RISK ANALYSIS SYSTEM FOR INVESTORS

Authors: Kunjan Mhaske and Mayur Jawalkar

Introduction to Checkpoint 4:

For this project, we worked on deciding the core algorithm to analyze the lending loan risk by classifying available loan data to categorize it as good loans and bad loans. We have considered 5 statistical machine learning models to get the analysis along with their accuracy and efficiency towards the available Lending Loan Club dataset. We worked on Decision trees Classification, KNN Classification, Logistic Regression, and Random Forest Classification. After observing the overall performance of these 4 models, for this instance, we have selected Random Forest Classification as the core algorithm for the system. Further, we worked on the 5th algorithm SVM, which is not covered in the class and tried to fine-tune it to get better outcome. We then compared all algorithms with each other and selected Random Forest Classifier as the core algorithm.

The Goal:

Investment in loan lending business is financially risky without a proper system to analyze the possibility of the existing loans being a good loan or bad loans. The investors should check the historic as well as current statistics of the borrower and deduce the result to invest more money towards improving bad loans or maintaining good loans. For this herculean task, we are proposing this model based on the historic and currently available data to find out the maximum possibility of existing loans becoming a good loan or a bad loan for investors.

Basic Steps of Data Preparation:

The different statistical machine learning models use a specific format of data and the absence of the proper format will lead to failure of the model. So, datasets generally require some amount of preparation before they can yield useful insights. Although our dataset has some missing values and irrelevant values that were taken care of in the previous checkpoint number 2, we have to select the particular set of features for each model. We have taken the measures not to miss out on relevant information for the final goal of the system as well as to not consider the most obvious and correlated attributes while features selection process.

Feature Selection:

After data pre-processing, we left with a cleaned dataset without any missing values or irrelevant values to perform classification models on it. The correlation between features can be found by performing linear regressions on some selective quantitative and qualitative attributes. After observing p-values from the lm() model, we have separated the set of suspiciously insignificant attributes. We will cross-check their importance in further processes of classification to find out if we are missing any important information or not.

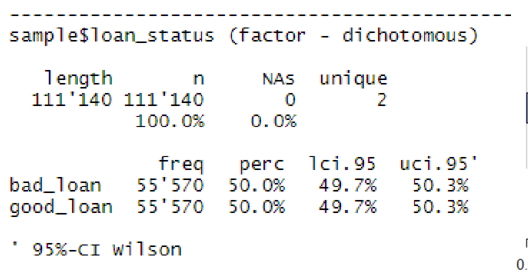
On the observation of the remaining dataset, we needed a deciding feature (response) to perform the classification of loans to good or bad. From existing labels of loan status for each case, we categorized the labels according to their description. This loan status attribute is now a response feature for our classification task. Now, we have to remove highly correlated features and direct dependant of loan status to avoid multicollinearity with loan status and overfitting our model with redundant features. Finally, we have the most important features to deploy classification models on them with loan status as the response.

Class Balancing:

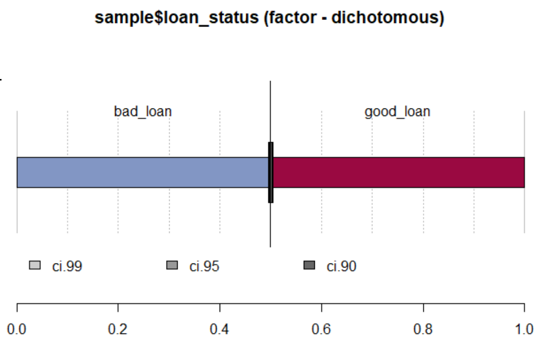
Imbalanced data means where the number of observations is not the same for all classes in the dataset. In our dataset, we have an 80:20 ratio of good_loans to bad_loans classes. However, this imbalanced condition is quite common in the real world and we must deal with it while performing a classification model on a dataset. Models like random forest fail to cope with such an imbalanced training dataset as they are sensitive to the proportions of the different classes. If performed on such a dataset, they always favor the majority class which may mislead the accuracies.

We have randomly selected the 50:50 portion of bad_loans and good_loans with an ample amount (around 100K) of observations in sample data and divide it to 80:20 portion for training and testing. This eliminates the imbalance of the class in the data so that the model will give actual accuracy.

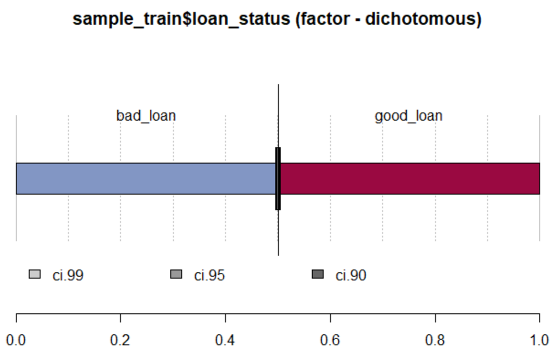
Sample Data:

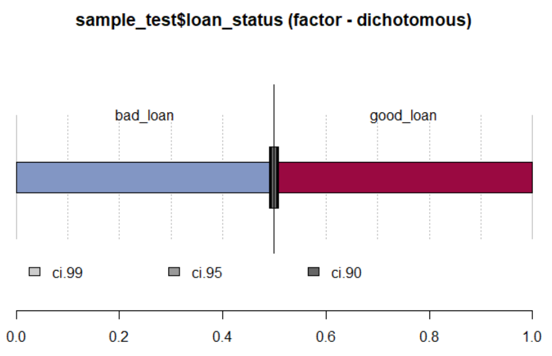


```
------------------------------------------------------
sample$loan_status (factor - dichotomous)

   length       n     NAs  unique
  111'140 111'140       0       2
           100.0%    0.0%

              freq    perc  lci.95  uci.95'
bad_loan    55'570   50.0%   49.7%   50.3%
good_loan   55'570   50.0%   49.7%   50.3%

' 95%-CI Wilson
```

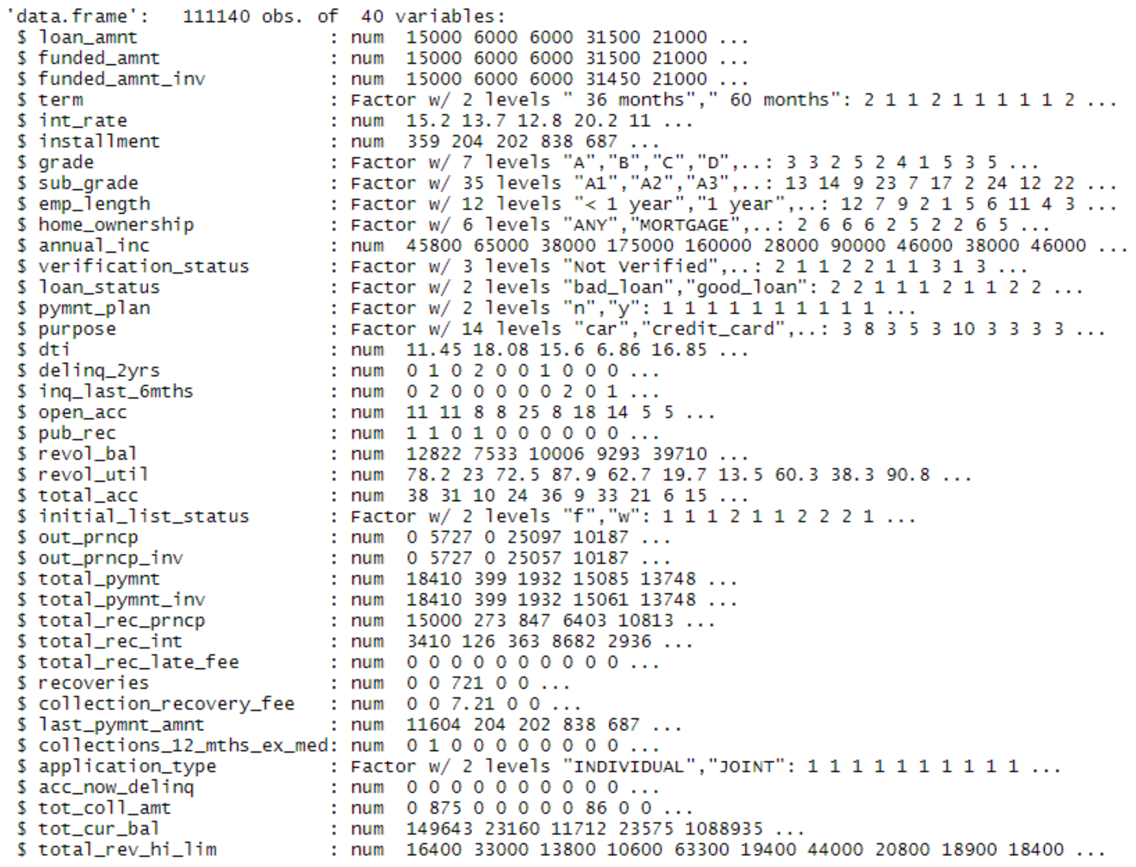


Sample_Training Data and Sample_Testing Data





Structure of Sample Data



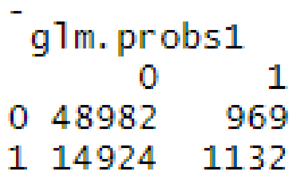
```
'data.frame':   111140 obs. of  40 variables:
 $ loan_amnt                 : num  15000 6000 6000 31500 21000 ...
 $ funded_amnt               : num  15000 6000 6000 31500 21000 ...
 $ funded_amnt_inv           : num  15000 6000 6000 31450 21000 ...
 $ term                      : Factor w/ 2 levels " 36 months"," 60 months": 2 1 1 2 1 1 1 1 1 2 ...
 $ int_rate                  : num  15.2 13.7 12.8 20.2 11 ...
 $ installment               : num  359 204 202 838 687 ...
 $ grade                     : Factor w/ 7 levels "A","B","C","D",..: 3 3 2 5 2 4 1 5 3 5 ...
 $ sub_grade                 : Factor w/ 35 levels "A1","A2","A3",..: 13 14 9 23 7 17 2 24 12 22 ...
 $ emp_length                : Factor w/ 12 levels "< 1 year","1 year",..: 12 7 9 2 1 5 6 11 4 3 ...
 $ home_ownership            : Factor w/ 6 levels "ANY","MORTGAGE",..: 2 6 6 6 2 5 2 2 6 5 ...
 $ annual_inc                : num  45800 65000 38000 175000 160000 28000 90000 46000 38000 46000 ...
 $ verification_status       : Factor w/ 3 levels "Not Verified",..: 2 1 1 2 2 1 1 3 1 3 ...
 $ loan_status               : Factor w/ 2 levels "bad_loan","good_loan": 2 2 1 1 1 2 1 1 2 2 ...
 $ pymnt_plan                : Factor w/ 2 levels "n","y": 1 1 1 1 1 1 1 1 1 1 ...
 $ purpose                   : Factor w/ 14 levels "car","credit_card",..: 3 8 3 5 3 10 3 3 3 3 ...
 $ dti                       : num  11.45 18.08 15.6 6.86 16.85 ...
 $ delinq_2yrs               : num  0 1 0 2 0 0 1 0 0 0 ...
 $ inq_last_6mths            : num  0 2 0 0 0 0 0 2 0 1 ...
 $ open_acc                  : num  11 11 8 8 25 8 18 14 5 5 ...
 $ pub_rec                   : num  1 1 0 1 0 0 0 0 0 0 ...
 $ revol_bal                 : num  12822 7533 10006 9293 39710 ...
 $ revol_util                : num  78.2 23 72.5 87.9 62.7 19.7 13.5 60.3 38.3 90.8 ...
 $ total_acc                 : num  38 31 10 24 36 9 33 21 6 15 ...
 $ initial_list_status       : Factor w/ 2 levels "f","w": 1 1 1 2 1 1 2 2 2 1 ...
 $ out_prncp                 : num  0 5727 0 25097 10187 ...
 $ out_prncp_inv             : num  0 5727 0 25057 10187 ...
 $ total_pymnt               : num  18410 399 1932 15085 13748 ...
 $ total_pymnt_inv           : num  18410 399 1932 15061 13748 ...
 $ total_rec_prncp           : num  15000 273 847 6403 10813 ...
 $ total_rec_int             : num  3410 126 363 8682 2936 ...
 $ total_rec_late_fee        : num  0 0 0 0 0 0 0 0 0 0 ...
 $ recoveries                : num  0 0 721 0 0 ...
 $ collection_recovery_fee   : num  0 0 7.21 0 0 ...
 $ last_pymnt_amnt           : num  11604 204 202 838 687 ...
 $ collections_12_mths_ex_med: num  0 1 0 0 0 0 0 0 0 0 ...
 $ application_type          : Factor w/ 2 levels "INDIVIDUAL","JOINT": 1 1 1 1 1 1 1 1 1 1 ...
 $ acc_now_delinq            : num  0 0 0 0 0 0 0 0 0 0 ...
 $ tot_coll_amt              : num  0 875 0 0 0 0 86 0 0 ...
 $ tot_cur_bal               : num  149643 23160 11712 23575 1088935 ...
 $ total_rev_hi_lim          : num  16400 33000 13800 10600 63300 19400 44000 20800 18900 18400 ...
```

Models Under Consideration:

1) <u>Logistic Regression:</u>

Logistic regression models the posterior probability that Y belongs to a category. Various models are trained using glm() to perform logistic regressions and p-values are taken into consideration in choosing the significant predictors which helps to reduce feature space and help in interpretability. The results of the best trained model is shown below.

Confusion Matrix:



```
  glm.probs1
        0     1
0  48982   969
1  14924  1132
```

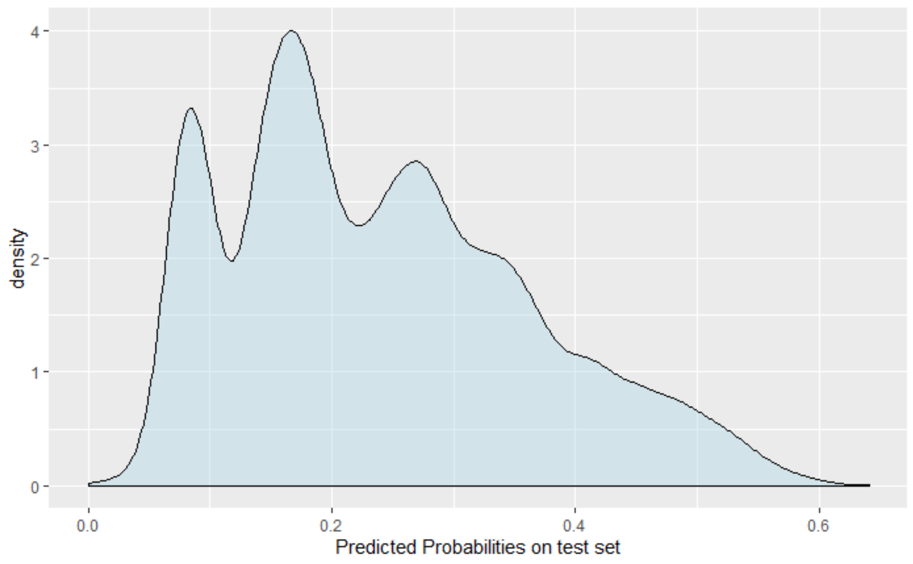
The accuracy obtained is 75.92%. As per confusion matrix, the Specificity of the model is 92.9% and Sensitivity is 98.06%, considering "0" as a positive class. (0 is Good Loan).

The coefficients of the following features are positive: Loan Amount, Interest Rate, Home Ownership - Other.

The coefficients of the following features are negative: Annual Income, Home Ownership - Own, Home Ownership - Rent,

There is no significant difference in the early years of employment. This means that the probability of defaulting is inversely proportional to the factors mentioned above.

Testing Error Plot

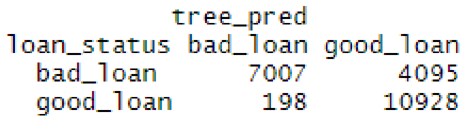


2) <u>Decision Tree:</u>

A classification tree is used to predict a qualitative response by assigning each observation to the most commonly occurring class of training observations in the region to which it belongs. The complexity parameter (cp) is used to control the size of the decision tree and to select the optimal tree size acts as tuning parameter for the bias-variance trade-off. If the cost of adding another variable to the decision tree from the current node > cp, then tree building does not continue. We prune the tree to avoid any overfitting of the data.

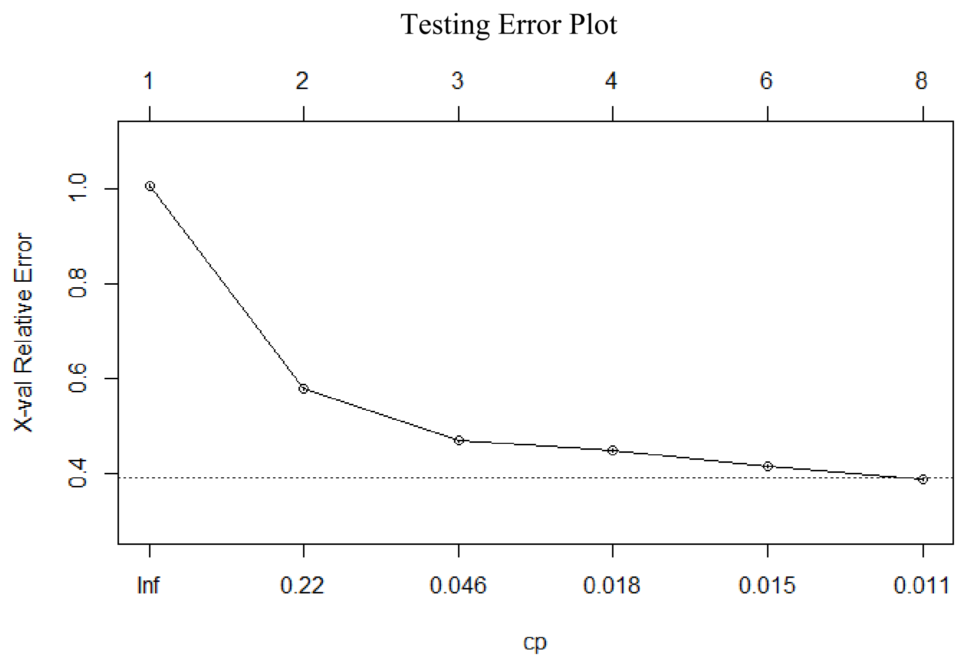
The x-error is the cross-validation error (R has built-in cross validation). We use rel_error(relative error i.e. training error rate), xerror and xstd together are used to help choose where to prune the tree. The one with least cross-validated error (xerror) is the optimal value of CP.

Confusion Matrix:



```
             tree_pred
loan_status bad_loan good_loan
  bad_loan      7007      4095
  good_loan      198     10928
```

The accuracy obtained is 80.68%. As per confusion matrix, the Specificity of the model is 36.8% and Sensitivity is 98.2%, considering “bad_loans” as a positive class.



As can be observed from the model the optimal complexity parameter value cp = 0.011 is chosen as this leads to low test error rate and an accuracy of 80.68%.

3) <u>KNN Classifier:</u>

In theory we would always like to predict qualitative responses using the Bayes classifier. But for real data, we do not know the conditional distribution of Y given X, and so computing the Bayes classifier is impossible. An attempt to estimate the conditional distribution of Y given X, and then classify a given observation to the class with highest estimated probability is the K-nearest neighbours (KNN) classifier.

Given a positive in-K and a test observation x0, the KNN classifier first identifies the neighbours K points in the training data that are closest to x0, represented by N0. It then estimates the conditional probability for class j as the fraction of points in N0 whose response values equal j. For K=1 the training error rate is always equal to zero. For low values of K the bias of the model is low and the variance is high and for high values of K the bias of the model is high and variance low. Hence, we need to find a Bias-Variance trade-off by choosing optimal value of K.

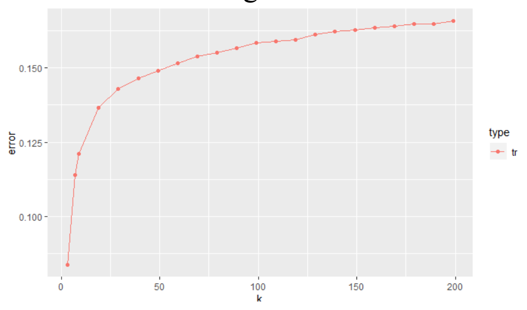
Confusion Matrix for KNN with K=19:

```
                      pred
sample_test$status        0        1    Total
-------------------------------------------
0                     7493     2481     9974
                      37.5%    12.4%
-------------------------------------------
1                     4526     5500    10026
                      22.6%    27.5%
```

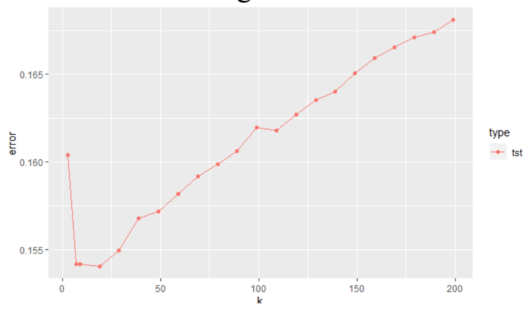
The accuracy obtained is 65%. As per confusion matrix, the Specificity of the model is 24.85% and Sensitivity is 54.8%, considering “0” as a positive class. (0 is Good Loan).

<u>Results of K-Fold Cross Validation: 5 folds for KNN with K up to 200.</u>

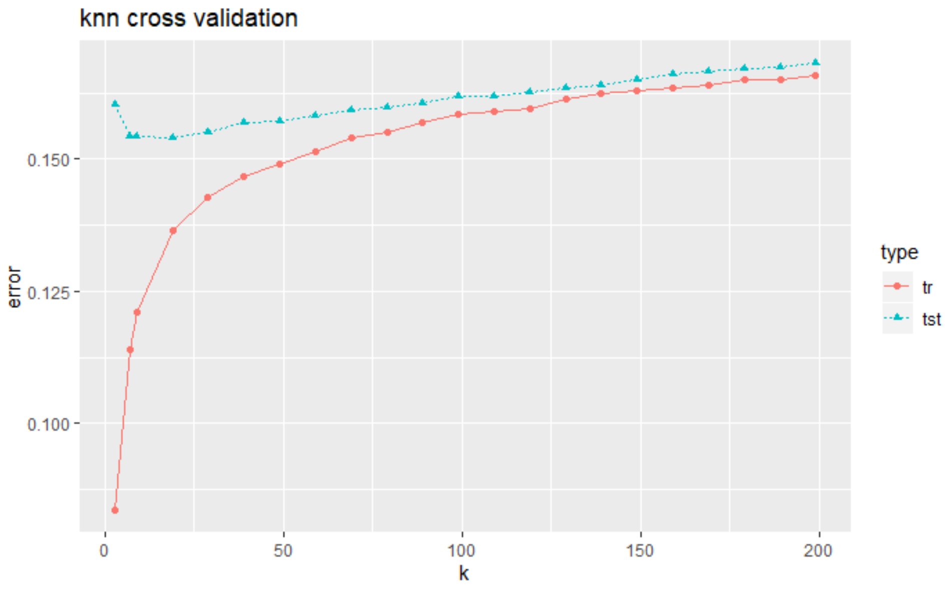
Training Error Plot



Testing Error Plot



Comparing Training and Testing Plots:



As can be observed from the model the optimal value of K where the testing error rate is minimum is at K=19 and the accuracy obtained is 65%.

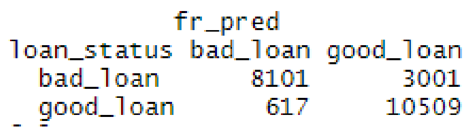
The Core Algorithm: Random Forest

The decision trees suffer from high variance. This means that if we split the training data into two parts at random, and fit a decision tree to both halves, the results that we get could be quite different. Bootstrap aggregation, or bagging, is a general-purpose procedure for reducing the bagging variance of a statistical learning method by averaging a set of observations which reduces variance.

A very straightforward way to estimate the test error of a bagged model, without the need to perform cross-validation or the validation set approach is Out-of-Bag Error Estimation (OOB). Bagging typically results in improved accuracy over prediction using a single tree. Bagging improves prediction accuracy at the expense of interpretability.

Although the collection of bagged trees is much more difficult to interpret than a single tree, we can obtain an overall summary of the importance of each predictor in the context of bagging classification trees, by adding up the total amount the Gini index is decreased by splits over a given predictor, averaged over all B trees. Random forests provide an improvement over bagged trees by decorrelating the trees by choosing a random sample of m predictors as split candidates from the full set of p predictors. The predictions from the bagged trees will be highly correlated. Unfortunately, averaging many highly correlated quantities does not lead to as large of a reduction in variance as averaging many uncorrelated quantities. This means that bagging will not lead to a substantial reduction in variance over a single tree in this setting. Random forests overcome this problem by forcing each split to consider only a subset of the predictors.

Confusion Matrix: Using 100 ntrees.



```
             fr_pred
loan_status bad_loan good_loan
  bad_loan      8101      3001
  good_loan      617     10509
```

The accuracy obtained is 83.85%. As per confusion matrix, the Specificity of the model is 5.5% and Sensitivity is 72.96%, considering “bad_loan” as a positive class.

Experimentation with number of trees:

Accuracy	Number of trees	Mtry
0.6452672	2	1
0.7857207	20	1
0.8303041	50	2
0.8350279	80	2
0.838582	100	Default
0.8388069	200	Default
0.8393918	300	Default

Fine Tuning of Random Forest:

Number of selected variables is denoted by **mtry** in randomForest() function. We tried to fine tune model by selecting the mtry value with minimum OOB (out of bag) error. We used tuneRF() function from randomForest library of R. After getting best mtry value, mtry = 5, to run random forest with 500 number of trees which resulted in the following confusion matrix.

```
      mtry OOBError
4.OOB    4  0.14916
5.OOB    5  0.14822
7.OOB    7  0.14854
```

As can be observed from the model best accuracy is obtained by the random forest where at each split in the tree 5(**mtry**) variables are chosen from the set of predictors available.

Confusion Matrix for tuned model:

```
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 5

        OOB estimate of  error rate: 14.92%
Confusion matrix:
      0     1 class.error
0 23798  1228  0.04906897
1  6231 18743  0.24949948
```

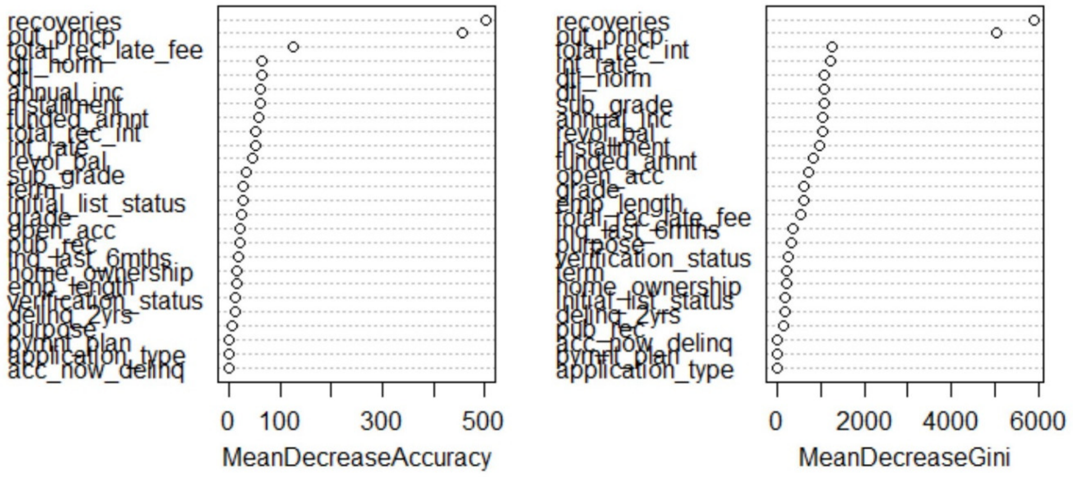
The accuracy obtained is 85.85%. As per confusion matrix, the Specificity of the model is 4.9% and Sensitivity is 75%, considering “0” as a positive class. (0 is Good Loan).

This shows 2% increase in the accuracy while maintaining the balance between specificity and sensitivity.

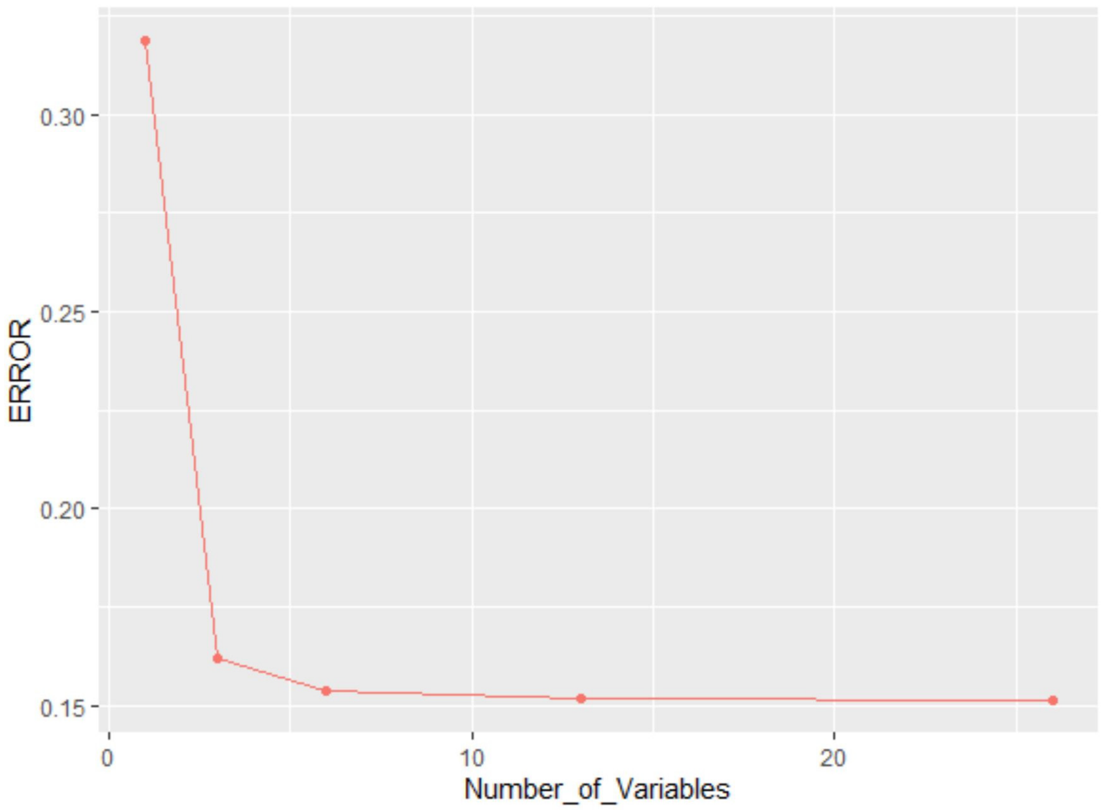
Important Variables:

```
                             0            1 MeanDecreaseAccuracy MeanDecreaseGini
funded_amnt          59.062974  -9.99808369           59.4375095     8.225606e+02
term                 25.220009  17.00610938           28.6329166     2.294903e+02
int_rate             41.187831  15.46472174           51.7064580     1.221226e+03
installment          64.381398 -10.52195514           61.9224445     9.871404e+02
grade                21.399746  10.04322715           23.7746139     6.105185e+02
sub_grade            30.695043  12.34153590           34.8491808     1.065119e+03
emp_length           12.222538   6.82629993           14.3095800     6.056729e+02
home_ownership       16.831934   1.44727322           14.5689830     2.197509e+02
annual_inc           58.399194  13.62109857           62.1901323     1.061248e+03
verification_status  13.661837   0.01781608           11.3422200     2.520511e+02
pymnt_plan            0.000000   0.00000000            0.0000000     3.047619e-03
purpose               9.079724  -0.44879015            7.3088611     3.392163e+02
dti                  63.128932 -18.66457785           62.9609215     1.072762e+03
delinq_2yrs          14.582744  -3.01136226           10.6742237     1.868868e+02
inq_last_6mths       15.848687   5.43737509           16.6559429     3.697262e+02
open_acc             25.347984  -2.94476260           20.2905139     7.204413e+02
pub_rec              24.207630  -4.83185467           20.1494673     1.527397e+02
revol_bal            55.311118 -16.94684166           44.9572881     1.039432e+03
initial_list_status  28.139929   7.71095230           27.1622863     1.873734e+02
out_prncp           433.859963 411.61494044          454.9763048     5.011962e+03
total_rec_int        31.905949  39.68577183           53.1169468     1.245195e+03
total_rec_late_fee  122.077035  42.61824331          126.4837268     5.466317e+02
recoveries          458.686587 468.48089102          501.3150562     5.880251e+03
application_type      0.000000   0.00000000            0.0000000     0.000000e+00
acc_now_delinq       -1.158723   0.70013984           -0.6821246     6.858953e+00
dti_norm             65.187336 -18.12528534           64.3720870     1.078666e+03
```

Important Variables with sorted order: topmost are most important



Cross Validation Error Plot for Random Forest Model



As we observe from the Cross-Validation Error Plot for Random Forest Model the optimal number of predictors used for building the random forests are p = 26.

Model not covered in the class: Support Vector Machine (SVM)

The support vector machine (SVM) is an extension of the support vector classifier that results from enlarging the feature space in a specific way, using kernels. A kernel is a function that quantifies the similarity of two observations. Support vector machines are intended for the binary classification setting in which there are two classes. But there are the extensions of support vector machines to the case of more than two classes. The tuning parameter C controls the bias-variance trade-off of the statistical learning techniques. It is generally chosen via cross-validation. C bounds the sum of the epsilons and hence determines the severity of the violation of the margin that it will tolerate. The smallest distance from the observations to the separating hyperplane is known as the margin. The separating hyperplane that is farthest from the training observations after computing perpendicular distance from each training observations to the given separating hyperplane.

Observations that lie directly on the margin, or on the wrong side of the margin for their class, are known as support vectors. These observations affect the support vector classifier. When the tuning parameter C is large, then the margin is wide, many observations violate the margin, and so there are many support vectors. The advantage of using SVM is computational cost because using kernels, one need only compute the kernel function for all distinct pairs of inputs. This can be done without explicitly working in the enlarged feature space. For some kernels, such as the radial kernel, the feature space is implicit and infinite-dimensional, hence we can not do all the computations using our limited resources.

<u>Training of SVM:</u>

Initially we trained the SVM model using radial kernel with gamma value as 1 and cost as 10.

```
# apply SVM
svmfit =svm(status~., data=train_data, kernel ="radial",
            gamma =1, cost = 10)
svmfit
```

```
Number of Support Vectors:  3494


predict_loan_status_label     0     1
                        0 15075  5786
                        1 18126 27513

Parameter tuning of 'svm':

- sampling method: 10-fold cross validation

- best parameters:

- best performance: 0.2120989
```

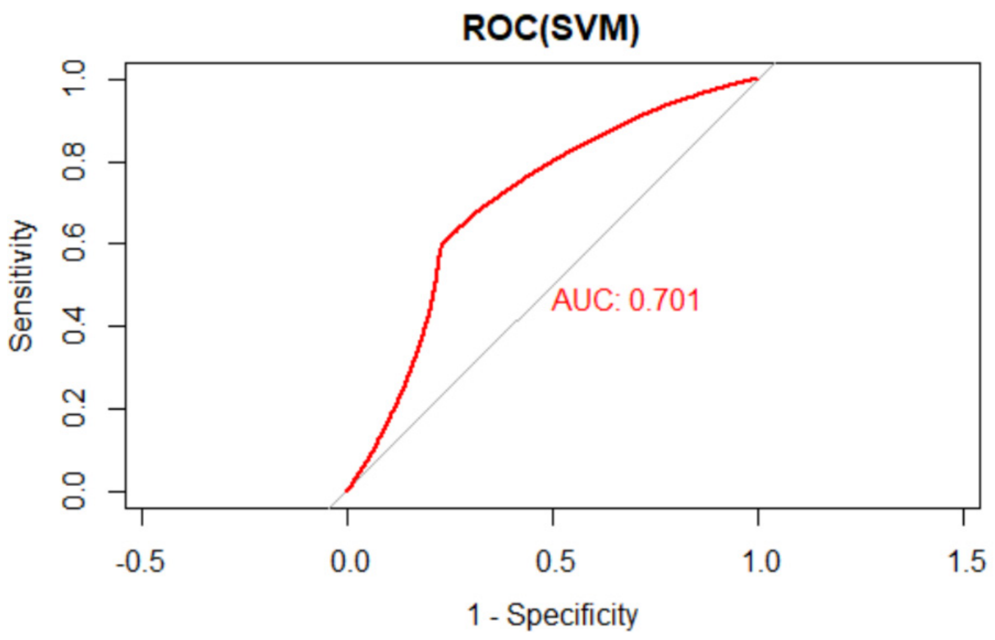
For SVM, we use Radial Basis as a kernel function. Because of limited computational power, we decided to use only a small subset of the data for training the model. We have sampled out 5% (3500 records) data out of 70000 equally balanced samples (equal number of positive and negative classes) for training the model.

Confusion Matrix:

```
predict_label     0     1
            0 19226  7030
            1 13975 26269
```

Accuracy <dbl>	Sensitivity <dbl>	Specificity <dbl>
64.042	45.405	82.624

Above model when tested on a test data sample containing about 66000 records, it produced around 64% accuracy with 82 % specificity and 45% sensitivity.



Fine Tuning of SVM:

Later we tried to improvise the accuracy of SVM model by fine-tuning the gamma and cost values. In this step we tried giving cost values as 0.1, 1, 10, 100 and 1000, and gamma values as 0.5, 1, 2, 3 and 4.

```
# Fine-Tune the SVM to improve the accuracy.

tune.out=tune(svm , status~., data=train_data, kernel ="radial",
ranges =list(cost=c(0.1 ,1 ,10 ,100 ,1000),
gamma=c(0.5,1,2,3,4)))

summary(tune.out)
```

After this training SVM produced the best result at Cost=1 and gamma = 0.5.

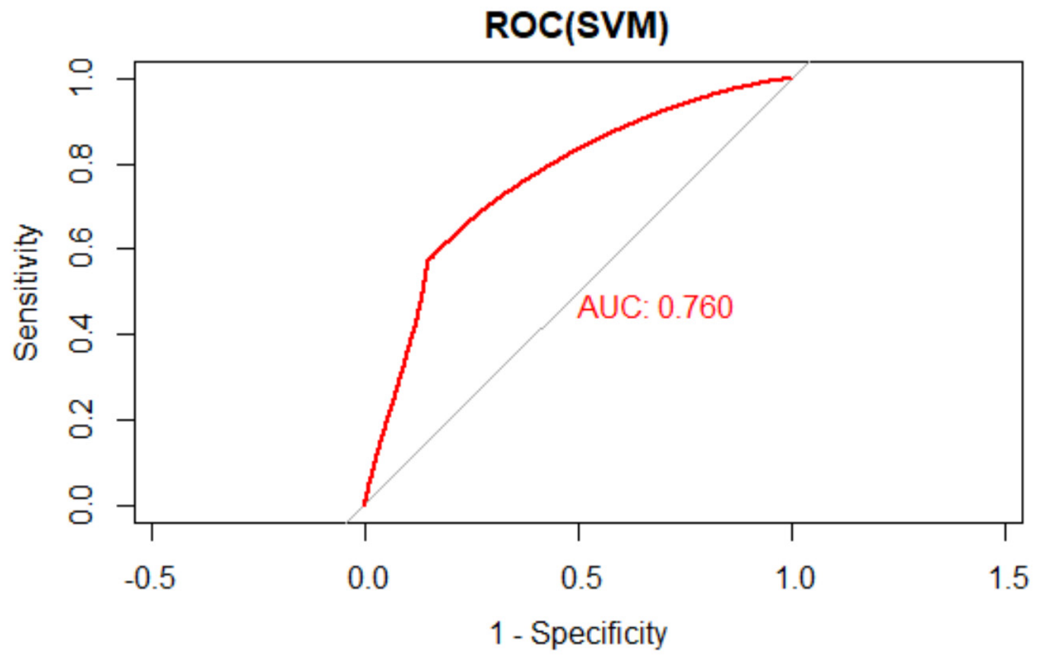
Confusion Matrix:

```
predict_label     0     1
            0 19237  7019
            1 14017 26227
```

Accuracy <dbl>	Sensitivity <dbl>	Specificity <dbl>
68.367	57.849	78.888

Upon testing on a test data sample containing about 66000 records, it produced around 68.36% accuracy with 78.88 % specificity and 57.85 % sensitivity.

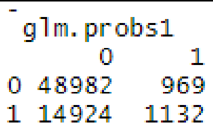
Hence, we can see about 4% improvement in the accuracy after fine-tuning the model.

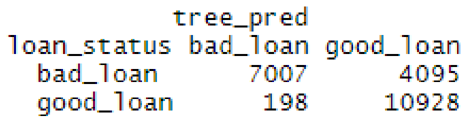


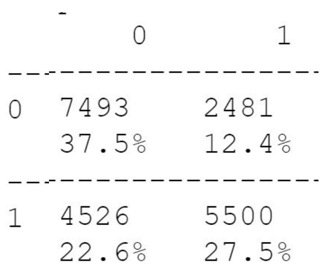
CONCLUSION:

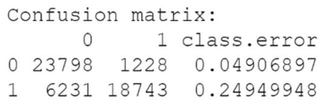
After working on fine tuning of all models, we can compare them with each other using outcomes and confusion matrices of them on testing dataset.

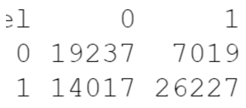
Confusion Matrices of Testing:

Logistic Regression	 `glm.probs1`
	 `     0     1`
	 `0 48982   969`
	 `1 14924  1132`
	 0 - good loan and 1 - bad loan

Decision Tree Classifier	 `              tree_pred`
	 `loan_status bad_loan good_loan`
	 `  bad_loan      7007      4095`
	 `  good_loan      198     10928`

KNN Classifier	 `     0       1`
	 `0  7493    2481`
	 `   37.5%   12.4%`
	 `1  4526    5500`
	 `   22.6%   27.5%`
	 0-good loan and 1- bad loan

Random Forest Classifier	 `Confusion matrix:`
	 `      0     1 class.error`
	 `0 23798  1228  0.04906897`
	 `1  6231 18743  0.24949948`
	 0-good loan and 1- bad loan

Support Vector Machine	 `el     0     1`
	 `0 19237  7019`
	 `1 14017 26227`
	 0- good loan and 1- bad loan

Accuracies of all models:

Logistic Regression	75.92 %
Decision Tree Classifier	80.68 %
KNN Classifier	65 %
Random Forest Classifier	85.85 %
Support Vector Machine	68.36 %

The specificity of the models suggests that the fraction of times that the model has wrongly predicted bad loans as good loans and having a large specificity for the model may have financial ramifications to the banking company like credit card companies. Logistic regression classifier has the highest specificity of the 5 models analysed and the value is **92.9%.** Hence even though SVM classifier has lower accuracy companies would still prefer to use this model due to its low specificity as compared to Logistic regression. KNN has lower specificity than SVM but also low accuracy compared to it. The lowest specificity is obtained by random forest classifier and value is **4.9%.**

The sensitivity of the models suggests the fraction of times that the model has correctly predicted bad loans from the total number of bad loans in the dataset. This indicator would still have some economic impact on the financial company but not as bad as wrongly predicting bad loans as good loans.

Ranking of the models based on their accuracies, specificity and sensitivity and assuming the economic impact of wrongly predicting bad loans as good loans is much higher.

1	Random Forest Classifier	85.85 %
2	Decision Tree Classifier	80.68 %
3	KNN Classifier	65 %
4	Support Vector Machine	68.36 %
5	Logistic Regression	75.92%

The KNN Classifier gives lowest accuracy of 65 % even after training and testing on different K values. SVM gives slightly better accuracy than KNN Classifier which is 68.36%. Hence, both SVM and KNN ruled out from consideration of the core algorithm.

Ultimately, models with admissible accuracies to be considered for the candidate of core algorithms are Logistic Regression, Decision Tree and Random Forest. Logistic regression gives 75.92 % accuracy which is obtained after experimentation of interactions and transformations of features to make them significant towards the response. However, Decision tree gives better accuracy than Logistic regression and hence Logistic regression ruled out from the candidates for core algorithm. Decision tree gives 80.68 % accuracy on the testing data after experimentation of pruning the tree and considering cp values for it. The decision trees have limited predictive power due to high variance and even after bagging, it may not be effective on test data due to highly correlated predictors and therefore, in general, the Random Forest model is preferred on the top of Decision trees.

In our case, Decision Tree ruled out all other models but Random Forest. At first Random Forest gave only 83.85 % accuracy which is just 3 % improvisation as compared to the tuned Decision tree. However, after fine tuning the Random Forest model and applying the best mtry value of lowest OOB validation error to it, the accuracy boosted up to 85.85 % on testing data which is very good improvisation over the Decision Tree. Also, the random forest provides default regularization by considering number of decision trees and the significant difference of their accuracies, we can safely say that Random Forest is the stronger and our core algorithm to analyze the loan lending risk to investors by classifying available loan data to categorize it as good loans and bad loans.