Random Forest and Boosting

Default Modeling

Contents

Random Forest and Boosting **2**

1. Upload and prepare data 2

1.1 Upload data 2

Create default_event 3

Create default flag 3

1.2 Select a subset of variables 3

1.3 Filter out NAs 3

1.4 Split train/test 3

Perform Random Forest analysis 3

2.1 Fit random forest 3

2.2 Variable importance analysis 4

3. Perform boosting analysis 6

3.1 Test sample analysis 8

3.2 Inclusion of shrinkage 8

Return the Optimal Number of Iterations 10

ML Calibration **12**

Create the data set and fit calibration function 12

Fit the calibration function 12

ML Model validation 13

Calibrated PD Validation 14


```r
library(dplyr)
library(caret)
library(randomForest)
library(gbm)
library(ROCR)
library(optiRum)
library(smbinning)
```

Random Forest and Boosting

1. Upload and prepare data

1.1 Upload data

```
oneypd_tree <- read.csv(file = 'Z:/Model Risk/Adam/IFRS9_CECL_MV/data/chap2oneypd.csv')
dplyr::glimpse(oneypd_tree)
```

```
## Observations: 25,906
## Variables: 45
## $ X                           <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1...
## $ id                          <int> 6670001, 9131199, 4963167, 39185...
## $ vintage_year                <int> 2005, 2006, 2004, 2005, 2006, 20...
## $ monthly_installment         <dbl> 746.70, 887.40, 1008.50, 458.23,...
## $ loan_balance                <dbl> 131304.44, 115486.51, 128381.73,...
## $ bureau_score                <int> 541, 441, 282, 461, 466, 470, 51...
## $ num_bankrupt_iva            <int> 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0...
## $ time_since_bankrupt         <int> 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0...
## $ num_ccj                     <int> 0, 0, 1, 0, 0, 0, 0, 0, NA, 0, 0...
## $ time_since_ccj              <int> 0, 0, 36, 0, 0, 0, 0, 0, NA, 0, ...
## $ ccj_amount                  <int> 0, 0, 459, 0, 0, 0, 0, 0, NA, 0,...
## $ num_bankrupt                <int> 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0...
## $ num_iva                     <int> 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0...
## $ min_months_since_bankrupt   <int> 0, 0, 0, 0, 0, 0, 0, 0, NA, 0, 0...
## $ pl_flag                     <int> 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0,...
## $ region                      <fct> r_a, r_b, r_c, r_d, r_e, r_c, r_...
## $ ltv                         <dbl> 0.7586, 0.6973, 0.6959, 0.1099, ...
## $ arrears_months              <dbl> 0.0000000, 0.0000000, 2.1882300,...
## $ origination_date            <fct> 9/14/2005, 1/20/2006, 12/21/2004...
## $ maturity_date               <fct> 9/30/2040, 1/31/2031, 12/31/2029...
## $ repayment_type              <fct> Non-IO, Non-IO, Non-IO, Non-IO, ...
## $ arrears_status              <int> 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2,...
## $ arrears_segment             <int> 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 1,...
## $ mob                         <int> 120, 116, 129, 123, 110, 120, 13...
## $ remaining_mat               <int> 300, 184, 171, 93, 310, 0, 166, ...
## $ loan_term                   <int> 35, 25, 25, 18, 35, 10, 25, 25, ...
## $ live_status                 <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
## $ repaid_status               <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ month                       <int> 9, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9,...
## $ arrears_event               <int> 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,...
## $ bankrupt_event              <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ term_expiry_event           <int> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,...
## $ worst_arrears_status        <int> 1, 1, 2, 1, 1, 1, 1, 1, 2, 1, 2,...
## $ max_arrears_12m             <dbl> 0.000000, 0.000000, 2.188230, 0....
## $ recent_arrears_date         <fct> NA, NA, 9/30/2015, NA, NA, NA, N...
## $ months_since_2mia           <int> NA, NA, 0, NA, NA, NA, NA, NA, N...
## $ avg_mia_6m                  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2,...
## $ max_arrears_bal_6m          <int> -42, 0, 1198, -114, 0, 0, -114, ...
## $ max_mia_6m                  <int> 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 2,...
## $ avg_bal_6m                  <int> 132080, 116972, 128500, 36610, 7...
## $ avg_bureau_score_6m         <int> 542, 494, 290, 460, 468, 484, 51...
```

```
## $ cc_util                       <dbl> 0.4578, 0.6299, 0.6331, 0.4990, ...
## $ annual_income                 <int> 76749, 78451, 31038, 56663, 7701...
## $ emp_length                    <int> 3, 10, 3, 8, 10, 3, 11, 5, 4, 1,...
## $ months_since_recent_cc_delinq <int> 11, 7, 6, 6, 3, 3, 13, 5, 3, 14,...
```

Create default_event

```
oneypd_tree <- mutate(oneypd_tree, default_event = if_else(
  oneypd_tree$arrears_event == 1 |
    oneypd_tree$bankrupt_event == 1 |
    oneypd_tree$term_expiry_event == 1,
  1,0))
```

Create default flag

From "default_event" derive "default_indicator" as "Yes" "No"

```
oneypd_tree <- mutate(oneypd_tree, default_indicator = if_else(
  oneypd_tree$default_event == 1, "Yes", "No"
))
oneypd_tree$default_indicator <- as.factor(oneypd_tree$default_indicator)
```

1.2 Select a subset of variables

```
oneypd_tree_sel_orig <-  dplyr::select(oneypd_tree,"default_indicator", "default_event", "bureau_score"
                "num_ccj", "time_since_ccj", "ccj_amount", "ltv", "mob", "max_arrears_12m",
                "max_arrears_bal_6m", "avg_bal_6m", "annual_income", "loan_balance", "loan_term",
                "cc_util", "emp_length", "months_since_recent_cc_delinq")
```

1.3 Filter out NAs

```
oneypd_tree_sel <- na.omit(oneypd_tree_sel_orig)
```

1.4 Split train/test

```
set.seed(123)
train_index <- createDataPartition(oneypd_tree_sel$default_event, p = 0.70, list=FALSE)
train <- oneypd_tree_sel[train_index, ]
test <- oneypd_tree_sel[-train_index, ]
```

Perform Random Forest analysis

2.1 Fit random forest

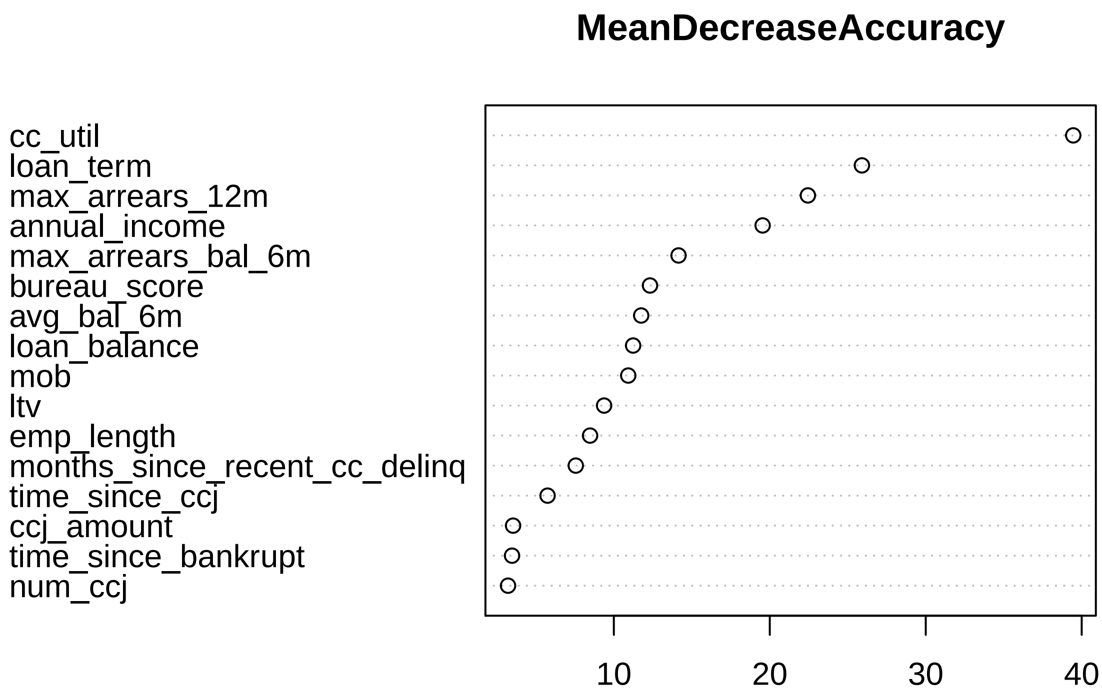
```
set.seed(123)
rf_oneypd <- randomForest(default_indicator ~ . - default_event, data = oneypd_tree_sel[train_index, ],
                          importance=TRUE, na.action=na.omit)
```

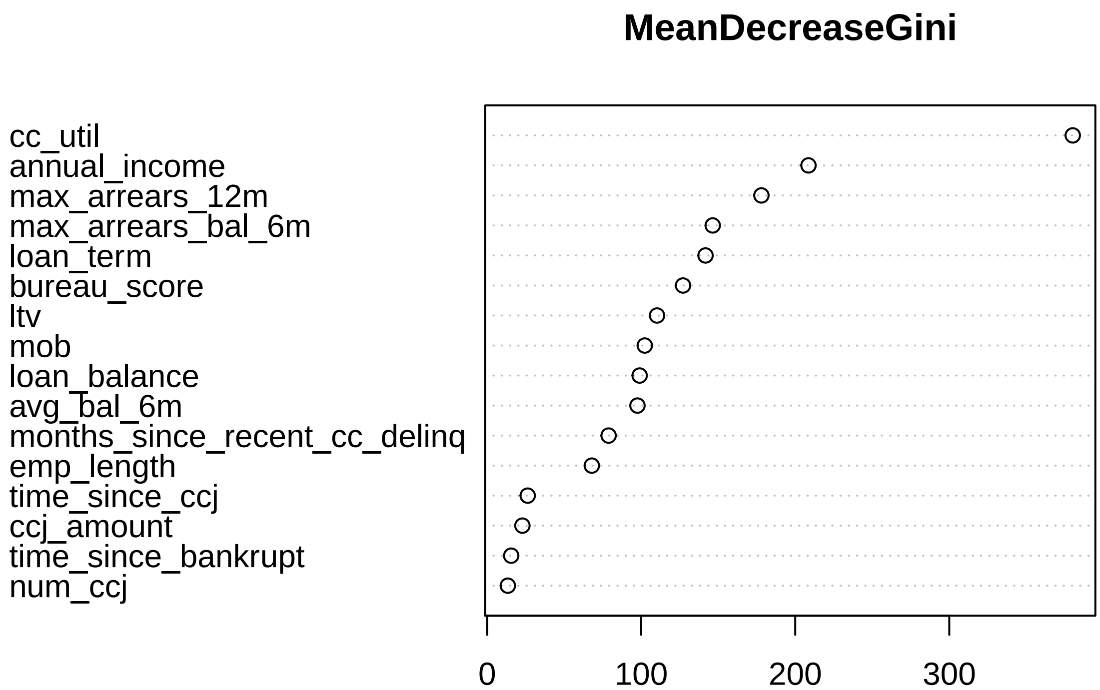
2.2 Variable importance analysis

```
imp <- importance(rf_oneypd)
print(imp)
```

```
##                                   No        Yes MeanDecreaseAccuracy
## bureau_score                7.439256 12.1392643            12.319947
## time_since_bankrupt         4.105219 -1.8075923             3.474521
## num_ccj                     3.127612 -0.8233618             3.216921
## time_since_ccj              5.848557 -1.1879736             5.751842
## ccj_amount                  3.729015 -1.6011149             3.539973
## ltv                         9.960364 -1.6058195             9.377171
## mob                         8.026681  7.5793098            10.924976
## max_arrears_12m            13.963881 16.0331846            22.442993
## max_arrears_bal_6m         11.014328 12.1261976            14.149976
## avg_bal_6m                 12.155950 -8.5200624            11.754771
## annual_income              10.577837 21.8618559            19.541862
## loan_balance               11.727443 -7.9372367            11.236998
## loan_term                  17.160199 32.8372513            25.907522
## cc_util                    12.908747 59.4728637            39.458262
## emp_length                  7.377317  1.7508620             8.476810
## months_since_recent_cc_delinq 4.821820  4.6924535            7.560639
##                            MeanDecreaseGini
## bureau_score                      127.14346
## time_since_bankrupt                15.60141
## num_ccj                            13.40092
## time_since_ccj                     26.31201
## ccj_amount                         22.87911
## ltv                               110.28143
## mob                               102.33820
## max_arrears_12m                   178.03480
## max_arrears_bal_6m                146.42412
## avg_bal_6m                         97.56750
## annual_income                     208.59031
## loan_balance                       98.99793
## loan_term                         141.74390
## cc_util                           380.16297
## emp_length                         67.93838
## months_since_recent_cc_delinq      78.84087
```

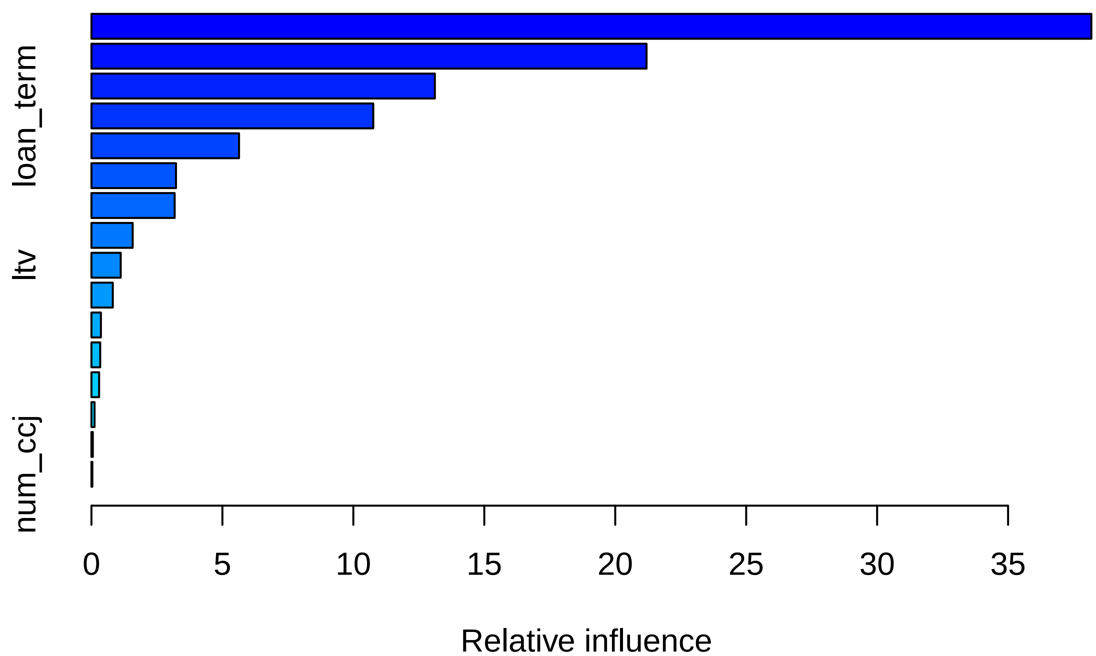
```
for (i in 3:4){
  ord <- order(imp[,i], decreasing=FALSE)
  dotchart(imp[ord, i],main=colnames(imp)[i], )
}
```


MeanDecreaseAccuracy
cc_util
loan_term
max_arrears_12m
annual_income
max_arrears_bal_6m
bureau_score
avg_bal_6m
loan_balance
mob
ltv
emp_length
months_since_recent_cc_delinq
time_since_ccj
ccj_amount
time_since_bankrupt
num_ccj
10
20
30
40



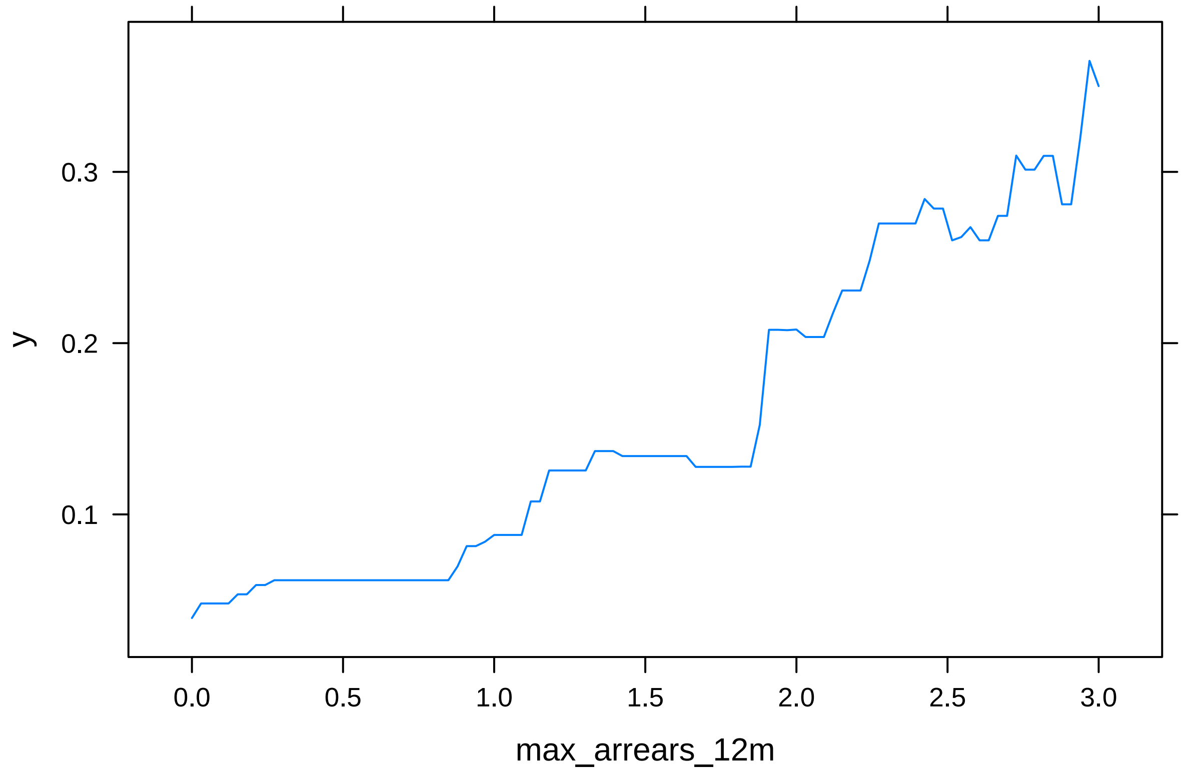
3. Perform boosting analysis

```
set.seed(1)
boost_oneypd = gbm(default_event ~ . - default_indicator , data = oneypd_tree_sel[train_index, ],
                   distribution='gaussian', n.trees=100, interaction.depth=4)
summary(boost_oneypd)
```




```
##                                                           var     rel.inf
## cc_util                                               cc_util 38.18022625
## max_arrears_12m                               max_arrears_12m 21.19633822
## annual_income                                   annual_income 13.11403783
## loan_term                                           loan_term 10.76196486
## bureau_score                                     bureau_score  5.63547561
## max_arrears_bal_6m                         max_arrears_bal_6m  3.22845020
## mob                                                       mob  3.17770493
## avg_bal_6m                                         avg_bal_6m  1.57542896
## ltv                                                       ltv  1.11793705
## months_since_recent_cc_delinq months_since_recent_cc_delinq  0.81599030
## loan_balance                                     loan_balance  0.36175061
## emp_length                                         emp_length  0.33470182
## ccj_amount                                         ccj_amount  0.29275840
## time_since_ccj                                 time_since_ccj  0.11924127
## time_since_bankrupt                       time_since_bankrupt  0.05333382
## num_ccj                                               num_ccj  0.03465986
```

```
plot(boost_oneypd, i.var='max_arrears_12m')
```



3.1 Test sample analysis

```
yhat_boost_oneyd = predict(boost_oneypd, newdata = oneypd_tree_sel[-train_index, ],
                           n.trees = 100
                           )
oneypd_test_boost = oneypd_tree_sel[-train_index,'default_event']
mean((yhat_boost_oneyd - oneypd_test_boost)^2)
```

```
## [1] 0.02794844
```

```
summary(yhat_boost_oneyd)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -0.054841 -0.002370  0.005617  0.052472  0.030983  1.265853
```

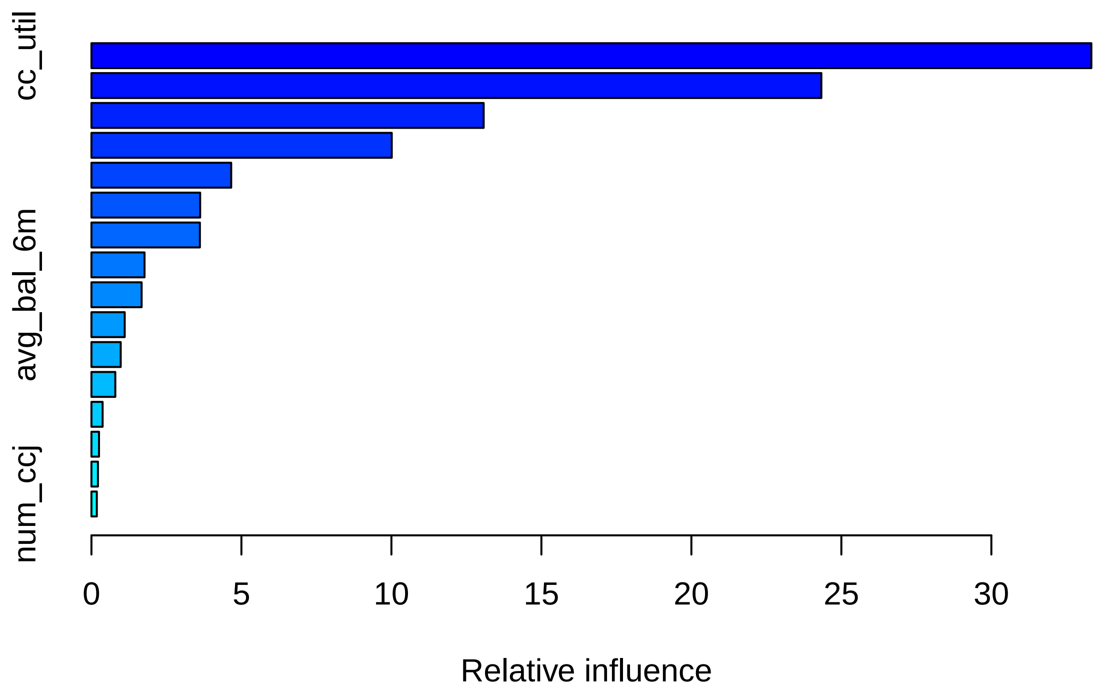
3.2 Inclusion of shrinkage

```
boost_oneypd_1 = gbm(default_event ~ . - default_indicator,
                     data=oneypd_tree_sel[train_index, ], distribution ='gaussian',
                     n.trees=100, interaction.depth=4, shrinkage=0.20, verbose=F, cv.folds=5)
yhat_oneypd_1 = predict(boost_oneypd_1,
```

```
                         newdata=oneypd_tree_sel[-train_index,],
                         n.trees=100)
mean((yhat_oneypd_1 - oneypd_test_boost)^2)
```

```
## [1] 0.02869406
```

```
summary(boost_oneypd_1)
```

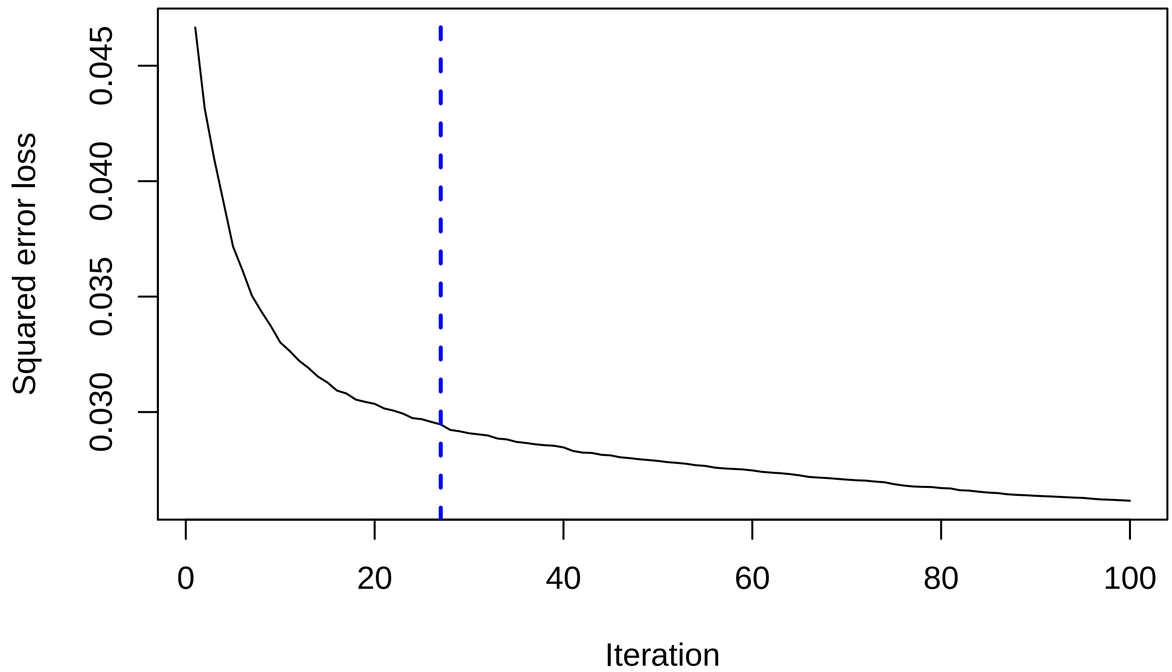



```
##                                                        var    rel.inf
## cc_util                                            cc_util 33.3340780
## max_arrears_12m                            max_arrears_12m 24.3333089
## annual_income                                annual_income 13.0763450
## loan_term                                        loan_term 10.0117294
## bureau_score                                  bureau_score  4.6593146
## mob                                                    mob  3.6244708
## max_arrears_bal_6m                      max_arrears_bal_6m  3.6152022
## ltv                                                    ltv  1.7703221
## avg_bal_6m                                      avg_bal_6m  1.6727006
## months_since_recent_cc_delinq months_since_recent_cc_delinq  1.1097567
## time_since_ccj                              time_since_ccj  0.9762302
## emp_length                                      emp_length  0.7957802
## ccj_amount                                      ccj_amount  0.3722192
## loan_balance                                  loan_balance  0.2522918
## time_since_bankrupt                    time_since_bankrupt  0.2169441
## num_ccj                                            num_ccj  0.1793060
```

Return the Optimal Number of Iterations

```
best.iter <- gbm.perf(boost_oneypd_1, method = "OOB", plot.it = TRUE,)
```

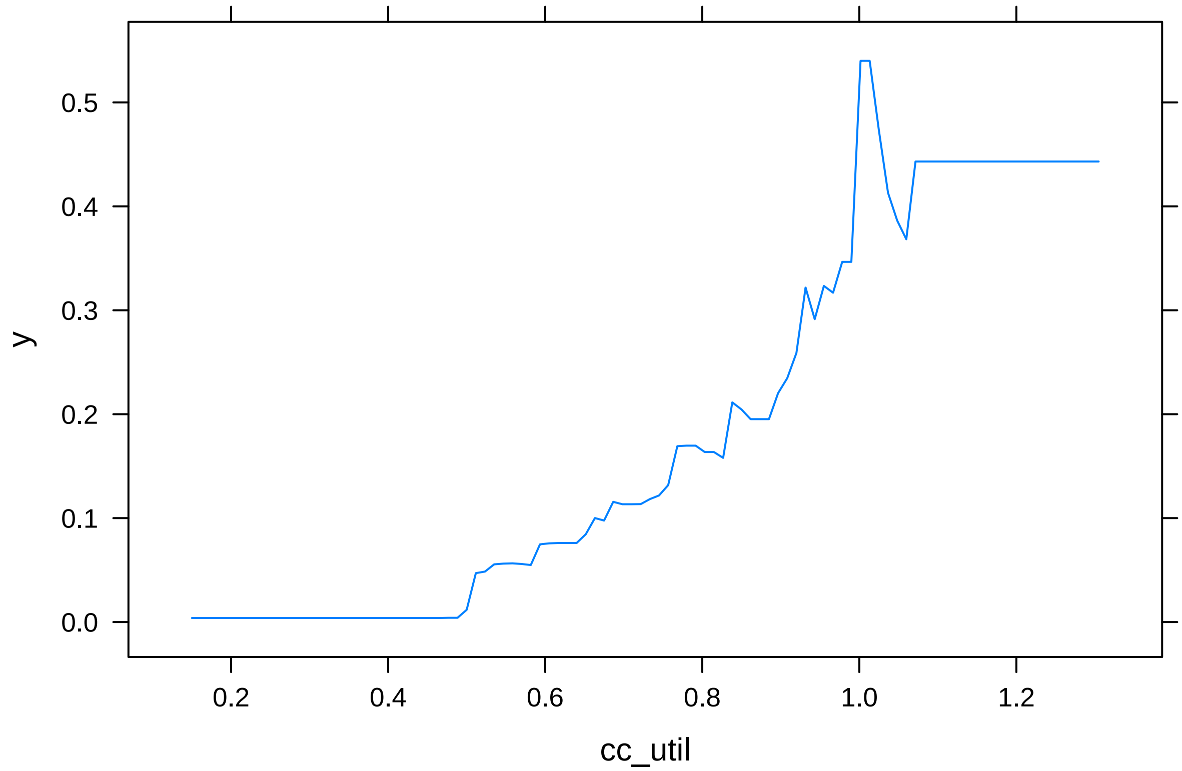
```
## OOB generally underestimates the optimal number of iterations although predictive performance is reas
```



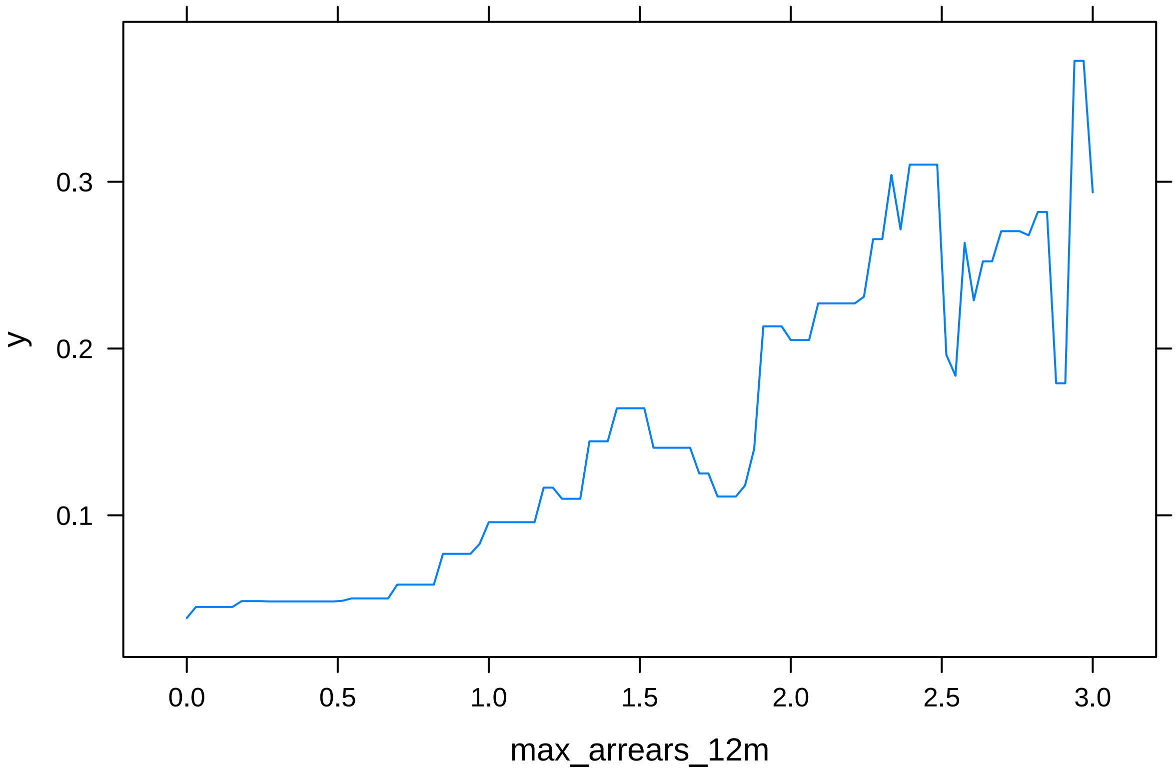

```
print(best.iter)
```

```
## [1] 27
## attr(,"smoother")
## Call:
## loess(formula = object$oobag.improve ~ x, enp.target = min(max(4,
##     length(x)/10), 50))
##
## Number of Observations: 100
## Equivalent Number of Parameters: 8.32
## Residual Standard Error: 0.0001924622
```

```
par(mfrow=c(1,2))
plot.gbm(boost_oneypd_1, i='cc_util')
```




```
plot.gbm(boost_oneypd_1, i='max_arrears_12m')
```



ML Calibration

Create the data set and fit calibration function

```
pred_orig <- as.matrix(predict(rf_oneypd, newdata = oneypd_tree_sel, type='prob'))
rf_pred <- as.matrix(pred_orig[,2])
rf_db_cal <- as.data.frame(cbind(oneypd_tree_sel$default_event, rf_pred))
colnames(rf_db_cal) <- c('def', 'pred')
```

Fit the calibration function

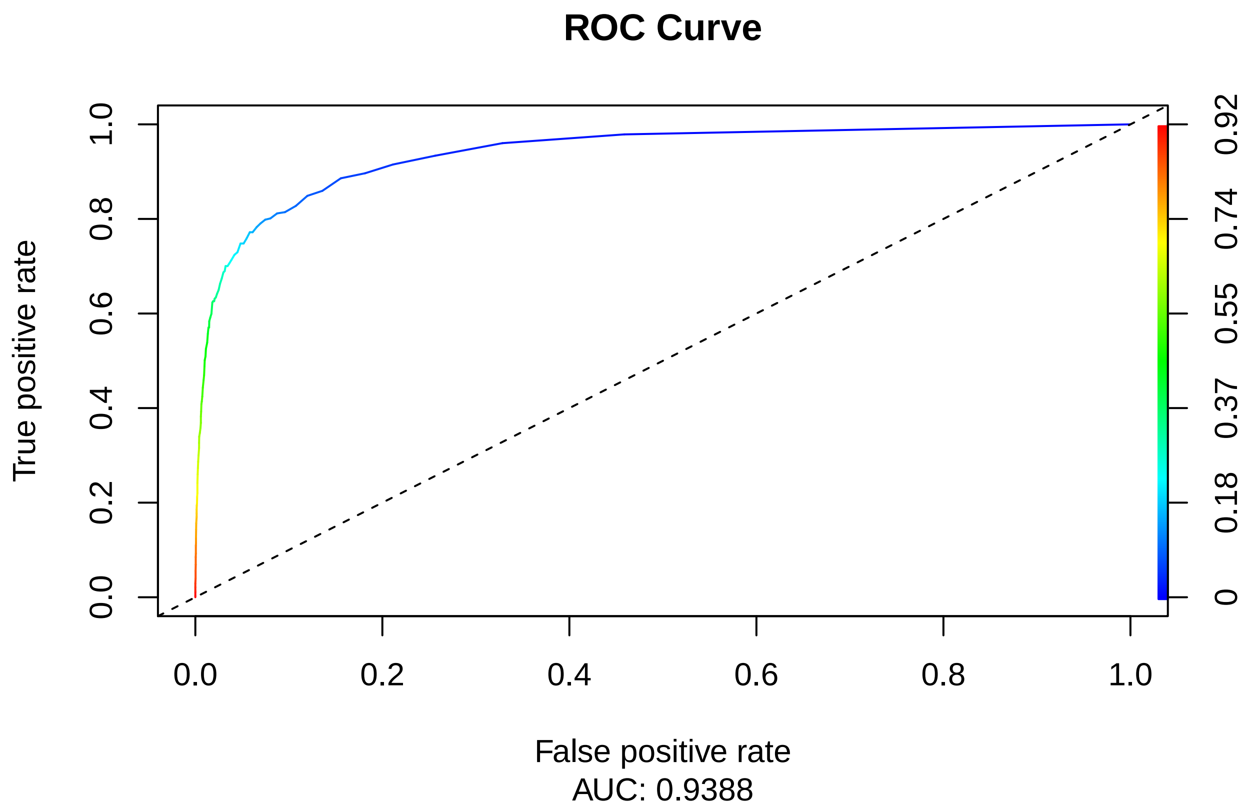
```
pd_model <- glm(def ~ pred, family=binomial(link='logit'), data=rf_db_cal)
summary(pd_model)
```

```
##
## Call:
## glm(formula = def ~ pred, family = binomial(link = "logit"),
##     data = rf_db_cal)
##
## Deviance Residuals:
```

```
##     Min       1Q   Median       3Q      Max
## -3.2682  -0.0845  -0.0793  -0.0793   3.3955
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -5.7616     0.1010  -57.04   <2e-16 ***
## pred         12.9038     0.2811   45.90   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 10477.1  on 25468  degrees of freedom
## Residual deviance:  1991.2  on 25467  degrees of freedom
## AIC: 1995.2
##
## Number of Fisher Scoring iterations: 8
```

ML Model validation

```
# ROC analysis

predict_test_orig <- as.matrix(
  predict(rf_oneypd, newdata = oneypd_tree_sel[-train_index, ], type='prob'))
predict_test <- as.matrix(predict_test_orig[,2])
oneypd_test <- oneypd_tree_sel[-train_index, 'default_indicator']
actual_test <- as.matrix(ifelse(oneypd_test=='Yes', 1, 0))
pred_test <- prediction(predict_test, actual_test)
perf_test <- performance(pred_test, 'tpr', 'fpr')
auc_test <- performance(pred_test, 'auc')

plot(perf_test, main='ROC Curve', sub=paste('AUC:', round(auc_test@y.values[[1]][1],5)), colorize=T)
abline(0,1, lty=8, col='black')
```




```
#KS (Kolmogrov-Smirnov) Analysis
ks_test <- max(attr(perf_test,'y.values')[[1]] - attr(perf_test,'x.values')[[1]])
print(paste('KS Test:', round(ks_test,5)))
```

```
## [1] "KS Test: 0.73036"
```

```
gini_test <- giniCoef(predict_test, actual_test)
print(paste('Gini Index:',round(gini_test,5)))
```

```
## [1] "Gini Index: 0.8776"
```

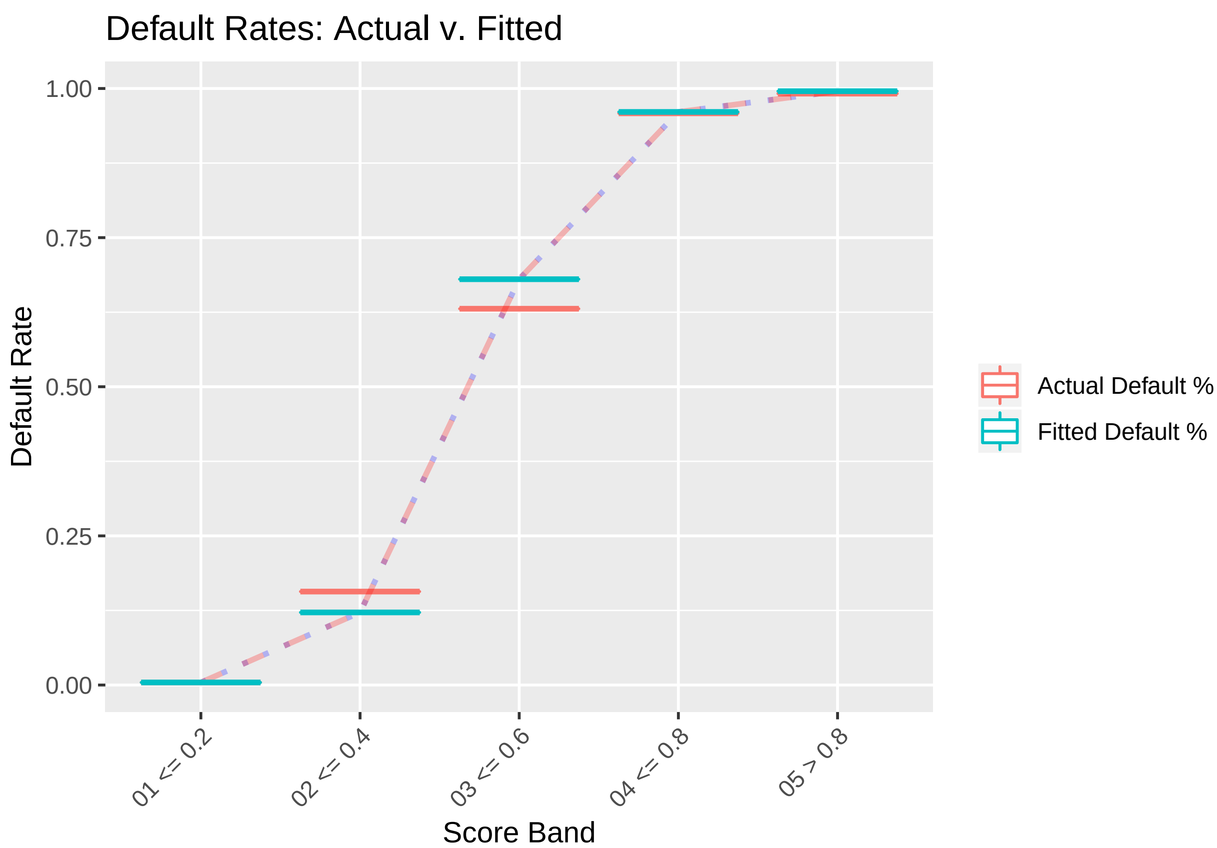
Calibrated PD Validation

```
# predict base on model params
rf_db_cal$pd <- predict(pd_model, new_data = rf_deb_cal, type='response')

#create bands
score_cust <- smbinning.custom(rf_db_cal, y='def', x='pred',
                               cuts=c(0.2, 0.4, 0.6, 0.8))

rf_db_cal <- smbinning.gen(rf_db_cal, score_cust, chrname='band')
```

```r
# Compare actual v. fitted values
# calc mean values
rf_db_cal_plot <- rf_db_cal %>%
  dplyr::group_by(band) %>%
  dplyr::summarise(mean_dr = round(mean(def), 4),
                   mean_pd = round(mean(pd), 4))

# Compute RMSe
rmse <- sqrt(mean((rf_db_cal_plot$mean_dr - rf_db_cal_plot$mean_pd)^2))
round(rmse,5)
```

```
## [1] 0.02719
```

```r
ggplot(data=rf_db_cal_plot, aes(x=band)) +
  geom_line(y=rf_db_cal_plot$mean_pd, group=1, color='blue', lwd=1,alpha=.25, linetype='dotted') +
  geom_boxplot(aes(y=rf_db_cal_plot$mean_dr, color='blue')) +


  geom_line(y=rf_db_cal_plot$mean_pd, group=1, color='red', lwd=1,alpha=.25, linetype='dashed') +
  geom_boxplot(aes(y=rf_db_cal_plot$mean_pd, color='red')) +


  ylab('Default Rate') +
  xlab('Score Band') +
  ggtitle("Default Rates: Actual v. Fitted") +
  scale_color_discrete(name = "", labels = c("Actual Default %", "Fitted Default %")) +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```


Default Rates: Actual v. Fitted
Default Rate
1.00
0.75
0.50
0.25
0.00
01 <= 0.2
02 <= 0.4
03 <= 0.6
04 <= 0.8
05 > 0.8
Score Band
Actual Default %
Fitted Default %

“‘