Introduction

In this report we analyse the “All Lending Club loan data”, in order to build a model to predict whether a customer will default on their loan. After the initial data pre-processing and feature engineering phase, we compare the performance of different classification models in terms of their ability to correctly predict loan defaults.

Data pre-processing

Our dataset represents loans that have been accepted from Dec 2014 to Jan 2016. The current status of the loan is given in the “charged-off” column (our variable of interest), which takes value 1 if a debtor is not repaying the loan and 0 otherwise. The dataset is split into the training and testing set based on a standard 70:30 split. To simulate realistic results for the test set, the data were sorted based on the issue date of the loan. 30% of the most recent loans were used as the test set, while 70% of the earlier loans were used to train the model. This division is an attempt to replicate the situation where you have past information and are building a model which you will test on the “future” as-yet unknown information.

Our dataset contains 20 features. All numeric features are normalised by subtracting the mean and scaling to unit variance. We first normalize the training set, and then we use that same (training) mean and variance to normalize the test set. This process ensures that no information about the test set is contained in the scaler which could bias the performance of our models. Numerical features with missing data are filled by mean imputation. Categorical features with missing data are filled by using the most frequent value along each column. To avoid any data leakage, the imputation of missing values is done on the training set and test set separately.

Another transformation applied, is the *one hot encoding* which converts categorical data to non-ordinal features. The transformations above, are sequentially applied using the Pipeline operator in the *sklearn* package. The purpose of the pipeline is to assemble several steps that can be cross validated together while setting different parameters.

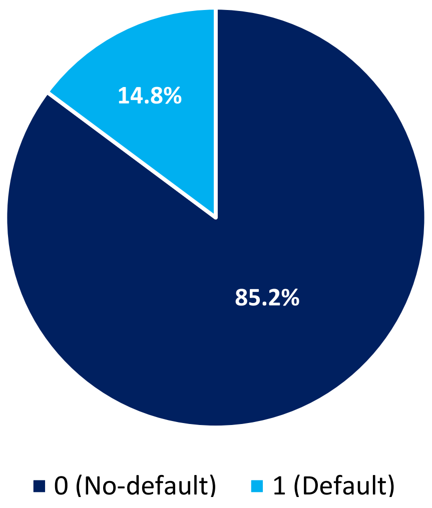
Exploratory data analysis

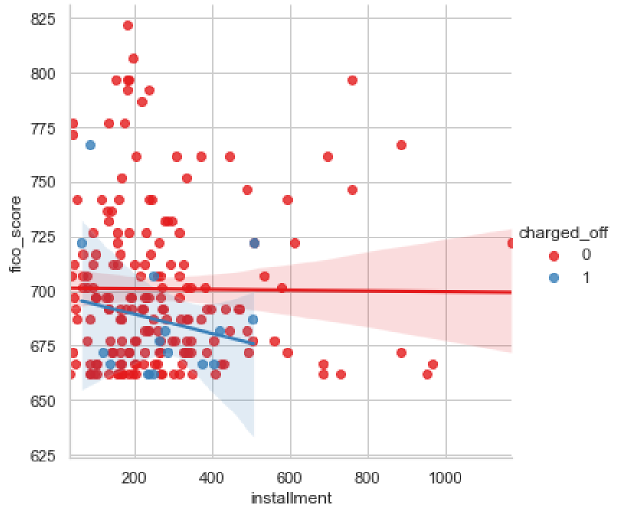
Two key insights arise from our exploratory data analysis:

Class Imbalances

A simple count of our variable of interest, suggests that our data is dominated by class 0 observations (i.e., 85.2% of the total loans belong to the no-default category). This implies that the minority class (i.e., defaulting loans), is significantly underrepresented, thus giving rise to imbalanced classification. The challenge of working with imbalanced datasets is that there are too few examples of the minority class for a model to effectively learn the decision boundary, which can lead to poor predictive performance on the minority class (although it is often the performance in the minority class that is the most important). In our case, the overrepresented class is that of fully paid loans, which can be a problem because we are more concerned with predicting defaulting loans well rather than with misclassifying a fully paid loan.

Figure 1: Class imbalances





Note: The figure on the LHS illustrates the dominance of class 0. On the RHS, we have plotted (for illustrative purposes) the first 500 observations of the training set as a function of two important features. Class 1 is underrepresented, and the decision boundary between the two classes is not clear.

We address this by employing a method of oversampling called **SMOTE** (Synthetic Minority Oversampling Technique).[1] The rationale of SMOTE is:

- first select a minority class instance x_i at random and find its *K* nearest minority class neighbours. The nearest neighbour algorithm is adopted to calculate the *K* nearest neighbours of each minority sample with Euclidean distance as the standard.
- create a synthetic instance of the minority class x_{new} by selecting one of the k-nearest neighbours x_n at random and connecting these two points to form a line segment in the feature space. The synthetic instances are generated as a convex combination of the two chosen instances: $x_{new} = x_i + rand(0,1) * |x_i - x_n|$

This technique allows us to achieve the following class distribution: 60% for majority class and 40% for minority class.

Feature Engineering

Multicollinearity

We have estimated the Pearson correlation coefficient for all the features in our dataset. Using a correlation threshold of 0.70 we have observed strong correlation between the following pairs: *i) installment and loan amount* (correlation 0.96*); ii) pub_rec_bankruptcies and pub_rec* (correlation 0.73). To eliminate multicollinearity, we carry out feature selection because giving priority to the features with high relevance to the target and removing irrelevant features can reduce the difficulty of learning. Across the pairs of highly correlated features, we keep only the most important ones (namely, *installment and pub_rec_bankruptcies). (see Appendix A1 for further details)*

Feature engineering using PCA and domain knowledge

[1] Zhu et al. (2019) "A study on predicting loan default based on the random forest algorithm"

We use a combination of our domain knowledge and PCA as a model-based method for feature engineering. We have run the PCA analysis using the five most important features. The signs and magnitudes of a component's loadings tell us what kind of variation is captured by each feature. *(see Appendix A2 for further details)*

To express the contrasts between features that have the highest explained variance vs. the ones that have the lowest explained variance for each component of the PCA we construct the following ratios:

- **annual inc/fico score**: represents the ratio of annual income per unit of fico score. The greater the ratio the lower the risk of default.
- **instalment/fico score**: represents the ratio of loan instalment per unit of fico score. The greater the ratio the higher the risk of default.
- **annual inc/instalment**: represents the instalment coverage by the annual income. The greater the value of this ratio, the lower the risk of default.
- **annual inc/total_acc** : represents the coverage of the total loan account by the annual income. The greater the value of this ratio, the lower the risk of default.

Methodology

After the initial data pre-processing stage, we first build three vanilla classification models which will be trained on the imbalanced data and the set of existing features. These will be our baseline models. Then, we will correct for class imbalances and we will use the insights from the feature engineering process, to enhance the performance of our baseline classification models. Finally, we will optimise the hyperparameters of each model and we will select the best model for the final out of sample testing.
The AUC-ROC is the scoring parameter that we aim to maximize in the cross-validation process when training the models. As indicated by *Turiel and Aste*[2],this metrics can have some drawbacks as it can lead to overfitting the majority class. This is due to AUC-ROC weighting accuracy as an average over predictions. Therefore, we will further cross-check our results using a complementary metrics, such as the unweighted recall average.

First step: Baseline Classification models

We will try the following three classification models on the train set with repeated K-fold cross validation 10-times with different randomisation in each repetition. *(the training set will be divided into 5 subsets, and for each iteration one subset is taken as the validation set and the remaining subsets as the training sets).*

- **Logistic Regression**: a linear algorithm that predicts the probability of a categorical dependent variable. The logistic regression function $p(\mathbf{x})$ is the sigmoid function which is often interpreted as the predicted probability that the output is equal to 1. The optimal parameters are determined by the method of the maximum likelihood estimation**.**
- **Decision Tree Classifier**: a supervised learning algorithm built through binary recursive partitioning. This is an iterative process of splitting the data into partitions, and then splitting it up further until the process stops when the algorithm determines the data within the subsets are sufficiently homogenous, or another stopping criterion has been met. We start the process at the tree root and split the data on the feature that results in the largest information gain (IG) (reduction in uncertainty towards the final decision).

[2] Turiel and Aste (2020): “Peer-to-peer loan acceptance and default prediction with artificial intelligence”

- **Random Forest:** is an ensemble learning method that operates by constructing a multitude of decision trees, usually trained with the "bagging" method. The general idea of the bagging method is that a combination of learning models increases the overall result, outputting the class that is the mode of the classes or means prediction of the individual trees.

Second step: Correct for class imbalances, address multicollinearity, and feature engineering

We then apply the correction suggested in Part (ii) to our baseline classification models. Specifically, we correct for class imbalances, address multicollinearity, and engineer new features.

Third step: Hyperparameter Optimisation

To further refine the models, we do a randomised search over a set of hyperparameter values for each model. Due to data scarcity in this domain, training and test sets alone were used in the analysis, with hyperparameter tuning performed through cross-validation. Hyperparameter tuning for these models was performed through extensive *randomised searches* over parameter settings. In contrast to grid search, in randomised search not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified values. This adjustment helps with the computation speed. (*See Appendix A3 for further details)*

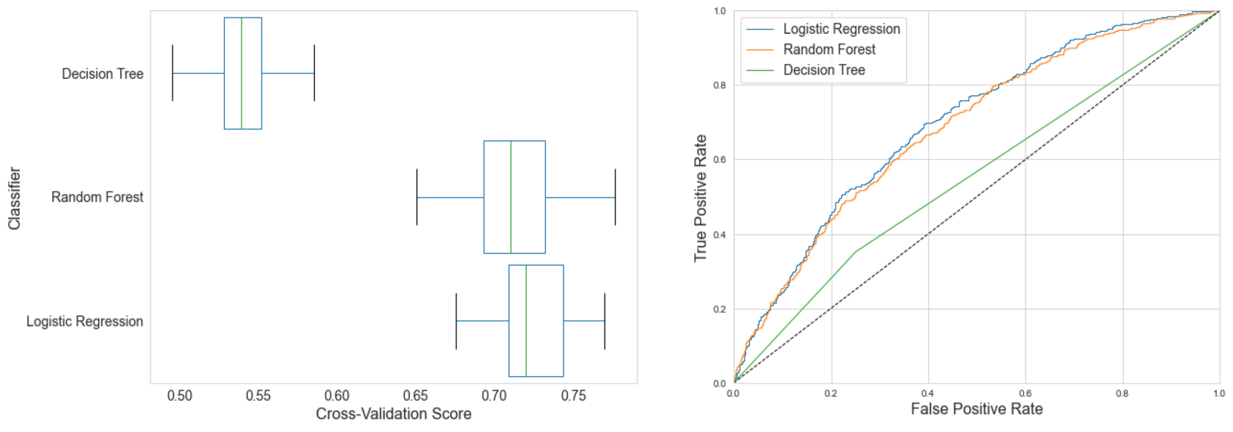
Results

We are using two key metrics for result validation and comparison across models: **recall macro** and **AUC-ROC** curve. Recall is the fraction of loans of a class (such as defaulted or fully paid loans) which are correctly classified. The AUC - ROC curve is a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represent the degree or measure of separability. It tells how much the model can distinguish between classes. The higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. The metric indicates whether defaulting loans are assigned a higher risk than fully paid loans, on average.

Baseline classification models

We have trained three classification models on the imbalanced data using the set of existing features. The cross-validation scores and the AUC-ROC curves suggest that: Logistic Regression performs marginally better than the Random Forest model and significantly better than the Decision Tree Classifier.

Figure 2: Cross-validation score and AUC-ROC score for baseline classification models

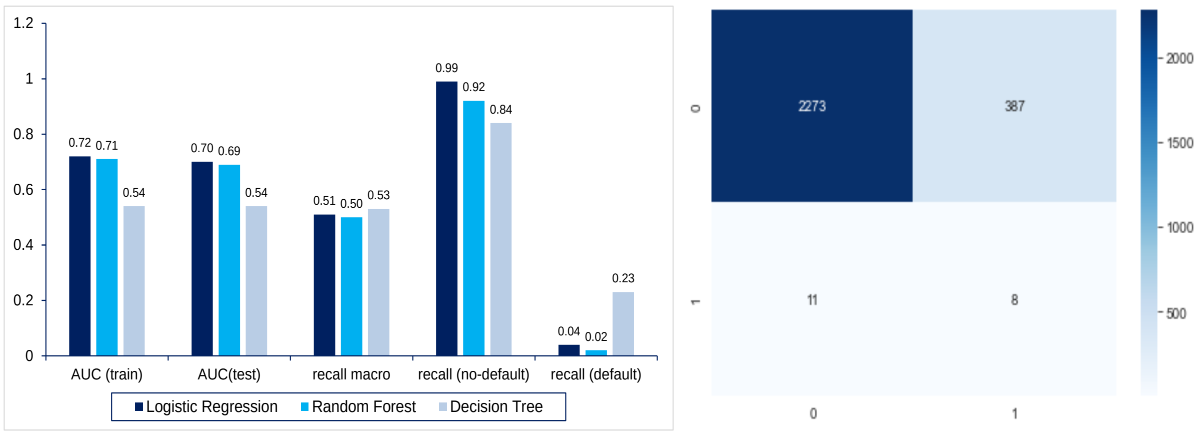


In Figure 3 below, we summarise the average training AUC score and the test scores for the AUC and recall macro metrics, respectively. We observe that the test scores are lower for all models, however the difference between the train and test score is marginal which does not indicate potential overfitting.

If we look at the recall score for both classes, we observe that the overrepresented class in the dataset may have benefitted from the higher quantity of training data, because it has consistently higher values than the recall score of the marginal class.[3] This is a problem because as discussed before we are more concerned with predicting defaulting loans well rather than with misclassifying a fully paid loan. As indicated by the confusion matrix, Logistic Regression –i.e., our best performing baseline model- can predict non-defaulting loans with a good accuracy of 99% but cannot predict well defaulting loans (only 8 out of 395 default loans were predicted correctly).

We will see that correcting for class imbalances and adjusting the probability threshold for classification, can help to significantly increase the performance of the model for defaulting loans.

Figure 3: Summary of performance: AUC score, recall score and confusion matrix



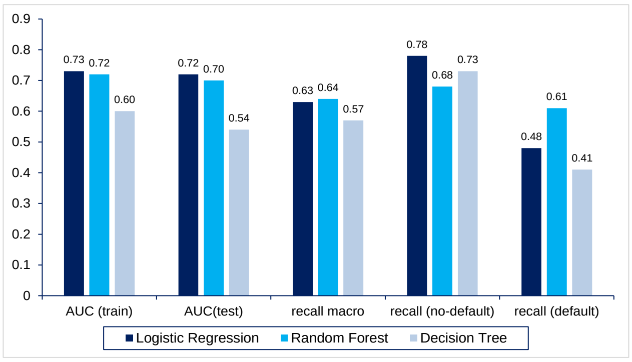
Improved classification models

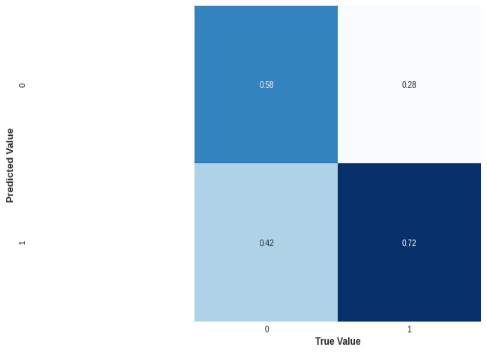
After addressing the issues of: i) class imbalances, ii) feature engineering and iii) hyperparameter optimisation, we observe that Random Forest is the best performing model in terms of the recall macro score and the recall score for the defaulting loans, while Logistics Regression performs marginally better in terms of the AUC score.

We choose Random Forest as the best model because of its superior classification performance in the defaulting loans class (as measured by the corresponding recall score). The test AUC and recall macro scores for this model were 70% and 64%, respectively. While the individual test recall scores for each class were 61% for defaults and 68% for fully paid loans. Clearly, correcting for class imbalances significantly improves the predictive accuracy of the model in the underrepresented class. *(See LHS in Figure 4)*
If we want to further increase the predictive accuracy for Class 1, we can adjust the probability threshold. The optimal threshold is 0.47. *(see Appendix A4 for further details)*. This adjustment increases the prediction accuracy of loan defaults from 61 to 72%. *(See RHS in Figure 4)*

[3]Similar result also observed in Turiel and Aste (2020): "Peer-to-peer loan acceptance and default prediction with artificial intelligence"

Figure 4: Summary of performance for improved models: AUC score, recall score and confusion matrix

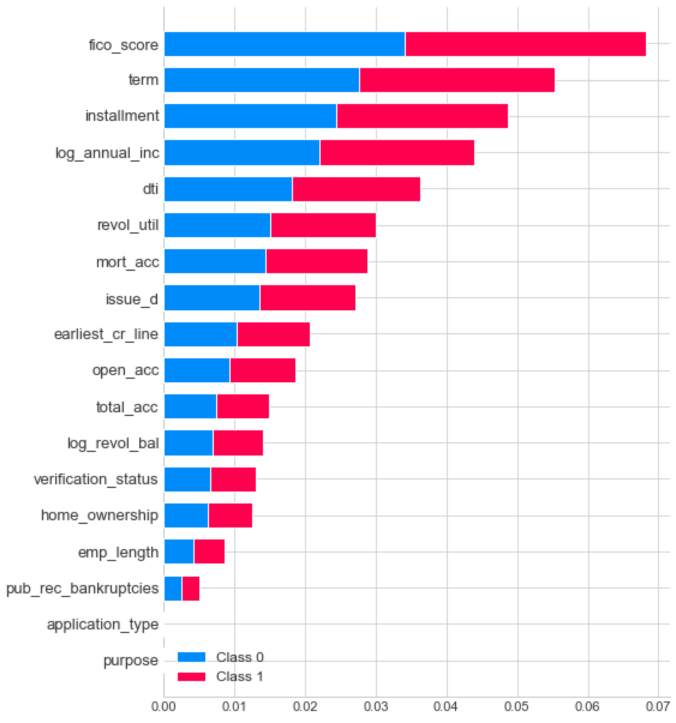




Interpretability

We use the improved random forest classifier to examine feature importance for the model on out-of-sample data. This consists of shuffling one feature at a time and monitoring the change in model loss with respect to the loss of the original data. High feature importance is observed when the randomization of the feature strongly affects the model's ability to predict.

Figure 5: Feature importance for Random Forest



We note that the "FICO score" is ranked first. This variable was expected to be an important feature as the FICO score is an aggregate measure used to estimate an individual's creditworthiness. We also note the 'term' feature is ranked high in order of importance. This is expected as an increase in loan term implies higher interest rates, longer duration risk as well as a longer-term exposure to the financial stability of the individual.

Log annual income is ranked third. This makes sense as the ability of an individual to repay the loan is highly dependent on their annual income.

The next feature by rank is the debt-to-income ratio. This was also expected to be a relevant feature as it represents an individual's level of debt as a fraction of his current income (which determines his ability to repay the outstanding debt). Features which are less intuitive in their relation to default are lowest in the rank, such as 'emp_length', 'application type, 'pub_rec_bankruptcies'.

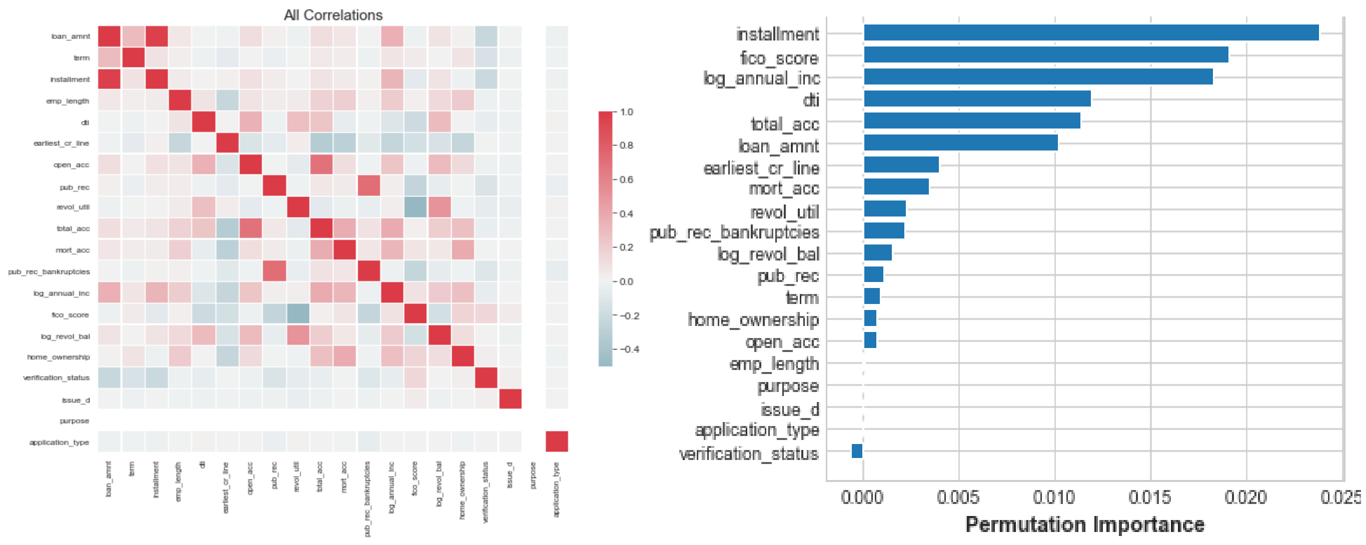
Our analysis confirms that the model is interpreting the phenomenon in a sensible way in relation to domain knowledge and human reasoning.

Appendix

A1. Addressing multicollinearity

We rank the features in order of importance using a permutation-based method. This method randomly shuffles each feature and computes the change in the model's performance (we have used a random forest classifier). Features are ranked based on the impact they have on model's performance. Across the pairs of highly correlated features, we keep only the most important ones (namely, *installment and pub_rec_bankruptcies).*

Figure 6: Feature correlations and permutation-based feature importance



A2. PCA for feature engineering

PCA finds the directions of maximum variance in high-dimensional data and projects the data onto a new subspace with equal or fewer dimensions than the original one. PCA performs eigen decomposition of the covariance matrix**.** The eigenvectors of the covariance matrix represent the principal components (the directions of maximum variance), whereas the corresponding eigenvalues will define their magnitude.

We have run the PCA analysis using the five most important features (as suggested by the permutation-based importance graph above). The signs and magnitudes of a component's loadings tell us what kind of variation is captured by each feature. [4]

	PC1	**PC2**	**PC3**	**PC4**	**PC5**
dti	0.16	0.74	-0.20	0.45	0.43
log_annual_inc	0.63	-0.28	0.05	-0.34	0.63
fico_score	0.05	-0.48	-0.77	0.41	0.02
installment	0.48	-0.20	0.49	0.65	-0.26
total_acc	0.59	0.31	-0.34	-0.31	-0.59

A3. Hyperparameter Tunning

- **Logistic Regression:** Regularization techniques were applied to avoid overfitting in the LR model. Both L2 and L1 regularizations were included in the search. The range for the regularization parameter *C* varied from $[10^{-5}, 10^{5}]$. Different values for the maximum

[4] Kaggle: "Feature engineering, scaling, PCA and predictions"

number of iterations were tested*:* [500,1000,3000]. We have also tested the performance of convergence with different *solvers*, namely: [‘newton-cg’, ‘lbfgs’].

- **Random Forest:** we do a grid search over hyperparameter values for the following parameters of the random forest:
  - **n_estimators**: number of trees under consideration. more trees should be able to produce a more generalized result, but by choosing more trees, the time complexity of the Random Forest model also increases. We try the following range for *n_estimators = [int(x) for x in np.linspace(start = 20, stop = 2000, num = 10)]*
  - **max_features**: This resembles the number of maximum features provided to each tree in a random forest. We consider the following options: *max_features = ['auto', 'sqrt']*
  - **criterion**: The function to measure the quality of a split. Supported criteria are: *"gini"* for the Gini impurity and *"entropy"* for the information gain.
  - **max_depth of** a tree in Random Forest is defined as the longest path between the root node and the leaf node.[5]We try the following range for the *max_depth parameter: [int(x) for x in np.linspace(10, 110, num = 10)]*
  - **min_sample_split –** a parameter that tells the decision tree in a random forest the minimum required number of observations in any given node in order to split it.[6] We try the following range for the min_sample_split parameter: [2, 5, 10]
- **Decision Tree Classifier:** for the decision tree classifier I have used similar hyperparameter to random forest except for the number of estimators, which is not a relevant parameter for this classifier.

Improved models after hyperparameter optimisation:

- The randomised grid search for Logistic Regression returned an optimal model with the following hyperparameter parameters: i) penalty =L2; ii) penalty term: C=0.02; iii) max_iter=500; iv) solver=’lbfgs’.
- For the Decision tree classifier, the search returned the following optimal hyperparameters: criterion='entropy'; max_depth=10; max_features=None; max_leaf_nodes= None; min_samples_leaf=1; min_samples_split=10.
- For the Random Tree classifier, the search returned the following optimal hyperparameters: criterion='entropy'; max_depth=3; max_features='auto'; min_samples_leaf=1; min_samples_split=5; n_estimators=3000.

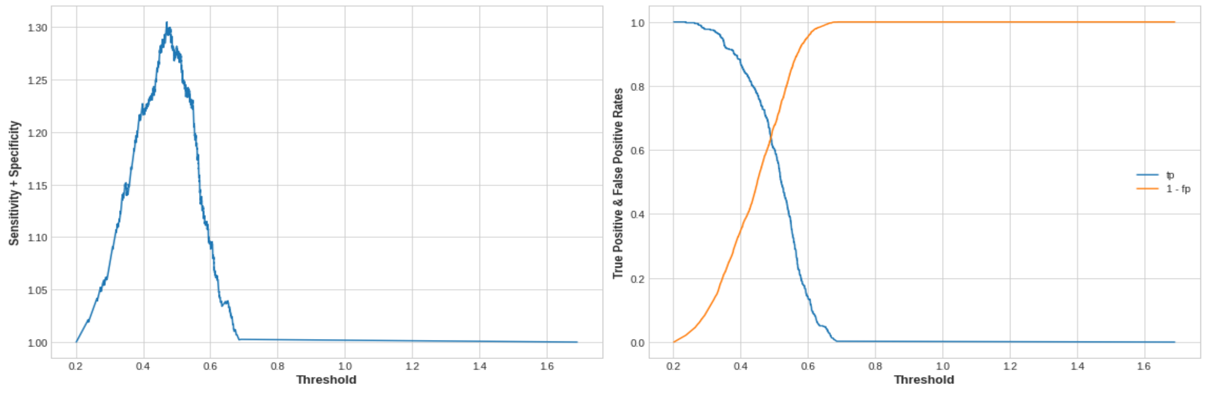
[5] As the max depth of the decision tree increases, the performance of the model over the training set increases continuously. On the other hand, as the max_depth value increases, the performance over the test set increases initially but after a certain point, it starts to decrease rapidly. The tree starts to overfit the training set and therefore is not able to generalize over the unseen points in the test set.

[6] The default value of the minimum_sample_split is assigned to 2. This means that if any terminal node has more than two observations and is not a pure node, we can split it further into subnodes. Having a default value as 2 poses the issue that a tree often keeps on splitting until the nodes are completely pure. As a result, the tree grows, and therefore overfits the data.

A4. Optimal threshold (Random Forest Classifier)

We choose a new prediction threshold according to the sensitivity and specificity of the model. This will create some balance in predicting the binary outcome. The optimal threshold is where the two graphs meet – this happens for the value of 0.47.

Figure 7: Optimal threshold for the improved random forest classifier


Sensitivity + Specificity
Threshold
True Positive & False Positive Rates
tp
1 - fp
Threshold