Class Imbalance Problems - Lab

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.

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

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:

```
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.

```
# Load a compressed csv file
df = pd.read_csv('creditcard.csv.gz', compression='gzip')
df.head()
```

<style scoped> .dataframe tbody tr th:only-of-type { vertical-align: middle; }

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

</style>

	Time	V1	V2	V3	V4	V5	V6
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.4623
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.8004
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.2472
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.0959

5 rows × 31 columns

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:

```
# Count the number of fraudulent/infraudulent purchases
df['Class'].value_counts()
```

```
0    284315
1       492
Name: Class, dtype: int64
```

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.

```
y = df['Class']
X = df.drop(columns=['Class'], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

Find the class imbalance in the training and test sets:

```
# Training set
print(y_train.value_counts())
print('\n')
# Test set
print(y_test.value_counts())
```

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


0    71082
1      120
Name: Class, dtype: int64
```

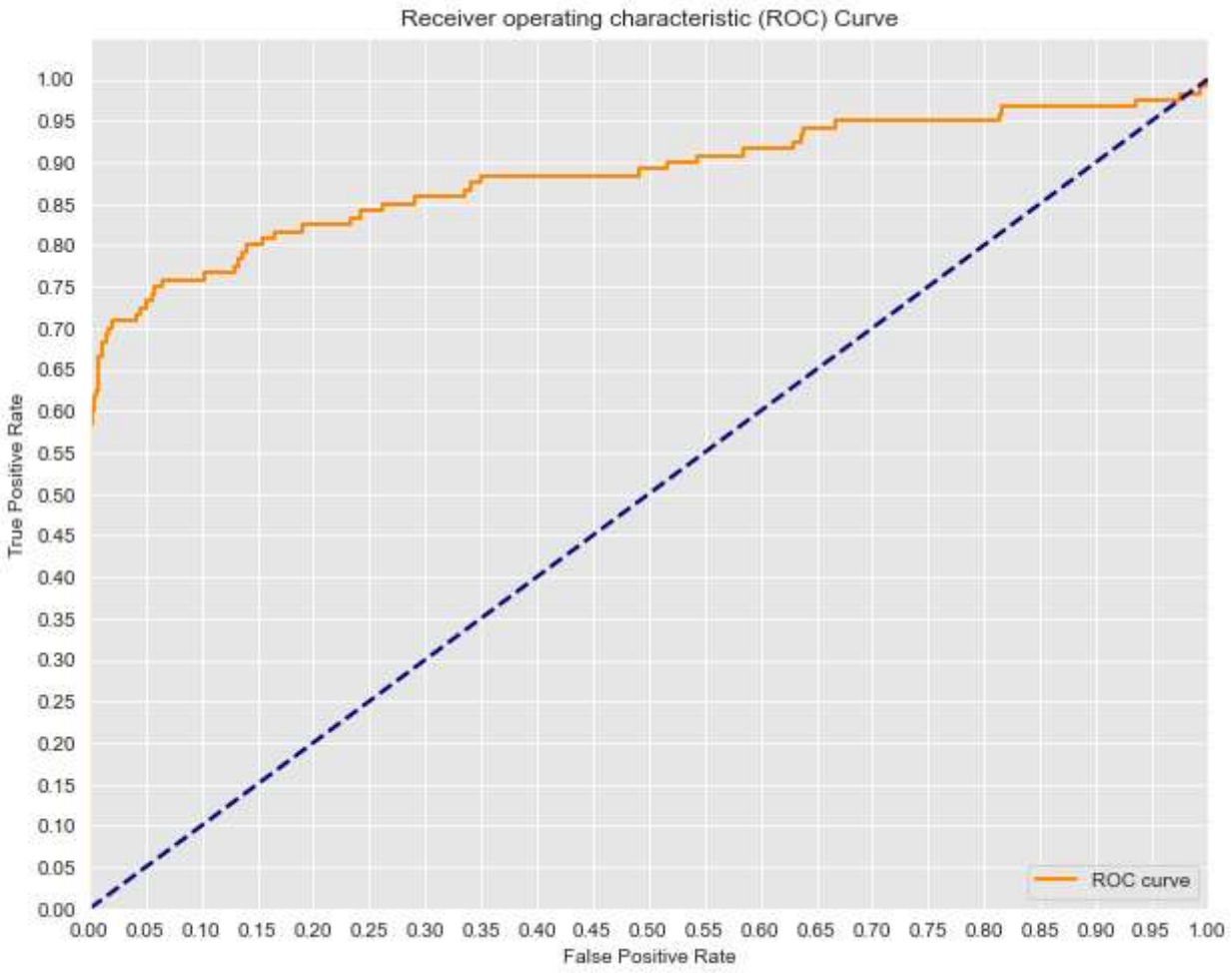
Create an initial model

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.

```
# Initial Model
logreg = LogisticRegression(fit_intercept=False, solver='liblinear')

# 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: {}'.format(auc(fpr, tpr)))

# Plot the ROC curve
plt.figure(figsize=(10, 8))
lw = 2
plt.plot(fpr, tpr, color='darkorange',
         lw=lw, label='ROC curve')
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()
```

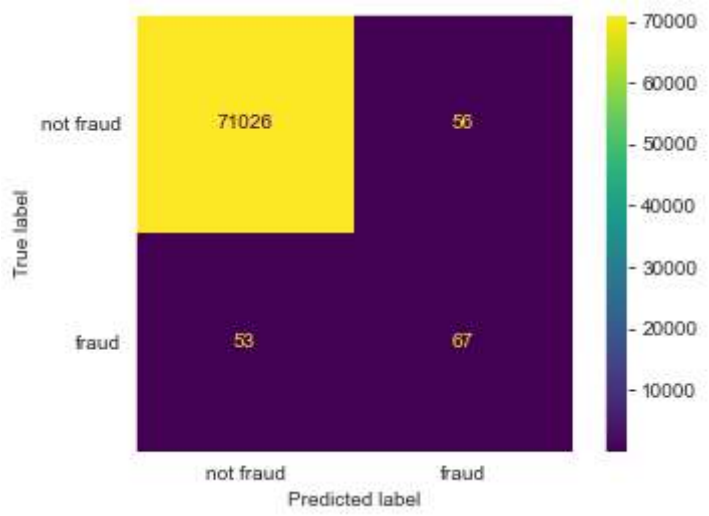
```
AUC: 0.8841412031175263
```



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

```
# Plot confusion matrix of the test set
plot_confusion_matrix(logreg, X_test, y_test,
                      display_labels=["not fraud", "fraud"],
                      values_format=".5g")
```

```
plt.grid(False) # removes the annoying grid lines from plot
plt.show()
```

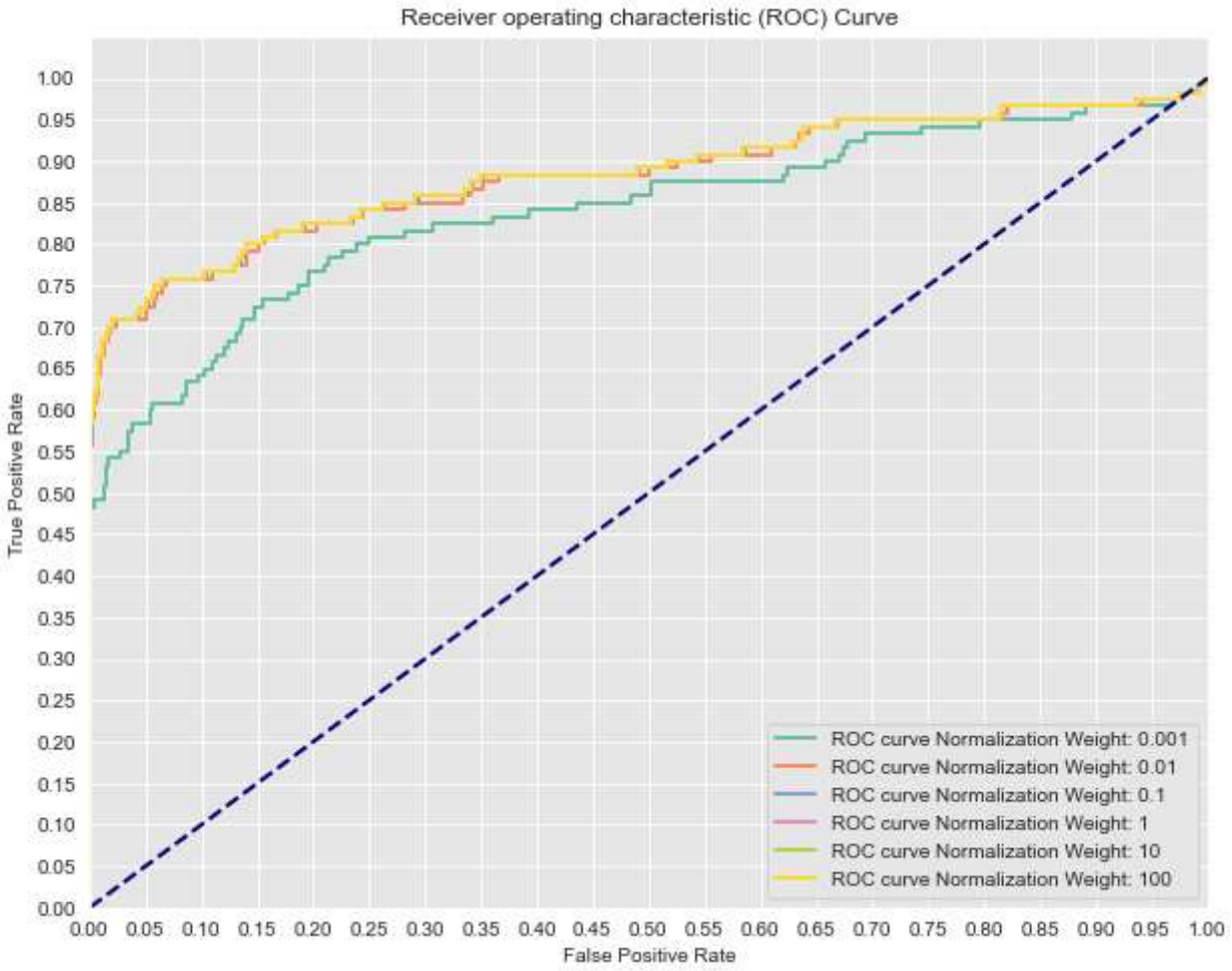


Tune the model

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

```
# 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 = 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('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: 0.8817812526377986
    -------------------------------------------------------
    LogisticRegression(C=0.1, fit_intercept=False, solver='liblinear')
    AUC for 0.1: 0.8839373305947121
    -------------------------------------------------------
    LogisticRegression(C=1, fit_intercept=False, solver='liblinear')
    AUC for 1: 0.8841412031175263
    -------------------------------------------------------
    LogisticRegression(C=10, fit_intercept=False, solver='liblinear')
    AUC for 10: 0.8841610159158905
    -------------------------------------------------------
    LogisticRegression(C=100, fit_intercept=False, solver='liblinear')
    AUC for 100: 0.8841630089192762
    -------------------------------------------------------
```



SMOTE

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

```
# Previous original class distribution
print(y_train.value_counts())

# 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())

# Note, if you get an Attribute Error: 'SMOTE' object has no attribute
# '_validate_data', then downgrade your version of imblearn to 0.6.2
# or upgrade your version of sklearn to 0.23
```

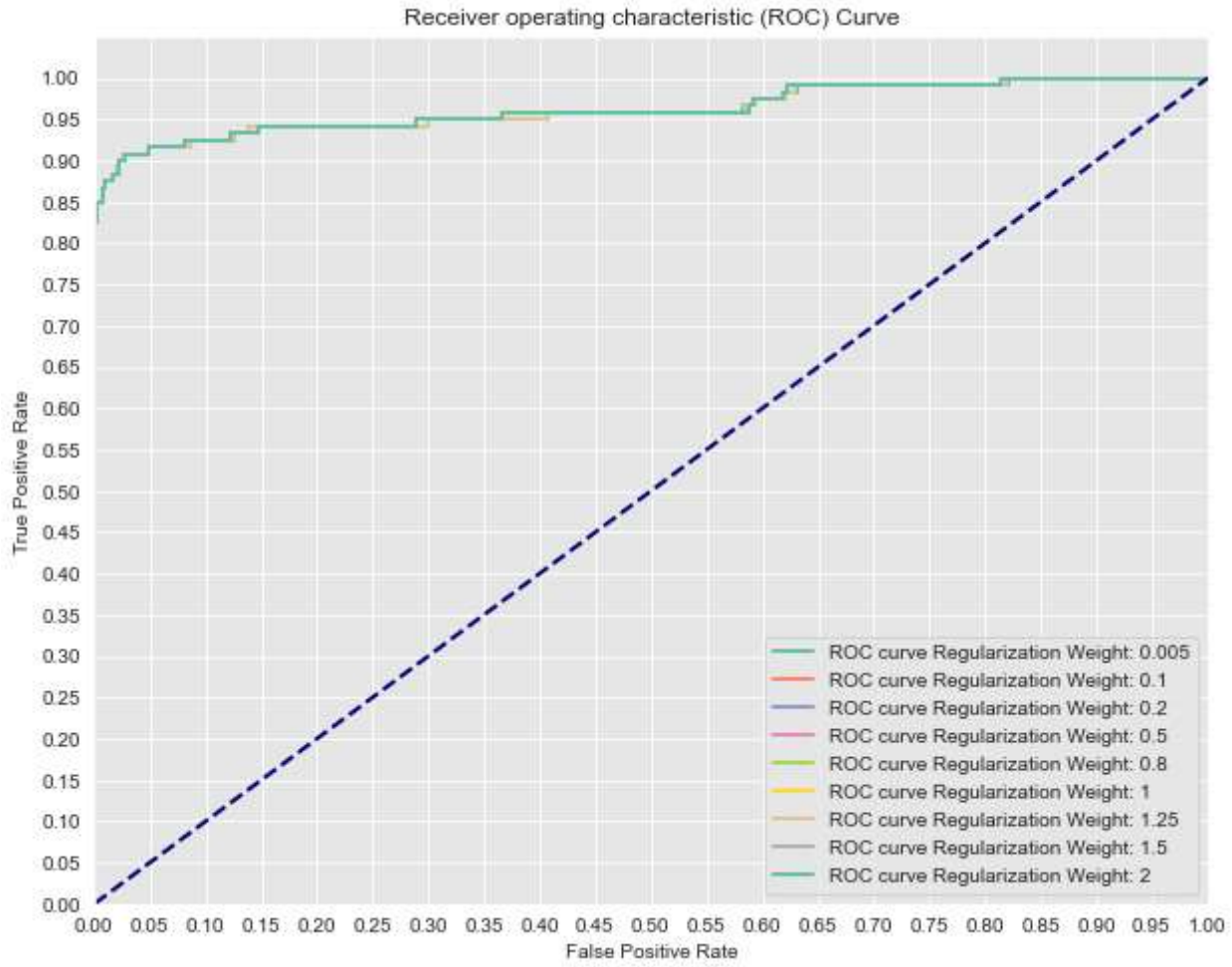
```
0    213233
1       372
Name: Class, dtype: int64


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

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

```
# 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))

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_resampled, y_train_resampled)
    print(model_log) # Preview model params

    # Predict
    y_hat_test = logreg.predict(X_test)

    y_score = logreg.fit(X_train_resampled, y_train_resampled).decision_function(X_t

    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 Regularization 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=0.005, fit_intercept=False, solver='liblinear')
AUC for 0.005: 0.9626324761074065
------------------------------------------------------
LogisticRegression(C=0.1, fit_intercept=False, solver='liblinear')
AUC for 0.1: 0.9626549853221162
------------------------------------------------------
LogisticRegression(C=0.2, fit_intercept=False, solver='liblinear')
AUC for 0.2: 0.9626553370285961
------------------------------------------------------
LogisticRegression(C=0.5, fit_intercept=False, solver='liblinear')
AUC for 0.5: 0.9632037646661602
------------------------------------------------------
LogisticRegression(C=0.8, fit_intercept=False, solver='liblinear')
AUC for 0.8: 0.9632035301951738
------------------------------------------------------
LogisticRegression(C=1, fit_intercept=False, solver='liblinear')
AUC for 1: 0.9632034129596805
------------------------------------------------------
LogisticRegression(C=1.25, fit_intercept=False, solver='liblinear')
AUC for 1.25: 0.9626555714995827
------------------------------------------------------
LogisticRegression(C=1.5, fit_intercept=False, solver='liblinear')
AUC for 1.5: 0.9632034129596804
------------------------------------------------------
LogisticRegression(C=2, fit_intercept=False, solver='liblinear')
AUC for 2: 0.9632034129596804
------------------------------------------------------
```



Something wrong here?

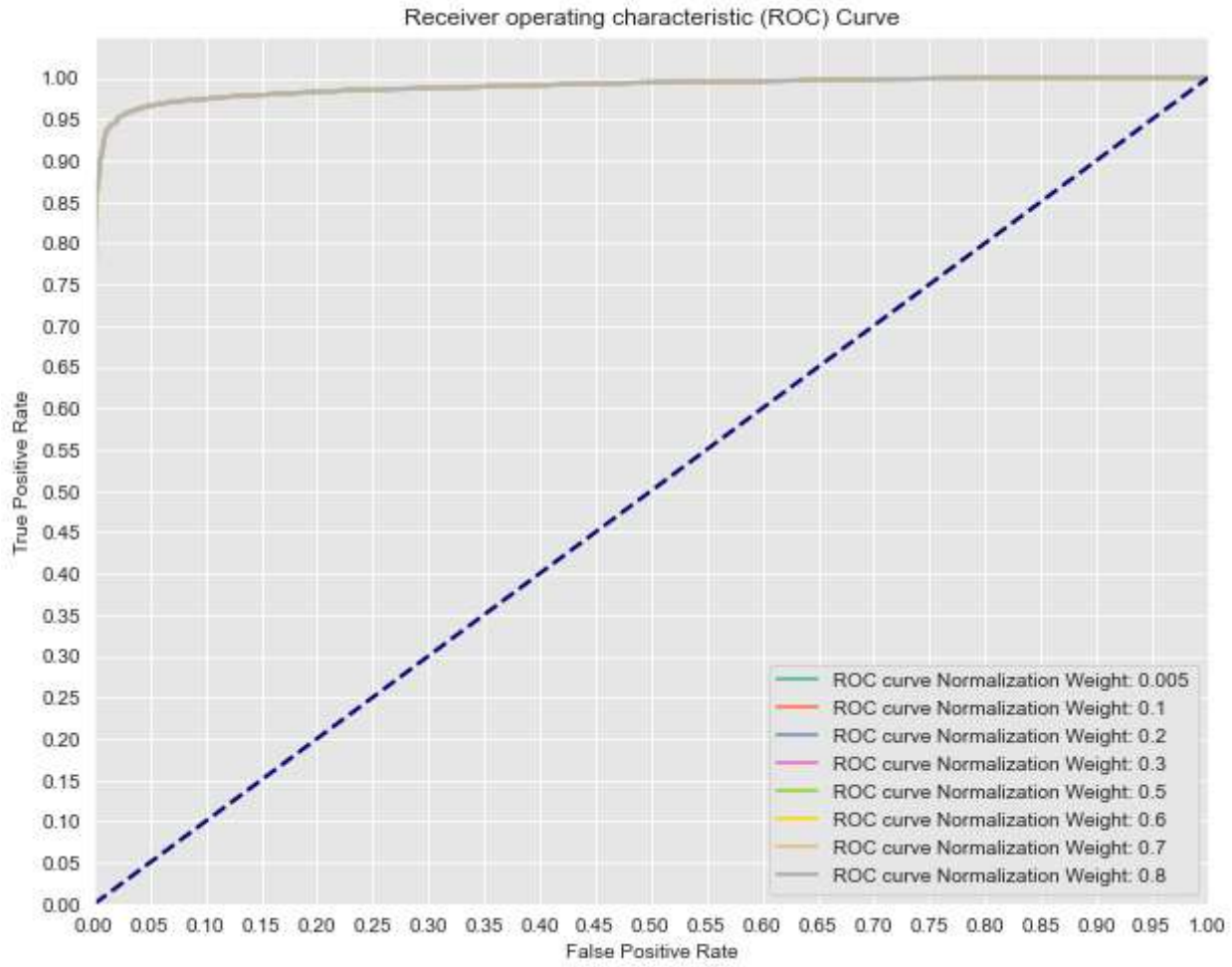
Describe what is misleading about the AUC score and ROC curves produced by this code:

```
# Previous original class distribution
print(y.value_counts())
X_resampled, y_resampled = SMOTE().fit_resample(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

# 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()
```

```
0    284315
1       492
Name: Class, dtype: int64
---------------------------------
1    284315
0    284315
Name: Class, dtype: int64
------------------------------------------------
AUC for 0.005: 0.9896247293367446
------------------------------------------------
AUC for 0.1: 0.9896269412555874
------------------------------------------------
AUC for 0.2: 0.9896269970731367
------------------------------------------------
AUC for 0.3: 0.9896270125120333
------------------------------------------------
AUC for 0.5: 0.9896270309199485
------------------------------------------------
AUC for 0.6: 0.9896270332951634
------------------------------------------------
```

```
AUC for 0.7: 0.9896270455671069
-----------------------------------------------
AUC for 0.8: 0.9896270390352662
```



Your response here

```
# 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 synth
# SMOTE should only be applied to training sets, and then from there, an accurate ga
# by using a raw test sample that has not been oversampled or undersampled.
```

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.

Releases

No releases published

Packages

No packages published

Contributors 8

Languages

- **Jupyter Notebook** 100.0%