A Solution to Classification on Imbalanced Data Credit Card Fraud Detection as an Example

Group # 8

Jimi He[1] (116010067)
Jiayuan Dong[2] (116010038)
Ziyue Jiang[3] (116010100)

[1] Algorithm Design, Slides and Report Management, Data Visualization, and Random Forest Modeling.
[2] AdaBoosting, XGBoosting, and Random Forest modeling, Model comparison report.
[3] Logistic Regression, Naïve Bayes, and SVM Modeling, Model comparison report.

Section I. Introduction to the Problem

According to Wikipedia, Credit card fraud is a wide-ranging term for theft and fraud committed using or involving a payment card, such as a credit card or debit card, as a fraudulent source of funds in a transaction. The purpose may be to obtain goods without paying, or to obtain unauthorized funds from an account.[4]

According to Kaggle, the datasets contains transactions made by credit cards in September 2013 by European cardholders. This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.[5]

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, ... V28 are the principal components obtained with PCA, the only features which have not been transformed with PCA are 'Time' and 'Amount'. Feature 'Time' contains the seconds elapsed between each transaction and the first transaction in the dataset. The feature 'Amount' is the transaction Amount, this feature can be used for example-dependent cost-sensitive learning. Feature 'Class' is the response variable and it takes value 1 in case of fraud and 0 otherwise.[5]

In this project, we utilize the dataset from Kaggle (https://www.kaggle.com/mlg-ulb/creditcardfraud). By comprehending the credit card fraud problem and exploring the dataset, we find the following characteristics.

1. As shown in Figure 1., the data has an extremely imbalanced distribution because over 99% of people have a good credit record.

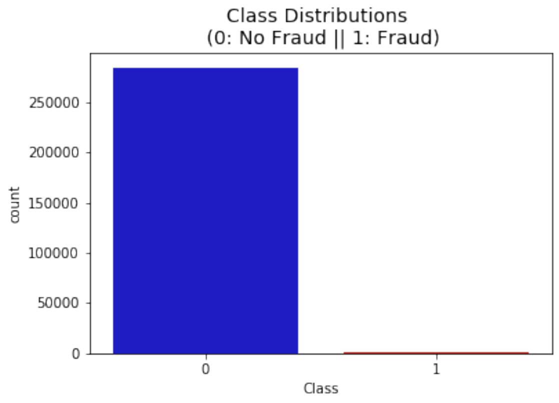


Figure 1. Distribution of Classes

2. The cost of false negative prediction and that of false positive prediction could be different. This is because committing a fraud transaction is definitely costlier than refusing to commit a normal transaction.
3. As mentioned in the documentation given by Kaggle, the correlation heat map shows that V1~V28 are mutually uncorrelated

[4] https://en.wikipedia.org/wiki/Credit_card_fraud
[5] https://www.kaggle.com/mlg-ulb/creditcardfraud

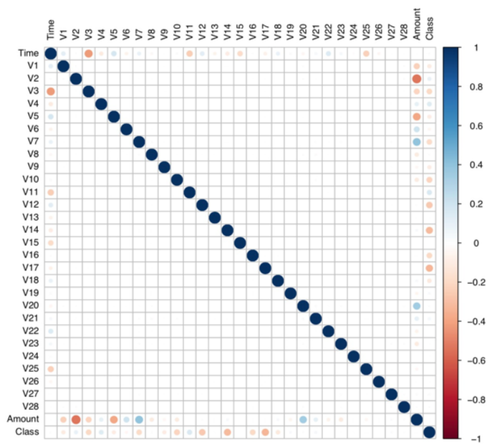


Figure 2. Correlation Heatmap

As a result, these characteristics results in that the conventional overall error rate fail to give satisfying evaluation for the models. Because the overall error rate gives equivalent weights to false negative prediction and false positive prediction, which is not consistent with the costs of different kinds of false prediction.

Section II. Methodology Design

In this case, we split the dataset as following graph. 70% of dataset is utilized to train models, 20% of dataset is utilized to train the threshold for given cost function, and 10% of data used to mimic the real prediction. (i. e. the last 10% of data is treated to be unknown until the final model and parameters are chosen.)

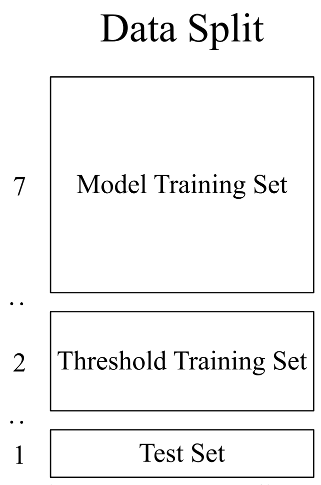


Figure 3. Data Split

In order to take use of prior knowledge, we only select classifiers that give responses, like probability, instead of direct classes, so we can adjust the threshold based on the weighted cost function. In this project, we select Logistic Regression, Naïve Bayes Classifier, Random Forest, and XGBoost as candidates.

As shown in Figure 4., we firstly train the models with model training set then optimize the thresholds such that the weighted cost functions are the lowest for each model. For a classifier with parameters set Θ, the weighted cost function is given below.

$$L(\Theta) = \alpha \times FP + \beta \times FN$$

Where α and β can be arbitrary positive number. Note that the weighted cost function is derived from prior knowledge of specific problem. For example, in following experiments, we assume that the cost of committing a fraud transaction is 50 times the cost of refusing to commit a normal transaction. Then, we assign $\alpha = 1$ and $\beta = 50$.

Finally, the algorithm outputs the model with globally lowest weighted cost function.

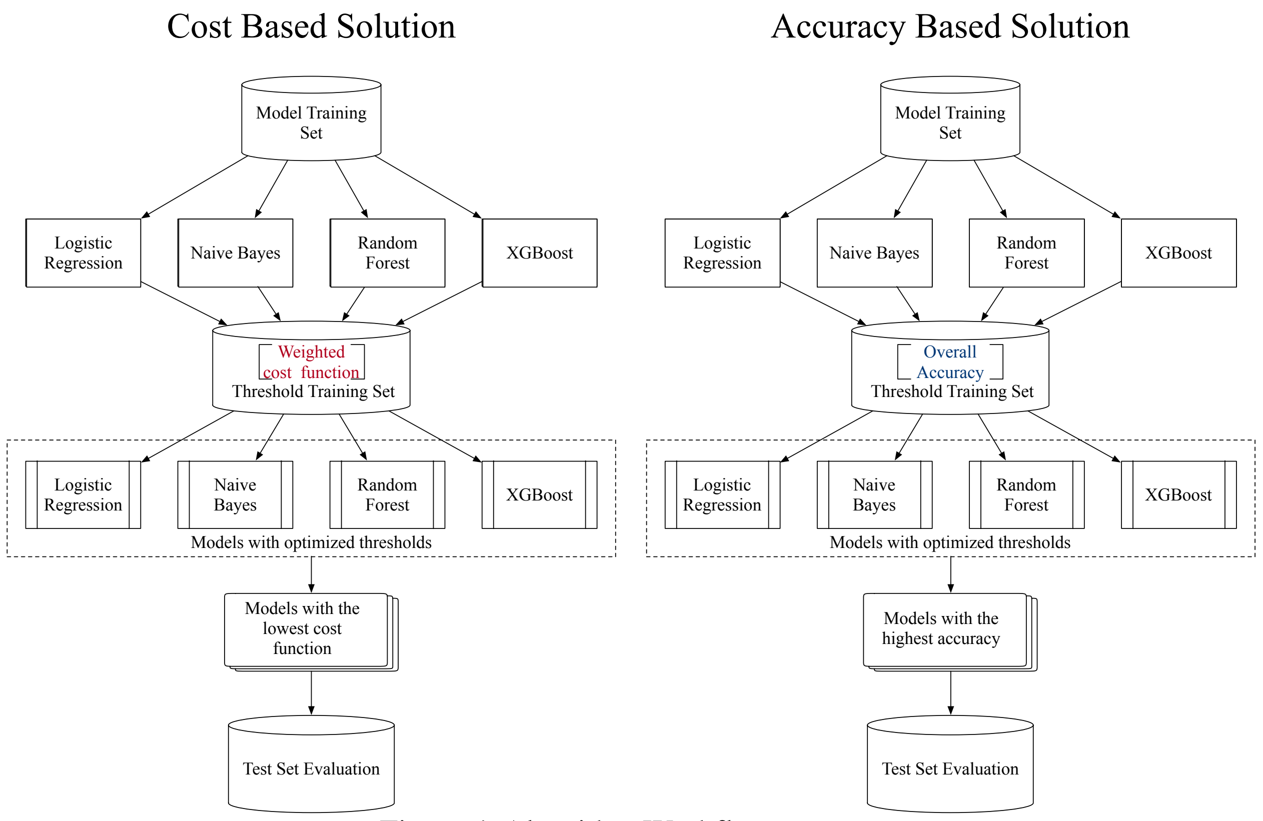


Figure 4. Algorithm Workflow

Section III. Model Comparison

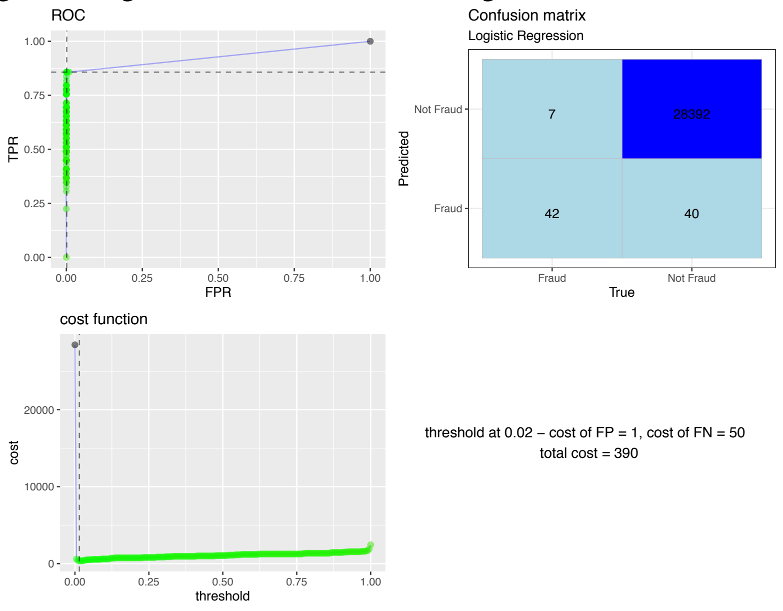
Logistic Regression & Naïve Bayes

Our first two model, logistic regression and Naive Bayes, are simple models that considering only the 28 PCA-transformed variables.

First of all, the test accuracy of both models is over 99%. Even if the accuracy is impressively high, we cannot draw a conclusion that, as the traditional solution did, the model is good because of the imbalance data set. Again, the model performance evaluation is totally based on the cost function with prior weights knowledge.

threshold	tpr	fpr	tp	fp	tn	fn	accuracy	cost	auc
0.0000000	1.0000000	1.0000000	99	56863	0	0	0.0017380	56863.00000	0.5000000
0.0050251	0.8888889	0.0090041	88	512	56351	11	0.9908184	1062.00000	0.9399424
0.0100503	0.8787879	0.0025324	87	144	56719	12	0.9972613	744.00000	0.9381277
0.0150754	0.8686869	0.0014772	86	84	56779	13	0.9982971	**734.00000**	0.9336048
0.0201005	0.8484848	0.0010376	84	59	56804	15	0.9987009	809.00000	0.9237236

Regarding the ROC graph on the top left corner, for most of the thresholds, the false positive rates are all tending to 0. This makes sense since the proportion of true negative is very large, just as what we can see on the second graph. This is the confusion matrix with the threshold equals to 0.015. As we introduced before, the threshold is selected by minimizing the weighted cost function. In this case, the mistakes made by predicting un-fraud account are 7 amount, mistakes made by predicting fraud account are 40 amount, and the corresponding lowest cost is 50*7+40*1=1298. The last picture is the cost function graph. The cost function is increasing, well it's not surprising because when the threshold moves up, more and more accounts tends to be predicted as not fraud, this mistake will be much costlier. Now we have the first result figure of logistic model on the following.



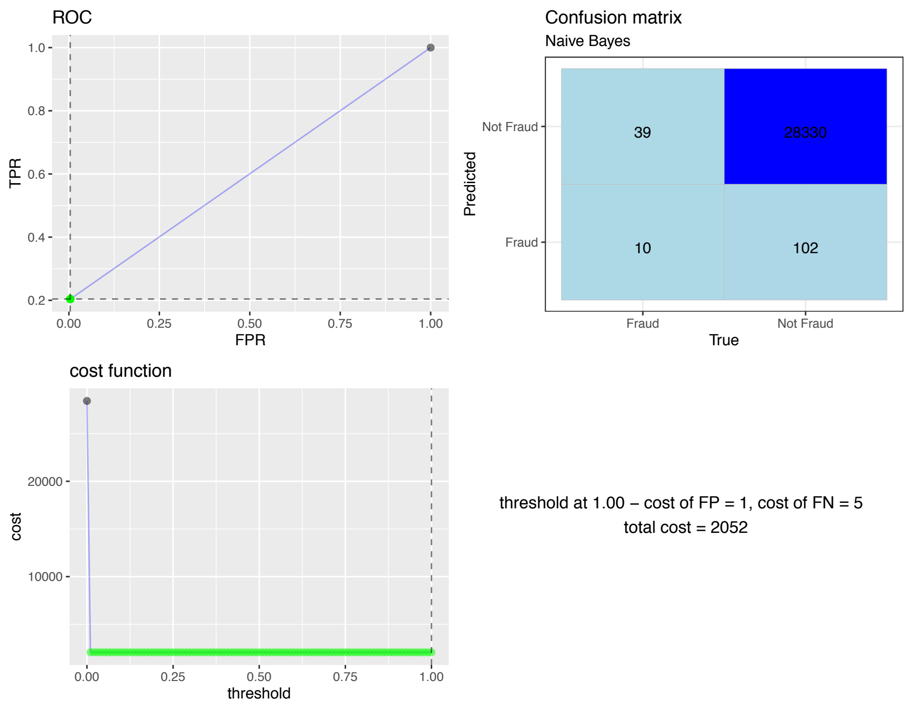
Considering the variable "t" and "amount", intuitively, they seems irrelevant to whether an account will fraud. In addition, since the two variables are not applied PCA transformation, the scale may be largely different from the other 28 variables and orthogonality does not fit for these two features. The following figures shows the result of logistic model containing "t" and "amount". The AUC, 0.9289 and cost performance, 744 are all weak than the model eliminating "t" and "amount"(0.9336 and 734) before.

threshold	tpr	fpr	tp	fp	tn	fn	accuracy	cost	auc
0.0000000	1.0000000	1.0000000	99	56863	0	0	0.0017380	56863.00000	0.5000000
0.0101010	0.8787879	0.0029545	87	168	56695	12	0.9968400	768.00000	0.9379167
0.0202020	0.8585859	0.0011079	85	63	56800	14	0.9986482	763.00000	0.9287390
0.0303030	0.8585859	0.0007738	85	44	56819	14	0.9989818	**744.00000**	0.9289060
0.0404040	0.8282828	0.0006859	82	39	56824	17	0.9990169	889.00000	0.9137985
0.0505051	0.7979798	0.0006331	79	36	56827	20	0.9990169	1036.00000	0.8986733

Using the package (glmulti), we select the solution variables of this function, i.e V3 + V7 + V10 + V11 + V12 + V14 + V16 + V17. However, even though the accuracy improves and the number of false positive cases drops dramatically, a sharp increase occurs on cost function (770). Since I apply AICc as criterion, perhaps the reason is that the solution focuses too much on the accuracy so that it sacrifices the accuracy in predicting negative cases but improve a lot on predicting positive cases. Nevertheless, we set 50:1 ratio so that the little lost are magnified.

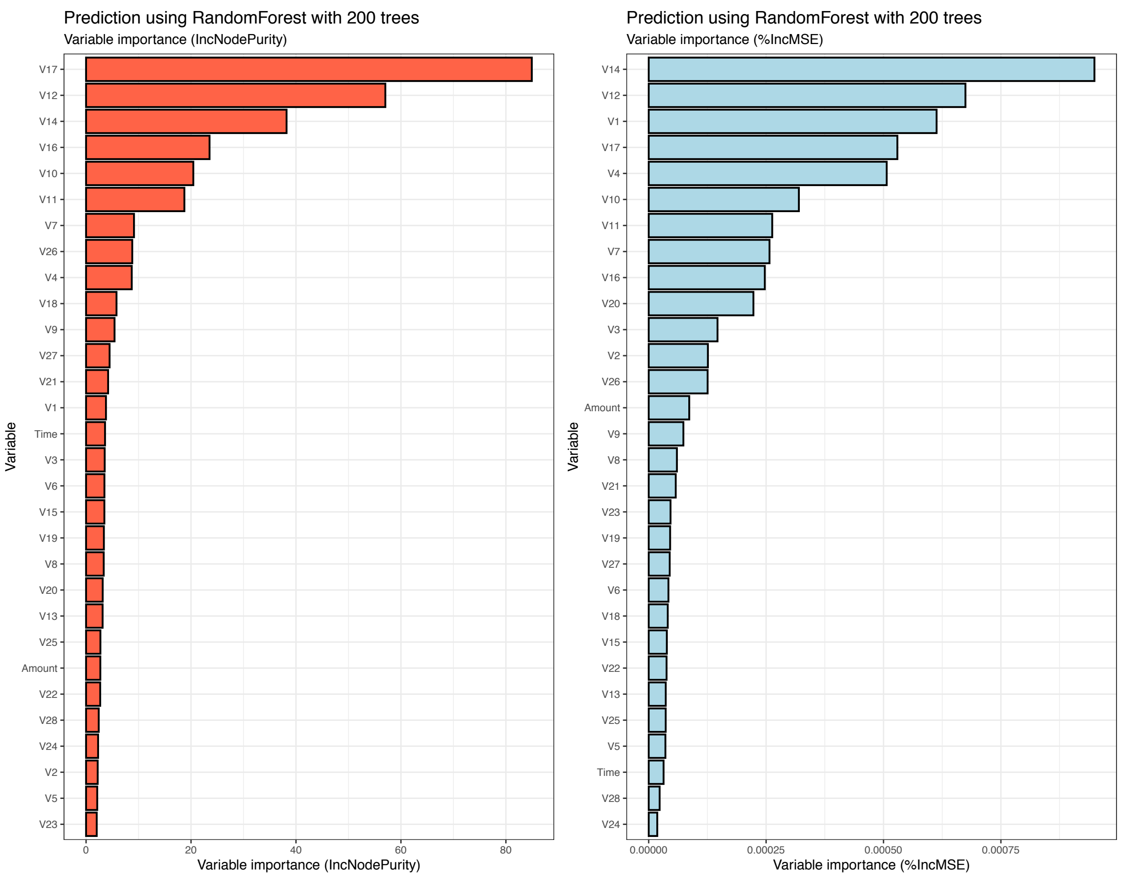
threshold	tpr	fpr	tp	fp	tn	fn	accuracy	cost	auc
0.0000000	1.0000000	1.0000000	99	56863	0	0	0.0017380	56863.00000	0.5000000
0.0050251	0.8787879	0.0090569	87	515	56348	12	0.9907482	1115.00000	0.9348655
0.0100503	0.8585859	0.0032886	85	187	56676	14	0.9964713	887.00000	0.9276486
0.0150754	0.8585859	0.0012662	85	72	56791	14	0.9984902	772.00000	0.9286598
0.0201005	0.8484848	0.0005803	84	33	56830	15	0.9991573	783.00000	0.9239523
0.0251256	0.8484848	0.0004045	84	23	56840	15	0.9993329	773.00000	0.9240402
0.0301508	0.8484848	0.0003517	84	20	56843	15	0.9993856	**770.00000**	0.9240666

To compare with the logistic model, we apply several simple classification methods to evaluate the cost function performance, such as Naive Bayes, LDA and QDA. The result shows that logistic regression has the best performance among these models. For example, as we can see the cost of Naive Bayes is nearly 6 times more than logistic model. So why is the logistic model performing so well with these data? Reminding that our dataset is transformed by PCA, so that these 28 variables are all orthogonal between each two of them. It means that there is no multi linearity among these variables. The independence of these features implies that the logistic regression will work so well.

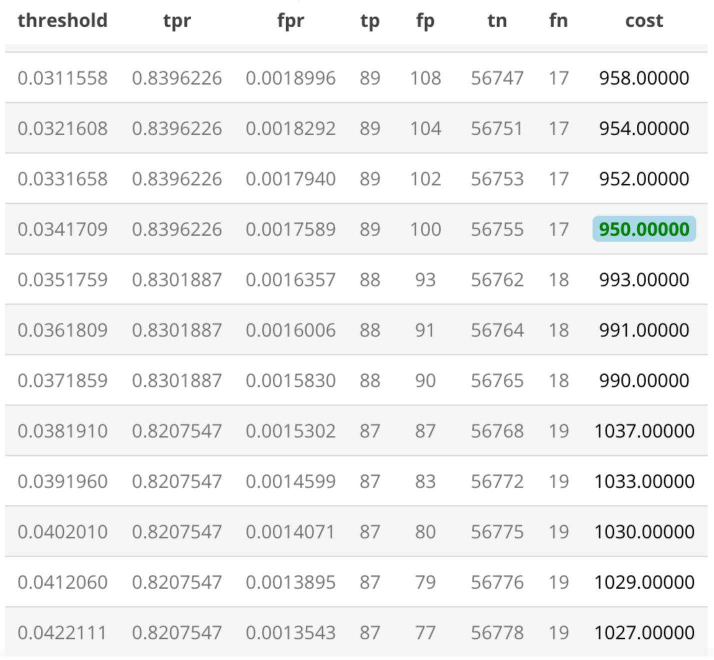


Random Forest

To obtain a more robust classifier, we directly adopted the random forest method with 200 trees to ensure that the correlation between each classifier is relatively low. After fitting the model on the training set, the variable importance graph is displayed as below:

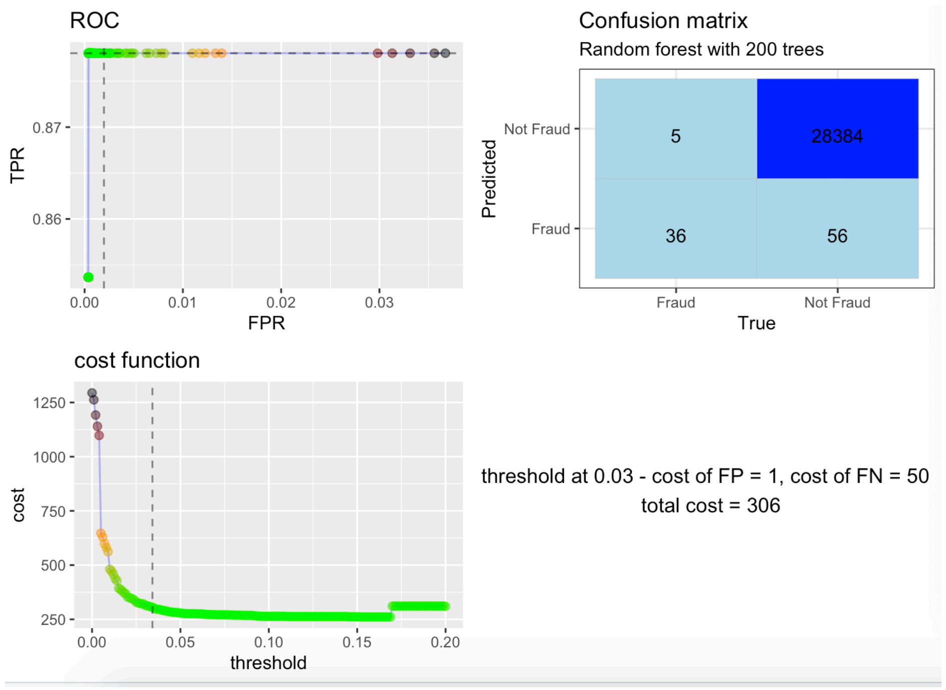


However, it should be aware that the variables are masked from the PCA so it is actually hard to interpret such result. Then we fit the model in the threshold evaluation set for threshold selection, and derived the following result:



threshold	tpr	fpr	tp	fp	tn	fn	cost
0.0311558	0.8396226	0.0018996	89	108	56747	17	958.00000
0.0321608	0.8396226	0.0018292	89	104	56751	17	954.00000
0.0331658	0.8396226	0.0017940	89	102	56753	17	952.00000
0.0341709	0.8396226	0.0017589	89	100	56755	17	**950.00000**
0.0351759	0.8301887	0.0016357	88	93	56762	18	993.00000
0.0361809	0.8301887	0.0016006	88	91	56764	18	991.00000
0.0371859	0.8301887	0.0015830	88	90	56765	18	990.00000
0.0381910	0.8207547	0.0015302	87	87	56768	19	1037.00000
0.0391960	0.8207547	0.0014599	87	83	56772	19	1033.00000
0.0402010	0.8207547	0.0014071	87	80	56775	19	1030.00000
0.0412060	0.8207547	0.0013895	87	79	56776	19	1029.00000
0.0422111	0.8207547	0.0013543	87	77	56778	19	1027.00000

Thus, from the evaluation process, it seems that the threshold of 0.034 would be the most reasonable one, we can then have the estimate of cost using random forest model with threshold = 0.034:

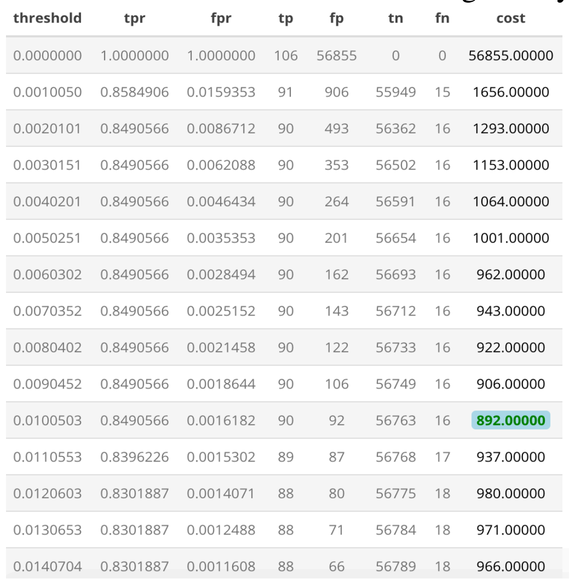


threshold at 0.03 - cost of FP = 1, cost of FN = 50
total cost = 306

Thus the total cost the on the test set is estimated to be about 306, which indicates that the random forest model is actually performing worse than the logistic regression, so we also tried some boosting-related methods to see if there is any improvement.

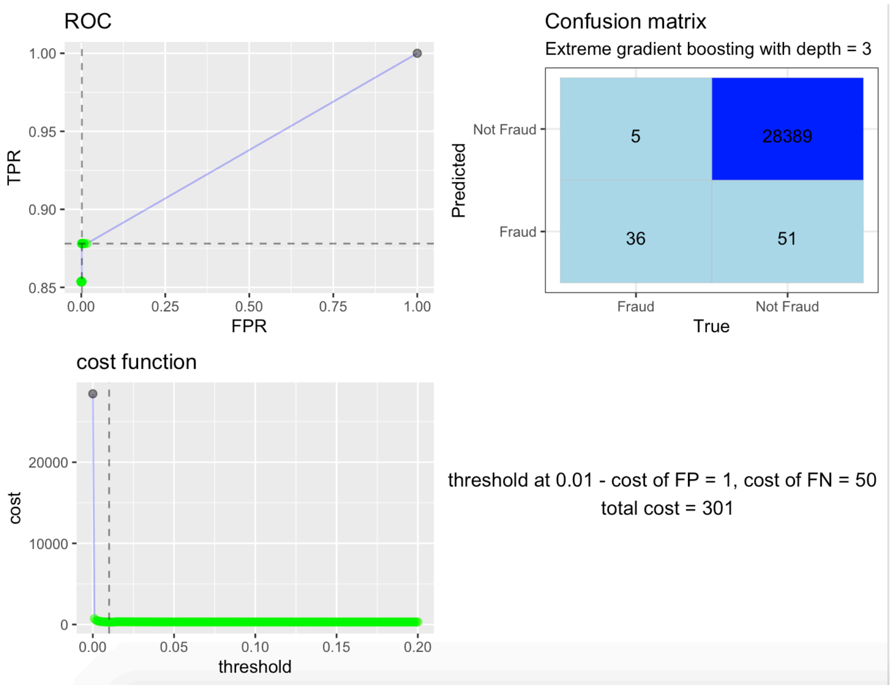
Boosting

Among all boosting method, we tried the gradient boosting and ada boosting while the number of trees were taken as 500, 1000, 1500 and 2000 separately. As for the extreme gradient boosting, we adopted depth of 3 and 7 for discussion. Following the similar steps mentioned above, the cost on the test set of every model is estimated and compared. From the comparison results, the extreme gradient boosting with depth of 3 generates the lowest estimated cost on the test set. The threshold evaluation table is given by:



threshold	tpr	fpr	tp	fp	tn	fn	cost
0.0000000	1.0000000	1.0000000	106	56855	0	0	56855.00000
0.0010050	0.8584906	0.0159353	91	906	55949	15	1656.00000
0.0020101	0.8490566	0.0086712	90	493	56362	16	1293.00000
0.0030151	0.8490566	0.0062088	90	353	56502	16	1153.00000
0.0040201	0.8490566	0.0046434	90	264	56591	16	1064.00000
0.0050251	0.8490566	0.0035353	90	201	56654	16	1001.00000
0.0060302	0.8490566	0.0028494	90	162	56693	16	962.00000
0.0070352	0.8490566	0.0025152	90	143	56712	16	943.00000
0.0080402	0.8490566	0.0021458	90	122	56733	16	922.00000
0.0090452	0.8490566	0.0018644	90	106	56749	16	906.00000
0.0100503	0.8490566	0.0016182	90	92	56763	16	892.00000
0.0110553	0.8396226	0.0015302	89	87	56768	17	937.00000
0.0120603	0.8301887	0.0014071	88	80	56775	18	980.00000
0.0130653	0.8301887	0.0012488	88	71	56784	18	971.00000
0.0140704	0.8301887	0.0011608	88	66	56789	18	966.00000

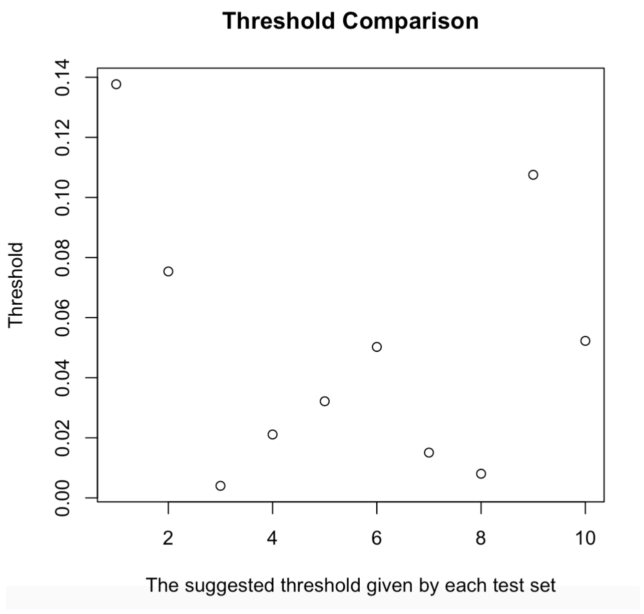
Then taking the threshold as 0.01, the estimated cost on the test set is:



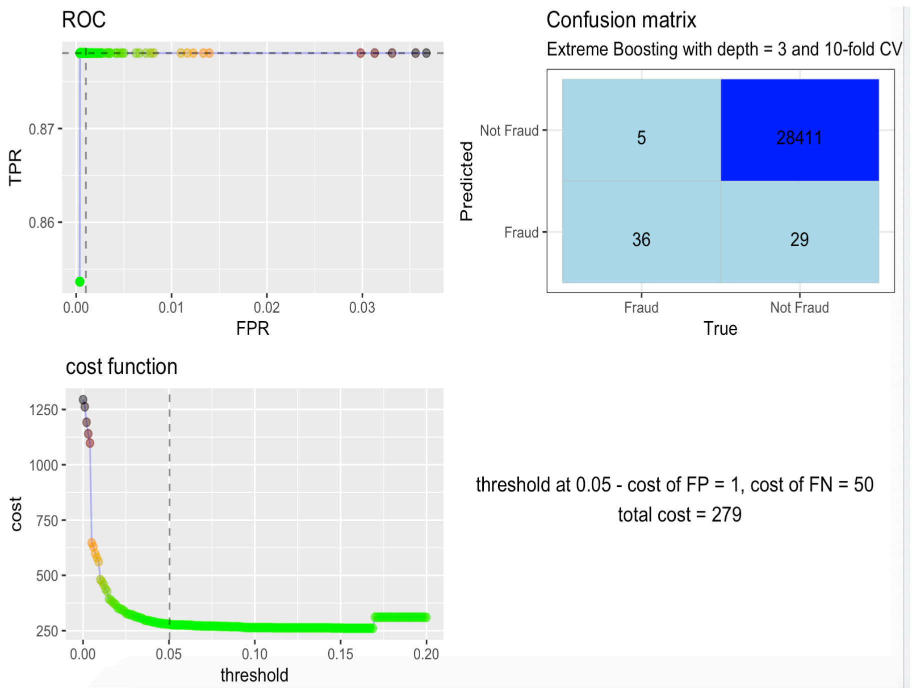
threshold at 0.01 - cost of FP = 1, cost of FN = 50
total cost = 301

The estimated cost is about 301 on test data and this was the best result that we had achieved.

However, up to now, the threshold evaluation based on the evaluation set seems to be a little greedy. Thus, for the extreme gradient boosting model, we tried to use 10-kold cross validation only on the training set to get an averaged threshold, and only applied it on the test set to check whether such method could generate a more robust threshold in the sacrifice of more training time.



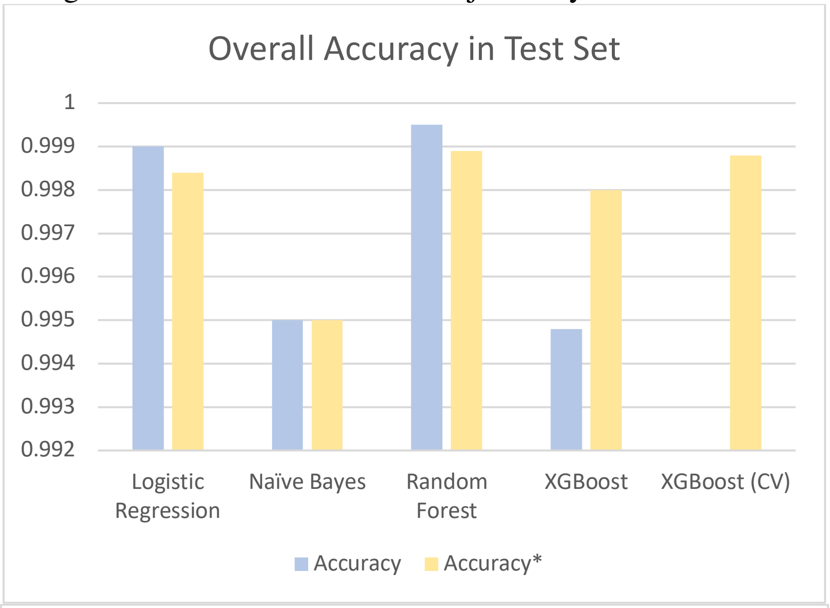
From the above plot, it seems that if a threshold is determined only through one test set, the variance is non-negligible and averaging is necessary if the actual requirement weighs more on the accuracy of threshold than the training speed. Here is the result obtained using the averaged threshold in the test set:

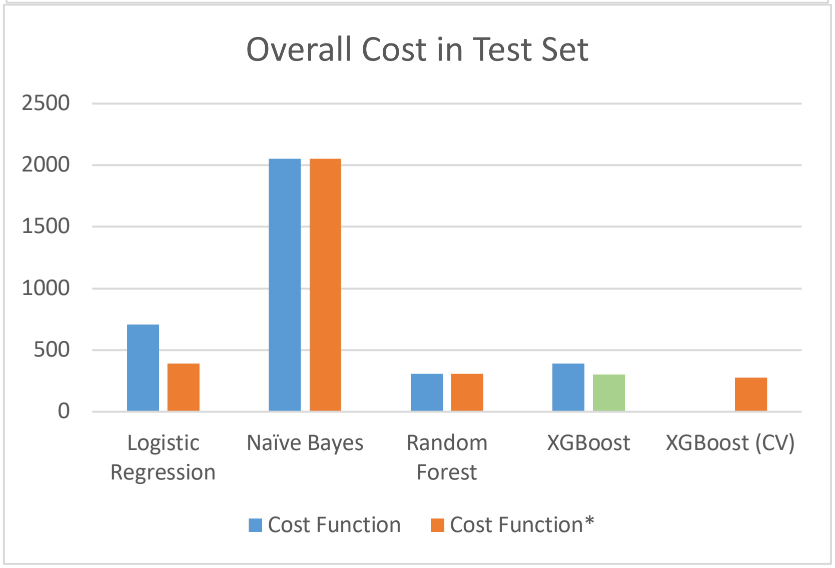


It can be concluded that the averaged threshold indeed has a more satisfactory performance on the test data, which indicates that in practical cases, when time is adequate and the model is required to be as accurate as possible, the best method should be trained on all available data and the best threshold should be determined by averaging all the suggested thresholds given by cross-validation method.

Final Comparison in test set

The sign * means that the model is adjusted by cost-based solution.





Section IV. Summary and Further Discussion

In this project, we already showed that models for extremely imbalanced data cannot be directly evaluated by accuracy, because of the imbalanced data and difference between the costs of two kinds of false prediction.

XGBoost performs best in the experiment, which is chosen to be our final model. However, although XGBoost and Random Forest give lower cost function, Logistic regression is more efficient and has higher scalability. This is also the reason why Logistic regression is widely used in industry.

The advantage of our solution is that we utilize flexible weighted cost function to simulate real world decision. Even though the weights of cost function can change, we only need to adjust the threshold based on threshold training set instead of repeating the whole training process.

Also, our experiment has its disadvantages. The binary classifiers like KNN, SVM, and so on are not suitable for this algorithm because they do not have thresholds. Due to limited time and computing resource, we did not adjust other parameters in models or implement cross validation to improve the model performance among all models.

For further improvement, one can try oversampling or undersampling to handle imbalanced data. Additionally, as, shown in the Boosting, cross validation can be implemented to find more precise thresholds.

```r
############################
## ECO 3080 Group Project ##
############################


library(xgboost)
library(randomForest)
library(caret)
library(gridExtra)
library(grid)
library(ggplot2)
library(lattice)
library(corrplot)
library(pROC)
library(knitr)
library(kableExtra)
library(formattable)
library(dplyr)
library(e1071)
library(readr)
library(caTools)
library(gbm)
library(glmulti)
library(rJava)
library(leaps)
library(e1071)
library(infotheo)


calculate_roc <- function(verset, cost_of_fp, cost_of_fn, n=200) {
  tp <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 1)
  }
  fp <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 0)
  }
  tn <- function(verset, threshold) {
    sum(verset$predicted < threshold & verset$Class == 0)
  }
  fn <- function(verset, threshold) {
    sum(verset$predicted < threshold & verset$Class == 1)
  }
  tpr <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 1) / sum(verset$Class == 1)
  }
  fpr <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 0) / sum(verset$Class == 0)
  }
  cost <- function(verset, threshold, cost_of_fp, cost_of_fn) {
    sum(verset$predicted >= threshold & verset$Class == 0) * cost_of_fp +
      sum(verset$predicted < threshold & verset$Class == 1) * cost_of_fn
  }

  roc <- data.frame(threshold = seq(0,0.2,length.out=n), tpr=NA, fpr=NA)
  roc$tp <- sapply(roc$threshold, function(th) tp(verset, th))
  roc$fp <- sapply(roc$threshold, function(th) fp(verset, th))
  roc$tn <- sapply(roc$threshold, function(th) tn(verset, th))
  roc$fn <- sapply(roc$threshold, function(th) fn(verset, th))
  roc$tpr <- sapply(roc$threshold, function(th) tpr(verset, th))
  roc$fpr <- sapply(roc$threshold, function(th) fpr(verset, th))
  roc$cost <- sapply(roc$threshold, function(th) cost(verset, th, cost_of_fp, cost_of_fn))
  return(roc)
}

#####the following function are used in Logistic and Naive Bayes
plot_confusion_matrix <- function(verset, sSubtitle, threshold) {
  tst <- data.frame(
    ifelse(verset$predicted < threshold, 0, 1),
    verset$Class
  )
  opts <-  c("Predicted", "True")
  names(tst) <- opts
  cf <- plyr::count(tst)
  cf[opts][cf[opts]==0] <- "Not Fraud"
  cf[opts][cf[opts]==1] <- "Fraud"

  ggplot(data =  cf, mapping = aes(x = True, y = Predicted)) +
    labs(title = "Confusion matrix", subtitle = sSubtitle) +
    geom_tile(aes(fill = freq), colour = "grey") +
    geom_text(aes(label = sprintf("%1.0f", freq)), vjust = 1) +
    scale_fill_gradient(low = "lightblue", high = "blue") +
    theme_bw() + theme(legend.position = "none")
}

plot_roc_logistic_bayes <- function(roc, threshold, cost_of_fp, cost_of_fn, testset,sSubtitle) {
  library(gridExtra)

  norm_vec <- function(v) (v - min(v))/diff(range(v))

  idx_threshold = which.min(abs(roc$threshold-threshold))

  col_ramp <- colorRampPalette(c("green","orange","red","black"))(100)
  col_by_cost <- col_ramp[ceiling(norm_vec(roc$cost)*99)+1]
  p_roc <- ggplot(roc, aes(fpr,tpr)) +
    geom_line(color=rgb(0,0,1,alpha=0.3)) +
    geom_point(color=col_by_cost, size=2, alpha=0.5) +
    labs(title = sprintf("ROC")) + xlab("FPR") + ylab("TPR") +
    geom_hline(yintercept=roc[idx_threshold,"tpr"], alpha=0.5, linetype="dashed") +
    geom_vline(xintercept=roc[idx_threshold,"fpr"], alpha=0.5, linetype="dashed")

  p_matrix = plot_confusion_matrix(testset,sSubtitle,threshold)

  p_cost <- ggplot(roc, aes(threshold, cost)) +
    geom_line(color=rgb(0,0,1,alpha=0.3)) +
    geom_point(color=col_by_cost, size=2, alpha=0.5) +
    labs(title = sprintf("cost function")) +
    geom_vline(xintercept=threshold, alpha=0.5, linetype="dashed")

  sub_title <- sprintf("threshold at %.2f - cost of FP = %d, cost of FN = %d \n total cost = %d",
    threshold, cost_of_fp, cost_of_fn ,roc$cost[which.min(abs(roc$threshold-threshold))])
  #
  grid.arrange(p_roc, p_matrix, p_cost, ncol=2,sub=textGrob(sub_title, gp=gpar(cex=1), just="bottom"))
}

calculate_roc_accuracy <- function(verset, cost_of_fp, cost_of_fn, n=100) {

  tp <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 1)
  }

  fp <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 0)
  }

  tn <- function(verset, threshold) {
    sum(verset$predicted < threshold & verset$Class == 0)
  }

  fn <- function(verset, threshold) {
    sum(verset$predicted < threshold & verset$Class == 1)
  }

  tpr <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 1) / sum(verset$Class == 1)
  }
  fpr <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 0) / sum(verset$Class == 0)
  }

  cost <- function(verset, threshold, cost_of_fp, cost_of_fn) {
    sum(verset$predicted >= threshold & verset$Class == 0) * cost_of_fp +
      sum(verset$predicted < threshold & verset$Class == 1) * cost_of_fn
  }
  fpr <- function(verset, threshold) {
    sum(verset$predicted >= threshold & verset$Class == 0) / sum(verset$Class == 0)
  }

  threshold_round <- function(value, threshold)
  {
    return (as.integer(!(value < threshold)))
  }
  #calculate AUC (https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve)
  auc_ <- function(verset, threshold) {
    auc(verset$Class, threshold_round(verset$predicted,threshold))
  }

  roc <- data.frame(threshold = seq(0,1,length.out=n), tpr=NA, fpr=NA)
  roc$tp <- sapply(roc$threshold, function(th) tp(verset, th))
  roc$fp <- sapply(roc$threshold, function(th) fp(verset, th))
  roc$tn <- sapply(roc$threshold, function(th) tn(verset, th))
  roc$fn <- sapply(roc$threshold, function(th) fn(verset, th))
  roc$accuracy = (roc$tp+roc$tn)/(roc$tn+roc$tp+roc$fp+roc$fn)
  roc$tpr <- sapply(roc$threshold, function(th) tpr(verset, th))
  roc$fpr <- sapply(roc$threshold, function(th) fpr(verset, th))
  roc$cost <- sapply(roc$threshold, function(th) cost(verset, th, cost_of_fp, cost_of_fn))
  roc$auc <-  sapply(roc$threshold, function(th) auc_(verset, th))

  return(roc)
}

accuracy_table = function(verroc){
  maxaccur <- max(verroc$accuracy)
  verroc %>%
    mutate(
      auc = ifelse(accuracy == maxaccur,
                   cell_spec(sprintf("%.5f", accuracy), "html",
                    color = "green", background = "lightblue", bold = T),
                   cell_spec(sprintf("%.5f", accuracy), "html",
                    color = "black", bold = F))
    ) %>%
    kable( "html", escape=F, align="c") %>%
    kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
    scroll_box(height = "600px")
}

cost_table = function(verroc){
  mincost <- min(verroc$cost)
  verroc %>%
    mutate(
      cost = ifelse(cost == mincost,
                    cell_spec(sprintf("%.5f", cost), "html",
                        color = "green", background = "lightblue", bold = T),
                    cell_spec(sprintf("%.5f", cost), "html",
                        color = "black", bold = F))
    ) %>%
    kable( "html", escape=F, align="c") %>%
    kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
    scroll_box(height = "600px")
}
####################

plot_roc <- function(roc, threshold, cost_of_fp, cost_of_fn) {
```

```r
  library(gridExtra)
  norm_vec <- function(v) (v - min(v))/diff(range(v))
  idx_threshold = which.min(abs(roc$threshold-threshold))

  col_ramp <- colorRampPalette(c("green","orange","red","black"))(100)
  col_by_cost <- col_ramp[ceiling(norm_vec(roc$cost)*99)+1]
  p_roc <- ggplot(roc, aes(fpr,tpr)) +
    geom_line(color=rgb(0,0,1,alpha=0.3)) +
    geom_point(color=col_by_cost, size=2, alpha=0.5) +
    labs(title = sprintf("ROC")) + xlab("FPR") + ylab("TPR") +
    geom_hline(yintercept=roc[idx_threshold,"tpr"], alpha=0.5, linetype="dashed") +
    geom_vline(xintercept=roc[idx_threshold,"fpr"], alpha=0.5, linetype="dashed") +
    scale_color_manual(palette="Spectral")

  p_conf <- plot_confusion_matrix(final.veriset, "Extreme Boosting with depth = 3 and 10-fold CV", threshold)

  p_cost <- ggplot(roc, aes(threshold, cost)) +
    geom_line(color=rgb(0,0,1,alpha=0.3)) +
    geom_point(color=col_by_cost, size=2, alpha=0.5) +
    labs(title = sprintf("cost function")) +
    geom_vline(xintercept=threshold, alpha=0.5, linetype="dashed")+
    scale_color_manual(palette="Spectral")

  sub_title <- sprintf("threshold at %.2f - cost of FP = %d, cost of FN = %d \n total cost = %d",
                       threshold, cost_of_fp, cost_of_fn, roc$cost[which.min(abs(roc$threshold-threshold))])

  grid.arrange(p_roc, p_conf, p_cost, ncol=2,sub=textGrob(sub_title, gp=gpar(cex=1), just="bottom"))
}

plot_confusion_matrix <- function(verset, sSubtitle, threshold=0.5) {
  tst <- data.frame(
    ifelse(verset$predicted < threshold, 0, 1),
    verset$Class
  )
  opts <-  c("Predicted", "True")
  names(tst) <- opts
  cf <- plyr::count(tst)
  cf[opts][cf[opts]==0] <- "Not Fraud"
  cf[opts][cf[opts]==1] <- "Fraud"

  ggplot(data =  cf, mapping = aes(x = True, y = Predicted)) +
    labs(title = "Confusion matrix", subtitle = sSubtitle) +
    geom_tile(aes(fill = freq), colour = "grey") +
    geom_text(aes(label = sprintf("%1.0f", freq)), vjust = 1) +
    scale_fill_gradient(low = "lightblue", high = "blue") +
    theme_bw() + theme(legend.position = "none")

}


############
setwd("~/Downloads")
pro_data = read.csv("creditcard.csv",header = T)

#attach(pro_data)
pro_data = pro_data[,-1]
set.seed(1)
pro.train = sample(1:nrow(pro_data),round(nrow(pro_data)*7/10))
pro.train.data = pro_data[pro.train,]
pro.test = pro_data[-pro.train,]
set.seed(1)
pro.thre = sample(1:nrow(pro.test),round(nrow(pro.test)*2/3))
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]

check = function(testpred,veri){    #####    return the threshold that minimize cost
  veri$predicted = testpred
  roc <- calculate_roc(veri, 1, 50, n = 200)
  a = roc$threshold[which.min(roc$cost)]
  return(a)
}

#logistic Regression and Naive Bayes
train = pro.train
cv = pro.thre.set
testset = final.veriset

#logistic Regression
glm.model <- glm(Class ~ ., data = train, family = "binomial", control = list(maxit = 50))
glm.predict <- predict(glm.model, cv, type = "response")
logtest = cv
logtest$Class = as.numeric(as.character(logtest$Class))
logtest$predicted = predict(glm.model, cv, type = "response")

######logistic but train with time and amount
glm.modeltm = glm(Class ~ ., data = train_with_time_and_amount,
    family = "binomial", control = list(maxit = 50))
logtesttm = cv
logtesttm$Class = as.numeric(as.character(logtesttm$Class))
logtesttm$predicted = predict(glm.modeltm, cv, type = "response")


######train with selected variable by glmulti
analysismodel = glmulti(glm.model,level = 1, crit = 'aicc', method = 'l')
summary(analysismodel)# The result is 1 + V3 + V7 + V10 + V11 + V12 + V14 + V16 + V17
glm.model2 = glm(Class ~ V3 + V7 + V10 + V11 + V12 + V14 + V16 + V17,
    data = train, family = "binomial", control = list(maxit = 50))
glm.predict2  <- predict(glm.model2, cv, type = "response")
logtest2 = cv
logtest2$Class = as.numeric(as.character(logtest2$Class))
logtest2$predicted = glm.predict2

######predict with test set
glm.predict.test <- predict(glm.model, testset, type = "response")
logtestfinal = testset
logtestfinal$Class = as.numeric(as.character(logtestfinal$Class))
logtestfinal$predicted = glm.predict.test

#plot the graph
roc_log <- calculate_roc_accuracy(logtest, 1, 50, n = 200)
roc_logtm <- calculate_roc_accuracy(logtesttm, 1, 50, n = 100)
roc_log2 = calculate_roc_accuracy(logtest2, 1, 50, n = 200)
roc_logfinal = calculate_roc_accuracy(logtestfinal, 1, 50, n = 200)

cost_table(roc_log)#threshold=0.0150754
cost_table(roc_logfinal)###find the cost with threshold = 0.0150754

accuracy_table(roc_logfinal) #accuracy 0.9926 with cost 707

plot_roc_logistic_bayes(roc_logfinal, 0.0150754, 1, 50,logtestfinal,"Logistic Regression")

#Naive Bayes
trainnb = discretize(train[1:28],"equalwidth",5)
trainnb = 6 - trainnb
trainnb$Class = train$Class
nb = naiveBayes(Class~., data = trainnb, laplace=1)
nbpredict = predict(nb,cv,type = 'raw')
nb_test = cv
nb_test$predicted = nbpredict[,1]
nb_test$Class = as.numeric(as.character(nb_test$Class))

###Bayes predict with test set
nbpredict_final = predict(nb,testset,type = 'raw')
nb_test_final = testset
nb_test_final$predicted = nbpredict_final[,1]
nb_test_final$Class = as.numeric(as.character(nb_test_final$Class))
```

```
###plot
roc_nb <- calculate_roc_accuracy(nb_test, 1, 50, n = 100)
roc_nb_final<- calculate_roc_accuracy(nb_test_final, 1, 50, n = 100)

cost_table(roc_nb)#threshold = 1
cost_table(roc_nb_final)

plot_roc_logistic_bayes(roc_nb_final, 1, 1, 50,nb_test_final,"Naive Bayes")

#Random Forest
rf.train <- randomForest(Class~., data = pro.train.data, ntree= 200, importance = T, do.trace = T)

varimp <- data.frame(rf_train$importance)
vi1 <- ggplot(varimp, aes(x=reorder(rownames(varimp),IncNodePurity), y=IncNodePurity)) +
  geom_bar(stat="identity", fill="tomato", colour="black") +
  coord_flip() + theme_bw(base_size = 8) +
  labs(title="Prediction using RandomForest with 100 trees",
       subtitle="Variable importance (IncNodePurity)",
       x="Variable", y="Variable importance (IncNodePurity)")

vi2 <- ggplot(varimp, aes(x=reorder(rownames(varimp),X.IncMSE), y=X.IncMSE)) +
  geom_bar(stat="identity", fill="lightblue", colour="black") +
  coord_flip() + theme_bw(base_size = 8) +
  labs(title="Prediction using RandomForest with 100 trees",
       subtitle="Variable importance (%IncMSE)",
       x="Variable", y="Variable importance (%IncMSE)")
grid.arrange(vi1, vi2, ncol=2)

#Threshold evaluation:
rf.eva.pred = predict(rf.train,pro.thre.set,type='response')
pro.thre.set$predicted = rf.eva.pred
check(rf.eva.pred,pro.thre.set)

#Comparison to the model without using cost evaluation
range(rf.eva.pred)
x = seq(0.1,0.3,0.01)
Comparison = function(x,testpred){
  b = c(); c = c();
  error = c()
  j  = 1
  for (i in x){
    predict.CLASS = as.numeric(ifelse(testpred > i,"1","0" ))
    b =  c(b,table(predict.CLASS, pro.thre.set$Class)[1,2])
    c =  c(c,table(predict.CLASS ,pro.thre.set$Class)[2,1])
    error[j] =(b[j]+c[j]) / sum(table(predict.CLASS, pro.thre.set$Class))
    j = j + 1
  }
  plot(error)
  (x[which.min(error)])
}
Comparison(x,rf.eva.pred)
# The threshold that minimize the overall accuracy is 0.26

rf.eva.test = predict(rf.train,final.veriset,type='response')
predict.CLASS = as.numeric(ifelse(rf.eva.test > 0.26,"1","0" ))
table(predict.CLASS, final.veriset$Class)
table(predict.CLASS, final.veriset$Class)[1,2]*50 + table(predict.CLASS, final.veriset$Class)[2,1]*1

#Estimate the test cost
roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
```

```r
                      color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

rf.eva.test = predict(rf.train,final.veriset,type='response')
final.veriset$predicted <- rf.eva.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#Gradient Boosting
(pro.train.data$Class = as.factor(pro.train.data$Class))
(pro.thre.set$Class = as.factor(pro.thre.set$Class))
(final.veriset$Class = as.factor(final.veriset$Class))
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
#500
boost.project.500 = gbm(Class ~., data = pro.train.data,
                        distribution = "bernoulli",  n.trees = 500,
                        shrinkage =  0.01,cv.folds = 3)
summary(boost.project.500) # Variable influence
boost.iter.500 = gbm.perf(boost.project.500,method = "cv")
pro.predict.500 = predict(boost.project.500,pro_data[-pro.train,],
    n.trees = boost.iter.500,type='response')

boost.500.eva.pred = predict(boost.project.500,pro.thre.set,
    n.trees = boost.iter.500,type='response')
pro.thre.set$predicted = boost.500.eva.pred
check(boost.500.eva.pred,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

boost.500.test = predict(boost.project.500,final.veriset,n.trees = boost.iter.500,type='response')
final.veriset$predicted <- boost.500.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#1000
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
boost.project.1000 = gbm(Class ~., data = pro.train.data,
                distribution = "bernoulli",  n.trees = 1000,
                shrinkage =  0.01,cv.folds = 3)
summary(boost.project.1000) # Variable influence
boost.iter.1000 = gbm.perf(boost.project.1000,method = "cv")
pro.predict.1000 = predict(boost.project.1000,pro_data[-pro.train,],
    n.trees = boost.iter.1000,type='response')

boost.1000.eva.pred = predict(boost.project.1000,pro.thre.set,
    n.trees = boost.iter.1000,type='response')
pro.thre.set$predicted = boost.1000.eva.pred
check(boost.1000.eva.pred,pro.thre.set)
```

```r
roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

boost.1000.test = predict(boost.project.1000,final.veriset,
    n.trees = boost.iter.1000,type='response')
final.veriset$predicted <- boost.1000.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#1500
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
boost.project.1500 = gbm(Class ~., data = pro.train.data,
                 distribution = "bernoulli",  n.trees = 1500,
                 shrinkage =  0.01,cv.folds = 3)
summary(boost.project.1500) # Variable influence
boost.iter.1500 = gbm.perf(boost.project.1500,method = "cv")
pro.predict.1500 = predict(boost.project.1500,pro_data[-pro.train,],
    n.trees = boost.iter.1500,type='response')

boost.1500.eva.pred = predict(boost.project.1500,pro.thre.set,
    n.trees = boost.iter.1500,type='response')
pro.thre.set$predicted = boost.1500.eva.pred
check(boost.1500.eva.pred,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

boost.1500.test = predict(boost.project.1500,final.veriset,
    n.trees = boost.iter.1500,type='response')
final.veriset$predicted <- boost.1500.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#2000
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
boost.project.2000 = gbm(Class ~., data = pro.train.data,
                 distribution = "bernoulli",  n.trees = 2000,
                 shrinkage =  0.01,cv.folds = 3)
summary(boost.project.2000) # Variable influence
boost.iter.2000 = gbm.perf(boost.project.2000,method = "cv")
pro.predict.2000 = predict(boost.project.2000,pro_data[-pro.train,],
    n.trees = boost.iter.2000,type='response')
```

```r
boost.2000.eva.pred = predict(boost.project.2000,pro.thre.set,
    n.trees = boost.iter.2000,type='response')
pro.thre.set$predicted = boost.2000.eva.pred
check(boost.2000.eva.pred,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

boost.2000.test = predict(boost.project.2000,final.veriset,n.trees = boost.iter.2000,type='response')
final.veriset$predicted <- boost.2000.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

# Extreme boost with depth = 3
pro_data = read.csv("creditcard.csv",header = T)
pro_data = pro_data[,-1]
set.seed(1)
pro.train = sample(1:nrow(pro_data),round(nrow(pro_data)*7/10))
pro.train.data = pro_data[pro.train,]
pro.test = pro_data[-pro.train,]
set.seed(1)
pro.thre = sample(1:nrow(pro.test),round(nrow(pro.test)*2/3))
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]

xgb.data.train = xgb.DMatrix(as.matrix(pro.train.data[,1:29]), label = pro.train.data$Class )
xgb.data.test = xgb.DMatrix(as.matrix(pro.thre.set[,1:29]), label = pro.thre.set$Class)
xgb.model = xgboost(xgb.data.train, max.depth = 3, nround = 100, objective = "binary:logistic")
xgb.eva.test = predict(xgb.model, newdata = as.matrix(pro.thre.set[,1:29]), ntreelimit =
                     xgb.model$best_ntreelimit, type = "response")
pro.thre.set$predicted <- xgb.eva.test
check(xgb.eva.test,pro.thre.set)

range(xgb.eva.test)
x = seq(0.001,0.2,0.001)
Comparison = function(x,testpred){
  b = c(); c = c();
  error = c()
  j  = 1
  for (i in x){
    predict.CLASS = as.numeric(ifelse(testpred > i,"1","0" ))
    b =  c(b,table(predict.CLASS, pro.thre.set$Class)[1,2])
    c =  c(c,table(predict.CLASS ,pro.thre.set$Class)[2,1])
    error[j] =(b[j]+c[j]) / sum(table(predict.CLASS, pro.thre.set$Class))
    j = j + 1
  }
  plot(error)
  (x[which.min(error)])
}
Comparison(x,xgb.eva.test)
# The threshold that minimize the overall accuracy is 0.15

xgb.test = predict(xgb.model, newdata = as.matrix(final.veriset[,1:29]), ntreelimit =
                     xgb.model$best_ntreelimit)
predict.CLASS = as.numeric(ifelse(xgb.test > 0.15,"1","0" ))
```

```r
table(predict.CLASS, final.veriset$Class)
table(predict.CLASS, final.veriset$Class)[1,2]*50 + table(predict.CLASS, final.veriset$Class)[2,1]*1
mean(predict.CLASS == final.veriset$Class)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

xgb.test = predict(xgb.model, newdata = as.matrix(final.veriset[,1:29]), ntreelimit =
                     xgb.model$best_ntreelimit)
final.veriset$predicted <- xgb.test
roc.test<- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#depth = 7
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
xgb.model.7 = xgboost(xgb.data.train, max.depth = 7, nround = 100, objective = "binary:logistic")
xgb.eva.test.7 = predict(xgb.model.7, newdata = as.matrix(pro.thre.set[,1:29]), ntreelimit =
                         xgb.model.7$best_ntreelimit, type = "response")
pro.thre.set$predicted <- xgb.eva.test.7
check(xgb.eva.test.7,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

xgb.test.7 = predict(xgb.model.7, newdata = as.matrix(final.veriset[,1:29]), ntreelimit =
                     xgb.model.7$best_ntreelimit)
final.veriset$predicted <- xgb.test.7
roc.test<- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#with histogram
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
xgb.model.hist = xgboost(xgb.data.train, max.depth = 7, nround = 100, objective = "binary:logistic")
xgb.eva.test.hist = predict(xgb.model.hist, newdata = as.matrix(pro.thre.set[,1:29]), ntreelimit =
                            xgb.model.hist$best_ntreelimit,tree_method = "hist", type = "response")
pro.thre.set$predicted <- xgb.eva.test.hist
check(xgb.eva.test.hist,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

xgb.test.hist = predict(xgb.model.hist, newdata = as.matrix(final.veriset[,1:29]), ntreelimit =
                       xgb.model.hist$best_ntreelimit)
final.veriset$predicted <- xgb.test.hist
roc.test<- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

######Adaboost
(pro.train.data$Class = as.factor(pro.train.data$Class))
(pro.thre.set$Class = as.factor(pro.thre.set$Class))
(final.veriset$Class = as.factor(final.veriset$Class))
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
#500
adaboost.project.500 = gbm(Class ~., data = pro.train.data,
                distribution = "adaboost",  n.trees = 500,
                shrinkage =  0.01,cv.folds = 3)
summary(adaboost.project.500) # Variable influence
adaboost.iter.500 = gbm.perf(adaboost.project.500,method = "cv")
adapro.predict.500 = predict(adaboost.project.500,pro_data[-pro.train,],
    n.trees = adaboost.iter.500,type='response')

adaboost.500.eva.pred = predict(adaboost.project.500,pro.thre.set,
    n.trees = adaboost.iter.500,type='response')
pro.thre.set$predicted = adaboost.500.eva.pred
check(adaboost.eva.pred,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

adaboost.500.test = predict(boost.project.500,final.veriset,n.trees = adaboost.iter.500,type='response')
final.veriset$predicted <- adaboost.500.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#1000
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
adaboost.project.1000 = gbm(Class ~., data = pro.train.data,
                   distribution = "adaboost",  n.trees = 1000,
                   shrinkage =  0.01,cv.folds = 3)
summary(adaboost.project.1000) # Variable influence
adaboost.iter.1000 = gbm.perf(adaboost.project.1000,method = "cv")
adapro.predict.1000 = predict(adaboost.project.1000,pro_data[-pro.train,],
    n.trees = adaboost.iter.1000,type='response')
```

```
adaboost.1000.eva.pred = predict(adaboost.project.1000,pro.thre.set,
    n.trees = adaboost.iter.1000,type='response')
pro.thre.set$predicted = adaboost.1000.eva.pred
check(adaboost.eva.pred,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

adaboost.1000.test = predict(boost.project.1000,final.veriset,n.trees = adaboost.iter.1000,type='response')
final.veriset$predicted <- adaboost.1000.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#1500
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
boost.project.1500 = gbm(Class ~., data = pro.train.data,
                 distribution = "bernoulli",  n.trees = 1500,
                 shrinkage =  0.01,cv.folds = 3)
summary(boost.project.1500) # Variable influence
boost.iter.1500 = gbm.perf(boost.project.1500,method = "cv")
pro.predict.1500 = predict(boost.project.1500,pro_data[-pro.train,],n.trees = boost.iter.1500,type='response')

boost.1500.eva.pred = predict(boost.project.1500,pro.thre.set,n.trees = boost.iter.1500,type='response')
pro.thre.set$predicted = boost.1500.eva.pred
check(boost.eva.pred,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

boost.1500.test = predict(boost.project.1500,final.veriset,n.trees = boost.iter.1500,type='response')
final.veriset$predicted <- boost.1500.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#2000
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
boost.project.2000 = gbm(Class ~., data = pro.train.data,
                 distribution = "bernoulli",  n.trees = 2000,
                 shrinkage =  0.01,cv.folds = 3)
summary(boost.project.2000) # Variable influence
boost.iter.2000 = gbm.perf(boost.project.2000,method = "cv")
pro.predict.2000 = predict(boost.project.2000,pro_data[-pro.train,],
```

```r
    n.trees = boost.iter.2000,type='response')

boost.2000.eva.pred = predict(boost.project.2000,pro.thre.set,
    n.trees = boost.iter.2000,type='response')
pro.thre.set$predicted = boost.2000.eva.pred
check(boost.eva.pred,pro.thre.set)

roc <- calculate_roc(pro.thre.set, 1, 50, n = 200)
mincost <- min(roc$cost)
roc %>%
  mutate(
    cost = ifelse(cost == mincost,
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "green", background = "lightblue", bold = T),
                  cell_spec(sprintf("%.5f", cost), "html",
                    color = "black", bold = F))
  ) %>%
  kable( "html", escape=F, align="c") %>%
  kable_styling(bootstrap_options = "striped", full_width = F, position = "center") %>%
  scroll_box(height = "600px")
Threshold = roc$threshold[which.min(roc$cost)]

boost.2000.test = predict(boost.project.2000,final.veriset,
    n.trees = boost.iter.2000,type='response')
final.veriset$predicted <- boost.2000.test
roc.test <- calculate_roc(final.veriset, 1, 50, n = 200)
plot_roc(roc.test, Threshold, 1, 50)

#10-fold Validation to find a better threshold:
setwd("~/Downloads")
pro_data = read.csv("creditcard.csv",header = T)
pro_data = pro_data[,-1]

set.seed(1)
pro.train = sample(1:nrow(pro_data),round(nrow(pro_data)*7/10))
pro.train.data = pro_data[pro.train,]
pro.test = pro_data[-pro.train,]
set.seed(1)
pro.thre = sample(1:nrow(pro.test),round(nrow(pro.test)*2/3))
pro.thre.set = pro.test[pro.thre,]
final.veriset = pro.test[-pro.thre,]
k = 10
set.seed(1)
folds = sample(1:k,nrow(pro_data[pro.train,]),replace = T)
Threshold = c()

for (i in 1:k){
  xgb.data.train = xgb.DMatrix(as.matrix(pro.train.data[folds != i,1:29]), l
    abel = pro.train.data[folds != i,]$Class)
  xgb.data.test = xgb.DMatrix(as.matrix(pro.train.data[folds==i,1:29]),
    label = pro.train.data[folds == i,]$Class)

  xgb.model = xgboost(xgb.data.train, max.depth = 3, nround = 100, objective = "binary:logistic")
  xgb.test = predict(xgb.model, newdata = as.matrix(pro.train.data[folds==i,1:29]), ntreelimit =
                     xgb.model$best_ntreelimit)
  Veri = pro.train.data[folds== i,]
  Threshold = c(Threshold,check(xgb.test,Veri))
}

xgb.test = predict(xgb.model, newdata = as.matrix(final.veriset[,1:29]), ntreelimit =
                   xgb.model$best_ntreelimit)
final.veriset$predicted <- xgb.test
roc <- calculate_roc(final.veriset, 1, 50, n = 200)

plot(Threshold,xlab="The suggested threshold given by each test set",main="Threshold Comparison")
cv.Threshold = mean(Threshold)

plot_roc(roc, cv.Threshold, 1, 50)
```