧并生成视频。

## 2.实验过程

### （1）行人轨迹坐标点的读取与划分：

根据READme.txt文件，students003.txt中四列数据分别为Time step、ID 、X 、Y。由于后续工作需要在每一帧中对数据点进行聚类，因此还需要将数据划分至每一帧。

利用pandas库读取数据：

# 读取行人轨迹坐标点  
Allpoints = pd.read\_csv('TrajectoryData\_students003/students003.txt', sep='\t',  
 names=['Timestep', 'ID', 'X', 'Y'], header=None)

将数据点按帧划分，共541帧：

# 获得每一帧的行人坐标位置,共541帧  
frame = []  
for i in range(0, 541):  
 timei = (Allpoints["Timestep"] == i \* 10) # 第i帧的时间  
 framei = Allpoints.loc[timei, "X":"Y"].values # 第i帧时各行人的坐标位置  
 frame.append(framei)

### （2）DBSCAN聚类算法：

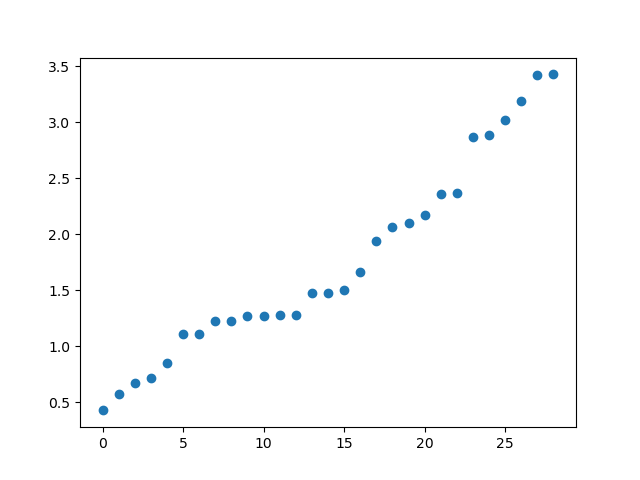
以欧氏距离作为样本点的距离度量：

# 计算两个点之间的欧式距离，参数为两个数组坐标  
def dist(t1, t2):  
 dis = math.sqrt((np.power((t1[0] - t2[0]), 2) + np.power((t1[1] - t2[1]), 2)))  
 # print("两点之间的距离为："+str(dis))  
 return dis

DBSCAN算法进行前有两个超参数待确定：Eps（指定半径参数）与MinPts（制定邻域密度阈值,包括点本身）。DBSCAN算法原文中定义了K-距离，即给定K邻域参数k,对于数据中的每个点，计算对应的第k个最近邻域距离，并将数据集所有点对应的最近邻域距离按照降序方式排序，称这幅图为排序的k距离图，选择该图中第一个谷值点位置对应的k距离值设定为Eps。按照常理，行人组的最小人数应为2人，因此设置k为2，取第一帧数据点绘制k距离图：

k\_dist = [] # k距离，取k=2  
Dist = np.zeros((len(frame[0]), len(frame[0])))  
for j in range(len(frame[0])):  
 for k in range(len(frame[0])):  
 Dist[j][k] = dist(frame[0][j], frame[0][k])  
 Dist[j].sort()  
 k\_dist.append(Dist[j][2])  
k\_dist.sort()  
x = [i for i in range(len(frame[0]))]  
plt.figure()  
plt.scatter(x, k\_dist)  
plt.show()

获得k距离图如下所示：



**图2 第0帧数据点的k距离**

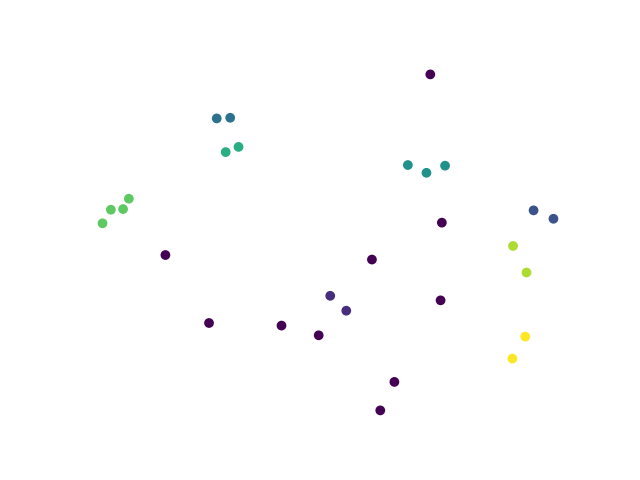
由上图可知，k距离图在距离约为1.1处变化距离，因此取Eps=1.1。又按照常理，行人组的最小人数应为2人，因此设置MinPts=2。由此，DBSCAN算法如下所示：

# DBSCAN算法，参数为数据集，Eps为指定半径参数，MinPts为制定邻域密度阈值,包括点本身  
def dbscan(Data, Eps, MinPts):  
 num = len(Data) # 点的个数  
 # print("点的个数："+str(num))  
 C = [-1 for i in range(num)] # C为输出结果，默认是一个长度为num的值全为-1的列表  
 # 用k来标记不同的簇，k = -1表示噪声点  
 k = -1  
 unvisited = set([i for i in range(num)]) # 没有访问到的点的列表  
 Nei\_list = [] # 存储每个数据点邻域点  
 coreset = [] # 核心点集  
 for j in range(num):  
 # N为xj的epsilon邻域中的对象的集合  
 N = []  
 for i in range(num):  
 if (dist(Data[i], Data[j]) <= Eps):  
 N.append(i)  
 Nei\_list.append(N)  
 if len(N) >= MinPts:  
 coreset.append(j)  
 coreset = set(coreset)  
 while len(coreset) != 0:  
 old = copy.deepcopy(unvisited)  
 o = random.choice(list(coreset))  
 Q = []  
 Q.append(o)  
 unvisited.remove(o)  
 while len(Q) != 0:  
 q = Q[0]  
 Q.remove(q)  
 # Nei为q的epsilon邻域中的对象的集合  
 Nei = set(Nei\_list[q])  
 delta = Nei & unvisited # delta为q的epsilon邻域与未访问点集的交集  
 Q = Q + list(delta)  
 unvisited = unvisited - delta  
 k = k + 1  
 Ck = old - unvisited  
 for i in Ck:  
 C[i] = k  
 coreset = coreset - Ck  
 return C

按照设置的参数进行DBSCAN聚类：

eps = 1.1  
min\_Pts = 2  
for i in range(0, 541):  
 C = dbscan(frame[i], eps, min\_Pts)  
 plt.figure()  
 plt.scatter(frame[i][:, 0], frame[i][:, 1], c=C)  
 plt.axis('off')  
 f = plt.gcf()  
 f.savefig(r'img\_ori\{}.png'.format(i)) # 保存第i帧聚类结果图  
 # plt.show()  
 plt.close()

以第0帧为例，聚类结果如下所示：



**图3 第0帧的聚类结果**

### （3）利用相邻k帧关系更新DBSCAN聚类结果：

首先获取行人组的坐标信息与ID信息：

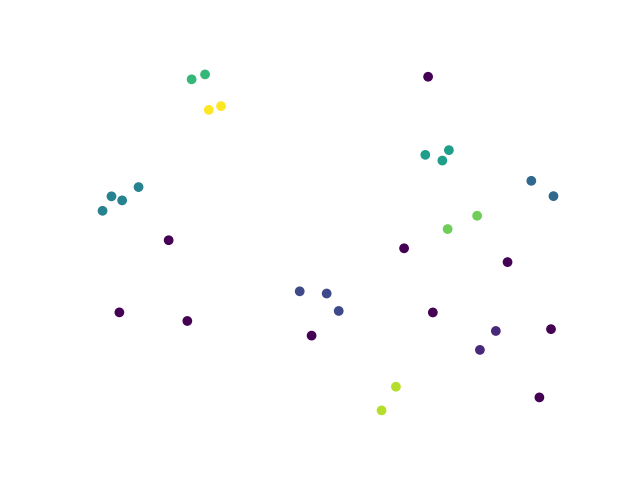
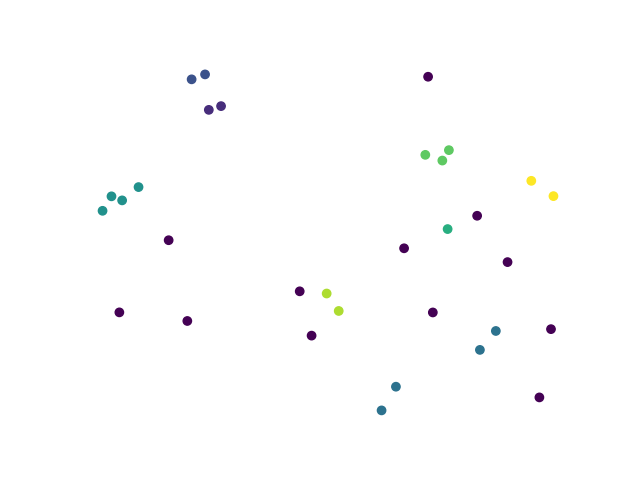
# 读取行人轨迹坐标点  
Allpoints = pd.read\_csv('TrajectoryData\_students003/students003.txt', sep='\t',  
 names=['Timestep', 'ID', 'X', 'Y'], header=None)

# 获得每一帧的行人坐标位置,共541帧  
frame = []  
ID = []  
for i in range(0, 541):  
 timei = (Allpoints["Timestep"] == i \* 10) # 第i帧的时间  
 framei = Allpoints.loc[timei, "X":"Y"].values # 第i帧时各行人的坐标位置  
 frame.append(framei)  
 IDi = Allpoints.loc[timei, "ID"].values # 第i帧时各行人的ID  
 ID.append(IDi)

对于第i帧视频内的每一个行人，根据ID在该帧前后k帧中进行寻找操作。如果在该帧前后k帧中找到第i帧内的某个行人，则将该行人在该帧及前后k帧所在的聚类簇加入cluster。以cluster中各聚类簇的出现频数作为相应的得分，最终该行人在该帧（第i帧）的聚类结果为得分最高的簇：

def most(list): # 寻找最大频数的类  
 count = Counter(list)  
 k = list[0]  
 for key in count.keys():  
 if count[key] > count[k]:  
 k = key  
 return k  
  
  
def find(IDk, x): # 在第t帧中寻找ID为x的人  
 for i in range(IDk.shape[0]):  
 if IDk[i] == x:  
 return i # 返回他的索引  
 return -1  
  
  
def union\_clustering(C, ID, k):  
 for i in range(len(C)): # i为帧数  
 for j in range(len(C[i])): # j为第i帧内的第j个人  
 cluster = []  
 for t in range(max(i - k, 0), min(i + k + 1, len(C))): # 只考虑第i帧前后k帧的联系  
 l = find(ID[t], ID[i][j]) # 如果第t帧内存在第i帧内第j个人  
 if l >= 0:  
 cluster.append(C[t][l]) # 如果第t帧内存在第i帧内第j个人，则该人在第t帧所属类的得分加1  
 C[i][j] = most(cluster) # 取出现频数最高的类作为第i帧第j个人的聚类簇  
 return C

以第3帧为例：下图分别为优化前后第3帧的行人聚类结果：

**图4 第3帧的原始聚类结果 图5 第3帧的优化聚类结果**

### （4）生成视频：

利用openCV中的VideoWriter将得到的聚类结果图片生成视频。由图片文件参数得，图片大小为640\*480。因此设置每秒帧数为5，视频流大小为 (640, 480)，帧数为541帧，视频格式为mp4：

import cv2  
  
fps = 5  
size = (640, 480)  
frames = 541  
fourcc = cv2.VideoWriter\_fourcc(\*"mp4v")  
video = cv2.VideoWriter('video\_ori\_5fps.mp4', fourcc, fps, size)  
for filename in ['img\_ori/{0}.png'.format(i) for i in range(frames)]:  
 img = cv2.imread(filename)  
 if img is None:  
 print(filename + " is error!")  
 continue  
 video.write(img)  
video.release()

## 3.实验评价

DBSCAN算法原始聚类的行人组视频及经前后k帧优化后的行人组划分视频如附件video\_ori\_5fps.mp4与video\_update\_5fps.mp4所示。对比优化前后视频可知，原视频普遍存在因忽略了时间维度下行人的运动轨迹方向导致不同行人组聚成一簇的错分现象，以及因行人轨迹重叠还可能导致多个行人组聚类成一个行人组的错误。而考虑了相邻k帧的视频对以上错分现象由一定程度的改善（同一行人组在一定时间内得到保持），但是效果不明显，还需要深入挖掘相邻帧信息进行进一步优化。

# 四、总结与收获

通过本次实验，我成功实现将DBSCAN算法从伪代码到python代码的转换，并在行人组划分的背景下成功实现DBSCAN聚类划分。考虑到视频流数据的特殊性，本次实验选择利用当前帧前后k帧的数据对原始DBSCAN聚类结果进行优化。由于聚类算法与优化算法较简单，实际划分结果不尽人意，行人轨迹数据特征还需要进一步的挖掘。