**程序报告**

学号： 2011937 姓名：姜志凯

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。

本实验将运用KMeans算法完成异常点检测。

本次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。网络广告运营商需要依照这些数据，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。不同于监督学习，该数据集中异常数据值并没有进行标注，即我们并不知道哪些数据点是异常点。因此，可以用无监督学习KMeans算法实现。

KMeans算法通过反复循环迭代，将数据集分为k个类别，先选择k个对象作为每个类的中心，然后对每一个点计算它到每个类心的距离，选择距离最小的那个类心，作为类的归属，将所有的点都分完之后，在每个类内重新选类心，再重复对每个点计算到每个新类心的距离，重新分类，再更新类心，直至类心不变。即尽量使类间分散、类内紧凑，最终寻找一个最好的分类方法，将数据集分类。对于某个测试样例，通过某种计算（如向量间的距离），对样例进行分类。对于异常值检测，划定一个异常域，超出这个域的样例划为异常值，或者假定一个异常值的比例，按比例划分异常值（距离平均点的距离越大，异常的可能性就越大）。

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**）

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

1. 数据预处理：

先读取数据，原始数据有cpc、cpm两个指标，仅仅使用两个特征是不够的，所以引进其他线性和非线性的特征，而且异常值的出现也可能与时间有关，可能每一次产生异常点的时间段总是集中在某一个时间点，或者在更大的粒度上，可能异常总是出现在白天或者晚上，因此向原来的数据集中添加有关时间信息的特征，可以提供额外的时间信息，帮助算法寻找异常点在空间维度上的信息。本实验加了cpc\*cpm和采集时间和白天/黑天三个特征。

df = df[['cpc','cpm','cpc X cpm','hours','daylight']]

1. PCA降维：

加上上述特征之后，维度大于3，无法观察数据的分布以及分多少个簇合适等。而且以上各个特征之间并不是相互独立的，存在一定程度上的相关性；故我们采用 PCA (主成分分析)来寻找数据集的低维度表达。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。

n\_components = 3

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

1. 运用KMeans算法进行聚类：

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

编写\_init\_函数，对默认参数进行初始化

n\_clusters=8, #即我们的k值，一般需要多试一些值以获得较好的聚类效果

n\_init=10, #用不同的初始化质心运行算法的次数

max\_iter=300, #运行一次算法的最大的迭代次数

编写random\_center(self, data)函数，随机初始化类心；

编写dis(self, v1, v2)函数，计算两个对象间的距离；

编写fit（self，x）函数对数据进行聚类：

初始化类心，然后根据各点到各类心的距离分类，然后对每个类内的类心进行更新，得到新的类心，重读计算距离，更新类心，直到类心不变。

整个过程重复n\_init遍，找到最佳分类结果

聚类评价指标有calinski\_harabasz\_score、silhouette\_score两种，尝试不同的类别数K，使得两个指标最优即可。

1. 异常检测：

假定一个异常值的比例ratio，得到异常域，距离超过异常域的样例即为异常值

preprocess\_data(df)#数据预处理及降维

get\_distance(data, kmeans, n\_features)#计算距离

get\_anomaly(data, kmeans, ratio)#运用计算距离的函数，得到异常值

predict(preprocess\_data)#预测

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 numpy as np

import os

import sklearn

import warnings

from copy import deepcopy

import random

import matplotlib

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import calinski\_harabasz\_score,silhouette\_score

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, #即我们的k值，一般需要多试一些值以获得较好的聚类效果

n\_init=10, #用不同的初始化质心运行算法的次数

max\_iter=300, #运行一次算法的最大的迭代次数

#init='k-means++', #初始值选择的方式

#algorithm='auto' #有“auto”, “full” or “elkan”三种选择,一般数据是稠密的，那么就是 “elkan”，否则就是"full","auto"根据数据自动选择

):

self.n\_clusters = n\_clusters

self.max\_iter = max\_iter

self.n\_init = n\_init

#self.init='k-means++'

#self.algorithm='auto'

def random\_center(self, data):

num, dimension = data.shape

# 创建 n 行 d 列的数组

cent = np.zeros((self.n\_clusters, dimension))

random.seed(48)

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

index = random.randint(0, num - 1)

cent[i, :] = data[index, :]

return cent

def dis(self, v1, v2):

return np.sqrt(sum(pow(v2 - v1, 2)))

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

"""

x = x.values

num = x.shape[0]

score=0

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

#初始化聚类心 centers

self.now\_centers\_ = np.zeros((self.n\_clusters, x.shape[1]))

cluster\_mark = np.mat(np.zeros((num, 2)))

self.now\_centers\_ = self.random\_center(x)

updating = True

times = 0

while updating:

times += 1

##超过最大迭代次数，直接停止

if times > self.max\_iter:

break

updating = False

for i in range(num):

min\_dis = 99999999.0

min\_cluster = 0

#计算该点与其余所有簇类中心点的距离

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

distance = self.dis(self.now\_centers\_[j, :], x[i, :])

if distance < min\_dis:

min\_dis = distance

min\_cluster = j

if cluster\_mark[i,0] != min\_cluster:

updating = True

cluster\_mark[i, :] = min\_cluster, min\_dis \*\* 2

##更新簇类中心

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

pointsInCluster = x[np.nonzero(cluster\_mark[:, 0].A == j)[0]]

self.now\_centers\_[j, :] = np.mean(pointsInCluster, axis=0)

new\_labels=[]

cluster\_mark[:, 0]

for i in range(num):

new\_labels.append(int(cluster\_mark[i, 0]))

if silhouette\_score(x,new\_labels)>score:

score=silhouette\_score(x,new\_labels)

best\_center=deepcopy(self.now\_centers\_)

best\_labels=deepcopy(new\_labels)

self.cluster\_centers\_ = best\_center

self.labels\_ = best\_labels

return self

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

# 寻找最佳聚类数目

from sklearn.metrics import calinski\_harabasz\_score,silhouette\_score

'''

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

init 指明初始聚类中心点的初始化方式，kmeans++是一种初始化方式，还可以选择为random

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

'''

score1\_list = []

score2\_list = []

for i in range(2,10):

kmeans = KMeans(n\_clusters=i,n\_init=10,max\_iter=100)

kmeans.fit(data)

score1 = round(calinski\_harabasz\_score(data,kmeans.labels\_), 2)

score2 = round(silhouette\_score(data,kmeans.labels\_), 2)

score1\_list.append(score1)

score2\_list.append(score2)

print('聚类数目:%s calinski\_harabasz\_score:%-10s silhouette\_score:%-10s'%(i,score1,score2))

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

import os

import sklearn

import numpy as np

import pandas as pd

import random

from copy import deepcopy

from sklearn.externals import joblib

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

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

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

# 请确认需要用到的列名，e.g.:columns = ['cpc','cpm']

# 例如

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

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

# data = scaler.transform(data)

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

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

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

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

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

data = df[columns]

scaler = StandardScaler()

data = scaler.fit\_transform(data)

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

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):

"""

计算样本点与聚类中心的距离

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

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

"""

# ====================计算样本点与聚类中心的距离======================== #

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, kmeans, 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,kmeans,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]

return anormal

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

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

#KMeans=joblib.load('./results/model.pkl')

kmeans = KMeans(n\_clusters=3, n\_init=10, max\_iter=800)

kmeans.fit(preprocess\_data)

# 获取异常点数据信息

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

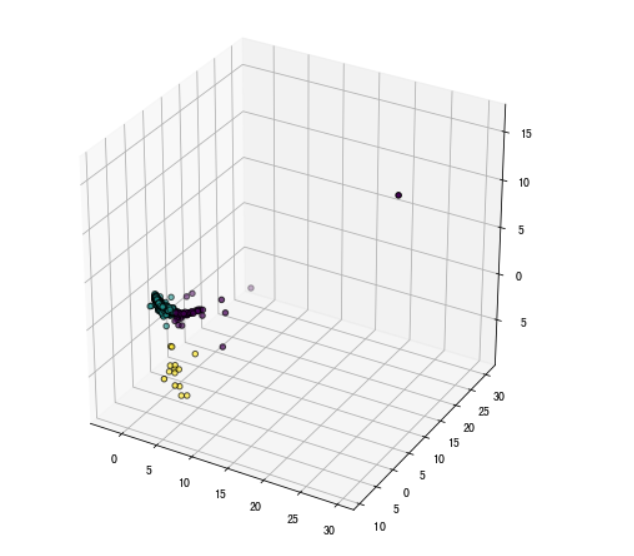
return is\_anomaly, preprocess\_data, kmeans, ratio

1. **实验结果**

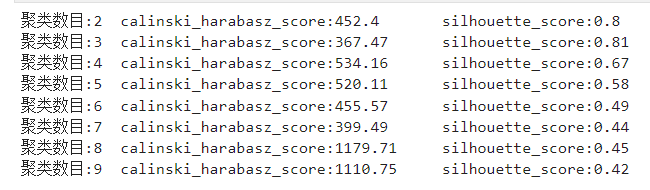
（实验结果，必填）

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

1. 聚类结果

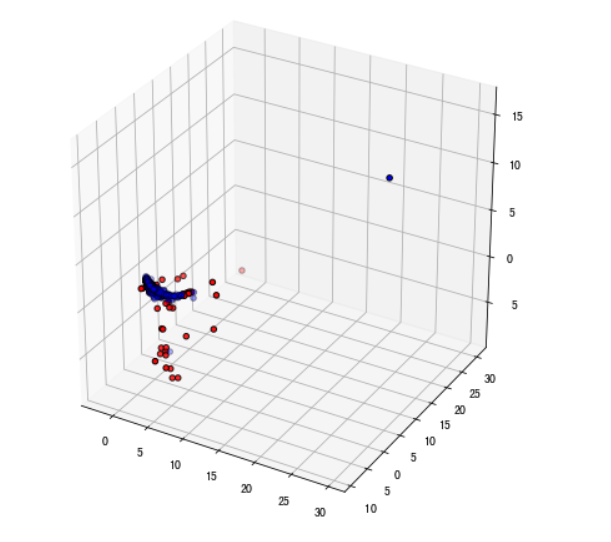


1. 尝试不同的簇数K：

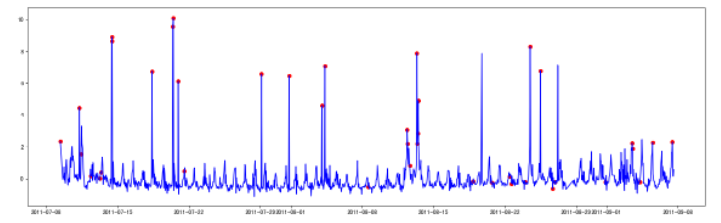


根据两个指标，选择聚类数目3

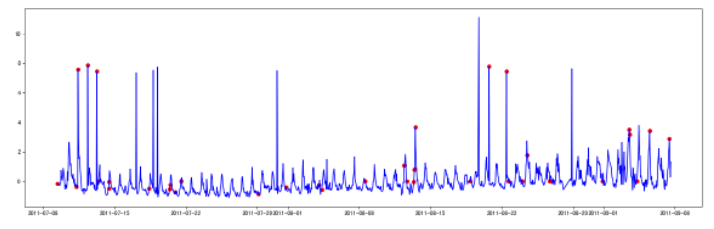
1. 异常值检测结果：



1. 聚类后cpc的异常点：



1. 聚类后cpm的异常点：



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**）

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

值明显偏离正常值的点基本都标记出了，大体达到目标预期。

困难：我的不能在聚类的各类心不变后及时停止循环，只能达到最大迭代次数后停止，导致运行较慢，但不影响结果。

KMeans算法各参数：

n\_clusters=8, #即我们的k值，一般需要多试一些值以获得较好的聚类效果

n\_init=10, #用不同的初始化质心运行算法的次数

max\_iter=300, #运行一次算法的最大的迭代次数

#init='k-means++', #初始值选择的方式，'k-means++'效果较好

#algorithm='auto' #有“auto”, “full” or “elkan”三种选择,一般数据是稠密的，那么就是 “elkan”，否则就是"full","auto"根据数据自动选择

通过多尝试可以达到n\_clusters参数的最优；

用不同的初始化质心运行算法的次数n\_init，一般10次足够得到最佳的初始化类心；

max\_iter运行一次算法的最大的迭代次数，多一点可能会优化，到一定次数类心不再改变即可，稍微大一点就行；

自己编写的算法，初始化类心是随机的，到我觉得python-sklearn包里的init='k-means++'初始化方式可能会更优；

algorithm='auto'可以根据数据的稀疏程度，自动选择算法，也可以优化。