**程序报告**

学号：2110957 姓名：蒋薇

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）

KMeans实现异常点检测,网络广告运营商需要依照数据，向广告投放方收取费用。  
但数据中某些时间点的数据可能存在造假的情况，某些点可能是运营商方面制造的虚假流量，我们的主要任务就是检测出这些异常的数据点。(实验使用的数据集用于异常点检测，数据集统计的是 2011 年 7 月 至 2011 年 9 月 时间段内，某网络广告的综合的曝光率,数据集依照时间顺序，分别统计了cpm、cpc两种指标。)

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

1. **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**）

数据的分布并不只有一个簇，在寻找异常点时候，应该首先将所有数据点分为多个簇，再计算各个簇中显著远离簇中心的点，作为数据中的异常点。

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。

Kmeans 方法中，每个数据点具有两个属性：

该点所属的簇

该点与各个簇中心点的距离

每一个点找到距离自己最近的簇，而且不同簇之间距离尽量大

KMeans 的分步优化的方法：

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时停止优化。

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

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）

def preprocess\_data(df):

df['timestamp'] = pd.to\_datetime(df['timestamp'])

# 将 df 数据按时间序列排序，方便数据展示

df = df.sort\_values(by='timestamp').reset\_index(drop=True)

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

df[['cpc','cpm']] = scaler.fit\_transform(df[['cpc','cpm']])

# 尝试引入非线性关系

df['cpc X cpm'] = df['cpm'] \* df['cpc']

df['cpc / cpm'] = df['cpc'] / df['cpm']

# 尝试获取时间关系

df['hours'] = df['timestamp'].dt.hour

df['daylight'] = ((df['hours'] >= 7) & (df['hours'] <= 22)).astype(int)

from sklearn.decomposition import PCA

#在进行特征变换之前先对各个特征进行标准化

columns = ['cpc', 'cpm', 'cpc X cpm','daylight']

data = df[columns]

scaler = StandardScaler()

data = scaler.fit\_transform(data)

data = pd.DataFrame(data, columns=columns)

#通过 n\_components 指定需要降低到的维度

n\_components = 3

pca = PCA(n\_components=n\_components)

data = pca.fit\_transform(data)

data = pd.DataFrame(data,columns=['Dimension' + str(i+1) for i in range(n\_components)])

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

def get\_distance(data, kmeans, n\_features):

distance = []

for i in range(0,len(data)):

point = np.array(data.iloc[i,:n\_features])

center = kmeans.cluster\_centers\_[kmeans.labels\_[i]]

distance.append(np.linalg.norm(point - center))

distance = pd.Series(distance)

return distance

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

def get\_anomaly(data, kmean, ratio):

from copy import deepcopy

ratio = 0.03

num\_anomaly = int(len(data) \* ratio)

new\_data = deepcopy(data)

new\_data['distance'] = get\_distance(new\_data,kmeans,n\_features=len(new\_data.columns))

threshould = new\_data['distance'].sort\_values(ascending=False).reset\_index(drop=True)[num\_anomaly]

# 根据阈值距离大小判断每个点是否是异常值

new\_data['is\_anomaly'] = new\_data['distance'].apply(lambda x: x > threshould)

normal = new\_data[new\_data['is\_anomaly'] == 0]

anormal = new\_data[new\_data['is\_anomaly'] == 1]

return data

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

def predict(preprocess\_data):

# 异常值所占比率

ratio = 0.03

# 加载模型

kmeans = joblib.load('./results/model.pkl')

# 获取异常点数据信息

is\_anomaly = get\_anomaly(preprocess\_data, kmeans, ratio)

return is\_anomaly, preprocess\_data, kmeans, ratio

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

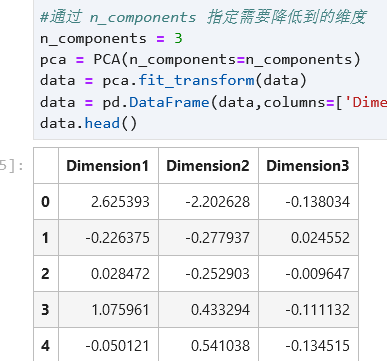
1. **实验结果**

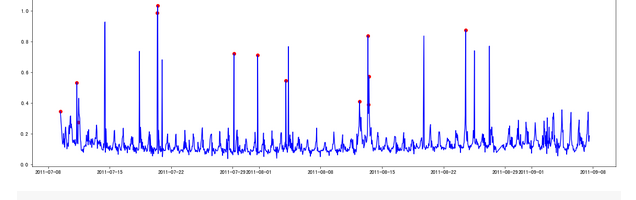
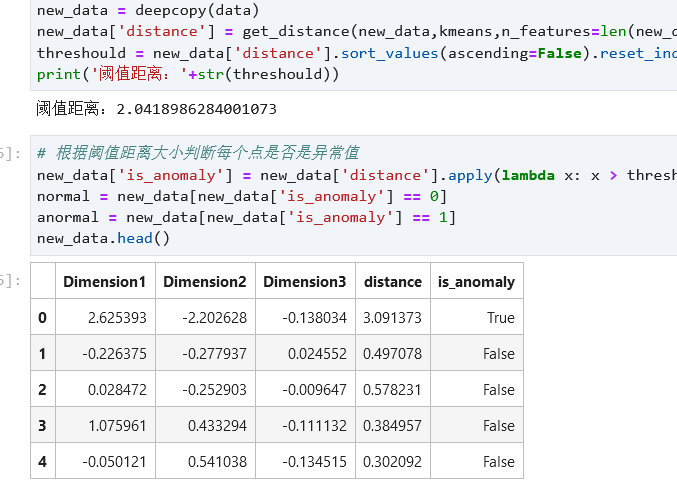
（实验结果，必填）

提交实验结果：



各步骤：





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

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**）

calinski\_harabasz\_score：通过计算簇中各点与簇中心的距离平方和来度量簇内的紧密度，通过计算各簇中心点与数据集中心点距离平方和来度量数据集的分离度，由分离度与紧密度的比值得到。即该指标越大代表着簇自身越紧密，簇与簇之间越分散，即聚类结果越好。

silhouette\_score(轮廓系数)结合了凝聚度和分离度，其计算步骤如下：

对于第 i 个对象，计算它到所属簇中所有其他对象的平均距离，记 ai （体现凝聚度）

对于第 i 个对象和不包含该对象的任意簇，计算该对象到给定簇中所有对象的平均距离，记 bi （体现分离度）

第 i 个对象的轮廓系数为 si = (bi-ai)/max(ai, bi)

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