Exercise 5

Objectives

- Create a classification model
- Analyze an imbalanced dataset
- Build several classification models that provide a solution from multiple approaches
- Evaluate and compare model results with appropriate "goodness of model" measurements

Instructions

1. You may use either Anaconda Jupyter Notebooks or Google Collab for this exercise.
2. Create a kaggle login.
3. Download the Caravan Insurance Challenge files
4. Investigate the tabs: Data, Overview, Discussion, Insights in the link
5. Run through the code exercises below.
6. Run code as a group to answer the question at the end as well as the questions after each part highlighted. Feel free to answer the question in the text box where the questions are but make sure you demonstrate how you got the answer with the appropriate code.
7. You will also be asked to write your own code when prompted.
8. 2 ways to submit:
    a. Share the Google Collab url link and post it for your submission
    b. Download your Jupyter or Google Collab notebook as a .ipynb file and submit that.

Note: if you are using Google Collab "markdown" blocks are the same as "text" blocks.

Overview

Purpose of Lab 5 is to generate classification models for determining the purchase of a mobile home policy from Caravan Insurance. Various decision tree classification models will be generated and compared including bagging, boosting, and random forest. All models will be evaluated on unbalanced data, undersampled data, oversampled data, and SMOTE (Synthetic Minority

Oversampling TEchnique).

Bagging, boosting, and random forest are all types of ensemble methods, which combine several models to produce one predictive model. They generally aren't as interpretable but are more accurate. They have nuances between them and in general boosting decreases bias, bagging and random forest decrease variance.

Unbalanced: Make and test predictors with skewed binary attribute

Undersampling: Makes binary attribute even by lowering the more dominant class (no Caravan Insurance)

Oversampling: Raises the minority class (Caravan Insurance) via duplicates

SMOTE: Synthetic Minority Oversampling Technique - Raises minority class by creating synthetic non-duplicate samples of the minority class. It selects similar records and alters that record one column at a time by a random amount within the difference to the neighboring records.

Ensembler Performance Measurement AUC - ROC curve is a performance measurement for classification problem at various thresholds settings. ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes.

Load in some packages that we will need

```
#conda install -c conda-forge imbalanced-learn
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import train_test_split
from sklearn.model_selection import ParameterGrid
from imblearn.over_sampling import SMOTE
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from scipy import interp
from sklearn.metrics import roc_curve, auc
```

EDA

1. Do we have any null values?
2. How many rows and columns do we have?
3. What is the target variable?
4. By filtering on the target variable, how many policies do we have and how many non-policies do we have?
5. Is our data unbalanced on the target variable?

Hint code:

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np


Not_Insured_with_caravan = sum(df_main['CARAVAN'] == 0)
Insured_with_caravan = sum(df_main['CARAVAN'] == 1)

plt.bar('Not Insured', Not_Insured_with_caravan, color = 'b', width = 0.25, label='Not Insured')
plt.bar('Insured', Insured_with_caravan, color = 'r', width = 0.25, label='Insured')

#X = np.arange(1)
print("Not Insured: ",Not_Insured_with_caravan)
print("Insured: ", Insured_with_caravan)

plt.title("Number of People with Caravan's Insured vs Not Insured")
plt.xlabel("Policy Status")
plt.ylabel("Number of People")
plt.legend(loc='upper right')
```

Example of Resampling (Oversampling)

Now let's resample our dataset by "oversampling" on the insured. What did this do to our data set?

```
from sklearn.utils import resample

not_insured= df_main[df_main['CARAVAN'] == 0]
insured = df_main[df_main['CARAVAN'] == 1]


insurance_upsampled = resample(insured,
```

```
                        replace=True, # sample with replacement
                        n_samples=len(not_insured), # match number in majority class
                        random_state=27) # reproducible results

# combine majority and upsampled minority
upsampled = pd.concat([not_insured, insurance_upsampled])

upsampled_not_insured = sum(upsampled['CARAVAN'] == 0)
upsampled_insured = sum(upsampled['CARAVAN'] == 1)

plt.bar('Not Insured', upsampled_not_insured, color = 'b', width = 0.25, label='Not Insured
Oversampled')
plt.bar('Insured', upsampled_insured, color = 'r', width = 0.25, label='Insured Oversampled')
plt.title("Number of People with Caravan's Insured vs Not Insured Oversampled")
plt.xlabel("Policy Status")
plt.ylabel("Number of People")
plt.legend(loc='upper center')

#X = np.arange(1)
print("Not Insured: ",upsampled_not_insured)
print("Insured: ", upsampled_insured)
```

Train and Test

1. Split the data into training and test sets (hint: there is a variable in the data set that already done this for you) and call them df_train and df_test
2. Create an a dataset that includes all the independent variables (hint: use iloc to skip the first and last column) and call them df_train_x, df_test_x
3. Create 2 dataframes that ONLY includes the target variable df_train_y and df_test_y

Dealing with Unbalanced Data

Since we have unbalanced data, let's rebalance the target for better classification results

```
# Random Undersampling
rus = RandomUnderSampler(random_state=77)
rus_x_train, rus_y_train = rus.fit_resample(df_train_x, df_train_y)
```

```
# Random Oversampling
#ENTER YOUR OWN CODE

# SMOTE
#ENTER YOUR OWN CODE

#put all the different datasets into a single list so we can iterate over it.
train_sample_labels = ["Unbalanced", "Undersample", "Oversample", "SMOTE"]
train_samples = [(df_train_x, df_train_y), (rus_x_train, rus_y_train), (ros_x_train, ros_y_train),
(sm_x_train, sm_y_train)]
```

Classification Modeling and Conclusions

Business question: Attempt to predict who would be interested in buying a caravan insurance policy by building multiple models and comparing the AUC (Area Under the Curve)

Include these models in your notebook:

- Classification using Bagging - Unbalanced Data, Undersampled Data, Oversampled Data, SMOTE

  Sample code:

```
ensemble = []


i = 0
for sample in train_samples:
    bag = BaggingClassifier(None, 20, random_state = 1)
    bag.fit(sample[0],sample[1])
    y_pred_prob = bag.predict_proba(df_test_x)[:,1]
    fpr, tpr, thresholds = roc_curve(df_test_y, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    ensemble.append((roc_auc, "Bagging", train_sample_labels[i]))
```

```
    label='%s (AUC = %0.2f)' % (train_sample_labels[i], roc_auc)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr, tpr, label=label, color=np.random.rand(3))
    i += 1

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Bagging ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

- Classification using Boosting - Unbalanced Data, Undersampled Data, Oversampled Data, SMOTE

WRITE YOUR OWN CODE

- Classification using Random Forest - Unbalanced Data, Undersampled Data, Oversampled Data, SMOTE

WRITE YOUR OWN CODE

Discuss the performance measures for all models and a comparison.
Discuss a brief explanation of undersampling, oversampling, and SMOTE techniques for imbalanced classes.

Which one did the best?

```
ensemble.sort()
for classifier in ensemble:
    print(classifier)
```

Question

1. **What did you learn from this group exercise?**

Reference: [cross-sellingCaravanInsuranceUsingDataMining]