



**数据挖掘**

**上机实验报告**

|  |  |
| --- | --- |
| 专 业： | 人工智能 |
| 班 级： | 192204 |
| 学 号： | 19220432 |
| 姓 名： | 陆昊宇 |

2024年12月02日

## 一、实验目的

1. 掌握使用孤立森林算法（Isolation forest）和局部离群点因子算法（Local outlier factor）进行异常检测。

## 二、实验内容

#### 1.信用卡欺诈检测实验

（1）了解CreditCardFraud 数据集中属性的含义；

（参考https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud）

（2）对数据集中的属性进行初步探索（正常/异常样本的比例、是否存在缺失值、属性类型、数据可视化等），选择合适的数据预处理操作；

（3）以90%和10%的比例划分训练集和测试集（注意：应先分离正常样本和异常样本，分别采样再合并，防止样本不平衡问题对划分的影响）；

（4）分别使用Isolation forest和Local outlier factor进行异常检测实验，尝试将检测结果可视化；

（from sklearn.ensemble import IsolationForest

from sklearn.neighbors import LocalOutlierFactor）

（5）调整两个算法的关键参数，观察指标的变化并对算法进行分析。

**提交入口：**

<https://send2me.cn/Vgg_LaCb/TCq-076N3kSjpA>

## 三、实验报告评分标准

1. 完成实验（1）（5分）

2. 完成实验（2）（20分）

3. 完成实验（3）（5分）

4. 完成实验（4）（25分）

5. 完成实验（5）（15分）

6. 实验过程与实验结论的记录完整（10分）

7. 表述逻辑清晰（10分）

8. 排版工整自洽，图表标题准确（10分）

## 四、实验过程

### 实验一 数据集属性

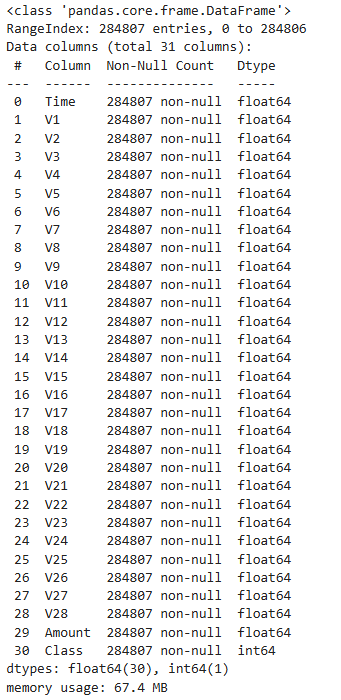
对于数据集中的每一个属性，我们根据其对应的名称、数据类型、数据集信息以及官方介绍[[[1]](#endnote-0)]来明确具体的含义。数据集共31列，以秒为单位记录了2013年欧美某地两天内信用卡的交易情况。由于数据涉及敏感信息，数据集中只保留了交易时间戳、交易金额和欺诈标签3个真实属性，其余28个属性均为使用PCA降维后的脱敏信息。具体如下表所示：

表1 数据集属性含义

|  |  |  |
| --- | --- | --- |
| 属性名称 | 数据类型 | 属性含义 |
| Time | float64 | 交易时间戳（从0开始） |
| V1 | float64 | 脱敏信息1 |
| V2 | float64 | 脱敏信息2 |
| ... | ... | ... |
| V27 | float64 | 脱敏信息27 |
| V28 | float64 | 脱敏信息28 |
| Amount | float64 | 交易金额 |
| Class | int64 | 欺诈标签（0或1） |

### 实验二 数据预处理

使用pandas库将数据集加载后，使用info方法打印数据集的信息，如下图所示：



可以看到数据集没有任何缺失数据，无需进行缺失值处理。

然后分别可视化交易金额与欺诈交易的比例：

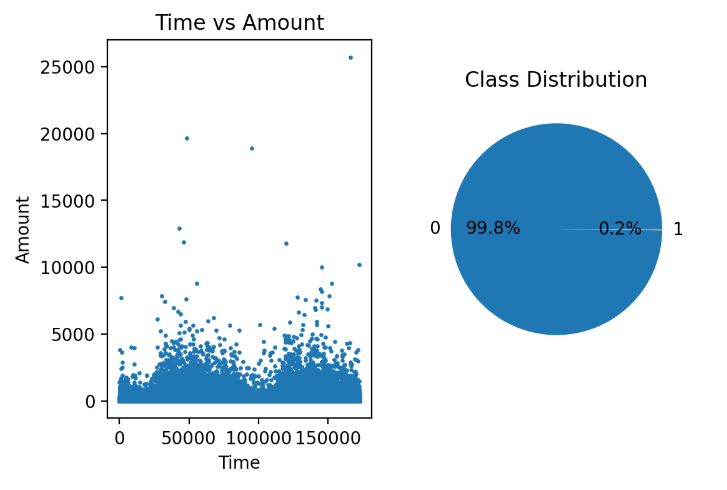


图2 交易金额（左）与是否欺诈（右）

从上左图可以看出，数据集中交易金额普遍集中在0到5000美金不等。从上右图可以看出，这些交易数据中欺诈比例仅占0.2%。

最后将Time和Amount两个属性进行均值方差规范化处理。

### 实验三 数据集划分

由于交易异常的数据占比极低，为了防止采样时出现类别不均衡问题，因此在划分训练集和测试集之前有必要先将交易异常的数据分离出来。之后再进行数据集的划分。

|  |
| --- |
| # 分离正常交易数据和异常交易数据  abnormal\_data = data[data['Class'] == 0]  normal\_data = data[data['Class'] == 1] |
| # 各自划分训练集和测试集  X\_train\_abnormal, X\_test\_abnormal, y\_train\_abnormal, y\_test\_abnormal = train\_test\_split(  abnormal\_data.drop('Class', axis=1), abnormal\_data['Class'], test\_size=0.1, random\_state=random\_state)  X\_train\_normal, X\_test\_normal, y\_train\_normal, y\_test\_normal = train\_test\_split(  normal\_data.drop('Class', axis=1), normal\_data['Class'], test\_size=0.1, random\_state=random\_state) |
| # 合并训练集和测试集  X\_train = pd.concat([X\_train\_abnormal, X\_train\_normal])  X\_test = pd.concat([X\_test\_abnormal, X\_test\_normal])  y\_train = pd.concat([y\_train\_abnormal, y\_train\_normal])  y\_test = pd.concat([y\_test\_abnormal, y\_test\_normal]) |
| # 打乱训练集  indices = np.arange(train.shape[0])  np.random.shuffle(indices)  X\_train = X\_train[indices]  y\_train = y\_train[indices] |

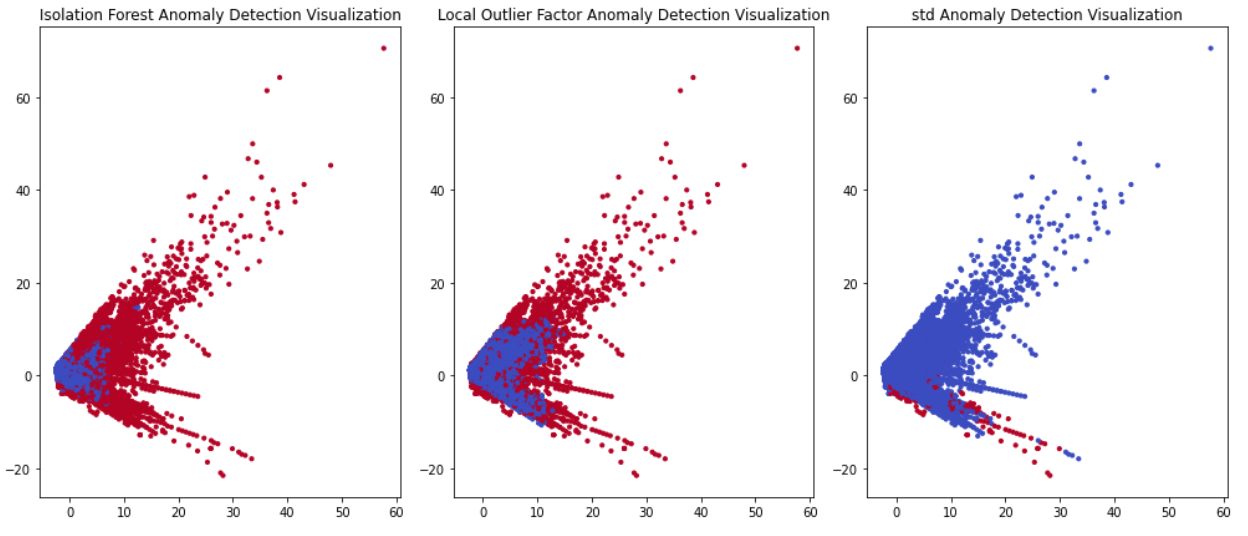
### 实验四 异常检测与可视化

对于Isolation Forest 算法，直接通过库函数IsolationForest 训练并预测，然后将数据降至2维可视化。

对于Local Outlier Factor 算法，由于计算效率随维数增长较快，因此先降至8维训练并预测，然后再降至2维可视化。

代码如下

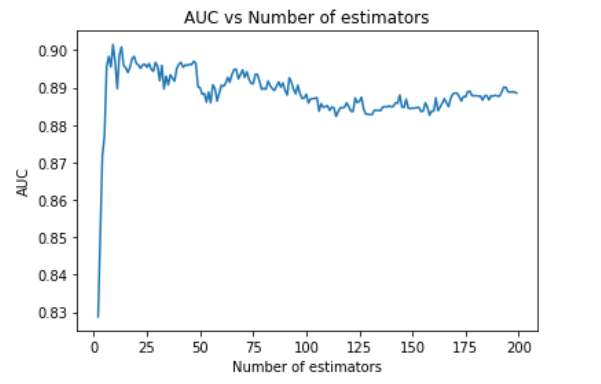
|  |
| --- |
| # --- 1. 使用 Isolation Forest 进行异常检测 ---  iso\_forest = IsolationForest(n\_estimators=10, random\_state=42, n\_jobs=-1)  iso\_outliers = iso\_forest.fit\_predict(X\_train)  iso\_outliers = [0 if o == 1 else 1 for o in iso\_outliers]  print('Isolation Forest AUC：', round(roc\_auc\_score(y\_train, iso\_outliers), 4)) |
| # --- 2. 使用 Local Outlier Factor 进行异常检测 ---  pca = PCA(n\_components=8)  reduced\_data = pca.fit\_transform(X\_train)  lof = LocalOutlierFactor(n\_neighbors=20, n\_jobs=-1)  lof\_outliers = lof.fit\_predict(reduced)  lof\_outliers = [0 if o == 1 else 1 for o in lof\_outliers]  print('Local Outlier Factor AUC：', round(roc\_auc\_score(y\_train, lof\_outliers), 4)) |
| # --- 可视化 ---  pca = PCA(n\_components=2)  reduced\_data = pca.fit\_transform(X\_train)  # 创建一个包含两个子图的图形  fig, axes = plt.subplots(1, 3, figsize=(14, 6))  axes[0].scatter(reduced\_data[:, 0], reduced\_data[:, 1], c=iso\_outliers, cmap='coolwarm', s=10)  axes[0].set\_title("Isolation Forest Anomaly Detection Visualization")  axes[1].scatter(reduced\_data[:, 0], reduced\_data[:, 1], c=lof\_outliers, cmap='coolwarm', s=10)  axes[1].set\_title("Local Outlier Factor Anomaly Detection Visualization")  axes[2].scatter(reduced\_data[:, 0], reduced\_data[:, 1], c=y\_train, cmap='coolwarm', s=10)  axes[2].set\_title("std Anomaly Detection Visualization")  # 显示图形  plt.tight\_layout()  plt.show() |

结果如下  


### 实验五 调整超参数并分析

#### ISO调参

只有一个参数n\_estimators ，由于计算开销小，所以从2 ~ 200 爆搜。

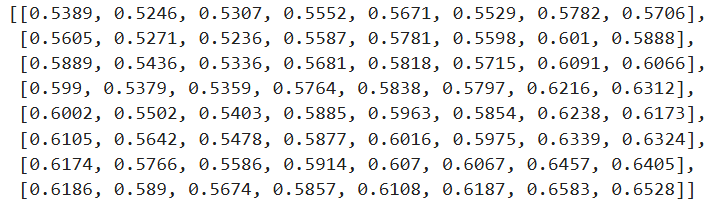


AUC数组表明，当 n\_estimators=9 时的结果最优，为90 %.

**LOF 调参**

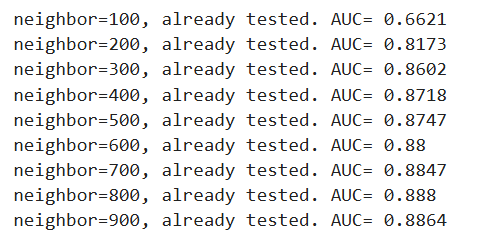
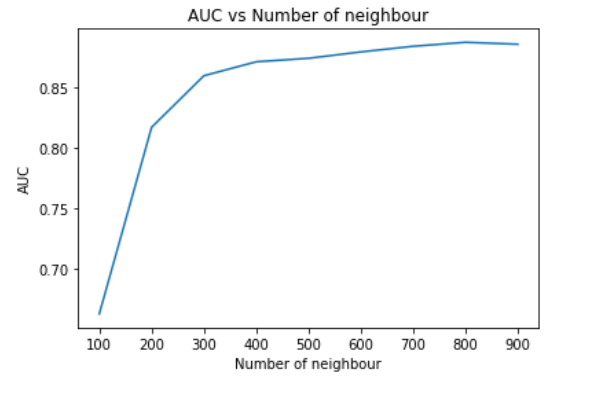
有两个参数：n\_neighbors和n\_components

先在 n\_neighbors=20:100:10, n\_components=2:10:1 中爆搜



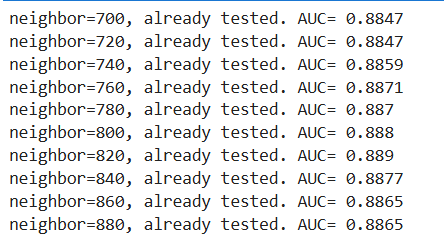
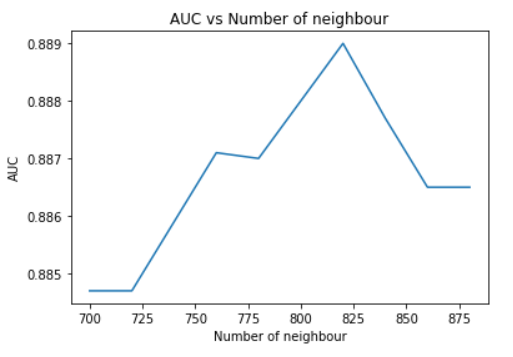
结果表明，n\_components=8是一个很好的参数，并且n\_neighbors尚未出现峰值。

在确定n\_components=8的情况下，搜索n\_neighbors=100:1000:100，结果如下



峰值在700至900间。

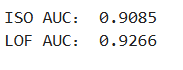
继续减小步长为20，结果如下



可以看到，模型在820时达到最好，AUC=88.9%.

#### 测试集测试

使用上面得到的最好的模型参数进行训练，然后在测试集上测试。



代码如下

|  |
| --- |
| iso\_forest = IsolationForest(n\_estimators=9, random\_state=42, n\_jobs=-1)  iso\_outliers = iso\_forest.fit\_predict(X\_test)  iso\_outliers = [0 if o == 1 else 1 for o in iso\_outliers] |
| pca = PCA(n\_components=8)  reduced\_data = pca.fit\_transform(X\_test)  lof = LocalOutlierFactor(n\_neighbors=820, n\_jobs=-1)  lof\_outliers = lof.fit\_predict(reduced\_data)  lof\_outliers = [0 if o == 1 else 1 for o in lof\_outliers] |
| print('ISO AUC：', round(roc\_auc\_score(y\_test, iso\_outliers), 4))  print('LOF AUC：', round(roc\_auc\_score(y\_test, lof\_outliers), 4)) |

## 实验结论

LOF有更好的拟合效果，但是需要更多的资源消耗和执行时间。

ISO可以快速的给出令人满意的结果，但是无法发现集体的离群点。

此外，本题所使用的数据样本非常不平衡，如果直接采样，很可能会导致训练集与测试集的离群点占比相差悬殊，调参后的结果在测试集上往往不太理想。

1. [↑](#endnote-ref-0)