**K-Means异常检测问题程序报告**

学号2112213姓名：冯思程

1. **问题重述**

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。

实验要求：

1) 了解 KMeans、PCA 算法，了解算法的基本原理。

2）运用 KMeans 算法完成异常点检测。

实验环境：

可以使用基于 Python 的 Numpy 等库进行数据处理，使用 sklearn 等框架建立深度学习模型，使用过程中请注意 Python 包（库）的版本。

构造非线性组合额外特征，同时考虑到异常程序的位置，帮助算法寻找异常点在时间维度上的信息，并使用 PCA 算法进行特征降维，最后使用 Kmeans 聚类算法完成异常点检测。要对异常值检测的性能进行改进。

1. **设计思想**

**方法：**

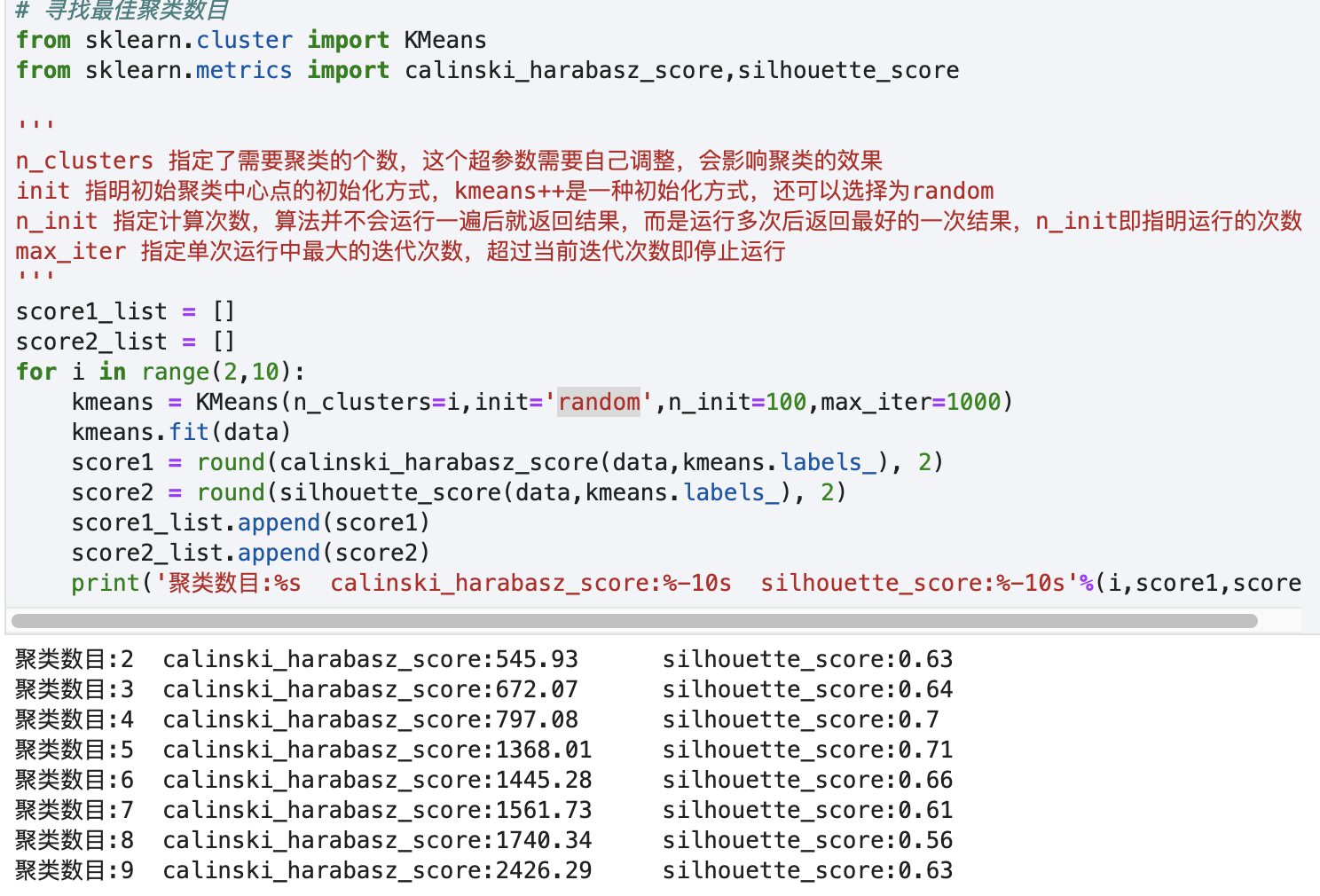
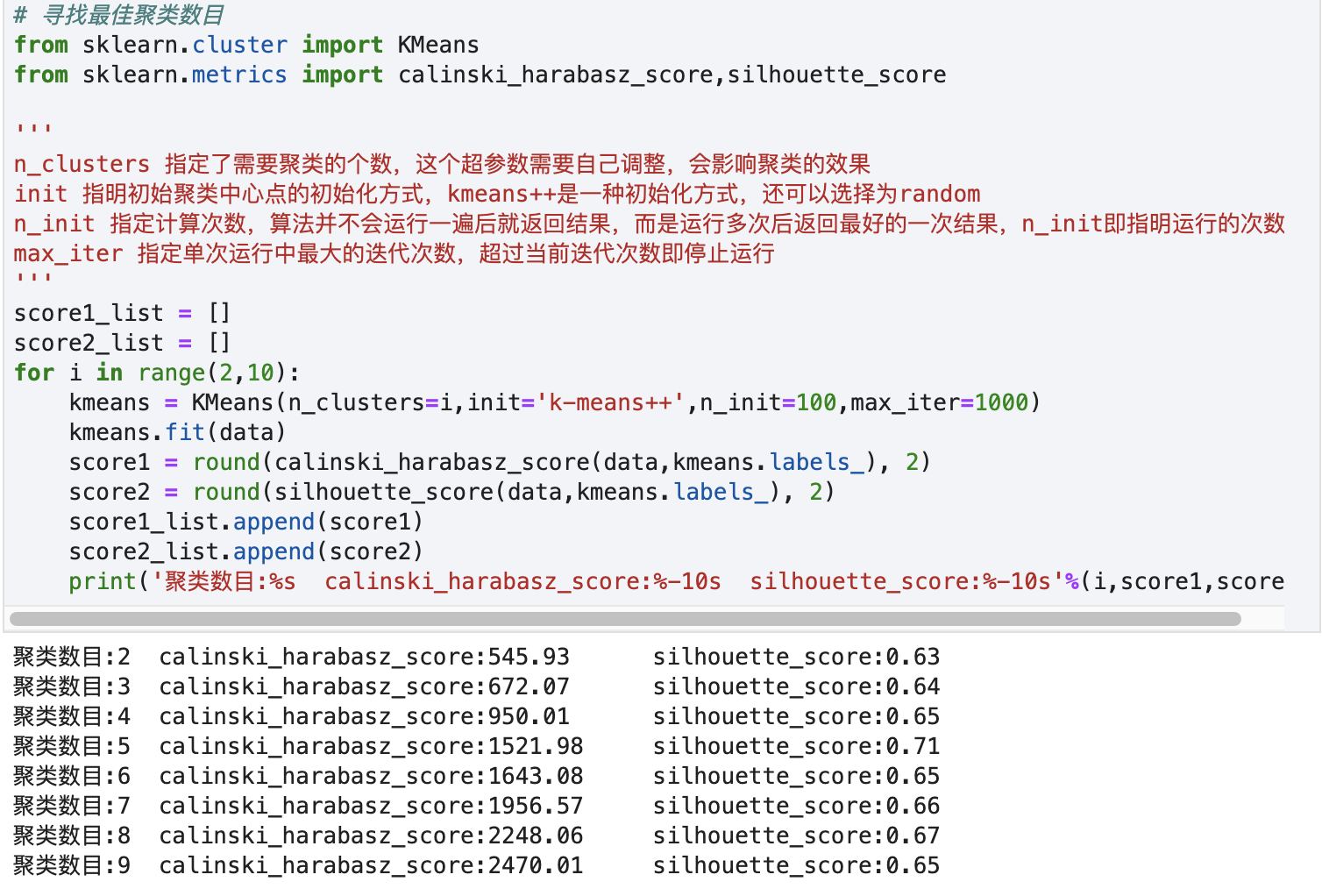
方法首先对数据进行预处理和特征构造，然后因为特征之间并不独立，所以利用PCA算法对特征组合进行降维，然后利用KMeans算法寻找异常值点，将样本集划分为 K 个簇。

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。然后对模型性能进行评价，判断模型的性能优劣。最后通过测试测试检测异常数据点的比例。

**优化方向：**

这里我考虑到的优化方向有如下：

1. 特征构造的时候进行优化：在最开始的两个线性的特征值基础上增加了非线性关系cpc X cpm和cpc / cpm，并向原来的数据集中添加有关时间信息的特征 daylight，然后通过排列组合对性能进行遍历测试挑选最优模型，最后选择了加入cpc X cpm和cpc / cpm、daylight三个特征。
2. 然后是PCA算法方面，经过简单的测试综合考虑，利用StandardScaler 对数据进行了归一化并使用 PCA 算法将特征降至 3 维，三维就可以表达80%以上的数据特征，这个降维步骤的目标维数不进行修改。
3. 然后是kmeans算法的参数优化：
4. 对n\_init，max\_iter两个参数进行优化，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。这里将n\_init设置成100,max\_iter设置成1000。
5. 然后是对初始化方式进行优化，我这里对照的比较了采用random初始化和k-means++初始化，对比图如下，于是我采用了k-means++初始化方法：



1. 对聚类数目进行优化，根据提供的寻找最佳聚类个数的函数，经过遍历测试比较后，将聚类个数设置成6，此时在测试结果下表现比较好，测试结果为4/5。

**局限性：**

1.需要预先指定簇的个数K，这个参数的选择对聚类结果有很大的影响，且需要手动调参。

2.对初始质心的选择非常敏感，初始质心的选择会影响最终的聚类结果。

3.对于离群点非常敏感，离群点可能会影响质心的计算，从而影响最终的聚类结果。

4.对于非凸数据集的聚类效果不佳。

5.KMeans算法不适用于处理文本数据、时间序列数据等类型的数据，这些类型的数据不具有明显的距离和几何结构特征。

4.伪代码：

①数据处理与特征构造，构造出可能影响检测效果的特征指标。

②数据归一化处理，把数据处理为五维。

③PCA 降维，降维数目为 3

④KMeans 算法聚类

⑤保存模型至 result 文件夹

⑥调用测试函数

1. **代码内容**

**Train.py**的代码是**：**

import os

import pandas as pd

from sklearn.externals import joblib

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.preprocessing import MinMaxScaler

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

import numpy as np

from copy import deepcopy

import matplotlib.pyplot as plt

file\_dir = './data'

# 读取数据并合并

df\_features = []

for col in ('cpc', 'cpm'):

path = os.path.join(file\_dir, col + '.csv')

df\_feature = pd.read\_csv(path)

df\_features.append(df\_feature)

df = pd.merge(left=df\_features[0], right=df\_features[1])

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

# 特征工程

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

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

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

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

# 特征标准化

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

data = df[columns]

scaler = StandardScaler()

data = scaler.fit\_transform(data)

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

# PCA降维

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)])

kmeans = KMeans(n\_clusters=6,init='k-means++',n\_init=100,max\_iter=1000)

kmeans.fit(data)

# 评分指标

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

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

print('calinski\_harabasz\_score:', score1)

print('silhouette\_score:', score2)

# 保存模型

joblib.dump(kmeans, './results/model.pkl')

joblib.dump(scaler, './results/scaler.pkl')

joblib.dump(pca, './results/pca.pkl')

print('over.')

**main.py**的代码是：

import os

import sklearn

import numpy as np

import pandas as pd

from copy import deepcopy

from sklearn.cluster import KMeans

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)

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

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

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

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

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

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

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

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

data = df[columns]

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

data = scaler.fit\_transform(data)

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

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

n\_components = 3

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, 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)

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

1. **实验结果**

测试结果：



最终模型的评价指标：（在处理数据的异常值检测问题时，两个指标都同样重要，但是在不断的测试中发现calinski\_harabasz\_score的分数更高会让在测试时候检测出更多异常点的概率变大）

截屏2023-05-11 下午5.20.21

1. **总结**

本次实验的目标是掌握异常值检测的方法，掌握基于PCA和KMeans算法的异常值检测流程。通过本次实验，我对数据预处理和特征工程的方法更加熟悉，也对PCA和KMeans算法的使用有了更深入的理解。

在实现过程中，我遇到了一些困难，如在使用KMeans算法时，需要对参数进行调优，才能够更好的达到异常值检测的效果。另外，模型的超参数和框架搜索也需要进行合理的选择和设置，才能够得到更优秀的性能。最终我成功的完成了实验，而且调试出的模型获得了不错的性能。框架和参数设置都较为合理。

从可改进的方向来看，我认为可以尝试使用其他的异常值检测算法，如孤立森林、LOF等方法，来进一步提升模型的性能和泛化能力。同时，可以针对实验数据进行更加深入的探究和分析，发现数据中隐藏的信息和异常点，提高模型的检测效果。

从模型性能的角度来看，我们可以进一步提高模型的性能，如增加样本量、对数据进行更好的处理等。同时，可以尝试调整算法的超参数和框架搜索策略，寻找最优的参数组合，进一步提高模型的性能和准确率。

综上所述，本次实验中，我通过对数据进行预处理和特征工程，使用PCA和KMeans算法实现了异常值检测，并对模型进行了性能评价。在未来的实践中，我将继续探索和尝试其他的异常值检测方法，不断提高模型的检测能力和性能。