**程序报告**

学号： 2013750 姓名：管昀玫

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）

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

异常值检测（outlier detection ）是一种数据挖掘过程，用于发现数据集中的异常值并确定异常值的详细信息。

当前数据容量大、数据类型多样、获取数据速度快；但是数据也比较复杂，数据的质量有待商榷；而数据容量大意味着手动标记异常值成本高、效率低下；因此能够自动检测异常值至关重要。

自动异常检测具有广泛的应用，例如信用卡欺诈检测、系统健康监测、故障检测以及传感器网络中的事件检测系统等。

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**）

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

使用PCA降维，使用KMeans完成异常点检测。

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。

我们使用 KMeans 聚类方法完成检测，在 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

我们希望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。因此出现了 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。

对于当前产生的聚类，由于缺失标签，我们可以使用一些其他的方式对聚类效果进行评分，一般有 `calinski\_harabasz\_score`、`silhouette\_score` 两种评价方式。

`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)

从上面可以看出，轮廓系数取值为 [-1, 1]，其值越大越好，且当值为负时，表明 ai < bi，样本被分配到错误的簇中，聚类结果不可接受。对于接近 0 的结果，则表明聚类结果有重叠的情况。

除此之外，还可以实现preprocess\_data/get\_distance/get\_anomaly/predict等函数，调整参数使模型更具有泛化能力。

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）

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

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 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 ####

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

# #

x1=np.array(x)

score1=0

score2=0

tmp\_centers=[]

tmp\_labels=[]

for init in range(self.n\_init):

#聚类中心随机初始化,centers[0]->(x,x,x)

self.centers=[]

init\_indices = np.random.randint(0,x1.shape[0],self.n\_clusters)

tmp = x1[init\_indices]

seed = np.random.permutation(tmp.shape[0])[:self.n\_clusters]

self.centers = tmp[seed]

#print(self.centers)

#print(self.centers)

#self.label=[]

# for i in range (self.n\_clusters):

# self.centers[i]=x1[i]#怎么随机？

# self.means[i]=[]

self.means={}#聚类，means[i]为第i个聚类中的点集

#计算距离，更新中心

for i in range (self.max\_iter):

for j in range (self.n\_clusters):

self.means[j]=[]

for point in x1:

distance=[]

for center in self.centers:

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

cen=distance.index(min(distance))

self.means[cen].append(point)

#self.label.append(cen)#分label

prev\_centers=deepcopy(self.centers)

for k in range(len(self.centers)):

if(len(self.means[k])!=0):

self.centers[k]=np.average(self.means[k],axis=0)

#找label

self.label=[]

for point in x1:

for k in self.means:

for p in self.means[k]:

if (point==p).all():

self.label.append(k)

tmp\_score1 = round(calinski\_harabasz\_score(x,self.label), 2)

tmp\_score2 = round(silhouette\_score(x,self.label), 2)

if tmp\_score2>=score2 and tmp\_score1>=score1:

score1=tmp\_score1

score2=tmp\_score2

tmp\_centers=deepcopy(self.centers)

tmp\_labels=deepcopy(self.label)

#精度

# optimized = True

# for center in self.centers:

# org\_centers = prev\_centers[center]

# cur\_centers = self.centers[center]

# if np.sum((cur\_centers - org\_centers) / org\_centers \* 100.0) > 0.00001:

# optimized = False

# if optimized:

# break

# else:

# #

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

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

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

best\_centers = np.array(tmp\_centers)

# print(best\_centers)

# print('\n')

#print(x1[0:5])

self.cluster\_centers\_ = best\_centers

# print(self.means)

self.labels\_ = np.array(tmp\_labels)

# print(self.labels\_.shape)

print(score1)

print(score2)

return self

def preprocess\_data(df):

"""

数据处理及特征工程等

:param df: 读取原始 csv 数据，有 timestamp、cpc、cpm 共 3 列特征

:return: 处理后的数据, 返回 pca 降维后的特征

"""

# 请使用joblib函数加载自己训练的 scaler、pca 模型，方便在测试时系统对数据进行相同的变换

# ====================数据预处理、构造特征等========================

# 例如

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

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

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

# df 作为最后输出的 DataFrame 初始化为空

# ======================== 模型加载 ===========================

# 请确认需要用到的列名，e.g.:

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

data = df[columns]

scaler = StandardScaler()

data = scaler.fit\_transform(data)

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

scaler = joblib.load('./results/scaler.pkl')

pca = joblib.load('./results/pca.pkl')

data = scaler.transform(data)

#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)])

return data

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

"""

计算距离函数

:param data: 训练 kmeans 模型的数据

:param kmeans: 训练好的 kmeans 模型

:param n\_features: 计算距离需要的特征的数量

:return: 每个点距离自己簇中心的距离

"""

distance = []

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

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

center = kmeans.cluster\_centers\_[kmeans.labels\_[i],:n\_features]

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

distance = pd.Series(distance)

return distance

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

"""

检验出样本中的异常点，并标记为 True 和 False，True 表示是异常点

:param data: preprocess\_data 函数返回值，即 pca 降维后的数据，DataFrame 类型

:param kmean: 通过 joblib 加载的模型对象，或者训练好的 kmeans 模型

:param ratio: 异常数据占全部数据的百分比,在 0 - 1 之间，float 类型

:return: data 添加 is\_anomaly 列，该列数据是根据阈值距离大小判断每个点是否是异常值，元素值为 False 和 True

"""

# ====================检验出样本中的异常点========================

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

new\_data = deepcopy(data)

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

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

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]

#data['is\_anomaly'] = None

return new\_data

def predict(preprocess\_data):

"""

该函数将被用于测试，请不要修改函数的输入输出，并按照自己的模型返回相关的数据。

在函数内部加载 kmeans 模型并使用 get\_anomaly 得到每个样本点异常值的判断

:param preprocess\_data: preprocess\_data函数的返回值，一般是 DataFrame 类型

:return:is\_anomaly:get\_anomaly函数的返回值，各个属性应该为（Dimesion1,Dimension2,......数量取决于具体的pca），distance,is\_anomaly，请确保这些列存在

preprocess\_data: 即直接返回输入的数据

kmeans: 通过joblib加载的对象

ratio: 异常点的比例，ratio <= 0.03 返回非异常点得分将受到惩罚！

"""

# 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

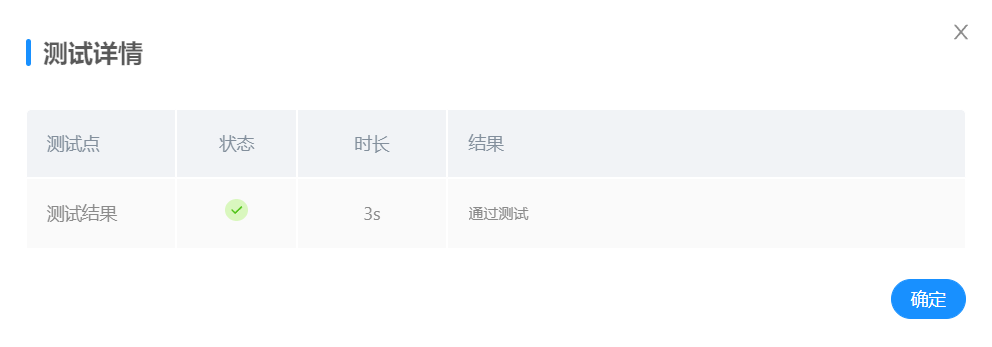
is\_anomaly = get\_anomaly(preprocess\_data, kmeans, ratio)

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

1. **实验结果**

（实验结果，必填）

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



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**）

在n\_clusters=3, n\_init=20, max\_iter=800时，calinski\_harabasz\_score为367.47，silhouette\_score为0.81，还算合理；

寻找最佳聚类数目的结果：

459.56

0.8

聚类数目:2 calinski\_harabasz\_score:459.56 silhouette\_score:0.8

339.4

0.58

聚类数目:3 calinski\_harabasz\_score:339.4 silhouette\_score:0.58

534.16

0.67

聚类数目:4 calinski\_harabasz\_score:534.16 silhouette\_score:0.67

1082.05

0.67

聚类数目:5 calinski\_harabasz\_score:1082.05 silhouette\_score:0.67

1282.6

0.58

聚类数目:6 calinski\_harabasz\_score:1282.6 silhouette\_score:0.58

1248.77

0.49

聚类数目:7 calinski\_harabasz\_score:1248.77 silhouette\_score:0.49

1179.71

0.45

聚类数目:8 calinski\_harabasz\_score:1179.71 silhouette\_score:0.45

1110.75

0.42

聚类数目:9 calinski\_harabasz\_score:1110.75 silhouette\_score:0.42