HW6

Note: The first two pages of HW6 are the same as the first two pages of HW5.

The LendingClub is a peer-to-peer lending company that directly connects borrowers and potential lenders/investors. In this homework, you will build classification models to predict whether or not a loan provided by LendingClub is likely to default.

In this homework you will use data from the LendingClub to predict whether a loan will be paid off in full or the loan will be charged off and possibly go into default.

Load the Lending Club dataset

Load the dataset into a data frame named **loans**.

Exploring features

Let's quickly explore what the dataset looks like. First, print out the column names to see what features we have in this dataset. Here, we should see that we have some feature columns that have to do with grade of the loan, annual income, home ownership status, etc.

Features for the classification algorithm

In this assignment, we will be using a subset of features (categorical and numeric). The features we will be using are described below. Extract these feature columns and target column from the dataset. We will only use these features.

```
features = ['grade',                     # grade of the loan
            'sub_grade',                 # sub-grade of the loan
            'short_emp',                 # one year or less of employment
            'emp_length_num',            # number of years of employment
            'home_ownership',            # home_ownership status: own, mortgage or rent
            'dti',                       # debt to income ratio
            'purpose',                   # the purpose of the loan
            'term',                      # the term of the loan
            'last_delinq_none',          # has borrower had a delinquincy
            'last_major_derog_none',     # has borrower had 90 day or worse rating
            'revol_util',                # percent of available credit being used
            'total_rec_late_fee',        # total late fees received to day
           ]
```

You may skip observations with missing values in these features.

Exploring the target column

The target column (label column) of the dataset that we are interested in is called bad_loans. In this column **1** means a risky (bad) loan **0** means a safe loan.

Now, let us explore the distribution of the target column. This gives us a sense of how many safe and risky loans are present in the dataset. Print out the percentage of safe loans and risky loans in the data frame.

It looks like most of these loans are safe loans (thankfully). But this does make our problem of identifying risky loans challenging.

Sample data to balance classes

Our data is disproportionally full of safe loans. Let's create two datasets: one with just the safe loans (safe_loans_raw) and one with just the risky loans (risky_loans_raw).

One way to combat class imbalance is to undersample the larger class until the class distribution is approximately half and half. Here, we will undersample the larger class (safe loans) in order to balance out our dataset. This means we are throwing away many data points. We will use seed=1 so everyone gets the same results. Use the following codes for this task.

```
# Since there are fewer risky loans than safe loans, find the ratio of the sizes
# and use that percentage to undersample the safe loans.
percentage = len(risky_loans_raw)/float(len(safe_loans_raw))

risky_loans = risky_loans_raw
safe_loans = safe_loans_raw.sample(percentage, random_state = 1)

# Append the risky_loans with the downsampled version of safe_loans
loans_data = risky_loans.append(safe_loans)
```

You can verify now that **loans_data** is comprised of approximately 50% safe loans and 50% risky loans.

One-hot encoding

For scikit-learn's implementation, it requires numerical values for it's data matrix. This means you will have to turn categorical variables into binary features via one-hot encoding.

Split data into training and validation and test sets

First, **s**plit the original data into train-validate set (90%) and test set (10%) using random state = 0.
Then, split the train-validate set into training set (80%) and validation set (20%) using random state = 0.

Build a MLP classifier

Use 5-fold GridSearchCV on the train-validate set. Build a MLP classifier and based on accuracy compare hyperparameters in the following grid:
hidden_layer_sizes: [(10,), (50,), (100,), (10, 10), (50, 50)]

Report the best model (call it MODEL1)'s accuracy, precision, recall, F1-score, balanced accuracy, ROC-AUC on the test set. Plot the precision-recall curve and the ROC curve.

Will you choose a different model (call it MODEL2) if the GridSearchCV comparison was based on ROC-AUC? Why?

Quantifying the cost of mistakes

Every mistake the model makes costs money. In this section, we will try and quantify the cost each mistake made by the model. Assume the following:
False negatives: Loans that were actually safe but were predicted to be risky. This results in an opportunity cost of losing a loan that would have otherwise been accepted.
False positives: Loans that were actually risky but were predicted to be safe. These are much more expensive because it results in a risky loan being given.
Correct predictions: All correct predictions don't typically incur any cost.

Let's write code that can compute the cost of mistakes made by the model. Complete the following 4 steps:
First, let us compute the predictions made by the model.
Second, compute the number of false positives.
Third, compute the number of false negatives.
Finally, compute the cost of mistakes made by the model by adding up the costs of true positives and false positives.

Let's assume that each mistake costs us money: a false negative costs $10,000, while a false positive costs $20,000. What is the total cost of mistakes made by MODEL1 and MODEL2 on the test data?

Optional Question

Can you find another MLP classifier that generates a lower total cost of mistakes?