index

March 10, 2022

1 Class Imbalance Problems - Lab

1.1 Introduction

Now that you've gone over some techniques for tuning classification models on imbalanced datasets, it's time to practice those techniques. In this lab, you'll investigate credit card fraud and attempt to tune a model to flag suspicious activity.

1.2 Objectives

You will be able to:

- Use sampling techniques to address a class imbalance problem within a dataset
- Create a visualization of ROC curves and use it to assess a model

1.3 Predicting credit card fraud

The following cell loads all the functions you will be using in this lab. All you need to do is run it:

[1]:
```
import pandas as pd
import numpy as np
import itertools

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import roc_curve, auc
from sklearn.metrics import confusion_matrix, plot_confusion_matrix

from imblearn.over_sampling import SMOTE, ADASYN

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

Use Pandas to load the compressed CSV file, `'creditcard.csv.gz'`.

> Note: You need to pass an additional argument (`compression='gzip'`) to read_csv() in order to load compressed CSV files.

[3]:
```
# Load a compressed csv file
df = pd.read_csv("creditcard.csv.gz", compression='gzip')

# Print the first five rows of data
df.head()
```

[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	\
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	

	V8	V9	…	V21	V22	V23	V24	V25	\
0	0.098698	0.363787	…	-0.018307	0.277838	-0.110474	0.066928	0.128539	
1	0.085102	-0.255425	…	-0.225775	-0.638672	0.101288	-0.339846	0.167170	
2	0.247676	-1.514654	…	0.247998	0.771679	0.909412	-0.689281	-0.327642	
3	0.377436	-1.387024	…	-0.108300	0.005274	-0.190321	-1.175575	0.647376	
4	-0.270533	0.817739	…	-0.009431	0.798278	-0.137458	0.141267	-0.206010	

	V26	V27	V28	Amount	Class
0	-0.189115	0.133558	-0.021053	149.62	0
1	0.125895	-0.008983	0.014724	2.69	0
2	-0.139097	-0.055353	-0.059752	378.66	0
3	-0.221929	0.062723	0.061458	123.50	0
4	0.502292	0.219422	0.215153	69.99	0

[5 rows x 31 columns]

1.4 Preview the class imbalance

Did you notice that the dataset has 31 columns? The first is a time field followed by columns V1 - V28, created by way of manual feature engineering done on the backend that we have little information about. Finally, there's the amount of the purchase and a binary `'Class'` flag. This last column, `'Class'`, is the indication of whether or not the purchase was fraudulent, and it is what you should be attempting to predict.

Take a look at how imbalanced this dataset is:

[5]:
```
# Count the number of fraudulent/infraudulent purchases
df["Class"].value_counts(normalize = True)
```

[5]:
```
0    0.998273
1    0.001727
Name: Class, dtype: float64
```

1.5 Define the predictor and target variables

Define `X` and `y` and perform a standard train-test split. Assign 25% to the test set and `random_state` to 0.

[7]:
```
# Your code here
y = df["Class"]
X = df.drop("Class", axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X,y
                                                    , test_size = 0.25
                                                    ,random_state = 0)
```

Find the class imbalance in the training and test sets:

[11]:
```
# Training set
print("Train Set: ", y_train.value_counts(normalize = True))
print('\n')
# Test set
print("Test Set: ", y_test.value_counts(normalize = True))
```

```
Train Set:  0    0.998258
1    0.001742
Name: Class, dtype: float64


Test Set:  0    0.998315
1    0.001685
Name: Class, dtype: float64
```

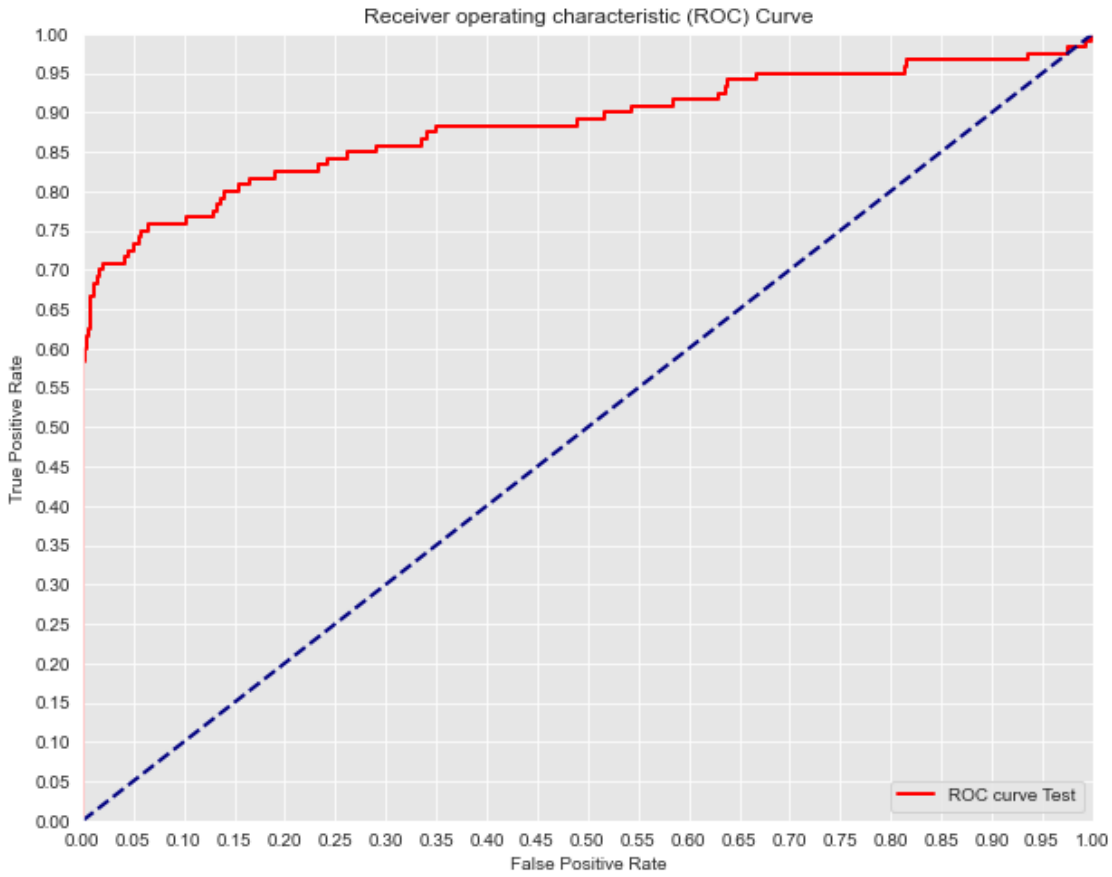
1.6 Create an initial model

As a baseline, train a vanilla logistic regression model. Then plot the ROC curve and print out the AUC. We'll use this as a comparison for how our future models perform.

[37]:
```
import warnings
warnings.simplefilter("ignore", UserWarning)
# Initial Model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver ="liblinear")
logreg.fit(X_train, y_train)

# Probability scores for test set
y_score = logreg.fit(X_train, y_train).decision_function(X_test)
# False positive rate and true positive rate
fpr, tpr, thresholds = roc_curve(y_test, y_score)

# Seaborn's beautiful styling
sns.set_style('darkgrid', {'axes.facecolor': '0.9'})

# Print AUC
```

```
print("AUC IS: ", auc(fpr,tpr))

# Plot the ROC curve
plt.figure(figsize=(10, 8))
plt.plot(fpr, tpr, color="red",
             lw=2, label='ROC curve Test')
plt.plot([0,1], [0,1], color='navy', lw=2, linestyle='--')
plt.xlim([0,1])
plt.ylim([0,1])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
AUC IS:  0.8841632433902629
```



Use scikit-learn's `plot_confusion_matrix` function to plot the confusion matrix of the test set:

[38]:
```
# Plot confusion matrix of the test set
labels=["not fraud", "fraud"]
plot_confusion_matrix(logreg, X_test, y_test,display_labels = labels,
                      cmap=plt.cm.Blues,
                     values_format=".5g");

plt.grid(False)
```

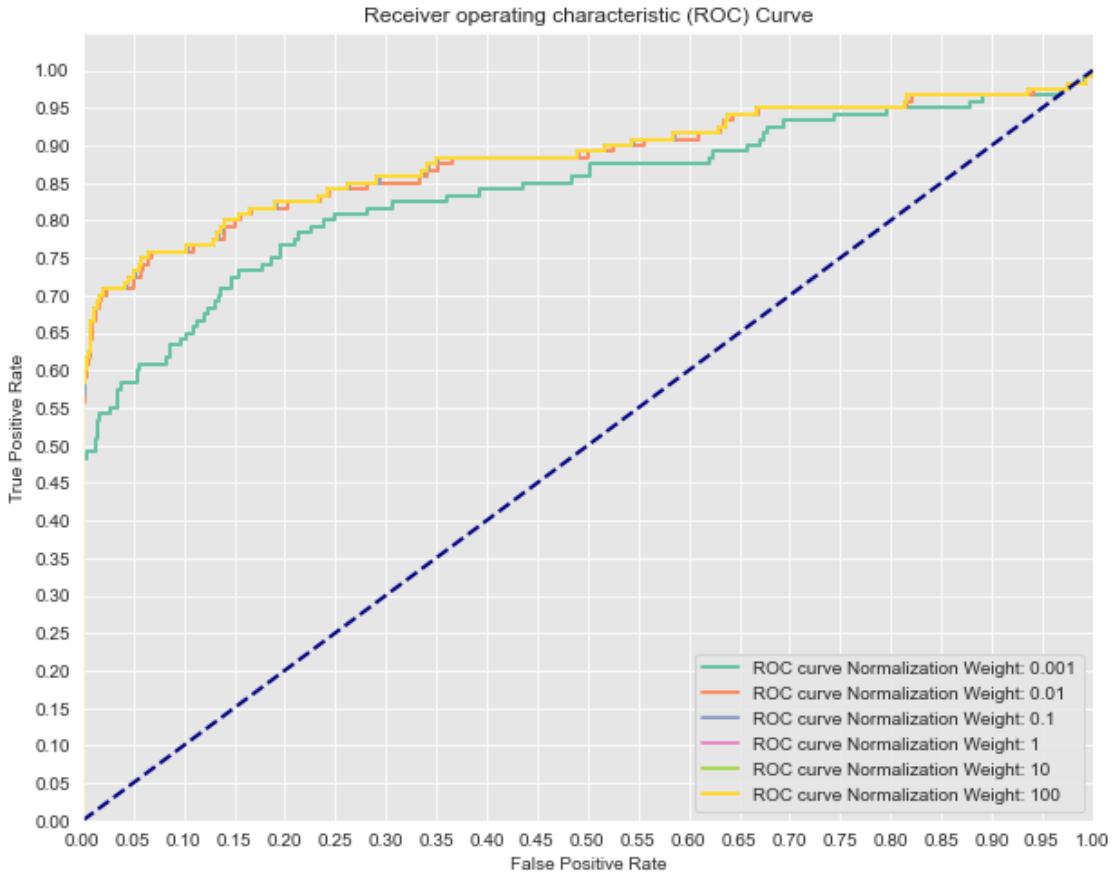
1.7 Tune the model

Try some of the various techniques proposed to tune your model. Compare your models using AUC and ROC curve.

[39]:
```
# Now let's compare a few different regularization performances on the dataset:
C_param_range = [0.001, 0.01, 0.1, 1, 10, 100]
names = [0.001, 0.01, 0.1, 1, 10, 100]
colors = sns.color_palette('Set2')

plt.figure(figsize=(10, 8))

for n, c in enumerate(C_param_range):
    # Fit a model
    logreg = LogisticRegression(fit_intercept=False, C=c, solver ="liblinear")
    model_log = logreg.fit(X_train, y_train)
    print(model_log) # Preview model params
```

```
    # Predict
    y_hat_test = model_log.predict(X_train)

    y_score = logreg.fit(X_train, y_train).decision_function(X_test)

    fpr, tpr, thresholds = roc_curve(y_test, y_score)

    print('AUC for {}: {}'.format(names[n], auc(fpr, tpr)))
    print('------------------------------------------------------------')
    lw = 2
    plt.plot(fpr, tpr, color=colors[n],
             lw=lw, label='ROC curve Normalization Weight: {}'.format(names[n]))

plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])

plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
LogisticRegression(C=0.001, fit_intercept=False, solver='liblinear')
AUC for 0.001: 0.8397641690817178
------------------------------------------------------------
LogisticRegression(C=0.01, fit_intercept=False, solver='liblinear')
AUC for 0.01: 0.8817811354023053
------------------------------------------------------------
LogisticRegression(C=0.1, fit_intercept=False, solver='liblinear')
AUC for 0.1: 0.8839374478302056
------------------------------------------------------------
LogisticRegression(C=1, fit_intercept=False, solver='liblinear')
AUC for 1: 0.8841412031175263
------------------------------------------------------------
LogisticRegression(C=10, fit_intercept=False, solver='liblinear')
AUC for 10: 0.884160781444904
------------------------------------------------------------
LogisticRegression(C=100, fit_intercept=False, solver='liblinear')
AUC for 100: 0.8841627744482896
------------------------------------------------------------
```



1.7.1 SMOTE

Use the `SMOTE` class from the `imblearn` package in order to improve the model's performance on the minority class.

[40]:
```
# Previous original class distribution
print(y_train.value_counts())
smote = SMOTE()
# Fit SMOTE to training data
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Preview synthetic sample class distribution
print('\n')
print(pd.Series(y_train_resampled).value_counts())
```

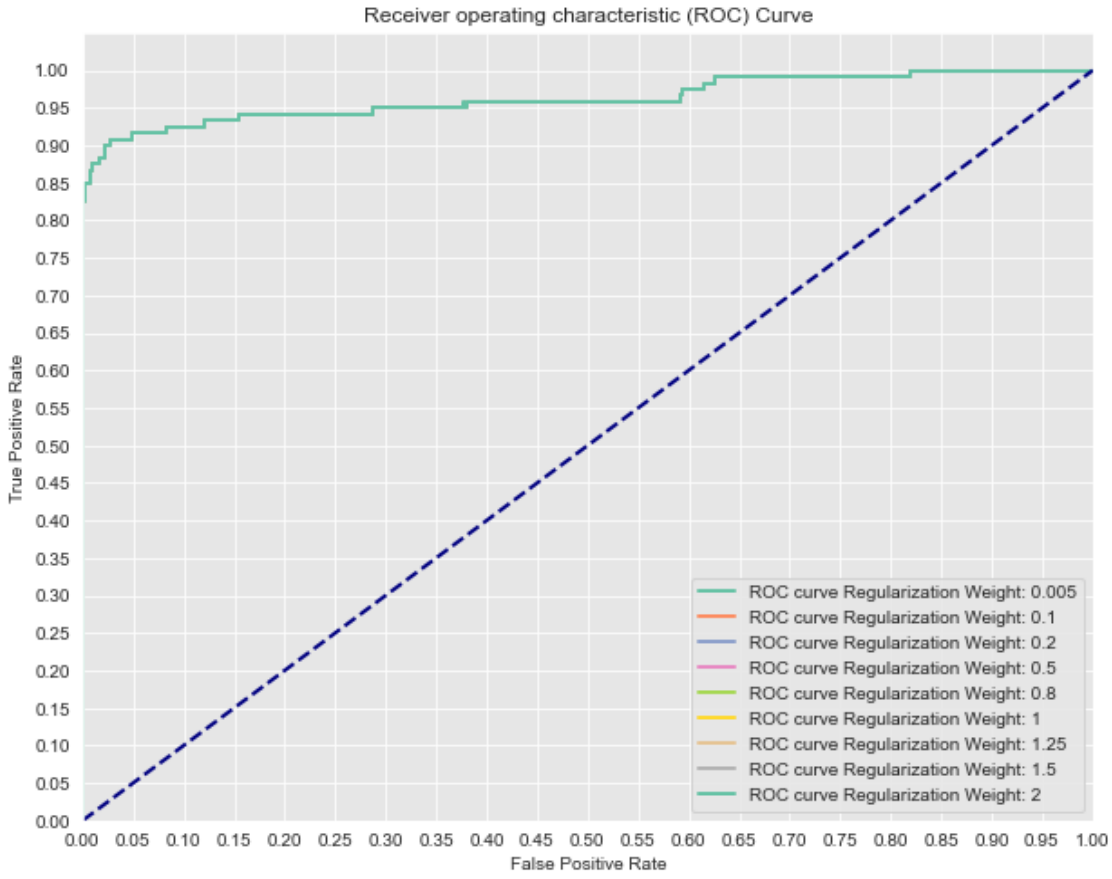
```
0    213233
1       372
Name: Class, dtype: int64
```

```
1    213233
0    213233
Name: Class, dtype: int64
```

Similar to what you did above, build models with this resampled training data:

[44]:
```
# Now let's compare a few different regularization performances on the dataset
C_param_range = [0.005, 0.1, 0.2, 0.5, 0.8, 1, 1.25, 1.5, 2]
names = [0.005, 0.1, 0.2, 0.5, 0.8, 1, 1.25, 1.5, 2]
colors = sns.color_palette('Set2', n_colors=len(names))

plt.figure(figsize=(10, 8))

# Write a for loop that builds models for each value of C_param_range,
# prints the AUC and plots the ROC

for i, c in enumerate(C_param_range):

    logreg=LogisticRegression(C=c,
                              fit_intercept=False,
                              solver = "liblinear")
    logreg.fit(X_train_resampled, y_train_resampled)

    y_score = logreg.fit(X_train_resampled,
                         y_train_resampled).decision_function(X_test)
    fpr, tpr, threshold = roc_curve(y_test, y_score)

    print(logreg)
    print('AUC for {}: {}'.format(names[i], auc(fpr, tpr)))
    print('------------------------------------------------------------')

    lw = 2
    plt.plot(fpr, tpr, color=colors[i],
             lw=lw, label='ROC curve Regularization Weight: {}'.
 ↪format(names[i]))



plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
LogisticRegression(C=0.005, fit_intercept=False, solver='liblinear')
AUC for 0.005: 0.9629383435093741
------------------------------------------------------
LogisticRegression(C=0.1, fit_intercept=False, solver='liblinear')
AUC for 0.1: 0.9629225167177813
------------------------------------------------------
LogisticRegression(C=0.2, fit_intercept=False, solver='liblinear')
AUC for 0.2: 0.9629222822467949
------------------------------------------------------
LogisticRegression(C=0.5, fit_intercept=False, solver='liblinear')
AUC for 0.5: 0.9629216960693283
------------------------------------------------------
LogisticRegression(C=0.8, fit_intercept=False, solver='liblinear')
AUC for 0.8: 0.9629218133048216
------------------------------------------------------
LogisticRegression(C=1, fit_intercept=False, solver='liblinear')
AUC for 1: 0.9629216960693282
------------------------------------------------------
LogisticRegression(C=1.25, fit_intercept=False, solver='liblinear')
AUC for 1.25: 0.9629215788338352
------------------------------------------------------
LogisticRegression(C=1.5, fit_intercept=False, solver='liblinear')
AUC for 1.5: 0.9629215788338352
------------------------------------------------------
LogisticRegression(C=2, fit_intercept=False, solver='liblinear')
AUC for 2: 0.9629214615983419
------------------------------------------------------
```



1.8 Something wrong here?

Describe what is misleading about the AUC score and ROC curves produced by this code:

[ ]:
```
# Previous original class distribution
print(y.value_counts())
X_resampled, y_resampled = SMOTE().fit_sample(X, y)
# Preview synthetic sample class distribution
print('----------------------------------')
print(pd.Series(y_resampled).value_counts())

# Split resampled data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,
                                                    random_state=0)

# Now let's compare a few different regularization performances on the dataset:
C_param_range = [0.005, 0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8]
names = [0.005, 0.1, 0.2, 0.3, 0.5, 0.6, 0.7, 0.8, 0.9]
colors = sns.color_palette('Set2', n_colors=len(names))
```

```
plt.figure(figsize=(10, 8))

for n, c in enumerate(C_param_range):
    # Fit a model
    logreg = LogisticRegression(fit_intercept=False, C=c, solver='liblinear')
    model_log = logreg.fit(X_train, y_train)

    # Predict
    y_hat_test = logreg.predict(X_test)

    y_score = logreg.fit(X_train, y_train).decision_function(X_test)

    fpr, tpr, thresholds = roc_curve(y_test, y_score)
    print('-----------------------------------------------')
    print('AUC for {}: {}'.format(names[n], auc(fpr, tpr)))
    lw = 2
    plt.plot(fpr, tpr, color=colors[n],
             lw=lw, label='ROC curve Normalization Weight: {}'.format(names[n]))
plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.yticks([i/20.0 for i in range(21)])
plt.xticks([i/20.0 for i in range(21)])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

1.9 Your response here

In the code above, we see that we also manipulate `X_test` and `y_test` meaning that we are not predicting the data to check our model.

[ ]:
```
## From GitHub

# This ROC curve is misleading because the test set was also manipulated
# using SMOTE.

# This produces results that will not be comparable to future cases as
# we have synthetically created test cases.

# SMOTE should only be applied to training sets, and then from there,
# an accurate gauge can be made on the model's performance
# by using a raw test sample that has not been oversampled or undersampled.
```

1.10 Summary

In this lab, you got some hands-on practice tuning logistic regression models. In the upcoming labs and lessons, you will continue to dig into the underlying mathematics of logistic regression, taking on a statistical point of view and providing you with a deeper understanding of how the algorithm works. This should give you further insight as to how to tune and apply these models going forward.