Exploring Ensemble Methods

In this homework we will explore the use of boosting. For this assignment, we will use the pre-implemented gradient boosted trees in Graphlab-Create. You will:

- Use SFrames to do some feature engineering.
- Train a boosted ensemble of decision-trees (gradient boosted trees) on the lending club dataset.
- Predict whether a loan will default along with prediction probabilities (on a validation set).
- Evaluate the trained model and compare it with a baseline.
- Find the most positive and negative loans using the learned model.
- Explore how the number of trees influences classification performance.

If you are doing the assignment with IPython Notebook

An IPython Notebook has been provided below to you for this assignment. This notebook contains the instructions, quiz questions and partially-completed code for you to use as well as some cells to test your code.

Make sure that you are using GraphLab Create 1.8.3. See this post for installing the correct version of GraphLab Create.

What you need to download

If you are using GraphLab Create:

- Download the Lending club data In SFrame format: lending-club-data.gl.zip
- Download the companion IPython Notebook: module-8-boosting-assignment-1-blank.ipynb
- Save both of these files in the same directory (where you are calling IPython notebook from) and unzip the data file.
- Follow the instructions contained in the IPython notebook.

If you are not using GraphLab Create

- If you are using SFrame, download the LendingClub dataset in SFrame format: lending-club-data.gl.zip
- If you are using a different package, download the LendingClub dataset in CSV format: lending-club-data.csv.zip

If you are using GraphLab Create and the companion IPython Notebook

Open the companion IPython notebook and follow the instructions in the notebook.

If you are using other tools

This section is designed for people using tools other than GraphLab Create. Even though some instructions are specific to scikit-learn, most part of the assignment should be applicable to other tools as well. However, **we highly suggest you use SFrame since it is open source.** In this part of the assignment, we describe general instructions, however we will tailor the instructions for SFrame and scikit-learn.

- If you choose to use SFrame and scikit-learn, you should be able to follow the instructions here and complete the assessment. **All code samples given here will be applicable to SFrame and scikit-learn**.
- You are free to experiment with any tool of your choice, but **some many not produce correct numbers for the quiz questions.**

Load the Lending Club dataset

We will be using a dataset from the LendingClub.

1. Load the dataset into a data frame named **loans**. Using SFrame, this would look like

```
import sframe
loans = sframe.SFrame('lending-club-data.gl/')
```

Note: To install SFrame (without installing GraphLab Create), run

```
pip install sframe
```

Exploring some features

2. Let's quickly explore what the dataset looks like. First, print out the column names to see what features we have in this dataset. On SFrame, you can run this code:

```
loans.column_names()
```

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.

Modifying 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.

In order to make this more intuitive and consistent with the lectures, we reassign the target to be:

- **+1** as a safe loan
- **-1** as a risky (bad) loan

3. We put this in a new column called **safe_loans**.

```
# safe_loans =  1 => safe
# safe_loans = -1 => risky
loans['safe_loans'] = loans['bad_loans'].apply(lambda x : +1 if x==0 else -1)
loans = loans.remove_column('bad_loans')
```

Selecting features

In this assignment, we will be using a subset of features (categorical and numeric). The features we will be using are **described in the code comments** below. If you are a finance geek, the LendingClub website has a lot more details about these features.

4. The features we will be using are described in the code comments below. Extract these feature columns and target column from the dataset. We will only use these features.

```
target = 'safe_loans'
features = ['grade',                     # grade of the loan (categorical)
            'sub_grade_num',             # sub-grade of the loan as a number from 0 to 1
            '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
            'payment_inc_ratio',         # ratio of the monthly payment to income
            'delinq_2yrs',               # number of delinquincies
             'delinq_2yrs_zero',          # no delinquincies in last 2 years
            'inq_last_6mths',            # number of creditor inquiries in last 6 months
            'last_delinq_none',          # has borrower had a delinquincy
            'last_major_derog_none',     # has borrower had 90 day or worse rating
            'open_acc',                  # number of open credit accounts
            'pub_rec',                   # number of derogatory public records
            'pub_rec_zero',              # no derogatory public records
            'revol_util',                # percent of available credit being used
            'total_rec_late_fee',        # total late fees received to day
            'int_rate',                  # interest rate of the loan
            'total_rec_int',             # interest received to date
            'annual_inc',                # annual income of borrower
            'funded_amnt',               # amount committed to the loan
            'funded_amnt_inv',           # amount committed by investors for the loan
            'installment',               # monthly payment owed by the borrower
           ]
```

Skipping observations with missing values

Recall from the lectures that one common approach to coping with missing values is to **skip** observations that contain missing values.

5. Using SFrame, we run the following code to do so:

```
loans, loans_with_na = loans[[target] + features].dropna_split()

# Count the number of rows with missing data
num_rows_with_na = loans_with_na.num_rows()
num_rows = loans.num_rows()
print 'Dropping %s observations; keeping %s ' % (num_rows_with_na, num_rows)
```

In Pandas, we'd run

```
loans = loans[[target] + features].dropna()
```

Your tool may provide a function to skip observations with missing values. Consult appropriate manuals.

Fortunately, as you should find, there are not too many missing values. We are retaining most of the data.

Notes to people using other tools

If you are using SFrame, proceed to the section "Make sure the classes are balanced".

If you are NOT using SFrame, download the list of indices for the training and validation sets: module-8-assignment-1-train-idx.json, module-8-assignment-1-validation-idx.json. Then follow the following steps:

- Apply one-hot encoding to **loans**. Your tool may have a function for one-hot encoding. Alternatively, see #7 for implementation hints.
- Load the JSON files into the lists **train_idx** and **validation_idx**.
- Perform train/validation split using **train_idx** and **validation_idx**. In Pandas, for instance:

```
train_data = loans.iloc[train_idx]
validation_data = loans.iloc[validation_idx]
```

IMPORTANT: If you are using a programming language with 1-based indexing (e.g. R, Matlab), make sure to increment all indices by 1.

Note. Some elements in loans are included neither in **train_data** nor **validation_data**. This is to perform sampling to achieve class balance.

Now proceed to the section "Gradient boosted tree classifier", skipping three sections below.

Make sure the classes are balanced

6. We saw in an earlier assignment that this dataset is also imbalanced. We will undersample the larger class (safe loans) in order to balance out our dataset. We used seed=1 to make sure everyone gets the same results.

```
safe_loans_raw = loans[loans[target] == 1]
risky_loans_raw = loans[loans[target] == -1]

# Undersample the safe loans.
percentage = len(risky_loans_raw)/float(len(safe_loans_raw))
safe_loans = safe_loans_raw.sample(percentage, seed = 1)
risky_loans = risky_loans_raw
loans_data = risky_loans.append(safe_loans)

print "Percentage of safe loans                 :", len(safe_loans) / float(len(loans_data))
print "Percentage of risky loans                :", len(risky_loans) / float(len(loans_data))
print "Total number of loans in our new dataset :", len(loans_data)
```

Note: There are many approaches for dealing with imbalanced data, including some where we modify the learning algorithm. These approaches are beyond the scope of this course, but some of them are reviewed in this paper. For this assignment, we use the simplest possible approach, where we subsample the overly represented class to get a more balanced dataset. In general, and especially when the data is highly imbalanced, we recommend using more advanced methods.

One-hot encoding¶

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

7. We've seen this same piece of code in earlier assignments. Again, feel free to use this piece of code as is. Refer to the API documentation for a deeper understanding.

```
loans_data = risky_loans.append(safe_loans)

categorical_variables = []
for feat_name, feat_type in zip(loans_data.column_names(), loans_data.column_types()):
    if feat_type == str:
        categorical_variables.append(feat_name)

for feature in categorical_variables:
    loans_data_one_hot_encoded = loans_data[feature].apply(lambda x: {x: 1})
    loans_data_unpacked = loans_data_one_hot_encoded.unpack(column_name_prefix=feature)

    # Change None's to 0's
    for column in loans_data_unpacked.column_names():
        loans_data_unpacked[column] = loans_data_unpacked[column].fillna(0)

    loans_data.remove_column(feature)
    loans_data.add_columns(loans_data_unpacked)

loans_data.column_names()
```

Note that the column names are slightly different now, since we used one-hot encoding.

Split data into training and validation

8. We split the data into training data and validation data. We used seed=1 to make sure everyone gets the same results. We will use the validation data to help us select model parameters.

```
train_data, validation_data = loans_data.random_split(.8, seed=1)
```

Call the training and validation sets **train_data** and **validation_data**, respectively.

Gradient boosted tree classifier

Gradient boosted trees are a powerful variant of boosting methods; they have been used to win many Kaggle competitions, and have been widely used in industry. We will explore the predictive power of multiple decision trees as opposed to a single decision tree.

Additional reading: If you are interested in gradient boosted trees, here is some additional reading material:

- GraphLab Create user guide
- Advanced material on boosted trees

We will now train models to predict safe_loans using the features above. In this section, we will experiment with training an ensemble of 5 trees.

9. Now, let's use the built-in scikit learn gradient boosting classifier (sklearn.ensemble.GradientBoostingClassifier) to create a gradient boosted classifier on the training data. You will need to import **sklearn**, **sklearn.ensemble**, and **numpy**.

You will have to first convert the SFrame into a numpy data matrix. See the API for more information. You will also have to extract the label column. **Make sure to set max_depth=6 and n_estimators=5.**

Making predictions

Just like we did in previous sections, let us consider a few positive and negative examples **from the validation set**. We will do the following:

- Predict whether or not a loan is likely to default.
- Predict the probability with which the loan is likely to default.

10. First, let's grab 2 positive examples and 2 negative examples. In SFrame, that would be:

```
validation_safe_loans = validation_data[validation_data[target] == 1]
validation_risky_loans = validation_data[validation_data[target] == -1]

sample_validation_data_risky = validation_risky_loans[0:2]
sample_validation_data_safe = validation_safe_loans[0:2]

sample_validation_data = sample_validation_data_safe.append(sample_validation_data_risky)
sample_validation_data
```

11. For each row in the **sample_validation_data**, write code to make **model_5** predict whether or not the loan is classified as a **safe loan**. (Hint: if you are using scikit-learn, you can use the **.predict()** method)

Quiz question: What percentage of the predictions on sample_validation_data did model_5 get correct?

Prediction Probabilities

12. For each row in the **sample_validation_data**, what is the probability (according **model_5**) of a loan being classified as **safe**? (Hint: if you are using scikit-learn, you can use the **.predict_proba()** method)

Quiz Question: Which loan has the highest probability of being classified as a **safe loan**?

Checkpoint: Can you verify that for all the predictions with probability >= 0.5, the model predicted the label **+1**?

Evaluating the model on the validation data

Recall that the accuracy is defined as follows:

$$\text{accuracy} = \frac{\#\text{ correctly classified examples}}{\#\text{ total examples}}$$

13. Evaluate the accuracy of the **model_5** on the **validation_data**. (Hint: if you are using scikit-learn, you can use the **.score()** method)

14. Calculate the number of **false positives** made by the model on the **validation_data**.

Quiz question: What is the number of **false positives** on the **validation_data**?

15. Calculate the number of **false negatives** made by the model on the **validation_data**.

Comparison with decision trees

In the earlier assignment, we saw that the prediction accuracy of the decision trees was around **0.64**. In this assignment, we saw that **model_5** has an accuracy of approximately **0.67**.

Here, we quantify the benefit of the extra 3% increase in accuracy of **model_5** in comparison with a single decision tree from the original decision tree assignment.

As we explored in the earlier assignment, we calculated the cost of the mistakes made by the model. We again consider the same costs as follows:

- **False negatives**: Assume a cost of $10,000 per false negative.
- **False positives**: Assume a cost of $20,000 per false positive.

Assume that the number of false positives and false negatives for the learned decision tree was

- **False negatives**: 1936
- **False positives**: 1503

Using the costs defined above and the number of false positives and false negatives for the decision tree, we can calculate the total cost of the mistakes made by the decision tree model as follows:

```
cost = $10,000 * 1936  + $20,000 * 1503 = $49,420,000
```

The total cost of the mistakes of the model is $49.42M. That is a **lot of money**!.

16. Calculate the cost of mistakes made by **model_5** on the **validation_data**.

Quiz Question: Using the same costs of the false positives and false negatives, what is the cost of the mistakes made by the boosted tree model (**model_5**) as evaluated on the **validation_set**?

Reminder: Compare the cost of the mistakes made by the boosted trees model with the decision tree model. The extra 3% improvement in prediction accuracy can translate to several million dollars! And, it was so easy to get by simply boosting our decision trees.

Most positive & negative loans.

In this section, we will find the loans that are most likely to be predicted **safe**. We can do this in a few steps:

- **Step 1**: Use the **model_5** (the model with 5 trees) and make **probability predictions** for all the loans in **validation_data**.
- **Step 2**: Similar to what we did in the very first assignment, add the probability predictions as a column called **predictions** into **validation_data**.
- **Step 3**: Sort the data (in descreasing order) by the probability predictions.

17. Start here with **Step 1** & **Step 2**. Make predictions using **model_5** for all examples in the **validation_data**.

Checkpoint: For each row, the probabilities should be a number in the range **[0, 1]**.

18. Now, we are ready to go to **Step 3**. You can now use the prediction column to sort the loans in **validation_data** (in descending order) by prediction probability. Find the top 5 loans with the highest probability of being predicted as a **safe loan**.

Quiz question: What grades are the top 5 loans?

19. Repeat this exercise to find the 5 loans (in the **validation_data**) with the **lowest probability** of being predicted as a **safe loan.**

Effects of adding more trees

In this assignment, we will train 5 different ensemble classifiers in the form of gradient boosted trees.

20. Train models with 10, 50, 100, 200, and 500 trees. Use the **n_estimators** parameter to control the number of trees. Remember to keep **max_depth = 6**.

Call these models **model_10, model_50, model_100, model_200,** and **model_500**, respectively. This may take a few minutes to run.

Compare accuracy on entire validation set

Now we will compare the predicitve accuracy of our models on the validation set.

21. Evaluate the **accuracy** of the 10, 50, 100, 200, and 500 tree models on the **validation_data**.

Quiz Question: Which model has the **best** accuracy on the **validation_data**?

Quiz Question: Is it always true that the model with the most trees will perform best on test data?

Plot the training and validation error vs. number of trees

Recall from the lecture that the classification error is defined as

$$\text{classification error} = 1 - \text{accuracy}$$

In this section, we will plot the training and validation errors versus the number of trees to get a sense of how these models are performing. We will compare the 10, 50, 100, 200, and 500 tree models. **You will need matplotlib in order to visualize the plots.**

22. First, make sure this block of code runs on your computer.

```
import matplotlib.pyplot as plt
%matplotlib inline
def make_figure(dim, title, xlabel, ylabel, legend):
    plt.rcParams['figure.figsize'] = dim
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    if legend is not None:
        plt.legend(loc=legend, prop={'size':15})
    plt.rcParams.update({'font.size': 16})
    plt.tight_layout()
```

In order to plot the classification errors (on the **train_data** and **validation_data**) versus the number of trees, we will need lists of all the errors.

Steps to follow:

- **Step 1:** Calculate the classification error for each model on the training data (**train_data**).
- **Step 2:** Store the training errors into a list (called **training_errors**) that looks like this: [train_err_10, train_err_50, ..., train_err_500]
- **Step 3:** Calculate the classification error of each model on the validation data (**validation_data**).
- **Step 4:** Store the validation classification error into a list (called **validation_errors**) that looks like this: [validation_err_10, validation_err_50, ..., validation_err_500]

Once that has been completed, we will give code that should be able to evaluate correctly and generate the plot.

23. Let us start with **Step 1**. Write code to compute the classification error on the **train_data** for models **model_10**, **model_50**, **model_100**, **model_200**, and **model_500**.

24. Now, let us run **Step 2**. Save the training errors into a list called **training_errors.**

```
training_errors = [train_err_10, train_err_50, train_err_100, train_err_200, train_err_500]
```

25. Now, onto **Step 3**. Write code to compute the classification error on the **validation_data** for models **model_10**, **model_50**, **model_100**, **model_200**, and **model_500**.

26. Now, let us run **Step 4**. Save the training errors into a list called **validation_errors.**

```
validation_errors = [validation_err_10, validation_err_50, validation_err_100, validation_err
_200, validation_err_500]
```

27. Now, we will plot the **training_errors** and **validation_errors** versus the number of trees. We will compare the 10, 50, 100, 200, and 500 tree models. We provide some plotting code to visualize the plots within this notebook.

28. Run the following code to visualize the plots.

```
plt.plot([10, 50, 100, 200, 500], training_errors, linewidth=4.0, label='Training error')
plt.plot([10, 50, 100, 200, 500], validation_errors, linewidth=4.0, label='Validation error')

make_figure(dim=(10,5), title='Error vs number of trees',
            xlabel='Number of trees',
            ylabel='Classification error',
            legend='best')
```

Quiz question: Does the training error reduce as the number of trees increases?

Quiz question: Is it always true that the validation error will reduce as the number of trees increases?