

【机器学习项目实战01】异常检测——信用卡交易数据检测（上篇）

深圳大学 机电与控制工程学院硕士在读

版权声明：小博主水平有限，希望大家多多指导。本文仅代表作者本人观点。

目录：

1. 【项目背景】
2. 【数据简介】
3. 【导入必备的工具包】
4. 【数据读取】
5. 【数据标签分布】—— ‘0’ 类+ ‘1’ 类
6. 【数据标准化处理】——sklearn处理Amount和Time数据，得到数据集样本分布情况
7. 【下采样方案】根据数据集样本分布情况，提出下采样方案。解决原始数据集样本不均衡，得到一个下采样数据集
8. 【数据集切分】——将 ‘原始数据集’ 和 ‘下采样数据集’ 切分成训练集+验证集+测试集
9. 【交叉验证】调用逻辑回归模型，采用交叉验证方法来评估（本方案使用KFlod）——在下采样的训练集+验证集中找到最好的模型参数
10. 上述 ‘最好的模型’ ，观察在 ‘下采样的测试集’ 中的表现——混淆矩阵
11. 上述 ‘最好的模型’ ，观察在 ‘原始数据的测试集’ 中的表现——混淆矩阵

12.【原始数据方案】——基于下采样方案的结果，如果直接使用原始数据方案会怎么样？

13.【阈值对结果的影响】

14.【过采样方案】——基于SMOTE算法对异常样本集（正例）进行样本生成，解决原始数据集样本不均衡，得到一个下采样数据集

15.【项目总结】

1.【项目背景】

从银行提供的“信用卡交易数据集”中，建立分类模型，找出信用卡欺诈样本。

2.【数据简介】

原始数据

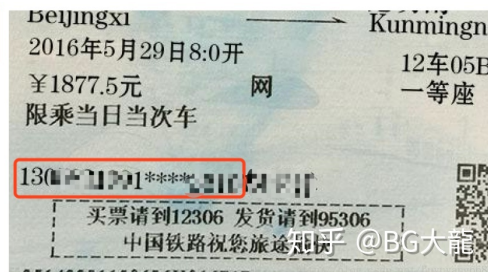
数据集是个人交易记录，但是考虑数据本身的隐私性，已经对原始数据进行了类似PCA（主成分分析法）的处理。换句话说，这些数据是脱敏后的。

什么是“数据脱敏”？

数据脱敏(Data Masking),又称数据漂白、数据去隐私化或数据变形。指对某些敏感信息（如身份证号、手机号、卡号、客户号等个人信息）通过脱敏规则进行数据的变形，实现敏感隐私数据的可靠保护。

应用

生活中不乏数据脱敏的例子，比如我们最常见的火车票、电商收货人地址都会对敏感信息做处理。

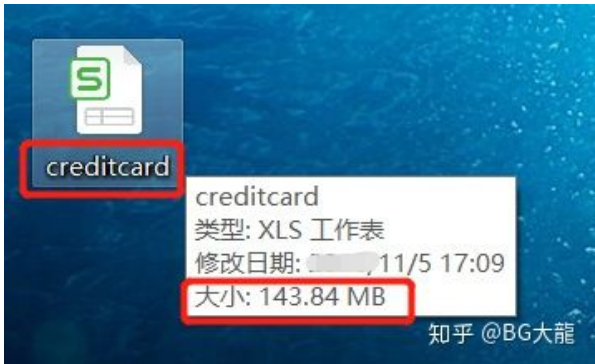


特征数据提取

现在已经把特征数据提取好了，接下来的目的就是“如何建立模型”使得检测的效果达到最好

数据规模

数据共284807条，后期算法选择需要注意复杂度。



数据特征

V1~V28是PCA的结果，而且进行了规范化。其中，Amount和Time数据需要进一步处理。

数据质量

无缺失值。

数据处理经验

时间字段最好可以处理为月份、小时和日期，直接的秒数字段往往无意义。

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474	0.066928
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288	-0.339846
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412	-0.689281
3	1.0	-0.960272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.758278	-0.137408	0.111267

	V25	V26	V27	V28	Amount	Class
0	0.128539	-0.189115	0.133558	-0.021053	149.62	0
1	0.167170	0.125895	-0.008983	0.014724	2.69	0
2	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
3	0.647376	-0.221929	0.062723	0.061458	123.50	0
4	-0.206010	0.502292	0.219422	0.215153	65.99	0

3. 【导入必备的工具包】

代码块：

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

```
%matplotlib inline
```

%matplotlib inline 是指，将图表嵌入到Notebook中，这是Jupyter中的符号形式。

备注：常用Python工具包有哪些？

这里列举Python科学计算的4个常用工具包+1个机器学习库

===**Numpy-科学计算库**===

——主要用来做矩阵运算。不知道哪里会用到矩阵？这样想吧，咱们的数据就是行（样本）和列（特征）组成的，数据本身就是一个矩阵。

===**Pandas-数据分析处理库**===

——说用python处理数据很容易，那么容易在哪？其实有了pandas，很复杂的操作我们也可以一行代码去解决。

===**Matplotlib-可视化库**===

——无论是分析还是建模，很有必要把结果和过程，可视化的展示出来。

===**Seaborn是一个用Python制作统计图形的库**===

——它构建在matplotlib之上，并与pandas数据结构紧密集成。它面向数据集的绘图功能对包含整个数据集的数据流和数组进行操作，并在内部执行必要的语义映射和统计聚合以生成信息图。Seaborn的目标是使可视化成为探索和理解数据的核心部分。

===**Scikit-Learn-机器学习库**===

——非常实用的机器学习算法库，这里面包含了基本你觉得你能用上所有机器学习算法。但还远不止如此，还有很多预处理和评估的模块。

4.【数据读取】

代码块：

```
data = pd.read_csv("creditcard.csv")
data.head()
```

代码解读：

```
data = pd.read_csv("creditcard.csv")
#pd.read_csv(): 是指用pd包去读取相应的.csv文件；
```

```
data.head()
```

#data.head(): data是DataFrame结构，dataFrame是一种二维表，类似Excel。

#head()是Pandas中的函数。data.head()会显示数据前n行；不指定n时，data.head则会显示所有的行。

```
data = pd.read_csv("creditcard.csv")
#pd.read_csv(): 是指用pd包去读取相应的.csv文件;

data.head()
#data.head(): data是DataFrame结构, dataframe是一种二维表, 类似Excel。
#head()是Pandas中的函数。data.head()会显示数据前n行; 不指定n时, data.head则会显示所有的行。
```

代码结果:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...

5 rows × 31 columns

...	V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
...	-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
...	-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
...	0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
...	-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
...	-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	56.30	0

备注:

在用Pandas读取数据之后, 我们往往想要观察一下数据读取是否准确, 这就要用到Pandas里面的head()函数。

为什么用head()函数只能读取前五五行数据???

分析一下head()函数原型。

可以看到, head()函数的原型中, 默认的参数size大小是 5, 所以会返回 5 个数据。

DataFrame.head(n=5)

Return the first n rows.

Parameters: n : int, default 5

Number of rows to select.

Returns: obj_head : type of caller

The first n rows of the caller object.

5.【数据标签分布】—— ‘0’ 类+ ‘1’ 类

代码块：

```
count_classes = pd.value_counts(data['Class'], sort = True).sort_index()
count_classes.plot(kind = 'bar')
print(count_classes)
print(data.shape)

plt.title("Fraud class histogram")
plt.xlabel("Class")
plt.ylabel("Frequency")
```

代码解读：

```
count_classes = pd.value_counts(data['Class'], sort = True).sort_index()
#value_counts()是指，查看列元素Class中“0”、“1”分别有多少个
#sort = True是指，定一种排序，把数量多的标签排在前面
#.sort_index()是对count()的结果进行索引排序
```

```
count_classes.plot(kind = 'bar')
#将结果用plot画图，为了更好地观察排序结果，选图形类别是条形图
```

```
print(count_classes)
print(data.shape)
#打印数目、打印类型
```

```
plt.title("Fraud class histogram")
plt.xlabel("Class")
plt.ylabel("Frequency")
```

#导入工具包的时候，我们用了import matplotlib.pyplot as plt，这里是指对图形的标题和坐标轴进行标

```
count_classes = pd.value_counts(data['Class'], sort = True).sort_index()
#value_counts()是指，查看列元素Class中“0”、“1”分别有多少个
#sort = True是指，定一种排序，把数量多的标签排在前面
#.sort_index()是对count()的结果进行索引排序

count_classes.plot(kind = 'bar')
#将结果用plot画图，为了更好地观察排序结果，选图形类别是条形图

print(count_classes)
print(data.shape)
#打印数目、打印类型

plt.title("Fraud class histogram")
plt.xlabel("Class")
plt.ylabel("Frequency")
#导入工具包的时候，我们用了import matplotlib.pyplot as plt，这里是指对图形的标题和坐标轴进行标
```

知乎 @BG大龍

备注：

1、**value_counts()**：一种查看表格某列中有多少个不同值的快捷方法，并计算每个不同值在该列中有多少重复值。

`value_counts()`在Series的用法。 例如:

```
In [1]: import pandas as pd
import numpy as np
from pandas import DataFrame
from pandas import Series
s1=Series(["timo", "mike", "anni", "timo"])
s1.value_counts()
```

```
Out[1]: timo    2
       anni    1
       mike    1
       dtype: int64
```

知乎 @BG大龍

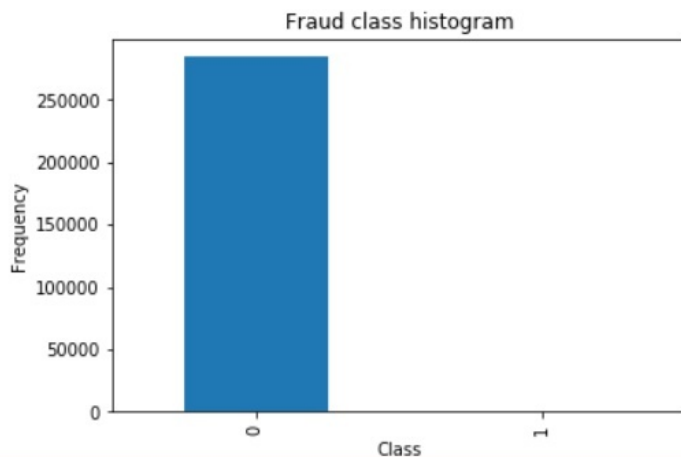
运行结果:

```
count_classes = pd.value_counts(data['Class'], sort = True).sort_index()
count_classes.plot(kind = 'bar')
print(count_classes)
print(data.shape)

plt.title("Fraud class histogram")
plt.xlabel("Class")
plt.ylabel("Frequency")
```

```
0    284315
1      492
Name: Class, dtype: int64
(284807, 30)
```

Text(0, 0.5, 'Frequency')



知乎 @BG大龍

结果分析:

观察可以看出, 在这个数据集中正负样本的比例极不平衡, 正样本数达到284315个, 而负样本只有492个。

图中class为1的并不是没有, 而是太少了, 少到基本看不出来。

这也符合正常情况：利用信用卡欺诈的情况毕竟是少数，大多数人都是守法的公民。但是面对一个问题——样本极度不均衡，怎么解决？

>>>这里提出两种解决方案，也是数据分析中最常用的两种方法，下采样和过采样！

6.【数据标准化处理】——sklearn处理Amount和Time数据，得到数据集样本分布情况

- 1 从数据表可以看到，V1-V28列的数据都在 $(-1,1)$ 的区间内，而Amount这一列浮动范围比较大，所以要将Amount这一列的数据进行归一化（标准化）操作。
- 2 数据中有一列是Time，这一列数据是无用的，也就是说对于信用卡欺诈预测是没有用的，所以我们要将其删掉。

代码块：

```
from sklearn.preprocessing import StandardScaler
data['normAmount'] = StandardScaler().fit_transform(data['Amount'].reshape(-1, 1))
data = data.drop(['Time', 'Amount'], axis=1)
data.head()
```

代码解读：

```
from sklearn.preprocessing import StandardScaler
    #从sklearn库的 “预处理操作preprocessing” 加载 “ 标准化模块StandardScaler”

data['normAmount'] = StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))
    #fit_transform(): 对数据进行一个运算和变换
    # “data['Amount']” 指取Amount这一列
    # “.values” 指转化成numpy array数据
    # “.reshape (-1,1)” 指将这个数组转换成X*1的形式，至于X是多少就要看原始数组中到底有多少元素了
    #最后将标准化后的结果，送给 “data['normAmount]” ， 得到新的数据表

data = data.drop(['Time', 'Amount'], axis=1)
    #data.drop()函数——删除表中的某一行或者某一列更明智的方法是使用drop
    它不改变原有的df中的数据，而是返回另一个dataframe来存放删除后的数据
    #axis=1是指按行对应的方向
    #data.drop(['Time','Amount'], axis=1) 代表将'Time','Amount'对应的列标签（们）沿着水平的方向依次

data.head()
```

```
from sklearn.preprocessing import StandardScaler
    #从sklearn库的 “预处理操作preprocessing” 加载 “ 标准化模块StandardScaler”

data['normAmount'] = StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))
    #fit_transform(): 对数据进行一个运算和变换
    # “data['Amount']” 指取Amount这一列
    # “.values” 指转化成numpy array数据
    # “.reshape (-1,1)” 指将这个数组转换成X*1的形式，至于X是多少就要看原始数组中到底有多少元素了
    #最后将标准化后的结果，送给 “data['normAmount']” ， 得到新的数据表

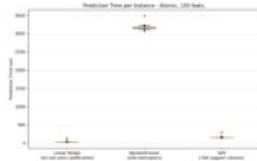
data = data.drop(['Time', 'Amount'], axis=1)
    #data.drop()函数——删除表中的某一行或者某一列更明智的方法是使用drop
    它不改变原有的df中的数据，而是返回另一个dataframe来存放删除后的数据
    #axis=1是指按行对应的方向
    #data.drop(['Time','Amount'], axis=1) 代表将'Time','Amount'对应的列标签（们）沿着水平的方向依次删掉

data.head()
```

知乎 @BG大龍

备注：

1、培养 “遇到问题先看API” 的习惯，API文档地址：



scikit-learn.org

2、fit_transform(): 对数据进行一个运算和变换。

The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

where u is the mean of the training samples or zero if `with_mean=False`, and s is the standard deviation of the training samples or one if `with_std=False`.

3、data['Amount'].values.reshape(-1, 1)



"data['Amount']" 指取Amount这一列;

".values" 指转化成numpy array数据;

".reshape (-1,1)" 指将这个数组变换成X*1的形式, 至于X是多少就要看原始数组中到底有多少元素了。

```
fit_transform(self, X, y=None, **fit_params)
```

Fit to data, then transform it.

Fits transformer to X and y with optional parameters `fit_params` and returns a transformed version of X .

Parameters: X : *numpy array of shape $[n_samples, n_features]$*

Training set.

y : *numpy array of shape $[n_samples]$*

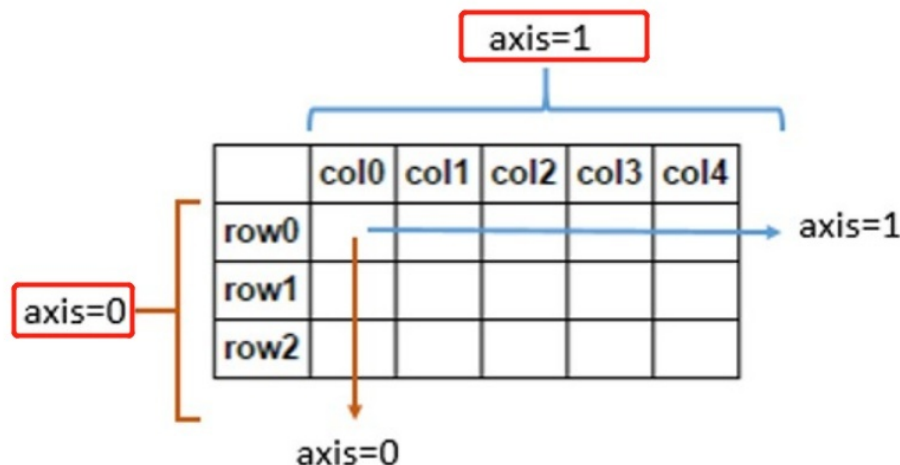
Target values.

Returns: X_new : *numpy array of shape $[n_samples, n_features_new]$*

Transformed array.

知乎 @BG大龍

4、axis=1什么意思? ——在DataFrame当中axis为0和1时分别代表的含义:



axis参数作用方向图示


知乎 @BG大龍

所以，`data.drop(['Time','Amount'], axis=1)` 代表将'Time','Amount'对应的列标签（们）沿着水平的方向依次删掉。

代码结果：

```
from sklearn.preprocessing import StandardScaler

data['normAmount'] = StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))
data = data.drop(['Time', 'Amount'], axis=1)
data.head()
```



	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074

5 rows × 30 columns

知乎 @BG大龍

7. 【下采样方案】根据数据集样本分布情况，提出下采样方案。解决原始数据集样本不均衡，得到一个下采样数据集

未完，见中篇.....

参考：

1、信用卡欺诈案例（终结） - stranger_man的博客 - CSDN博客

https://blog.csdn.net/stranger_man/article/details/79055095blog.csdn.net

2、【迪哥有点愁】唐宇迪的机器学习博客 - CSDN博客

<https://blog.csdn.net/tangyudi?t=1blog.csdn.net>