Capstone Project 2 Final Report

Home Credit Default Risk

Samin Rastgoufard
samin.rastgoufard@gmail.com

08-03-2019

Springboard

Abstract

In this report different possible solutions for the Home Credit Default Risk, competition on Kaggle has been explored. Data cleaning, exploratory data analysis, and feature selection techniques have been implemented. Base on the type of each feature the abundance of missing entries has been compensated. Several machine learning algorithms like Logistic Regression, Random Forest, Gradient boosting, Adaboost, and Multi-Layer Perceptron have been trained and hyperparameters tuned to get the best performances. Since the target value was very unbalanced, the area under the receiver operating characteristic curve, log loss and average precision-recall score used for learner evaluation. The total training elapsed time for each learner has been evaluated as well.

Business Objective: Home Credit Default Risk

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.
Home Credit needs an algorithm that will take as inputs various personal and alternative financial information originally taken from a loan applicant's profile, and then determine a probability of the applicant has at least one late payment when repaying their loan. This probability will be in the range [0.0, 1.0], where 1.0 represents a 100% certainty that the applicant will have at least one delinquent repayment and 0.0 indicates that there is zero chance that the applicant will ever be delinquent.
The algorithm will be tested on a set of 48,744 individuals who previously borrowed from Home Credit. Home Credit knows which borrowers ultimately paid off their loans, and which ones had one or more late payments. A good algorithm will need to predict a high probability of delinquency for the majority of borrowers who did actually make one or more late payments. This algorithm will also need to predict a low probability of delinquency for the majority of borrowers who eventually did successfully repay their loans with no late payments.

Data

The dataset as it is mentioned above was provided by Home Credit Group's data scientists. It contains a wide variety of personal and financial information belonging to 356,255 individuals who had previously been recipients of loans from Home Credit. These individuals are divided into training and testing sets. The training group contains 307,511 individuals' records. The test group contains 48,744 records. (https://www.kaggle.com/c/home-credit-default-risk)

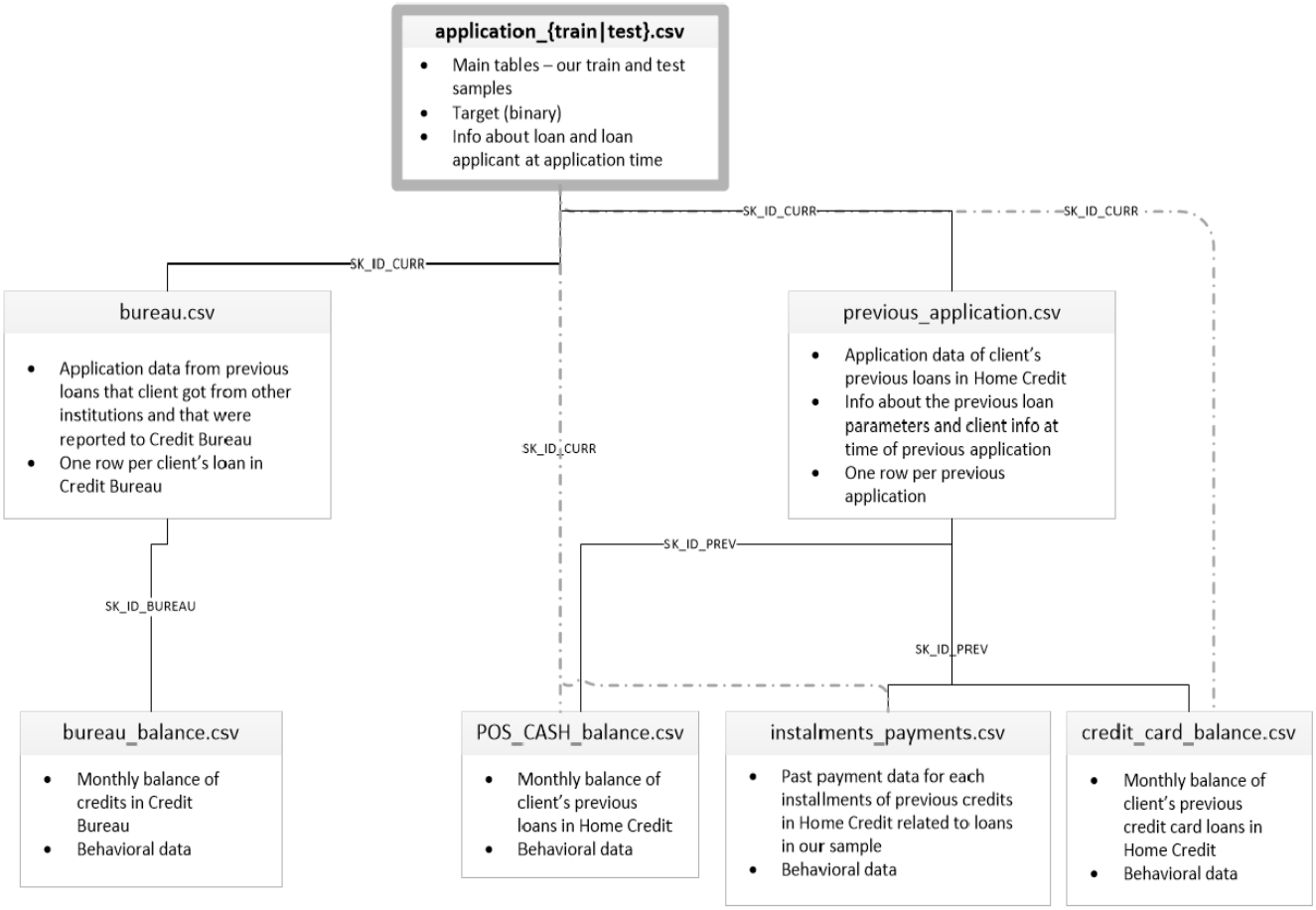
The dataset is anonymized, with each individual their loan ID. Any personally identifying, such as name, phone number, or address, has been omitted. Because Home Credit targets the unbanked

population, it is unable to rely on traditional measures, such as a credit score, that mainstream financial institutions use when making lending decisions.

Home Credit works around this obstacle by looking at an extensive and diverse array of personal and financial information for each of its applicants. These features range from common characteristics, such as marital status, age, type of housing, a region of residence, job type, and education level, to some incredible niche characteristics, such as how many elevators are in an applicant's apartment building.

Home Credit also looks at aspects of applicants' financial backgrounds, including month-by-month payment performance on any loans or credit card balances that the applicant has previously had with Home Credit, as well as the amount and monthly repayment balances of any loans that the applicant may have received from other lenders. All of these features are spread across seven data tables.

The main data table ('application_train.csv') contains 120 features that comprise applicants' personal background information. The other six data tables contain applicants' previous loan and credit card balance payment histories. The following diagram provides a brief summary:



Workflow

1. Collecting data
2. Starting exploratory data analysis to find trends and Storytelling
3. Conduct further data analysis to identify relationships between different variables
4. Feature Selection
5. Implement learning algorithms: Logistic Regression, Random Forest, Gradient Boosting, Adaboost Classifier, Multi-Layer Network
6. Model Evaluation and Validation (AUC, Log-Loss, Precision_Recall score)

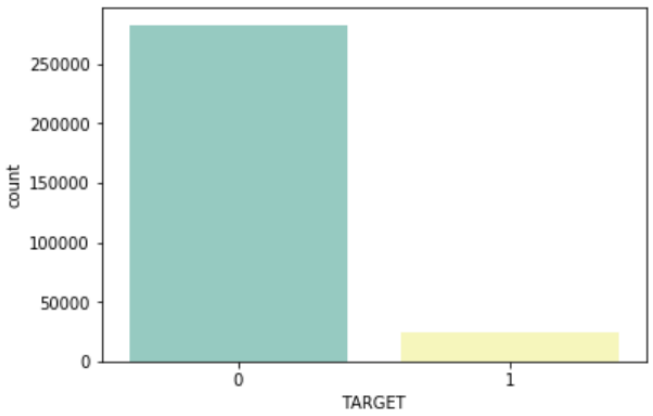
Implementing Steps

1. Collecting data

After importing necessary libraries, application_train.csv data with 307,511 observations and 122 features have been read.

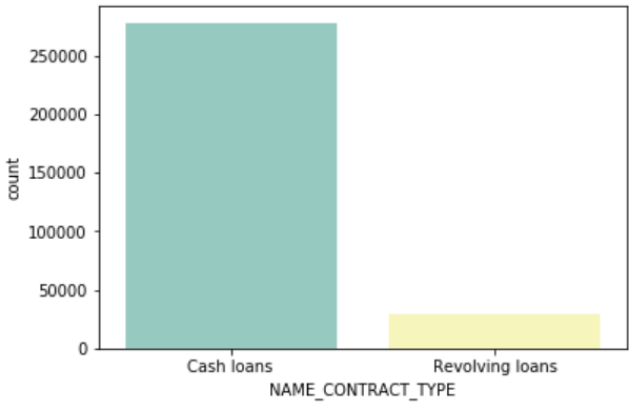
The Target value is unbalanced since most clients are getting cleared for getting the loan.

Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases).

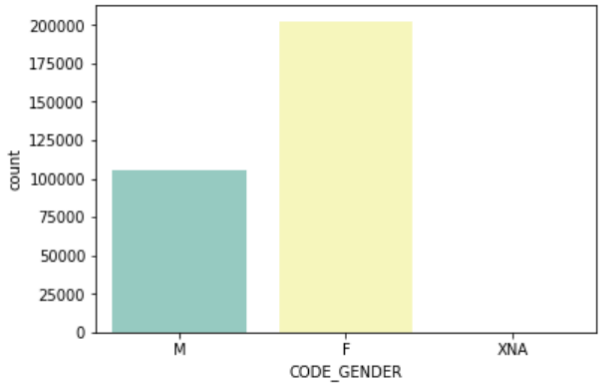


2. Exploratory Data Analysis

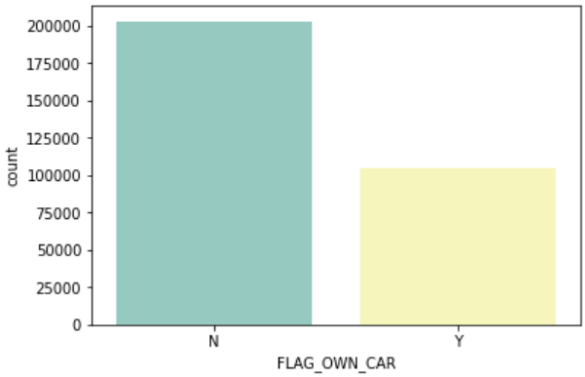
Around 90% of the contracts are 'Cash loans' and less than 10% are 'Revolving loans'.



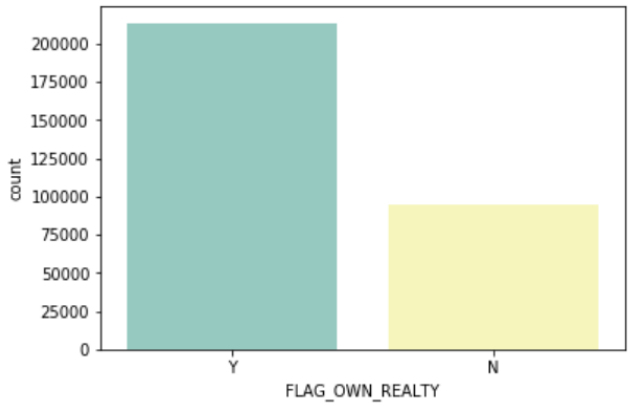
More than 66% of applicants are females.



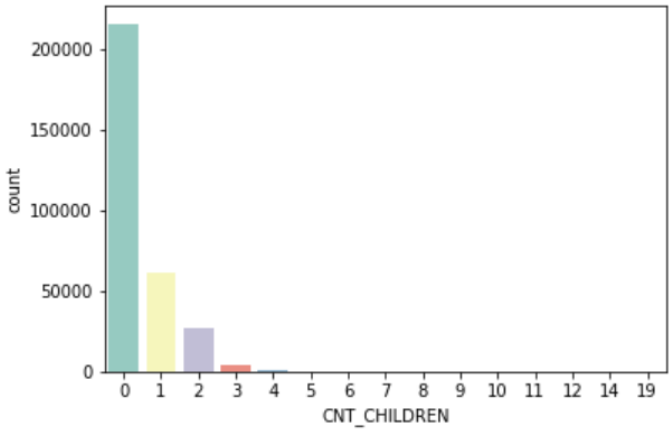
Around 66% of applicants do not have their own car.



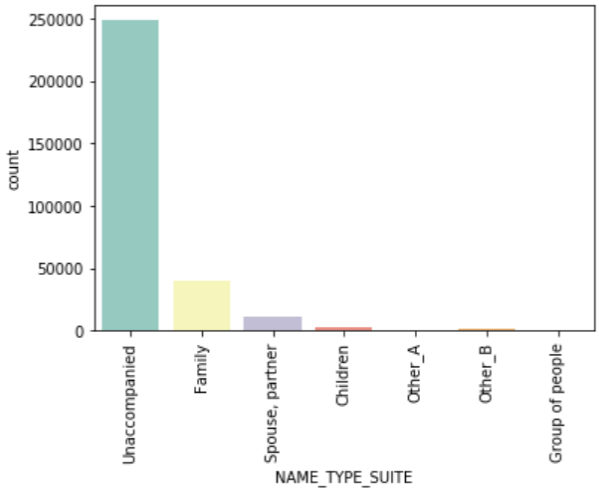
Around 70% of the applicants have their own reality.



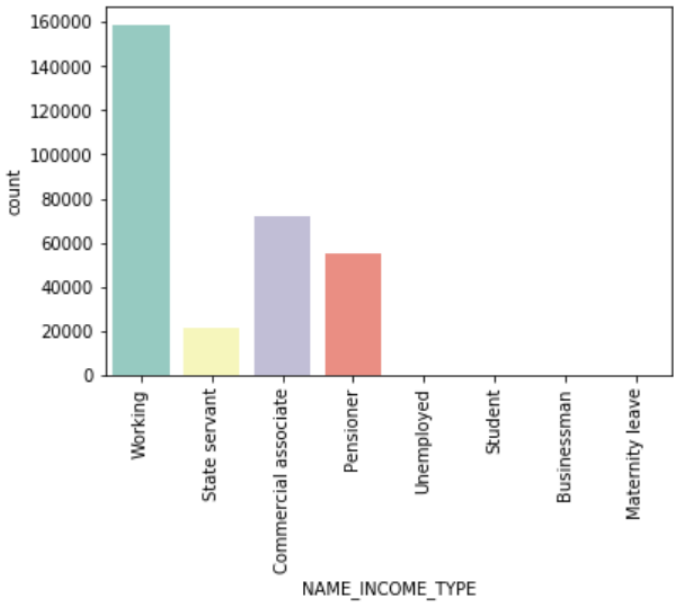
Most applicants don't have children.



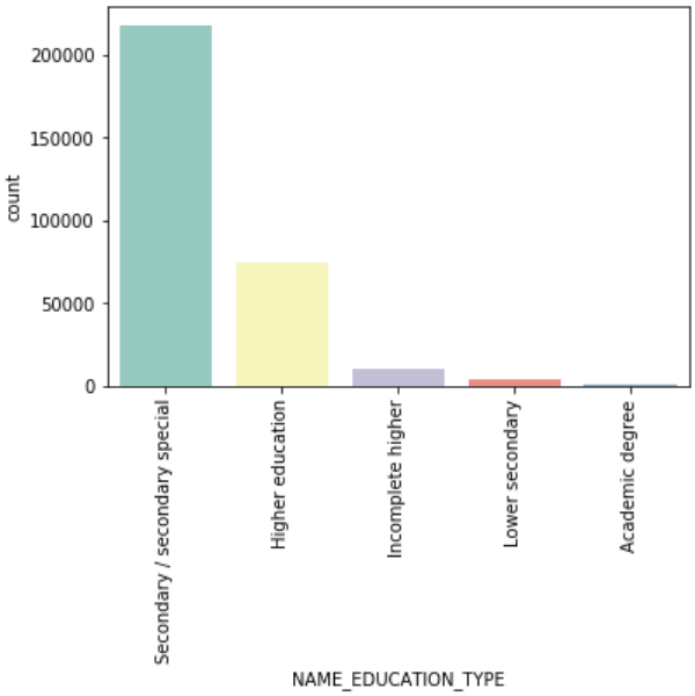
Most applicants are unaccompanied.



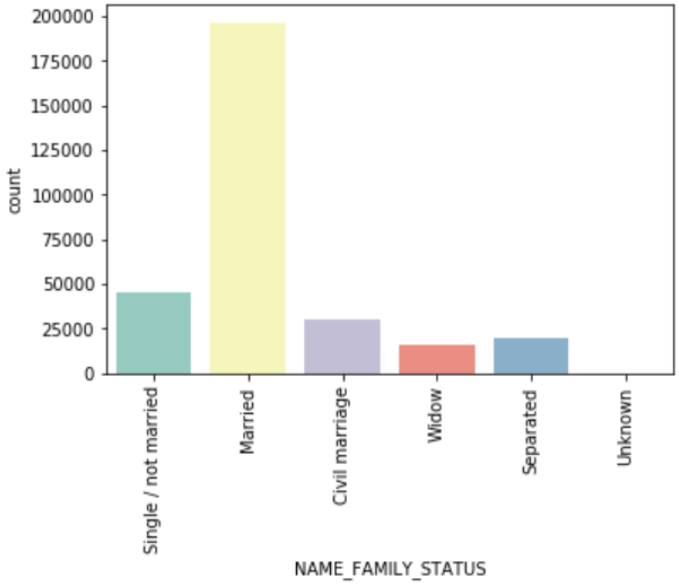
Income type for most applicants is from Working.



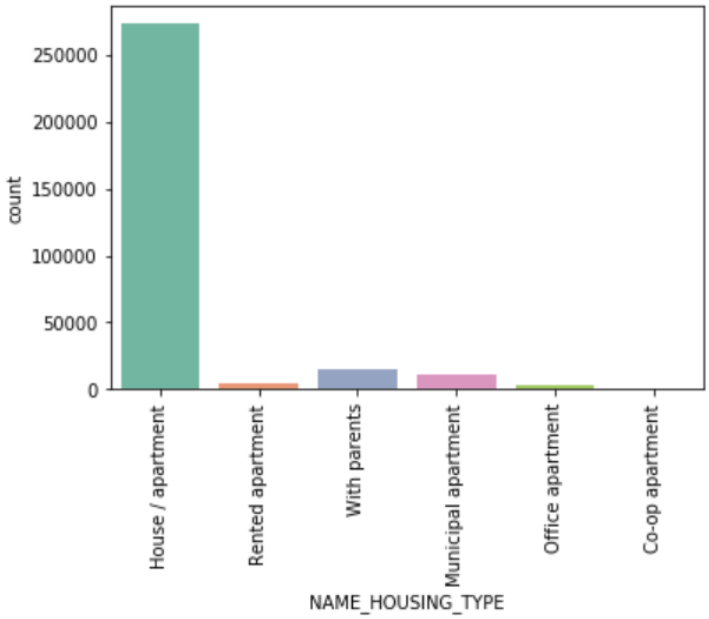
Education type is mostly Secondary.



Most Applicants are Single.



The housing type is mostly House/Apartment.



3. Feature Selection

Dimensionality reduction is a process of selecting subset of features for use in model construction. It helps the abundance of redundant and irrelevant features. It helps to overcome the curse of dimensionality and overfitting. The executing time will be much less as well. I tried different dimensionality techniques for this project as bellow:

I. **Percent Missing Values:** After changing the dummy variables to integers values, I checked the total number of missing values of each column. If the the total missing values are more than 50% I dropped that feature from the dataset. (Total drop: 41 columns)
II. **Amount of Variation:** Drop the variable with very low variation. (Total drop: 2 columns)
III. **Pairwise Correlation:** Many variables are often correlated with each other, and hence are redundant. If two variables are highly correlated, keeping only one will help reduce dimensionality without much loss information. But which variable to keep? The one that has a higher correlation with the target.
IV. **Correlation with the Target:** Drop variables that have a very low correlation with the target. If a variable has a very low correlation with the target, it's not going to be useful for the model.
V. **The weight of evidence and information value:** Weight of evidence (WOE) and Information value (IV) are simple, yet powerful techniques to perform variable transformation and selection. These concepts have a huge connection with the logistic regression modeling technique. It is widely used in credit scoring to measure the separation of good vs bad customers.

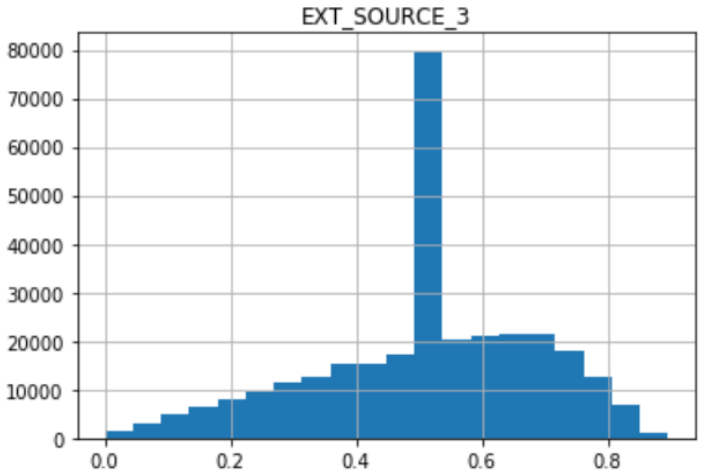
4. Fill NaN Value

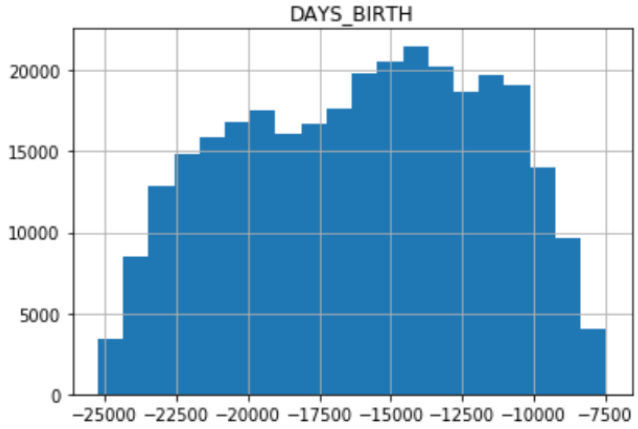
Fill NaN values of columns with a float or integer values with average, Fill NaN values of columns with dummy variables with the max value counts.

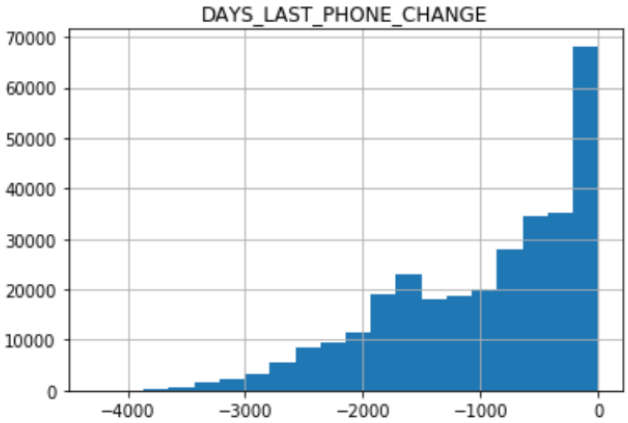
5. Information and Histogram of Selected features

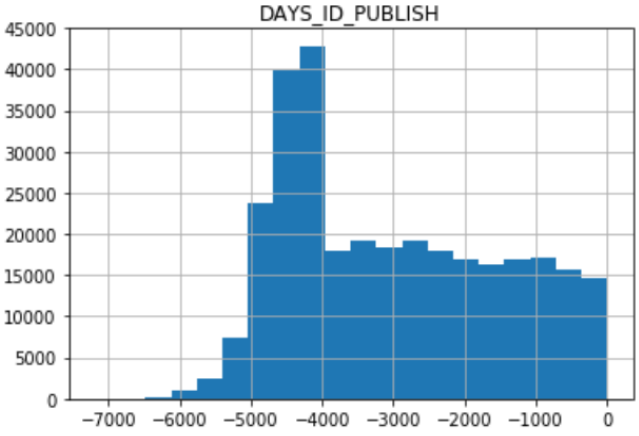
The 25 Selected features are:
EXT_SOURCE_2: Normalized score from external data source { normalized, scaled to range [0,1]
EXT_SOURCE_3: Normalized score from external data source { normalized, scaled to range [0,1]
DAYS_BIRTH: Client's age in days at the time of application { time only relative to the application
NAME_EDUCATION_TYPE: Level of highest education the client achieved
DAYS_LAST_PHONE_CHANGE: How many days before application did client change phone
DAYS_ID_PUBLISH: How many days before the application did the client change the identity document with which he applied for the loan { time only relative to the application
CODE_GENDER: Gender of the client 'AMT_GOODS_PRICE',
FLAG_EMP_PHONE: Did client provide work phone (1=YES, 0=NO)
REG_CITY_NOT_WORK_CIT: Flag if the client's permanent address does not match work address (1=different, 0=same, at the city level)
LIVE CITY NOT WORK CITY: Flag if client's contact
FLAG_DOCUMENT_3: Did the client provide document 3
REGION_RATING_CLIENT: Our rating of the region where the client lives (1,2,3)
FLOORSMAX_AVG: Normalized information about the building where the client lives, What is average (AVG sux), modus (MODE sux), median (MEDI sux) apartment size, common area, living area, the age of the building, number of elevators, number of entrances, state of the building, number of floor { normalized, scaled to range [0,1]
DAYS_REGISTRATION: How many days before the application did the client change his registration-time only relative to the application

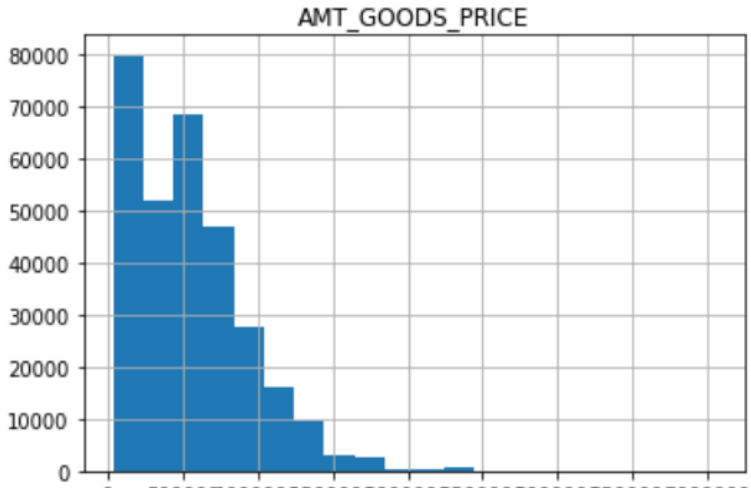
REG_CITY_NOT_LIVE_CITY: Flag if the client's permanent address does not match contact address (1=different, 0=same, at the city level)
TOTALAREA_MODE: Normalized information about building where the client lives, What is average (AVG sux), modus (MODE sux), median (MEDI sux) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor { normalized, scaled to range [0,1]
NAME_CONTRACT_TYPE: Identification if the loan is cash or revolving
DAYS_EMPLOYED: How many days before the application the person started current employment { time only relative to the application
FLAG_DOCUMENT_6: Did the client provide document 6
FLAG_WORK_PHONE: Did client provide home phone (1=YES, 0=NO)
FLAG_PHONE: Did client provide home phone (1=YES, 0=NO)
FLAG_OWN_CAR: Flag if the client owns a car
REGION_POPULATION_RELATIVE: Normalized population of the region where client lives (higher number means the client lives in the more populated region) { normalized (although distribution is normal with minimal skewness, values are in approximate range [0.00,0.07] { this is the only normalized feature in the main data table not distributed over the range [0,1])
NAME_HOUSING_TYPE: What is the housing situation of the client (renting, living with parents, ...)
HOUR_APPR_PROCESS_START: Approximately at what hour did the client apply for the loan rounded

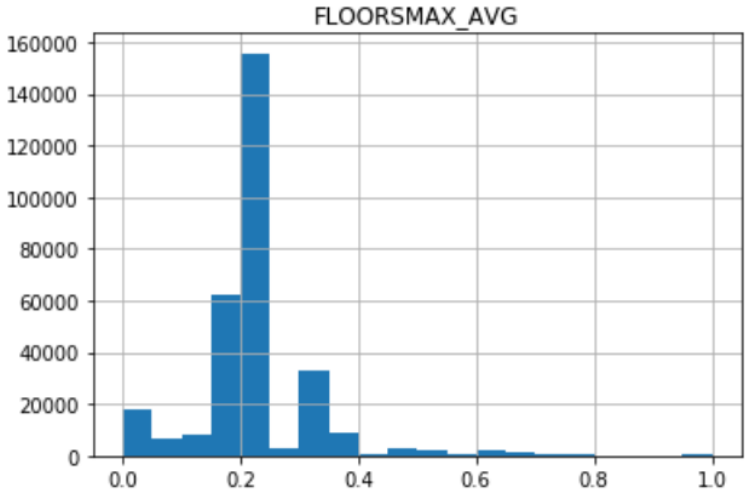


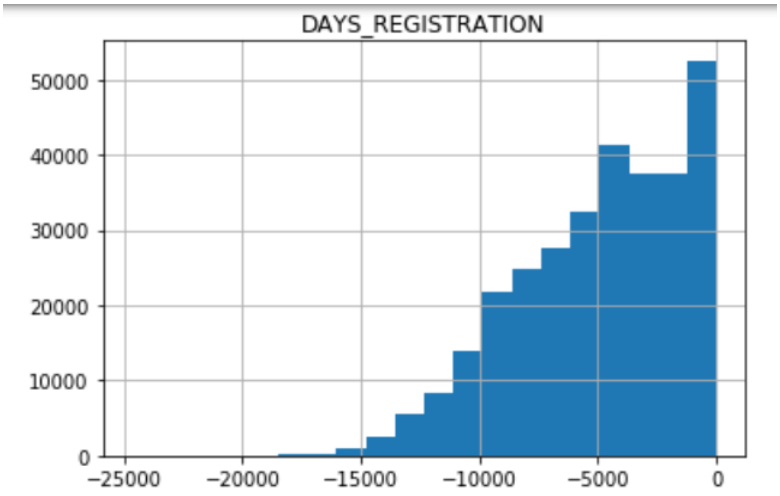

DAYS_BIRTH
20000
15000
10000
5000
0
−25000 −22500 −20000 −17500 −15000 −12500 −10000 −7500

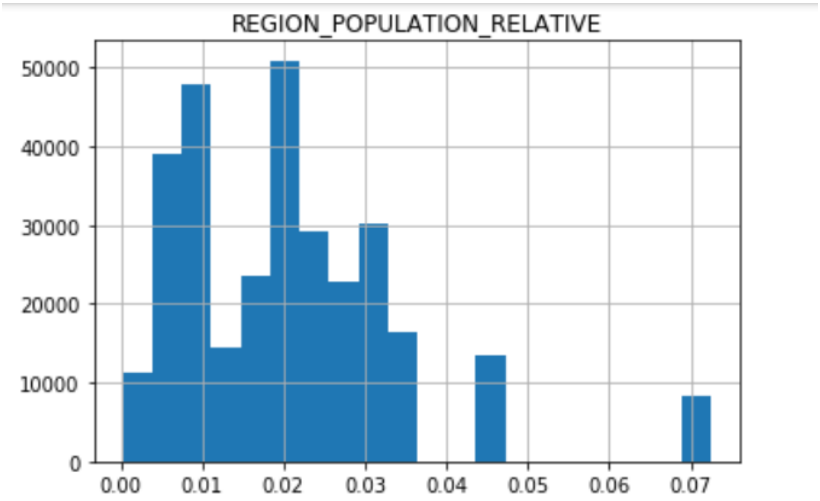

DAYS_LAST_PHONE_CHANGE
70000
60000
50000
40000
30000
20000
10000
0
−4000 −3000 −2000 −1000 0

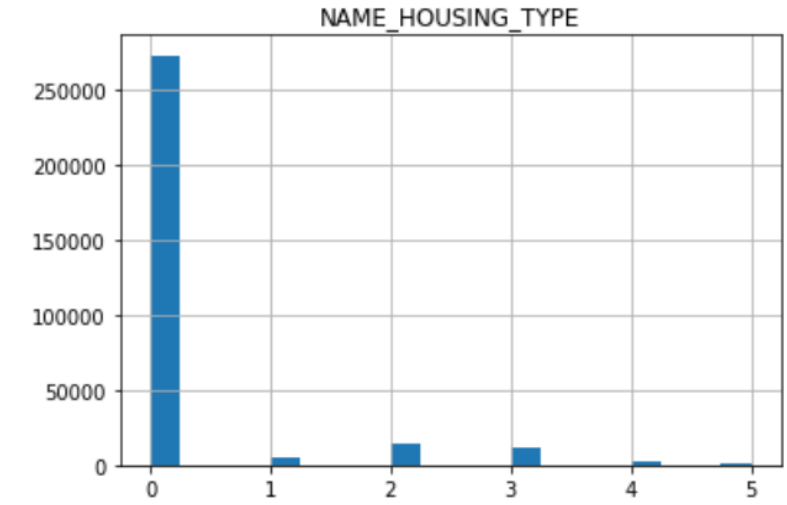

DAYS_ID_PUBLISH
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
−7000 −6000 −5000 −4000 −3000 −2000 −1000 0

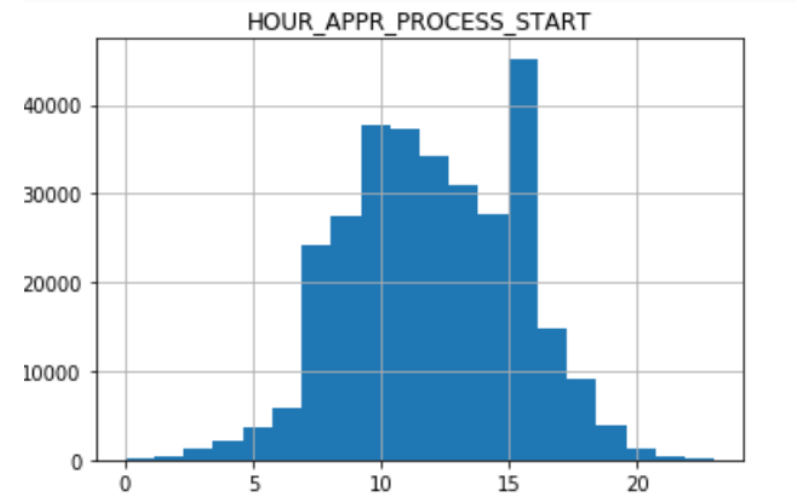

AMT_GOODS_PRICE
80000
70000
60000
50000
40000
30000
20000
10000
0


FLOORSMAX_AVG
160000
140000
120000
100000
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0


DAYS_REGISTRATION
50000
40000
30000
20000
10000
0
−25000
−20000
−15000
−10000
−5000
0


REGION_POPULATION_RELATIVE
50000
40000
30000
20000
10000
0
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07


NAME_HOUSING_TYPE
250000
200000
150000
100000
50000
0
0
1
2
3
4
5


HOUR_APPR_PROCESS_START
40000
30000
20000
10000
0
0
5
10
15
20

6. Classification

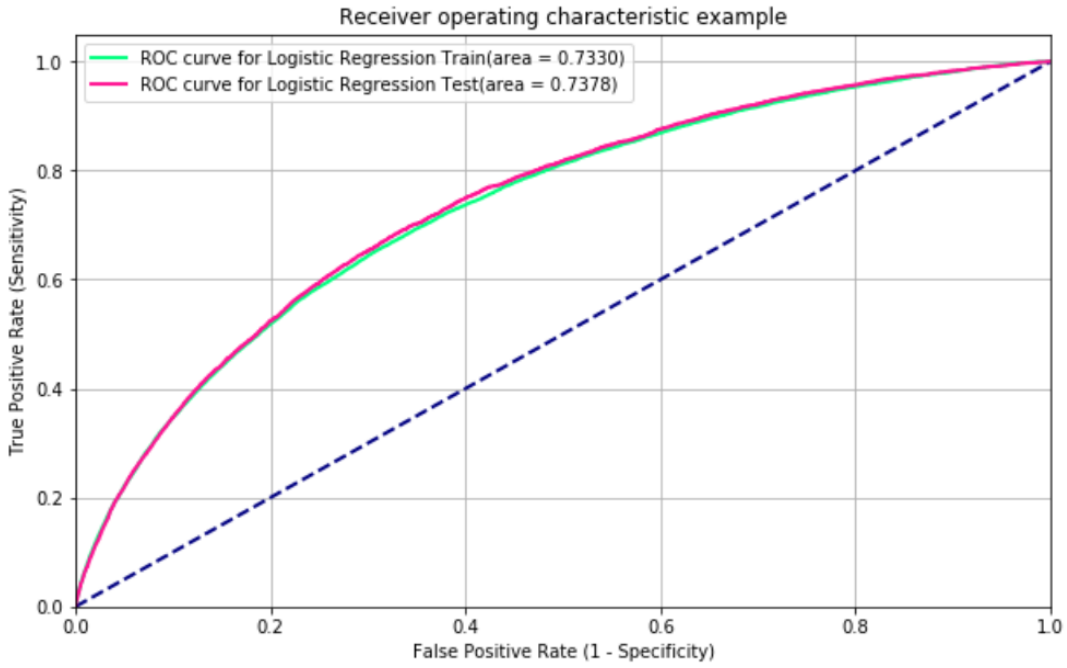
Different classification techniques have been implemented for this capstone project. The detail about each one is presented below. Before implementing each learner, the dataset split for training process I used train test split from sklearn.cross validation to create a test validation set that is 30% of the size of the total training set. I also used StandardScaler() to normalize the data.

6.1. Logistic Regression

Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. A downside of logistic regression is that it assumes all features are independent.
The threshold of 0.5 is used by default (for binary problems) to convert predicted probabilities into class predictions, The threshold can be adjusted to increase sensitivity or specificity.

```
Correct accuracy of the train set with Logistic Regression is: 91.9198 %
Correct accuracy of the test set with Logistic Regression is: 91.9352 %
```

Since Target is unbalance correct accuracy is not a right metric evaluation for this problem.
- AUC is useful as a single number summary of classifier performance.
- If you randomly chose one positive and one negative observation, AUC represents the likelihood that your classifier will assign a higher predicted probability to the positive observation.
- AUC is useful even when there is a high-class imbalance (unlike classification accuracy).




```
Total elapsed time is: 3.071  sec
Confusion Matrix is:
[[84758    62]
 [ 7378    55]]
```

```
Logistic Regression log_loss is: 2.7855
Average precision-recall score: 0.0835
AUC Train: 0.733, AUC Test: 0.738
```

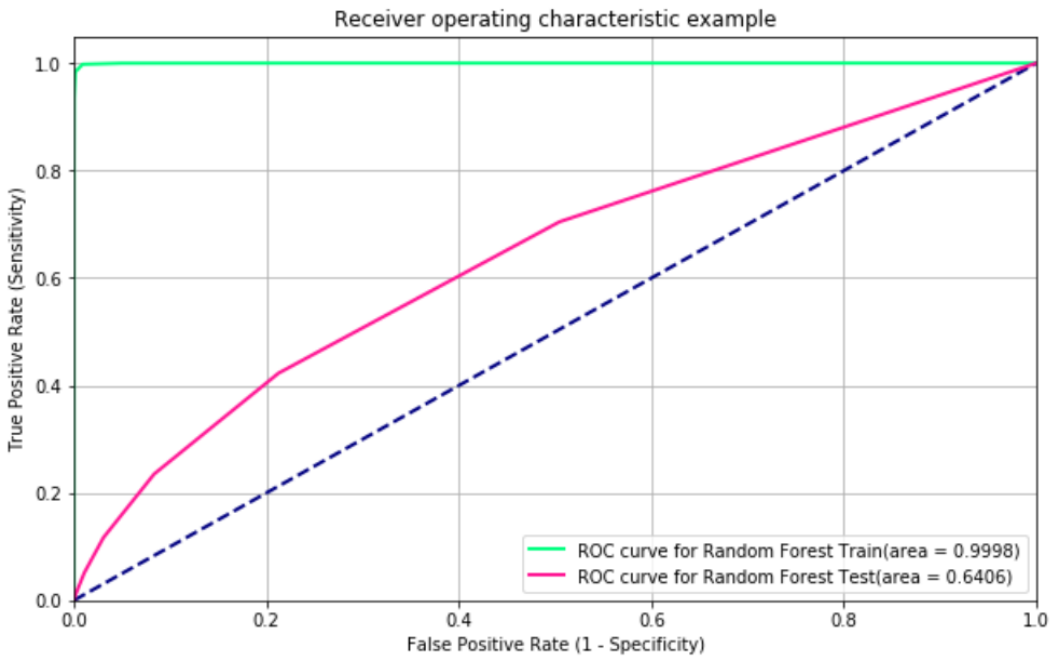
6.2. Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes or mean prediction of the individual trees.

Random Forest without hyperparameter tuning is overfitted in this section.

```
Correct accuracy of the train set with Random Forest Classifier is: 98.5622 %
Correct accuracy of the test set with Random Forest Classifier is: 91.765
 Confusion Matrix is:
[[84511   309]
 [ 7288   145]]

Total elapsed time is: 27.3902  sec
Random Forest log_loss is: 2.8443
Average precision-recall score: 0.0852
AUC Train: .999, AUC Test: 0.641
```



6.3. Random Forest with Hyperparameters

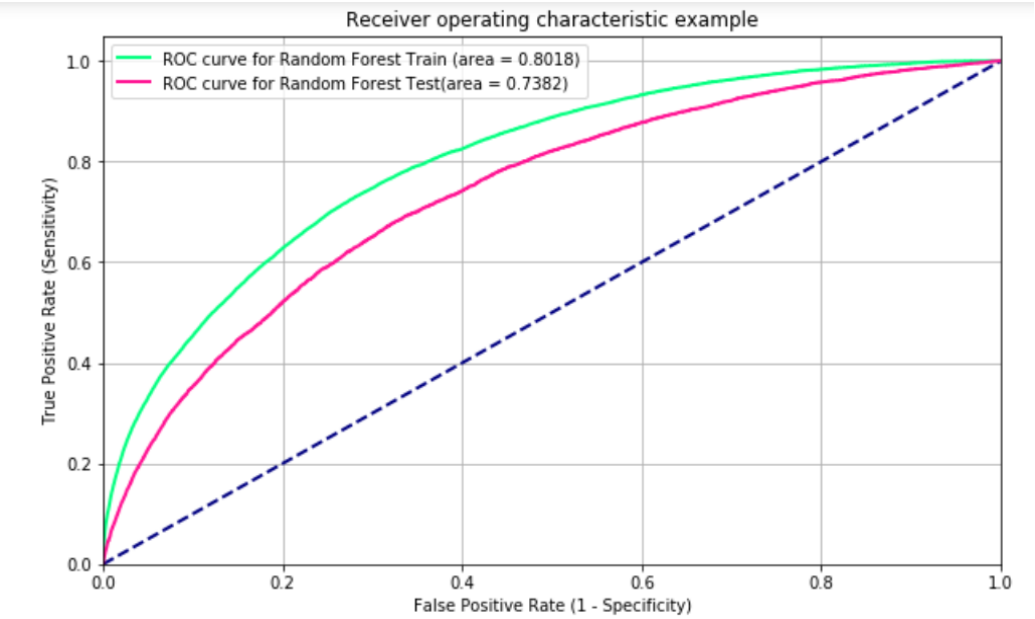
To overcome the overfitting problem of the previous section, I tuned the hyperparameters with RandomizedSearchCV.

The parameters to tune are:

- n_estimators = number of trees in the forest
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement)

Best Parameters after tuning are:

```
n_estimators=100,max_depth=10,min_samples_leaf=2
Total elapsed time is: 75.8383  sec
Correct accuracy of the train set with Random Forest Classifier is: 91.9393 %
Correct accuracy of the test set with Random Forest Classifier is: 91.945 %
Confusion Matrix is:
[[84820     0]
 [ 7431     2]]
Random Forest log_loss is: 2.7821
Average precision-recall score: 0.0808
AUC Train: 0.802 AUC Test: 0.738
```



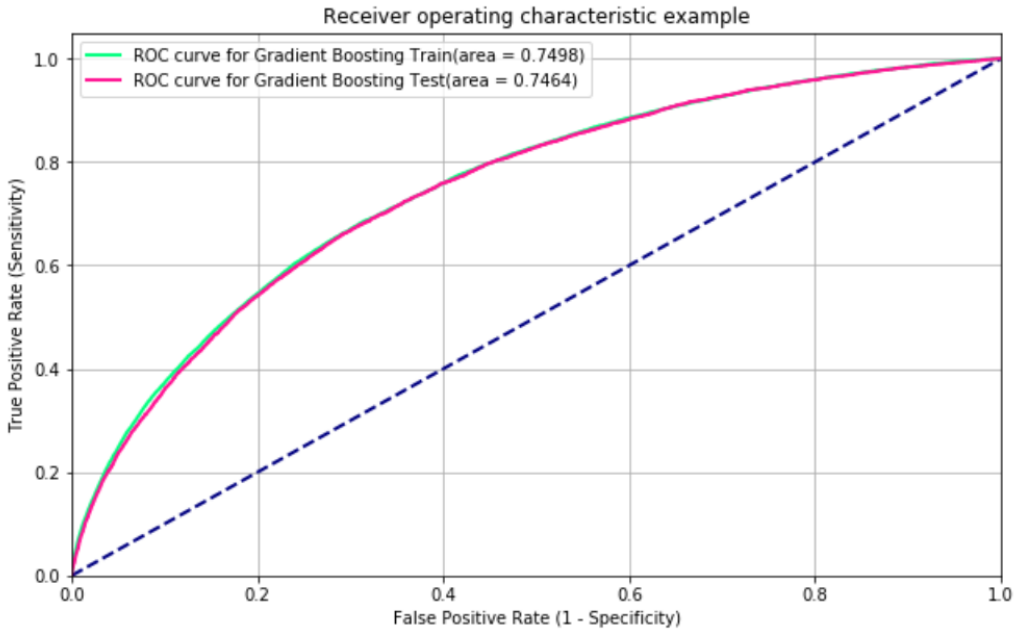
By tuning the hyperparameters, the Random forest is not anymore overfitted and the AUC is larger for the testing samples.

6.4. Gradient Boosting Classifier

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Boosting algorithms play a crucial role in dealing with bias variance trade-off. Unlike bagging algorithms, which only controls for high variance in a model, boosting controls both the aspects (bias & variance), and is considered to be more effective.

```
CA of the train set with Gradient Boosting Classifier is: 91.9667 %
CA of the test set with Gradient Boosting Classifier is: 91.9504 %
Confusion Matrix is:
[[84731    89]
 [ 7337    96]]
Total elapsed time is: 85.7874  sec
GradClassifier log_loss is: 2.7802
Average precision-recall score: 0.0862
AUC Train: 0.75 AUC Test: 0.746
```



Gradient boosting without hyperparameter tuning provides a good output however it is slower than other classifiers.

6. 4. Gradient Boosting Classifier with Hyperparameters

In this section the gradient boosting hyperparameters are tuned with GridSearchCV.

6.4.1 GridSearchCV

GridSearchCV is useful for automatically tuning hyperparameters. Although it can be more straightforward, and less computationally expensive, to manually tune a classier's hyperparameters one parameter at a time, this can sometimes lead to an algorithm getting stuck at a local performance maxima. Indeed, there are situations where one particular combination of hyperparameter values may be superior to a different combination, but the chances of discovering the superior combination by adjusting one parameter at a time could be very remote. GridSearchCV generates a grid of hyperparameter combinations after the user species ranges of discrete values for each hyperparameter that they wish to tune. GridSearchCV then methodically trains a given classier using each unique combination of hyperparameter values, and uses K-Fold cross validation and a scoring function specified by the user to compare the hyperparameter combinations and return the combination belonging to the classier that scored highest. I will use GridSearchCV with AdaBoost in order to nd an ideal trade off between AdaBoost's learning rate and n-estimators parameters. Unfortunately GridSearchCV can be computationally expensive to run if there are several hyperparameters that can be tuned. This is all the more true when hyperparameters have large ranges of possible values.

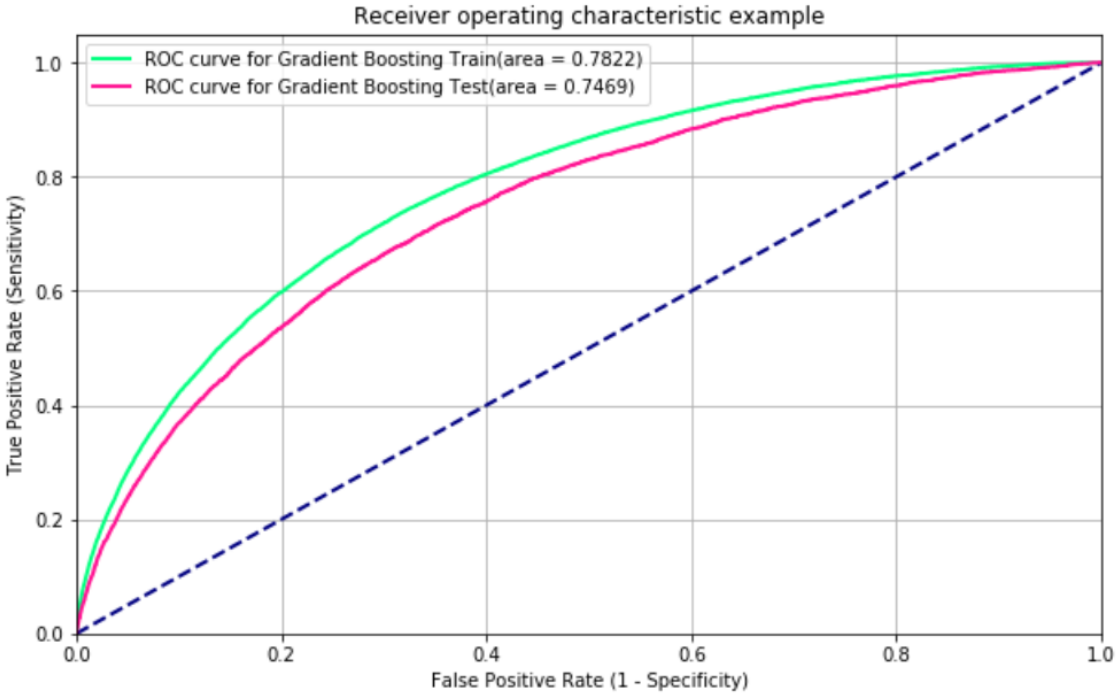
Best Parameters are:

learning_rate=0.1, n_estimators=60, max_depth=9,max_features='sqrt', subsample=0.8, random_state=0 ,min_samples_leaf=60, min_samples_split=1800

```
Correct accuracy of the train set with Gradient Boosting Classifier is:
91.9941 %
Correct accuracy of the test set with Gradient Boosting Classifier is:
91.9396 %
Confusion Matrix is:
[[84714   106]
 [ 7330   103]]
Total elapsed time is: 51.2964  sec
GradClassifier log_loss is: 2.784
Average precision-recall score: 0.0863
```

AUC Train: 0.782. AUC Test: 0.747

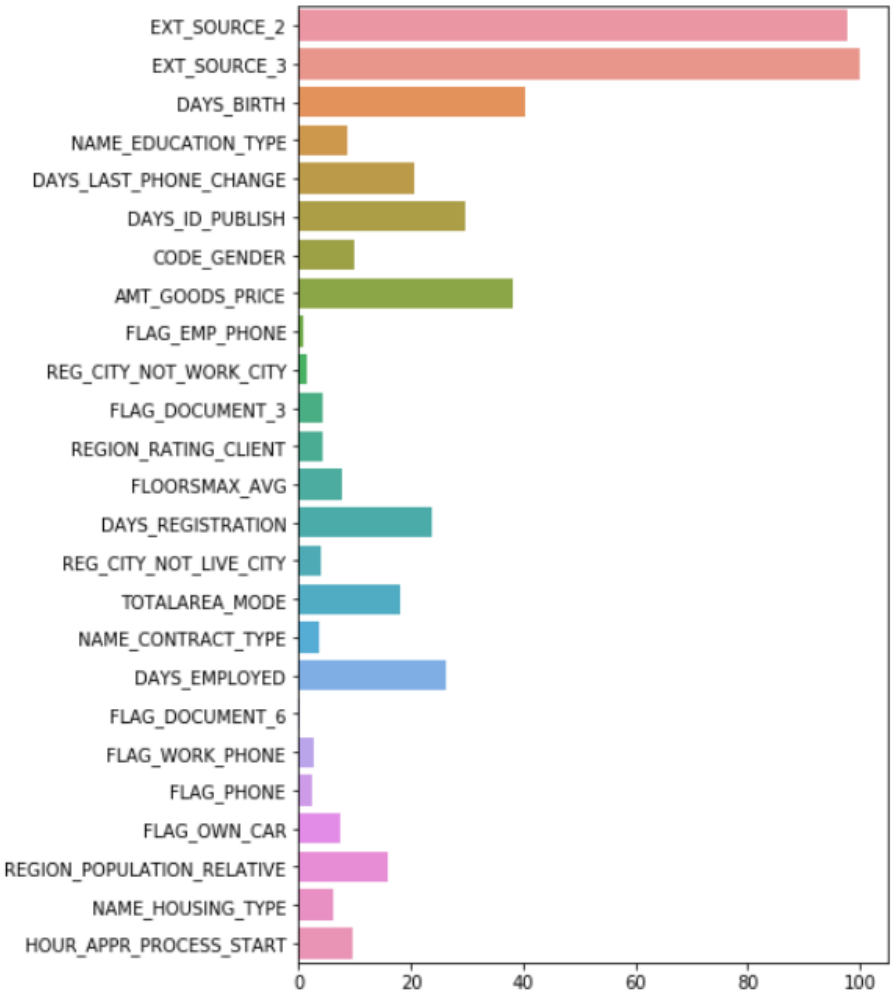
The test AUC improved by 0.0005 with hyperparameter tuning.


Receiver operating characteristic example
ROC curve for Gradient Boosting Train(area = 0.7822)
ROC curve for Gradient Boosting Test(area = 0.7469)
True Positive Rate (Sensitivity)
False Positive Rate (1 - Specificity)
0.0
0.2
0.4
0.6
0.8
1.0

6.5. Gradient Boosting with 15 Features

Since Gradient boosting was slow, I tried to see If I could reduce the demension more. So I use the gradient boosting for feature sorting.

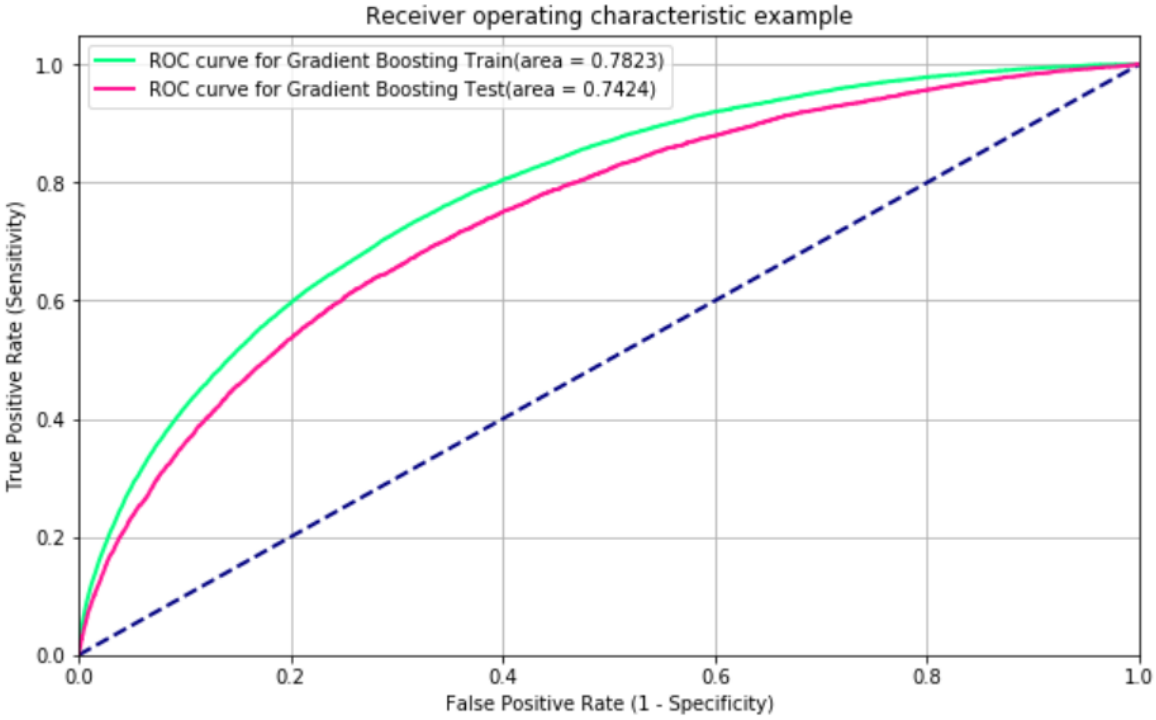
I selected the most 15 important features before training.



15 Selected Features are:

```
['EXT_SOURCE_3', 'EXT_SOURCE_2', 'DAYS_BIRTH', 'AMT_GOODS_PRICE',
 'DAYS_ID_PUBLISH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION',
  'DAYS_LAST_PHONE_CHANGE', 'TOTALAREA_MODE',
  'REGION_POPULATION_RELATIVE', 'CODE_GENDER', 'HOUR_APPR_PROCESS_START',
'NAME_EDUCATION_TYPE', 'FLOORSMAX_AVG', 'FLAG_OWN_CAR']
CA of the train set with Gradient Boosting Classifier is: 92.0053 %
CA of the test set with Gradient Boosting Classifier is: 91.9168 %
```

```
Confusion Matrix is:
[[84700   120]
 [ 7337    96]]
Total elapsed time is: 37.1995  sec
GradClassifier log_loss is: 2.7918
Average precision-recall score: 0.0853
AUC Train: 0.7823, AUC Test: 0.7424
```

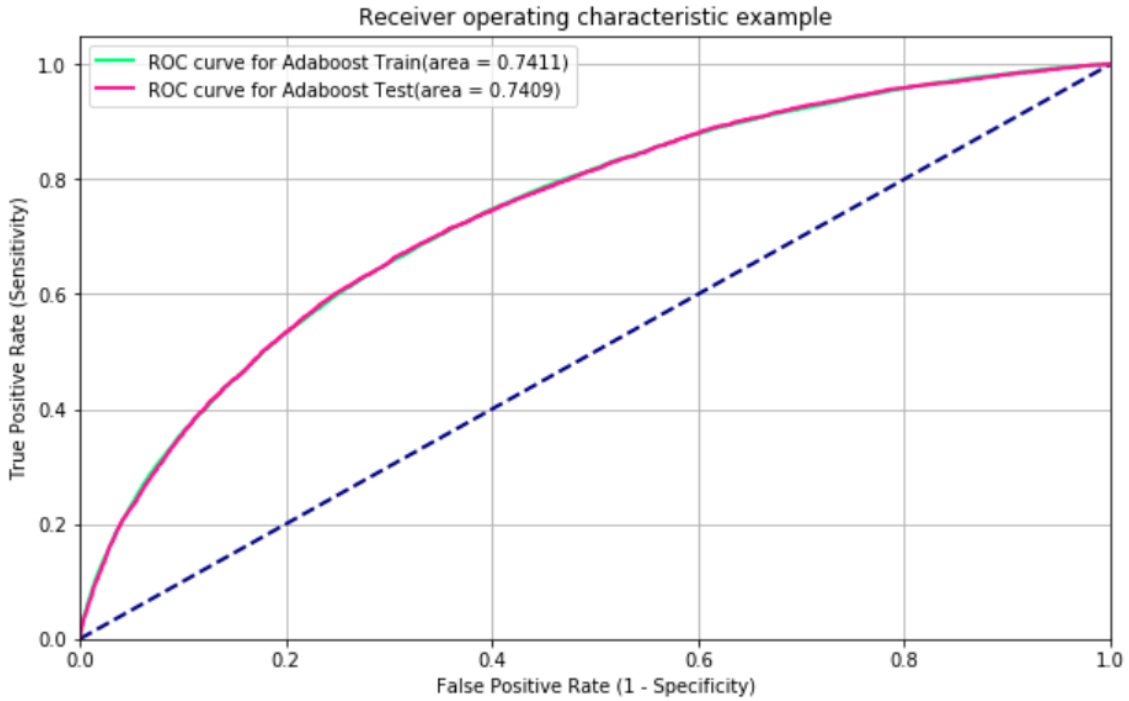


The AUC is lower than previous section, however the training time is shorter.

6.7. Adaboost Classifier

Adaboost, an adaptive learning algorithm, like Random Forest Classifier is another ensemble classifier. (Ensemble classifier are made up of multiple classifier algorithms and whose output is combined result of output of those classifier algorithms). Adaboost classifier combines weak classifier algorithm to form strong classifier. A single algorithm may classify the objects poorly. But if we combine multiple classifiers with selection of training set at every iteration and assigning right amount of weight in final voting, we can have good accuracy score for overall classifier.

```
CA of the train set with Adaboost Classifier is: 91.9235 %
CA of the test set with Adaboost Classifier is: 91.9623 %
Confusion Matrix is:
[[84709   111]
 [ 7304   129]]
Total elapsed time is: 37.7566  sec
GradClassifier log_loss is: 2.7761
Average precision-recall score: 0.0885
AUC Train: 0.7411. AUC Test: 0.7409
```

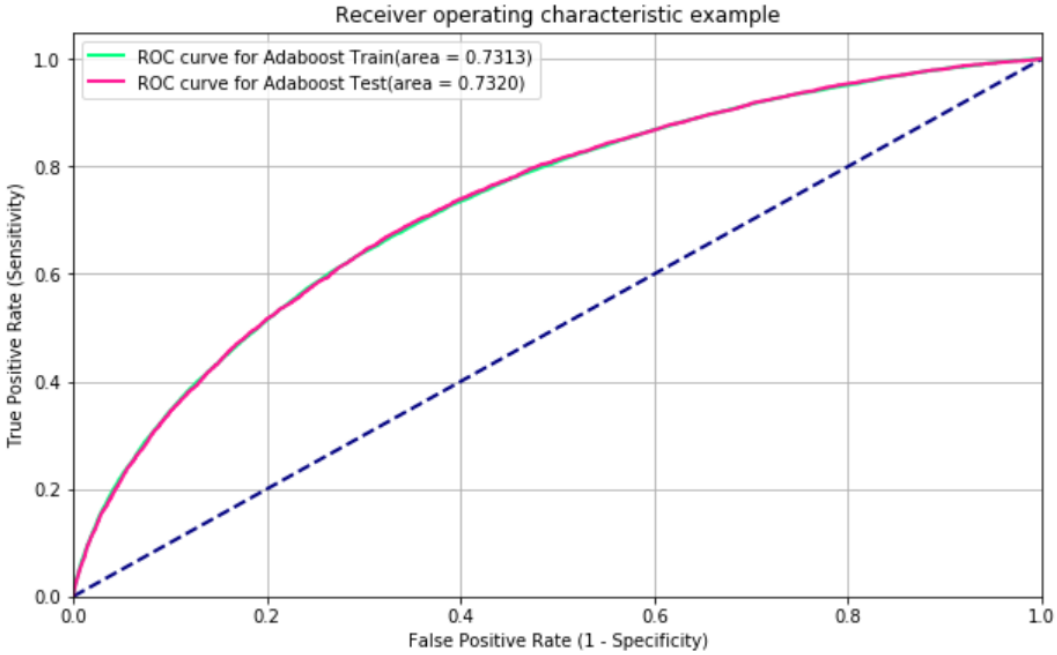


8. Adaboost Classifier with 15 Features

Adaboost classifier with most 15 important features from previous section:

```
Correct accuracy of the train set with Adaboost Classifier is: 91.9221 %
Correct accuracy of the test set with Adaboost Classifier is: 91.9352 %
```

```
Confusion Matrix is:
[[84703   117]
 [ 7323   110]]
Total elapsed time is: 40.7927  sec
GradClassifier log_loss is: 2.7855
Average precision-recall score: 0.0866
AUC Train: 0.731, AUC Test: 0.732
```



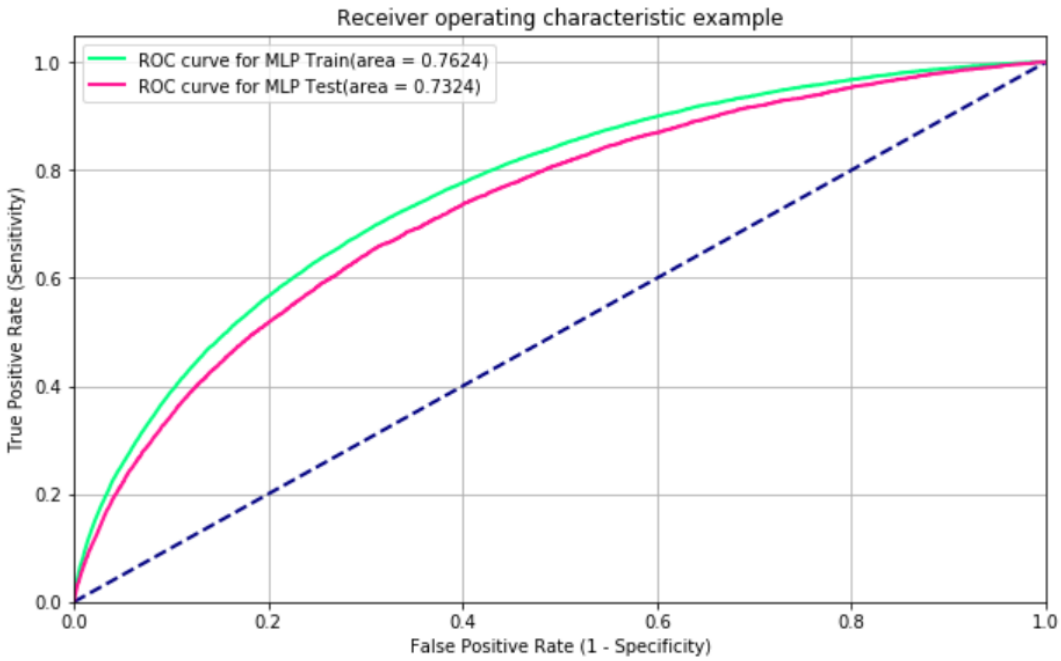
8. Multi-Layer Perceptron Classifier

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. A MLP consists of, at least, three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

Multi-layer perceptron classifiers build a neural-network comprised of several hidden layers that outputs a set of probability predictions for each point in the dataset. The predictions contain the probabilities that a point, or borrower in our case, belongs to each of the dataset's Target categories. Multilayer perceptron classifiers can learn non-linear models, and can support a variety of activation functions, such as `relu' or `tanh', for generating probability predictions. I wanted to use a multi-layer perceptron simply to see how its performance compared to that of a linear logistic regression classier when trained on this dataset. MLPs are sensitive to feature scaling and this is yet another reason I had to ensure that all my numerical features were scaled to the range [0,1].

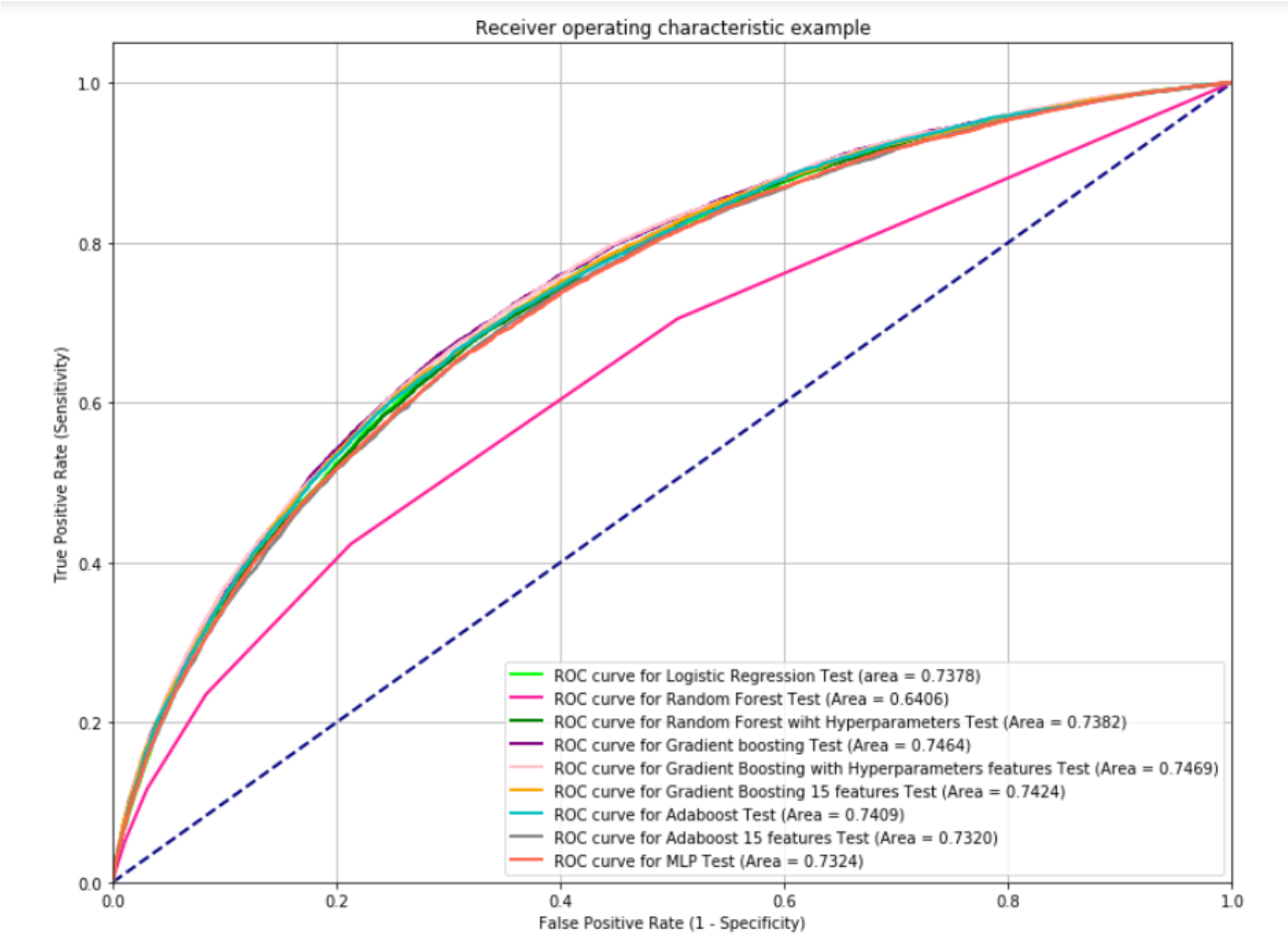
```
CA of the train set with Multi-Layer Perceptron Classifier is: 91.9709 %
CA of the test set with Multi-Layer Perceptron Classifier is: 91.9276 %
```

```
Confusion Matrix is:
[[84733    87]
 [ 7360    73]]
Total elapsed time is: 101.2572  sec
MultiLayer Perceptron log_loss is: 2.7881
Average precision-recall score: 0.0843
AUC Train: 0.762 AUC Test: 0.732
```



7. Conclusion

Below are plots of ROC curves for each classifier and feature set on which it was trained.



The table below shows the AUC for both the train and test set, the Log-Loss score, Precision-Recall score and the total training elapsed time of each classifier.

Classifier	AUC Train	AUC Test	Log Loss Score	Precision-Recall Score	Elapsed Time
Logistic Regression	0.733	0.738	8.553	0.0834	**3.02 sec**
Random Forest 1	0.999	0.641	2.844	0.0852	27.48 sec
Random Forest 2	0.802	0.738	2.782	0.0808	133.46 sec
Gradient Boosting 1	0.749	0.746	2.780	0.0862	136.04 sec
Gradient Boosting 2	0.782	**0.747**	2.784	0.0862	88.79 sec
Gradient Boosting 3	0.782	0.742	2.792	0.0853	48.16 sec
Adaboost 1	0.741	0.741	**2.776**	0.0885	53.21 sec
Adaboost 2	0.731	0.732	2.786	0.0866	20.96 sec
Multilayer Perceptron	0.762	0.732	2.788	0.0843	85.96 sec

The most challenging part of this project was understanding the dataset and its various features. I budgeted the majority of my time toward understanding what information each feature contained, how this information was distributed, and what sort of preprocessing techniques would need to be applied. Dealing with the dataset's overabundance of missing values, or `NaN' entries, proved to be trickiest part of the data preprocessing phase and since most machine learning algorithms aren't designed to overlook missing entries, I had to overcome this challenge.

I implemented several steps to pick the best features and reduce the dimensionality as much as possible. The steps were: removing base on percentage of missing values, The amount of variation in each column, Pairwise correlation among features, Correlation with the target, and The weight of evidence and information value

Several Classifier techniques have been trained and the Logistic regression was the fastest technique since it was the only linear technique, however the AUC of Gradient Boosting with hyperparameter tuning provides the best AUC test score. Random Forest with tuned parameters and Adaboost classifier were also working fine.

In future, I like to spend more time to understand the reason of existence of outliers of some features more. I also like to implement LightGBM classifier or other optimization techniques for tuning the hyperparameters of each ensemble technique.