程序报告

学号：2112514　　姓名：辛浩然

1　问题重述

本次实验的主要目标是探索使用 KMeans 算法来检测异常值的方法。在现实生活中，数据集中的异常值往往会影响到数据的准确性和可靠性，因此检测并处理这些异常值对于数据分析和建模非常重要。

本次实验使用的数据集描述了某网络广告在一定时间范围内的综合曝光率，包括 cpm 和 cpc 两种指标。然而，该数据集中并没有标记异常值，因此需要使用无监督学习方法来检测潜在的异常点。可以使用 KMeans 算法，将数据点聚类，进而检测可能存在的异常点。

2　设计思想

2.1　数据处理

2.1.1　特征构造

特征构造是指通过一些转换方式，从原始数据中提取新的特征。这些新的特征可以更好地描述数据的特点，提高模型的性能。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，还可以引入其他相关特征。

2.1.2　PCA

数据通常由多个特征组成，而这些特征之间往往存在一定程度上的相关性。可以使用主成分分析（PCA）来找到数据集的低维度表达。PCA 是一种无监督学习的方法，其主要思想是通过线性变换将原始特征空间转换为新的特征空间，并找到新特征空间中方差最大的方向，这些方向被称为主成分方向或特征方向。这些特征方向是按照其解释数据方差的大小排序的，因此，我们可以选择前几个特征方向来描述数据集的大部分变化，从而实现数据降维。

2.2　KMeans

数据的分布并不只有一个簇，在寻找异常点时候，应该首先将所有数据点分为多个簇，再计算各个簇中显著远离簇中心的点，作为数据中的异常点。

KMeans 算法按照样本之间的距离大小，将样本集划分为 K 个簇。KMeans 算法的目标是最小化簇内点的距离，同时最大化簇间的距离，以此更好地区分不同的簇，并更容易找到异常点。

对于每个簇，可以计算每个样本点与中心点之间的距离，并据此判断哪些点距离中心点较远，可能是异常点。通常情况下，可以将距离中心点超过某个阈值的点标记为异常值。阈值的设定可以基于数据分布的特征或具体应用的要求进行。

3 代码内容

3.1 模型训练

3.1.1 数据处理

选择 4 个特征，'cpc', 'cpm', 'hours', 'daylight'，通过 PCA 算法将数据从 4 维降低到 3 维。

```
from sklearn.decomposition import PCA

#在进行特征变换之前先对各个特征进行标准化
columns=['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)])
data.head()
```

3.1.2 KMeans 聚类

调整 KMeans 的参数，聚类个数为 3；采用 kmeans++ 方法初始化聚类；计算次数为 50；最大迭代次数为 800.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3, init='k-means++', n_init=50, max_iter
   =800)
kmeans.fit(data)
```

3.2 函数改进

3.2.1 数据预处理

```
def preprocess_data(df):

   # 数据预处理、构造特征
   df['timestamp'] = pd.to_datetime(df['timestamp'])
   df['hours'] = df['timestamp'].dt.hour
   df['daylight'] = ((df['hours'] >= 7) & (df['hours'] <= 22)).astype(
   int)
```

```
    # 模型加载
    scaler = joblib.load('./results/scaler.pkl')
    pca = joblib.load('./results/pca.pkl')

    # 在进行特征变换之前先对各个特征进行标准化
    columns = ['cpc', 'cpm', 'hours', 'daylight']
    data = df[columns]
    data = scaler.transform(data)
    data = pd.DataFrame(data, columns=columns)

    # 使用PCA对数据进行降维
    n_components = 3
    data = pca.transform(data)
    data = pd.DataFrame(data,columns=['Dimension' + str(i+1) for i in
    range(n_components)])

    return data
```

3.2.2 计算样本点与聚类中心的距离

```
def get_distance(data, kmeans, n_features):
    # 初始化一个空列表distance，用于保存每个样本点到它所属聚类中心的距离
    distance = []

    # 遍历数据集中的每个样本点
    for i in range(0,len(data)):
        # 从数据集中取出第i个样本点的前n_features个特征，转换为NumPy数组
        point = np.array(data.iloc[i,:n_features])
        # 获取第i个样本点所属的聚类中心的前n_features个特征，并转换为
    NumPy数组
        center = kmeans.cluster_centers_[kmeans.labels_[i],:n_features]
        # 计算第i个样本点到它所属聚类中心的欧几里得距离，并将其加入
    distance列表中
        distance.append(np.linalg.norm(point - center))

    # 将distance列表转换为Pandas Series对象，并返回该对象
    distance = pd.Series(distance)
    return distance
```

3.2.3 检验异常点

```
def get_anomaly(data, kmean, ratio):
```

```
    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]
    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)
    normal = new_data[new_data['is_anomaly'] == 0]
    anormal = new_data[new_data['is_anomaly'] == 1]

    return new_data
```

3.2.4 测试函数

```
def predict(preprocess_data):
    # 该函数将被用于测试, 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

    # 异常值所占比率
    ratio = 0.022
    # 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, 0.022
```

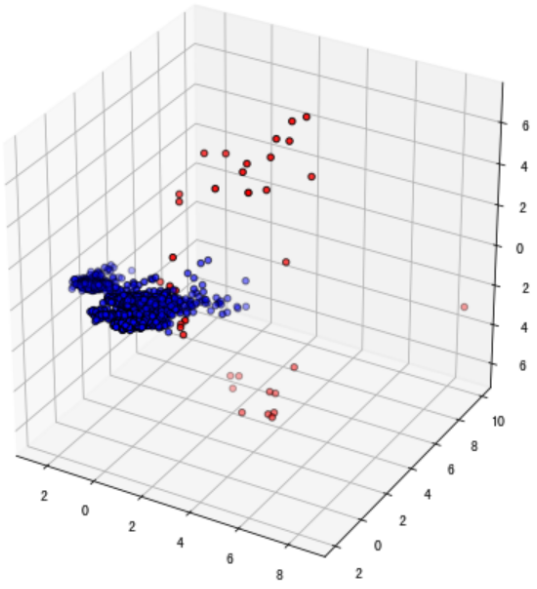
4 实验结果

KMeans 聚类指标得分：

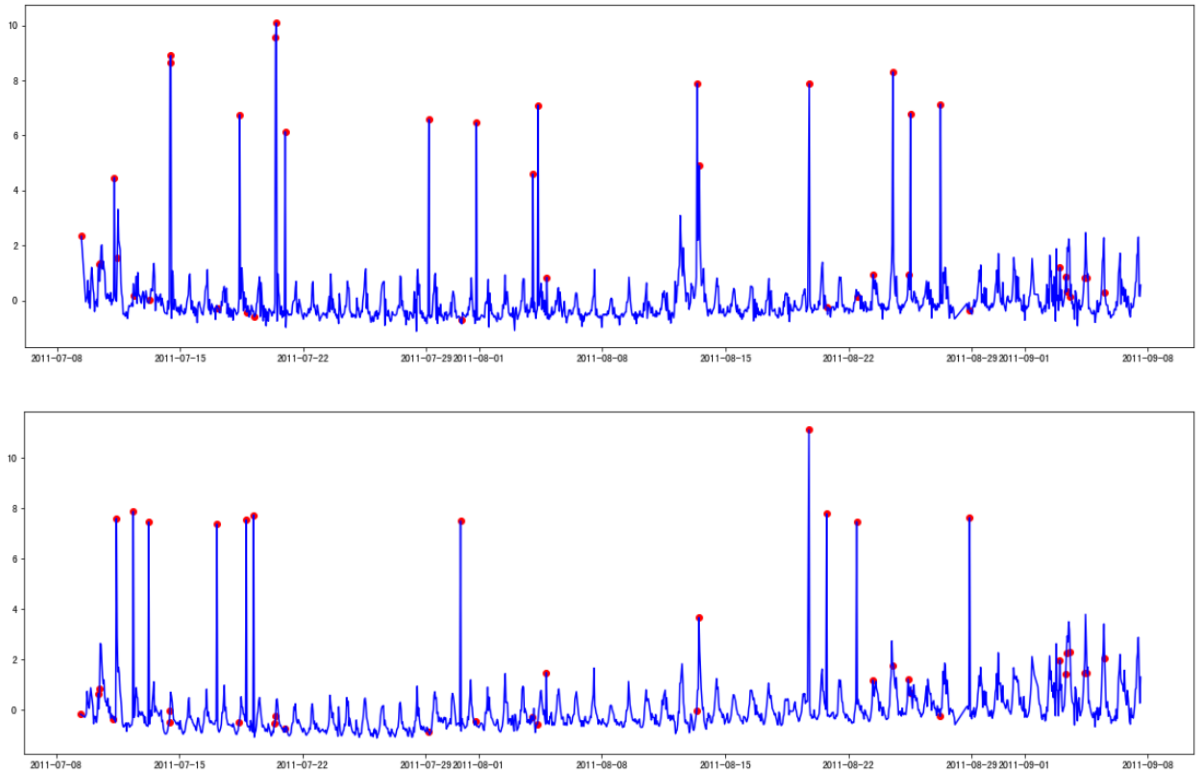
calinski_harabasz_score: 1090.243760751125

silhouette_score: 0.6021444126620 34

在 ratio = 0.03 时可视化观察模型的表现：



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



提交测试，通过测试：

测试详情

测试点	状态	时长	结果
测试结果		0s	通过测试

5 总结

本次实验基本达到了识别异常值的目的。还有一些可以进行优化的方向：

1. 继续调整超参数：KMeans 算法的性能很大程度上取决于超参数的选择，如簇数和初始质心等。

2. 使用更高级的聚类算法：KMeans 算法的一个问题是它对于非球形簇结构表现不佳。因此，使用一些更高级的聚类算法，例如 DBSCAN 或 OPTICS 等，可能会产生更好的结果。

3. 基于模型的方法：另一个改进方向是使用基于模型的方法来检测异常点。例如，可以使用一些混合模型或高斯混合模型来建模数据的分布，然后通过比较数据点和模型预测的概率来判断它们是否为异常点。

4. 使用异常点检测专用算法：最后，考虑使用专门设计的异常点检测算法，例如孤立森林或局部异常因子等，这些算法在异常点检测方面表现得很好。这些算法通常比基于聚类的方法更有效和更准确。