STAT3612: STATISTICAL MACHINE LEARNING
GROUP PROJECT FINAL REPORT

TOPIC: INTERPRETABLE MACHINE LEARNING ON HELOC DATASET

Group 4

ID: 3612202007

Members:

Name	UID
JANG Haeyoon	3035493581
CHIA Dong Ern	3035362500
LOH Xi Zhe	3035603174
KIM Bohyun	3035425259
CHAN Jun Yi	3035602558

Abstract

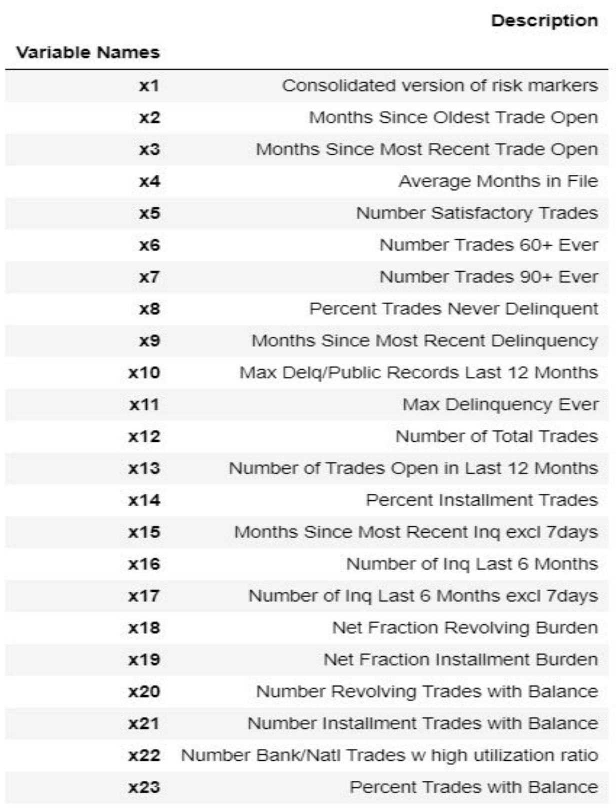
Our goal was to select a best performing machine learning model in predicting home equity loan performance using the individual information available since the trade was requested. We also aimed to produce sufficient interpretation of the selected model including features that largely contribute to the prediction result and relationships between the features. The data has 10459 rows and 23 columns which stand for different features. Eight different machine learning models ranging from linear regression to Gradient boosting tree were investigated then comparisons were made based on prediction accuracy score. As a result, we selected the LogisticGAM model for its best test accuracy score among white-box models and the LightGBM classifier model for its best test performance among black-box models. Then we have investigated the difference in monotonic constraint knowledge when added to the model and if it improves prediction result.We have reviewed the selected variables from the models and analyzed the variable importance measures using various post-hoc interpretation tools. Hyper-parameters were found using grid search cross validation to optimize our selected model.

Table of Contents

1. Introduction

Home equity line of credit, which is also called HELOC, is a loan that can offer extremely competitive interest rates by using the borrower's home as a collateral. HELOC comes from different funds sources, such as writing a cheque, online transfer or credit cards. Since it involves a large sum of credit in every transaction, banks have to carefully evaluate the financial situation of every client to avoid any default risk.

The FICO HELOC dataset contains anonymized information about home equity line of credit (HELOC) applications made by real homeowners. In this dataset, the customers have requested a credit line in the range of USD 5,000 - 150,000. The target variable in this dataset is a binary variable called RiskPerformance, where the aim of this project is to predict whether the customer will repay their HELOC account within 24 months. The value “Bad” indicates that an applicant was 90 days past due or worse at least once over a period of 24 months from when the credit account was opened. The value “Good” indicates that they have made their payments without ever being more than 90 days overdue. Besides the binary response variable ‘RiskFlag’, the dataset also contains 23 input variables used in making predictions and there are 10459 rows of data in total. The names of variables in the data are listed below.



	Description
Variable Names	
x1	Consolidated version of risk markers
x2	Months Since Oldest Trade Open
x3	Months Since Most Recent Trade Open
x4	Average Months in File
x5	Number Satisfactory Trades
x6	Number Trades 60+ Ever
x7	Number Trades 90+ Ever
x8	Percent Trades Never Delinquent
x9	Months Since Most Recent Delinquency
x10	Max Delq/Public Records Last 12 Months
x11	Max Delinquency Ever
x12	Number of Total Trades
x13	Number of Trades Open in Last 12 Months
x14	Percent Installment Trades
x15	Months Since Most Recent Inq excl 7days
x16	Number of Inq Last 6 Months
x17	Number of Inq Last 6 Months excl 7days
x18	Net Fraction Revolving Burden
x19	Net Fraction Installment Burden
x20	Number Revolving Trades with Balance
x21	Number Installment Trades with Balance
x22	Number Bank/Natl Trades w high utilization ratio
x23	Percent Trades with Balance

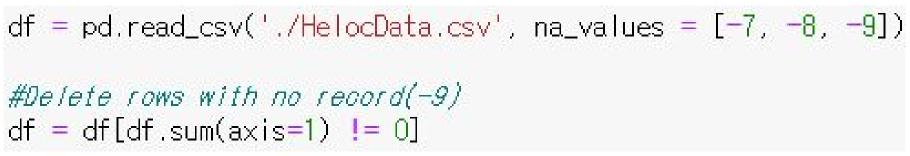
In this project, our main goals are to select the best performing machine learning model in predicting the target variable RiskPerformance, to investigate the difference brought by the monotonicity constraint to the selected model and to produce sufficient interpretation of the selected model. We will build various machine learning models and make comparisons among these models in terms of accuracy. Then, we will focus on the two models with highest accuracy, which are Generalized Additive Model, Light Gradient Boosting Machine and carry out interpretation, particularly for black box models with post-hoc explainability analysis. Through the black-box model results, we can extract variables which significantly influenced the prediction results and further analyze their relationships. We conclude our project by making detailed interpretations on the results.

2. Data Preprocessing

Data in real life is not refined to be used directly. It usually requires a large amount of effort to preprocess before starting the training. Since HELOC dataset is obtained from real past records, preprocessing was compulsory.

2.1. Missing value management

A major part of preprocessing is to deal with missing values. Three negative sign values -7, -8, -9 are regarded as missing values for the reasons 'Condition not met', 'No Usable/Valid Trades or Inquiries' and 'No Bureau record or No investigation' respectively. They were set to be na_values when loading. Also, we have observed that some rows are mostly filled with one of these missing values. We judged the 'empty row with no records' to be better removed, and there were 588 rows of them. Now we have continued the analysis with 9871 rows in total.



```
df = pd.read_csv('./HelocData.csv', na_values = [-7, -8, -9])

#Delete rows with no record(-9)
df = df[df.sum(axis=1) != 0]
```

Since we do not have a new set of data for validating trained models from the future, we have used 20% of our rows to be kept separate for testing. Hence we splitted the data into training and test sets with the ratio of 8:2.

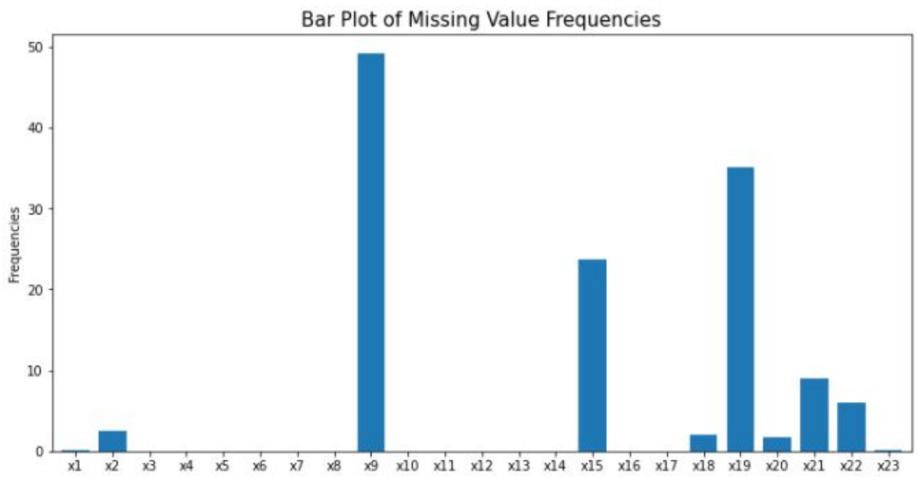
Then, to prevent leakage of training data to test data, we have applied imputation to still missing values separately. For each training and test data, we have imputed mean values from each set using sklearn SimpleImputer.

```
def SImputer(df):
    df_imputed = df.copy()
    SI = SimpleImputer(missing_values=np.nan)
    SI.fit(df.iloc[:, 1:])
    df_imputed.iloc[:, 1:] = SI.transform(df.iloc[:, 1:])
    return df_imputed
```

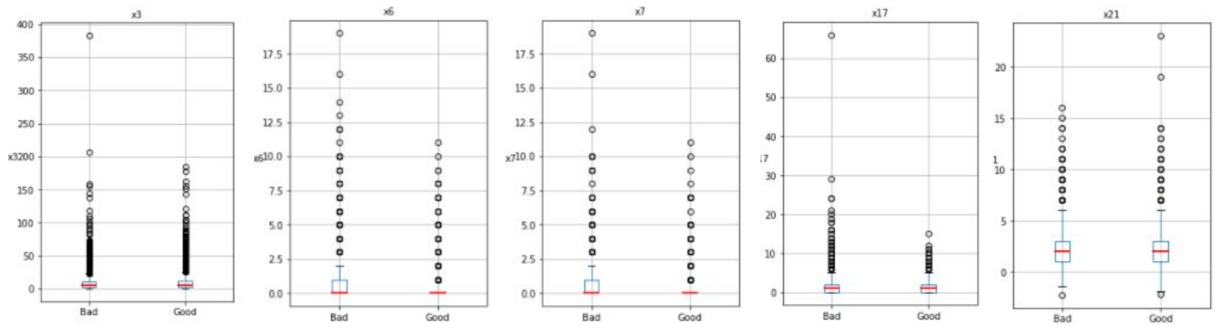
2.2. Feature engineering

We have explored meanings and distributions of each variable and found that some of the variables are critical to rely on. The reasons are as follows: high missing value frequency, no difference in distribution between two response 'RiskFlag', too many or widely spreaded outliers, Large p-value in GLM full model summary report, and high correlation in between variables.

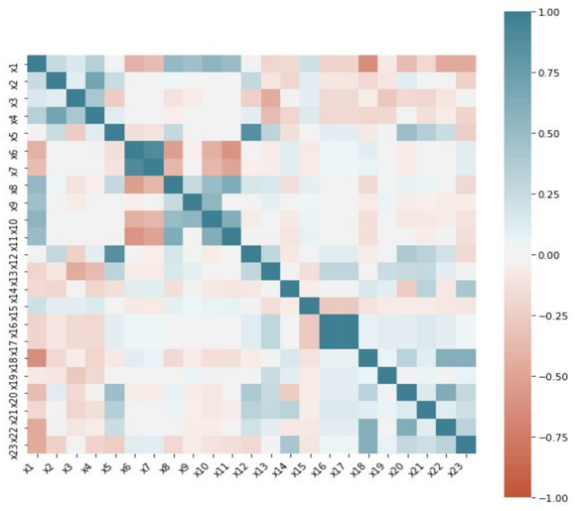
Although we are to impute the remaining missing values, we thought imputing too many null values is critical and may not well represent the impact of the variable. Missing value frequency higher than 30 rows are chosen to be discarded. Those were variables x9 and x19.



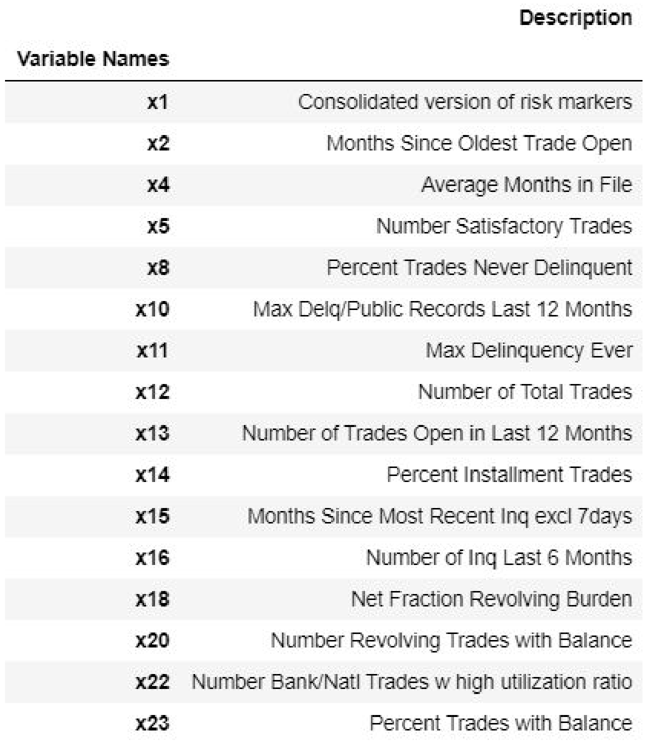
The next criterion was to observe distribution and abnormality of each variable by boxplot. In order to be helpful in predicting binary responses, we assume that the predictive variable should have different box plots between two response groups; Good or Bad. Those with similar shape of the distribution particularly the location of medians are regarded as helpless in making a good prediction hence we decided to drop variables x3, x6,x7,x17 and x21. Outliers were also taken into account if there are too many and widely spreaded candidates. x3, x6, x7 fell into this criterion.



Also, referring to the summary result of GLM Logistic regression model with full variable indicated less useful variables by high p-values and those were x7,x16 and x17. However these variables were not deleted straightaway since such detection may be due to the effect of correlation between variables. Hence, we looked at the correlation matrix between the explanatory variables by correlation heatmap. It turned out that variables x6,x7 and x16,x17 are highly correlated to each other. Their definitions in fact, are quite overlapping. X6 and x7 which both fall into other deletion criteria were determined to be discarded, and for x16 and x17, we decided to take in x16 and drop x17 since x16 contains slightly more information, plus that x17 falls into the second deletion criterion. By such removal, the correlation problem was solved.



As a result, the final variables to use were selected by dropping those variables that fall into at least one of above criterion badly. Final selected variables are ['x1','x2','x4','x5','x8','x10','x11','x12','x13','x14','x15','x16','x18','x20','x22','x23'] and the ones excluded are [x3,x6,x7,x9,x17,x19,x21].



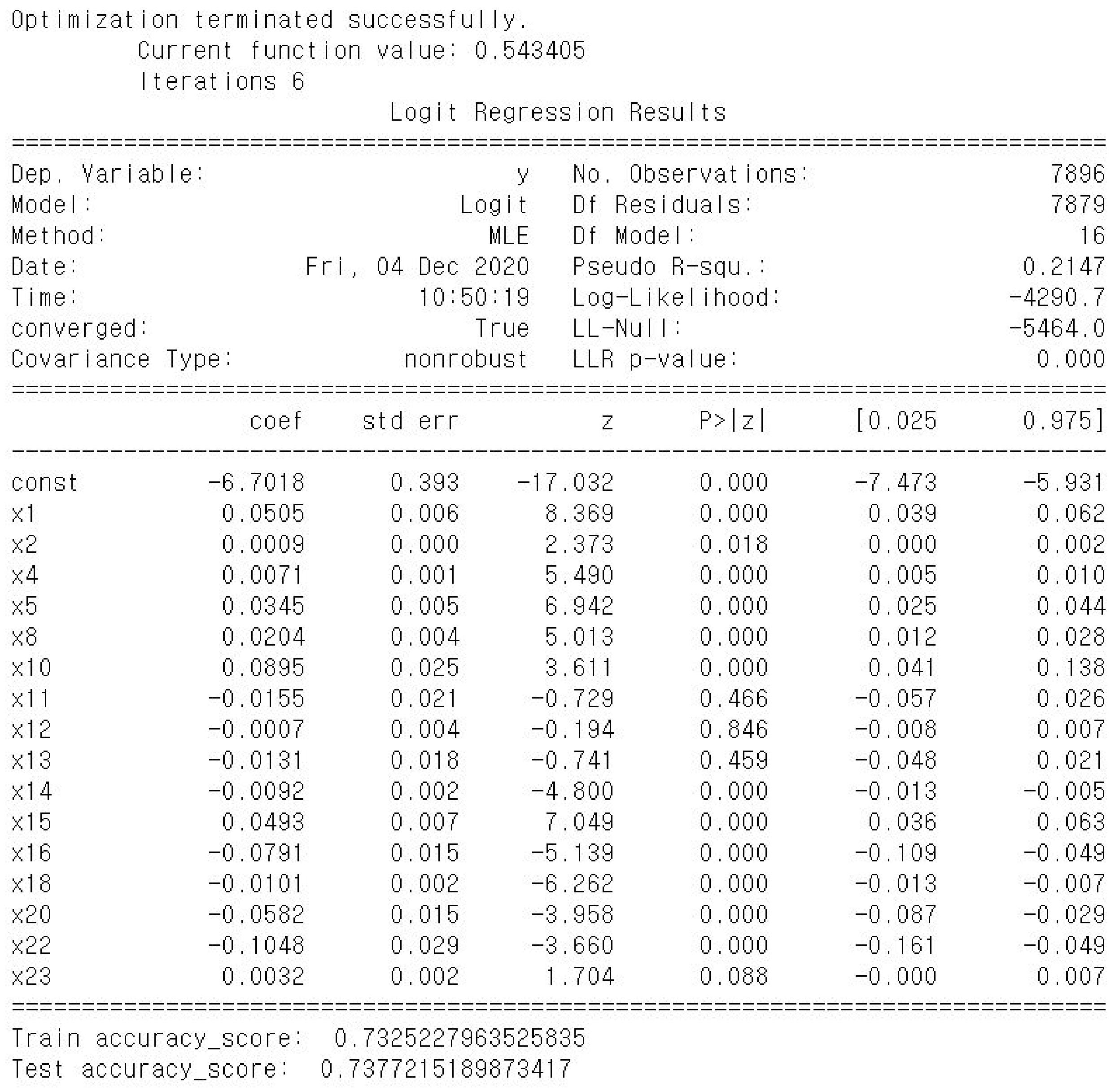
Variable Names	Description
x1	Consolidated version of risk markers
x2	Months Since Oldest Trade Open
x4	Average Months in File
x5	Number Satisfactory Trades
x8	Percent Trades Never Delinquent
x10	Max Delq/Public Records Last 12 Months
x11	Max Delinquency Ever
x12	Number of Total Trades
x13	Number of Trades Open in Last 12 Months
x14	Percent Installment Trades
x15	Months Since Most Recent Inq excl 7days
x16	Number of Inq Last 6 Months
x18	Net Fraction Revolving Burden
x20	Number Revolving Trades with Balance
x22	Number Bank/Natl Trades w high utilization ratio
x23	Percent Trades with Balance

3. Prediction performance comparison

We have investigated eight different machine learning models: Generalized Linear Model, Generalized Additive Model, Support Vector Machine, Decision Tree, Random Forest, Neural Network, XGBoost and LightGBM. Performance of each model was compared by the test prediction accuracy which was measured using accuracy_score in sklearn.metrics library. Simple and interpretable white box models, which include GLM and GAM were evaluated through summary report and partial dependence plot. On the other hand, black box models which include SVM, Decision Tree(bagging), Random Forest, Neural Network, XGBoost and LightGBM, were firstly evaluated by the accuracy score, and more detailed interpretation was done later by post-hoc interpretation tools.

3.1. Generalized Linear Model

A logistic regression model with newton-cg solver was selected to predict binary response with the estimated probability of ‘Good’ event. We have tried models with and without different regularization penalties including Lasso(L1), Ridge(L2) and Elastic Net and Ridge regularization showed the largest improvement in prediction accuracy, so we decided to run logistic regression with Ridge(L2) regularization penalty. Based on the selected model, the train and test accuracy scored 0.733 and 0.737 respectively. Summary report is shown below.

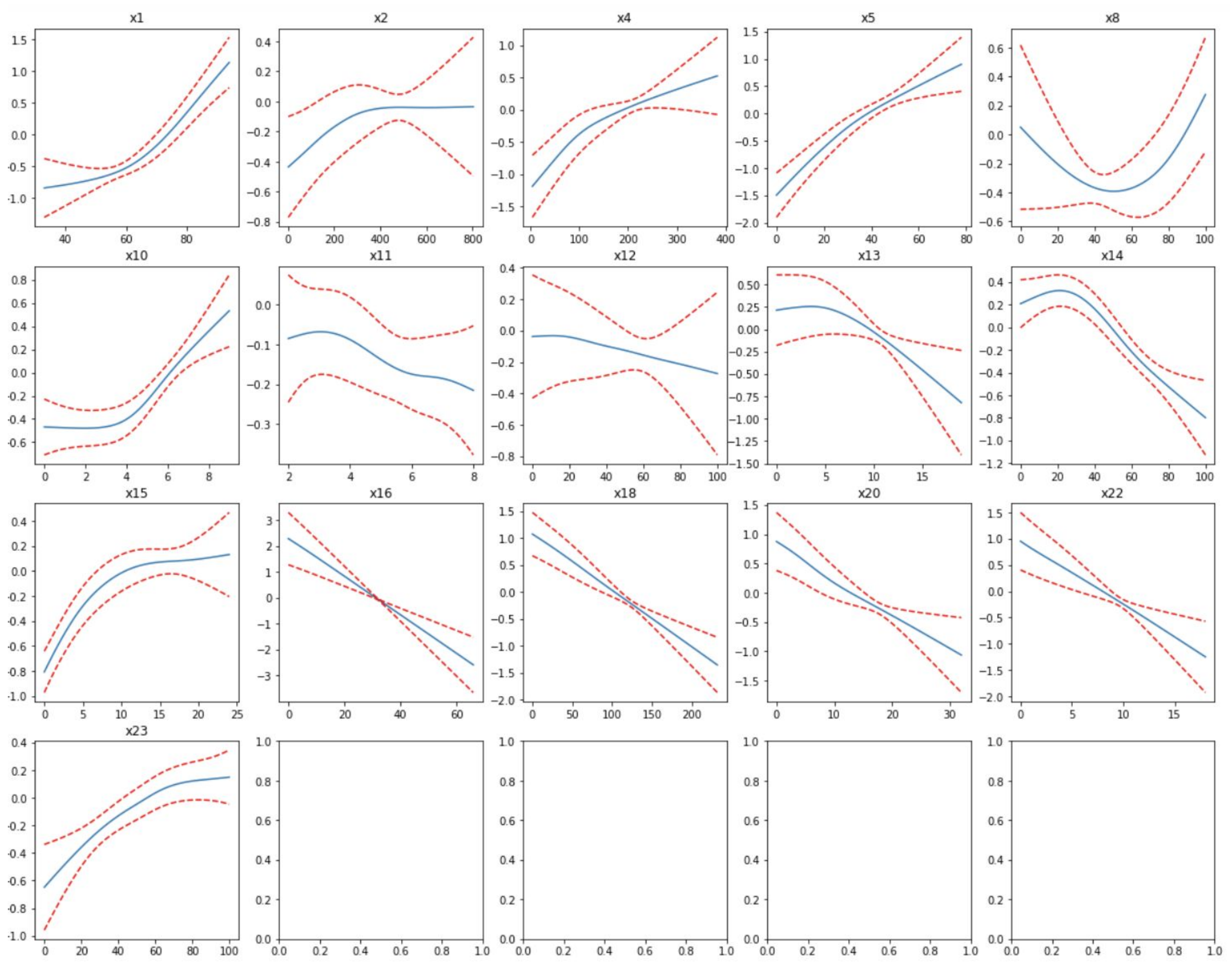


```
Optimization terminated successfully.
         Current function value: 0.543405
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                 7896
Model:                          Logit   Df Residuals:                     7879
Method:                           MLE   Df Model:                           16
Date:                Fri, 04 Dec 2020   Pseudo R-squ.:                  0.2147
Time:                        10:50:19   Log-Likelihood:                -4290.7
converged:                       True   LL-Null:                       -5464.0
Covariance Type:            nonrobust   LLR p-value:                     0.000
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -6.7018      0.393    -17.032      0.000      -7.473      -5.931
x1             0.0505      0.006      8.369      0.000       0.039       0.062
x2             0.0009      0.000      2.373      0.018       0.000       0.002
x4             0.0071      0.001      5.490      0.000       0.005       0.010
x5             0.0345      0.005      6.942      0.000       0.025       0.044
x8             0.0204      0.004      5.013      0.000       0.012       0.028
x10            0.0895      0.025      3.611      0.000       0.041       0.138
x11           -0.0155      0.021     -0.729      0.466      -0.057       0.026
x12           -0.0007      0.004     -0.194      0.846      -0.008       0.007
x13           -0.0131      0.018     -0.741      0.459      -0.048       0.021
x14           -0.0092      0.002     -4.800      0.000      -0.013      -0.005
x15            0.0493      0.007      7.049      0.000       0.036       0.063
x16           -0.0791      0.015     -5.139      0.000      -0.109      -0.049
x18           -0.0101      0.002     -6.262      0.000      -0.013      -0.007
x20           -0.0582      0.015     -3.958      0.000      -0.087      -0.029
x22           -0.1048      0.029     -3.660      0.000      -0.161      -0.049
x23            0.0032      0.002      1.704      0.088      -0.000       0.007
==============================================================================
Train accuracy_score:  0.7325227963525835
Test accuracy_score:  0.7377215189873417
```

To summarize the result, first, the likelihood ratio test p-value being close to 0 and the pseudo-R-squared value being in between the good-fit range (0.2,0.4) tells that the logistic regression model is suitable enough, capturing sufficient information. Coefficients of almost all variables except that of x11, x12 and x13 are significant in terms of the Wald test. X11,x12,x13,x14,x16,x18,x20,x22 have negative impacts on the log-odds which means if these variables are large, the loan tends to be 'Bad' with the rest features given. On the other hand, x1,x2,x4,x5,x8,x10,x15,x23 shows a positive relation on estimating the response to be 'Good'. By reading p-values, x1(Consolidated version of risk markers) has the largest absolute impact followed by x15(Months Since Most Recent Inq excl 7days), x5(Number Satisfactory Trades), x18(Net Fraction Revolving Burden), and x11(Max Delinquency Ever) has the minimal influence on the output prediction.

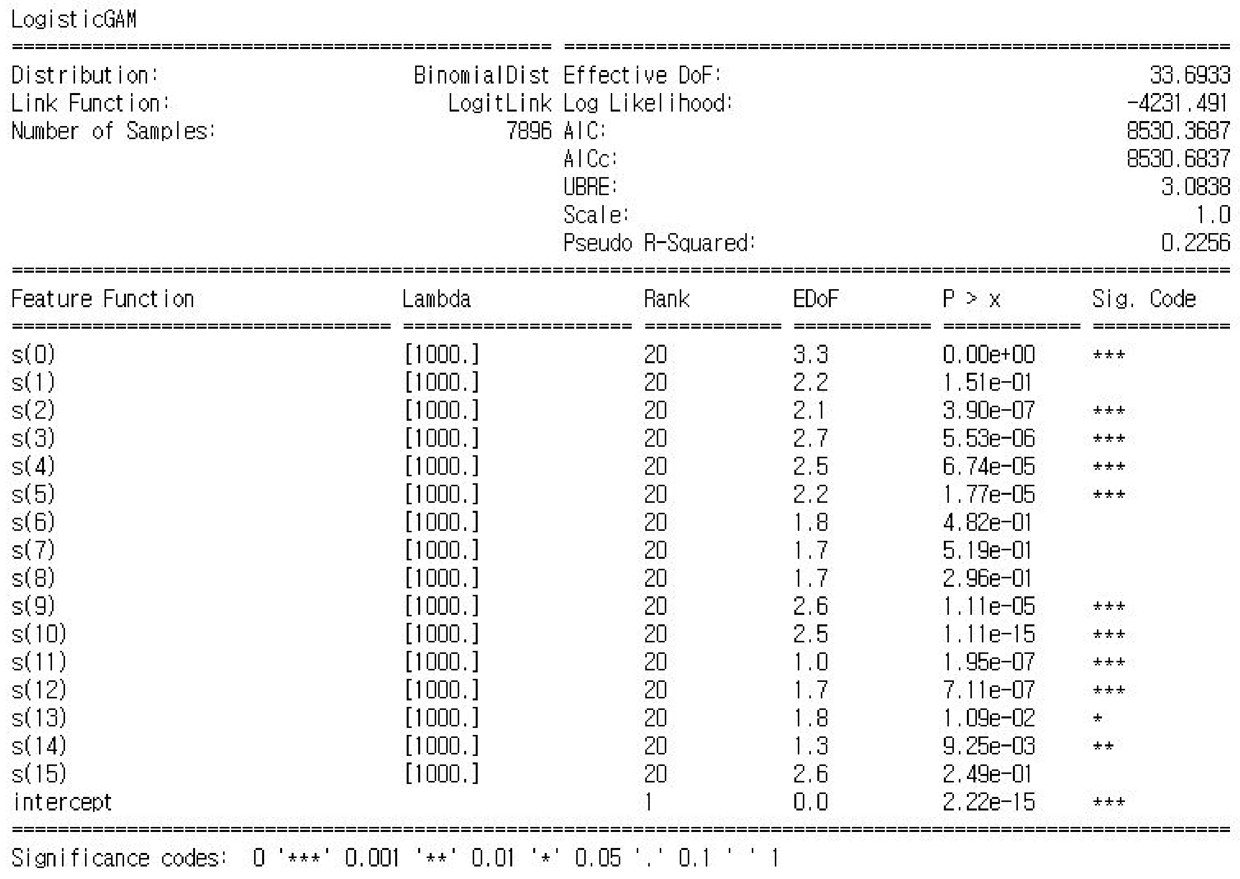
3.2. Generalized Additive Model

Next, a logistic generalized additive model (GAM) was applied to the dataset to predict the binary responses. We have used the GridSearch function to find the optimal hyperparameters and improve the accuracy score of the performance and prediction of the model. As a result, the train and test accuracy scores were found to be 0.738 and 0.739 respectively. We have also plotted the partial dependency plots to uncover the interpretability of the logistic GAM model, as shown below.



Variables that have a positive correlation with the response variable include: x1, x2, x4, x5, x8, x10, x15, x23. The higher the value is, the more likely it is to be good. On the other hand, variables that have a negative correlation with the response variable include: x11, x12, x13, x14, x16, x18, x20, x22.

Next, we have ran the summary report of the logistic GAM model without any prior knowledge (Explanation of comparisons between the logistic GAM model with or without monotonicity constraints will be explained later on in Section 4.1.) and it is shown as follows:



```
LogisticGAM
=============================================== ==========================================================
Distribution:                        BinomialDist Effective DoF:                                    33.6933
Link Function:                          LogitLink Log Likelihood:                                 -4231.491
Number of Samples:                           7896 AIC:                                            8530.3687
                                                  AICc:                                           8530.6837
                                                  UBRE:                                              3.0838
                                                  Scale:                                                1.0
                                                  Pseudo R-Squared:                                  0.2256
==========================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code
================================= ==================== ============ ============ ============ ============
s(0)                              [1000.]              20           3.3          0.00e+00     ***
s(1)                              [1000.]              20           2.2          1.51e-01
s(2)                              [1000.]              20           2.1          3.90e-07     ***
s(3)                              [1000.]              20           2.7          5.53e-06     ***
s(4)                              [1000.]              20           2.5          6.74e-05     ***
s(5)                              [1000.]              20           2.2          1.77e-05     ***
s(6)                              [1000.]              20           1.8          4.82e-01
s(7)                              [1000.]              20           1.7          5.19e-01
s(8)                              [1000.]              20           1.7          2.96e-01
s(9)                              [1000.]              20           2.6          1.11e-05     ***
s(10)                             [1000.]              20           2.5          1.11e-15     ***
s(11)                             [1000.]              20           1.0          1.95e-07     ***
s(12)                             [1000.]              20           1.7          7.11e-07     ***
s(13)                             [1000.]              20           1.8          1.09e-02     *
s(14)                             [1000.]              20           1.3          9.25e-03     **
s(15)                             [1000.]              20           2.6          2.49e-01
intercept                                              1            0.0          2.22e-15     ***
==========================================================================================================
Significance codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In the summary report above, we can see that the logistic GAM model captures sufficient information, with the pseudo-R-squared value being 0.2256, which is between the good-fit range (0.2,0.4). Furthermore, the p-value of the feature variables show that the variables of importance are x1, x15 x16, x4, x18, x5 in descending order of importance. This explains that the variable x1 (Consolidated version of risk markers) affects the output prediction the most, followed by x15 (Months Since Most Recent Inq. excluding 7 days), followed by x16 (Number of Inq. Last 6 Months) and so on.

3.3. Support Vector Machine

Support Vector Machine is known for its advantage being effective in cases of large dimension. We have applied SVM in nonlinear kernels; rbf kernel; which is using base expansion to handle the non separable classification. 'SVC' function in sklearn.svm is used. Before hyperparameter tuning, we obtained training and test accuracies 0.730 and 0.735

respectively. Proportion of support vectors used was 64.2%. After hyperparameter tuning by both GridSearch and RandomizedSearch with 5 cross validations and 100 iteration, we obtained the best parameters *{'C': 2.1277500415674804, 'gamma': 15.269235186568546}* and training accuracy 0.827. However test accuracy turned out to be 0.702 which indicates overfitting.

3.4 Decision Trees

Decision Trees use the most classical tree-based method, which is the classification and regression trees. Although the decision tree is easy to interpret, it lacks prediction performance, especially on the test set as it suffers from high variance and poor generalization performance. In this section, we implement the decision tree algorithm by using the DecisionTreeClassifier function from sklearn.tree package. Besides, we conduct hyperparameter tuning on *max_depth* in the beginning by GridSearch with 5 cross validations to obtain the best parameter value. We managed to obtain a training accuracy of 0.7213 and testing accuracy of 0.6967.

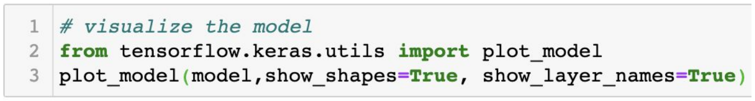
Then, we use bagging techniques to reduce the variance of the decision tree. The package we used is sklearn.ensemble.BaggingClassifier. We fit a bagging classifier with 100 base estimators in the ensemble and use out-of-bag samples to estimate the generalization error. We are able to obtain an out-of-bag samples training accuracy of 0.99 and testing accuracy of 0.725.

3.5 Random Forest

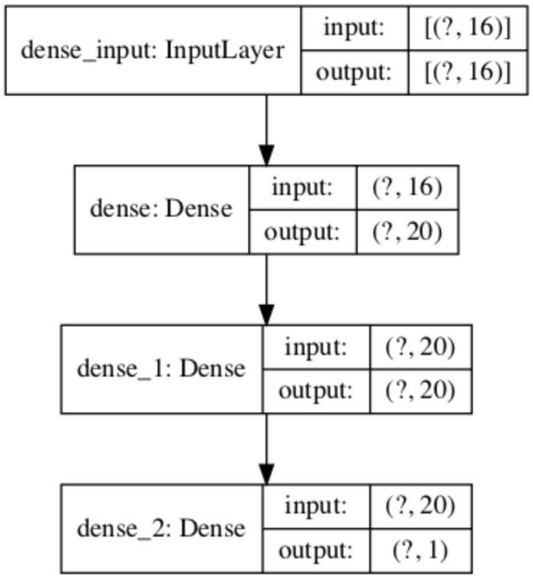
One main disadvantage of Decision Trees is that they often overfit. They perform well on training data, but they are not easy to deal with when it comes to making predictions on unseen samples. Here comes our solution: the Random Forest model. It is the ensemble of individual trees. It utilises the flexibility and power of an ensemble model as well as the simplicity of Decision Trees. We obtained training accuracy of 0.758 and testing accuracy of 0.735 respectively. We then arranged the variables in descending order of their importance, we could see that “x1 - Consolidated version of risk markers” being the most important variable, while “x11 - Max Delinquency Ever” being the least important variable.

3.6 Neural Network

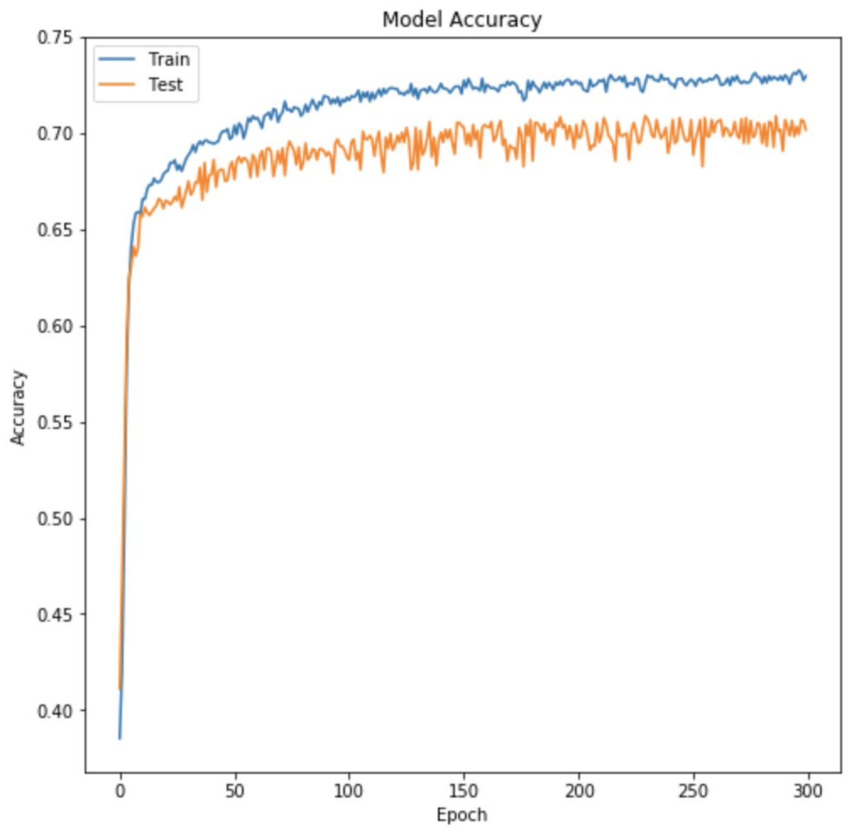
A Neural Network is a series of algorithms that aims to recognize underlying relationships among vast amounts of data through a process that mimics operations in the human brain. Neural Network models are blackbox models that involve input layers, hidden layers and output layers. Among the many types of Neural Networks, we have implemented the *sequential* neural network model onto our dataset, using the *Tensorflow Keras* model. Because there is no set rule as to how many hidden layers there should be, as well as the number of nodes in each layer, we have built several models involving different numbers of layers and nodes to determine the optimal number of layers and nodes for our model by comparing train and test accuracy scores. As a result, we decided on *2 hidden layers with 20 nodes each*, using the ReLU function for hidden layers and the sigmoid function for the output layer. Below is a visual representation of the model. The input layer involves 16 variables and the output layer prints a single result.



```
# visualize the model
from tensorflow.keras.utils import plot_model
plot_model(model,show_shapes=True, show_layer_names=True)
```



The model was compiled with the Adam optimizer, and the training and testing accuracy scores obtained were 0.715 and 0.708 respectively. The training and testing accuracy values are plotted as below.

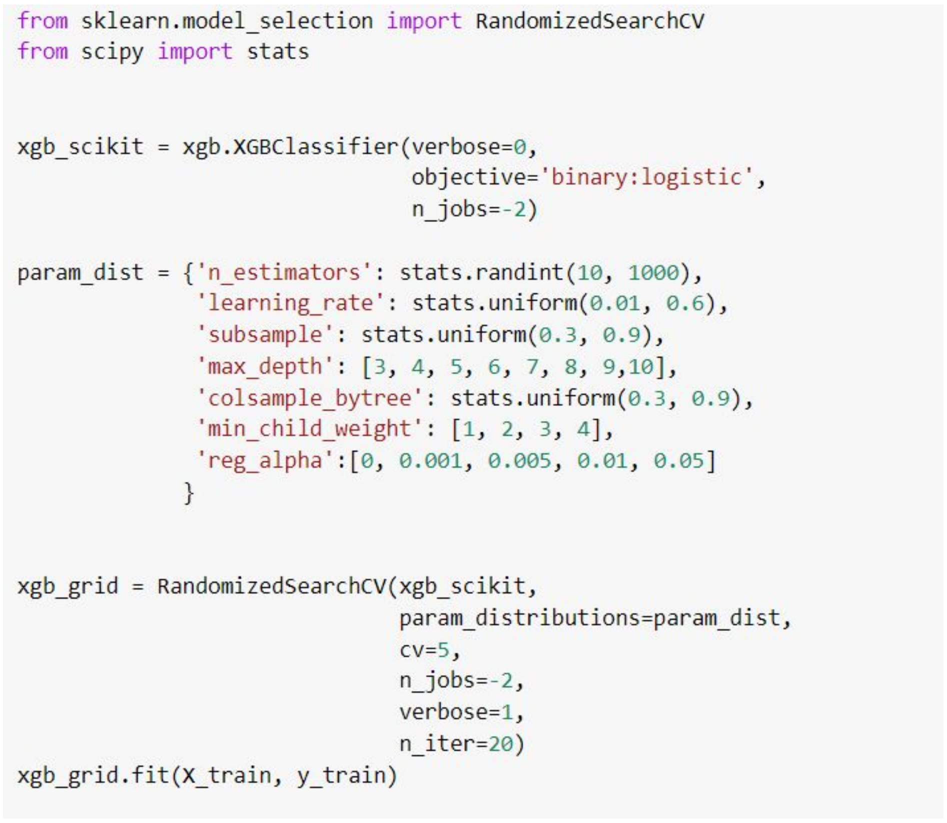


3.7. XG Boost

XGBoost is an implementation of level/depth wise gradient boosted decision trees designed for speed and performance. In order to avoid overfitting, hyperparameter tuning is important since it is the process of searching for the ideal model architecture. Parameters for tree boosters include *max_depth, min_child_weight, subsample* and *colsample_bytree.* Parameter *max_depth* explains the maximum depth of a tree. It means the higher value of it will result in a more complex and overfit model. Parameter *min_child_weight* is the minimum sum of instance weight needed in a child. If the sum of instance weight of the leaf node is less than the *min_child_weight*, the further partitioning process will be stopped.

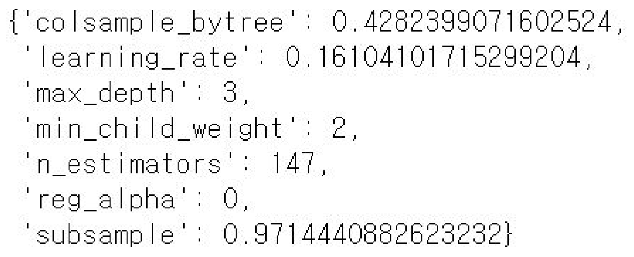
Parameter *subsample* is the ratio of training instances while *colsample_bytree* is the subsample ratio of columns when constructing each tree. Both values have to be set in order to prevent overfitting.

XGBoost Classifier was trained initially. In order to tune the best parameters, we have done a RandomSearch with 5 cross validation and 20 iterations.



```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats


xgb_scikit = xgb.XGBClassifier(verbose=0,
                               objective='binary:logistic',
                               n_jobs=-2)

param_dist = {'n_estimators': stats.randint(10, 1000),
              'learning_rate': stats.uniform(0.01, 0.6),
              'subsample': stats.uniform(0.3, 0.9),
              'max_depth': [3, 4, 5, 6, 7, 8, 9,10],
              'colsample_bytree': stats.uniform(0.3, 0.9),
              'min_child_weight': [1, 2, 3, 4],
              'reg_alpha':[0, 0.001, 0.005, 0.01, 0.05]
             }


xgb_grid = RandomizedSearchCV(xgb_scikit,
                              param_distributions=param_dist,
                              cv=5,
                              n_jobs=-2,
                              verbose=1,
                              n_iter=20)
xgb_grid.fit(X_train, y_train)
```

Next, we use the optimization methods to test the accuracy of the model with the best parameters below. The results are 0.770 for the training set and 0.728 for the testing set.



```
{'colsample_bytree': 0.4282399071602524,
 'learning_rate': 0.16104101715299204,
 'max_depth': 3,
 'min_child_weight': 2,
 'n_estimators': 147,
 'reg_alpha': 0,
 'subsample': 0.9714440882623232}
```

<u>3.8. LightGBM</u>

Light Gradient Boosting Machine is a boosting application of decision trees in leaf-wise growing approach. Such leaf-wise split algorithms can produce more complex trees and reduce more loss than level-wise algorithms(e.g. XGBoost) hence enhancing the accuracy of the model as long as the overfitting is well controlled. Therefore hyperparameter tuning was extremely important to prevent overfitting. Parameters like *max_depth* and *min_child_weight* allow direct control of the tree complexity and parameters like *subsample* and *colsample_bytree* are controlled in the purpose of adding randomness to the model so that it allows the model to be more robust to noise. Stepsize *Learning_rate* can be also adjusted accordingly.

LightGBM Classifier was trained with initial parameters as shown below and a RandomSearch with 5 cross validation and 20 iterations has been done to tune the best parameters.

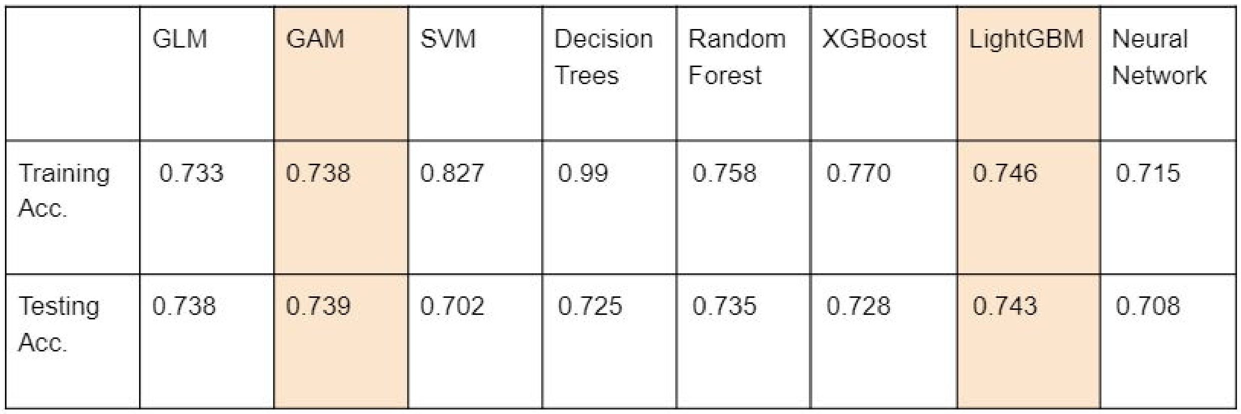
```
import lightgbm
from lightgbm import LGBMClassifier
LGBM = LGBMClassifier(n_estimators = 99,
                       learning_rate=0.075171,
                      num_leaves=10,
                      subsample=0.563710,
                      subsample_freq=1,
                      colsample_bytree=0.363497,
                      random_state = 37,
                    verbose=0,
                    force_col_wise=True)

LGBM_param_dist = {'n_estimators': stats.randint(10, 500),
              'learning_rate': stats.uniform(0.01, 0.3),
              'subsample': stats.uniform(0.3, 0.9),
              'max_depth': [3, 4, 5, 6, 7, 8, 9,10],
              'colsample_bytree': stats.uniform(0.3, 0.9),
              'min_child_weight': [1, 2, 3, 4, 5],
              'num_leaves': [4,5,6,7,8,9] }

LGBM_grid = RandomizedSearchCV(LGBM,
                             param_distributions=LGBM_param_dist,
                             cv=5,
                             verbose=-1,
                             n_jobs=-2,
                             n_iter=20)

LGBM_grid.fit(X_train, y_train)
```

As a result, training and test accuracy scores of 0.746, 0.743 were obtained with the best parameters as shown below:

```
{'colsample_bytree': 0.31245576980888967,
 'learning_rate': 0.03125094707840759,
 'max_depth': 8,
 'min_child_weight': 5,
 'n_estimators': 132,
 'num_leaves': 9,
 'subsample': 0.3083134162521744}
```

3.9. Model Selection

We then compared the performances of each model by their test accuracy scores since the ultimate goal is to make good predictions based on newly introduced rows. So it is more important to perform well with the test set than in the training set which might have given high accuracy value due to overfitting. We have selected top 2 accurate models (GAM, LightGBM; one from white box models and one from black box models) to further investigate the impact of adding monotonicity constraints to the variables. A table summarizing the model performances is shown below:



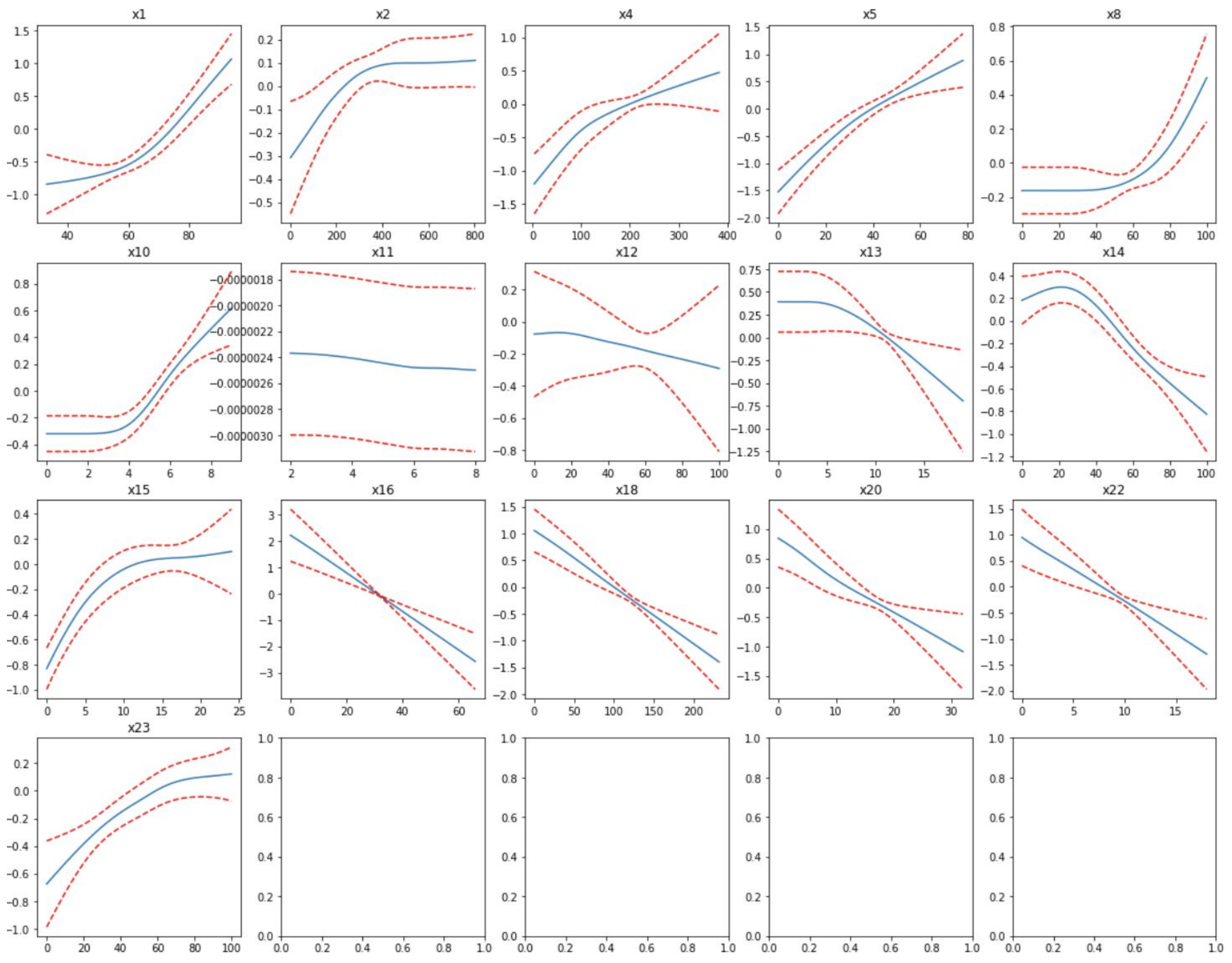
	GLM	GAM	SVM	Decision Trees	Random Forest	XGBoost	LightGBM	Neural Network
Training Acc.	0.733	0.738	0.827	0.99	0.758	0.770	0.746	0.715
Testing Acc.	0.738	0.739	0.702	0.725	0.735	0.728	0.743	0.708

4. Monotonicity Constraints

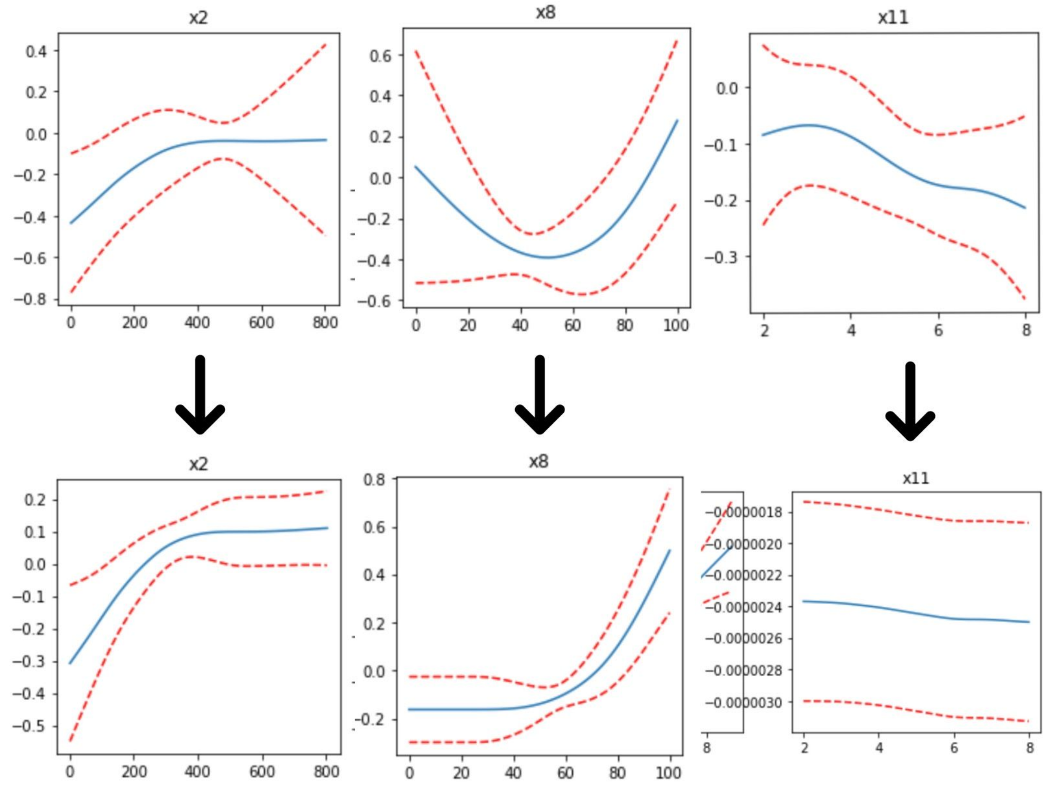
To incorporate with the Bayseian inference approach, addition of prior knowledge on relation between the response and predictor variables yields an improved prediction performance in most cases. Particularly for ‘monotonic’ increasing or decreasing patterns, it is a very useful information if confident in prior. We assume such monotonicity constraints on each variable with the respect to prob(Good) = 1.

4.1. GAM

Monotonicity constraints were added to the logistic GAM model by applying *monotonic_inc* to variables that are monotonically increasing and *monotonic_dec* to variables that are monotonically decreasing. We have then created the partial dependence plots according to the newly updated model.

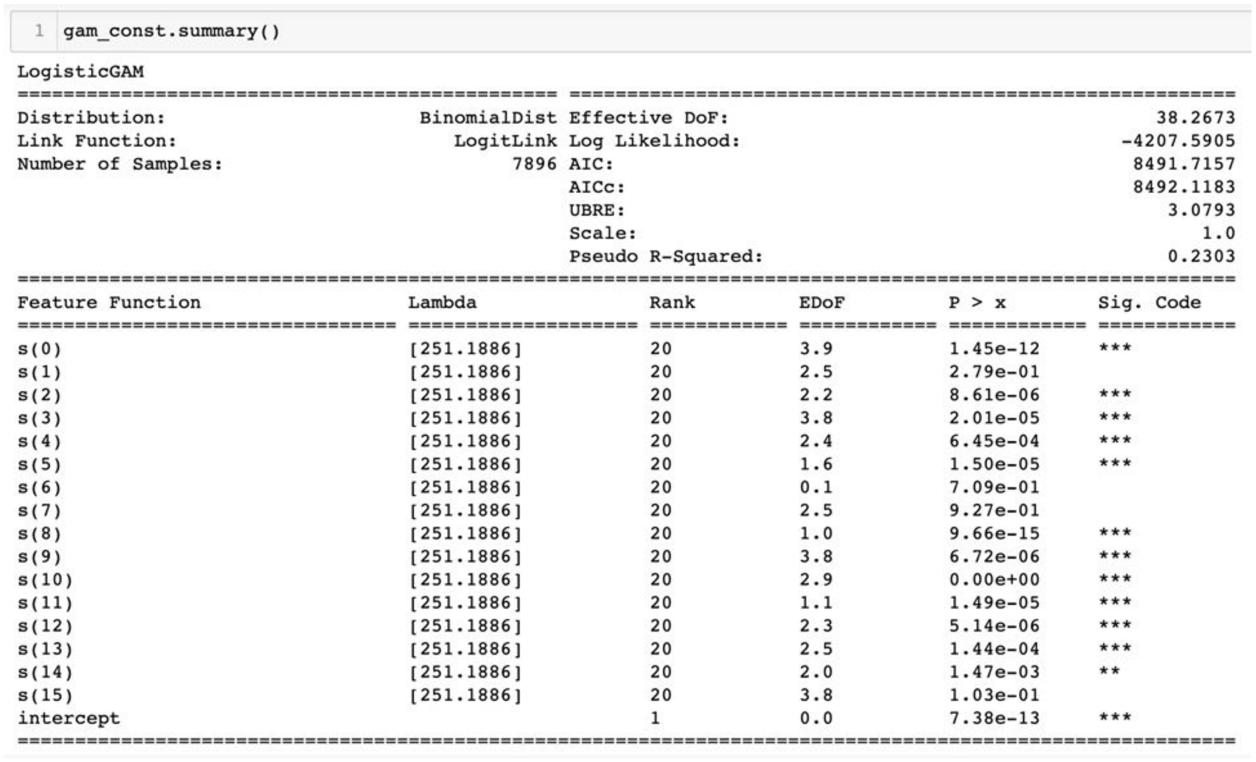


We were able to spot differences in the fitting of certain variables, such as x2, x8 and x11. With the monotonicity constraints added, these variables were able to give off better results, as can be shown in the diagram below comparing the plots with and without monotonicity constraints.



(Figure: Without constraints: top x2, x8, x11, with constraints: bottom x2, x8, x11)

As a result, the addition of the monotonicity constraints have enhanced the prediction performance and improved the model. After further implementing hyperparameter tuning with GridSearch, the final training and testing accuracy scores obtained were 0.737 and 0.739 respectively, producing a summary report below.



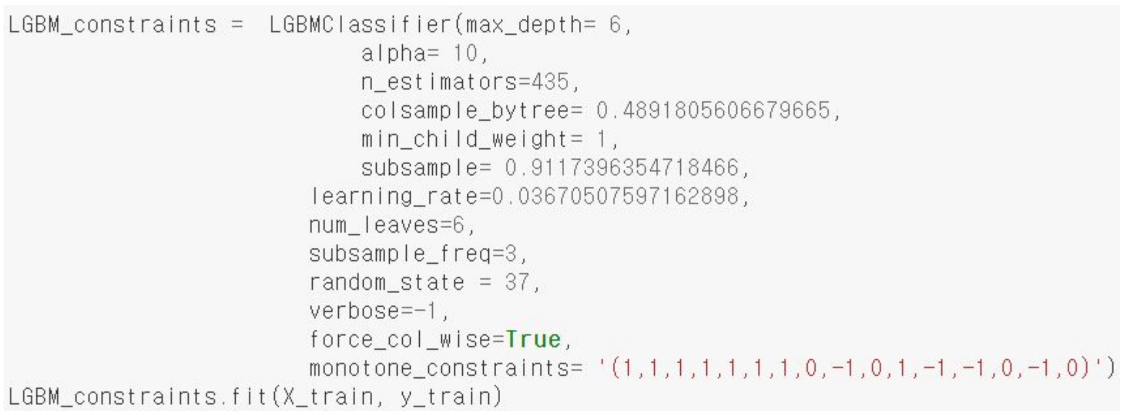
```
1  gam_const.summary()
LogisticGAM
=============================================== ==========================================================
Distribution:                        BinomialDist Effective DoF:                                     38.2673
Link Function:                          LogitLink Log Likelihood:                                 -4207.5905
Number of Samples:                           7896 AIC:                                             8491.7157
                                                  AICc:                                            8492.1183
                                                  UBRE:                                               3.0793
                                                  Scale:                                                 1.0
                                                  Pseudo R-Squared:                                   0.2303
==========================================================================================================
Feature Function                  Lambda               Rank         EDoF         P > x        Sig. Code
================================= ==================== ============ ============ ============ ============
s(0)                              [251.1886]           20           3.9          1.45e-12     ***
s(1)                              [251.1886]           20           2.5          2.79e-01
s(2)                              [251.1886]           20           2.2          8.61e-06     ***
s(3)                              [251.1886]           20           3.8          2.01e-05     ***
s(4)                              [251.1886]           20           2.4          6.45e-04     ***
s(5)                              [251.1886]           20           1.6          1.50e-05     ***
s(6)                              [251.1886]           20           0.1          7.09e-01
s(7)                              [251.1886]           20           2.5          9.27e-01
s(8)                              [251.1886]           20           1.0          9.66e-15     ***
s(9)                              [251.1886]           20           3.8          6.72e-06     ***
s(10)                             [251.1886]           20           2.9          0.00e+00     ***
s(11)                             [251.1886]           20           1.1          1.49e-05     ***
s(12)                             [251.1886]           20           2.3          5.14e-06     ***
s(13)                             [251.1886]           20           2.5          1.44e-04     ***
s(14)                             [251.1886]           20           2.0          1.47e-03     **
s(15)                             [251.1886]           20           3.8          1.03e-01
intercept                                              1            0.0          7.38e-13     ***
==========================================================================================================
```

In the summary report above, we can see that the logistic GAM model captures sufficient information, with the pseudo-R-squared value being 0.2303, which is between the good-fit range (0.2,0.4). Furthermore, the p-value of the feature variables show that the variables of importance are x15, x13,x1, x18, x14, x4 in descending order of importance. This explains that the variable x15 (Months Since Most Recent Inq. excluding 7 days) affects the output prediction the most, followed by x13 (Number of Trades Open in Last 12 Months), followed by x1 (Consolidated version of risk markers) and so on.

<u>4.2. LightGBM</u>

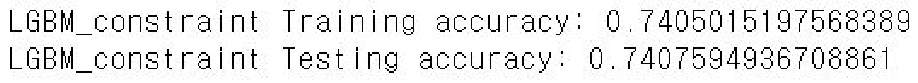
In tree-based models, addition of monotonicity constraints takes a crucial role as a decision rule when branching out nodes in a tree. The underlying algorithm is as follows. Every child node should follow a monotonic pattern from its sibling, parent and its ancestor nodes. For instance, if it is a monotonic increasing relationship, the weight assigned to the right child node must be greater than the weight on the left child, parent node and all the previous nodes. Any candidate that violates such a relationship is replaced by negative infinity so that the split is abandoned, and the search continues until it finds the next optimal split. The monotonic relationship is then able to be kept throughout the descendant nodes.

XGBoost and LightGBM models have a parameter called *‘monotone_contraints’* in ‘train’ function that enables a monotonic relationship to be fed. A list of constraint directions indicated with -1,0 and 1 are fed to our LightGBM model and the result is as follows:

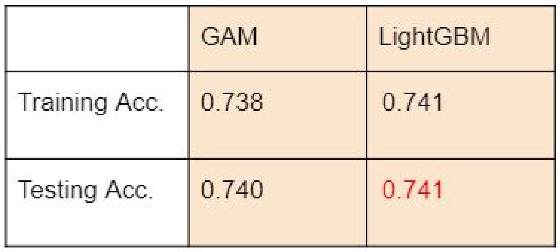


```
LGBM_constraints =  LGBMClassifier(max_depth= 6,
                           alpha= 10,
                           n_estimators=435,
                           colsample_bytree= 0.4891805606679665,
                           min_child_weight= 1,
                           subsample= 0.9117396354718466,
                       learning_rate=0.03670507597162898,
                       num_leaves=6,
                       subsample_freq=3,
                       random_state = 37,
                       verbose=-1,
                       force_col_wise=True,
                       monotone_constraints= '(1,1,1,1,1,1,1,0,-1,0,1,-1,-1,0,-1,0)')
LGBM_constraints.fit(X_train, y_train)
```

After conducting hyper-parameter tuning by RandomizedSearch, the final training and test accuracy scores obtained were 0.741 and 0.741 respectively, which is slightly higher accuracy than the Logistic GAM model, therefore we decided to conduct post-hoc interpretation on LightGBM model and to compare with white box model interpretation of GAM.



```
LGBM_constraint Training accuracy: 0.7405015197568389
LGBM_constraint Testing accuracy: 0.7407594936708861
```



	GAM	LightGBM
Training Acc.	0.738	0.741
Testing Acc.	0.740	0.741

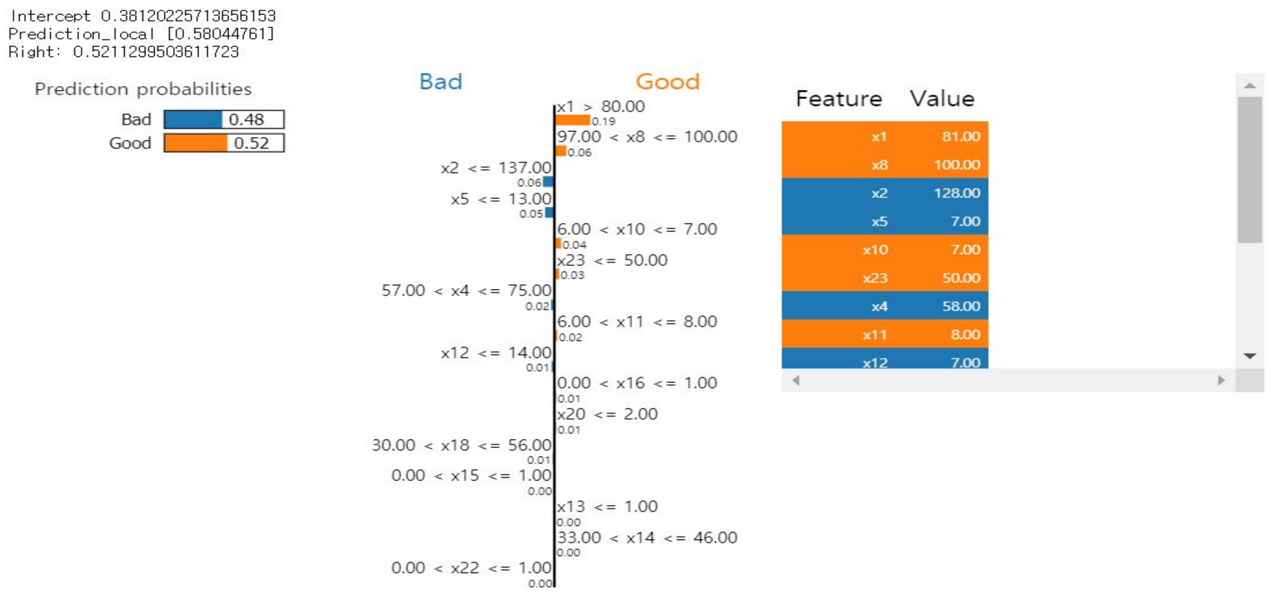
5. LightGBM post-hoc interpretation

We are interpreting the LightGBM model in two ways: overall interpretation and local interpretation. In local interpretation, we are looking into a given data point and its prediction results, thus observing which variables result in this specific prediction. While in overall interpretation, we are observing which variables have the highest and lowest predictive powers.

5.1. Local Explainability

5.1.1. LIME

We see that the prediction probabilities for bad is 0.48 and that for good is 0.52. It means that the model is 52% confident to be good. Features like "x1 - consolidated version of risk markers" and "x8 - percent trades never delinquent" increase its chance to be classified as good while features like "x5 - number of satisfactory trades" and "x2 - months since oldest trade open" increase its chance to be classified as bad.

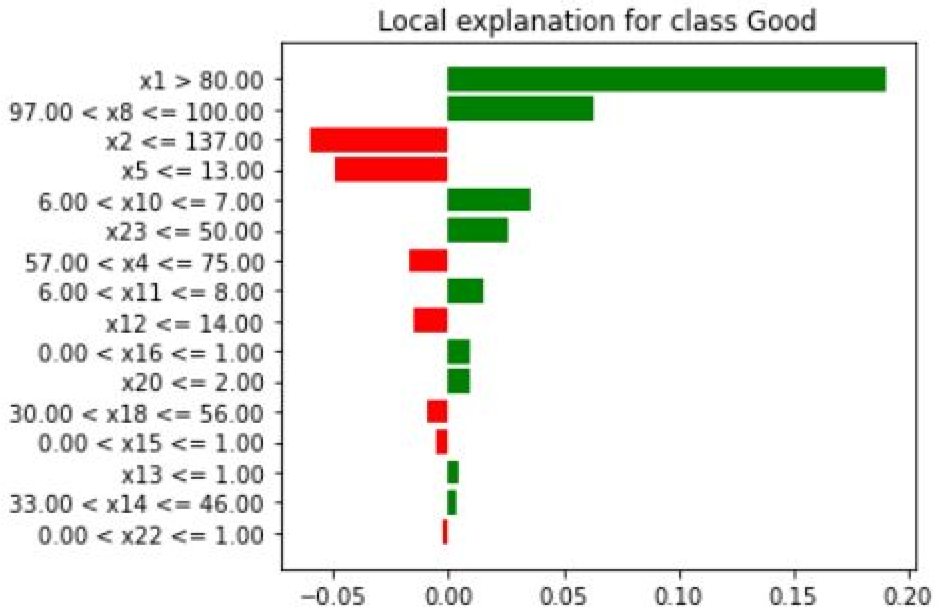


```
Intercept 0.38120225713656153
Prediction_local [0.58044761]
Right: 0.5211299503611723
```

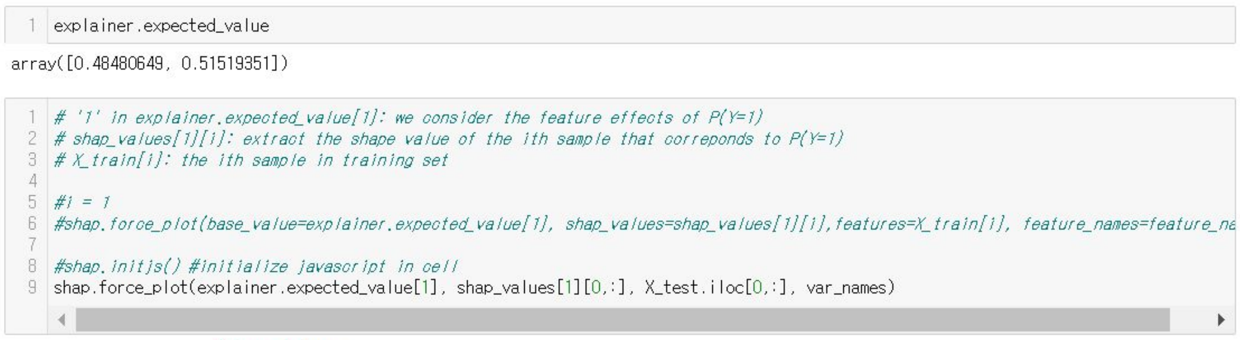
Feature	Value
x1	81.00
x8	100.00
x2	128.00
x5	7.00
x10	7.00
x23	50.00
x4	58.00
x11	8.00
x12	7.00

In the following graph, features which are positively correlated with the target are shown in green while features which are negatively correlated with the target are shown in red.

In more than 80.00, we have "x1 - consolidated version of risk markers" shown in green, which means it is positively correlated with high credit for good. And "x8 - percent trades never delinquent" between 97 and 100 is also in green, which indicates that it is also positively correlated with the “good” class. Moving on we have “x2 - months since oldest trade open” and “x5 - number of satisfactory trades” shown in red, this shows that they are negatively correlated with credit for good.



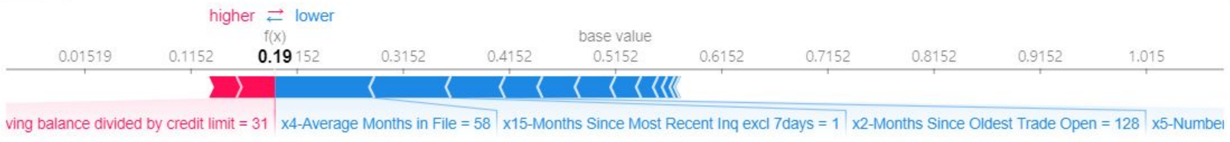
5.1.2. SHAP



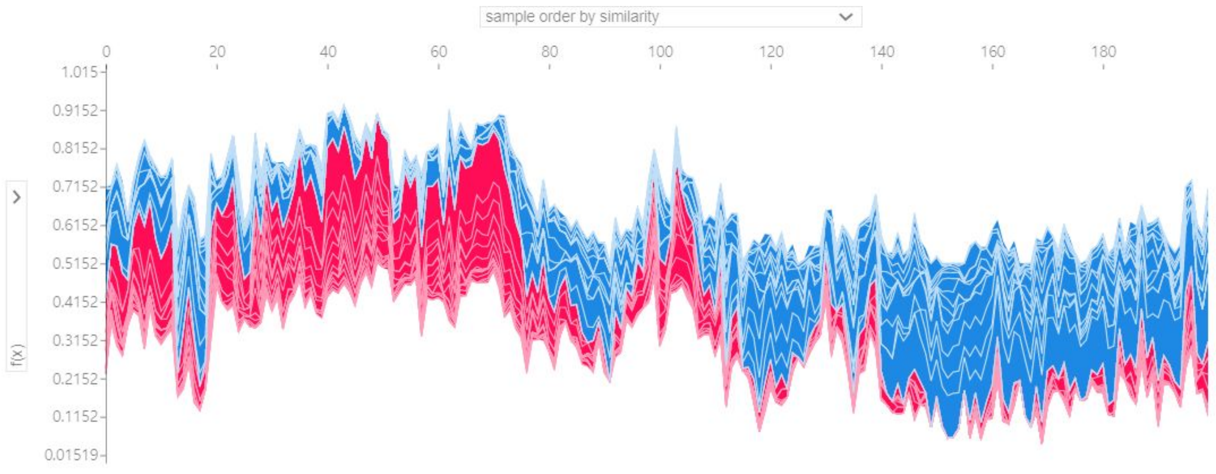
```
explainer.expected_value
```

```
array([0.48480649, 0.51519351])
```

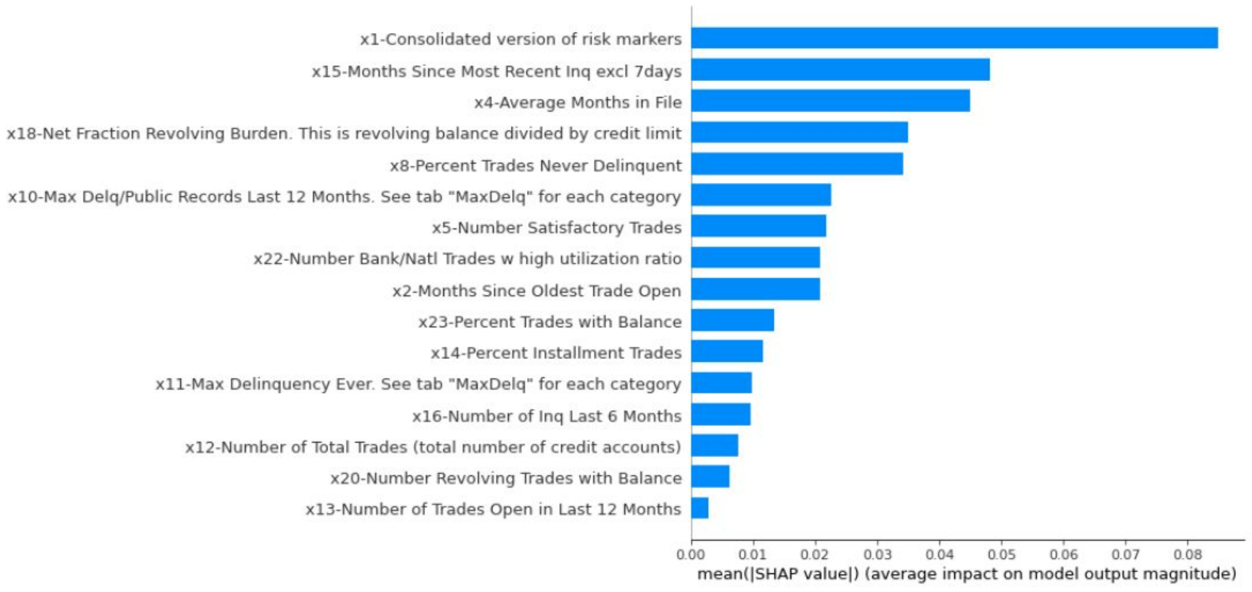
```
# '1' in explainer.expected_value[1]: we consider the feature effects of P(Y=1)
# shap_values[1][i]: extract the shape value of the ith sample that correponds to P(Y=1)
# X_train[i]: the ith sample in training set

#i = 1
#shap.force_plot(base_value=explainer.expected_value[1], shap_values=shap_values[1][i],features=X_train[i], feature_names=feature_na

#shap.initjs() #initialize javascript in cell
shap.force_plot(explainer.expected_value[1], shap_values[1][0,:], X_test.iloc[0,:], var_names)
```



The features in red color bring positive influences, which means they drag the prediction value closer to 1, while features in blue color bring negative influences, they drag the prediction value away from 1. We have a base value of 0.5152, which is the average of all the output values of the model on the training. The output value we have is 0.19.

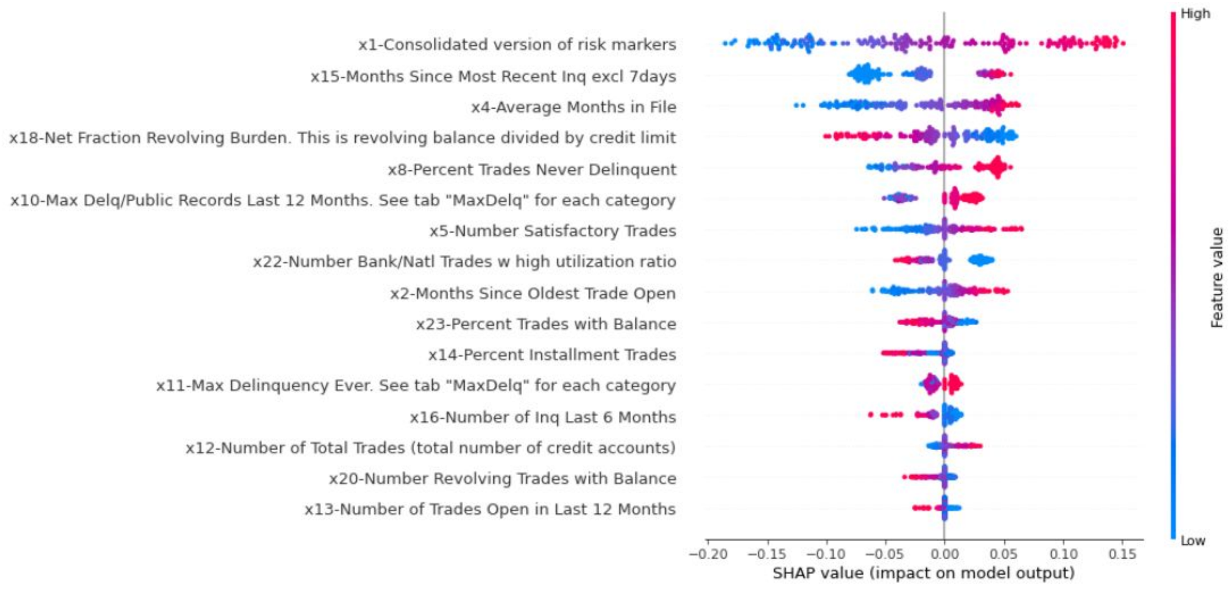


From the figure below, we can see that “x1 - consolidated version of risk markers" has the highest mean SHAP value, which means it has the highest average impact on the model output variable. While “x13 - number of trades open in the last 12 months” is having the lowest mean SHAP value, its average impact on the model output variable is the lowest.

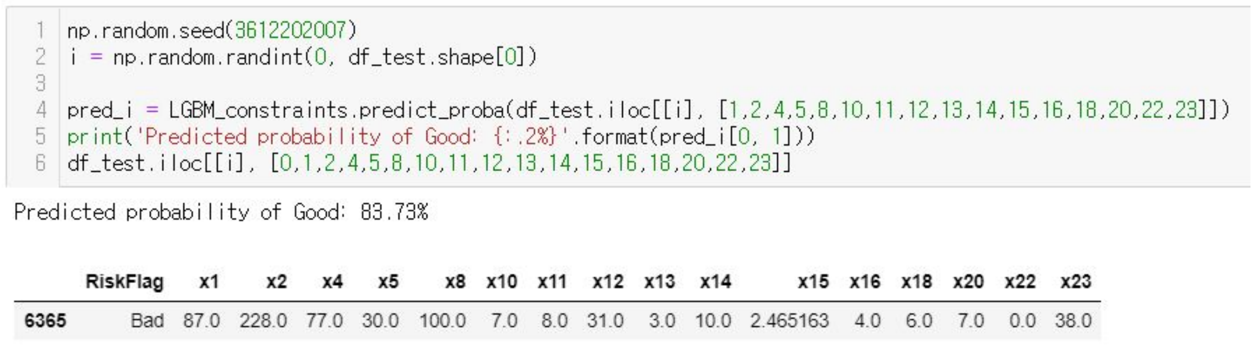


The SHAP value plot below can further show the positive and negative relationships of the predictors with the target variable. The variables are ranked in descending order according to their importance. The horizontal location shows whether the effect of that value is associated with a higher or lower prediction, while the colors show whether that variable is of high (represented in red) or low (represented in blue) influence for that observation.

A high level of the “x1 - consolidated version of risk markers” has a high and positive impact on the model output. The “high” comes from the red color, and the “positive” impact is shown on the X-axis. Similarly, we will say the “x13 - number of trades open in the last 12 months” is having a low impact and it is negatively correlated with the target variable.



Let us observe an instance of a customer and the prediction made based on his/her information. According to the local variable importance, x1(Consolidated version of risk markers), x15(Months Since Most Recent Inq. excluding 7 days) and x4(Average Months in File) would have influenced the most in making such a prediction. X1 would have increased the chance of prediction to ‘Good’ whereas x15 and x4 would have decreased the probability. In this instance, the probability of his/her RiskFlag to be ‘Good’ is predicted to be 83.73%.



```
np.random.seed(3612202007)
i = np.random.randint(0, df_test.shape[0])

pred_i = LGBM_constraints.predict_proba(df_test.iloc[[i], [1,2,4,5,8,10,11,12,13,14,15,16,18,20,22,23]])
print('Predicted probability of Good: {:.2%}'.format(pred_i[0, 1]))
df_test.iloc[[i], [0,1,2,4,5,8,10,11,12,13,14,15,16,18,20,22,23]]
```

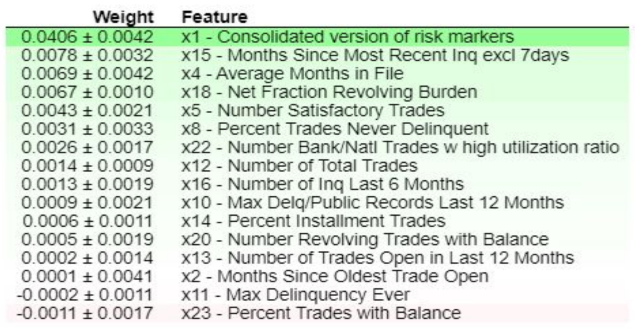
```
Predicted probability of Good: 83.73%
```

	RiskFlag	x1	x2	x4	x5	x8	x10	x11	x12	x13	x14	x15	x16	x18	x20	x22	x23
6365	Bad	87.0	228.0	77.0	30.0	100.0	7.0	8.0	31.0	3.0	10.0	2.465163	4.0	6.0	7.0	0.0	38.0

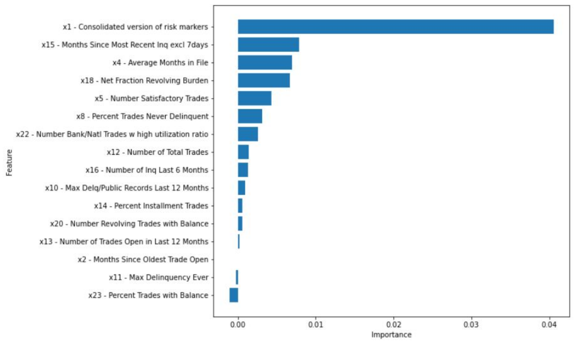
5.2. Global Explainability

5.2.1. Variable Importance

We first look at the global explainability by Variable Importance, the features were arranged in descending order of their weights. The top feature “x1 - consolidated version of risk markers” is having the highest weight compared to the rest of the 16 chosen features, being around 0.0406. “x23 - percent trades with balance” in contrast is having the lowest weight of -0.0011. The Variable Importance is shown in both table form and graph form as follow:



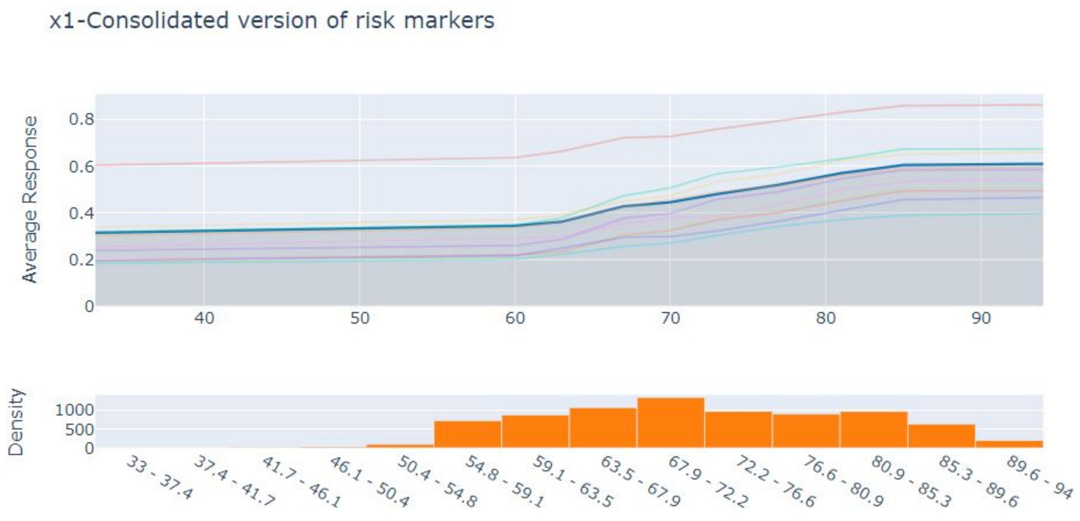
Weight	Feature
0.0406 ± 0.0042	x1 - Consolidated version of risk markers
0.0078 ± 0.0032	x15 - Months Since Most Recent Inq excl 7days
0.0069 ± 0.0042	x4 - Average Months in File
0.0067 ± 0.0010	x18 - Net Fraction Revolving Burden
0.0043 ± 0.0021	x5 - Number Satisfactory Trades
0.0031 ± 0.0033	x8 - Percent Trades Never Delinquent
0.0026 ± 0.0017	x22 - Number Bank/Natl Trades w high utilization ratio
0.0014 ± 0.0009	x12 - Number of Total Trades
0.0013 ± 0.0019	x16 - Number of Inq Last 6 Months
0.0009 ± 0.0021	x10 - Max Delq/Public Records Last 12 Months
0.0006 ± 0.0011	x14 - Percent Installment Trades
0.0005 ± 0.0019	x20 - Number Revolving Trades with Balance
0.0002 ± 0.0014	x13 - Number of Trades Open in Last 12 Months
0.0001 ± 0.0041	x2 - Months Since Oldest Trade Open
-0.0002 ± 0.0011	x11 - Max Delinquency Ever
-0.0011 ± 0.0017	x23 - Percent Trades with Balance



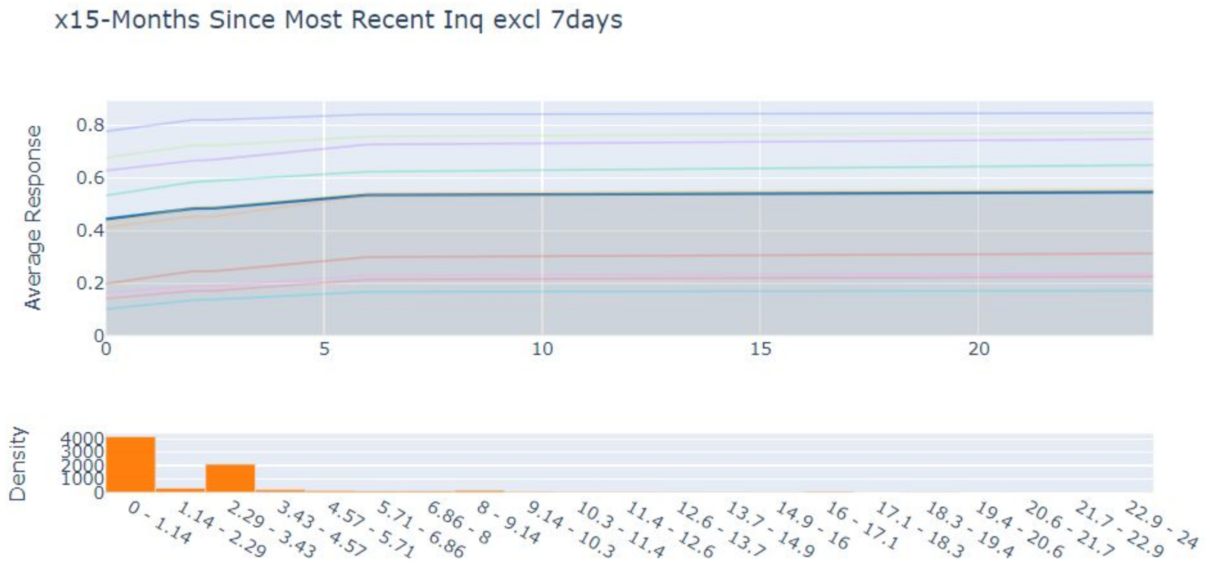
5.2.2. Partial Dependence Plot

In our global interpretation by PDP, we have included the top 4 features with Variable Importance larger than 0.015 as there is a relatively huge gap between the VI of the 4th and 5th variables.

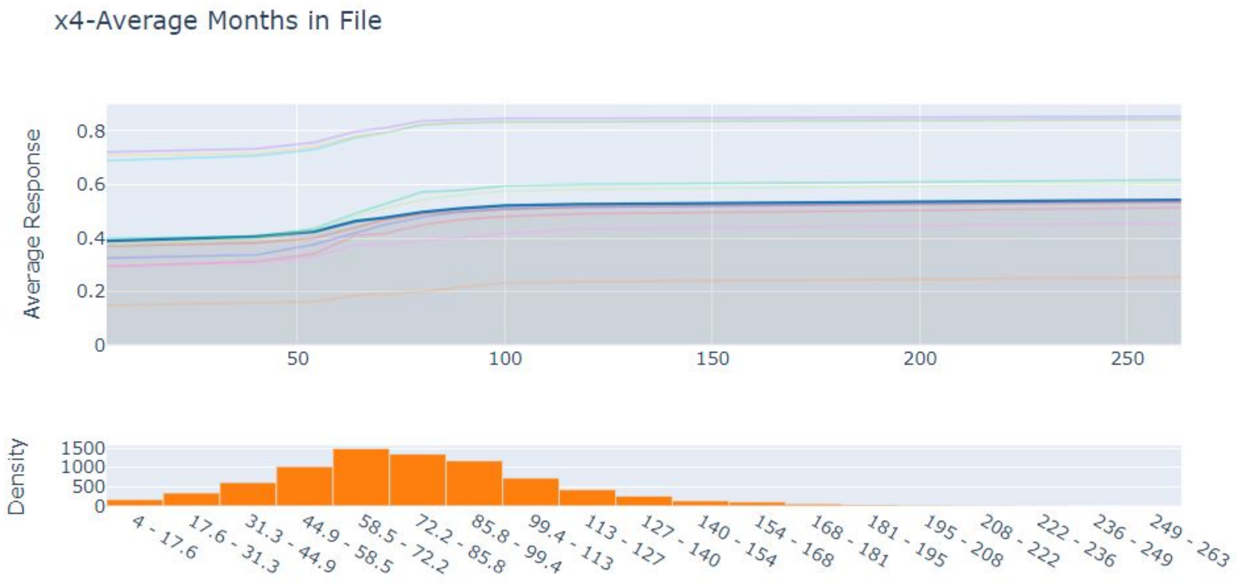
The top feature being “consolidated version of risk markers”, its average response decreases as it moves across the x-axis and the density shows a similar trend.



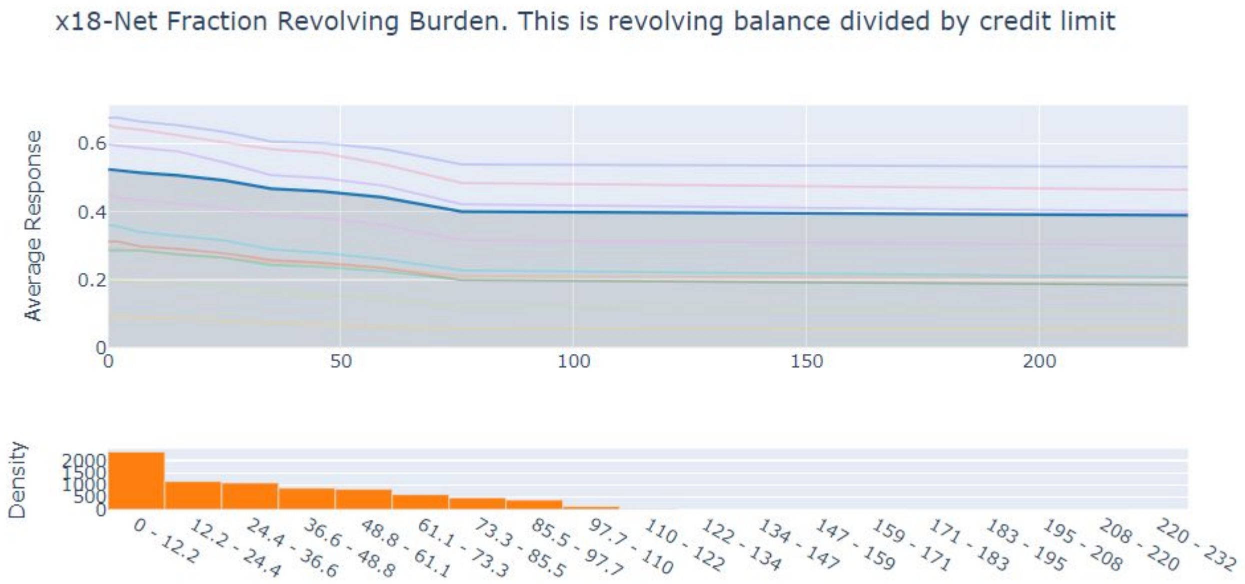
The second feature is “months since most recent inq excluding 7days”, the average response increases gradually while its density shows almost a normal distribution skewed to the right.



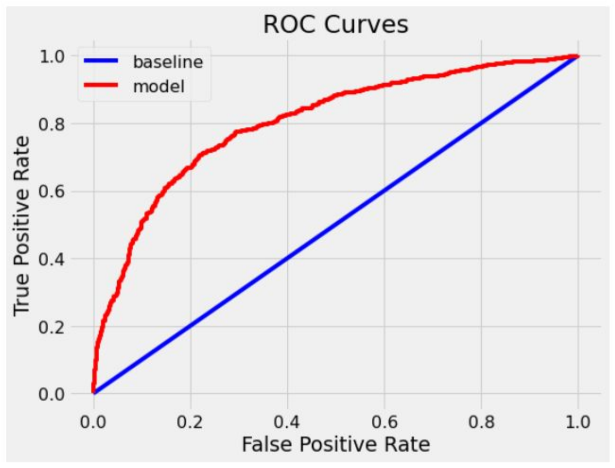
In the third graph we have “average months in file”, the average response decreases initially, and fluctuates a little with the pattern of the density graph when it shows a normal distribution graph skewed to the left.

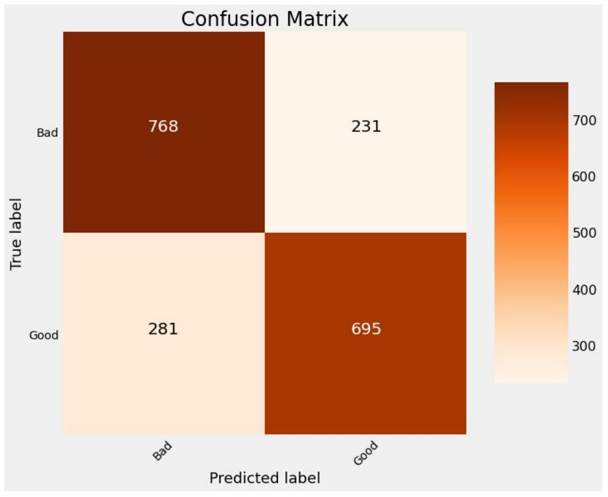


The fourth feature “net fraction revolving burden” shares a similar but more obvious increment than our second feature, and the density graph is skewed to the right.



```
Recall Baseline: 1.0 Test: 0.71 Train: 0.7
Precision Baseline: 0.49 Test: 0.75 Train: 0.74
Roc Baseline: 0.5 Test: 0.8 Train: 0.81
```





The ROC improves both the testing and training sets to 0.8 and 0.85 from 0.75 and 0.74 respectively. Looking at the ROC curves on the left, we achieve 1.0 for both models. The red curve which is to the top and left is better for the model. We can also use the confusion matrix for the testing predictions. The figure at the right shows the predictions of the model were correct in the top left (768) and bottom right (695) corners, and the predictions of the model in the lower left and upper right corners were incorrect. The high value in our ROC Curve and the power in the confusion matrix further justified that our model is good enough.

6. Conclusion

From results, it was shown that the LightGBM model is the best machine learning model in predicting the home equity loan performance. For interpretations based on the LightGBM model, it turns out that x1 and x15 are the most important features for a majority of our interpretations, and they both bring positive impacts while x23 brings the most negative impact. Hence, when a person's 'consolidated version of risk markers' and 'months since most recent inq excl 7 days' values are high, we need to be vigilant before approving a loan. The logistic GAM model has also shown the top variables of importance to be x15, x13 and x1, which show similar results with the LightGBM model. Hence, we can verify that 'months since most recent inq excl 7 days' and 'consolidated version of risk markers' clearly largely impact the output prediction.

Our results also show that having prior knowledge on monotonicity constraints and applying them to the model are helpful in improving the prediction accuracies and performance of the model. It would be helpful to have more definite knowledge about the explainable variables to improve the model performance in the future. The more information to be fed into the model as a prior knowledge, the more robust the prediction will be.