一、 实验流程

基本步骤如下所示：

1.加载数据，观察问题

①导入所需工具包

②数据读取

③查看数据标签分布

④数据标准化处理

2.数据集切分

①下采样方案

②数据集划分

3.逻辑回归模型

①调用逻辑回归模型

②交叉验证与不同参数结果

4.建模结果分析

①下采样方案在原始数据集中的结果

②原始数据直接建模结果

③阈值对结果的影响

5.方案效果对比

①SMOTE 过采样方案

②基于 SMOTE 算法来进行样本生成

实验流程图如下：

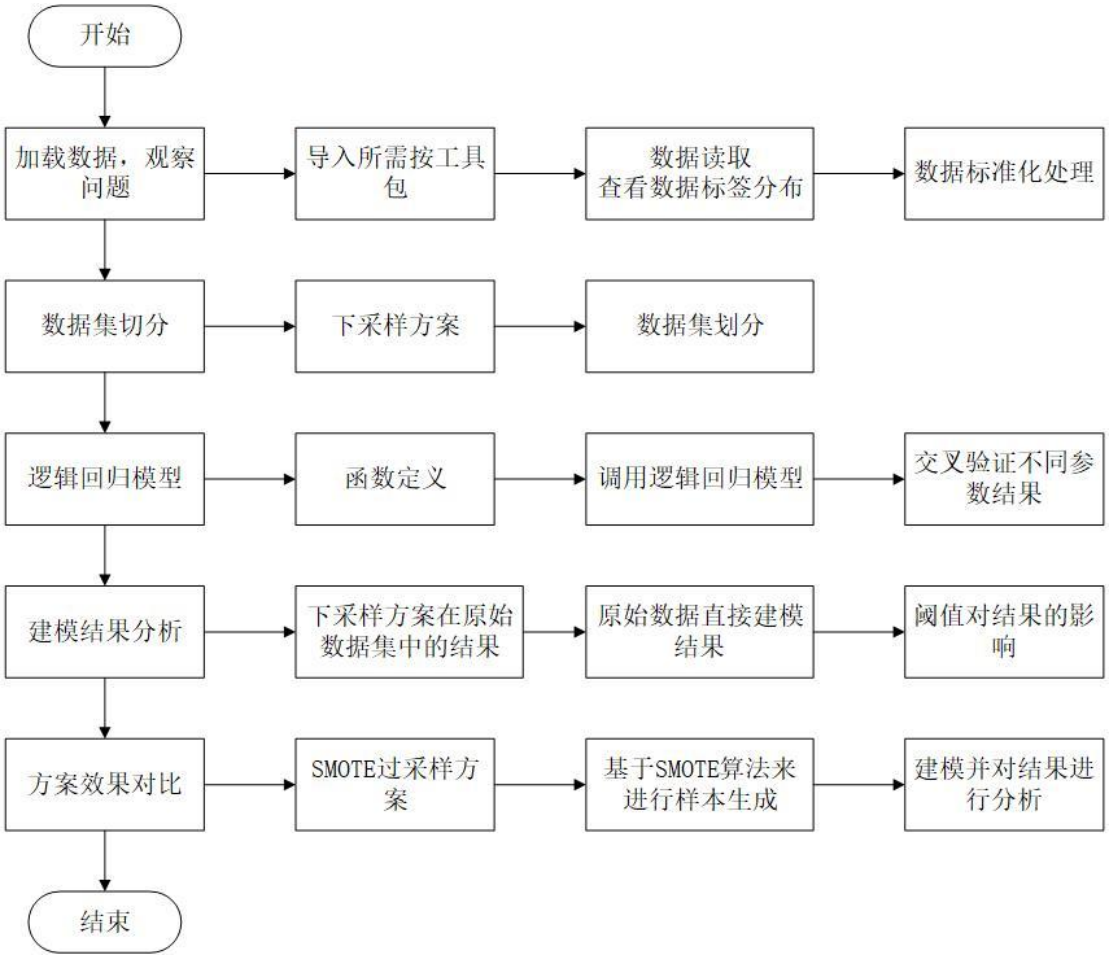


图 1-1 实验流程图

二、 实验代码

1. 项目挑战与解决方案制定

- 导入工具包

```
import pandas as pd  #数据处理和数据分析
import matplotlib.pyplot as plt  #可视化展示
import numpy as np   #矩阵计算

%matplo   tlib inline  #可以在 Notebook 中直接画图
```

- 数据读取

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

Out[56]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	V24	V25
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	0.12853
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	0.16717
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	-0.32764
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321	-1.175575	0.64737
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458	0.141267	-0.20601

5 rows × 31 columns

图 2-1 数据读取截图

2. 数据标签分布

当前数据出现的个数，并排序；然后绘图

```
    count_classes = pd.value_counts(data['Class'], sort =
True).sort_index()
    count_classes.plot(kind = 'bar')
    plt.title("Fraud class histogram")
    plt.xlabel("Class")
    plt.ylabel("Frequency")
```

```
Text(0, 0.5, 'Frequency')
```

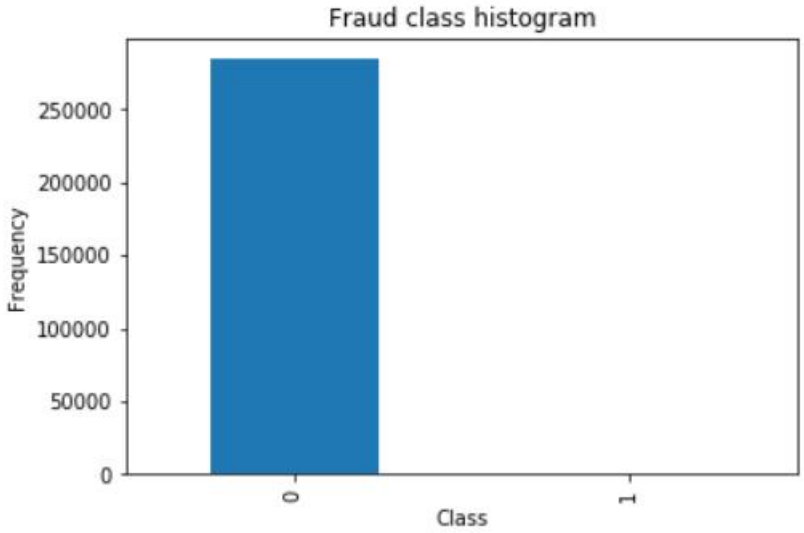


图 2-2 数据标签分布

- 希望 0 和 1 一样多（过采样：oversample）-》造假数据，可能导致模型结果下降
- 0 和 1 一样少（下采样：undersample）-》数据没有充分利用，效果下降。

3. 数据标准化处理

```
    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()
```

4. 下采样数据集制作

选 500 个

```
    X = data.iloc[:, data.columns != 'Class']
    y = data.iloc[:, data.columns == 'Class']

    # 得到所有异常样本的索引
    number_records_fraud = len(data[data.Class == 1])
    fraud_indices = np.array(data[data.Class == 1].index)

    # 得到所有正常样本的索引
    normal_indices = data[data.Class == 0].index

    # 在正常样本中随机采样出指定个数的样本，并取其索引
    random_normal_indices = np.random.choice(normal_indices,
number_records_fraud, replace = False)
    random_normal_indices = np.array(random_normal_indices)

    # 有了正常和异常样本后把它们的索引都拿到手
    under_sample_indices =
np.concatenate([fraud_indices,random_normal_indices])

    # 根据索引得到下采样所有样本点
    under_sample_data = data.iloc[under_sample_indices,:]

    X_undersample = under_sample_data.iloc[:,
under_sample_data.columns != 'Class']
    y_undersample = under_sample_data.iloc[:,
under_sample_data.columns == 'Class']

    # 下采样 样本比例
    print("正常样本所占整体比例: ",
len(under_sample_data[under_sample_data.Class ==
0])/len(under_sample_data))
    print("异常样本所占整体比例: ",
len(under_sample_data[under_sample_data.Class ==
1])/len(under_sample_data))
    print("下采样策略总体样本数量: ", len(under_sample_data))
```

```
正常样本所占整体比例:  0.5
异常样本所占整体比例:  0.5
下采样策略总体样本数量:  984
```

图 2-3 下采样数据分析

5. 数据集切分

```
    from sklearn.model_selection import train_test_split
    # 整个数据集进行划分
    X_train, X_test, y_train, y_test =
train_test_split(X,y,test_size = 0.3, random_state = 0)
    print("原始训练集包含样本数量: ", len(X_train))
    print("原始测试集包含样本数量: ", len(X_test))
    print("原始样本总数: ", len(X_train)+len(X_test))

    # 下采样数据集进行划分
    X_train_undersample, X_test_undersample, y_train_undersample,
y_test_undersample = train_test_split(X_undersample

                                    ,y_undersample

                                    ,test_size = 0.3

                                    ,random_state = 0)
    print("")
    print("下采样训练集包含样本数量: ", len(X_train_undersample))
    print("下采样测试集包含样本数量: ", len(X_test_undersample))
    print("下采样样本总数: ",
len(X_train_undersample)+len(X_test_undersample))
```

```
原始训练集包含样本数量:  199364
原始测试集包含样本数量:  85443
原始样本总数:  284807

下采样训练集包含样本数量:  688
下采样测试集包含样本数量:  296
下采样样本总数:  984
```

图 2-4 原始数据和下采样数据对比

6. 训练逻辑回归模型

```
    def printing_Kfold_scores(x_train_data,y_train_data):
        fold = KFold(5,shuffle=False)

        # 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','Mean
recall score'])
        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
        for c_param in c_param_range:
            print('-------------------------------------------')
            print('正则化惩罚力度: ', c_param)
            print('-------------------------------------------')
            print('')

            recall_accs = []

            #一步步分解来执行交叉验证  iteration：第几次验证
indices：索引标签
            for iteration, indices in
enumerate(fold.split(x_train_data)):

                # 指定算法模型，并且给定参数
                lr = LogisticRegression(C = c_param, penalty =
'l1',solver='liblinear')

                # 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[indic
es[0],:].values.ravel())

                # 建立好模型后，预测模型结果，这里用的就是验证集，索引
为 1
                y_pred_undersample =
lr.predict(x_train_data.iloc[indices[1],:].values)

                # 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_und
ersample)
                # 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)

            # 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
            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_parameter']

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

交叉验证与不同参数结果

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

通过定义不同力度的正则化惩罚力度，分别是0.01,0.1,1,10,100。一步步分解来执行交叉验证，当执行完所有的交叉验证后，计算平均结果。找到最好的参数，哪一个 Recall 高，自然就是最好的了。打印最好的结果如下（过程截图太长省略）

```
-------------------------------------------
正则化惩罚力度:  0.1
-------------------------------------------

Iteration  1 : 召回率 =  0.8493150684931506
Iteration  2 : 召回率 =  0.863013698630137
Iteration  3 : 召回率 =  0.9661016949152542
Iteration  4 : 召回率 =  0.9459459459459459
Iteration  5 : 召回率 =  0.9242424242424242

平均召回率  0.9097237664453823
```

图 2-5 惩罚力度为 0.1 的召回率

```
*********************************************************************************
效果最好的模型所选参数 =  0.01
*********************************************************************************
```

图 2-6 通过比较得出最好的模型参数

7. 混淆矩阵评估分析

Recall = 131/（18+131）

正确率 = （131+18）/ 总和

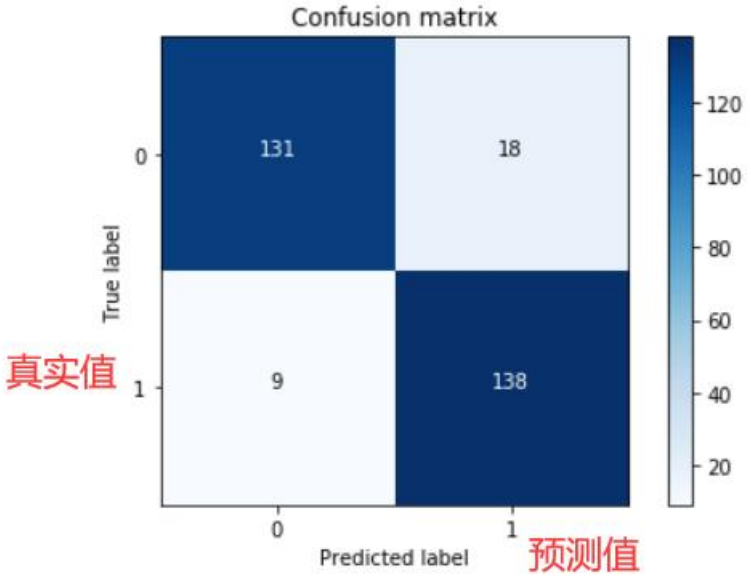


图 2-7 混淆矩阵分析

8. 测试集遇到的问题

下采样方案在原始数据集中的结果如下：

召回率： 0.9319727891156463

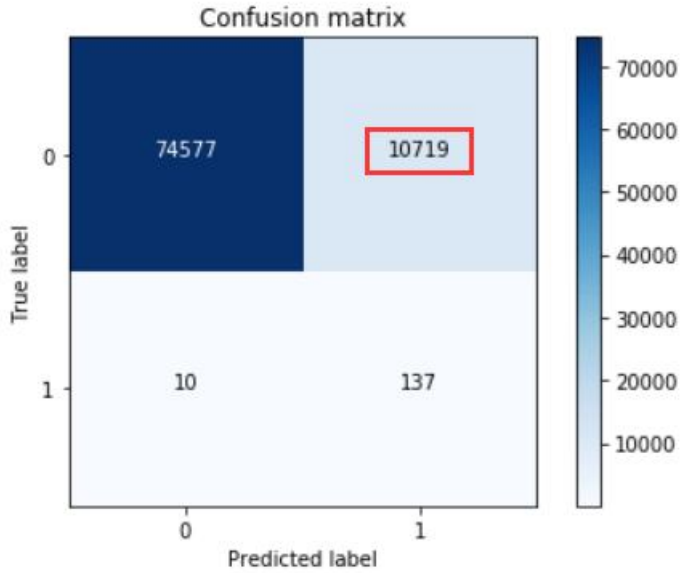


图 2-8 原始数据结果分析

问题是误差过大。于是看原始数据直接建模结果：

Recall metric in the testing dataset: 0.6190476190476191

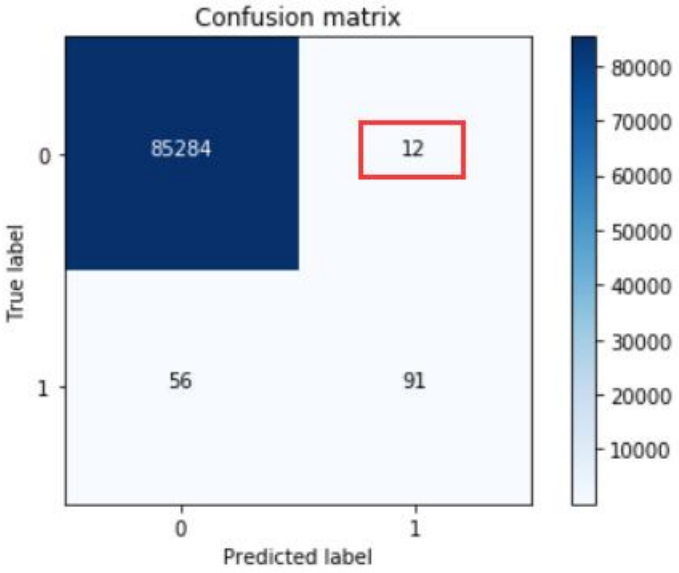


图 2-9 原始数据直接建模

9. 阈值对结果的影响

```
# 用之前最好的参数来进行建模
    lr = LogisticRegression(C = 0.01, penalty =
'l1',solver='liblinear')
```

```
    # 训练模型，还是用下采样的数据集

lr.fit(X_train_undersample,y_train_undersample.values.ravel())

    # 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
    thresholds = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]

    plt.figure(figsize=(10,10))

    j = 1

    # 用混淆矩阵来进行展示
    for i in thresholds:
        #预测完的结果
        y_test_predictions_high_recall =
y_pred_undersample_proba[:,1] > i

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

        cnf_matrix =
confusion_matrix(y_test_undersample,y_test_predictions_high_re
call)
        np.set_printoptions(precision=2)

        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]))

        class_names = [0,1]
        plot_confusion_matrix(cnf_matrix
                              , classes=class_names
                              , title='Threshold >= %s'%i)
```

```
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:  0.9931972789115646
给定阈值为: 0.5 时测试集召回率:  0.9387755102040817
给定阈值为: 0.6 时测试集召回率:  0.891156462585034
给定阈值为: 0.7 时测试集召回率:  0.8367346938775511
给定阈值为: 0.8 时测试集召回率:  0.7755102040816326
给定阈值为: 0.9 时测试集召回率:  0.5918367346938775
```

图 2-10 不同阈值得到的召回率

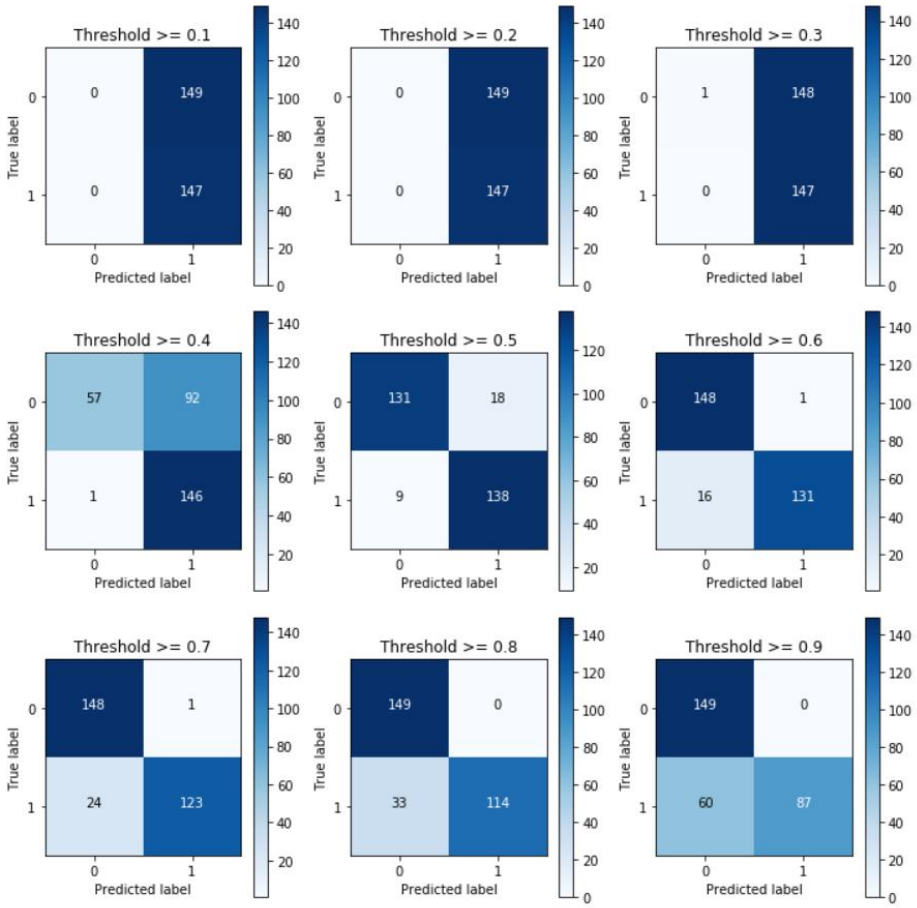


图 2-11 混沌矩阵分析

10. SMOTE 样本生成策略

(1) 针对少数样本做

(2) 计算少数样本中，到其他样本的距离 d

(3) 按距离从小到大排序 d12, d13, d14

(4) 选择倍率 k=2（选择最近的 k 个）

(5)选 k 个样本 d12,d13　　x1'=x1+random(0,1)·d12

x1''=x1+random(0,1)·d13

安装工具包 imblearn：pip3 install imblearn。安装过程会因为 python 环境和电脑版本的不同遇到不同问题，主要体现为访问 anaconda 官网的过程会有问题，可以换用国内的镜像源以及百度找问题的答案，前人已经为我们铺好了路。

11. 过采样效果

- 导包

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

- 读数据，找到特征和标签

```
credit_cards=pd.read_csv('creditcard.csv')

columns=credit_cards.columns
# 在特征中去除掉标签
features_columns=columns.delete(len(columns)-1)

features=credit_cards[features_columns]
labels=credit_cards['Class']
```

- 对数据集进行切分

```
  features_train, features_test, labels_train, labels_test =
train_test_split(features, labels, test_size=0.3,
random_state=0)
```

- 样本生成。基于 SMOTE 算法来进行样本生成，这样正例和负例样本数量就是一致的了，得到了比例均衡的样本。

```
oversampler=SMOTE(random_state=0)

os_features,os_labels=oversampler.fit_sample(features_train,la
bels_train)
```

- 训练集样本数量

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

图 2-12 训练集样本数量

```
os_features = pd.DataFrame(os_features)
os_labels = pd.DataFrame(os_labels)
best_c = printing_Kfold_scores(os_features,os_labels)
```

通过定义不同力度的正则化惩罚力度，分别是 0.01, 0.1, 1, 10, 100。

打印最好的结果如下（过程截图太长省略）

```
-------------------------------------------
正则化惩罚力度:  100
-------------------------------------------

Iteration  1 : 召回率 =  0.9142857142857143
Iteration  2 : 召回率 =  0.88
Iteration  3 : 召回率 =  0.9730905412240769
Iteration  4 : 召回率 =  0.9644252389865213
Iteration  5 : 召回率 =  0.9642368133455601

平均召回率  0.9392076615683745

*********************************************************************************
效果最好的模型所选参数 =  100.0
*********************************************************************************
```

图 2-13 不同惩罚力度的召回率

- 混淆矩阵

```
    lr = LogisticRegression(C = best_c, penalty =
'l1',solver='liblinear')

    lr.fit(os_features,os_labels.values.ravel())
    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()
```

召回率： 0.8843537414965986

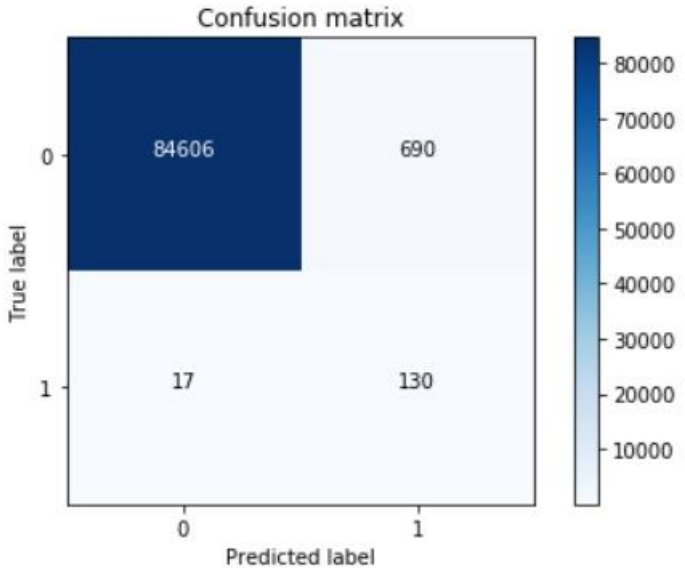
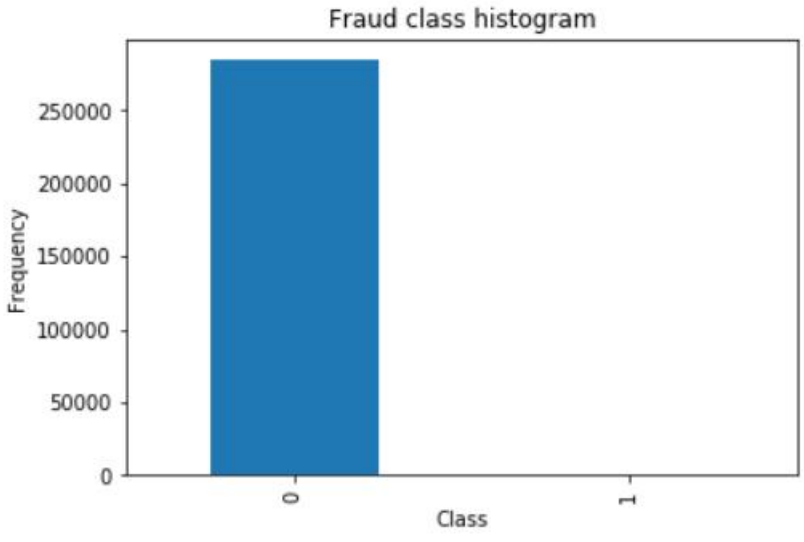


图 2-14 过采样分析

三、实验问题

Q1：附上原始数据标签分布图。

Text(0, 0.5, 'Frequency')



答：原始数据标签分布图如上图（以及图 2-2）所示，从分布图中可以得出，原始数据分布集中，存在数据分布不均匀的情况，即存在信用卡欺诈行为的样本数量远小于正常样本数量。从而引出下文我们适用下采样和过采样的方案解决这个问题

Q2：关于阈值对结果的影响，请附上给定不同阈值时，测试集的召回率，以及混淆矩阵。

答：如上一章节的图 2-10 和图 2-11 所示，具体内容为下面两个图片所示：

从图 2-10 中我们可以得出，在阈值为 0.1、0.、0.3 的时候，测试机的召回率都是 1，及所有样本都被成功预测；当阈值为 0.4 时，召回率为 0.993；当阈值为 0.5 时，召回率为 0.938；当阈值为 0.6 时，召回率为 0.891；当阈值为 0.7 时，召回率为 0.836；当阈值为 0.8 时，召回率为 0.775；当阈值为 0.9 时，召回率为 0.591；

综上可以看出，当阈值设置较小时，对异常样本的检测效果较好，但是此时的模型精度较低；当阈值开始增大时，模型的精度会逐渐升高，对异常样本的检测效果会略微降低；但是当阈值过大时，模型的精度会适当减少，并且此时对异常样本的检测效果也大大降低；因此应该选择合适的阈值，才能够得到最好的检测效果。

```
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：  0.9931972789115646
给定阈值为：0.5 时测试集召回率：  0.9387755102040817
给定阈值为：0.6 时测试集召回率：  0.891156462585034
给定阈值为：0.7 时测试集召回率：  0.8367346938775511
给定阈值为：0.8 时测试集召回率：  0.7755102040816326
给定阈值为：0.9 时测试集召回率：  0.5918367346938775
```

图 2-10 不同阈值得到的召回率

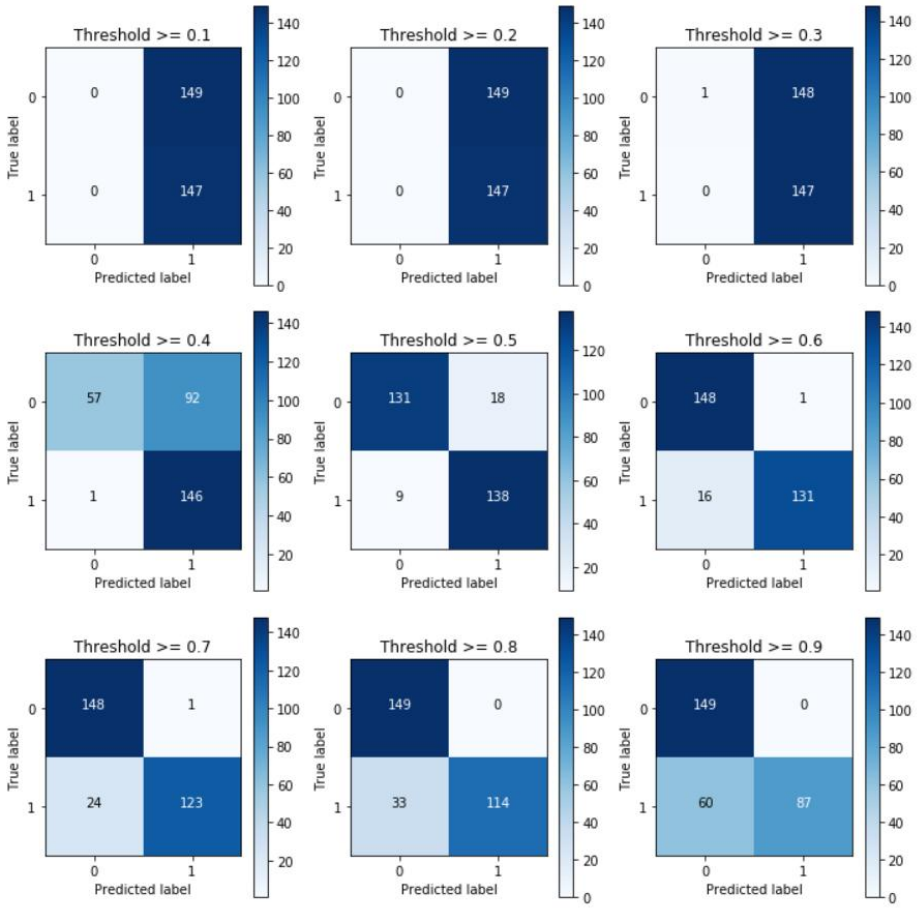


图 2-11 混沌矩阵分析

Q3：讨论用到的两种方法，下采样和过采样，哪种更适合我们的模型?

召回率： 0.9115646258503401

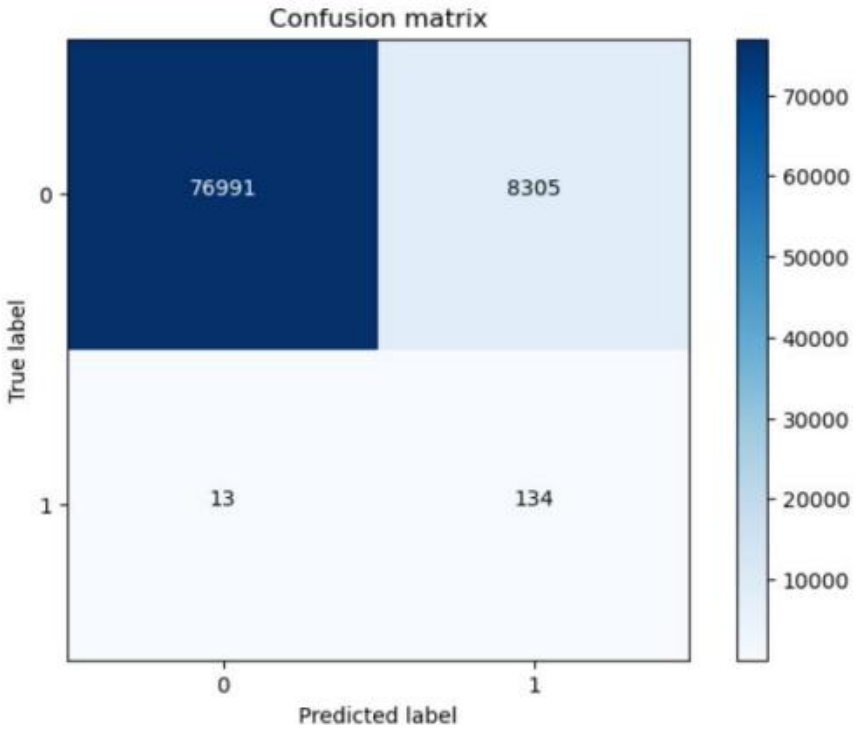


图 3-1 下采样方案

召回率： 0.8843537414965986

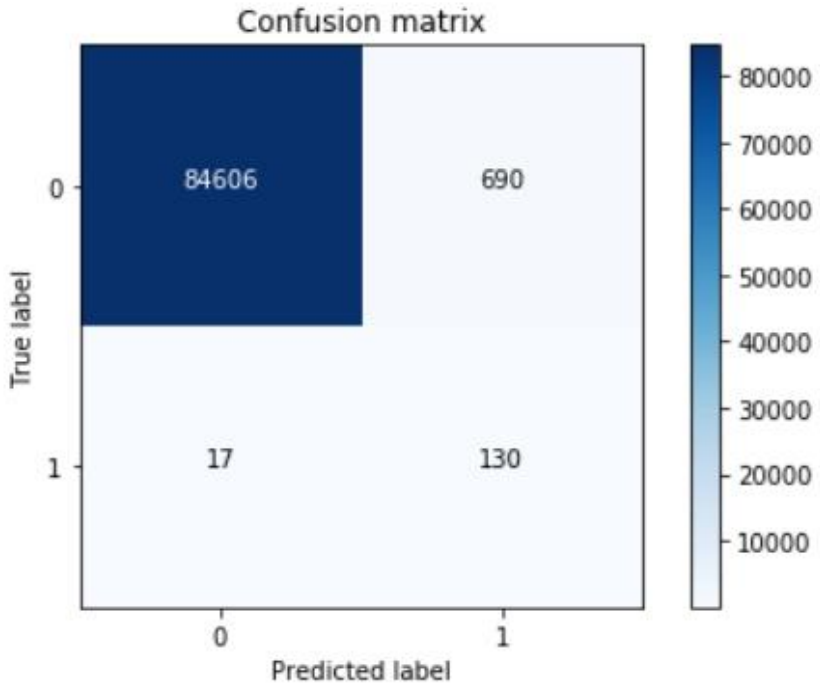


图 3-2 过采样方案

答：下采样方案可以得到召回率为 0.931，精度为（76991+134）

/（76991+8305+13+134）=0.903；图 3-2 采用过采样方案对数据集进行预测得出的混淆矩阵和召回率，召回率为0.884，精度为（84606+130）/（84606+690+17+130）=0.991。对比可知，虽然过采样方案对于异常样本的检测效果略差，但是其对于整体样本的预测精度大大提高，降低了误判的概率；因此在本次实验中，过采样更加适合我们的模型；在出现数据不均衡的情况下，较经常使用的是生成数据而不是减少数据，但是随着数据量的增加，会导致训练时间变长。

Q4：探讨调参对结果的影响。

答：逻辑回归是一种常用的分类算法，在实际应用中需要对其进行参数调优以提高模型的准确率和鲁棒性。逻辑回归模型调参对结果的影响主要包括以下几个方面：

- 正则化参数：逻辑回归中的正则化参数有两种，分别是 L1 正则化和 L2 正则化。正则化参数的选择可以控制模型的复杂度，过大或过小的正则化参数都可能导致模型效果不佳。
- 学习率：学习率控制了模型在每一步的调整幅度，过小的学习率会导致模型收敛缓慢，过大的学习率会导致模型震荡或发散。
- 迭代次数：迭代次数控制了模型的收敛速度，过小的迭代次数会导致模型未能完全收敛，过大的迭代次数会导致模型过拟合。
- 特征选择：逻辑回归模型需要选取对分类结果有重要影响的特征进行建模，不同的特征选择方法可能会对模型的准确率和鲁棒性产生不同的影响。

总之，逻辑回归模型调参对结果的影响很大，正确的调参方法和评估指标可以提高模型的准确率，从而更好地解决实际问题。

四、总结与体会

逻辑回归是一种常见的分类算法，通常用于处理二分类问题。在实际应用中，逻辑回归常被用于预测二元事件的发生概率，例如信用违约、疾病诊断等领域。

在应用逻辑回归进行建模时，需要考虑多种因素，如特征选择、模型训练、参数优化等。特别是在特征选择方面，需要结合实际场景，选择对分类结果影响较大的特征进行建模，以提高模型的准确率和鲁棒性。

同时，在交易数据异常检测方面，逻辑回归也具有一定的应用价值。例如，在支付欺诈检测方面，可以通过分析交易数据中的各项特征，如交易金额、支付方式等，预测该交易是否为欺诈交易。

然而，在实际应用中，逻辑回归仍存在一些限制。例如，对于多类别问题，逻辑回归的效果不如其他分类算法，如决策树、随机森林等。此外，在数据量较大时，逻辑回归的训练时间也可能较长。

在进行逻辑回归建模时，需要注意以下几点：

1. 特征选择：在选择特征时，需要结合实际场景选择对分类结果影响较大的特征。
2. 数据清洗：在进行建模前，需要对数据进行清洗，去除异常值和缺失值等。
3. 参数调优：逻辑回归模型中有多个参数，需要进行参数调优以提高模型的准确率和鲁棒性。
4. 模型评估：在训练完模型后，需要对模型进行评估，以确定其准确率和召回率等指标，并进行模型优化。

在交易数据异常检测方面，需要考虑以下几点：

1. 特征选择：需要选择与欺诈检测相关的特征进行建模，如交易金额、交易时间、支付方式等。
2. 模型建立：可以使用逻辑回归等分类算法对交易数据进行建模，以预测交易是否为欺诈交易。
3. 模型评估：需要对模型进行评估，以确定其准确率和召回率等指标，并进行模型优化。

综上所述，逻辑回归在分类问题和异常检测等领域都有广泛的应用，但在实际应用中需要注意特征选择、数据清洗、参数调优和模型评估等问题，以提高模型的准确率和鲁棒性。

通过本次实验的学习，我更深入的了解了人工智能和机器学习，在未来的发展中，我也会利用自己所学，继续提高自己。面对困难不言放弃，坚持努力，车到山前必有路。