|  |
| --- |
| OPIM-5604 – PREDICTIVE MODELING |
| LENDING CLUB - PREDICTING BAD LOANS |
|  |

|  |
| --- |
| TEAM-4 RAJAN MEHTA, CHARANJITH THOTTATHIL, KIRAN VARRE, SRI PRIYA BANDI, KRISHNA SHAH  4-27-2018 |

**DEFINITION**

**INTRODUCTION:**

Lending Club is the world’s largest peer-to-peer marketplace connecting borrowers and investors.

Since its inception in 2007, the number of loans in the marketplace has increased exponentially.

Lending Club provides its historical loan information every year. In this paper we attempt to use this data to build a number of supervised learning techniques that can predict whether a borrower will default so that investors can avoid those borrowers. A loan status is either good/bad so that the company can approve/decline the new loan applications.

**PROBLEM DESCRIPTION:**

Predicting loan defaults is a *binary classification* problem - a borrower will either default at some time during the loan term or finish the payment. The dataset contains approximately 42,500 records of loan information from year 2007 to 2011, having 145 columns (predictors). Out of these records, only 20% represents the loan default data. Therefore, the machine learning task here is an *imbalanced two-classed classification*.

We followed the *SEMMA* approach to come up with the final model and each step is documented in the sections further.

**METRICS:**

We chose the Area Under Curve (AUC) as metrics to evaluate learning performance of our models. This is because the calculation of AUC is independent of the proportions in which two classes exist in the test dataset. This makes the AUC useful for evaluating the performance of classifiers on unbalanced data sets. Also, we use Lift Curve to visualise the performance of our classification model as opposed to the baseline model.

With the models generated, we found that Random Forest showed the best performance with an AUV of 0.89 and with top 30% portion of data, it predicts twice as good as the baseline model.

**ANALYSIS**

**DATA PREPROCESSING**

The data pre-processing was done in *SAS 9.4*. The corresponding SAS data preparation program & log files and the field layout file which consists of the list of fields we have in the data and the action performed on them are attached below for reference:

*Programs & Logs*:

****

****

*Field Layout File:*

****

**Listed below are the actions performed on the data:**

1. **Removed Unnecessary Data Fields:**

Out of 145 fields, 125 fields were removed from the data that did not contribute any information pertaining to the default loan prediction. The categories are as follows:

* 1. *Blank Fields*: 82 fields were removed as there were no values available in these fields throughout the file (Ex: ‘Mths\_Since\_Last\_Major\_Derog’, ‘Max\_Bal\_Bc’)
  2. *Less than 10% Data***:** 3 fields were removed before there was less than 10% of the lines populated with information
  3. *Single Value:*9 fields were removed as the was only one value populated in these fields throughout the file (Ex: ‘Application\_Type’ = “Individual”, ‘Disbursement\_Method’ = “Cash”)
  4. *Irrelevant:* 23 fields were removed as they were considered irrelevant for the prediction analysis. They were related to investors or settlement which did not affect the prediction for good or bad loans.
  5. *Collinearity:* 3 fields were removed dueto its collinearity with the ‘loan\_amnt’ field. The collinearity checks results are present in the sheet: ‘Collinearity’
  6. *Part of Grade:* As per the information available from Lending Club, the field ‘Grade’ already captures the variability in the fields ‘Sub\_Grade’ and ‘Int\_Rate’. Hence, the ‘Grade’ field was retained and the remaining two fields were excluded from the analysis. Please refer sheet: ‘Grades’.
  7. *Descriptions:* 2 description fields like ‘Title’ which is the applicant title and ‘Desc’ the description provided by the applicant were removed as they were not deemed relevant for the analysis as there is no pattern or variability which could be captured.

1. **Removed Outliers:**

From the remaining numeric fields, all the lines with values 3 standard deviations away from its mean were excluded from the analysis.

1. **Data Cleansed**:

The numerical data fields with incorrect format were cleansed and converted to numeric (Ex: 21% - 21, 36 Months - 36).

1. **Values Grouped:**

The values in few of the fields were grouped as it meant the same (Ex: Verification = “Source Verified” or “Verified” were considered as “Yes” and Verification=”Not Verified” was considered as “No”).

1. **Converted Ordinal Variables to Ordered Numeric Variables:**

All theordinal variables were converted to numeric variables. (Ex: Grade = A, B, C…. G was converted to Grade\_Order = 1,2,3….7 respectively).

1. **Converted Nominal Variables to Numeric Variables**:

All the nominal variables were converted to ordered numeric variables by creating dummy variables for each of the unique values present in the respective field. (Ex: Verification\_Status = “Yes” or “No” converted to Verification\_Status\_Yes=1 or 0 and Verification\_Status\_No=1 or 0).

1. **Replaced Nulls with 0***:*

All the fields which had null values were replaced with ‘0’ as it appeared to be missing. (Ex: ‘Open\_Acc’, ‘Pub\_Rec’).

1. **Derived Variables:**

A variable ‘years since credit line’ was created. It was considered as the difference between the years from the first time the applicant has opened a credit line till the issue date. (Years\_Since\_CR\_Line).

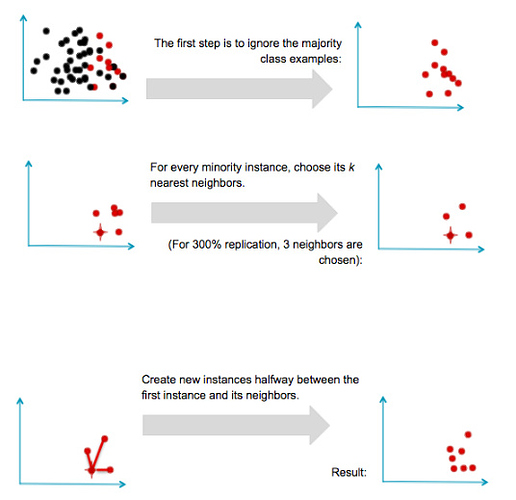
1. **Classifier:**

The fields ‘Loan\_Status’ is the classifier field which indicated if the loans are ‘Bad’ or ‘Good’. Any record containing the value – ‘Charged off’ in this field was considered as bad loan – “1” and the rest were considered as good loan – “0”.

**Algorithms and Techniques**

**Over-sampling for imbalanced data**

The loan dataset is a time-series dataset with over 400,000 rows and imbalanced data (20% of data belongs to one class). (figure below left)

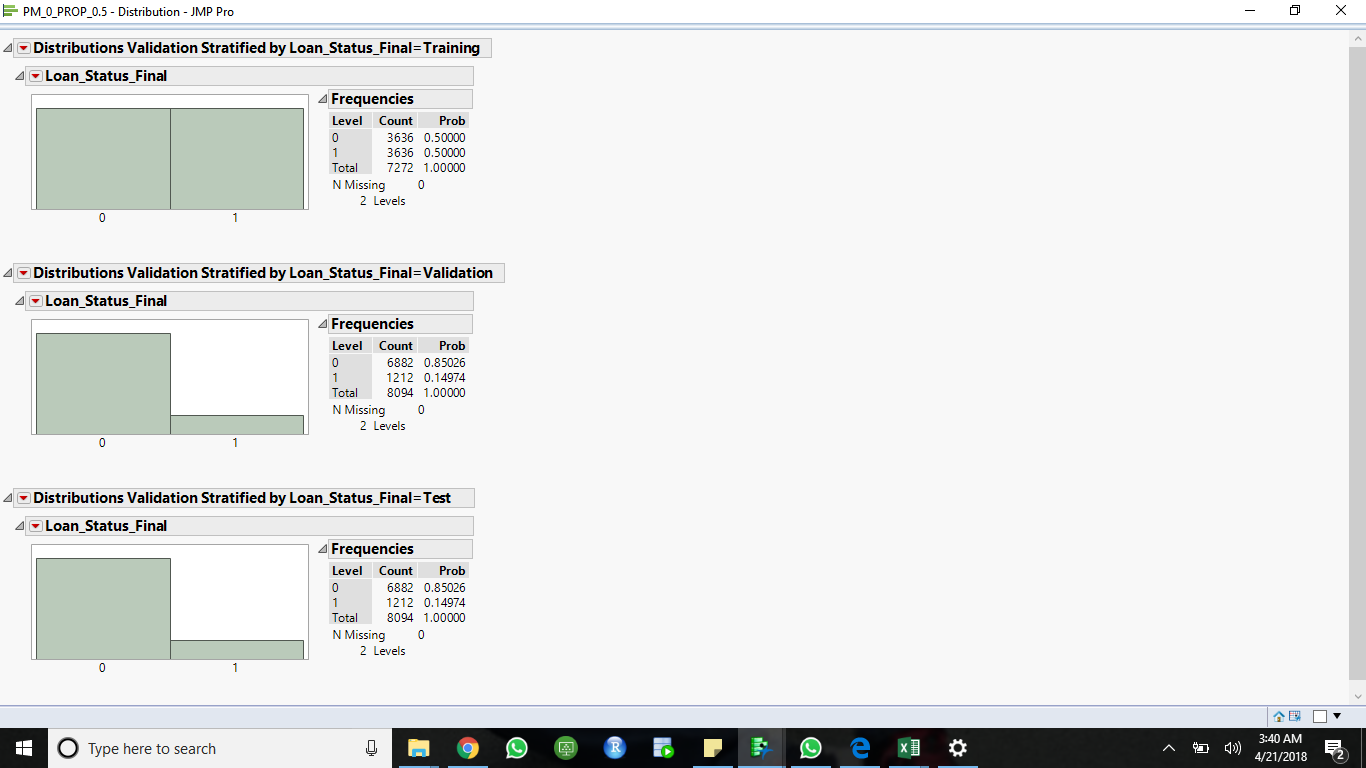
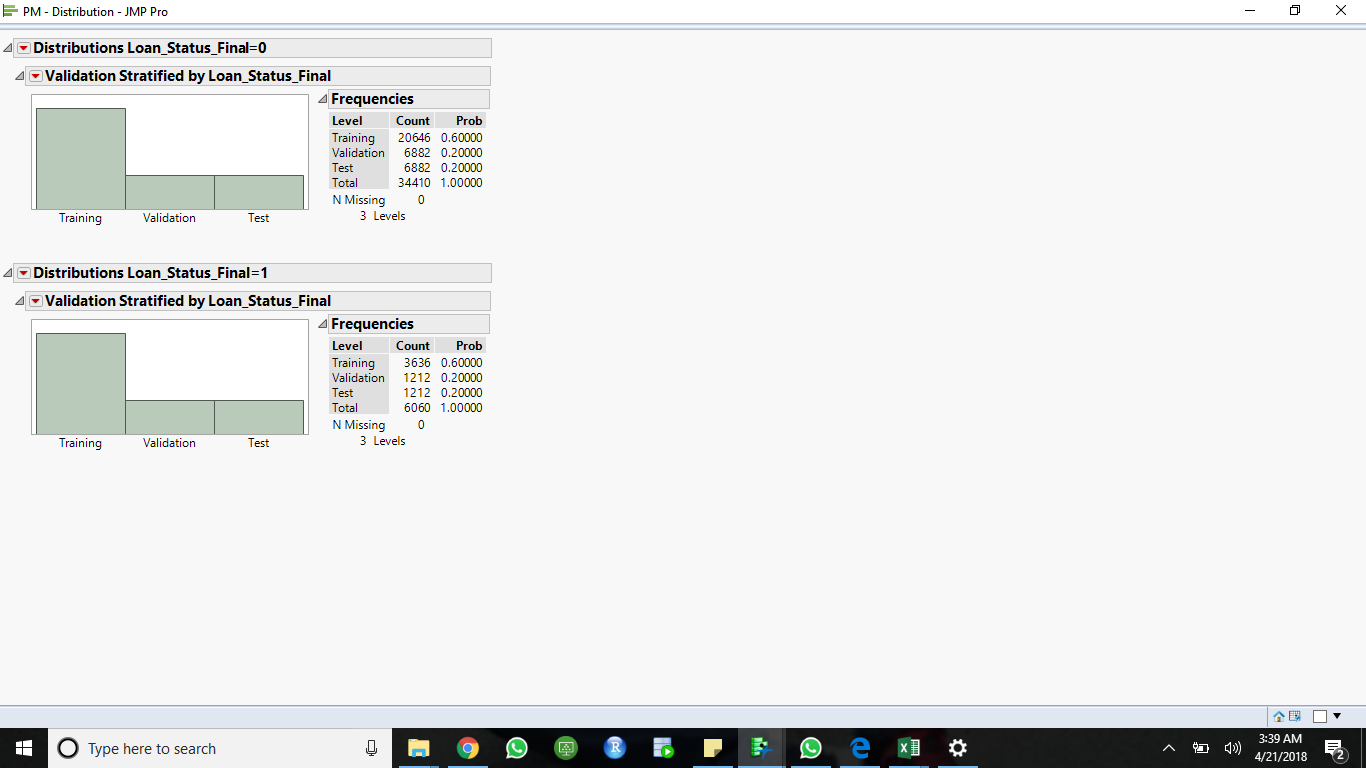


Therefore, over-sampling was performed on the data to achieve an equal number of sample with the majority or minority class. Without over-sampling, the classifier would be more sensitive to detecting the majority class and less sensitive to minority class. In Python, SMOTE (Synthetic Minority Over-sampling TEchnique) was used for oversampling the dataset. The process is as described in image[2].

**Note:** In Python, the dataset was split into Training data and Test data (75-25 ratio). No Validation set was

used. The algorithms were trained using 10-fold cross validation.

In JMP, an add-in called *Stratified Split Balanced* was used for the purpose of over-sampling. (figure above right). The train-test-split was done in ratio 60-20-20.



**Classification Algorithms:**

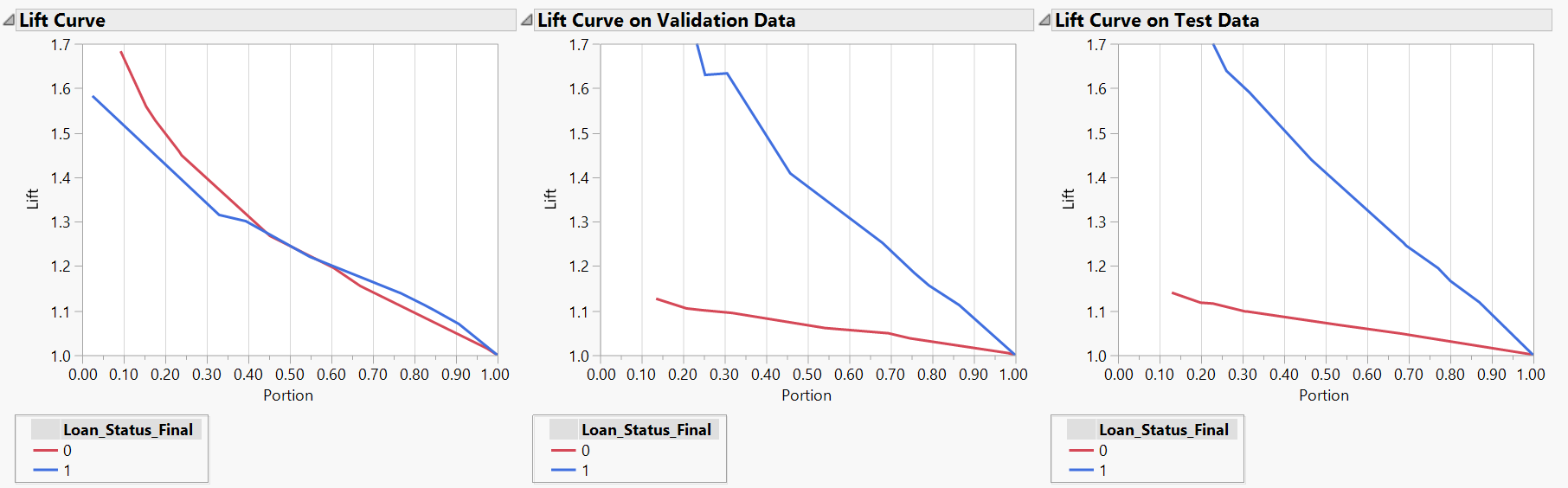
We have implemented the following Classification Algorithms using SAS JMP and Scikit-learn – a Machine Learning library in Python:

1. Classification Tree
2. Bootstrap Forest
3. Boosted Tree
4. Random Forest
5. Support Vector Machines
6. Logistic Regression
7. **CLASSIFICATION TREE**

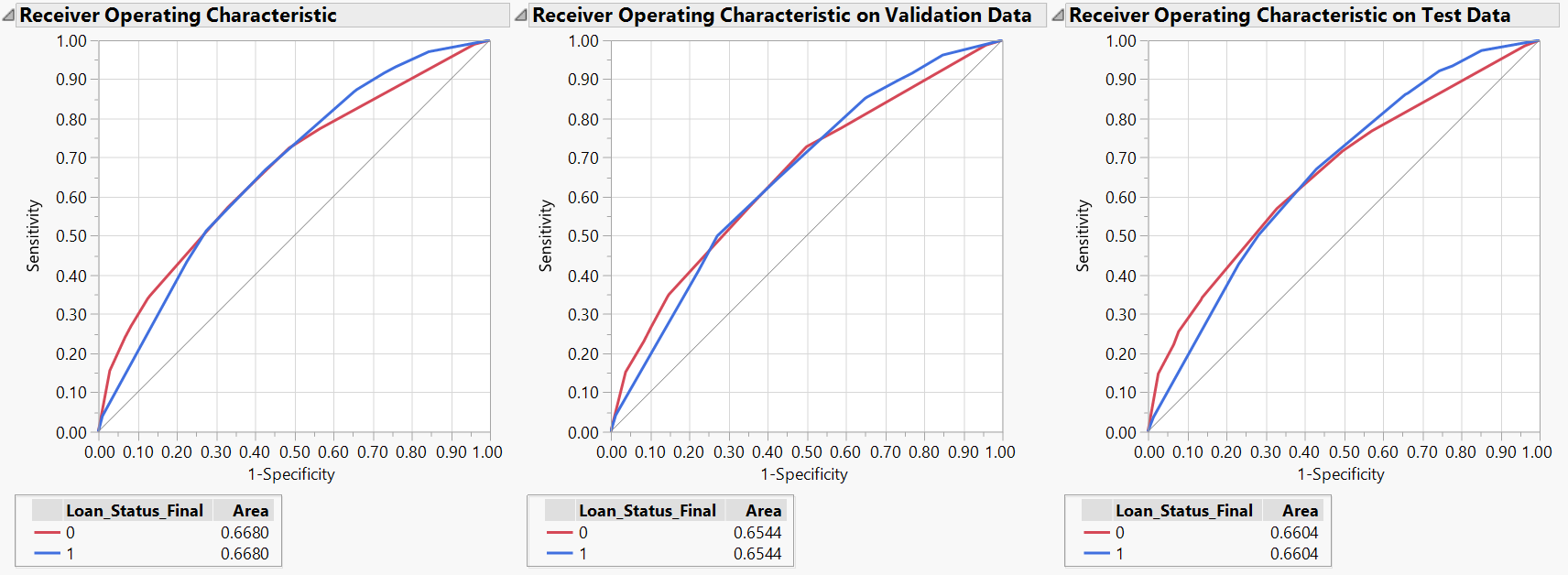
Firstly, the full model was built using all the columns. The lift on test data showed that for top 30% portion of sample data, default loans would be predicted 1.7 times better than the base model while good loans would be predicted as good as the base model. The AUC rate is close to 0.50 which is undesirable.

Therefore, we reduced the number of variables using Principal Component Analysis (PCA). Unfortunately, the results were not that satisfactory. The AUC of the new model was 0.66 and lift was still 1.7

Lift Curve:



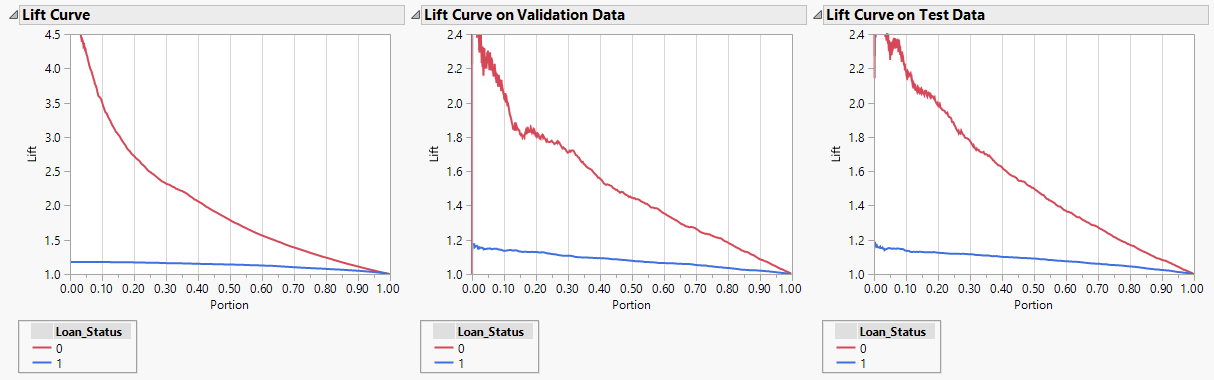
ROC:



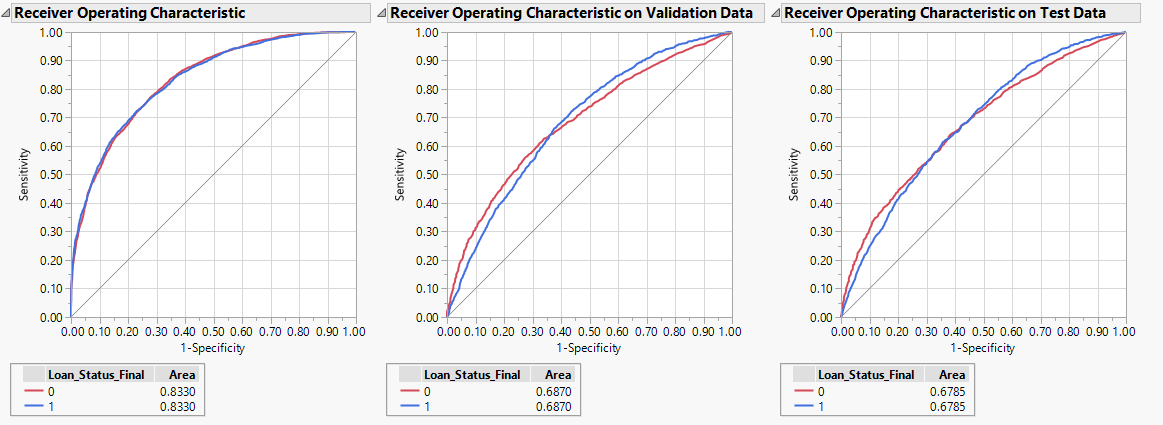
1. **BOOTSTRAP FOREST**

The misclassification rate for training set is 0.26 while for validation set is 0.38. The test set misclassification rate is 0.39 which implies that the model is not fit for the data. The AUC for the test data is 0.6785.

Lift Curve:

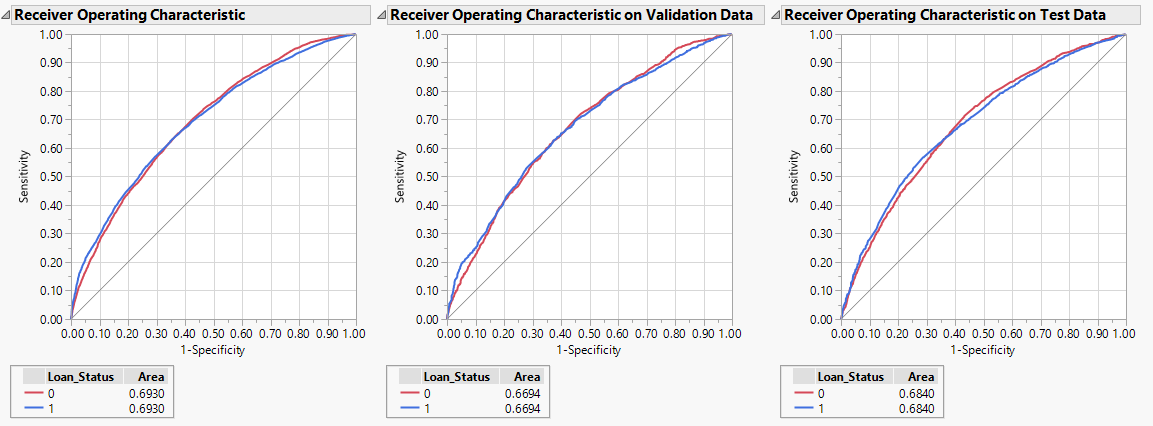


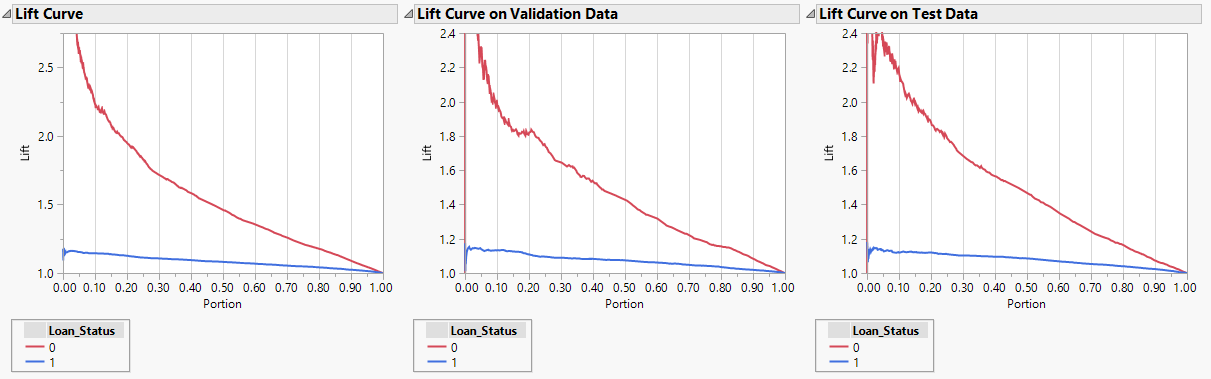
ROC:



1. **BOOSTED FOREST**

The AUC for training data set is 0.693 while 0.67 for validation data. The AUC for the test data is 0.684. Lift for training and validation data sets are 1.8 and 1.65 respectively. Lift for 30% of test data is 1.65 approximately. The Iift values decreased for the test data compared to training which implies that the model is not best fit for given data. Similarly, AUC of test data is also lower compared to training data reflecting that the output prediction is good for future input data.





1. **RANDOM FOREST**
2. **CHOOSING THE BEST PARAMETERS**

We automated the choosing of the best tuning parameters (Hyper-parameters) for training the model. Hyper-parameters are the parameters specified outside of the training procedure. The training process doesn’t derive the values of these parameters. Therefore, we have to try all possible combinations of these parameters to derive the best one.

Examples of Hyperparameters:

For knn - **n\_neighbors** , **weights , etc**

**For decision trees - criterion** , **splitter** , **max\_features** , **max\_depth** , etc.

For Random forest classifiers - **n\_estimators** , **max\_depth, max\_features** , etc

As in the example above above, there are a number of parameters that can be tweaked in order to get the best Random Forest model. The goal is to find the parameters that can give the best cross-validation score.

Given below is the pseudo-code of the hyperparameter tuner. It outputs the setting that yields the best performing model.

Hyperparameter\_tuning[1] (training\_data, hp\_list):

hp\_perf = []

foreach hp\_setting in hp\_list:

validation\_results = cross\_val\_score (training\_data)

hp\_perf.append (validation\_results)

best\_hp\_setting = hp\_list [max\_index (hp\_perf)]

best\_model = train\_model (training\_data, best\_hp\_setting)

return (best\_hp\_setting, best\_model)

Now, with respect to the Random Forest, we need to find the best tuning parameters. To reduce the computation time, we fixed the number of trees in the forest (**n\_estimators**) to 25 and implemented the model with all combinations of *max\_features* and *max\_depth.*

*max\_features* is the number of features to consider when looking for the best split with possible values:

* If “sqrt”, then max\_features=sqrt(number of features).
* If “log2”, then max\_features=log2(number of features).

*max\_depth* is the maximum depth of the tree ranging from 3 to 30.

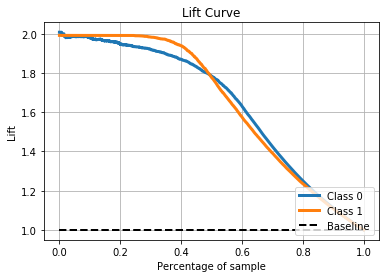
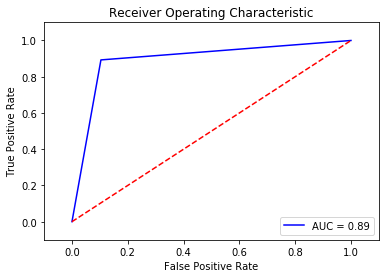
The criterion for split is gini index.

From all possible combinations, the model based on best cross validation score (mean accuracy) had 'max\_features' : 'sqrt' and 'max\_depth': 27. The last step is to train a new model on the entire dataset (training and validation) under the best hyperparameter setting.

1. **Building the Model**

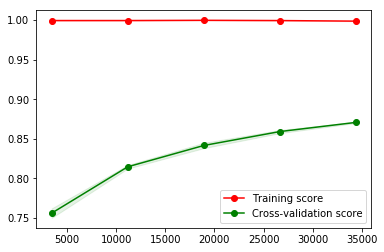
From this exhaustive search, we got the values for our parameters. Now, we can build our Random Forest Classifier model using the training data and above derived parameters.

The model gave an AUC of 0.89

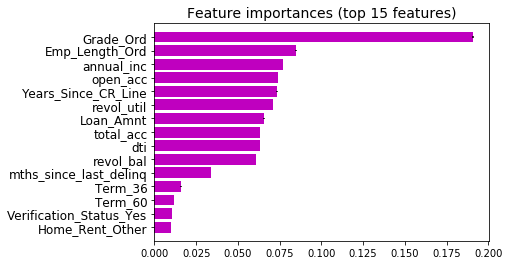
****

The lift curve shows that for top 30% sample, the model performs 2 times better than the baseline model.

The learning curve of the Random Forest model is as shown below:



The top 15 important features according to impact on the output variable are shown below:

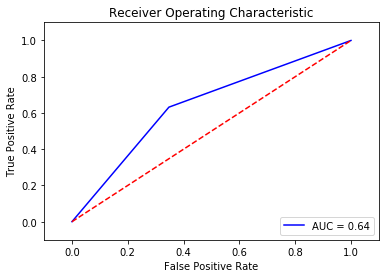
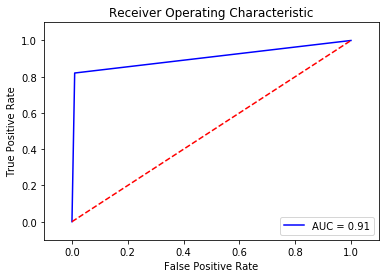


Next, we took the top 10 features from the graph shown above and trained a new Random Forest model using the same parameters used earlier. Thus, the complexity reduced, but that was no improvement in the performance.

Also, initially we fixed the number of trees in the forest as 25 to reduce the computation time. Now, we tried different values of “n\_estimators” i.e. trees with values like 10, 15, 25, 50, 100, 200. Upto 200, the AUC only increased upto 0.91 while it decreased upto 0.76 when the number of trees where reduced to 10. There was no change in AUC in case of 20 trees in the Random Forest. So, finally, we decided to go for a model with 20 trees.

1. **SUPPORT VECTOR MACHINES**

As support vectors are effective in high dimensional spaces and versatile (different kernel functions can be used), we decided to implement it. So, we tried random values of **C (penalty)** and **gamma (Kernel coefficient for gaussian).** We got an AUC = 0.64 for linear SVM (C=1) and we got an AUC of 0.91 for gaussian SVM (C=1, gamma=2).



The possible parameters for hyperparameter tuning of an SVM model could be:

* kernel type to be used in the algorithm with possibilities [‘linear’, ‘poly’, ‘rbf’] for linear SVM, polynomial SVM and gaussian SVM respectively.
* Penalty parameter C with possible values [1, 10, 100, 1000]
* Degree of the polynomial kernel function (‘poly’) with values [2, 3, 4]
* Gamma (Kernel coefficient for ‘rbf’, ‘poly’) with values [0.001, 0.0001]

Unfortunately, the fit time complexity is more than quadratic and with over 40,000 rows it proved impossible to do an exhaustive search for best parameters.

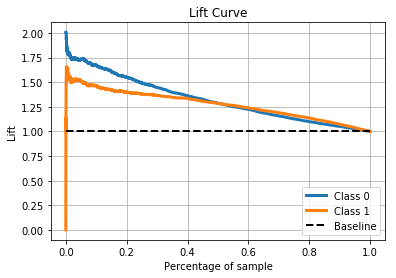
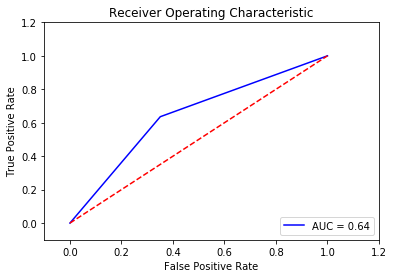
1. **LOGISTIC REGRESSION**

Taking similar approach as in Random Forest, with respect to the Logistic Regression, we need to find the best tuning parameters. The possible parameters are:

* Inverse of regularization strength C: a positive float. Like in support vector machines, smaller values specify stronger regularization.
* Weights associated with classes, ‘class\_weight’ with values [None, 'balanced']. The “balanced” mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data.
* Algorithm to use in the optimization problem, ‘solver’ with values ['sag', 'liblinear'].

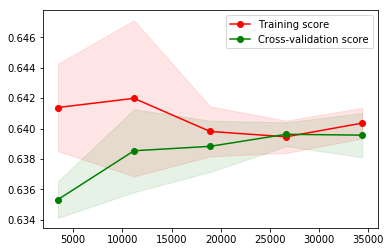
From all possible combinations, the model based on best cross validation score (mean accuracy) had

C=1.0, class\_weight=None, solver='liblinear'. By default, the norm used in the penalization is l2.



The lift curve shows that for top 30% sample, default loans would be predicted 1.3 times better than the base model while good loans would be predicted 1.5 times better than the base model.

The learning curve of the Logistic Regression model is as shown below:



**CONCLUSION**

Random Forest works best on the given unbalanced data based on the AUC metrics.

**FUTURE WORK**

1. In the analysis, we have removed the date variables. Actually, this is a case of chronological data. The loan data is from year 2007 to 2011. Since we are using 10-fold cross-validation, there are definitely going to be scenarios where we train the model on 2010 data and test it on 2007 data. Predicting earlier loans using later loans is not reasonable. Therefore, there should be a cross validation method where a fold containing future data is not tested by past data.
2. The number of loan applications are increasing at an exponential rate every year. As opposed to that, the proportion of bad loans is decreasing every year. So, exploring different types of oversampling methods and finding the one suitable for this problem would be fruitful.

**Acknowledgements**

We would like to thank Dr. Lee and Svecha for guiding us and consistently assisting us throughout the project.

**REFERENCES**

[1] <https://www.oreilly.com/ideas/evaluating-machine-learning-models/page/5/hyperparameter-tuning>

[2] <https://discuss.analyticsvidhya.com/t/smote-implementation-in-python/19740>

[3] <http://scikit-learn.org/stable/> - referred all documentation for machine learning in Python.