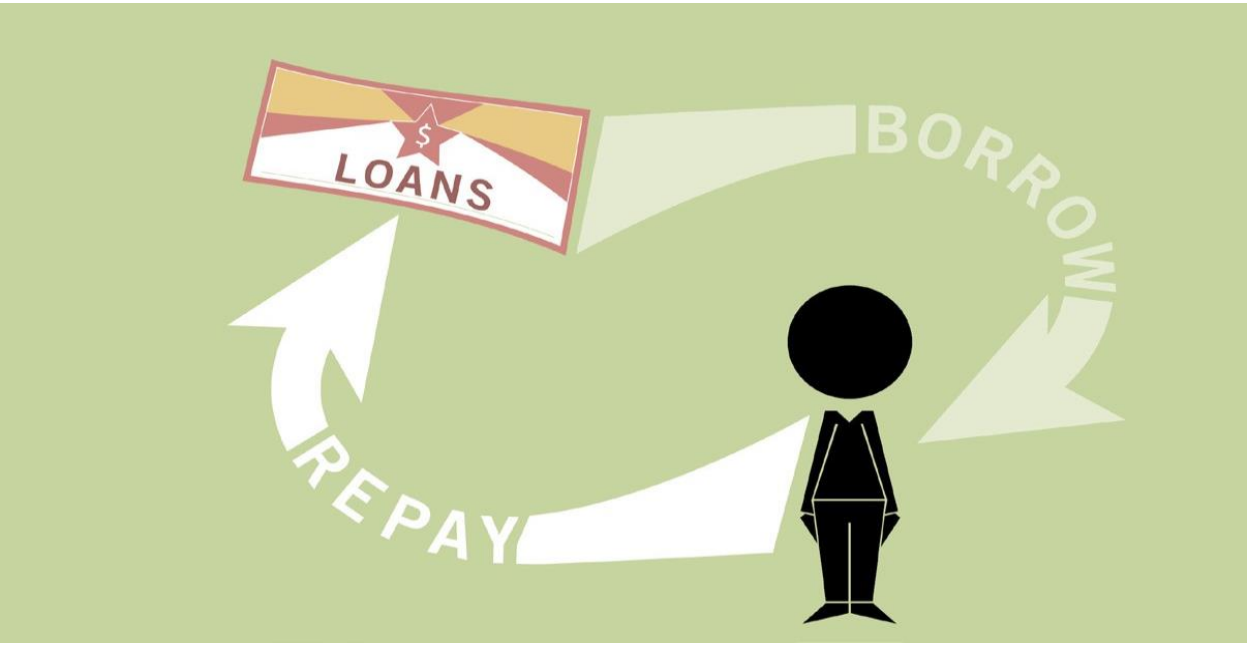
Home Credit – Default Risk



Link for demo : https://default-risk.herokuapp.com

By Jay Borkar
Instructor : Prof. Dr. Saed Sayad

Table of Contents

Overview

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders. Companies like 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 (e.g., including telco and transactional information) to predict their clients' repayment abilities. By using various Statistical, Machine Learning and Deep learning methods we unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

The objective of this project is to use historical loan application data to predict whether or not an applicant will be able to repay a loan. This is a standard supervised classification task:

- **Supervised**: The labels are included in the training data and the goal is to train a model to learn to predict the labels from the features
- **Classification**: The label is a binary variable, 0 (will repay loan on time), 1 (will have difficulty repaying loan)

In this project, the goals achieved are :

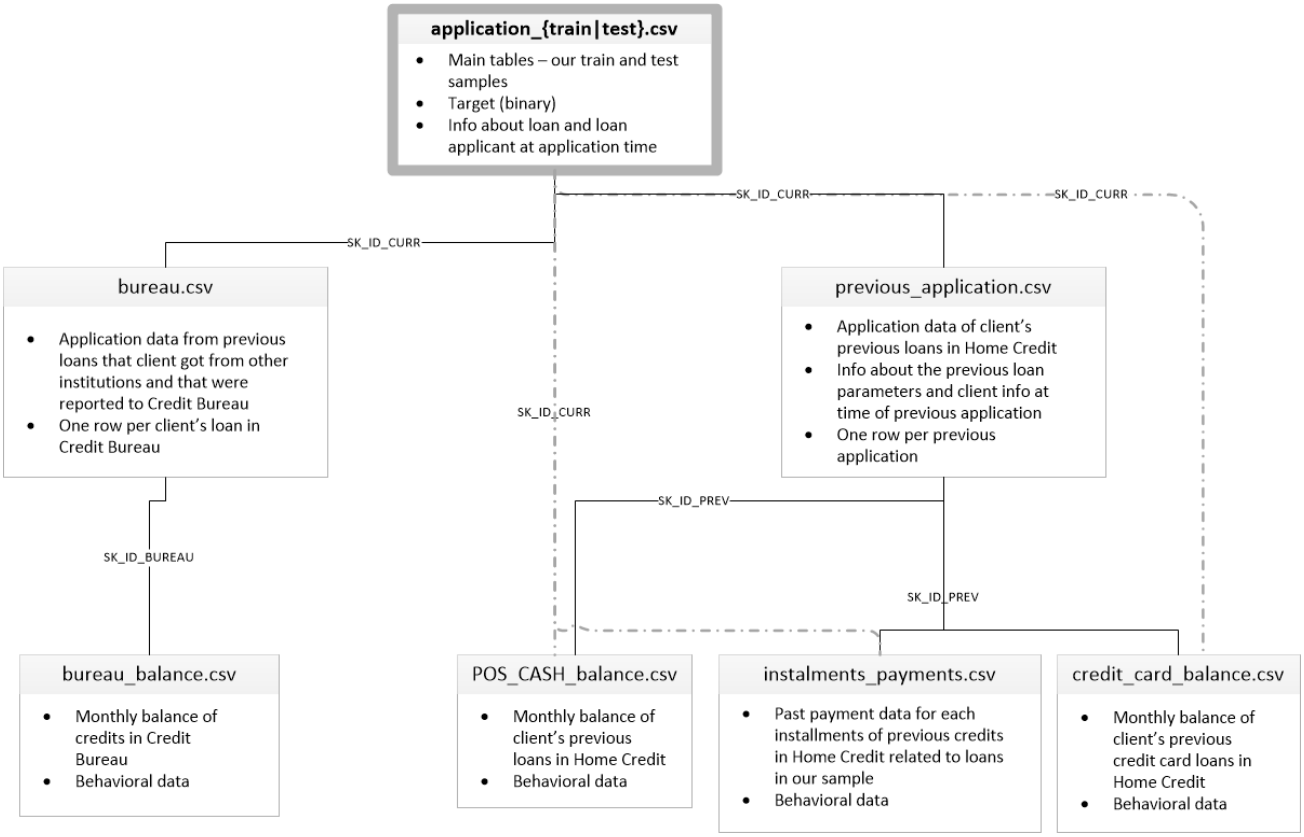
- **Data Exploration** routines are designed and implemented to do Statistical analysis and Visualization.
- **Classification models** such as Naïve Bayes, Logistic Regression, Support Vector machine (SVM), Decision Tree, Random Forest, Gradient Boosting Machine (GBM) and Deep Learning are built to predict whether or not an applicant will be able to repay a loan.
- **Evaluated Classification models** by Accuracy, Confusion Matrix, Precision, Recall, True Negative Rate (TNR), False Discovery Rate (FDR), Gain Chart, Lift Chart, K-S Chart, ROC – AUC chart.
- Deployed the Final solution as a Web application (Restful API).

Data sources

The data is provided by Home Credit, a service dedicated to provided lines of credit (loans) to the unbanked population. Predicting whether or not a client will repay a loan or have difficulty is a critical business need, and Home Credit wants to unlock the full potential of their data to see what sort of machine learning/deep learning models can be develop to help them in this task.

There are 7 different sources of data:

- **application_train/application_test**: the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR which is loan id. The training application data comes with the TARGET indicating 0: the loan was repaid or 1: the loan was not repaid.
- **bureau**: data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance**: monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application**: previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE**: monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- **credit_card_balance**: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment**: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.



Data Exploration

Data Exploration is an open-ended process where we calculate statistics and make figures to find trends, anomalies, patterns, or relationships within the data. The goal of Data Exploration is to learn what our data can tell us. It generally starts out with a high level overview, then narrows in to specific areas as we find intriguing areas of the data. The findings may be interesting in their own right, or they can be used to inform our modeling choices, such as by helping us decide which features to use.

First, we find the size of the 7 tables which are stored in MySQL database, then we find number of columns which have numeric and categorical values, int64 and float64 are numeric variables (which can be either discrete or continuous) and object columns contain strings and are categorical features. In application_train data, there are 106 numeric variables (columns) and 16 categorical variables.

```
Size of application_train data (307511, 122)
Size of bureau data (1716428, 17)
Size of bureau_balance data (27299925, 3)
Size of previous_application data (1670214, 37)
Size of POS_CASH_balance data (10001358, 8)
Size of installments_payments data (13605401, 8)
Size of credit_card_balance data (3840312, 23)
```

```
app_train.dtypes.value_counts()
```

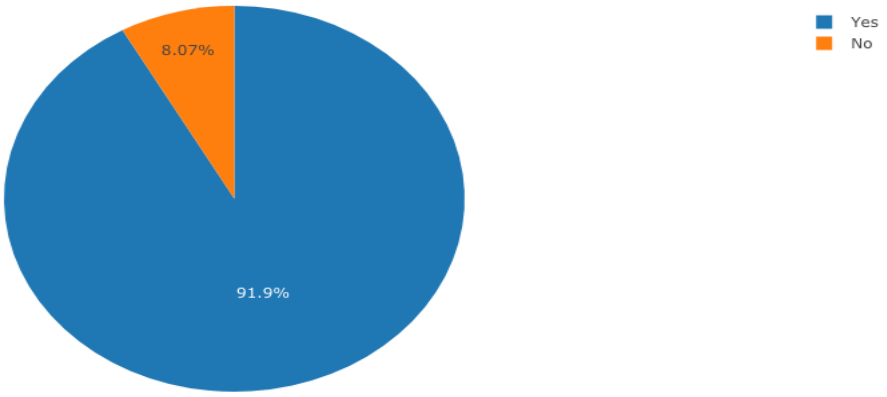
```
float64    65
int64      41
object     16
dtype: int64
```

Examining the Distribution of the Target Column

The target is what we are asked to predict: either a 0 for the loan was repaid on time, or a 1 indicating the client had payment difficulties. We first examine the number of loans falling into each category and plot the pie-chart for count percent.

```
app_train['TARGET'].value_counts()
```

```
0    282686
1     24825
Name: TARGET, dtype: int64
```



There are 282,686 loans that were repaid on time and 24,825 loans that were not repaid. From the count and the plot, we see this is an imbalanced class problem. There are far more loans that were repaid on time than loans that were not repaid.

Examining missing values

We look at the number and percentage of missing values in each column. In the application_train data, there are 122 columns and 67 columns of them have missing values. Below image shows the output along with top 10 columns that have missing values with the percent of missing values.

```
Your selected dataframe has 122 columns.
There are 67 columns that have missing values.
```

	Missing Values	% of Total Values
COMMONAREA_MEDI	214865	69.9
COMMONAREA_AVG	214865	69.9
COMMONAREA_MODE	214865	69.9
NONLIVINGAPARTMENTS_MEDI	213514	69.4
NONLIVINGAPARTMENTS_MODE	213514	69.4
NONLIVINGAPARTMENTS_AVG	213514	69.4
FONDKAPREMONT_MODE	210295	68.4
LIVINGAPARTMENTS_MODE	210199	68.4
LIVINGAPARTMENTS_MEDI	210199	68.4
LIVINGAPARTMENTS_AVG	210199	68.4

We fill in these missing values (known as imputation). For numeric variables, we impute the missing values with the median value in each column and for categorical variables, we impute with the most frequent category.

Encoding Categorical Variables

We need to deal with categorical variables as not all machine learning models can deal with categorical variables. Therefore, we have to find a way to encode (represent) these variables as numbers before handing them off to the model. There are two main ways to carry out this process:

- **Label encoding** : assign each unique category in a categorical variable with an integer. No new columns are created.
- **One-hot encoding**: create a new column for each unique category in a categorical variable. Each observation receives a 1 in the column for its corresponding category and a 0 in all other new columns.

The problem with label encoding is that it gives the categories an arbitrary ordering. The value assigned to each of the categories is random and does not reflect any inherent aspect of the category.

We use Label Encoding for any categorical variables with only 2 categories and One-Hot Encoding for any categorical variables with more than 2 categories.

Most Valuable Plot (MVP) – Combination Charts

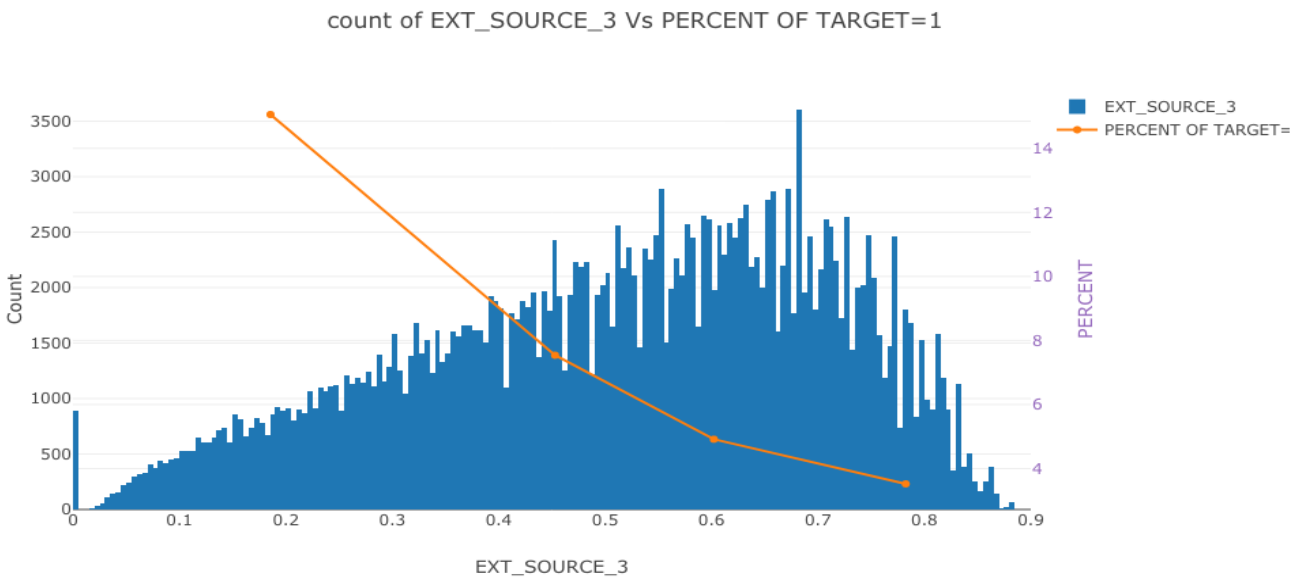
We do statistical analysis and visualization for both numeric and categorical variables. We generate descriptive statistics that summarize the central tendency, dispersion and shape of a data's distribution.

For numeric variables, we plot histograms to show the distribution of the numerical variable with one y-axis which is for count of the values in the numeric variable and also plot line graph with another y-axis which for the percent of Target=1 (will have difficulty repaying loan).

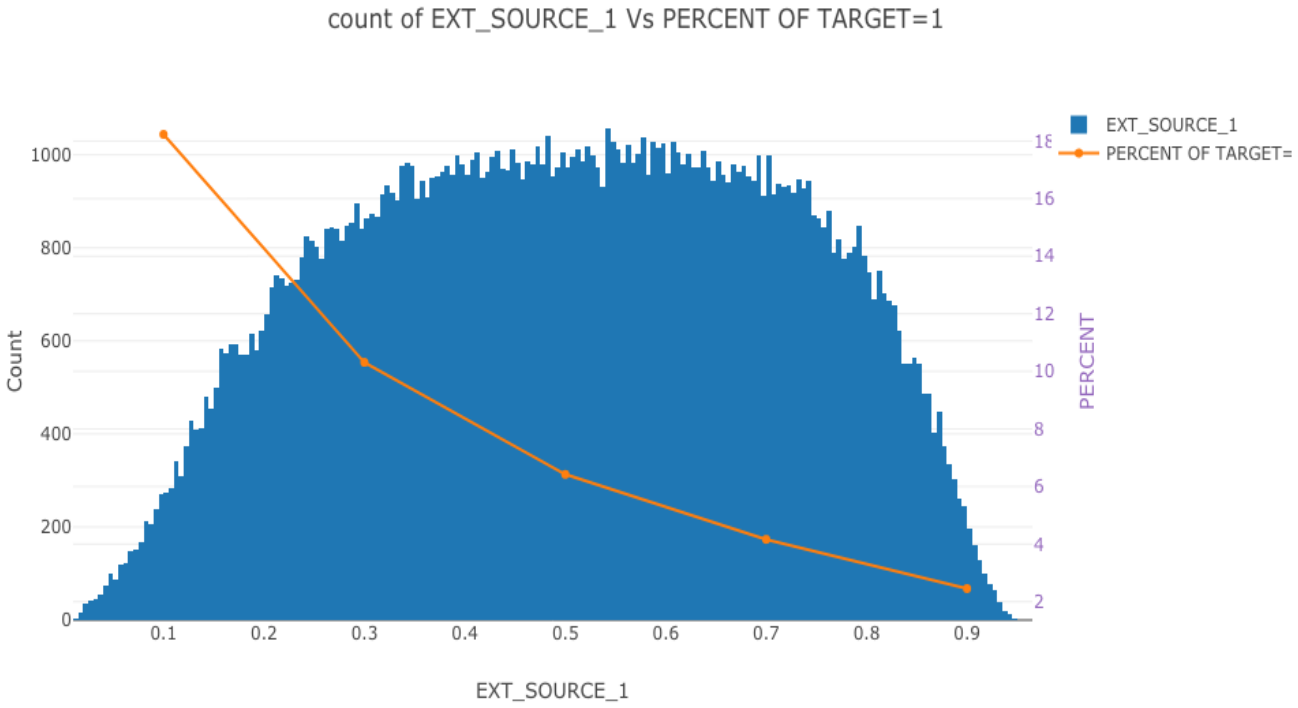
For categorical variables, we plot bar chart to show the distribution of the categorical variable and sort in decreasing order with one y-axis which is for count of the values for the categories and also plot line graph with another y-axis which for the percent of Target=1 (will have difficulty repaying loan).

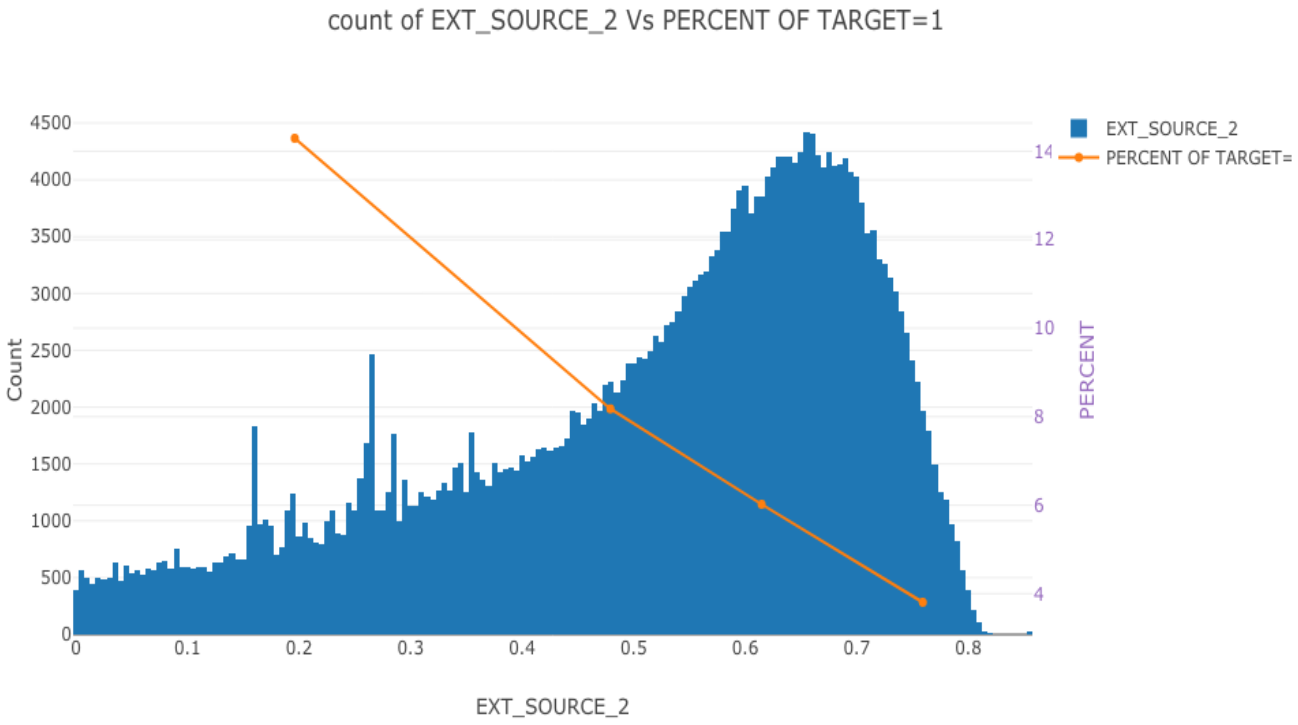
This plot is called the Most Valuable Plot (MVP), also called Combination Chart. The combination chart is the best visualization method to demonstrate the predictability power of a predictor (X-axis) against a target (Y-axis). They show us how the variable affecting the percent of Target=1, thereby demonstrate the predictability power of a variable for our predictive modeling. From the visual, we can say that as the count of the variable is increasing/decreasing, the percent of Target=1 is decreasing or increasing depending on the steepness of the line, so the loan are repaid more or less depending the percent decrease or increase respectively.

Below Plot shows that as the count of numeric variable EXT_SOURCE_3 is increasing, the percent of Target=1 is decreasing, so with the increasing in EXT_SOURCE_3, more loans are being repaid. EXT_SOURCE_3 can be a potential feature for our predictive modeling

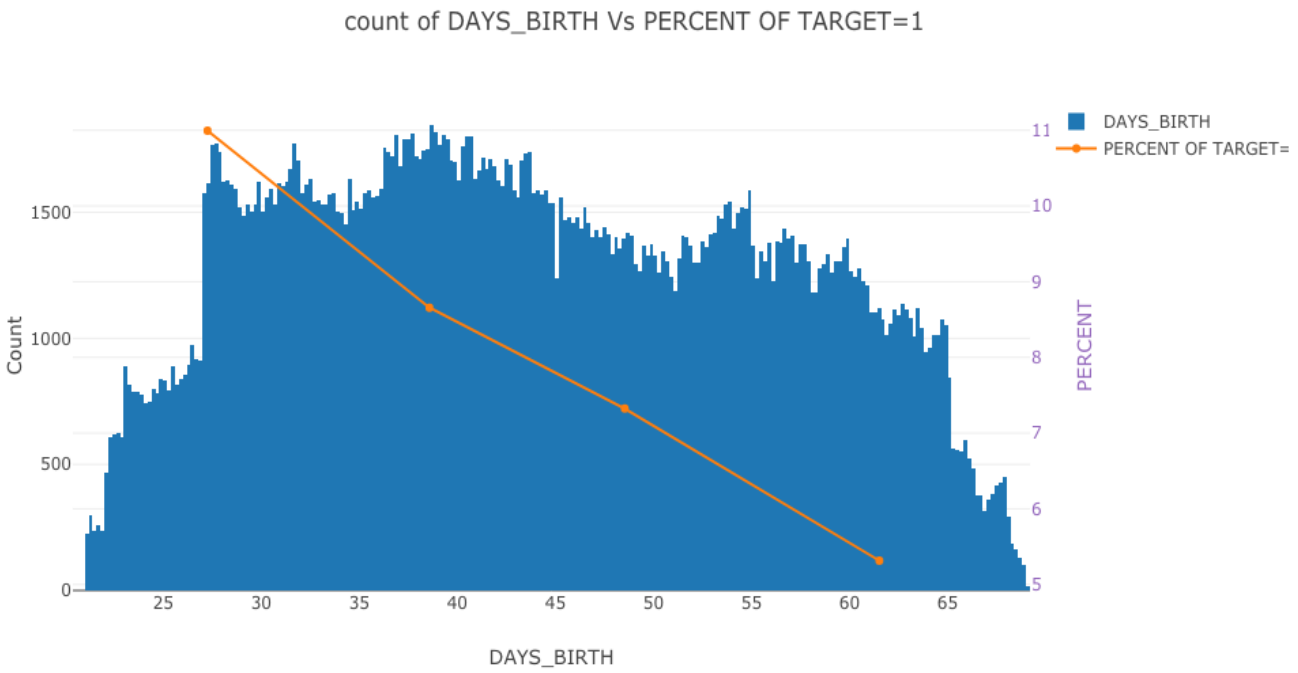


We can say the same for EXT_SOURCE_1 and EXT_SOURCE_2

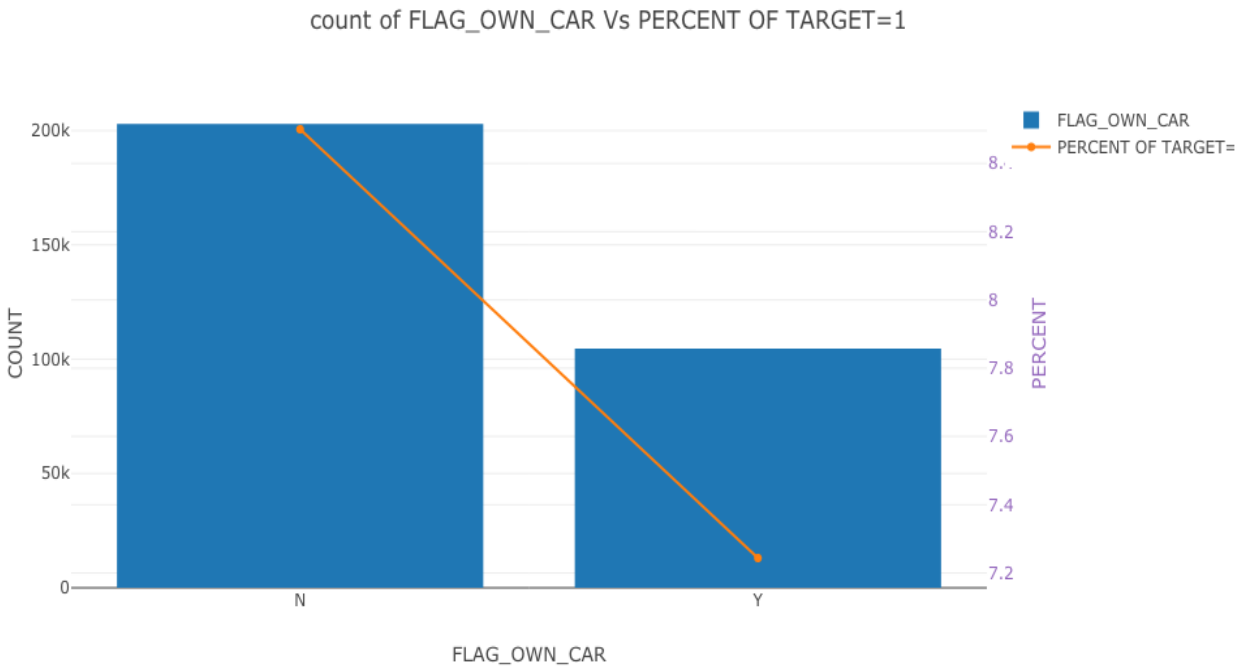




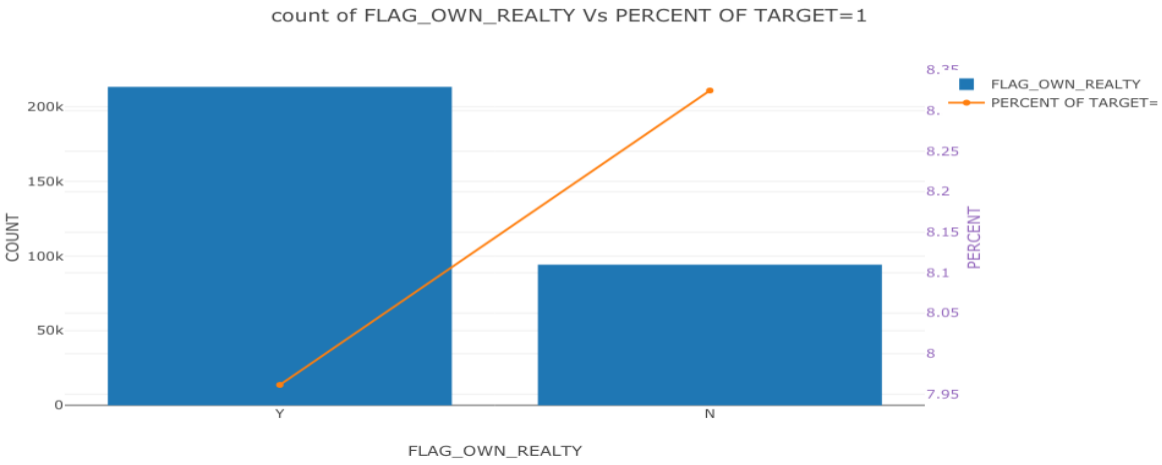
Below plot shows as the DAYS_BIRTH (Age) is decreasing, the percent of Target=1 is decreasing, so with the decrease in DAYS_BIRTH (Age), more loans are being repaid.

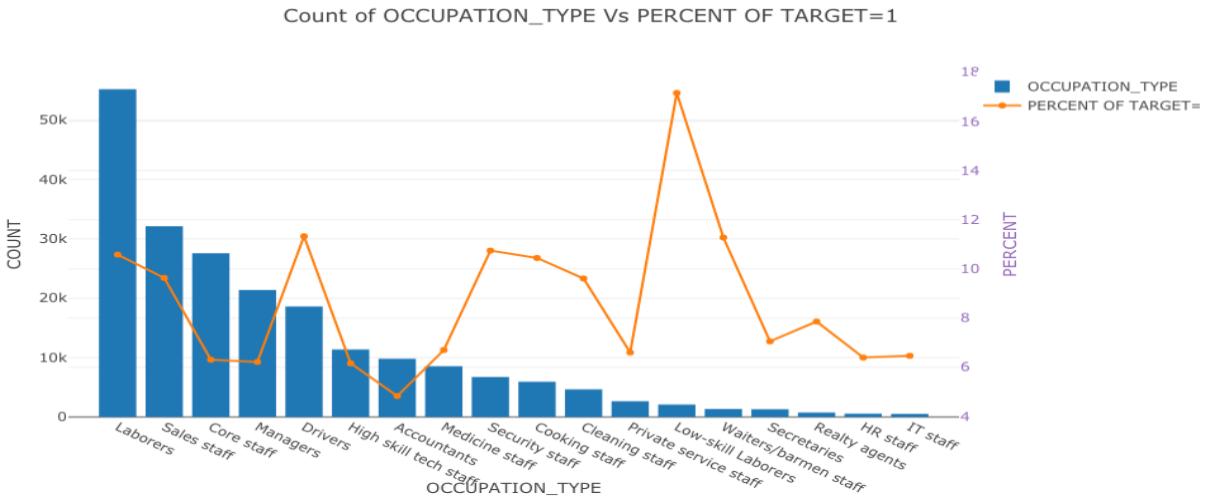


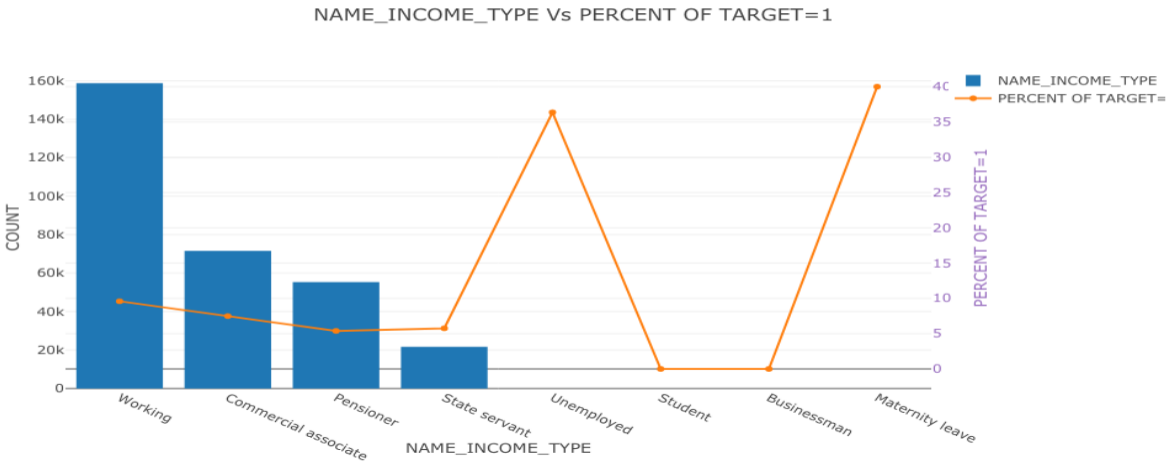
Below plot shows as the FLAG_OWN_CAR is decreasing, the percent of Target=1 is decreasing, so with the decrease in FLAG_OWN_CAR , more loans are being repaid.



Below plot shows as the FLAG_OWN_REALTY is decreasing, the percent of Target=1 is increasing, so with the decrease in FLAG_REALTY, less loans are being repaid.







Data Modeling

We use the following Machine learning algorithms and built Classification models for our supervised classification task.

1) Naïve Bayes
2) Logistic Regression
3) Decision Tree
4) Random Forest
5) Gradient Boosting Machine (GBM)
6) Deep Learning

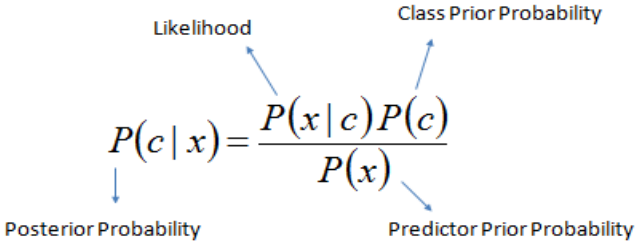
Out of the above, the first four are Baseline models and last three are improved models.

Naïve Bayes

The Naive Bayesian classifier is based on Bayes' theorem with the independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.

Algorithm

Bayes theorem provides a way of calculating the posterior probability, $P(c|x)$, from $P(c)$, $P(x)$, and $P(x|c)$. Naive Bayes classifier assume that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.



$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- $P(c|x)$ is the posterior probability of *class* (*target*) given *predictor* (*attribute*
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of *predictor* given *class*.
- $P(x)$ is the prior probability of *predictor*.

Naive Bayesian includes all predictors using Bayes' rule and the independence assumptions between predictors. The posterior probability can be calculated by first, constructing a frequency table for each attribute against the target. Then, transforming the frequency tables to likelihood tables and finally use the Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

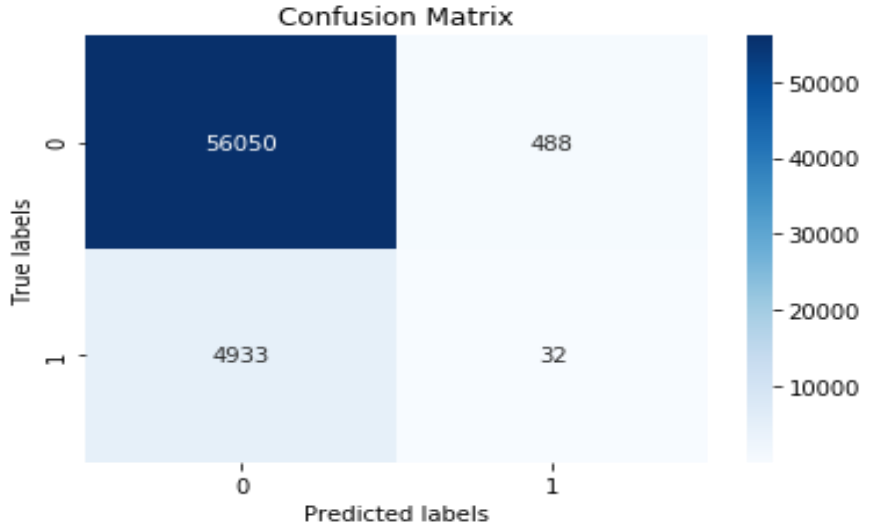
Model Evaluation

Confusion Matrix

A confusion matrix shows the number of correct and incorrect predictions made by the classification model compared to the actual outcomes (target value) in the data. The matrix is NxN, where N is the number of target values (classes). Performance of such models is commonly evaluated using the data in the matrix. The following table displays a 2x2 confusion matrix for two classes (Positive and Negative).

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	a	b	*Positive Predictive Value*	a/(a+b)
	Negative	c	d	*Negative Predictive Value*	d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)	
		a/(a+c)	d/(b+d)		

- **Accuracy** : the proportion of the total number of predictions that were correct.
- **Positive Predictive Value** or **Precision** : the proportion of positive cases that were correctly identified.
- **Negative Predictive Value** : the proportion of negative cases that were correctly identified.
- **Sensitivity** or **Recall :** the proportion of actual positive cases which are correctly identified.
- **Specificity** : the proportion of actual negative cases which are correctly identified.



TN = 56050, FP = 488 , FN= 4933, TP = 32

Accuracy = (TP+TN) / (TP+TN+FP+FN)	0.911857958148383
Sensitivity, Recall or TPR = TP / (TP+FN)	0.006445115810674723
Specificity or TNR = TN / (TN+FP)	0.9913686370228872
Precision or Positive Predictive Value (PPV) = TP/ (TP+FP)	0.06153846153846154
Negative Predictive Value (NPV) = TN/(TN+FN)	0.9191086040371907
False Discovery Rate (FDR) = 1-Precision	0.9384615384615385

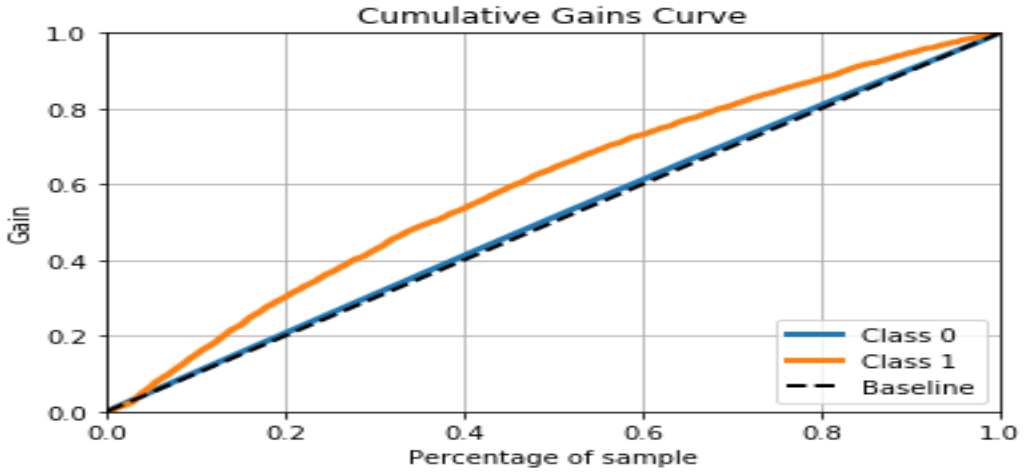
F1 score (harmonic mean of precision and sensitivity(recall))	0.011668185961713765

Gain and Lift Chart

Gain or lift is a measure of the effectiveness of a classification model calculated as the ratio between the results obtained with and without the model. Gain and lift charts are visual aids for evaluating performance of classification models. However, in contrast to the confusion matrix that evaluates models on the whole population gain or lift chart evaluates model performance in a portion of the population.

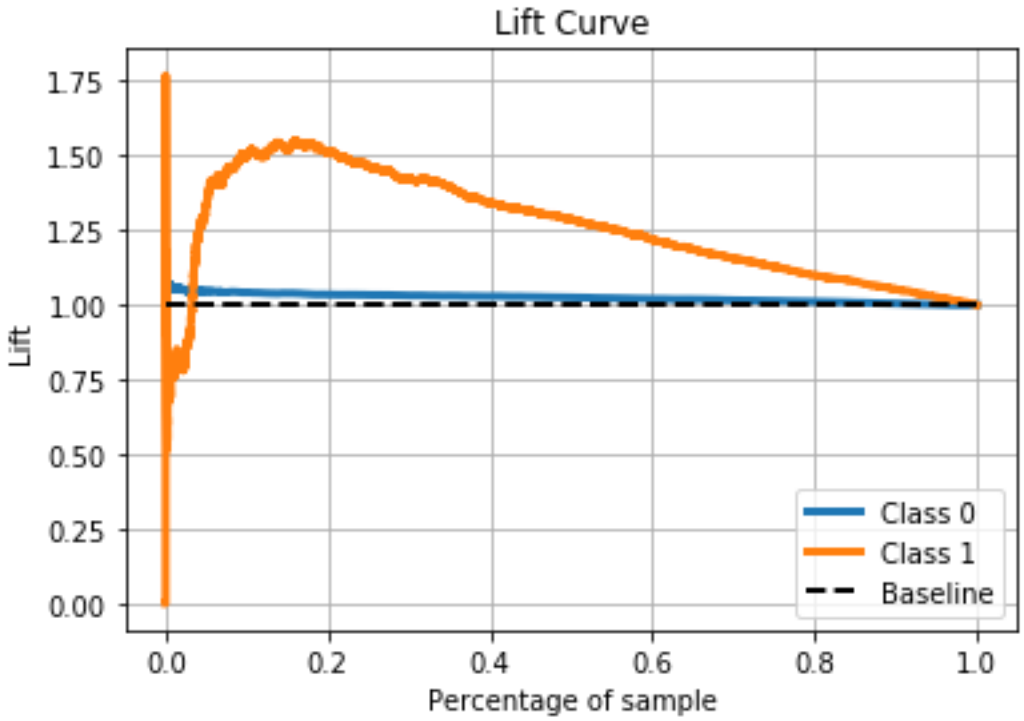
Cumulative Gains Chart

- The *y*-axis shows the percentage of positive responses. This is a percentage of the total possible positive responses.
- The *x*-axis shows the percentage of customers contacted.
- **Baseline (overall response rate):** If we contact *X*% of customers then we will receive *X*% of the total positive responses.
- **Lift Curve:** Using the predictions of the response model, calculate the percentage of positive responses for the percent of customers contacted and map these points to create the lift curve.



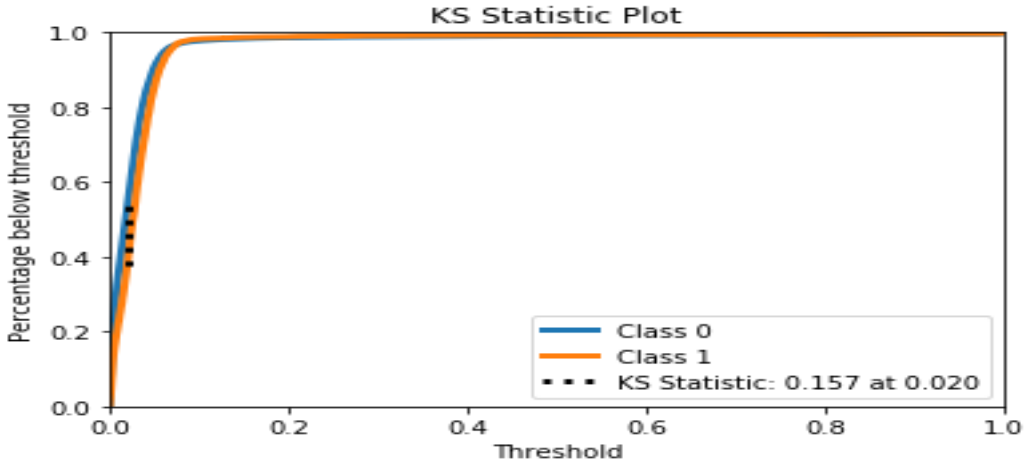
Lift Chart

The lift chart shows how much more likely we are to receive positive responses than if we contact a random sample of customers. For example, by contacting only 10% of customers based on the predictive model we will reach 3 times as many respondents, as if we use no model. It shows the actual lift.



K-S Chart

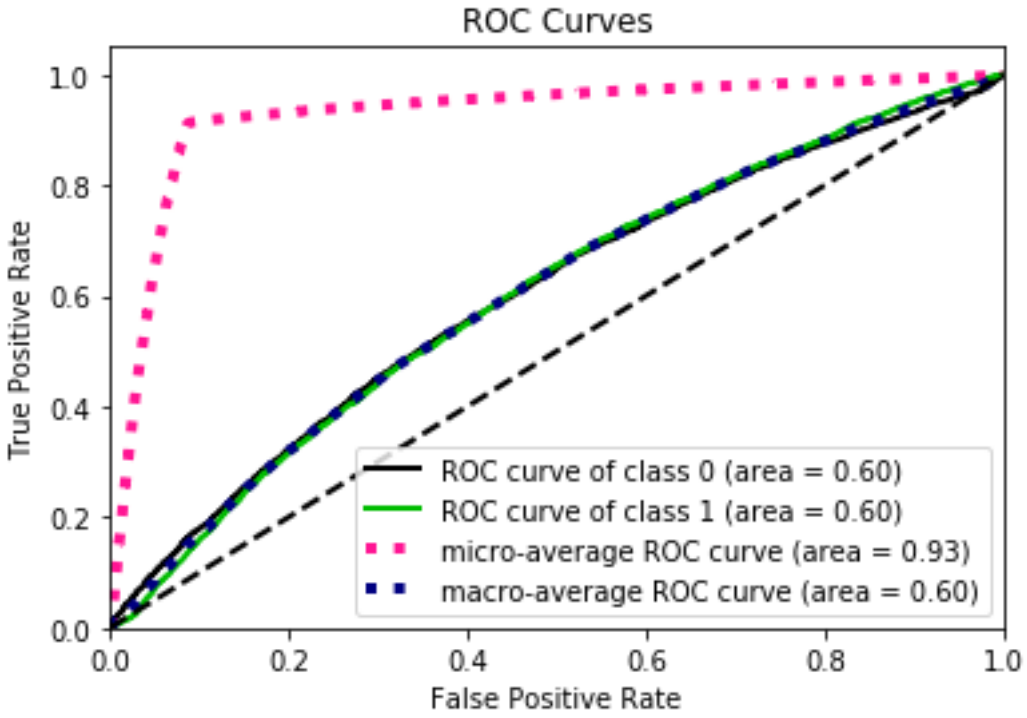
K-S or Kolmogorov-Smirnov chart measures performance of classification models. More accurately, K-S is a measure of the degree of separation between the positive and negative distributions. The K-S is 100 if the scores partition the population into two separate groups in which one group contains all the positives and the other all the negatives. On the other hand, If the model cannot differentiate between positives and negatives, then it is as if the model selects cases randomly from the population. The K-S would be 0. In most classification models the K-S will fall between 0 and 100, and that the higher the value the better the model is at separating the positive from negative cases.



ROC Chart

The ROC chart is similar to the gain or lift charts in that they provide a means of comparison between classification models. The ROC chart shows false positive rate (1-specificity) on X-axis, the probability of target=1 when its true value is 0, against true positive rate (sensitivity) on Y-axis, the probability of target=1 when its true value is 1. Ideally, the curve will climb quickly toward the top-left meaning the model correctly predicted the cases. The diagonal line is for a random model.

Area under ROC curve is often used as a measure of quality of the classification models. A random classifier has an area under the curve of 0.5, while AUC for a perfect classifier is equal to 1. In practice, most of the classification models have an AUC between 0.5 and 1

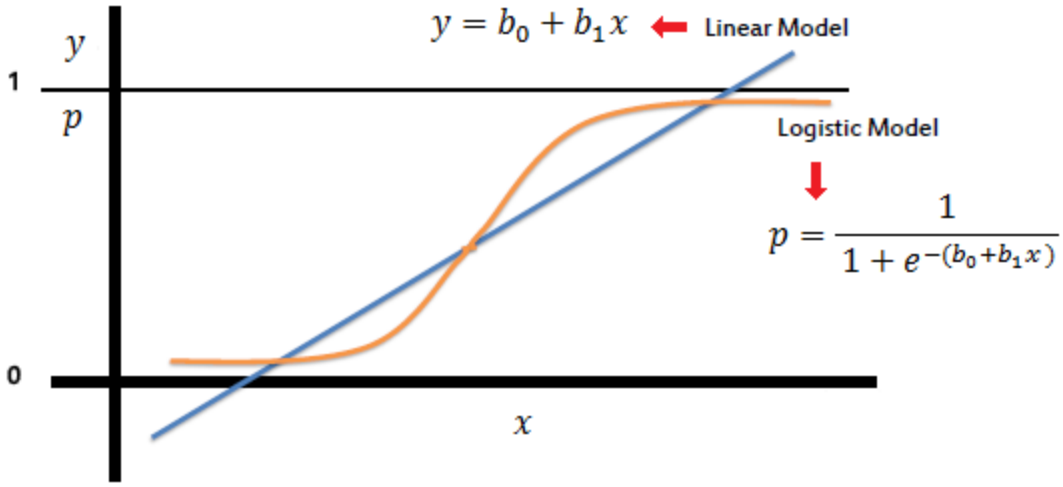


Logistic Regression

Logistic regression predicts the probability of an outcome that can only have two values (i.e. a dichotomy). The prediction is based on the use of one or several predictors (numerical and categorical). A linear regression is not appropriate for predicting the value of a binary variable for two reasons:

- A linear regression will predict values outside the acceptable range (e.g. predicting probabilities outside the range 0 to 1)
- Since the dichotomous experiments can only have one of two possible values for each experiment, the residuals will not be normally distributed about the predicted line.

On the other hand, a logistic regression produces a logistic curve, which is limited to values between 0 and 1. Logistic regression is similar to a linear regression, but the curve is constructed using the natural logarithm of the "odds" of the target variable, rather than the probability. Moreover, the predictors do not have to be normally distributed or have equal variance in each group.



In the logistic regression the constant (b_0) moves the curve left and right and the slope (b_1) defines the steepness of the curve. By simple transformation, the logistic regression equation can be written in terms of an odds ratio.

$$\frac{p}{1-p} = \exp(b_0 + b_1 x)$$

Finally, taking the natural log of both sides, we can write the equation in terms of log-odds (logit) which is a linear function of the predictors. The coefficient (b_1) is the amount the logit (log-odds) changes with a one unit change in x.

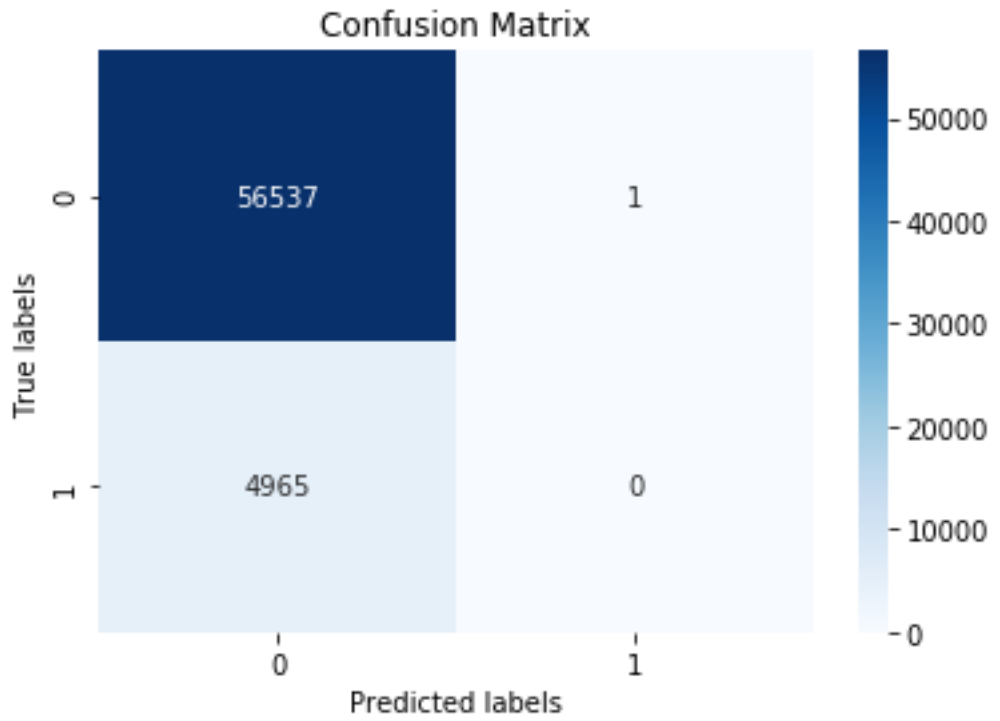
$$ln\left(\frac{p}{1-p}\right) = b_0 + b_1 x$$

As mentioned before, logistic regression can handle any number of numerical and/or categorical variables.

$$p = \frac{1}{1 + e^{-(b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_p x_p)}}$$

Model Evaluation

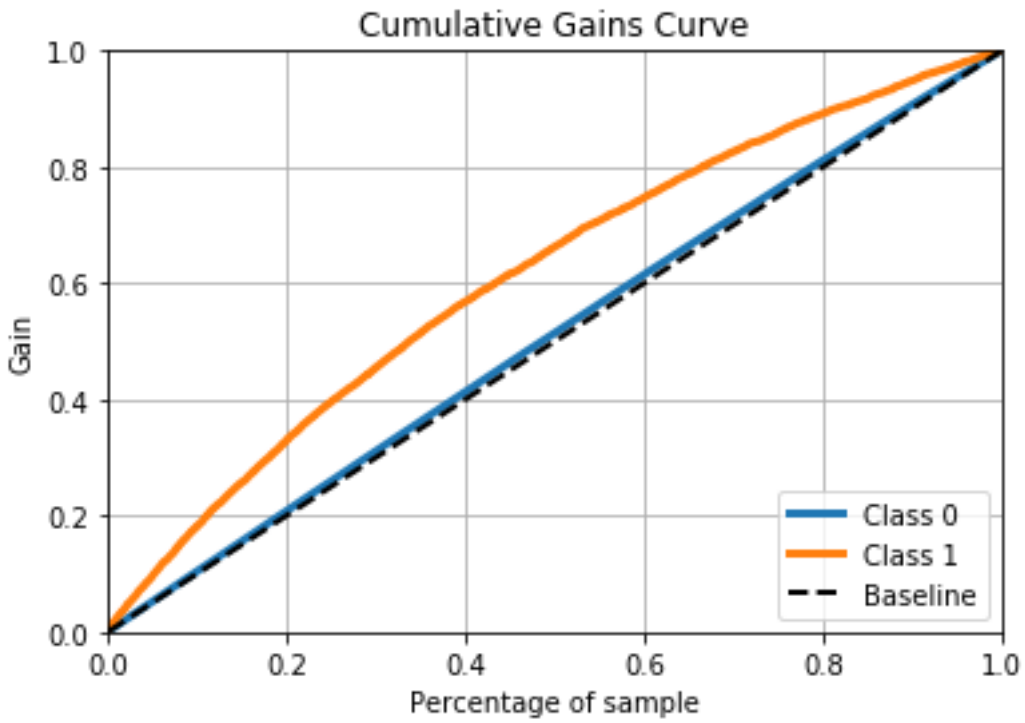
Confusion Matrix



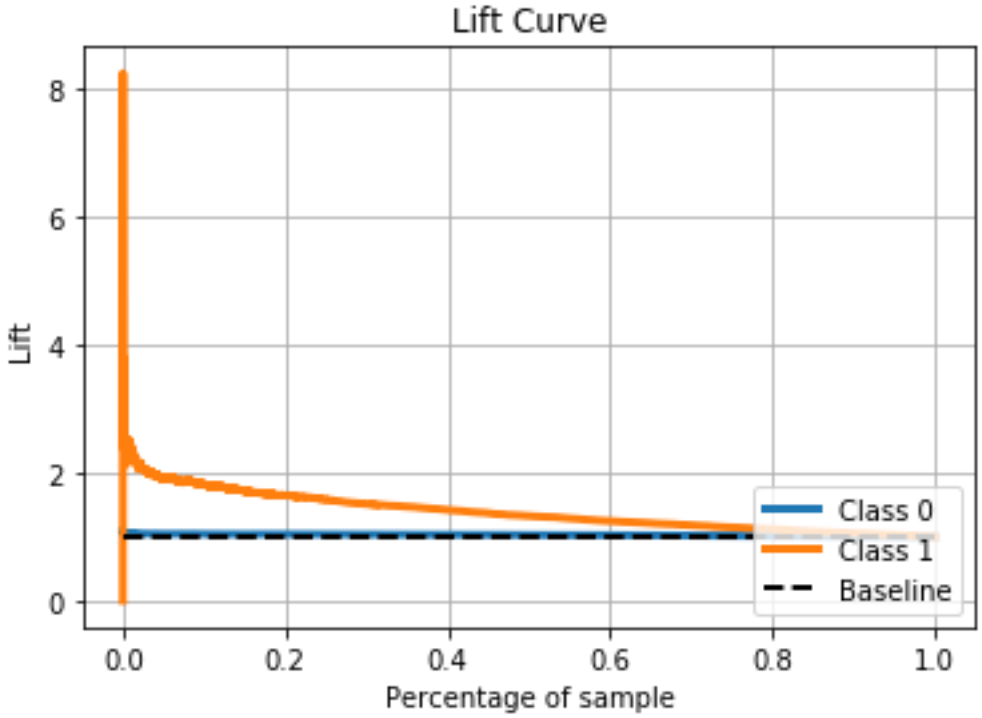
TN = 56537, FP = 1 , FN= 4965, TP = 0

Accuracy = (TP+TN) / (TP+TN+FP+FN)	0.9192559712534348
Sensitivity, Recall or TPR = TP / (TP+FN)	0.00
Specificity or TNR = TN / (TN+FP)	0.9999823127807846
Precision or Positive Predictive Value (PPV) = TP/ (TP+FP)	0.00
Negative Predictive Value (NPV) = TN/(TN+FN)	0.9192709180189262
False Discovery Rate (FDR) = 1-Precision	1.0
F1 score (harmonic mean of precision and sensitivity(recall))	0.00

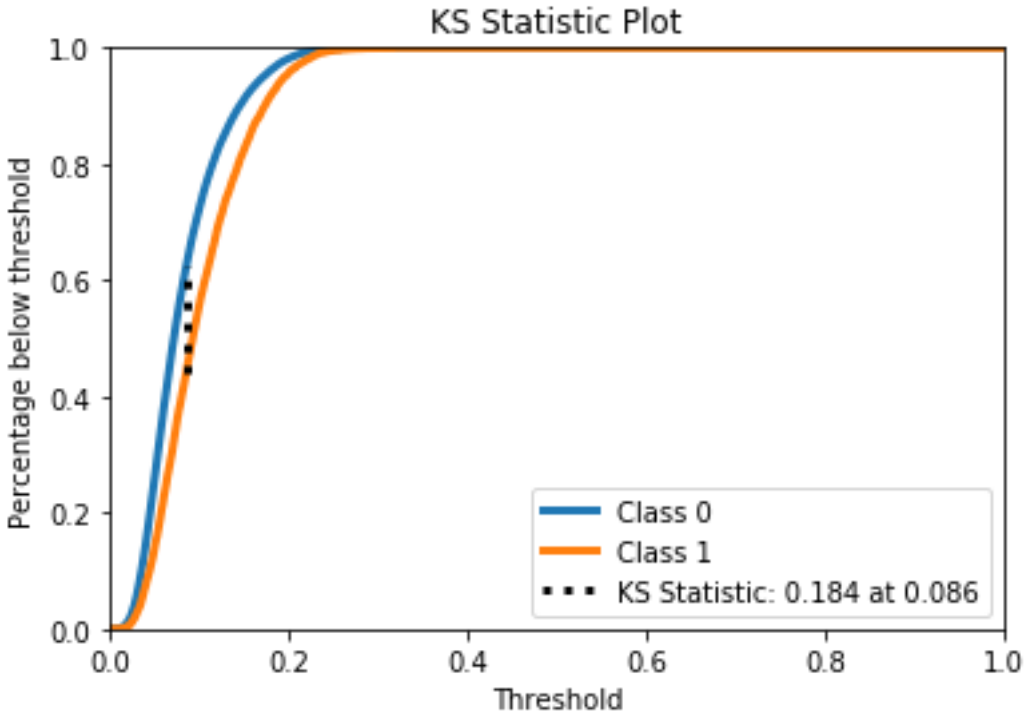
Gain and Lift Chart



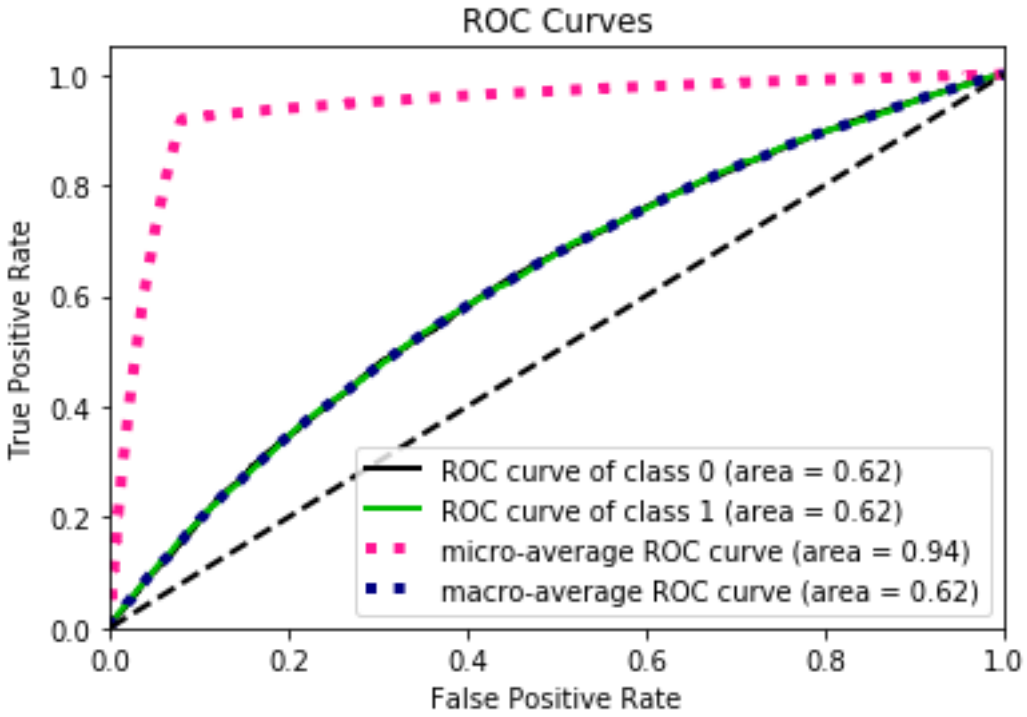
Lift Chart



K-S Chart


KS Statistic Plot
Percentage below threshold
Threshold
Class 0
Class 1
KS Statistic: 0.184 at 0.086
0.0
0.2
0.4
0.6
0.8
1.0

ROC Chart


ROC Curves
True Positive Rate
False Positive Rate
ROC curve of class 0 (area = 0.62)
ROC curve of class 1 (area = 0.62)
micro-average ROC curve (area = 0.94)
macro-average ROC curve (area = 0.62)
0.0
0.2
0.4
0.6
0.8
1.0

Decision Tree

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**. A decision node (e.g., Outlook) has two or more branches (e.g., Sunny, Overcast and Rainy). Leaf node (e.g., Play) represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called **root node**. Decision trees can handle both categorical and numerical data.

Algorithm

The core algorithm for building decision trees called **ID3** by J. R. Quinlan which employs a top-down, greedy search through the space of possible branches with no backtracking. ID3 uses *Entropy* and *Information Gain* to construct a decision tree. In ZeroR model there is no predictor, in OneR model we try to find the single best predictor, naive Bayesian includes all predictors using Bayes' rule and the independence assumptions between predictors but decision tree includes all predictors with the dependence assumptions between predictors.

Entropy

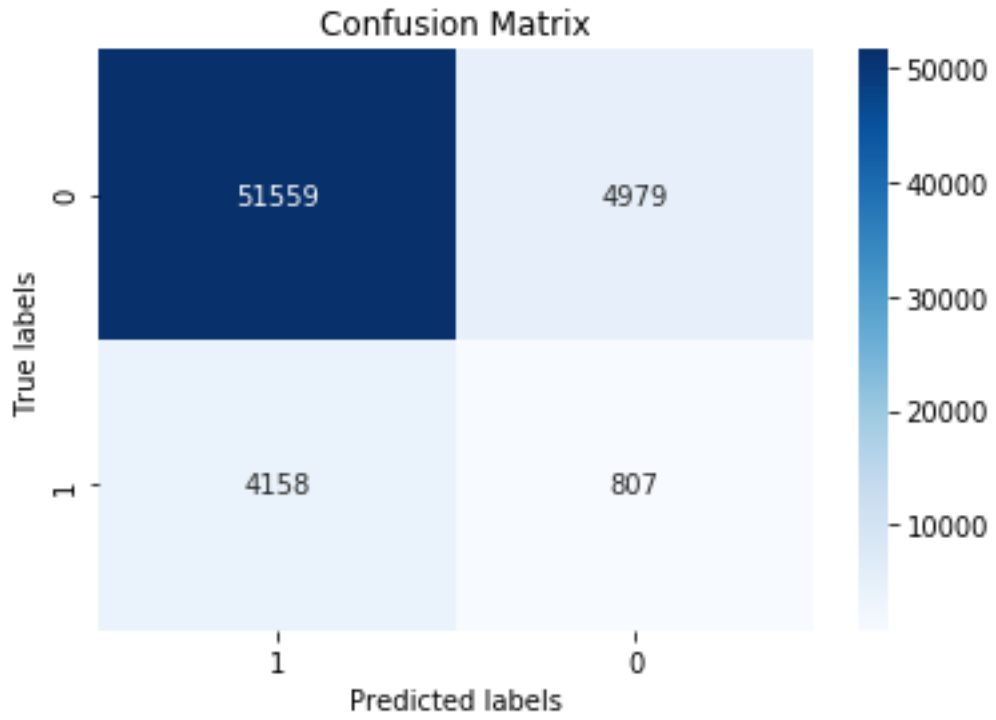
A decision tree is built top-down from a root node and involves partitioning the data into subsets that contain instances with similar values (homogenous). ID3 algorithm uses entropy to calculate the homogeneity of a sample. If the sample is completely homogeneous the entropy is zero and if the sample is an equally divided it has entropy of one.

Information Gain

The information gain is based on the decrease in entropy after a dataset is split on an attribute. Constructing a decision tree is all about finding attribute that returns the highest information gain (i.e., the most homogeneous branches).

Model Evaluation

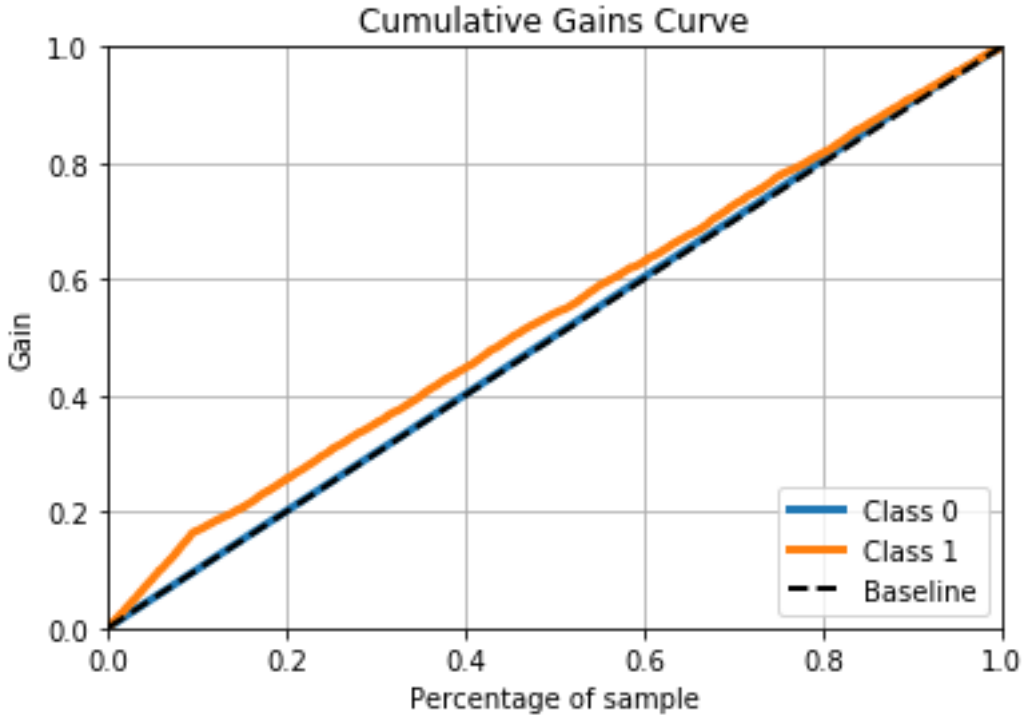
Confusion Matrix



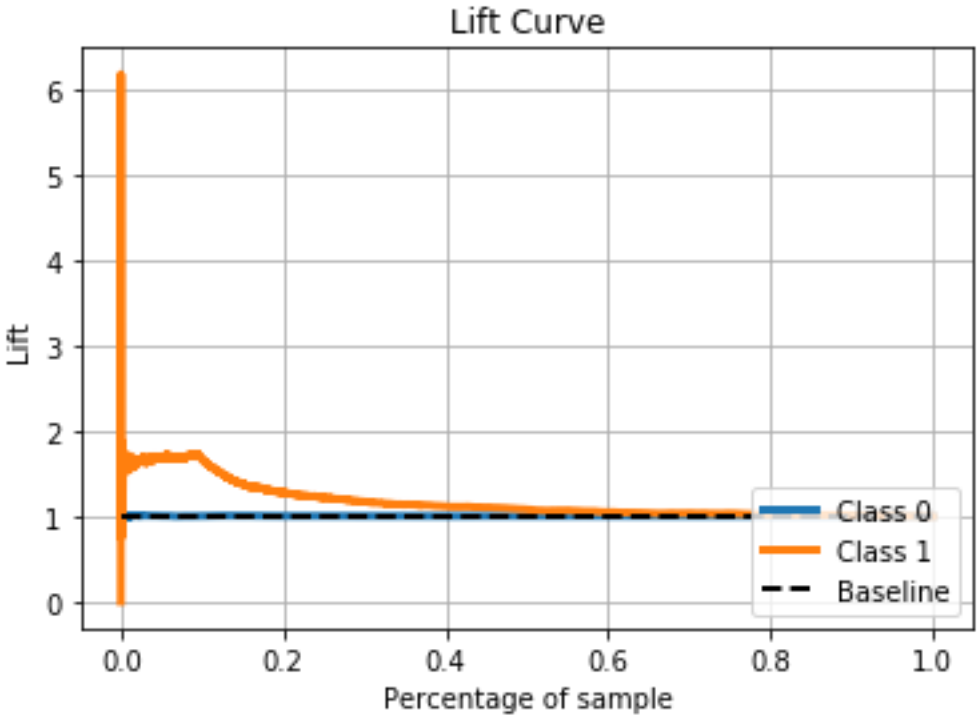
TN = 51559, FP = 4979 , FN= 4158, TP = 807

Accuracy = (TP+TN) / (TP+TN+FP+FN)	0.8514381412288832
Sensitivity, Recall or TPR = TP / (TP+FN)	0.16253776435045317
Specificity or TNR = TN / (TN+FP)	0.9119353355265485
Precision or Positive Predictive Value (PPV) = TP/ (TP+FP)	0.13947459384721742
Negative Predictive Value (NPV) = TN/(TN+FN)	0.925372866450096
False Discovery Rate (FDR) = 1-Precision	0.8605254061527826
F1 score (harmonic mean of precision and sensitivity(recall))	0.15012556971444516

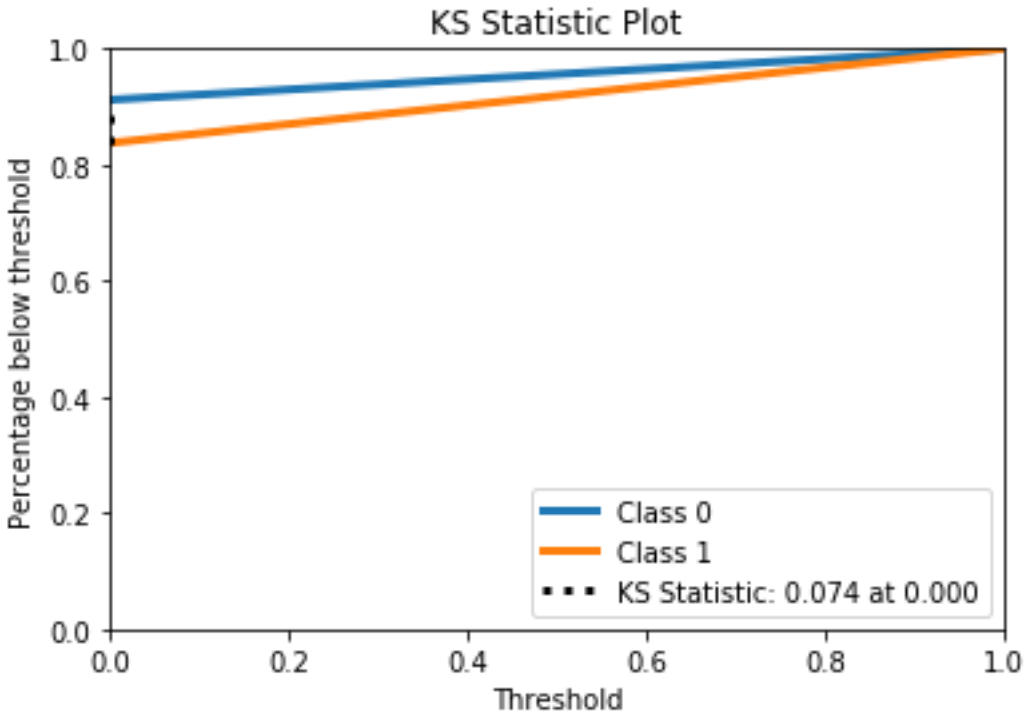
Gain and Lift Chart



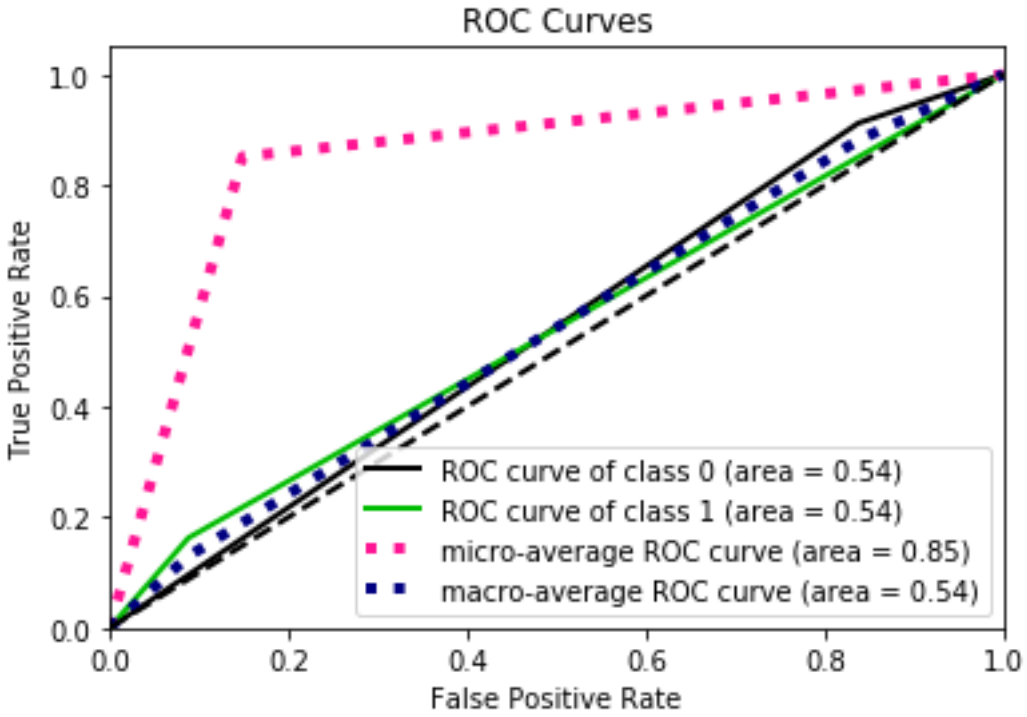
Lift Chart



K-S Chart



ROC Chart



Random Forest

Random forests, also known as random decision forests, are a popular ensemble method that can be used to build predictive models for both classification and regression problems.
Random Forests grows many classification trees. To classify a new object from an input vector, put the input vector down each of the trees in the forest. Each tree gives a classification, and we say the tree "votes" for that class. The forest chooses the classification having the most votes (over all the trees in the forest).

Each tree is grown as follows:

1. If the number of cases in the training set is N, sample N cases at random - but *with replacement*, from the original data. This sample will be the training set for growing the tree.
2. If there are M input variables, a number m<<M is specified such that at each node, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the forest growing.
3. Each tree is grown to the largest extent possible. There is no pruning.

In the original paper on random forests, it was shown that the forest error rate depends on two things:

- The *correlation* between any two trees in the forest. Increasing the correlation increases the forest error rate.
- The *strength* of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing m reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of m - usually quite wide. Using the oob error rate (see below) a value of m in the range can quickly be found. This is the only adjustable parameter to which random forests is somewhat sensitive.

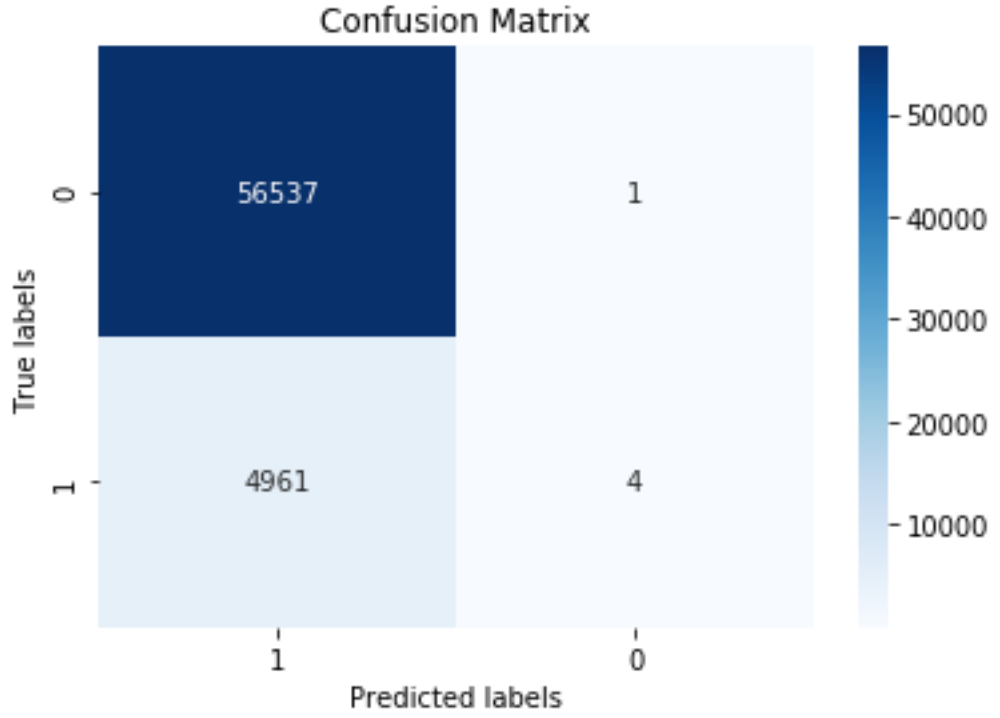
Features of Random Forests

- It is unexcelled in accuracy among current algorithms.
- It runs efficiently on large data bases.
- It can handle thousands of input variables without variable deletion.
- It gives estimates of what variables are important in the classification.
- It generates an internal unbiased estimate of the generalization error as the forest building progresses.
- It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing.
- It has methods for balancing error in class population unbalanced data sets.
- Generated forests can be saved for future use on other data.
- Prototypes are computed that give information about the relation between the variables and the classification.
- It computes proximities between pairs of cases that can be used in clustering, locating outliers, or (by scaling) give interesting views of the data.

- The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection.
- It offers an experimental method for detecting variable interactions.

Model Evaluation

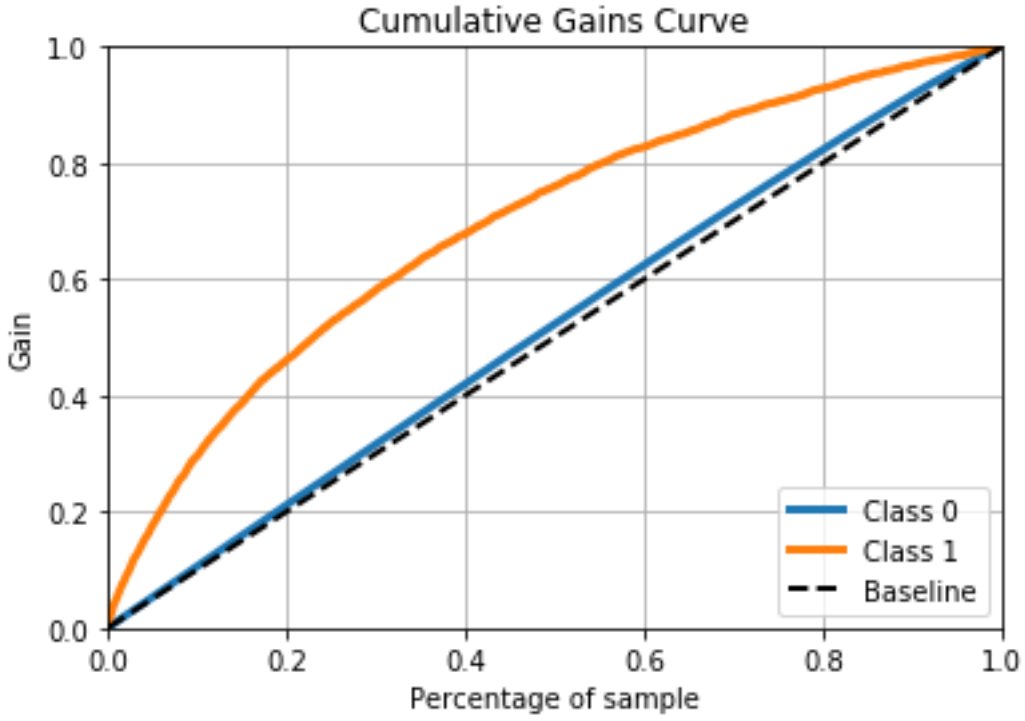
Confusion Matrix



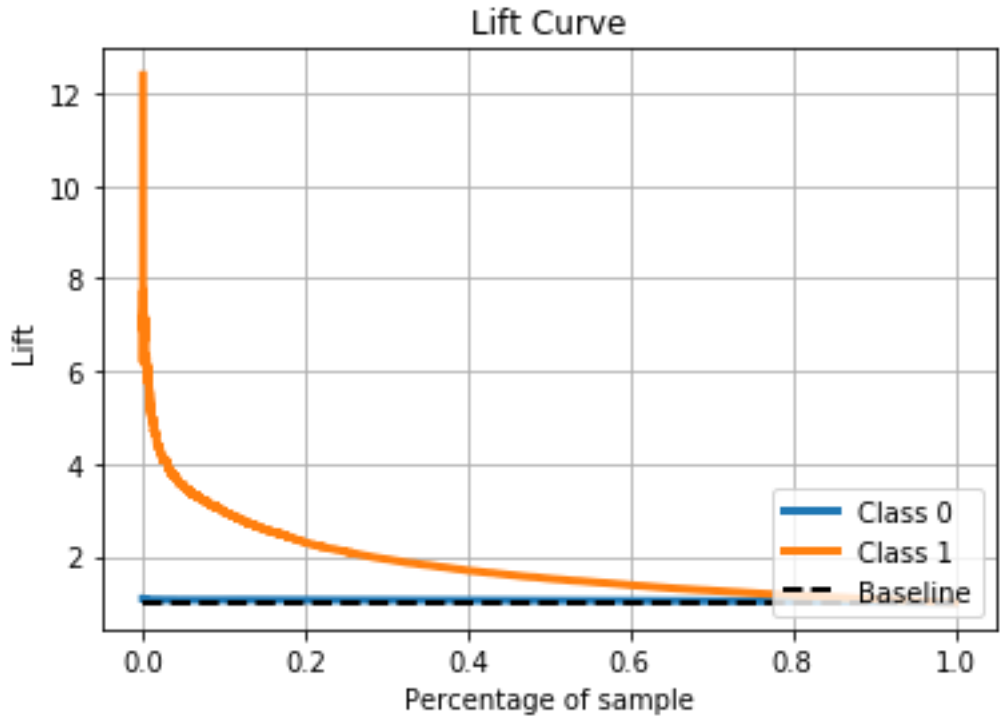
TN = 51559, FP = 1 , FN= 4961, TP = 4

Accuracy = (TP+TN) / (TP+TN+FP+FN)	0.9193210087312814
Sensitivity, Recall or TPR = TP / (TP+FN)	0.0008056394763343404
Specificity or TNR = TN / (TN+FP)	0.9999823127807846
Precision or Positive Predictive Value (PPV) = TP/ (TP+FP)	0.8
Negative Predictive Value (NPV) = TN/(TN+FN)	0.9193307099417867
False Discovery Rate (FDR) = 1-Precision	0.19999999999999996
F1 score (harmonic mean of precision and sensitivity(recall))	0.0016096579476861167

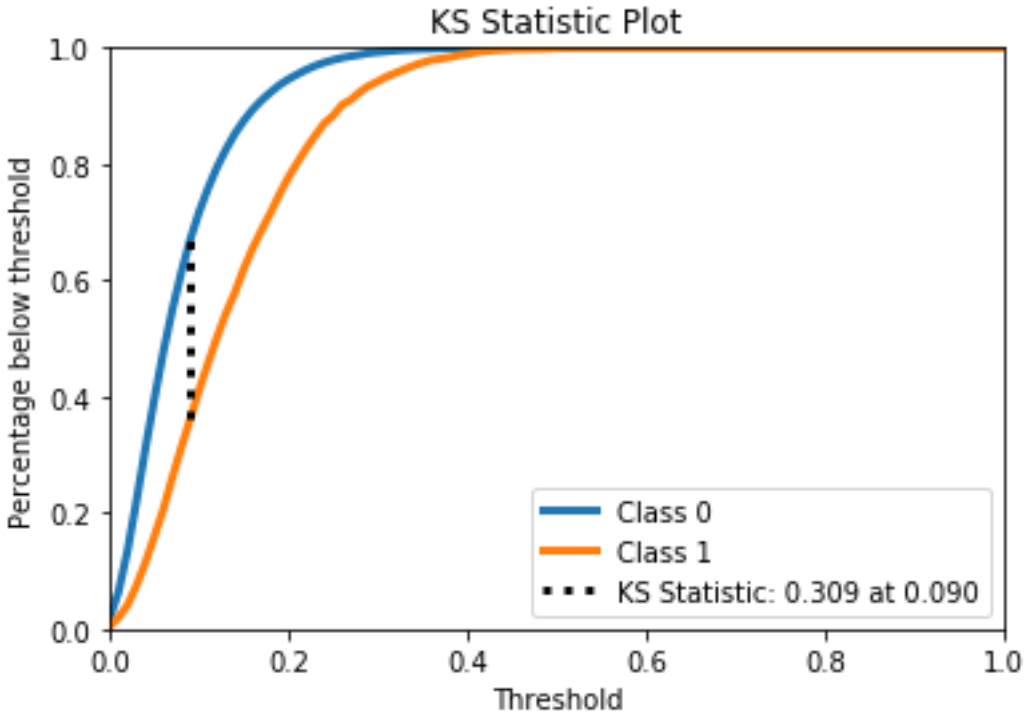
Gain and Lift Chart



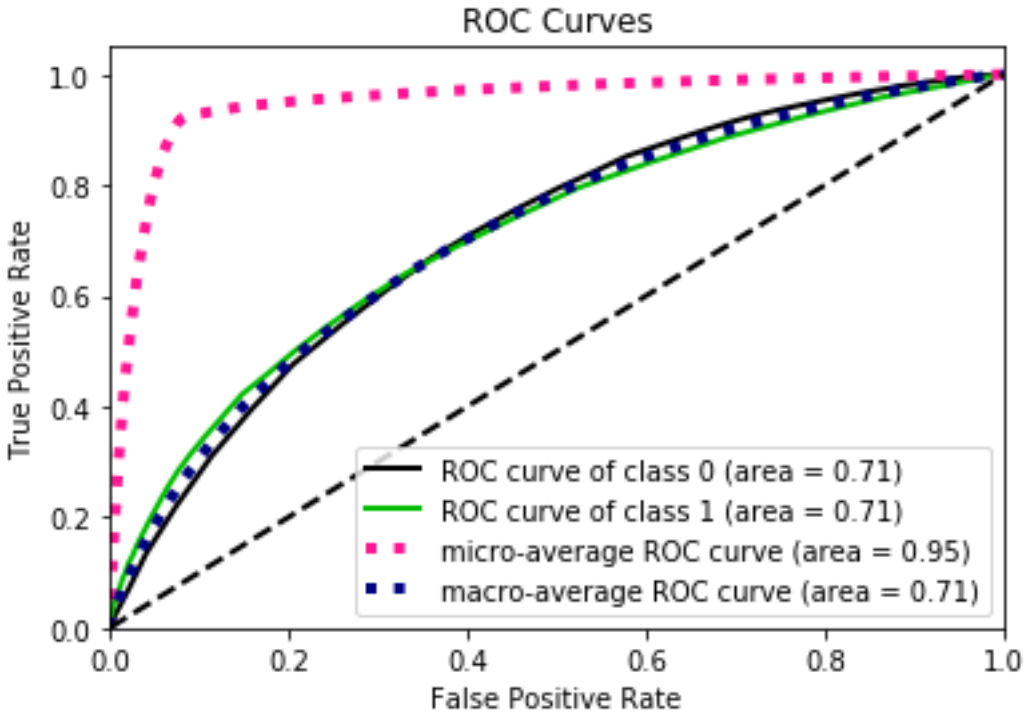
Lift Chart



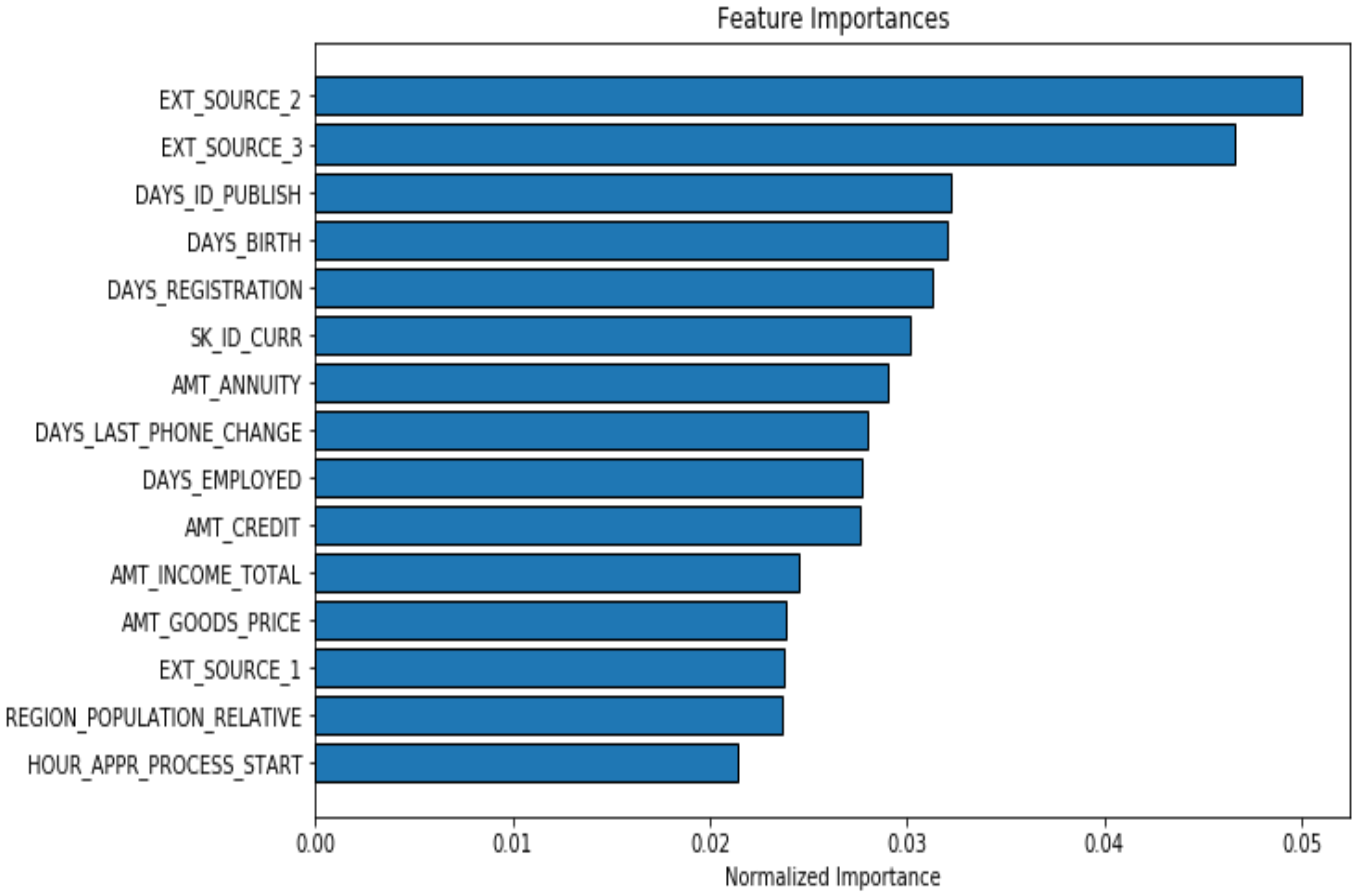
K-S Chart



ROC Chart



Top 15 Features for the prediction from the model



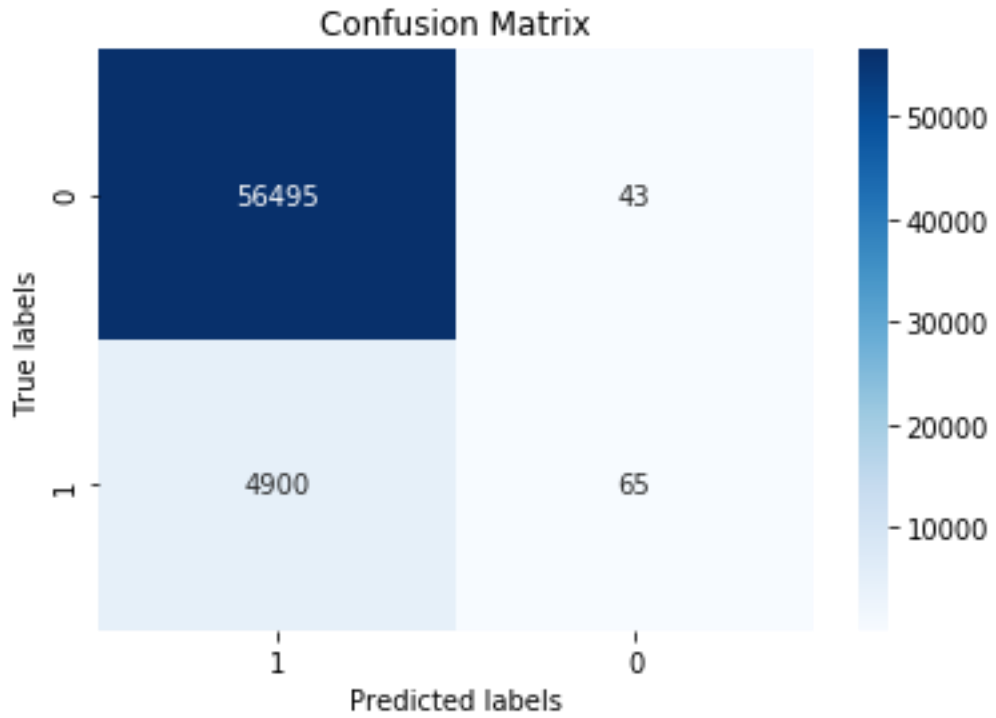
Gradient Boosting Machine (GBM)

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.

The idea of gradient boosting originated in the observation by Leo Breiman that boosting can be interpreted as an optimization algorithm on a suitable cost function.[1] Explicit regression gradient boosting algorithms were subsequently developed by Jerome H. Friedman, simultaneously with the more general functional gradient boosting perspective of Llew Mason, Jonathan Baxter, Peter Bartlett and Marcus Frean. The latter two papers introduced the view of boosting algorithms as iterative *functional gradient descent* algorithms. That is, algorithms that optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction. This functional gradient view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification.

Model Evaluation

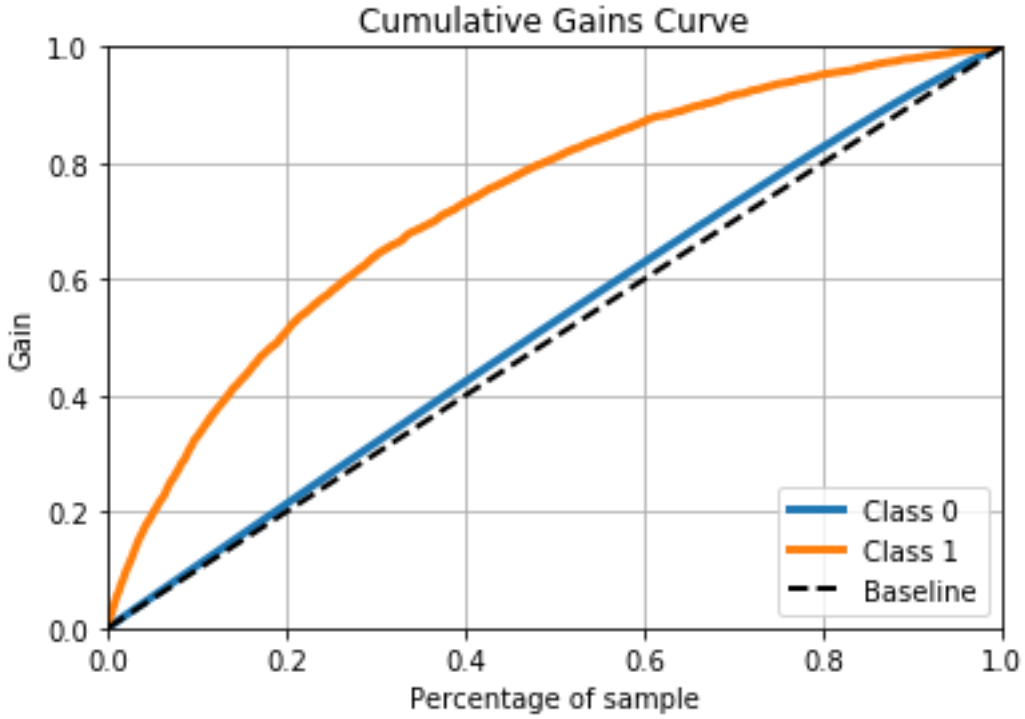
Confusion Matrix



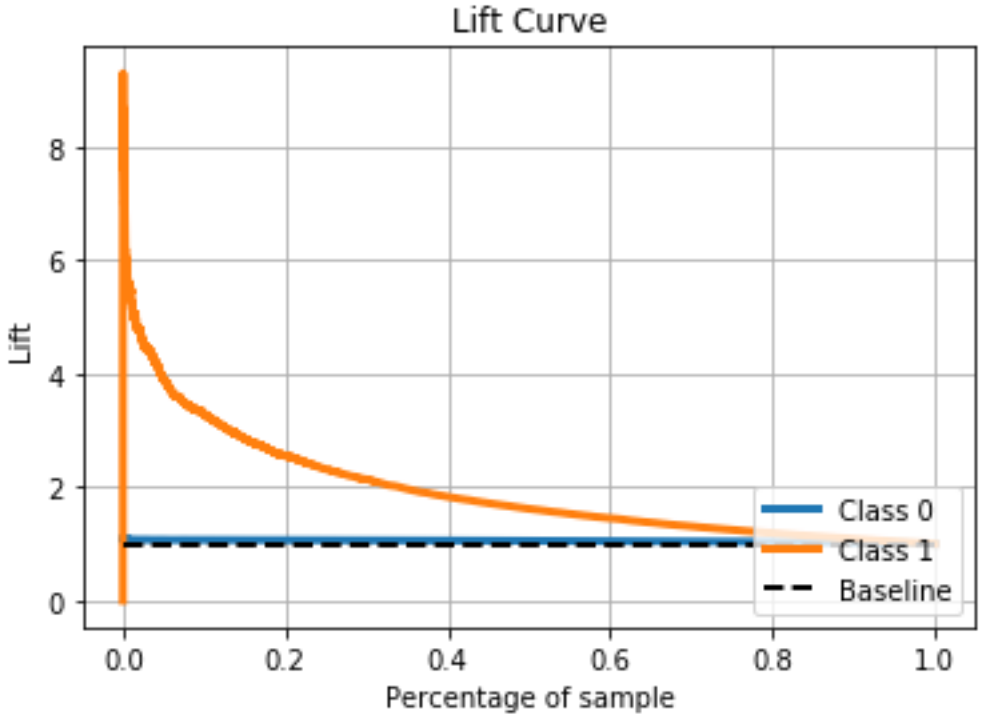
TN = 56495, FP = 43 , FN= 4900, TP = 65

Accuracy = (TP+TN) / (TP+TN+FP+FN)	0.9196299367510528
Sensitivity, Recall or TPR = TP / (TP+FN)	0.013091641490433032
Specificity or TNR = TN / (TN+FP)	0.999239449573738
Precision or Positive Predictive Value (PPV) = TP/ (TP+FP)	0.6018518518518519
Negative Predictive Value (NPV) = TN/(TN+FN)	0.9201889404674648
False Discovery Rate (FDR) = 1-Precision	0.39814814814814814
F1 score (harmonic mean of precision and sensitivity(recall))	0.025625862408831072

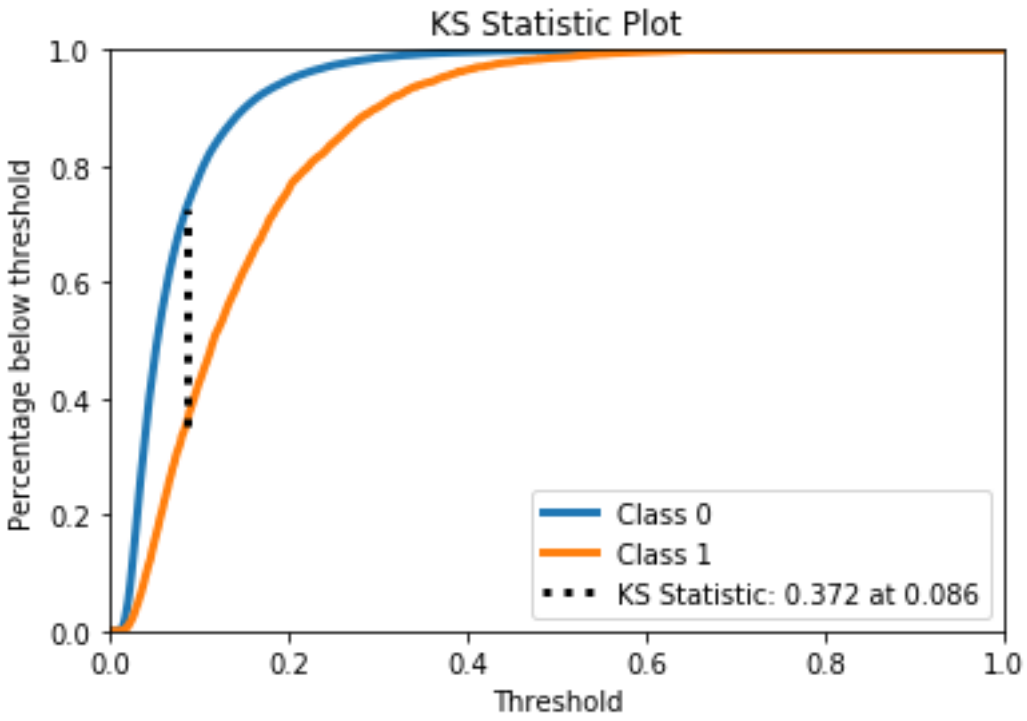
Gain and Lift Chart



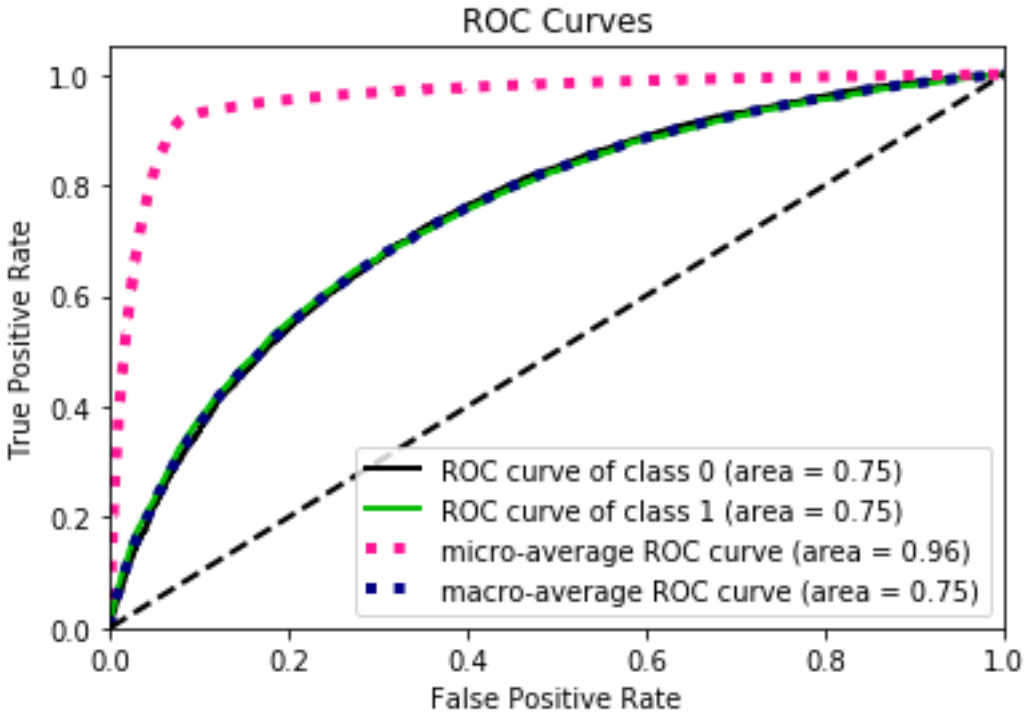
Lift Chart



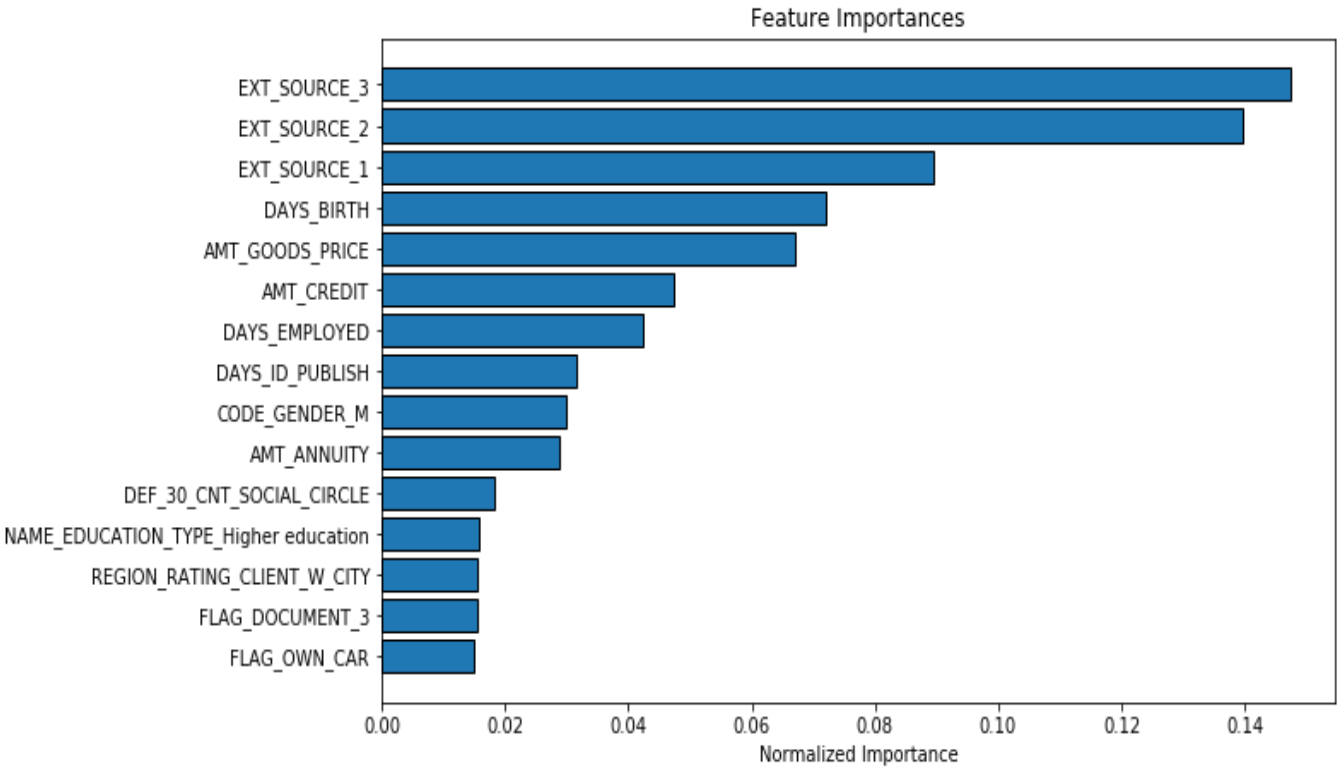
K-S Chart



ROC Chart



Top 15 Features for the prediction from the model



Deep Learning

Before we fed the data into the neural network, we perform feature scaling on the dataset. We make the dataset as numpy arrays. We use MinMax scaler for this preprocessing step. We fit_transform the features_train data and transform the features_test data.

Deep neural network with 4 layers

Deep neural network with 4 layers having 2 hidden layer and each hidden layer has 80 neurons. Dropout layer is added between the 2 hidden layers and between second hidden layer and the output layer to prevent overfitting. The Binary_crossentropy is used as loss function and updating method is ADAM. The number of Epochs is 500.

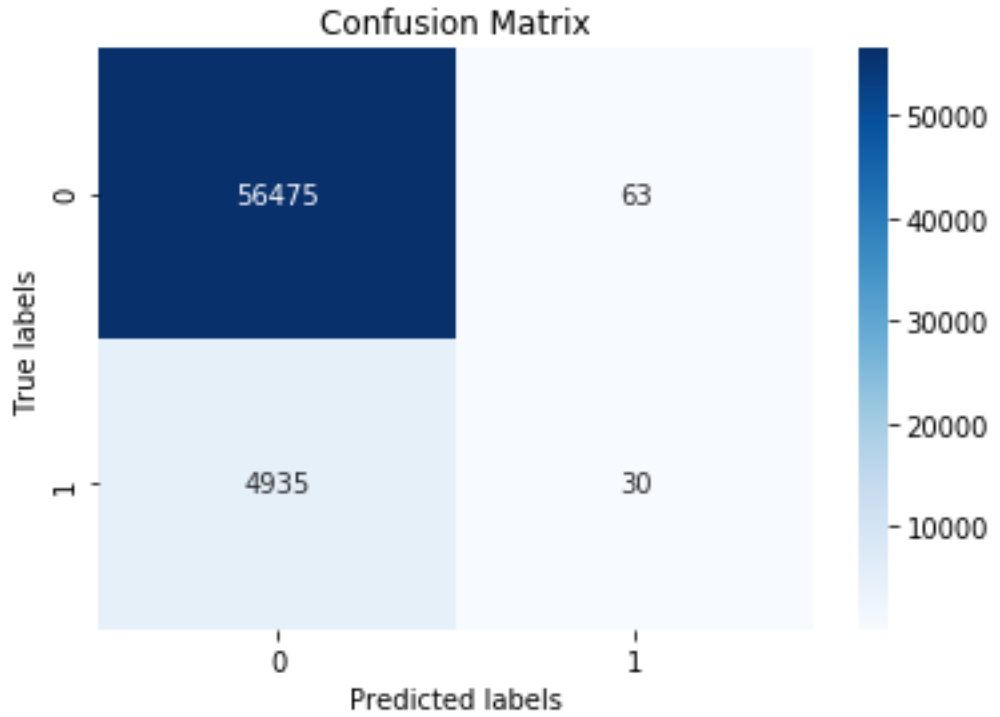
The input layer has a dimension of 242 for our data, the 2 hidden layers has rectifier as its activation function with 80 neurons each. The output layer has sigmoid activation function as we are doing binary classification. The Deep neural network is compiled with Adam optimizer, Binary_crossentropy loss function and the evaluation metrics is accuracy.

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 80)                19440
_________________________________________________________________
dropout_1 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_2 (Dense)              (None, 80)                6480
_________________________________________________________________
dropout_2 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 81
=================================================================
Total params: 26,001
Trainable params: 26,001
Non-trainable params: 0
```

Model Evaluation

Confusion Matrix



TN = 56475, FP = 63 , FN= 4935, TP = 30

Accuracy = (TP+TN) / (TP+TN+FP+FN)	0.918735671417094
Sensitivity, Recall or TPR = TP / (TP+FN)	0.006042296072507553
Specificity or TNR = TN / (TN+FP)	0.9988857051894301
Precision or Positive Predictive Value (PPV) = TP/ (TP+FP)	0.3225806451612903
Negative Predictive Value (NPV) = TN/(TN+FN)	0.9196384953590621
False Discovery Rate (FDR) = 1-Precision	0.6774193548387097
F1 score (harmonic mean of precision and sensitivity(recall))	0.011862396204033215

Deep neural network with 5 layers

Deep neural network with 5 layers having 3 hidden layer, first and second hidden layer has 80 neurons and third hidden layer has 40 neurons. Dropout layer is added between the hidden layers and between third hidden layer and the output layer, Batch normalization layer is inserted between third hidden layer and output layer to prevent overfitting. The number of Epochs is 400.

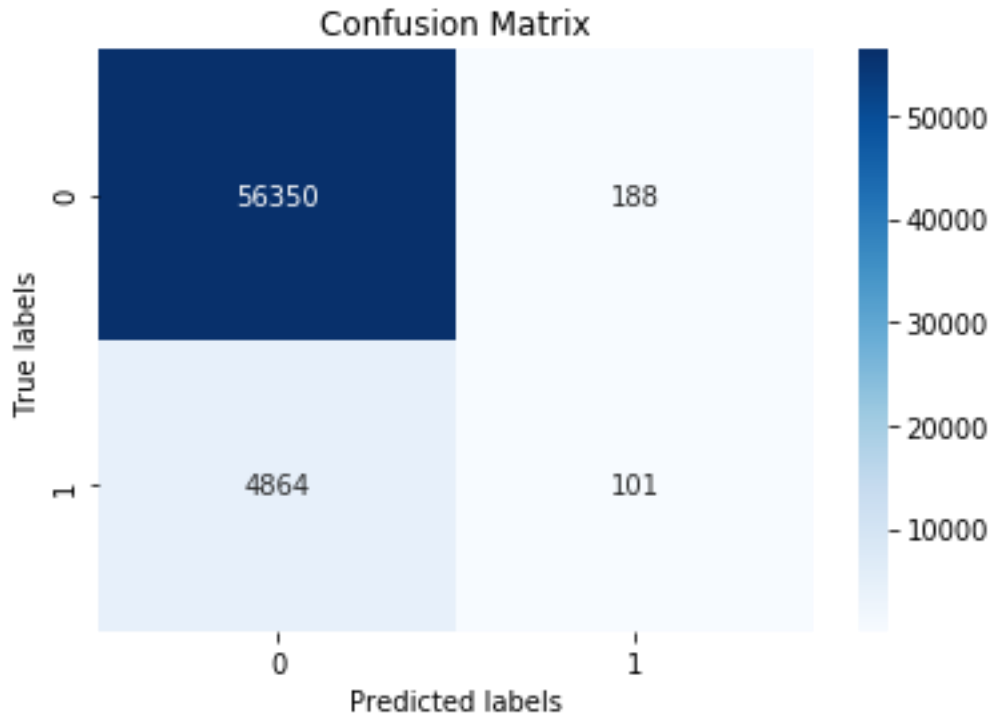
The input layer has a dimension of 242 for our data, the 3 hidden layers has rectifier as its activation function, first and second hidden layer has 80 neurons and third hidden layer has 40 neurons. The output layer has sigmoid activation function as we are doing binary classification. The Deep neural network is compiled with Adam optimizer, Binary_crossentropy loss function and the evaluation metrics is accuracy.

```
model_1.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 80)                19440
_________________________________________________________________
dropout_5 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_9 (Dense)              (None, 80)                6480
_________________________________________________________________
dropout_6 (Dropout)          (None, 80)                0
_________________________________________________________________
dense_10 (Dense)             (None, 40)                3240
_________________________________________________________________
batch_normalization_2 (Batch (None, 40)                160
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 41
=================================================================
Total params: 29,361
Trainable params: 29,281
Non-trainable params: 80
```

Model Evaluation

Confusion Matrix



TN = 56350, FP = 188 , FN= 4864, TP = 101

Accuracy = (TP+TN) / (TP+TN+FP+FN)	0.9178576654661649

Sensitivity, Recall or TPR = TP / (TP+FN)	0.020342396777442096
Specificity or TNR = TN / (TN+FP)	0.9966748027875058
Precision or Positive Predictive Value (PPV) = TP/ (TP+FP)	0.3494809688581315
Negative Predictive Value (NPV) = TN/(TN+FN)	0.9205410527003627
False Discovery Rate (FDR) = 1-Precision	0.6505190311418685
F1 score (harmonic mean of precision and sensitivity(recall))	0.03844689760182718

After comparing the performance of all the above technique, **Gradient Boosting machine (GBM)** performs the best followed by the **Deep Learning neural network classifier** followed by **Random Forest.** We save the model to disk using Python's built in persistence model (pickle or dill) and use this model for prediction on new data.

Web Application - Restful API

Python -> Machine Learning/Deep Learning model-> pickle model -> flask -> deploy on Heroku cloud

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. It is relatively easy to set up a website on Flask using Jinja2 templating.

Heroku is a cloud platform as a service (PaaS) supporting several programming languages.

We create the simple flask app. The flask app consists of 2 main components: the python app (app.py) and the HTML templates. While we can return HTML code from the python file itself, it would be cumbersome to code entire HTML as a string in the python file. Templating come to the rescue!

References

1. Home Credit Data sources : https://www.kaggle.com/c/home-credit-default-risk
2. Sayad, Saed. (2017). Real Time Data Mining, http://www.saedsayad.com
3. Consumer Credit Risk models via Machine Learning Algorithms by AE Khandani, AJ Kim, AW Lo, https://dspace.mit.edu/openaccess-disseminate/1721.1/66301
4. Python : https://www.python.org
5. Scikit-learn – Machine Learning in python : https://scikit-learn.org/stable/#
6. Scikit-plot : https://scikit-plot.readthedocs.io/en/stable/Quickstart.html
7. Data visualization : https://matplotlib.org , https://seaborn.pydata.org , https://plot.ly
8. Heroku cloud platform : https://www.heroku.com
9. Home Credit Logo : www.homecreditus.com
10. Cover page Loan repay image :
    www.towardsdatascience.com/predicting-loan-repayment-5df4e0023e92