# 1 代码说明

下面给出程序源说明，完整Python文件见附件。数据集名称为data.csv，k-means.py、DBSCAN.py和Agglomerative.py为对比测试的程序，draw\_result.py绘制了实验部分图像，main\_fusion.py为本文方法的程序，数据集和程序在同一个目录下才可以运行。

# 2 附录I(主程序源代码)

下面给出程序源代码，完整Python文件见附件。

# -\*- coding: utf-8 -\*-

# ！/usr/bin/python3

# @Time   : 2022.4.10

# @Author : Yue Gaofeng

# @version: V1.0

# @Des    : Multi-sensor fusion

from time import \*

import scipy as sp

from sklearn import cluster

import matplotlib.pyplot as plt

import numpy as np

import scipy.stats as st

if \_\_name\_\_ == "\_\_main\_\_":

    begin\_time = time()      # 计算程序运行时间

    dataset = np.loadtxt('data\_z.csv',delimiter=',')  # 读取原始数据集

    # 初始化各种变量

    N = 4            # 传感器数量

    delta\_t = 6      # 时间差定义为6，可变化

    lamda = 4        # 分组个数定义为4

    t = 0            # 启动时刻

    seq = len(dataset)

    W = np.array([])  # 权值向量,后期可加

    result = np.zeros([len(dataset),1])

    while(t < seq):  # 无法启动聚类滤波核时

        if (t < delta\_t):

            result[t,0] = np.mean(dataset[t,1:5])

        else:

            data = dataset[t-delta\_t:t,1:5]         # 确定滤波核元素

            d\_max, d\_min= np.max(data), np.min(data)

            L = (d\_max - d\_min)/2.0                  # 确定核的高度

            # 计算K-Means中的k值

            for i in range(np.size(data)): Dp = data.reshape(1,delta\_t\*N)  # 投影值计算

            Dp = sorted(Dp[0].tolist())       # 排序

            per = (Dp[-1]-Dp[0])/lamda

            # 计算K

            Sp = [0]\*lamda

            for i in range(lamda):

                count = 0

                for j in range(delta\_t\*N):

                    if (Dp[0]+i\*per) <= Dp[j] < (Dp[0]+(i+1)\*per):  # 统计投影特征值

                        count += 1

                Sp[i] = count/(delta\_t\*N)  # 投影值计算

            hi\_Sp = max(Sp) #根据最大可能准则（Maximal Feasible Criterion）

            k = int(st.poisson.ppf(hi\_Sp,mu=4)) # 参数值 mu=3 的泊松分布在 2 处的概率密度值

            print(k)

            # 聚类开始

            km = cluster.KMeans(n\_clusters=k, init='k-means++', max\_iter=10, n\_init=1)

            km.fit(data)           # 调用该对象的聚类方法

            Ck\_2d = km.cluster\_centers\_   # 计算类中心

            Ck= np.average(Ck\_2d,axis=1)

            #result[t-0] = np.dot(W[0:k],np.array(Ck).reshape(k,1))   # 可加传感器精确度（权值）

            p = 0   # 后验概率（posterior）

            result[t-p] = np.average(Ck)   # 简单化均值就行

        t += 1    # 每次往前走一步

    plt.figure()

    plt.rcParams['font.sans-serif']=['SimHei']

    plt.rcParams['axes.unicode\_minus'] = False

    plt.tick\_params(labelsize=14)

    plt.title('滑动聚类融合',fontsize=18)

    plt.xlabel('时间(s))',fontsize=18)

    plt.ylabel('Z轴加速度((m/s)/s)',fontsize=18)

    colors=['b','g','y','c']

    markers=['o','s','D','+']

    #for i, j in enumerate(km.labels\_):

        #plt.scatter(data[i][0], data[i][1], color=colors[j], marker=markers[j], s=5)

    for i in range(N):

        plt.plot(dataset[:,0], dataset[:,i+1], color=colors[i])

    plt.plot(dataset[:,0], result[:,0], 'r')

    plt.legend(['传感器1','传感器2','传感器3','传感器4','本文算法处理'],\

                       fontsize = 18, loc='best')

    plt.show()

    end\_time = time()

    run\_time = end\_time - begin\_time

    print('Working is over! step\_running time:', run\_time/seq)