

## 【机器学习项目实战01】异常检测——信用卡交易数据检测（下篇） - 知乎

笔记本： 机器学习\_调参优化  
创建时间： 2019/11/28 22:00  
URL: <https://zhuanlan.zhihu.com/p/77540404>

## 【机器学习项目实战01】异常检测——信用卡交易数据检测（下篇）



BG大龍

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：

BG大龍：【机器学习项目实战01】异常检测——信用卡交易数据检测（上篇）

[zhuanlan.zhihu.com](https://zhuanlan.zhihu.com)



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

BG大龍：【机器学习项目实战01】异常检测——信用卡交易数据检测（中篇）

[zhuanlan.zhihu.com](https://zhuanlan.zhihu.com)



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平衡，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

BG大龍：【机器学习项目实战01】异常检测——信用卡交易数据检测（下篇）



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】

## 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？

**【1】【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）——找到最好的模型参数**

### 分步骤理解代码

【步骤1】导入工具包

【步骤2】编写Kflod函数，打印“正则化惩罚力度c\_param”

【步骤3】5次迭代“正则化惩罚系数c\_param”后，计算每一次迭代的召回率，并打印出来

【步骤4】找到best\_c，使得召回率Recall最高

```
# 【1】导入工具包
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import KFold, cross_val_score
from sklearn.metrics import confusion_matrix, recall_score, classification_report
from sklearn.model_selection import cross_val_predict

# 【2】编写Kflod函数—printing_Kfold_scores，实际中我们可以直接调用
def printing_Kfold_scores(x_train_data, y_train_data):
    fold = KFold(len(y_train_data), 5, shuffle=False)    # shuffle=False是指数据集不用洗

    # 定义不同力度的正则化惩罚力度
    c_param_range = [0.01, 0.1, 1, 10, 100]

    # 展示结果用的表格
    results_table = pd.DataFrame(index = range(len(c_param_range), 2), columns = ['C_parameter', 'Recall'])
    results_table['C_parameter'] = c_param_range

    # k-fold 表示K折的交叉验证，这里会得到两个索引集合：训练集 = indices[0]，验证集 = indices[1]
    j = 0

    # 循环遍历不同的参数（这里的c_param_range是5个——5折交叉验证）
    for c_param in c_param_range:
        print('-----')
        print('正则化惩罚力度：', c_param)
```

```

print('-----')
print('')

# 【3】 计算每一次迭代后的召回率，一次5次
recall_accs = []

#一步步分解来执行交叉验证
for iteration, indices in enumerate(fold,start=1):

# 建模。选择算法模型+给定参数
    lr = LogisticRegression(C = c_param, penalty = 'l1') #L1正则化防止过拟合，

# 训练模型。注意索引不要给错了，训练的时候一定传入的是训练集，所以X和Y的索引都是0
    lr.fit(x_train_data.iloc[indices[0],:],y_train_data.iloc[indices[0],:].values)

# 测试模型。这里用验证集预测模型结果，这里用的就是验证集，索引为1，验证集 = indices[1]
    y_pred_undersample = lr.predict(x_train_data.iloc[indices[1],:].values)

# 评估模型。有了预测结果之后就可以来进行评估了，这里recall_score需要传入预测值和真实值。
    recall_acc = recall_score(y_train_data.iloc[indices[1],:].values,y_pred_undersample)
# 一会还要算平均，所以把每一步的结果都先保存起来。
    recall_accs.append(recall_acc)
print('Iteration ', iteration,': 召回率 = ', recall_acc)

# 【4】当执行完所有的交叉验证后，计算平均结果
    results_table.loc[j,'Mean recall score'] = np.mean(recall_accs)
    j += 1 #在这儿的意思是 num = num + 1，如果不输入这一行代码
print('')
print('平均召回率 ', np.mean(recall_accs))
print('')

# 找到最好的参数，哪一个Recall高，自然就是最好的了。
    best_c = results_table.loc[results_table['Mean recall score'].astype('float32').idxmax(), 'C']

# 打印最好的结果
print('*****')
print('效果最好的模型所选参数 = ', best_c)
print('*****')

return best_c
best_c = printing_Kfold_scores(X_train,y_train)

```

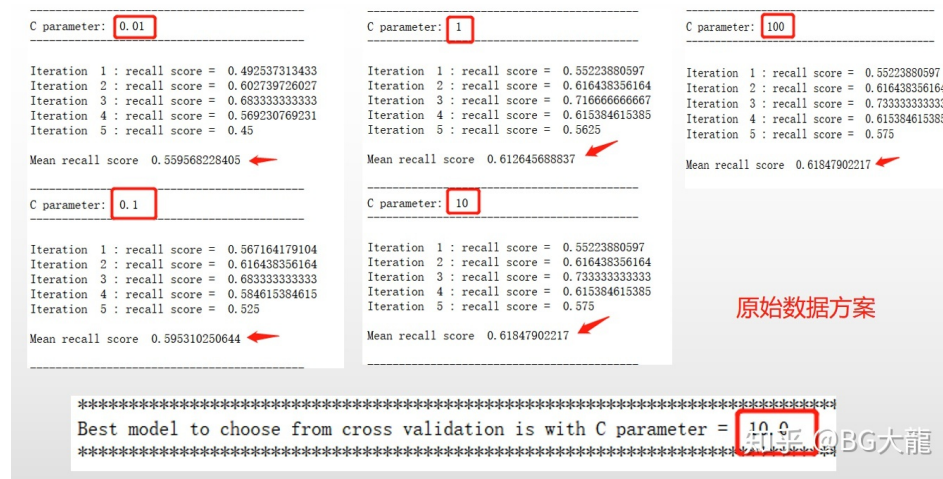
```
best_c = printing_Kfold_scores(X_train_undersample,y_train_undersample)
```

下采样方案

```
best_c = printing_Kfold_scores(X_train,y_train)
```

原始数据方案

结果：



对比下采样方案:



## 【2】上述 ‘最好的模型’，在 ‘原始数据的测试集’ 中的表现——混淆矩阵

```
# 建模。选择算法+给定参数
lr = LogisticRegression(C = best_c, penalty = 'l1')

# 训练模型
lr.fit(X_train,y_train.values.ravel())

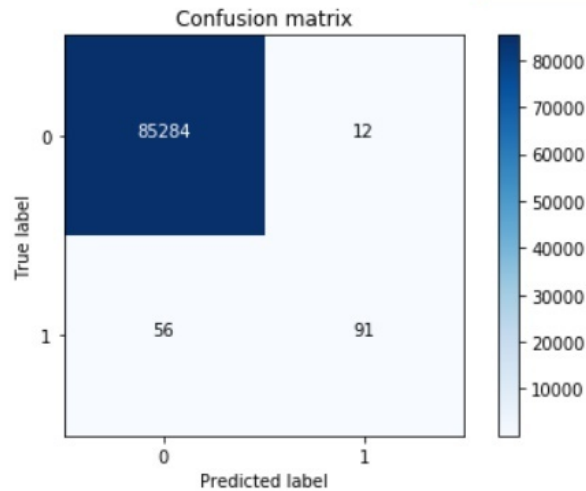
# 测试模型
y_pred_undersample = lr.predict(X_test.values)

# 评估模型。计算混淆矩阵
cnf_matrix = confusion_matrix(y_test,y_pred_undersample)
np.set_printoptions(precision=2)

print("Recall metric in the testing dataset: ", cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_m
```

```
# Plot non-normalized confusion matrix
class_names = [0,1]
plt.figure()
plot_confusion_matrix(cnf_matrix
                      , classes=class_names
                      , title='Confusion matrix')
plt.show()
```

Recall metric in the testing dataset: **0.619047619048**



知乎 @BG大龍

### 13. 【阈值对结果的影响】

#### 分步骤理解代码：

逻辑回归中的激活函数  $\sigma(\theta)$ ，默认阈值为0.5

```
# 【1】
# 建模。选择算法+给定参数
lr = LogisticRegression(C = 0.01, penalty = 'l1')
# 训练模型。还是用下采样的训练集
lr.fit(X_train_undersample, y_train_undersample.values.ravel())
# 预测概率。得到预测结果的概率值
y_pred_undersample_proba = lr.predict_proba(X_test_undersample.values)
# 指定不同的阈值。为了展示，这里用了9个值
thresholds = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
```

```
【2】
# 指定图的大小
plt.figure(figsize=(10,10))

j = 1
# 用混淆矩阵来进行展示
for i in thresholds:
    # 阈值判断
    y_test_predictions_high_recall = y_pred_undersample_proba[:,1] > i
    # 画出3*3的图，j是第几个图
```

```

plt.subplot(3,3,j)
j += 1

cnf_matrix = confusion_matrix(y_test_undersample,y_test_predictions_high_recall)
#精度为小数点后2位
np.set_printoptions(precision=2)
#打印不同阈值时候的召回率
print("给定阈值为:",i,"时测试集召回率: ", cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1]))
#定义种类的标签“0”、“1”
class_names = [0,1]
plot_confusion_matrix(cnf_matrix
                      , classes=class_names
                      , title='Threshold >= %s'%i)

```

```

# 建模。选择算法+给定参数
lr = LogisticRegression(C = 0.01, penalty = 'l1')
# 训练模型。还是用下采样的训练集
lr.fit(X_train_undersample,y_train_undersample.values.ravel())
# 预测概率。得到预测结果的概率值
y_pred_undersample_proba = lr.predict_proba(X_test_undersample.values)
# 指定不同的阈值。为了展示，这里用了9个值
thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
# 指定图的大小
plt.figure(figsize=(10,10))

j = 1
# 用混淆矩阵来进行展示
for i in thresholds:
    ... #阈值判断
    ... y_test_predictions_high_recall = y_pred_undersample_proba[:,1] > i ...
    ... #画出3*3的图，j是第几个图
    ... plt.subplot(3,3,j)
    ... j += 1
    ...
    ... cnf_matrix = confusion_matrix(y_test_undersample,y_test_predictions_high_recall)
    ... #精度为小数点后2位
    ... np.set_printoptions(precision=2) ...
    ... #打印不同阈值时候的召回率
    ... print("给定阈值为:",i,"时测试集召回率: ", cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1]))
    ... #定义种类的标签“0”、“1”
    ... class_names = [0,1] ...
    ... plot_confusion_matrix(cnf_matrix
    ...                       , classes=class_names
    ...                       , title='Threshold >= %s'%i)

```

知乎 @BG大龍

【注解】plt.figure()——绘图；

plt.figure(figsize=(10,10))——指定figure的宽和高，分别为10英寸

语法：

figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None, frameon=True)

参数：

num:图像编号或名称，数字为编号，字符串为名称

figsize:指定figure的宽和高，单位为英寸；

dpi参数指定绘图对象的分辨率，即每英寸多少个像素，缺省值为80 ——1英寸等于2.5cm,A4纸是 21\*30cm的纸张

facecolor:背景颜色

edgecolor:边框颜色

frameon:是否显示边框

【注解】 confusion\_matrix函数的使用——以矩阵形式将数据集中的记录，按照 ‘ture类别’ 与 ‘predict类别’ 两个标准进行汇总

语法:

`sklearn.metrics.confusion_matrix(y_true, y_pred, labels=None, sample_weight=None)`

参数:

`y_true`: 是样本真实分类结果

`y_pred`: 是样本预测分类结果

`labels`: 是所给出的类别, 通过这个可对类别进行选择

`sample_weight`: 样本权重

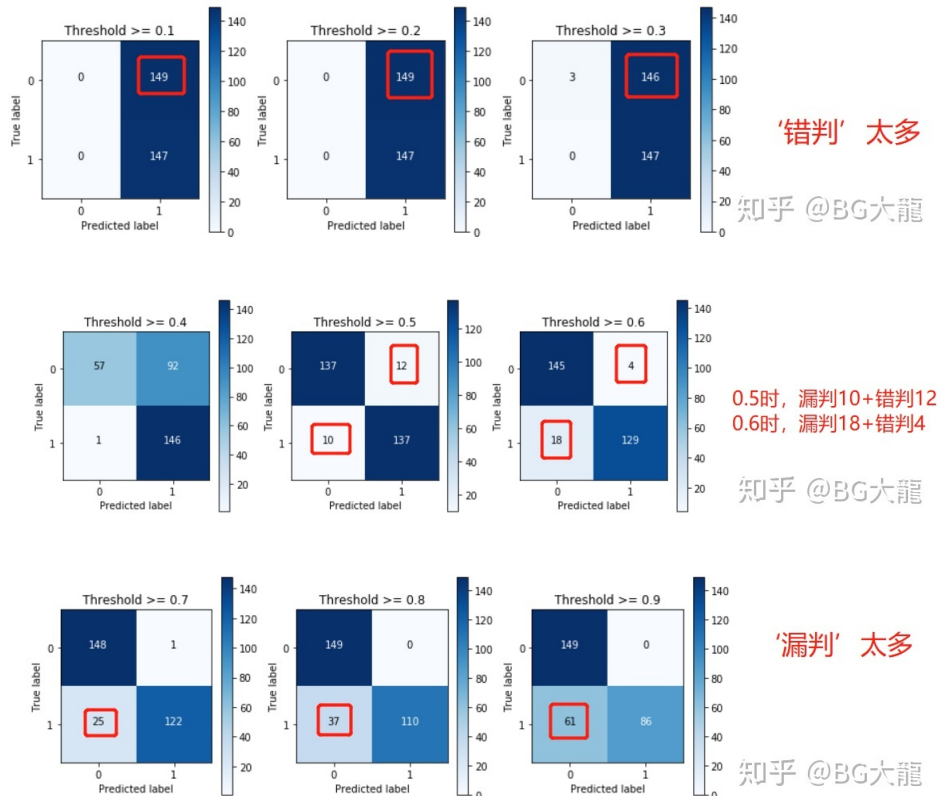


知乎 @BG大龍

## 结果

给定阈值为: 0.1 时测试集召回率: 1.0  
给定阈值为: 0.2 时测试集召回率: 1.0  
给定阈值为: 0.3 时测试集召回率: 1.0  
给定阈值为: 0.4 时测试集召回率: 0.993197278912  
给定阈值为: 0.5 时测试集召回率: 0.931972789116  
给定阈值为: 0.6 时测试集召回率: 0.877551020408  
给定阈值为: 0.7 时测试集召回率: 0.829931972789  
给定阈值为: 0.8 时测试集召回率: 0.748299319728  
给定阈值为: 0.9 时测试集召回率: 0.585034013605

知乎 @BG大龍



知乎 @BG大龍

知乎 @BG大龍

知乎 @BG大龍



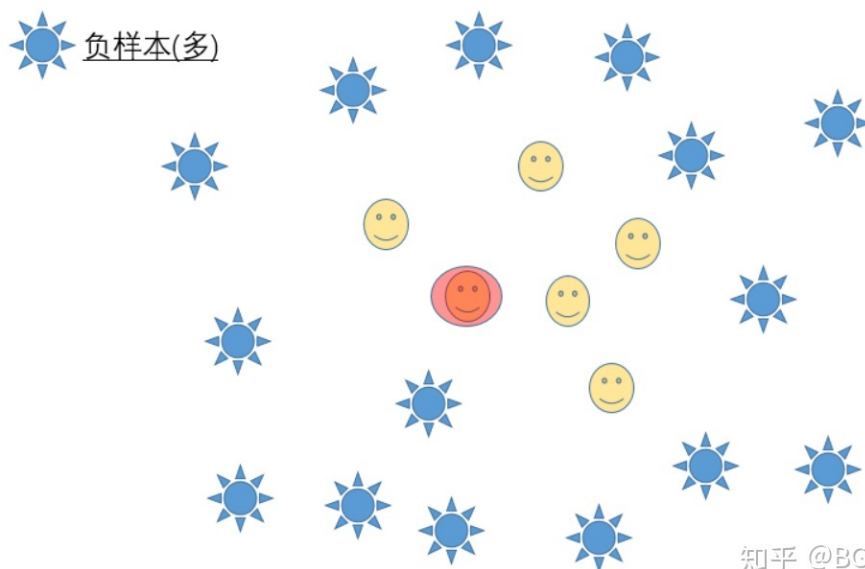
#### 14.【过采样方案】——基于SMOTE算法对异常样本集（正例）进行样本生成，解决原始数据集样本不均衡，得到一个下采样数据集

##### 概念：SMOTE算法原理

步骤1-选一个少数样本（针对少数样本来说）

😊 正样本(少)

☀ 负样本(多)



知乎 @BG大龍

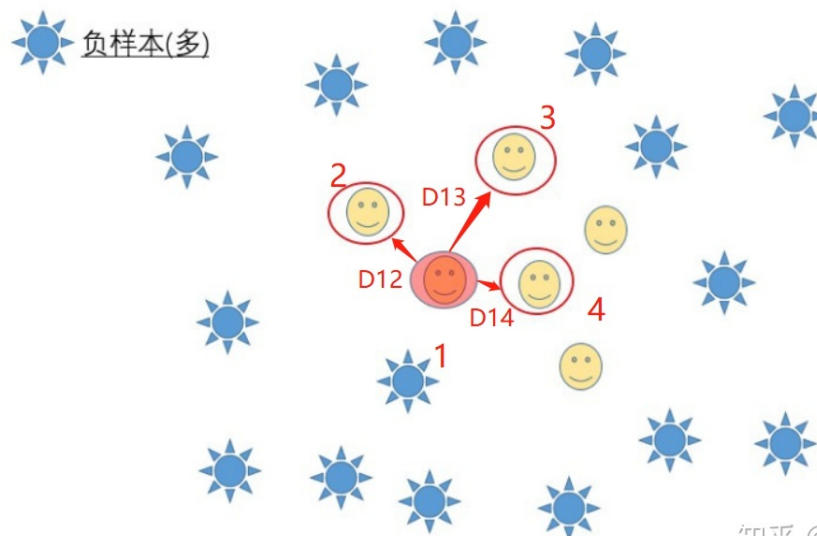
[https://zhuanlan.zhihu.com/p/104111111](#)

步骤2-找到该样本的K个近邻（假设K=3）

计算相距的距离D——》按距离的由近到远排序——》按K值，选最近的K个，这就是该样本的K个近邻

😊 正样本(少)

☀ 负样本(多)



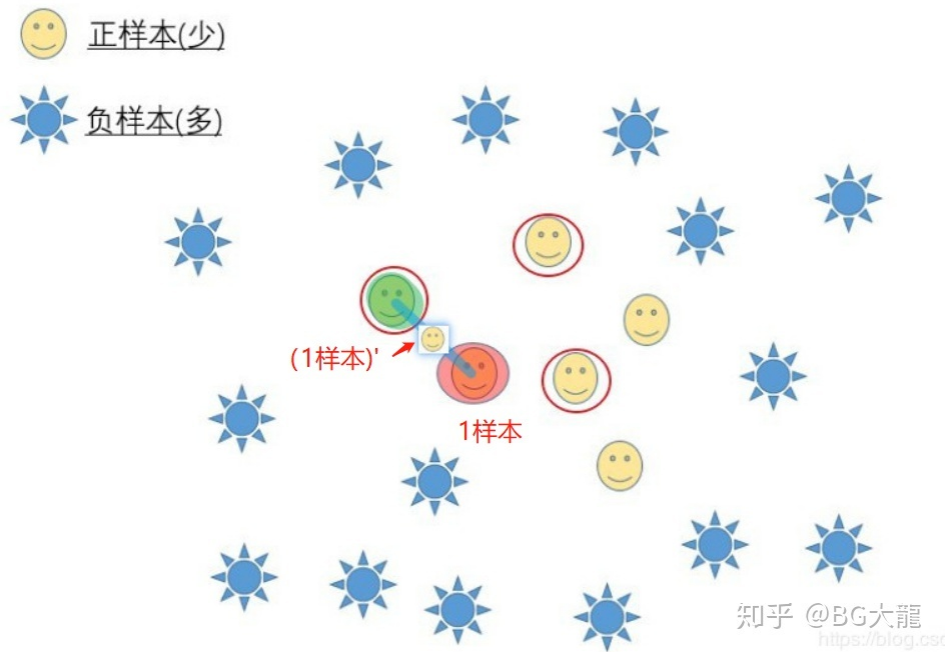
知乎 @BG大龍

[https://zhuanlan.zhihu.com/p/104111111](#)

步骤3-从1样本的K个近邻中，随机选出一个样本。在1样本和随机选出的这个近邻之间的连线上，随机找一点。这个点就是人工合成的新样本

$(1\text{样本})' = (1\text{样本}) + \text{随机数}R * D_{12}$ ，随机数R介于0-1之间





步骤4-想要找更多的，那就进行遍历.....

参考：SMOTE\_简单原理图示\_算法实现及R和Python调包简单实现 - 小小数分 - CSDN博客

[https://blog.csdn.net/Scce\\_hy/article/details/84190080](https://blog.csdn.net/Scce_hy/article/details/84190080)  
blog.csdn.net



知乎 @BG大龍

### 分步骤理解代码：

【步骤1】导入工具包

【步骤2】预处理

【步骤3】找到最佳参数

【步骤4】模型评估



知乎 @BG大龍

【步骤1】导入工具包

```
import pandas as pd
from imblearn.over_sampling import SMOTE
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
```

用anaconda安装imblearn, 如图即可

知乎 @BG大龍

[illegible]

```

# 读取数据集
credit_cards=pd.read_csv('creditcard.csv')
# 取数据集中的列
columns=credit_cards.columns
# 在特征中去除掉标签
features_columns=columns.delete(len(columns)-1)
#分成特征和标签两个
features=credit_cards[features_columns]
labels=credit_cards['Class']

#数据集的切分
features_train, features_test, labels_train, labels_test = train_test_split(features,
                                                                              labels,
                                                                              test_size=0.3,
                                                                              random_state=0)

```

知乎 @BG大龍



知乎 @BG大龍

### 【步骤3】找到最佳参数

#基于SMOTE算法来进行样本生成，这样正例（异常）和负例（正常）样本数量就是一致的了  
oversampler=SMOTE(random\_state=0)

os\_features,os\_labels=oversampler.fit\_sample(features\_train,labels\_train)

#计算样本数量

len(os\_labels[os\_labels==1])

#数据转换

os\_features = pd.DataFrame(os\_features)

os\_labels = pd.DataFrame(os\_labels)

# 交叉验证，找到最好的参数

best\_c = printing\_Kfold\_scores(os\_features,os\_labels)

```

#基于SMOTE算法来进行样本生成，这样正例（异常）和负例（正常）样本数量就是一致的了
oversampler=SMOTE(random_state=0)

```

```
os_features,os_labels=oversampler.fit_sample(features_train,labels_train)
```

```
#计算样本数量
```

```
len(os_labels[os_labels==1])
```

```
#数据转换
```

```
os_features = pd.DataFrame(os_features)
```

```
os_labels = pd.DataFrame(os_labels)
```

```
# 交叉验证，找到最好的参数
```

```
best_c = printing_Kfold_scores(os_features,os_labels)
```

知乎 @BG大龍

## 训练集样本数量

```
len(os_labels[os_labels==1])
```

199019

知乎 @BG大龍

## 结果:

正则化惩罚力度: 0.01

Iteration 1 : 召回率 = 0.914285714286  
Iteration 2 : 召回率 = 0.88  
Iteration 3 : 召回率 = 0.971699544765  
Iteration 4 : 召回率 = 0.962088761039  
Iteration 5 : 召回率 = 0.961724471466

平均召回率 0.937959698311

正则化惩罚力度: 0.1

Iteration 1 : 召回率 = 0.914285714286  
Iteration 2 : 召回率 = 0.88  
Iteration 3 : 召回率 = 0.972913505311  
Iteration 4 : 召回率 = 0.961347620184  
Iteration 5 : 召回率 = 0.963947894029

平均召回率 0.938498946762

正则化惩罚力度: 1

Iteration 1 : 召回率 = 0.914285714286  
Iteration 2 : 召回率 = 0.88  
Iteration 3 : 召回率 = 0.973065250379  
Iteration 4 : 召回率 = 0.96436243044  
Iteration 5 : 召回率 = 0.964400115568

平均召回率 0.939222702134

正则化惩罚力度: 10

Iteration 1 : 召回率 = 0.914285714286  
Iteration 2 : 召回率 = 0.88  
Iteration 3 : 召回率 = 0.973039959535  
Iteration 4 : 召回率 = 0.964412677277  
Iteration 5 : 召回率 = 0.963445425654

平均召回率 0.93903675535

正则化惩罚力度: 100

Iteration 1 : 召回率 = 0.914285714286  
Iteration 2 : 召回率 = 0.88  
Iteration 3 : 召回率 = 0.973065250379  
Iteration 4 : 召回率 = 0.964425238987  
Iteration 5 : 召回率 = 0.963922770611

平均召回率 0.939139794852

\*\*\*\*\*  
效果最好的模型所选参数 = 1.0  
\*\*\*\*\*

知乎 @BG大龍



知乎 @BG大龍

## 【步骤4】模型评估

```
# 建模。选算法+给参数
lr = LogisticRegression(C = best_c, penalty = 'l1')
# 训练模型
lr.fit(os_features,os_labels.values.ravel())
# 测试模型
y_pred = lr.predict(features_test.values)

# 计算混淆矩阵
cnf_matrix = confusion_matrix(labels_test,y_pred)
np.set_printoptions(precision=2)

print("召回率: ", cnf_matrix[1,1]/(cnf_matrix[1,0]+cnf_matrix[1,1]))

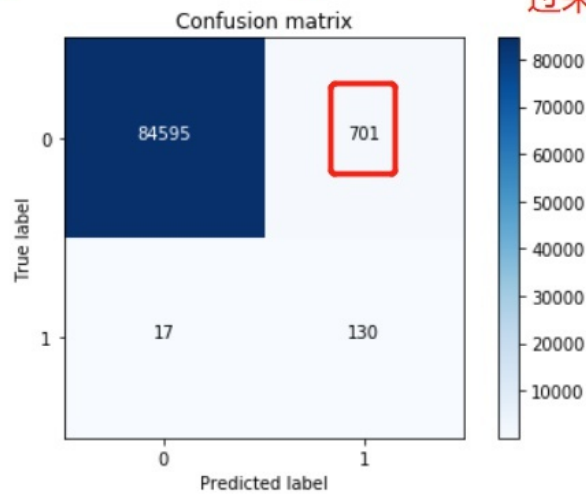
# 绘制
class_names = [0,1]
plt.figure()
plot_confusion_matrix(cnf_matrix
                      , classes=class_names
                      , title='Confusion matrix')

plt.show()
```

结果:

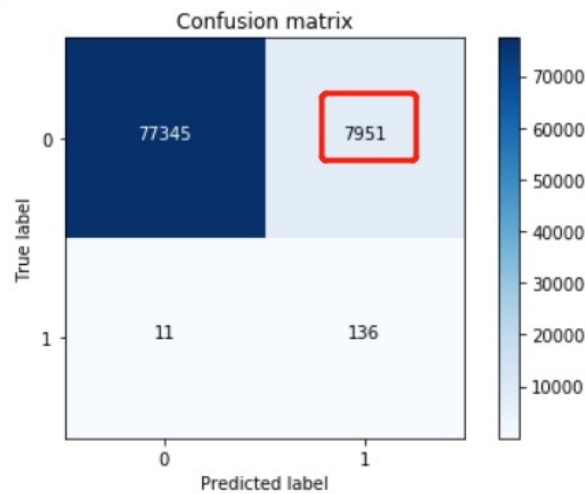
【过采样】漏检17+错检701, 【下采样】漏检11+错检7951, 【过采样】更好

召回率: 0.884353741497



知乎 @BG大龍

召回率: 0.925170068027




知乎 @BG大龍

## 15. 【项目总结】

(数据预处理) ——》(针对数据情况, 提出不同解决方案) ——》选好评估方法, 找最佳参数  
——》建模=选算法+调参——》从结果出发, 根据应用场景去选择方案


## 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/79055095](https://blog.csdn.net/stranger_man/article/details/79055095)  
 [blog.csdn.net](https://blog.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=1>  
 [blog.csdn.net](https://blog.csdn.net)



- 3、python 机器学习实战：信用卡欺诈异常值检测 - dengheCSDN的博客 - CSDN博客

<https://blog.csdn.net/dengheCSDN/article/details/79079364>  
 [blog.csdn.net](https://blog.csdn.net)



编辑于 2019-08-11

机器学习

Python 开发