**Capstone Project 1**

Final Project

Model Report

For

URLease

***Executive Summary***

URLease is a leasing company that has had trouble being profitable because of a good number of their clients’ inability to pay their leases back in full. To remedy this problem, they hired me, their Data Scientist to build a predictive analytics model that would provide a score to be used at time of application to automatically approve or deny lease applications based on internal data and third party bureau data. The goal of the model would be to predict the likelihood of an applicant’s ability to pay back the lease in full.

From the loan\_status variable in the main dataset from Lending Club I generated a binary response variable called “Perf” (short for Performance) that takes one of two values: either a 1 for Good Performance (Status indicating Paid In Full) or a 0 for Bad Performance (Status indicating ChargeOff or Number of days Late.)

Since the response variable is a binary variable, I decided to build Classification models, and as such I decided to build a Logistic Regression model, a Random Forest model, an XGBoost model and a Gradient Boosting model. My hypothesis is that the XGBoost model would outperform the other 3 out of the box, or maybe with some hyperparameter tuning.

This report will show the steps taken in order to gather the data, merge it, wrangle it, do Exploratory Data Analysis (EDA), prepare the data for modeling, and finally build several models to predict Payment In Full.

***Data Acquisition, Wrangling and EDA***

The data I used in building the model is comprised of three data sets:

1. A Lending Club data set from Kaggle that consists of accepted applications from 2007 through 2018 which I will be filtering down to the last year: 2018.

2. A Bureau of Labor Statistics – State Unemployment data set for 2017-2018 which I had to build from several files downloaded from their website.

3. A World Population Review State Abbreviation data set to be used to link the first data set with the second.

The Lending Club data set from Kaggle came somewhat clean and thus did not need much prep work. From the second file, I removed a few variables such as 'labor force' ,'employment' and ,'unemployment' which I wasn’t going to use. I also removed the '(R)' from the unemployment variable values and I renamed variables to not include spaces in the names. Finally, I removed all records associated with the State of Puerto Rico from the data set because the data was erroneous. From the third file I removed the 'Abbrev' variable.

After merging the files together, I removed merge key variables that the merging process generated. I also removed variables that had 0 non-nulls. Then I removed a variable that had a constant value. I then removed 20 variables that had 100% missing values or virtually 100% (99.9%) as these would have no value to the model. With the data merged, I generated the binary response variable”Perf” (short for Performance) which is set to 1 when the loan\_status is “Fully Paid” and 0 if it is in('Late (16-30 days)','Late (31-120 days)','Charged Off'). I then removed all records where the “Perf” variable was not set to either 1 or 0.

I then generated a list of categorical columns which I first imputed with the word “Unknown” as preparation for Label Encoding. Once imputed, I Label Encoded them.

After imputing and Label Encoding the categorical columns, I worked on building two lists of numeric columns: 1) A list of columns with continuous values and another 2) A list of columns with discrete values. Using the list of columns with continuous values, I imputed the missing values by inserting the mean of each column in lieu of the missing values. Using the list of columns with discrete values, I imputed the missing values by inserting the median of each column in lieu of the missing values.

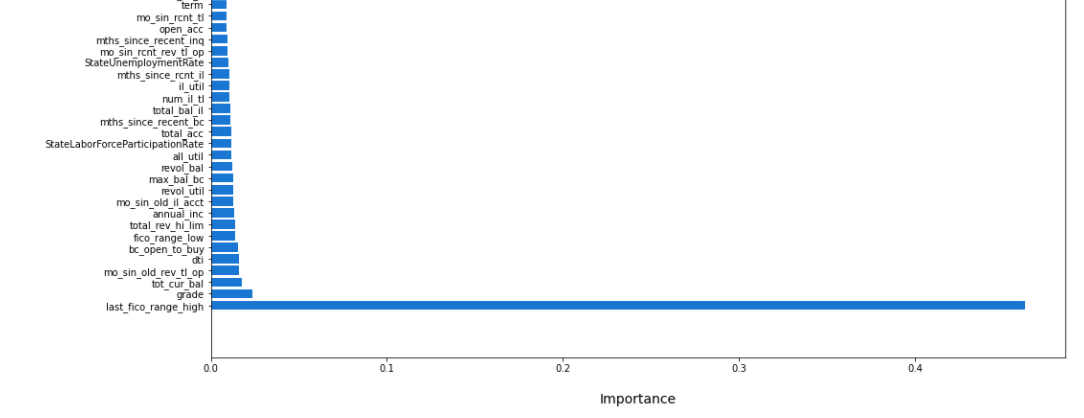
In taking another peak at the data in the modeling dataset, I realized I still had some variables that were related to performance metrics, and hence would have not been available at time of application. I therefore removed those.

At this point my modeling dataset no longer had missing data, so it was time to work on ridding the dataset of outliers. To accomplish this, I built a function that takes two parameters: a dataset and a list of columns.The function loops through the list of columns from the dataset and finds the lower and upper bounds from which to remove outliers. It does this by looking at the value 3 standard deviations away from the mean to the left of the mean (lower bound) and the value 3 standard deviations away from the mean to the right of the mean (upper bound). Any value in the column that is to the left of the lower bound or to the right of the upper bound gets removed.

The function also builds a Box Plot with whiskers and a histogram of each column prior to removing the outliers, but also after removing the outliers to check the proper functioning of the outlier removing function.

After removing the outliers, I wanted to use Recursive Feature Elimination with Cross Validation (RFECV) so my dataset would be left with the most important variables for modeling. Prior to running RFECV, I used the DataFrame built in correlation function to find which variables were highly correlated ( > 0.8). I then removed them from the modeling dataset.

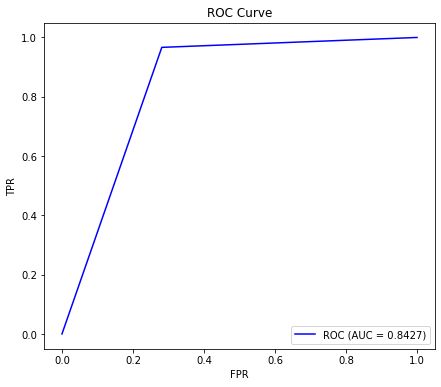
Before actually running the RFECV process I had to generate a subset of my modeling dataset (too many records and variables) otherwise my machine wouldn’t be able to handle it. I used a 25% random sample. To run RFECV, I used a RandomForestClassifier as my estimator and my cross validation was set to 5-fold and my scoring set to ‘accuracy’. I used the n\_features\_ attribute to look at the resulting optimal number of features and support\_ attribute to find out which features were set to False for support\_ attribute. I then removed those. Having done all the necessary data processing prior to model building, I wanted to take a quick look at the remaining 82 features available for modeling. Below is a snippet of the RFECV - Feature Importances graph I built. As you can see the one feature that sticks out as very important is the last\_fico\_range\_high variable.

****

At this point, I was ready to start the modeling process, so I used train\_test\_split from the model\_selection in the sklearn package to split my modeling dataset into X\_train, X\_test, y\_train, and y\_test using a test size of 20%.

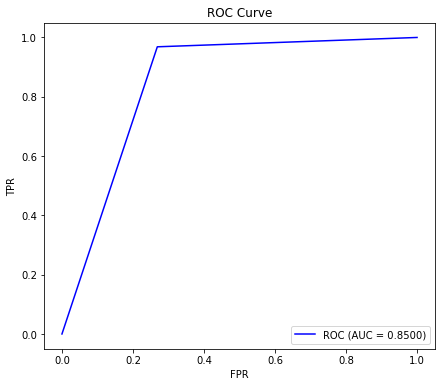
***Logistic Regression***

The first model I decided to build was a Logistic Regression model. For this model, the first step was to use StandardScaler() from preprocessing in the sklearn package. I used the StandardSaler() to fit the X\_train dataset and generate a scaler. With this scaler, I transformed X\_train and X\_test to scale the data. Then I generated a LogisticRegression object and I fit it with X\_train, y\_train and used it to predict on X\_test. I then printed out the accuracy\_score of 0.8989022798802487 and the AUC & ROC score of 0.8427205591748035 and I built a ROC Curve graph:



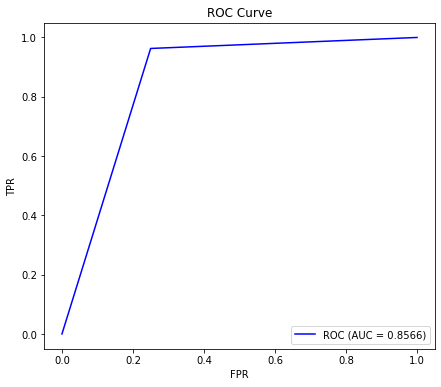
***Random Forest Classifier***

The second model I decided to build was a RandomForestClassifier model. For this model, I generated a RandomForestClassifier object, I used it to fit X\_train and y\_train and used it to predict on X\_test. Again, I printed out the accuracy\_score of 0.9037383894987334 and the AUC & ROC score of 0.8499914546449344 and I built a ROC Curve graph:



***XGBClassifier***

The third model I decided to build was an XGBClassifier model. For this model, I generated a XGBClassifier object, I used it to fit X\_train and y\_train and used it to predict on X\_test. Again, I printed out the accuracy\_score of 0.9048130805250634 and the AUC & ROC score of 0.8566023697931192 and I built a ROC Curve graph:

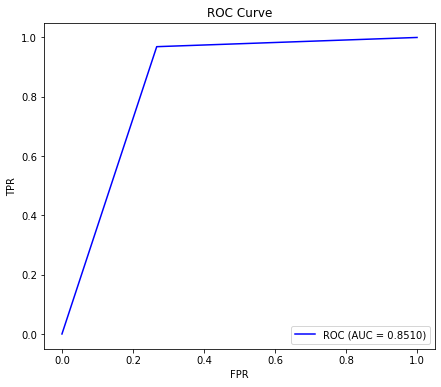


For this model, since it had a better accuracy score and better ROC & AUC score than the first two, I decided to go deeper and start tuning it. To this end, I generated a dictionary with two keys: learning\_rate and n\_estimators which both had lists for values. These lists are lists of values to test for tuning. I then used GridSearchCV() to generate a tuning object by passing in my dictionary as the param\_grid. I then used the tuning object to fit on X\_train and y\_train. Then I printed out the XGBClassifier parameters that returned the best accuracy score. My accuracy score was now 0.9134681363456659, which is an improvement from my first run for this particular model.

Based on the good results above, I decided to tune the model a bit more. This time I would tune the max\_depth parameter. Again I created a new dictionary. This time it would only have one key: max\_depth. I then used GridSearchCV() to generate a tuning object by passing in my dictionary as the param\_grid. I then used the tuning object to fit on X\_train and y\_train. Then I printed out the XGBClassifier parameter that returned the best accuracy score. My accuracy score was now 0.9141398329705319, which was an improvement from even my second run for this particular model. The final run for this particular model was more accurate than the last two models above.

***GradientBoostingClassifier***

The fourth model I decided to build was a GradientBoostingClassifier model. For this model, I generated a GradientBoostingClassifier object, I used it to fit X\_train and y\_train and used it to predict on X\_test. Again, I printed out the accuracy\_score of 0.904506025946112 and the AUC & ROC score of 0.8510454114410436 and I built a ROC Curve graph:



For this model like the one just above, since it had a better accuracy score and better ROC & AUC score than the first two straight out of the box, I decided to go deeper and start tuning it as well. To this end, I generated a dictionary with two keys: learning\_rate and n\_estimators which both had lists for values. These lists are lists of values to test for tuning. I then used GridSearchCV() to generate a tuning object by passing in my dictionary as the param\_grid. I then used the tuning object to fit on X\_train and y\_train. Then I printed out the GradientBoostingClassifier parameters that returned the best accuracy score. My accuracy score was now 0.9125085907843223, which is an improvement from my first run for this particular model.

Based on the good results above, I decided to tune the model a bit more. This time I would tune the max\_depth parameter. Again I created a new dictionary. This time it would only have one key: max\_depth. I then used GridSearchCV() to generate a tuning object by passing in my dictionary as the param\_grid. I then used the tuning object to fit on X\_train and y\_train. Then I printed out the GradientBoostingClassifier parameter that returned the best accuracy score. My accuracy score was now 0.9130843218041098, which was an improvement from even my second run for this particular model. However, the final run for this particular model was a bit less accurate than the last run for the XGBoost model.

***Conclusion:***

We went through the steps that were necessary during the Data Acquisition, Wrangling and EDA. Those included loading data from three different sources, processing the data in preparation for merging the records together, then merging, then cleaning data in columns, removing some columns, generating the response variable, removing records with no response variable values, imputing missing values on categorical variables and Label Encoding them, imputing missing values on variables with continuous numeric values by their mean, imputing missing values on variables with discrete numeric values by their median and removing outliers using a custom function.

We also went through removing highly correlated variables from the modeling dataset, using Recursive Feature Elimination with Cross Validation (RFECV) to reduce the number of variables in the final modeling dataset.

Then we went through the process of building the four classification models: Logistic Regression, Random Forest Classifier, XGBoost Classifier, and Gradient Boosting Classifier. We went through the process of tuning the best two: XGBoost Classifier and Gradient Boosting Classifier and concluded that after tuning the best Classifier turned out to be the XGBoost Classifier as we had theorized.

Final thoughts: I could have built more models to compare to, straight out of the box, but also after tuning. I could have also used pipelines to make the code clearer and more organized. I will seriously consider building more models and using pipelines for the next project.