逻辑回归之信用评估实例

链接：<https://blog.csdn.net/qq_41627642/article/details/104690186>

## 一、数据展示

第一步我们先导入数据，查看一下要区分的类别得类别数，以及每个类别得样本数，并进行展示。信用贷款得CLASS只有两类，0和1,两者拥有的数据量差异极大，面临数据不平衡问题。

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import os

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression#引入逻辑回归分类器

from sklearn.model\_selection import KFold, cross\_val\_score#引入K折交差验证

from sklearn.metrics import confusion\_matrix,recall\_score,classification\_report #引入混淆矩阵，召回率，以及分类报告

import itertools

from imblearn.over\_sampling import SMOTE

if \_\_name\_\_=="\_\_main\_\_":

path="D:\Python base\Test\逻辑回归-信用卡欺诈检测"+os.sep +"creditcard.csv"

data= pd.read\_csv(path)

if \_\_name\_\_=="\_\_main\_\_":

path="D:\Python base\Test\逻辑回归-信用卡欺诈检测"+os.sep +"creditcard.csv"

data= pd.read\_csv(path)

#1.1进行数据展示

class\_number=data.loc[:,'Classnew'].value\_counts()#统计某一列各个数值得出现次数

count\_classes = pd.value\_counts(data['Classnew'], sort = True).sort\_index()#统计某一列各个数值得出现次数

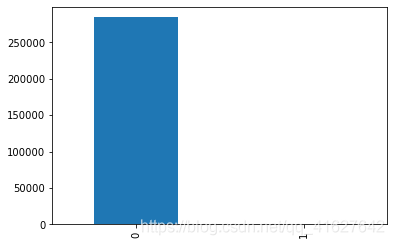
class\_number.plot(kind = 'bar')#kind可以是’line’, ‘bar’, ‘barh’, ‘kde’

plt.title("Fraud class histogram")

plt.xlabel("Class")

plt.ylabel("Frequency")

plt.show()



## 二、归一化和下采样

#对某列数据进行归一化

data['normAmount'] = StandardScaler().fit\_transform(data['Amount'].values.reshape(-1, 1))#对某列数据进行归一化

data = data.drop(['Time','Amount'],axis=1)#删除没用的两列

X=data.loc[:,data.columns != "Classnew"]

y=data.loc[:,data.columns == "Classnew"]

#1.2进行下采样

class1=data[data.Classnew == 1] #利用条件充当Index挑选class=1的数据

print("class=1的数据为：",class1)

#class0=data[data["Classnew"] == 0] #利用

class1\_number=len(class1)#统计class为1数据有多少

class1\_index=np.array(data[data.Classnew == 1].index)

class0\_index=np.array(data[data.Classnew == 0].index)

class0\_indices = np.random.choice(class0\_index, class1\_number, replace = False)#随机选取class1\_number个为零的数值索引

class0\_indices = np.array(class0\_indices)

under\_sample\_indices = np.concatenate([class1\_index,class0\_indices])#下采样的所有数据的Index

under\_sample\_data = data.iloc[under\_sample\_indices,:]#获取的下采样数据

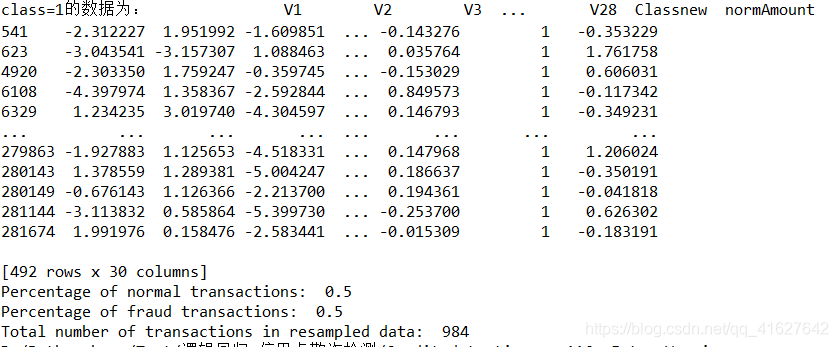
X\_undersample = under\_sample\_data.ix[:, under\_sample\_data.columns != 'Classnew']

y\_undersample = under\_sample\_data.ix[:, under\_sample\_data.columns == 'Classnew']

print("Percentage of normal transactions: ", len(under\_sample\_data[under\_sample\_data.Classnew == 0])/len(under\_sample\_data))

print("Percentage of fraud transactions: ", len(under\_sample\_data[under\_sample\_data.Classnew == 1])/len(under\_sample\_data))

print("Total number of transactions in resampled data: ", len(under\_sample\_data))



## 三、原始数据和下采样数据（训练集和测试集的划分）

#训练集和验证集划分

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size = 0.3, random\_state = 0)

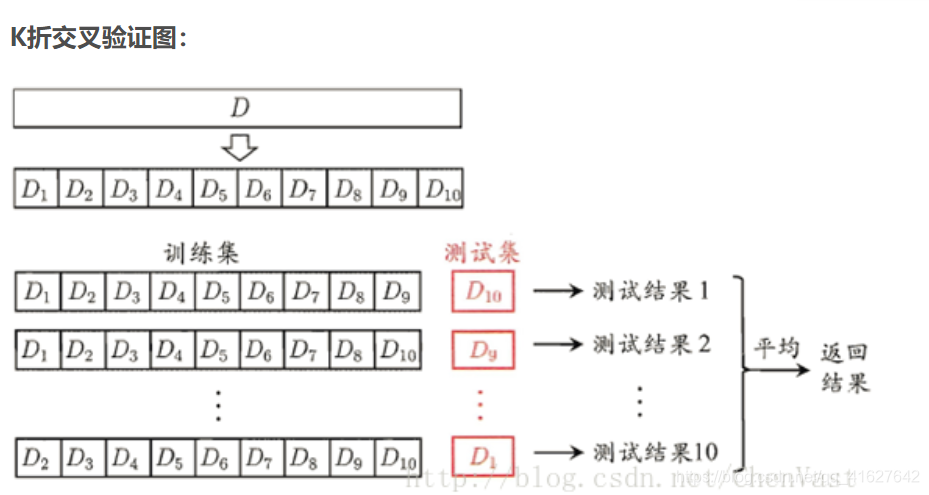
X\_train\_undersample, X\_test\_undersample, y\_train\_undersample, y\_test\_undersample = train\_test\_split(X\_undersample

,y\_undersample

,test\_size = 0.3

,random\_state = 0)

## 四、K折交叉验证（对划分的下采样训练集数据）

假如train分为10分，前9分用于训练，后一份用于测试  


#K折交叉验证

def printing\_Kfold\_scores(x\_train\_data,y\_train\_data):

#k折交叉验证

fold = KFold(n\_splits=5,shuffle=False)

#不同的C参数

c\_param\_range = [0.01,0.1,1,10,100]

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折操作将会给出两个列表：train\_indices = indices[0], test\_indices = indices[1]

j = 0

for c\_param in c\_param\_range:

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

print('C parameter: ', c\_param)

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

print('')

recall\_accs = []

#enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。

for iteration,indices in enumerate(fold.split(x\_train\_data),start=1):

#把c\_param\_range代入到逻辑回归模型中，并使用了l1正则化，C代表正则化系数

#solver优化算法选择参数，{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’}, default: ‘liblinear’

lr = LogisticRegression(C = c\_param,penalty = 'l1',solver='liblinear')

#使用indices[0]的数据进行拟合曲线，使用indices[1]的数据进行误差测试

lr.fit(x\_train\_data.iloc[indices[0],:],y\_train\_data.iloc[indices[0],:].values.ravel())#模型训练

#在indices[1]数据上预测值

y\_pred\_undersample = lr.predict(x\_train\_data.iloc[indices[1],:].values)

#根据不同的c\_parameter计算召回率

recall\_acc = recall\_score(y\_train\_data.iloc[indices[1],:].values,y\_pred\_undersample)

recall\_accs .append(recall\_acc)

print('Iteration ', iteration,': recall score = ', recall\_acc)

#求出我们想要的召回平均值

results\_table.loc[j,'Mean recall score'] = np.mean(recall\_accs)#计算召回率的平均值

j += 1

print('')

print('Mean recall score ', np.mean(recall\_accs))

print('')

print("results\_table is:",results\_table) #输出每个参数c所对应的 K折交叉验证的平均召回率

best\_c = results\_table.loc[results\_table['Mean recall score'].values.argmax()]['C\_parameter']#返回最大平均召回率所对应的参数C

#最后选择最好的 C parameter

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Best model to choose from cross validation is with C parameter = ', best\_c)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

return best\_c

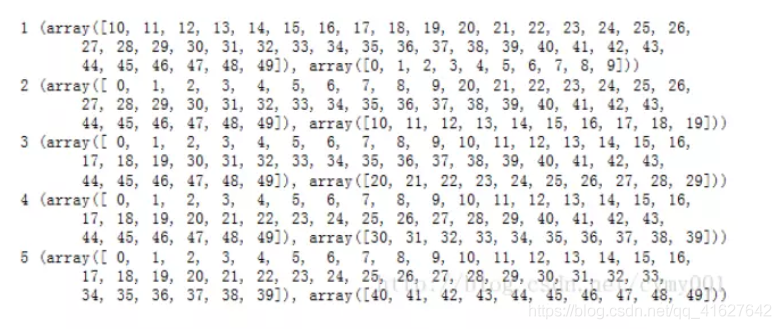
对于这个代码：

from sklearn.cross\_validation import KFold

fold = KFold(50,5,shuffle=False)

for iteration, indices in enumerate(fold,start=1):

print(iteration, indices)



## 五、利用K折交叉验证返回的参数C进行模型的训练和预测

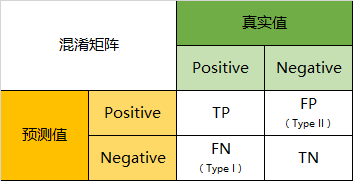
best\_c = printing\_Kfold\_scores(X\_train\_undersample,y\_train\_undersample)

lr = LogisticRegression(C = best\_c, penalty = 'l1',solver='liblinear')#创建模型

lr.fit(X\_train\_undersample,y\_train\_undersample.values.ravel())#模型预训练

y\_pred\_undersample = lr.predict(X\_test\_undersample.values)#模型预测

## 六、混淆矩阵（混淆矩阵的y轴代表真实值，X轴代表预测值）

  
TP代表真积极，T代表真，P代表正类；FP代表假积极，F代表假，P代表判为正类；FN代表假消极，F代表假，N代表负类；TN代表真负类，T代表判断正确，判为父类。召回率（recall)=TP/(TP+FN)

#混淆矩阵绘制

#cm为混淆矩阵，classes为混淆矩阵的类标

def plot\_confusion\_matrix(cm, classes,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

"""

plt.imshow(cm, interpolation='nearest', cmap=cmap)#interpolation插值方式，#cmap表示绘图时的样式，cm是混淆矩阵

plt.title(title)

plt.colorbar()#给图配渐变色时，常常需要在图旁边把colorbar显示出来

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=0)

plt.yticks(tick\_marks, classes)

thresh = cm.max() / 2. #阈值

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, cm[i, j],

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(y\_test\_undersample,y\_pred\_undersample)#计算混淆矩阵

np.set\_printoptions(precision=2)#控制输出的小数点个数是2

print("混淆矩阵为：",cnf\_matrix)

print("Recall metric in the testing dataset: ", cnf\_matrix[1,1]/(cnf\_matrix[1,0]+cnf\_matrix[1,1]))

# Plot non-normalized confusion matrix

class\_names = [0,1]

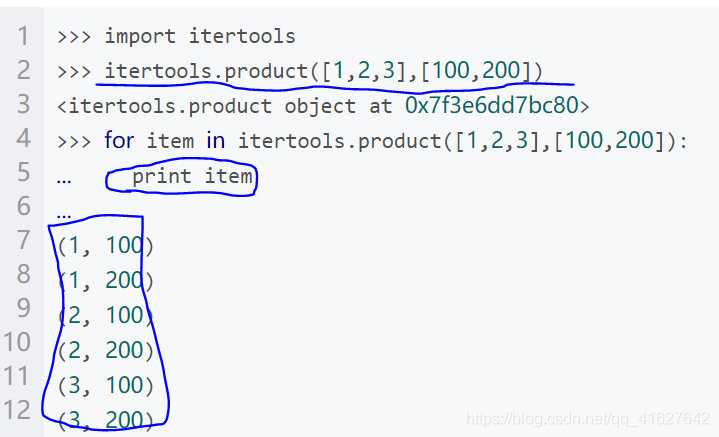
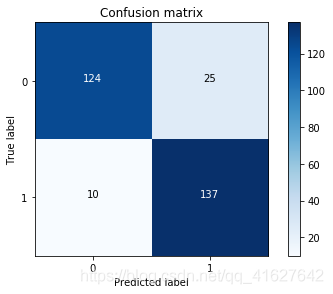
plt.figure()

plot\_confusion\_matrix(cnf\_matrix

, classes=class\_names

, title='Confusion matrix')

plt.show()

## 七、查看不同阈值的混淆矩阵效果

#循环查看不通同阈值效果

lr = LogisticRegression(C = 0.01, penalty = 'l1',solver='liblinear')

lr.fit(X\_train\_undersample,y\_train\_undersample.values.ravel())

y\_pred\_undersample\_proba = lr.predict\_proba(X\_test\_undersample.values)#预测y值的概率

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

for i in thresholds:

y\_test\_predictions\_high\_recall = y\_pred\_undersample\_proba[:,1] > i

plt.subplot(3,3,j)

j += 1

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(y\_test\_undersample,y\_test\_predictions\_high\_recall)

np.set\_printoptions(precision=2)

print("Recall metric in the testing dataset: ", cnf\_matrix[1,1]/(cnf\_matrix[1,0]+cnf\_matrix[1,1]))

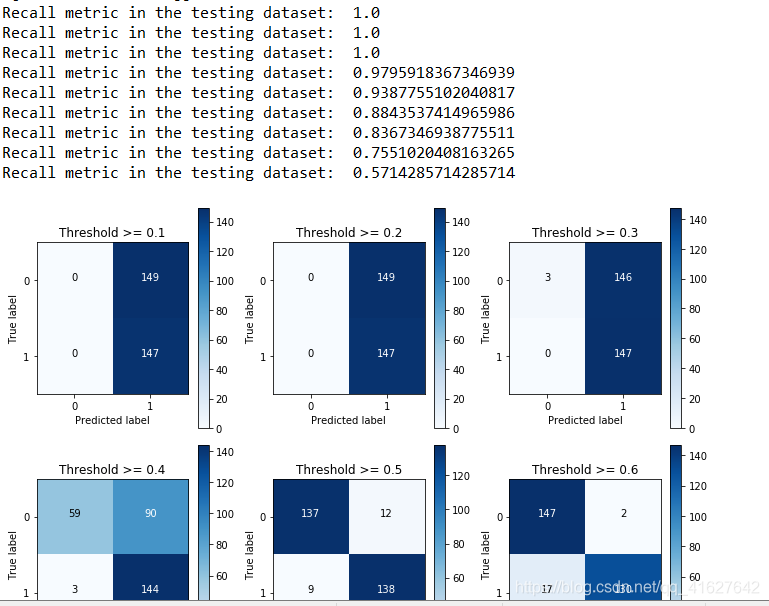
# Plot non-normalized confusion matrix

class\_names = [0,1]

plot\_confusion\_matrix(cnf\_matrix

, classes=class\_names

, title='Threshold >= %s'%i)



## 八、标题

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import os

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression#引入逻辑回归分类器

from sklearn.model\_selection import KFold, cross\_val\_score#引入K折交差验证

from sklearn.metrics import confusion\_matrix,recall\_score,classification\_report #引入混淆矩阵，召回率，以及分类报告

import itertools

from imblearn.over\_sampling import SMOTE

#过采样

features\_train, features\_test, labels\_train, labels\_test = train\_test\_split(X,

y,

test\_size=0.2,

random\_state=0)

oversampler=SMOTE(random\_state=0)

os\_features,os\_labels=oversampler.fit\_sample(features\_train,labels\_train)#对训练特征和标签数据过采样

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

print(len(os\_labels[os\_labels==0])

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

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

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



## 九、整个代码

# -\*- coding: utf-8 -\*-

"""

Created on Fri Feb 28 07:07:23 2020

@author: User

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import os

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression#引入逻辑回归分类器

from sklearn.model\_selection import KFold, cross\_val\_score#引入K折交差验证

from sklearn.metrics import confusion\_matrix,recall\_score,classification\_report #引入混淆矩阵，召回率，以及分类报告

import itertools

from imblearn.over\_sampling import SMOTE

#K折交叉验证

def printing\_Kfold\_scores(x\_train\_data,y\_train\_data):

#k折交叉验证

fold = KFold(n\_splits=5,shuffle=False)

#不同的C参数

c\_param\_range = [0.01,0.1,1,10,100]

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折操作将会给出两个列表：train\_indices = indices[0], test\_indices = indices[1]

j = 0

for c\_param in c\_param\_range:

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

print('C parameter: ', c\_param)

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

print('')

recall\_accs = []

#enumerate() 函数用于将一个可遍历的数据对象(如列表、元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用在 for 循环当中。

for iteration,indices in enumerate(fold.split(x\_train\_data),start=1):

#把c\_param\_range代入到逻辑回归模型中，并使用了l1正则化，C代表正则化系数

#solver优化算法选择参数，{‘newton-cg’, ‘lbfgs’, ‘liblinear’, ‘sag’}, default: ‘liblinear’

lr = LogisticRegression(C = c\_param,penalty = 'l1',solver='liblinear')

#使用indices[0]的数据进行拟合曲线，使用indices[1]的数据进行误差测试

lr.fit(x\_train\_data.iloc[indices[0],:],y\_train\_data.iloc[indices[0],:].values.ravel())#模型训练

#在indices[1]数据上预测值

y\_pred\_undersample = lr.predict(x\_train\_data.iloc[indices[1],:].values)

#根据不同的c\_parameter计算召回率

recall\_acc = recall\_score(y\_train\_data.iloc[indices[1],:].values,y\_pred\_undersample)

recall\_accs .append(recall\_acc)

print('Iteration ', iteration,': recall score = ', recall\_acc)

#求出我们想要的召回平均值

results\_table.loc[j,'Mean recall score'] = np.mean(recall\_accs)#计算召回率的平均值

j += 1

print('')

print('Mean recall score ', np.mean(recall\_accs))

print('')

print("results\_table is:",results\_table) #输出每个参数c所对应的 K折交叉验证的平均召回率

best\_c = results\_table.loc[results\_table['Mean recall score'].values.argmax()]['C\_parameter']#返回最大平均召回率所对应的参数C

#最后选择最好的 C parameter

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

print('Best model to choose from cross validation is with C parameter = ', best\_c)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

return best\_c

#混淆矩阵绘制

#cm为混淆矩阵，classes为混淆矩阵的类标

def plot\_confusion\_matrix(cm, classes,

title='Confusion matrix',

cmap=plt.cm.Blues):

"""

This function prints and plots the confusion matrix.

"""

plt.imshow(cm, interpolation='nearest', cmap=cmap)#interpolation插值方式，#cmap表示绘图时的样式，cm是混淆矩阵

plt.title(title)

plt.colorbar()#给图配渐变色时，常常需要在图旁边把colorbar显示出来

tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=0)

plt.yticks(tick\_marks, classes)

thresh = cm.max() / 2. #阈值

for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):

plt.text(j, i, cm[i, j],

horizontalalignment="center",

color="white" if cm[i, j] > thresh else "black")

plt.tight\_layout()

plt.ylabel('True label')

plt.xlabel('Predicted label')

if \_\_name\_\_=="\_\_main\_\_":

path="D:\Python base\Test\逻辑回归-信用卡欺诈检测"+os.sep +"creditcard.csv"

data= pd.read\_csv(path)

#1.1进行数据展示

class\_number=data.loc[:,'Classnew'].value\_counts()#统计某一列各个数值得出现次数

count\_classes = pd.value\_counts(data['Classnew'], sort = True).sort\_index()#统计某一列各个数值得出现次数

class\_number.plot(kind = 'bar')#kind可以是’line’, ‘bar’, ‘barh’, ‘kde’

plt.title("Fraud class histogram")

plt.xlabel("Class")

plt.ylabel("Frequency")

plt.show()

#对某列数据进行归一化

data['normAmount'] = StandardScaler().fit\_transform(data['Amount'].values.reshape(-1, 1))#对某列数据进行归一化

data = data.drop(['Time','Amount'],axis=1)#删除没用的两列

X=data.loc[:,data.columns != "Classnew"]

y=data.loc[:,data.columns == "Classnew"]

#1.2进行下采样

class1=data[data.Classnew == 1] #利用条件充当Index挑选class=1的数据

print("class=1的数据为：",class1)

#class0=data[data["Classnew"] == 0] #利用

class1\_number=len(class1)#统计class为1数据有多少

class1\_index=np.array(data[data.Classnew == 1].index)

class0\_index=np.array(data[data.Classnew == 0].index)

class0\_indices = np.random.choice(class0\_index, class1\_number, replace = False)#随机选取class1\_number个为零的数值索引

class0\_indices = np.array(class0\_indices)

under\_sample\_indices = np.concatenate([class1\_index,class0\_indices])#下采样的所有数据的Index

under\_sample\_data = data.iloc[under\_sample\_indices,:]#获取的下采样数据

X\_undersample = under\_sample\_data.ix[:, under\_sample\_data.columns != 'Classnew']

y\_undersample = under\_sample\_data.ix[:, under\_sample\_data.columns == 'Classnew']

print("Percentage of normal transactions: ", len(under\_sample\_data[under\_sample\_data.Classnew == 0])/len(under\_sample\_data))

print("Percentage of fraud transactions: ", len(under\_sample\_data[under\_sample\_data.Classnew == 1])/len(under\_sample\_data))

print("Total number of transactions in resampled data: ", len(under\_sample\_data))

#

#训练集和验证集划分

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size = 0.3, random\_state = 0)

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)

features\_train, features\_test, labels\_train, labels\_test = train\_test\_split(X,

y,

test\_size=0.2,

random\_state=0)

oversampler=SMOTE(random\_state=0)

os\_features,os\_labels=oversampler.fit\_sample(features\_train,labels\_train)#对训练特征和标签数据过采样

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

print(len(os\_labels[os\_labels==0])

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

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

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

best\_c = printing\_Kfold\_scores(X\_train\_undersample,y\_train\_undersample)

lr = LogisticRegression(C = best\_c, penalty = 'l1',solver='liblinear')#创建模型

lr.fit(X\_train\_undersample,y\_train\_undersample.values.ravel())#模型预训练

y\_pred\_undersample = lr.predict(X\_test\_undersample.values)#模型预测

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(y\_test\_undersample,y\_pred\_undersample)#计算混淆矩阵

np.set\_printoptions(precision=2)#控制输出的小数点个数是2

print("混淆矩阵为：",cnf\_matrix)

print("Recall metric in the testing dataset: ", cnf\_matrix[1,1]/(cnf\_matrix[1,0]+cnf\_matrix[1,1]))

# Plot non-normalized confusion matrix

class\_names = [0,1]

plt.figure()

plot\_confusion\_matrix(cnf\_matrix

, classes=class\_names

, title='Confusion matrix')

plt.show()

#循环查看不通同阈值效果

lr = LogisticRegression(C = 0.01, penalty = 'l1',solver='liblinear')

lr.fit(X\_train\_undersample,y\_train\_undersample.values.ravel())

y\_pred\_undersample\_proba = lr.predict\_proba(X\_test\_undersample.values)#预测y值的概率

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

for i in thresholds:

y\_test\_predictions\_high\_recall = y\_pred\_undersample\_proba[:,1] > i

plt.subplot(3,3,j)

j += 1

# Compute confusion matrix

cnf\_matrix = confusion\_matrix(y\_test\_undersample,y\_test\_predictions\_high\_recall)

np.set\_printoptions(precision=2)

print("Recall metric in the testing dataset: ", cnf\_matrix[1,1]/(cnf\_matrix[1,0]+cnf\_matrix[1,1]))

# Plot non-normalized confusion matrix

class\_names = [0,1]

plot\_confusion\_matrix(cnf\_matrix

, classes=class\_names

, title='Threshold >= %s'%i)

#