实验目录

实验内容

1. 基于逻辑回归的信用欺诈预测识别模型

知识点

1) 逻辑回归通过预测分类概率判断分类
2) 对于平衡集，默认分类概率阈值 0.5
3) 对于不平衡集，使用阈值 0.5 可能和业务要求不符
4) 可以通过重抽样或 ROC 选择阈值的方法解决不平衡集问题

实验目的

1) 学习在逻辑回归中使用重抽样方法
2) 学习在逻辑回归中使用 ROC 计算概率阈值
3) 比较数据集未处理、重抽样、ROC 方法的逻辑回归模型结果

实验步骤

1) 打开 Jupyter，并新建 python 工程

2) 读取数据

1. Jupyter 输入代码后，使用 shift+enter 执行，下同。
2. 数据集包含欧洲持卡人于 2013 年 9 月通过信用卡进行的交易。数据集提供两天内的交易数据，在 284,807 笔交易中有 492 起欺诈行为。数据集非常不平衡，正面类别（欺诈）占所有交易的 0.172%。数据经过脱敏处理，V1~V28 是主成分，Time 是每次交易与第一次交易之间距离的时间，单位为秒。Amount 代表消费金额，Class 为因变量，1 表示欺诈，0 表示正常。
3. 使用 pandas 读取 csv 文件

[Code 001]:

```
import pandas as pd
df = pd.read_csv('/root/experiment/datas/creditcard.csv')
# 查看数据维度
df.shape
```

```
import pandas as pd

df = pd.read_csv('/root/experiment/datas/creditcard.csv')
df.shape
```

```
(284807, 31)
```

3) 描述性分析与可视化分析

1. 查看数据的随机五项

[Code 002]:

df.sample(5)

```
df.sample(5)
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23	
92906	64162.0	1.271152	-0.094375	0.578396	0.583881	-0.682050	-0.517704	-0.254191	-0.177471	-1.299405	...	-0.224204	-0.035673	0.022118	0.3
181991	125209.0	2.060753	-0.106978	-1.077315	0.426283	-0.218084	-1.237582	0.132239	-0.291339	0.658977	...	-0.297535	-0.761412	0.350515	0.0
62856	50459.0	0.866669	-0.274743	0.360535	1.431041	-0.360001	0.060282	0.097299	0.118012	0.086117	...	0.032325	-0.017397	-0.152403	0.2
191478	129270.0	-2.114596	0.956760	-3.404809	-0.670049	-0.991710	-0.258851	0.878415	1.121784	-0.762714	...	0.280020	0.934799	0.381783	-0.9
219250	141651.0	-0.408603	0.548039	1.465247	-0.961440	0.205615	-0.142031	0.436857	0.137149	-0.167917	...	-0.146824	-0.456057	-0.004966	-0.3

5 rows × 31 columns

2. 查看缺失值

[Code 003]:

df.isnull().sum().sum()

```
df.isnull().sum().sum()
```

```
0
```

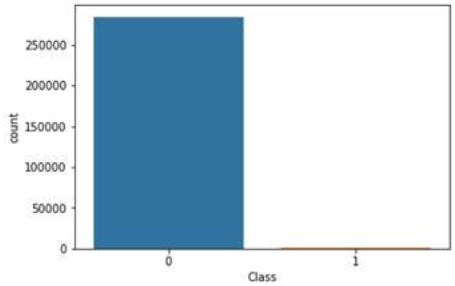
3. 查看因变量分布，因变量极不平衡（绘图时，由于 jupyter 的问题，执行时可能需重复执行才能显示绘图结果，下同）

[Code 004]:

import matplotlib.pyplot as plt
import seaborn as sns
sns.countplot(x='Class', data=df)
plt.show()

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.countplot(x='Class', data=df)
plt.show()
```

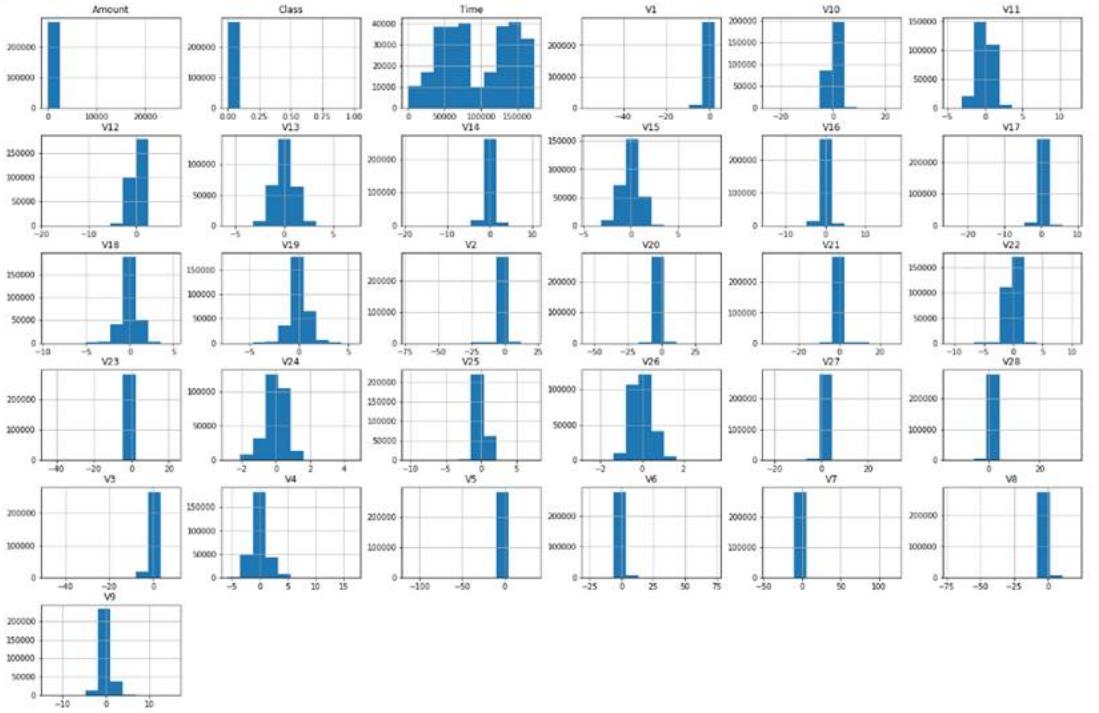


4. 查看数据分布

[Code 005]:

df.hist(figsize=(24,16))
plt.show()

```
df.hist(figsize=(24,16))
plt.show()
```



4) 数据预处理

1. 字段处理

[Code 006]:

 # 对 Amount 进行标准化处理，存储在 normal_amount 字段中
from sklearn.preprocessing import StandardScaler
df['normal_amount'] =
StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
 # 去掉 Amount 和 Time 字段
df = df.drop(['Amount','Time'], axis=1)
df.columns

```
from sklearn.preprocessing import StandardScaler

df['normal_amount'] = StandardScaler().fit_transform(df['Amount'].values.reshape(-1,1))
df = df.drop(['Amount','Time'], axis=1)

df.columns
```

```
Index(['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11',
       'V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21',
       'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'Class',
       'normal_amount'],
      dtype='object')
```

2. 划分自变量和因变量，训练集和测试集

[Code 007]:

 # 划分自变量和因变量
X = df.loc[:,df.columns != 'Class']

y = df.loc[:,df.columns == 'Class']
 # 划分训练集和测试集
from sklearn.model_selection import train_test_split
X_tr,X_ts,y_tr,y_ts = train_test_split(X,y,test_size=0.2)
X_tr.shape,X_ts.shape

```
X = df.loc[:,df.columns != 'Class']
y = df.loc[:,df.columns == 'Class']
```

```
from sklearn.model_selection import train_test_split

X_tr,X_ts,y_tr,y_ts = train_test_split(X,y,test_size=0.2)
X_tr.shape,X_ts.shape
```

```
((227845, 29), (56962, 29))
```

5) 建立模型

1. 建立普通逻辑回归模型

[Code 008]:

from sklearn.linear_model import LogisticRegression
model_original = LogisticRegression()
model_original.fit(X_tr,y_tr.values.ravel())

```
from sklearn.linear_model import LogisticRegression

model_original = LogisticRegression()
model_original.fit(X_tr,y_tr.values.ravel())
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

2. 建立重抽样平衡后的逻辑回归模型

[Code 009]:

model_rs = LogisticRegression(class_weight='balanced')
model_rs.fit(X_tr,y_tr.values.ravel())

```
model_rs = LogisticRegression(class_weight='balanced')
model_rs.fit(X_tr,y_tr.values.ravel())
```

```
LogisticRegression(C=1.0, class_weight='balanced', dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

3. 建立普通逻辑回归模型，使用 ROC 曲线选取阈值

[Code 010]:

model_roc = LogisticRegression()
model_roc.fit(X_tr,y_tr.values.ravel())

```
model_roc = LogisticRegression()
model_roc.fit(X_tr,y_tr.values.ravel())
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

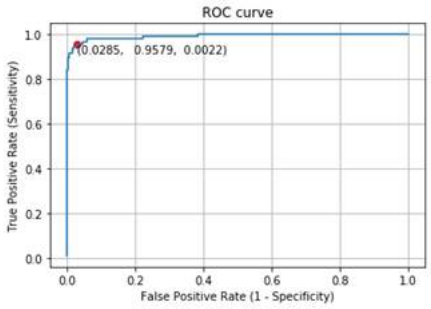
4. 使用 ROC 计算 FPR、TPR 以及对应的阈值

[Code 011]:

```
from sklearn.metrics import roc_curve
import numpy as np
preds = model_roc.predict_proba(X_ts)[:,1]
fpr, tpr, thresh = roc_curve(y_ts, preds)
df = pd.DataFrame(dict(fpr=fpr, tpr=tpr))
# 计算 ROC 曲线最优阈值
idx = np.argmax(tpr-fpr)
Thresh = thresh[idx]
# 绘制 ROC 曲线，查看阈值
plt.plot(fpr,tpr)
plt.scatter(fpr[idx],tpr[idx],c='r')
plt.text(fpr[idx],tpr[idx],'(%.4f,    %.4f,    %.4f)'%(fpr[idx],tpr[idx],Thresh),v
a='top')
plt.title('ROC curve')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```

```
from sklearn.metrics import roc_curve
import numpy as np

preds = model_roc.predict_proba(X_ts)[:,1]
fpr, tpr, thresh = roc_curve(y_ts, preds)

df = pd.DataFrame(dict(fpr=fpr, tpr=tpr))
idx = np.argmax(tpr-fpr)
Thresh = thresh[idx]
```

```
plt.plot(fpr,tpr)

plt.scatter(fpr[idx],tpr[idx],c='r')
plt.text(fpr[idx],tpr[idx],'(%.4f,    %.4f,  %.4f)'%(fpr[idx],tpr[idx],Thresh),va='top')

plt.title('ROC curve')
plt.xlabel('False Positive Rate (1 - Specificity)')
plt.ylabel('True Positive Rate (Sensitivity)')
plt.grid(True)
```



6) 模型预测与评估

1. 使用普通逻辑回归模型对测试集预测，查看混淆矩阵

[Code 012]:

```
y_original_pred = model_original.predict(X_ts)
from sklearn.metrics import confusion_matrix,classification_report
confusion_matrix(y_ts,y_original_pred)
```

```
y_original_pred = model_original.predict(X_ts)
```

```
from sklearn.metrics import confusion_matrix,classification_report

confusion_matrix(y_ts,y_original_pred)
```

```
array([[56858,     9],
       [   32,    63]])
```

2. 使用重抽样方法对测试集预测，并查看混淆矩阵

[Code 013]:

y_rs_pred = model_rs.predict(X_ts)
confusion_matrix(y_ts,y_rs_pred)

```
y_rs_pred = model_rs.predict(X_ts)
confusion_matrix(y_ts,y_rs_pred)
```

```
array([[55505,  1362],
       [    4,    91]])
```

3. 使用 ROC 阈值预测，并查看混淆矩阵

[Code 014]:

pred_prob = model_roc.predict_proba(X_ts)
pred_thresh = [int(x[1]>Thresh) for x in pred_prob]
confusion_matrix(y_ts, pred_thresh)

```
pred_prob = model_roc.predict_proba(X_ts)
pred_thresh = [int(x[1]>Thresh) for x in pred_prob]
confusion_matrix(y_ts, pred_thresh)
```

```
array([[55244,  1623],
       [    5,    90]])
```

7) 实验结论

1. 对于信用欺诈分析而言，业务目的是尽可能准确的识别欺诈用户，为此可以一定程度上“忍受”对非欺诈客户的“误伤”。
2. 未经处理的数据集模型，识别出 63 名欺诈用户，放过 32 名欺诈用户，误伤 9 名非欺诈用户。
3. 经过重抽样的模型，识别出 91 名欺诈用户，放过 4 名欺诈用户，误伤 1362 名非欺诈用户。
4. 使用 ROC 的模型，识别出 90 名欺诈用户，放过 5 名欺诈用户，误伤 1623 名非欺诈用户。
5. 本次分析中，原始的逻辑回归模型未能有效识别出欺诈用户，重抽样方法和 ROC 方法都比较准确的识别出了欺诈用户。
6. 本次分析中，重抽样方法略优于 ROC 方法。