**程序报告**

学号： 2113997 姓名：齐明杰

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。  
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

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**）

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

**方法：**本实验采用 K-Means 聚类算法对数据进行聚类，进而检测异常点。首先，对数据进行预处理和特征工程，然后进行标准化和 PCA 降维。接着，利用 K-Means 对处理后的数据进行聚类。最后，通过计算样本点与聚类中心的距离来判断异常点。

**改进：**在本实验中，对原始数据进行了特征工程，包括计算小时和是否为白天，以及 cpc 和 cpm 的乘积与比值。此外，使用 PCA 进行降维，减少了特征数量，降低了计算复杂度。

**优化方向：**

1、参数调整：可以尝试调整 K-Means 的参数，例如聚类数、初始质心选择方法、最大迭代次数等。此外，可以调整 PCA 的主成分数量。

2、框架调整：可以尝试其他聚类算法，如 DBSCAN、层次聚类等，比较它们的性能。

方法局限性：K-Means 算法对初始质心选择敏感，容易陷入局部最优。另外，该算法假设类簇具有相似的形状和大小，这可能不适用于所有数据集。

**伪代码：**

- 读取数据

- 数据预处理和特征工程

- 特征标准化

- PCA 降维

- 使用 K-Means 聚类

- 计算轮廓系数评估聚类性能

- 保存模型和预处理对象

**除此之外，我还使用了GridSearchCV模块来自动化探寻K-Means的最佳参数。**

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）

**GridSearchCV.py(用来寻找最佳参数)**

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

import os

import pandas as pd

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import silhouette\_score

from sklearn.model\_selection import GridSearchCV

from sklearn.base import BaseEstimator, ClusterMixin

# 自定义 KMeans 类，使其兼容 GridSearchCV

class CustomKMeans(BaseEstimator, ClusterMixin):

def \_\_init\_\_(self, n\_clusters=2, init='k-means++', n\_init=10, max\_iter=300, tol=1e-6):

self.n\_clusters = n\_clusters

self.init = init

self.n\_init = n\_init

self.max\_iter = max\_iter

self.tol = tol

def fit(self, X, y=None):

self.kmeans\_ = KMeans(n\_clusters=self.n\_clusters, init=self.init, n\_init=self.n\_init, max\_iter=self.max\_iter, tol=self.tol)

self.kmeans\_.fit(X)

return self

def predict(self, X):

return self.kmeans\_.predict(X)

# 定义评分函数

def silhouette\_scorer(estimator, X):

labels = estimator.predict(X)

score = silhouette\_score(X, labels)

return score

# 加载数据并预处理

file\_dir = './data'

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', 'cpc X cpm', 'cpc / cpm']

data = df[columns]

scaler = StandardScaler()

data\_scaled = scaler.fit\_transform(data)

# PCA 降维

n\_components = 3

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

data\_pca = pca.fit\_transform(data\_scaled)

# 设置参数网格

param\_grid = {

'n\_clusters': range(2, 9),

'init': ['k-means++', 'random'],

'n\_init': range(10, 60, 10),

'max\_iter': [300, 500]

}

# GridSearchCV

model = CustomKMeans()

grid\_search = GridSearchCV(model, param\_grid, scoring=silhouette\_scorer, cv=5, n\_jobs=-1, verbose=1)

grid\_search.fit(data\_pca)

# 输出最佳参数

print("Best parameters found: ", grid\_search.best\_params\_)

print("Best silhouette score: ", grid\_search.best\_score\_)

**train.py**

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

import os

import pandas as pd

from sklearn.externals import joblib

from sklearn.cluster import KMeans

from sklearn.decomposition import PCA

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

from sklearn.preprocessing import StandardScaler

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', '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聚类

kmeans = KMeans(n\_clusters=2, init='k-means++', n\_init=30, max\_iter=300)

kmeans.fit(data)

# 计算轮廓系数

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

print("Silhouette score:", score)

# 计算 Calinski-Harabasz

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

print("Calinski-Harabasz score:", score)

# 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**

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

import numpy as np

import pandas as pd

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

"""

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']

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

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

data = df[columns]

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

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

data = scaler.fit\_transform(data)

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

"""

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

"""

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

new\_data = deepcopy(data)

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

threshold = 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 > threshold)

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 = 0.027

# 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

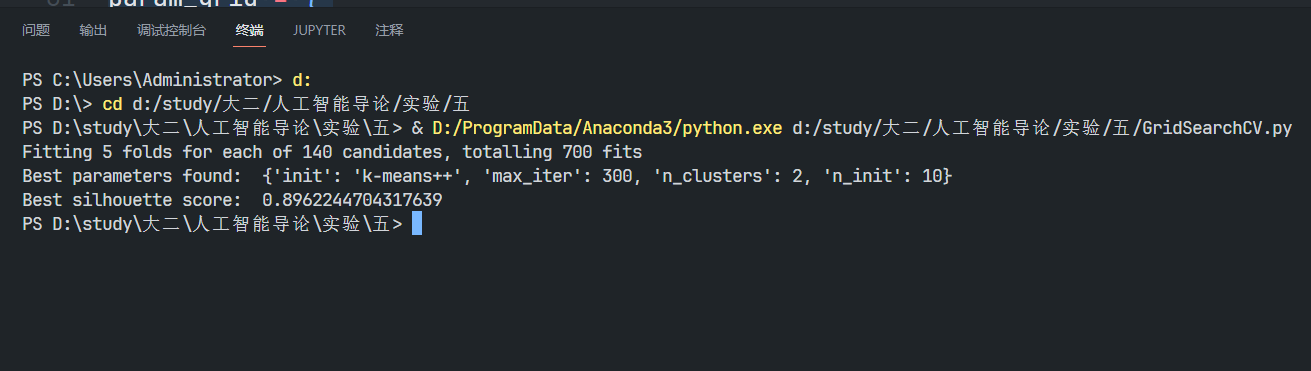
1. **实验结果**

（实验结果，必填）

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

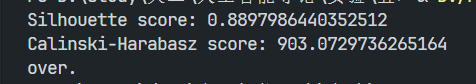
GridSearchCV.py ：

通过使用 GridSearchCV，我们尝试了多个 K-Means 聚类算法的参数组合，并使用交叉验证来评估模型性能，如下图所示：



train.py:

使用上述最佳参数组合训练 K-Means 聚类模型后，我们得到了一个轮廓系数为 0.8897 ，ch得分为903的模型。这表明我们的模型聚类效果相对较好，而且与通过网格搜索得到的最佳轮廓系数相差不大，说明模型性能稳定。如下图所示：



提交到系统中，结果如下：



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**）

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

本次实验采用 K-Means 聚类算法对数据进行了聚类分析。整体而言，实验目标预期达到，但仍有一些改进空间。

**是否达到目标预期：**通过使用 GridSearchCV 进行参数组合的搜索，我们找到了较为优化的模型参数，模型在聚类任务上表现较好，轮廓系数为 0.8898，说明我们的模型预期基本达到。

**可能改进的方向：**

1、尝试其他聚类算法，例如 DBSCAN、Agglomerative Clustering 等，以发现可能的更好的聚类效果。

2、进一步优化特征工程，可能有更多有价值的特征可供挖掘。

3、在模型评估时，可以尝试使用其他指标，如 Calinski-Harabasz 分数或 Davies-Bouldin 分数，以更全面地评估模型性能。

**实现过程中遇到的困难：**

对于聚类算法的参数设置，如何选择合适的参数对模型性能影响较大。在本实验中，我们通过 GridSearchCV 来解决了这个问题。

**从哪些方面可以提升性能：**

提升性能的方向包括：优化特征工程、尝试其他聚类算法、使用更多的评估指标来评估模型。

**模型的超参数和框架搜索是否合理：**

本实验通过 GridSearchCV 对超参数进行了网格搜索，并使用交叉验证评估模型性能，这是一种相对合理的方法。然而，GridSearchCV 的搜索空间可能会受限于我们预先设定的参数范围，因此可能存在一些未被探索到的优秀参数组合。在未来实验中，可以考虑使用随机搜索或贝叶斯优化等方法进一步优化模型参数。