**程序报告**

学号：2013536 姓名：汤清云

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

1. 使用Kmeans，PCA算法完成异常点检测。
2. 在当前特征下，我们将数据聚集于一个聚类中，尝试使用计算当前点（cpc，cpm）到平均值点（cpc\_mean，cpm\_mean）的几何距离（L2 范数）寻找聚类中的异常点。
3. 设置一定的异常点比例（ratio = 0.005），根据距离的大小进行划分，将属于该部分的点设置为异常点。
4. PCA 通过对数据特征的变换，寻找特征空间中，数据分布方差最大的方向，称为特征方向或主成分方向，选择其中特征值较大的几个特征方向，将数据点投影到这些方向上，完成数据降维。
5. KMeans 算法的思想很简单，按照样本之间的距离大小，将样本集划分为 K 个簇。  
   让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。
6. 第一步根据当前各个簇中心，计算每个数据点与各个簇中心的距离，将该点划分为距离簇中心最近的簇类别；第二步根据重新划分的簇，更新每个簇的中心位置。直至簇中心不再变化获得迭代次数超过我们所预定的值时停止优化
7. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

采用方法：迭代。

改进： 无

局限性： 需要手动调整参数以达到最优比例，无法自我更正。

1. **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

====================================================================

import sys

import numpy as np

import pandas as pd

from copy import deepcopy

from matplotlib import pyplot as plt

import warnings

import numpy as np

import scipy.sparse as sp

class KMeans():

"""

Parameters

----------

n\_clusters 指定了需要聚类的个数，这个超参数需要自己调整，会影响聚类的效果

n\_init 指定计算次数，算法并不会运行一遍后就返回结果，而是运行多次后返回最好的一次结果，n\_init即指明运行的次数

max\_iter 指定单次运行中最大的迭代次数，超过当前迭代次数即停止运行

"""

def \_\_init\_\_(

self,

n\_clusters=8,

n\_init=10,

max\_iter=300

):

self.n\_clusters = n\_clusters

self.max\_iter = max\_iter

self.n\_init = n\_init

def calculate\_distance\_multi\_dims(self,sample, centroid, axis=1, method='euclidean'):

"""

计算样本与聚类中心的欧氏距离（默认）

----

axis = 1，单次计算多个样本之间的欧氏距离

axis = 0，单次计算两个样本之间的欧氏距离

"""

return np.linalg.norm(sample - centroid, axis=axis)

def do\_init\_centroids(self,X, k, method='random'):

"""

初始化聚类中心

----

X: array, dataset

k: int, cluster number

method: str, 'random'(k-means); 'k-means++'

----

KMeans++改进了KMeans算法选择初始质心的方式。

其核心思想是：在选择一个聚类中心时，距离已有的聚类中心越远的点，被选取作为聚类中心的概率越大。

"""

# 样本个数

n\_samples = X.shape[0]

# 样本特征数

n\_features = X.shape[1]

# 生成样本索引

indexs = np.arange(0, n\_samples)

# 打乱顺序

# 此函数仅沿多维数组的第一个轴对数组进行打乱。子数组的顺序改变，但内容不变。

np.random.shuffle(indexs)

# 初始化聚类簇中心，shape=(k, n\_features)

centroids = np.zeros((k, n\_features))

# 类型转换，统一格式为 numpy.array

if type(X) == pd.core.frame.DataFrame:

X = X.to\_numpy()

if method == 'k-means++':

# 从数据集中随机选择一个样本点作为第一个聚类中心

centroids[0, :] = X[indexs[0], :]

print(centroids.shape)

# 从剩余样本中选择 k - 1 个聚类中心

for centroid in range(k - 1):

# 定义一个列表存储离聚类中心最近的样本点

dists = []

for i in range(n\_samples):

# 单一样本

point = X[i, :]

# 初始化距离

min\_dist = sys.maxsize

# 计算 point 与之前的每一个聚类中心的距离

# 选择质心并存储最小距离

for j in range(len(centroids)):

# temp\_dist = calculate\_distance\_multi\_dims(point, centroids[j], axis=0)

temp\_dist =self.calculate\_distance(point, centroids[j], method='euclidean', p=None)

# 存储最小距离

min\_dist = min(min\_dist, temp\_dist)

dists.append(min\_dist)

# 遍历完样本之后，选择距离最大的数据点作为下一个质心

max\_dist = np.argmax(np.array(dists))

next\_centroid = X[max\_dist, :]

# 存储第二个及其之后的聚类中心

centroids[centroid+1, :] = next\_centroid

# dists 清零

dists = []

# 随机初始化：即随机从样本中选择 k 个样本点作为初始聚类中心

else:

# 取打乱顺序之后的前 k 个样本作为初始聚类中心

top\_k\_index = indexs[:k]

# 用这k个样本的值作为初始化的簇中心

centroids = X[top\_k\_index, :]

return centroids

def k\_means(self,X, n\_cluster, init\_method='random', n\_iter=100, plot\_process=False):

init\_centroids = self.do\_init\_centroids(X, k=n\_cluster, method=init\_method)

# print(init\_centroids.shape)

# 类型转换，统一格式为 numpy.array

if type(X) == pd.core.frame.DataFrame:

X = X.to\_numpy()

# 用于保存聚类中心更新前的值

old\_centroids = np.zeros(init\_centroids.shape)

# print(old\_centroids.shape)

# 更新后的聚类中心的值

new\_centroids = deepcopy(init\_centroids)

# 用于保存数据所属的簇

n\_samples = len(X)

clusters = np.zeros(n\_samples)

# 迭代标识符，计算新旧聚类中心的距离

distance\_flag =self. calculate\_distance\_multi\_dims(init\_centroids, old\_centroids, axis=1)

current\_iter = 1

iteration\_flag = (current\_iter < n\_iter)

# 若聚类中心不再变化或者迭代次数超过n\_iter次(可取消)，则退出循环

while distance\_flag.any() != 0 and iteration\_flag:

# 1. 计算每个样本点所属的簇（距离最近的簇）

for i in range(n\_samples):

# 样本与k个聚类中心的距离

distances =self. calculate\_distance\_multi\_dims(X[i], new\_centroids, axis=1)

# 当前样本与k个聚类中心的最近距离

cluster = np.argmin(distances)

# 记录当前样本点所属的聚类中心

clusters[i] = cluster

# 2. 更新聚类中心

# 记录更新前的聚类中心

old\_centroids = deepcopy(new\_centroids)

# 属于同一个簇的样本点放到一个数组中，然后按照列的方向取平均值

for i in range(n\_cluster):

points = [X[j] for j in range(len(X)) if clusters[j] == i]

new\_centroids[i] = np.mean(points, axis=0)

# 3. 判断是否满足迭代停止条件

#if current\_iter % 5 == 0:

# print(f"[INFO] Iteration {current\_iter}：distance\_flag = {distance\_flag}.")

distance\_flag =self. calculate\_distance\_multi\_dims(new\_centroids, old\_centroids, axis=1)

current\_iter += 1

iteration\_flag = (current\_iter <= n\_iter)

#if plot\_process: # 如果绘制图像

# plt = plot\_cluster\_process\_2d(X, new\_centroids,old\_centroids) # 画聚类中心的移动过程

#if plot\_process: # 显示最终的绘制结果

# plt.show()

# 返回每个样本所属的类以及更新后的聚类中心

return clusters, new\_centroids

def fit(self, x):

"""

用fit方法对数据进行聚类

:param x: 输入数据

:best\_centers: 簇中心点坐标 数据类型: ndarray

:best\_labels: 聚类标签 数据类型: ndarray

:return: self

"""

###################################################################################

#### 请勿修改该函数的输入输出 ####

###################################################################################

# #

self.labels\_,self.cluster\_centers\_ = self.k\_means(x, self.n\_clusters,'random', self.max\_iter,False)

return self

# #

###################################################################################

############# 在生成 main 文件时, 请勾选该模块 #############

###################################################################################

#best\_centers = []

#best\_labels = []

#self.cluster\_centers\_ = best\_centers #cluster\_centers\_[labels\_[i],:n\_features]，令j=labels\_[i]，则j有 n\_clusters个，由下标j可索引出聚类中心的属性，由nfeatures来限定索引属性的个数

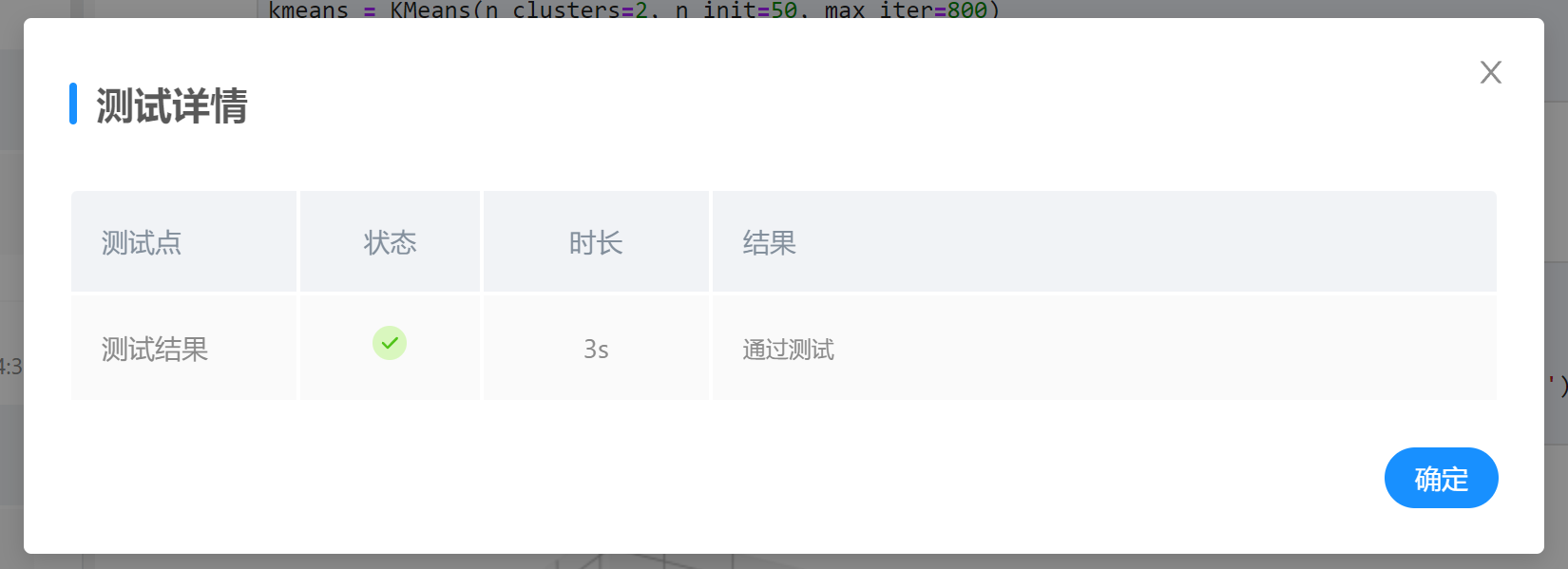
#self.labels\_ = best\_labels #labels\_有1357个，labels\_[i]表示第i个数据所属的类别,i从0~1356

#return self

1. **实验结果**

（实验结果，必填）

====================================================================



1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

遇到困难：

1. PCA算法较为困难，对PCA算法的python实现不熟悉。
2. K-means方法的实现与python自带包中代码差异较大，修改时遇到许多困难。
3. 参数n\_clusters进行多次调整，尚未摸索出最佳取值。