KMeans 实现异常点检测实验报告

学号：3210102328

姓名：牛钟仪

一. 问题重述

1. 题目内容：运用 KMeans 算法完成异常点检测

2. 数据集

i. 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。

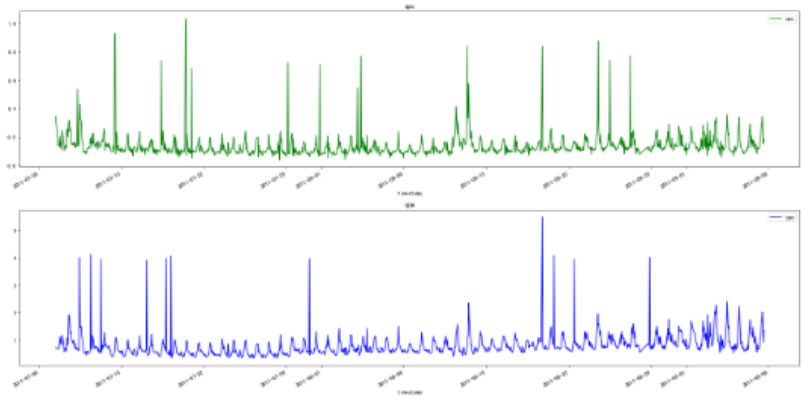
网络广告运营商需要依照这些数据，向广告投放方收取费用。

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。

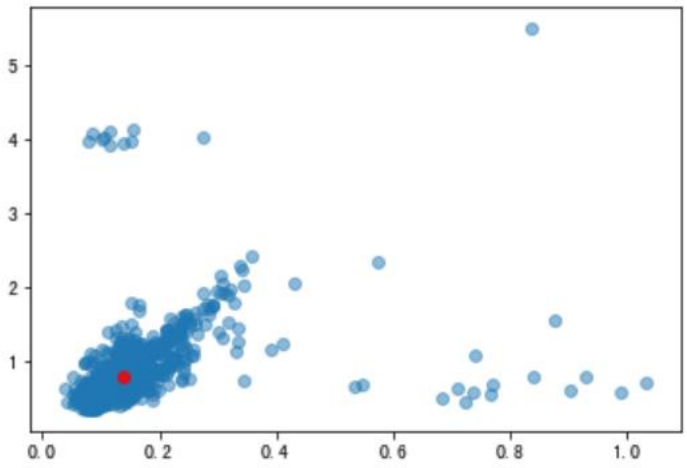
ii. 简单尝试数据预处理

处理数据集中的缺失值、数据归一化等。在该阶段，我们首先可以初步查看数据的分布，并通过函数构造下面的图表



从上图中可以看出，显然有一些时间点的数据值超出均值非常大，这些点出现十分可疑，应该被检测为异常点

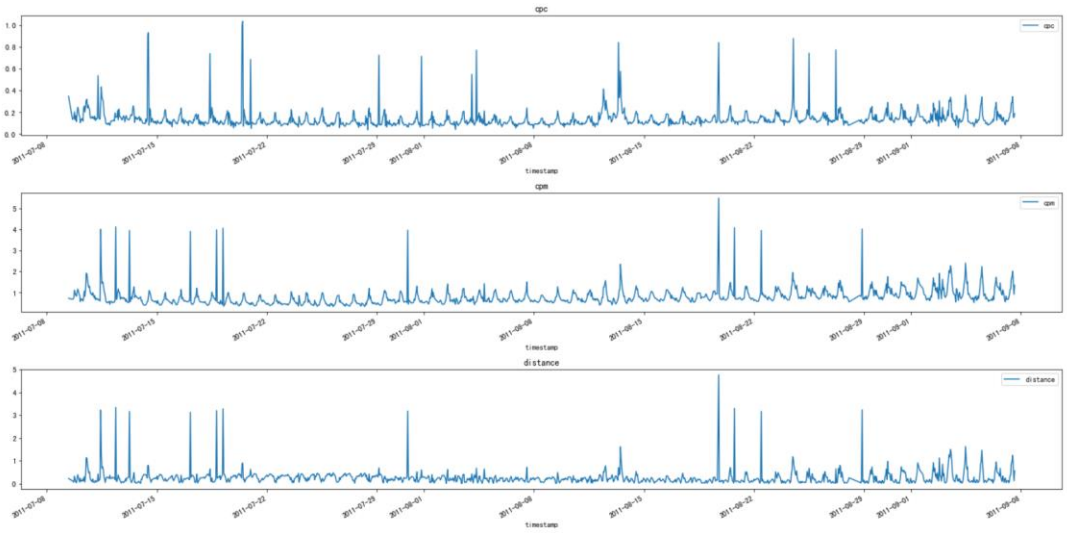
绘制 2 维图



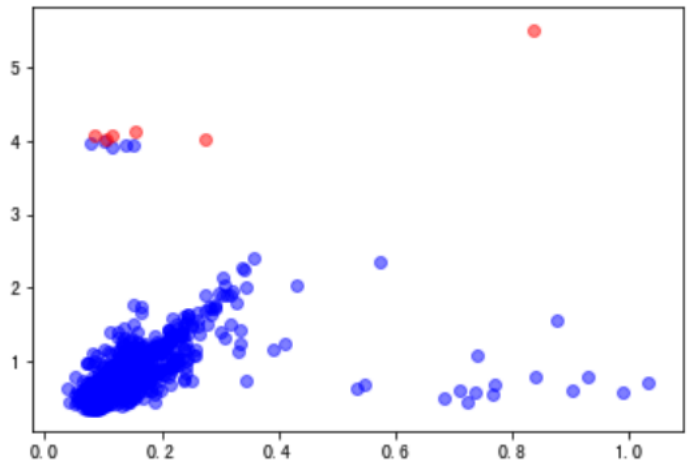
图中红点位置的横坐标是 cpc 的平均值，纵坐标是 cpm 的平均值。

在当前特征下，我们将数据聚集于一个聚类中，尝试使用计算当前点（cpc，cpm）到平均值点（cpc_mean，cpm_mean）的几何距离（L2 范数）寻找聚类中的异常点。

即距离平均值点的距离越大，说明该点是异常点的可能性就越大。



设置一定的异常点比例（ratio = 0.005），根据距离的大小进行划分，将属于该部分的点设置为异常点



我们发现直观的方法并不合适，右下角像异常点的数据并没有被分类为异常点，原因可能是两个特征分布的区间并不相同；

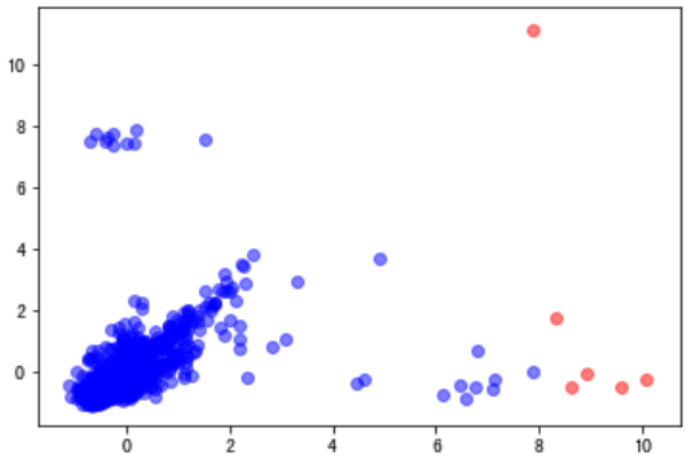
cpm 特征分布在更大的范围内，因此在寻找异常点之前进行数据标准化，使用 sklearn 中的标准化函数，数据标准化后分布在

（-1，1）之间。

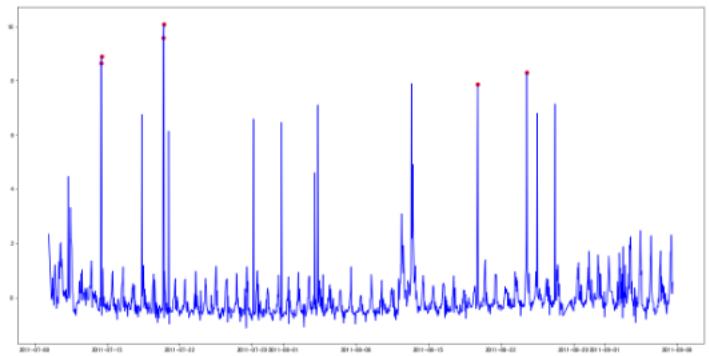
标准化公式：

$$x^* = \frac{x - \mu}{\sigma}$$

其中 μ 为所有样本数据的均值，σ 为所有样本数据的标准差。



从时间维度查看被检测为异常点的数据点，从下图可以发现某些明显高过均值的点已经被标记为异常点



通过上述的流程我们似乎已经比较好的解决了问题，但是由于数据集比较简单，解决方案存在一些**问题**：

- 当前的数据特征为 2 维，我们容易可视化特征，可以观察到数据是一个聚类，可以计算数据的中心点，但当数据的特征数量大于 3，无法有效的可视化，无法得知当前分布情况是否是一个簇？
- 只使用一个簇来衡量数据集是否是合理的？如果确实存在其他潜在的隐藏变量左右了可以观测到的特征的联合分布，存在一个远离大多点的簇是合理的，例如我们数据中右下角的簇，只使用一个簇划分数据有时可能会损失重要的信息。

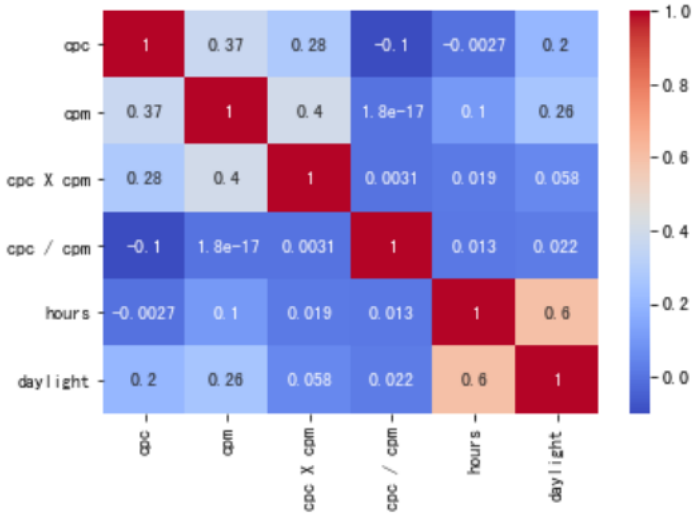
3. 特征构造与 PCA

i. 在上面的简单例子中，我们仅仅使用到了 2 个特征，接下来我们将构造一些额外的特征，进一步用于探究异常点检测。

1. 对于目前存在的 2 个特征可以尝试引入他们的线性、非线性组合作为新的特征，在构建特征的过程中往往需要具体考虑特征所代表的意义，需要相关方面的专业知识。
2. 同时考虑到异常出现的位置，其往往可能与时间点有关，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。

ii. 尝试引入非线性关系和时间关系



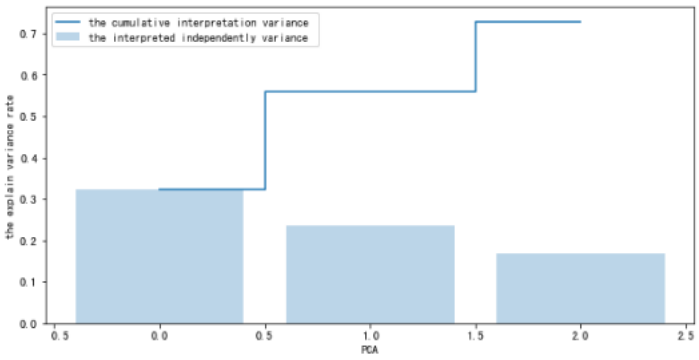
从上图中可以看出，daylight 与 cpc、cpm 仍然具有一定的相关性，说明时间是白天还是晚上对于广告的点击量具有一定的影响。加上 cpc X cpm、cpc / cpm、hours、daylight 等特征之后，数据的特征数量就大于 3，无法观察数据的分布以及分多少个簇合适等；

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

iii. 使用 sklearn 包中的 PCA 方法可以方便的进行特征降维。在 PCA 对象中存储了数据降维过程中的信息，包括特征方向，各个投影方向上的方差等等。

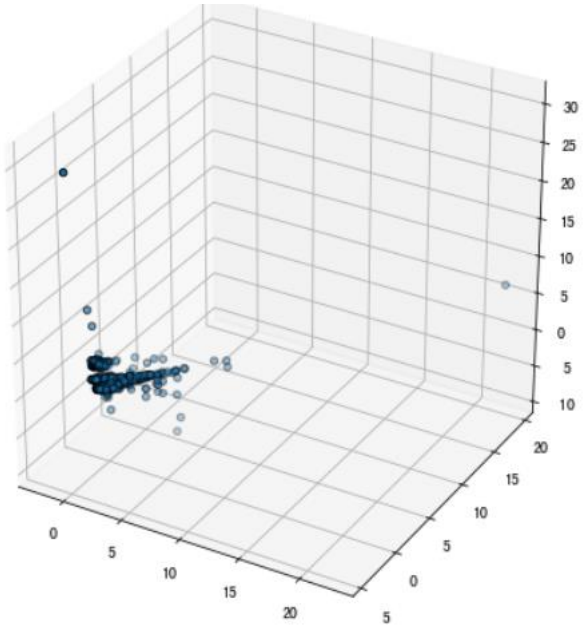


可以看到，原数据集的特征经过一些列的数据处理后，特征维数变多；

而我们认知（比较容易理解）的图形主要有二维平面几何图形和三维立体图形；

特征经过 PCA 处理后的 3 个特征就可以表达原数据 80%以上的数据信息；

因此用这新的 3 个特征就可以较好的表达原数据，那么我们看看这 3 个新特征的数据信息：



4. KMeans 聚类与异常检测

i. 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。

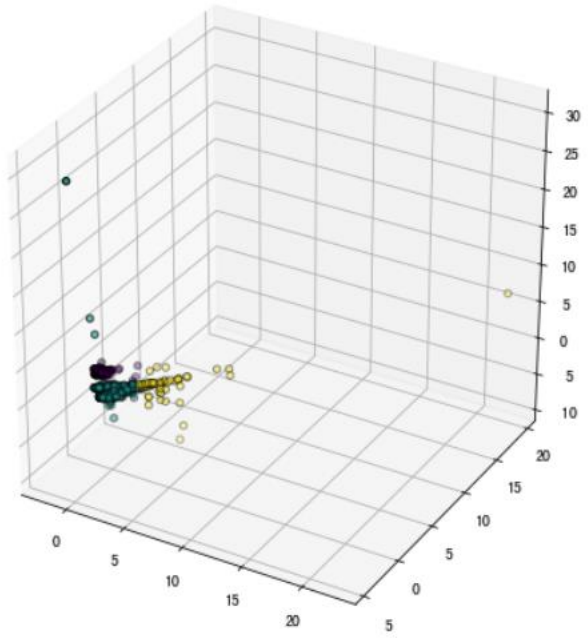
让簇内的点尽量紧密的连在一起，而让簇间的距离尽量的大。

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。



5. 聚类评价指标

i. 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。

二. 设计思想

1. KMeans 模块

i. 设计思路

1. K 值的选择

a) k 的选择一般是按照实际需求进行决定，或在实现算法时直接给定 k 值。在本实验中我们根据各种 k

值的 score 来确定最终的 K 值(本实验中选择了 k = 3)

2. 度量距离

a) 在本代码中我使用了欧式距离

$$d(x,y) := \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2} = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}$$

b) 此时得到 k 个新的簇，每个样本都被分到 k 个簇中的某一个簇。得到 k 个新的簇后，当前的质心就会失效，需要计算每个新簇的自己的新质心

3. 新质心的计算

a) 对于分类后的产生的 k 个簇，分别计算到簇内其他点距离均值最小的点作为质心（对于拥有坐标的簇可以计算每个簇坐标的均值作为质心）

4. 何时停止 KMeans

a) 质心不再改变，或给定 loop 最大次数 loopLimit

三. 代码内容

1. KMeans 模块

```python
# 计算欧拉距离
def calcDis(dataSet, centroids, k):
    clalist=[]
    for data in dataSet:
        diff = np.tile(data, (k, 1)) - centroids  #相减   (np.tile(a,(2,1))就是把a先沿x轴复制1倍，即没有复制，仍然是 [0,1,2]。 再把结果沿y方向复制2倍得到a
        squaredDiff = diff ** 2     #平方
        squaredDist = np.sum(squaredDiff, axis=1)   #和  (axis=1表示行)
        distance = squaredDist ** 0.5  #开根号
        clalist.append(distance)
    clalist = np.array(clalist)  #返回一个每个点到质点的距离len(dateSet)*k的数组
    return clalist

# 计算质心
def classify(dataSet, centroids, k):
    # 计算样本到质心的距离
    clalist = calcDis(dataSet, centroids, k)
    # 分组并计算新的质心
    minDistIndices = np.argmin(clalist, axis=1)    #axis=1 表示求出每行的最小值的下标
    newCentroids = pd.DataFrame(dataSet).groupby(minDistIndices).mean() #DataFramte(dataSet)对DataSet分组，groupby(min)按照min进行统计分类，mean()对分
    newCentroids = newCentroids.values

    # 计算变化量
    changed = newCentroids - centroids

    return changed, newCentroids


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
    """
        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
        """
        dataSet = x.to_numpy()
        k = self.n_clusters
        ##########################################################################
        #### 请勿修改该函数的输入输出 ####
        ##########################################################################
        # #
        # 随机取质心
        centroids = random.sample(list(dataSet), k)

        # 更新质心 直到变化量全为0
        changed, newCentroids = classify(dataSet, centroids, k)
        while np.any(changed != 0):
            changed, newCentroids = classify(dataSet, newCentroids, k)

        centroids = sorted(newCentroids.tolist())   #tolist()将矩阵转换成列表 sorted()排序

        # 根据质心计算每个集群
        cluster = []
        labels = [None for i in range(len(dataSet))]
        clalist = calcDis(dataSet, centroids, k) #调用欧拉距离
        minDistIndices = np.argmin(clalist, axis=1)
        for i in range(k):
            cluster.append([])
        for i, j in enumerate(minDistIndices):   #enymerate()可同时遍历索引和遍历元素
            labels[i] = j
             cluster[j].append(dataSet[i])
        # #
        ##########################################################################
        ############# 在生成 main 文件时，请勾选该模块 #############
        ##########################################################################

        best_centers = centroids
        best_labels = labels
        self.cluster_centers_ = best_centers
        self.labels_ = best_labels
        return self
```

2. Preprocess_data 函数

```python
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
    file_dir = './data'
    csv_files = os.listdir(file_dir)
    # 尝试引入非线性关系
    df['cpc X cpm'] = df['cpm'] * df['cpc']
    df['cpc / cpm'] = df['cpc'] / df['cpm']

    df['timestamp'] = pd.to_datetime(df['timestamp'])
    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',timestamp]
    #columns = None
    #data = df[columns]
    #scaler = None
    #pca = None

    #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', 'cpc / cpm', 'hours','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)])

    # 例如
    #scaler = joblib.load('./results/scaler.pkl')
    #pca = joblib.load('./results/pca.pkl')
    #data = scaler.transform(data)

    return data
```

3. Get_distance 函数

```python
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])
        kmeans.cluster_centers_ = np.array(kmeans.cluster_centers_)
        center = kmeans.cluster_centers_[kmeans.labels_[i],:n_features]
        distance.append(np.linalg.norm(point - center))
    distance = pd.Series(distance)

    return distance
```

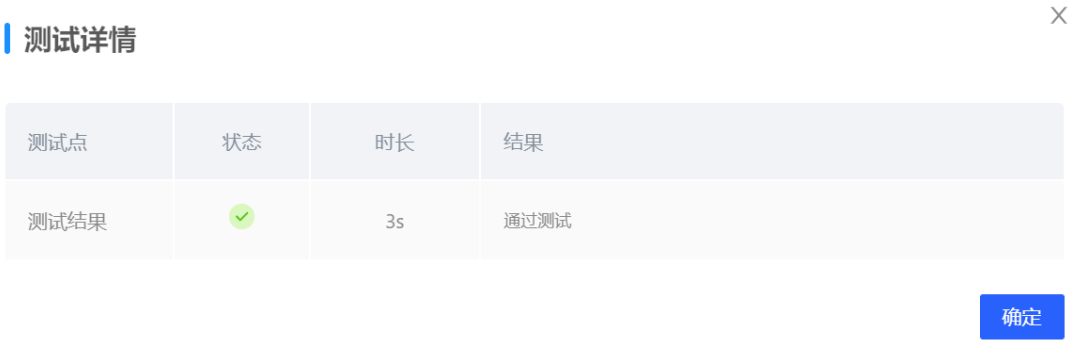
4. Get_anomaly 函数

```python
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)

    #
    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]
    #print('阈值距离: '+str(threshould))
    # 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)
    return new_data
```

5. Predict 函数

```python
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
    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
    :param preprocess_data: preprocess_data函数的返回值，一般是 Da
    :return:is_anomaly:get_anomaly函数的返回值，各个属性应该为（Dim
            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
    """
    # 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
```

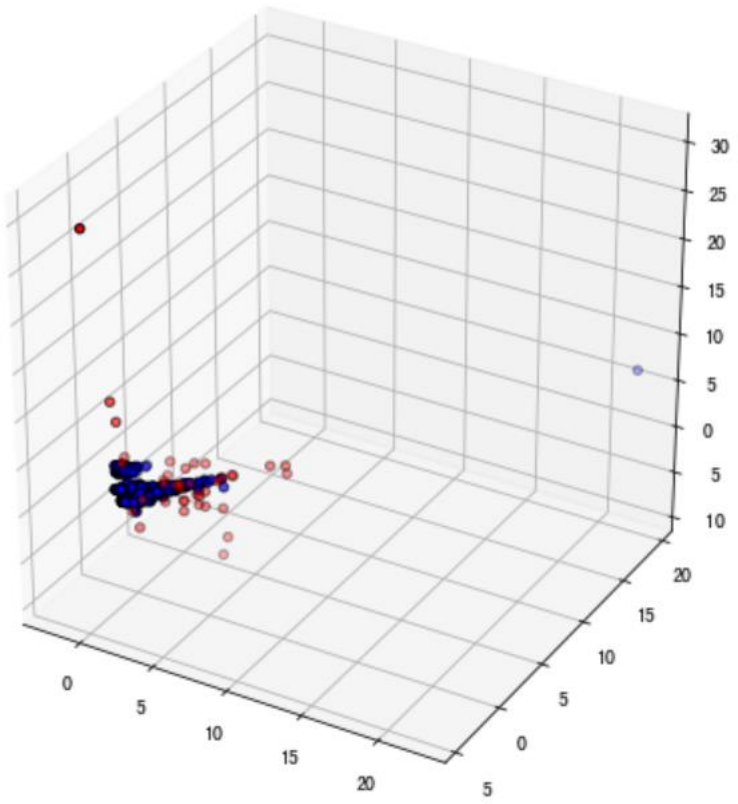
四． 实验结果



测试通过

```
聚类数目:2  calinski_harabasz_score:731.43     silhouette_score:0.6
聚类数目:3  calinski_harabasz_score:726.73     silhouette_score:0.65
聚类数目:4  calinski_harabasz_score:492.87     silhouette_score:0.55
聚类数目:5  calinski_harabasz_score:497.46     silhouette_score:0.51
聚类数目:6  calinski_harabasz_score:404.94     silhouette_score:0.43
聚类数目:7  calinski_harabasz_score:1278.45    silhouette_score:0.55
聚类数目:8  calinski_harabasz_score:522.48     silhouette_score:0.46
聚类数目:9  calinski_harabasz_score:1006.99    silhouette_score:0.42
```

计算各个样本点到中心的距离用于判断样本点是否为异常点



五． 总结

在本次实验中我实现了 KMeans 算法检测异常点，在本次实验中我先选择了四个变量，没有加入时间变量，结果无论选择哪一个 k 值都无法通过测试，加上了时间维度的变量后，顺利通过测试，可见多维度对于结果分析准确的必要性以及提高结果说服力的不可或缺性。

实验的改进方向：显然本代码并不是最佳的代码，在 kmeans 聚类中我并没有使用到 init 这个参数，可以加入这个参数迭代增加聚类的准确性。此外还可以增加其他的变量等进一步提高实验结果的准确性