Machine Learning Assignment Phase 2

Chi Ting Low (s3611774) & Vidya Viswanathan (s3613547)

5/7/2018

Contents

Introduction

The aim of the current project is to replicate the research study by Yeh and Lien (2009). The purpose of Yeh and Lien's research was to compare the predictive accuracy of probability of default using three different machine learning methods: k-nearest neighbor classifiers (Altman, 1992), naive Bayes classifiers (Russell & Norvig, 2016) and Classification Trees (Kelleher, Mac Namee, & D'Arcy, 2015). The current project aims to examine all these methods using the *mlr* package (Bischl et. al, 2016) in R.

Methodology

The dataset is acquired from the UCI Machine Learning Repository. There are two phases in the current project. Phase I will focus on data cleaning, data exploration and data visualization. Phase II will focus on model building. Three classification methods are used in this project, as stated above. Each method is trained to make probability predictions regarding the probability of customer default on their credit given the information of the other variables. Each method has its own adjustment threshold to improve the classifier's performance, presented below. The dataset is split into a training set and test set, using a 70/30% split. The training set contains 2686 rows of observation and test set contains 1344 rows of observation, randomly selected by splitting.

To tune parameters, each classifier ran with 10-fold cross-validation stratified sampling. This is to ensure that the relative frequencies of the levels of a specific stratification feature are maintained in the sampled dataset (Kelleher, Mac Namee, & D'Arcy, 2015). Prediction on the test set is applied after parameter fine-tuning. To access the performance of the dataset, mean misclassification error rate (mmcce), area under curve (AUC), receiver operating characteristic curve (ROC), false positive rate (fpr) and false negative rate (fnr) are used to measure the performance for the training set. Furthermore, a confusion matrix is also applied to evaluate the performance on both the training and test set.

Data Preparation

```
options("java.home"="/Library/Java/JavaVirtualMachines/jdk1.8.0_73.jdk/Contents/Home/jre/lib/server/lib
Sys.setenv(LD_LIBRARY_PATH='/Library/Java/JavaVirtualMachines/jdk1.8.0_73.jdk/Contents/Home/jre/lib/ser
dyn.load('/Library/Java/JavaVirtualMachines/jdk1.8.0_73.jdk/Contents/Home/jre/lib/server/libjvm.dylib')
library(rJava) #for feature selection
library(mlr) #machine learning packages
library(rpart.plot) #plot for classification tree
data <- read.csv('/Users/anambiar/vidya_projects/phase2/recode.csv') ##read the data
summarizeColumns(data) #summary of the data
```

```
##              name    type na         mean         disp   median
## 1               X integer  0   2015.50000 1.163505e+03   2015.5
## 2   Limit_balance numeric  0 171657.56824 1.259438e+05 150000.0
## 3             Sex  factor  0           NA 4.081886e-01       NA
## 4       Education  factor  0           NA 5.749380e-01       NA
## 5        Marriage  factor  0           NA 4.848635e-01       NA
## 6             Age integer  0     36.52208 9.180649e+00     35.0
## 7      Sept_repay  factor  0           NA 4.119107e-01       NA
## 8       Aug_repay  factor  0           NA 3.937965e-01       NA
## 9      July_repay  factor  0           NA 3.972705e-01       NA
## 10      Jun_repay  factor  0           NA 3.754342e-01       NA
## 11      May_repay  factor  0           NA 3.632754e-01       NA
## 12      Apr_repay  factor  0           NA 3.727047e-01       NA
```

```
## 13 Sept_statement integer  0  22090.75732 4.428846e+04   4405.0
## 14  Aug_statement numeric  0  22256.28983 4.432866e+04   4414.0
## 15 July_statement integer  0  22321.20968 4.447871e+04   4238.5
## 16  Jun_statement integer  0  22618.34268 4.494709e+04   4180.0
## 17  May_statement integer  0  22590.34640 4.452332e+04   4087.0
## 18  Apr_statement numeric  0  22701.86898 4.553050e+04   4162.0
## 19    Sept_amtpay numeric  0   4654.05806 1.088290e+04   1600.0
## 20     Aug_amtpay integer  0   4609.70000 1.197973e+04   1595.0
## 21    July_amtpay integer  0   4719.06576 1.341075e+04   1443.0
## 22     Jun_amtpay integer  0   4547.37519 1.109397e+04   1443.5
## 23     May_amtpay numeric  0   4605.91985 1.353810e+04   1228.0
## 24     Apr_amtpay integer  0   4590.06799 1.495288e+04   1048.0
## 25        default  factor  0           NA 3.563275e-01       NA
##             mad     min    max nlevs
## 1    1493.7195       1   4030     0
## 2  133434.0000   10000 740000     0
## 3           NA    1645   2385     2
## 4           NA      10   1713     4
## 5           NA      34   2076     3
## 6      10.3782      21     72     0
## 7           NA       3   2370     9
## 8           NA       1   2443     9
## 9           NA       1   2429     9
## 10          NA       1   2517     9
## 11          NA       1   2566     8
## 12          NA       2   2528     8
## 13   6047.5254   -4316 581775     0
## 14   6075.6948  -24704 572677     0
## 15   5815.4985  -61506 471175     0
## 16   5765.8314   -3903 486776     0
## 17   5614.6062   -3876 503914     0
## 18   5874.0612 -339603 527711     0
## 19   2372.1600       0 187206     0
## 20   2364.7470       0 302961     0
## 21   2139.3918       0 417588     0
## 22   2140.1331       0 193712     0
## 23   1820.6328       0 303512     0
## 24   1553.7648       0 345293     0
## 25          NA    1436   2594     2
```

Making Learners

```
set.seed(12345) #for reproducible result
#making classification task
classif.task <- makeClassifTask(id = 'X',data = data, target = 'default')

#making classification learners
naivebayes.lrn = makeLearner('classif.naiveBayes', predict.type = 'prob')
knn.lrn = makeLearner('classif.kknn', predict.type = 'prob')
tree.lrn = makeLearner('classif.rpart', predict.type = 'prob')
c50.lrn = makeLearner('classif.C50', predict.type = 'prob')
```

```
#split dataset to train and test set based on 70% and 30% split
n = nrow(data)
train.set = sample(n, size = 0.7*n)
test.set = setdiff(1:n, train.set)
```

Naive Bayes

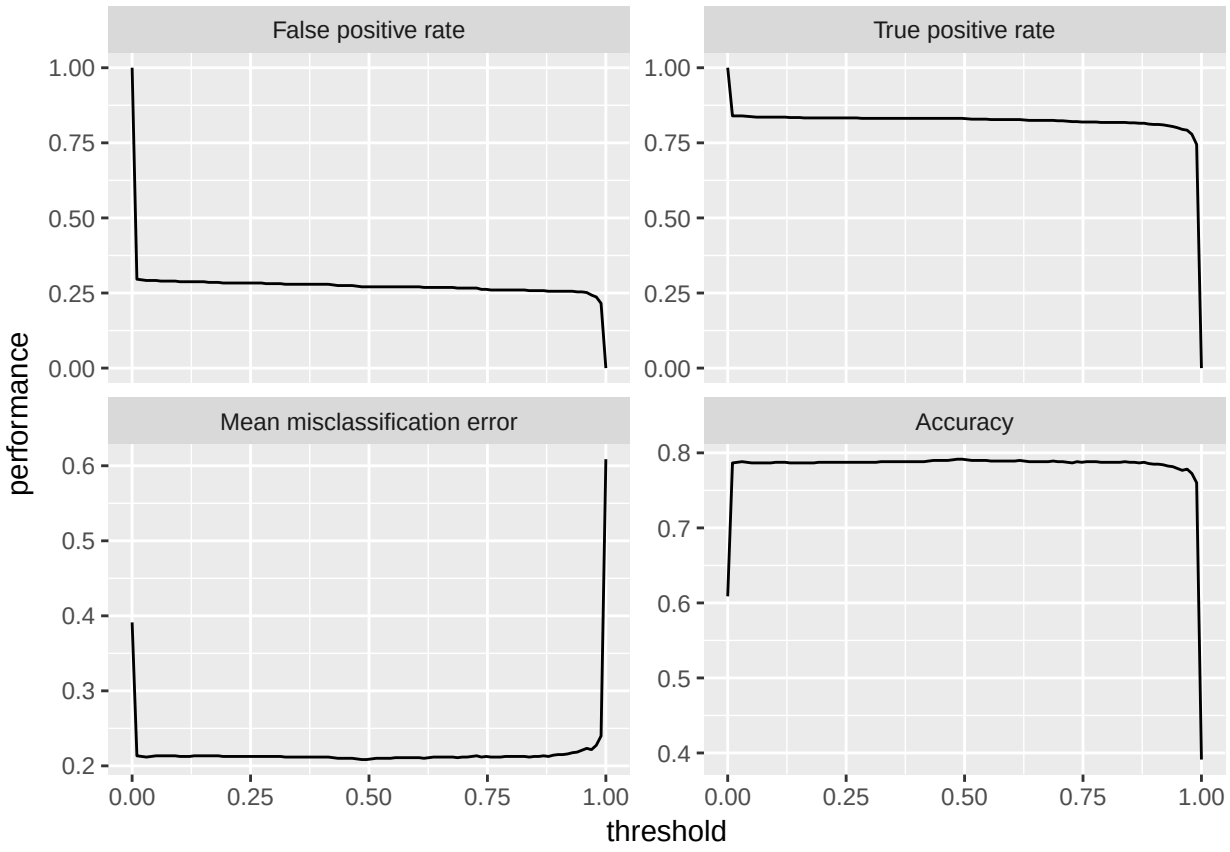
To study the probability of customer default risk based on the given variables, a naive Bayes classification is applied. Naive Bayes models compute the prior probabilities of the target features taking each level in its domain and the conditional probability of each feature taking each level that target can take (Kelleher, Mac Namee, & D'Arcy, 2015). An initial analysis without parameter fine-tuning shows that 740 of the observations are classified as not likely to default their credit, compared to 469 observations that are. It also showed an fpr of 27.06%, fnr of 16.85%, mmce of 20.84% and acc of 79.16%. Based on the mlr tutorial (Bisehl et. al, 2016), the lower the fpr, fnr and mmce rate, the better the performance of the model. Conversely, the higher the accuracy rate, the better the model. Furthermore, the Receiver Operating Characteristic (ROC) curve analysis (Metz, 1978; Zweig & Campbell, 1993) shows a tendency closer to the upper left corner, which indicated a better accuracy rate (Zweig & Campbell, 1933)

```
model = train(naivebayes.lrn, classif.task, subset = train.set)
pred = predict(model, task = classif.task, subset = test.set)
```

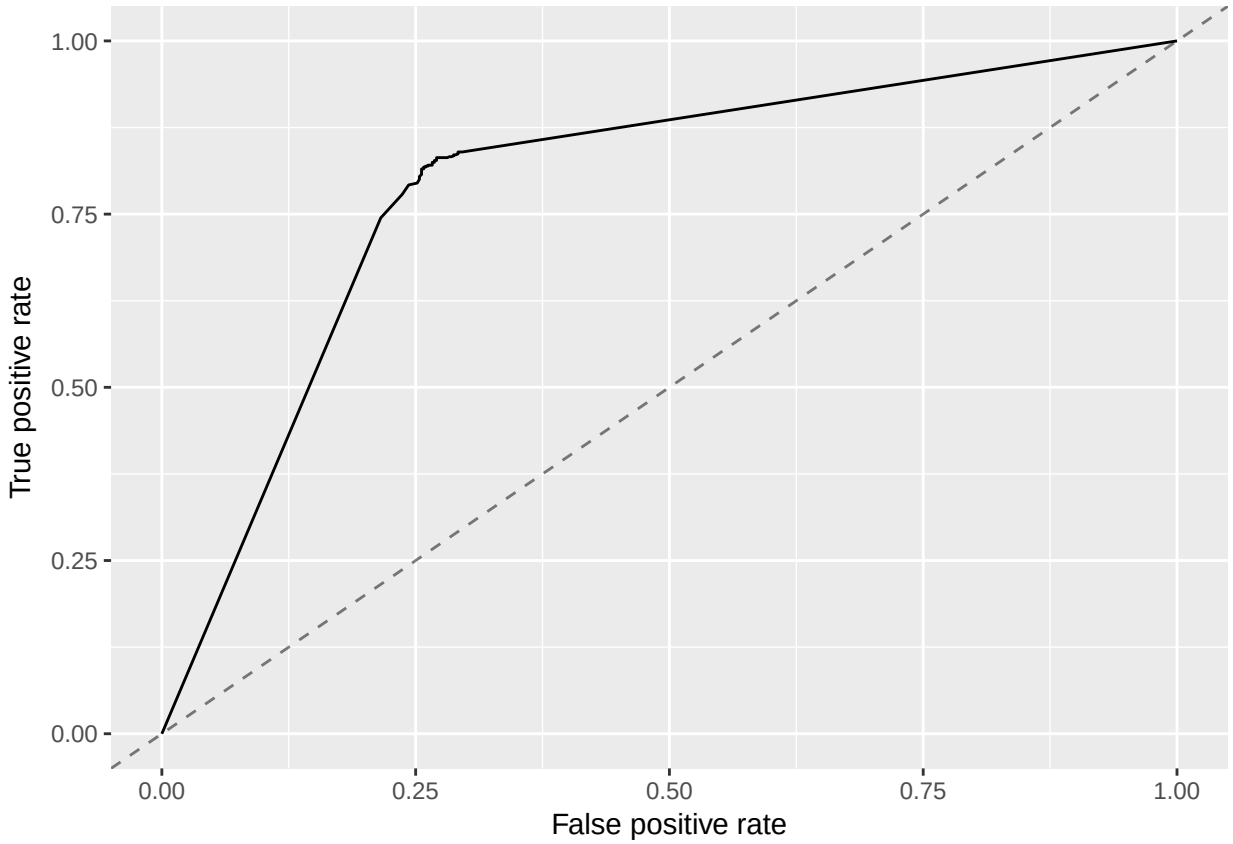
```
performance(pred, measures = list(fpr, fnr, mmce, acc))
```

```
##       fpr       fnr      mmce       acc
## 0.2706131 0.1684783 0.2084367 0.7915633
```

```
df = generateThreshVsPerfData(pred, measures = list(fpr, tpr, mmce, acc))
plotThreshVsPerf(df)
```




```
plotROCCurves(df)
```




```
calculateConfusionMatrix(pred, relative = TRUE, sum = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##          predicted
## true      No         Yes        -err.-     -n-
##   No      0.83/0.83 0.17/0.26 0.17       740
##   Yes     0.27/0.17 0.73/0.74 0.27       469
##   -err.-       0.17       0.26 0.21       <NA>
##   -n-     736        473        <NA>       2821
##
##
## Absolute confusion matrix:
##          No Yes -err.-  -n-
## No      612 124    124  736
## Yes     128 345    128  473
## -err.- 128 124    252   NA
## -n-    740 469     NA 2821
```

Naive Bayes Hyperparameter tune-fining

Naive Bayes Cross-validation

To improve the performance of the naive Bayes model, hyperparmeter fine-tuning is applied using Laplace smoothing. By setting the parameter from 0 to 25 and using 10-folded stratified resampling method it is possible to verify and improve the performance of the model. The tuned analysis shows that using Laplace value of 10 will produce the lowest mmce rate (0.219). By fusing the produced parameters, the new prediction

performance has improved. The new performance results show that the fpr, fnr, and mmce (27.91%, 16.84%, 21.17%) have decreased, respectively. Furthermore, there is a small improvement in the accuracy, from 75.77% to 78.82%. Based on the confusion matrix, it reclassified the default risk from the test set. The new confusion matrix shows that there are 744 observations classified as "will not default" compared to 465 observations which will default their credit.

```
#setting parameters
ps <- makeParamSet(
  makeDiscreteParam('laplace',  values = c(0, 0.1, 0.5, 2, 3, 5, 10, 20, 25))
)
```

```
#Stratified resampling using 10 fold
ctrl <- makeTuneControlGrid()
rdesc <- makeResampleDesc('CV', iters = 10L, stratify = T)
```

```
#Tune process
res <- tuneParams('classif.naiveBayes', task = classif.task, resampling = rdesc, par.set = ps, control =
res
```

```
## Tune result:
## Op. pars: laplace=10
## mmce.test.mean=0.2198461
```

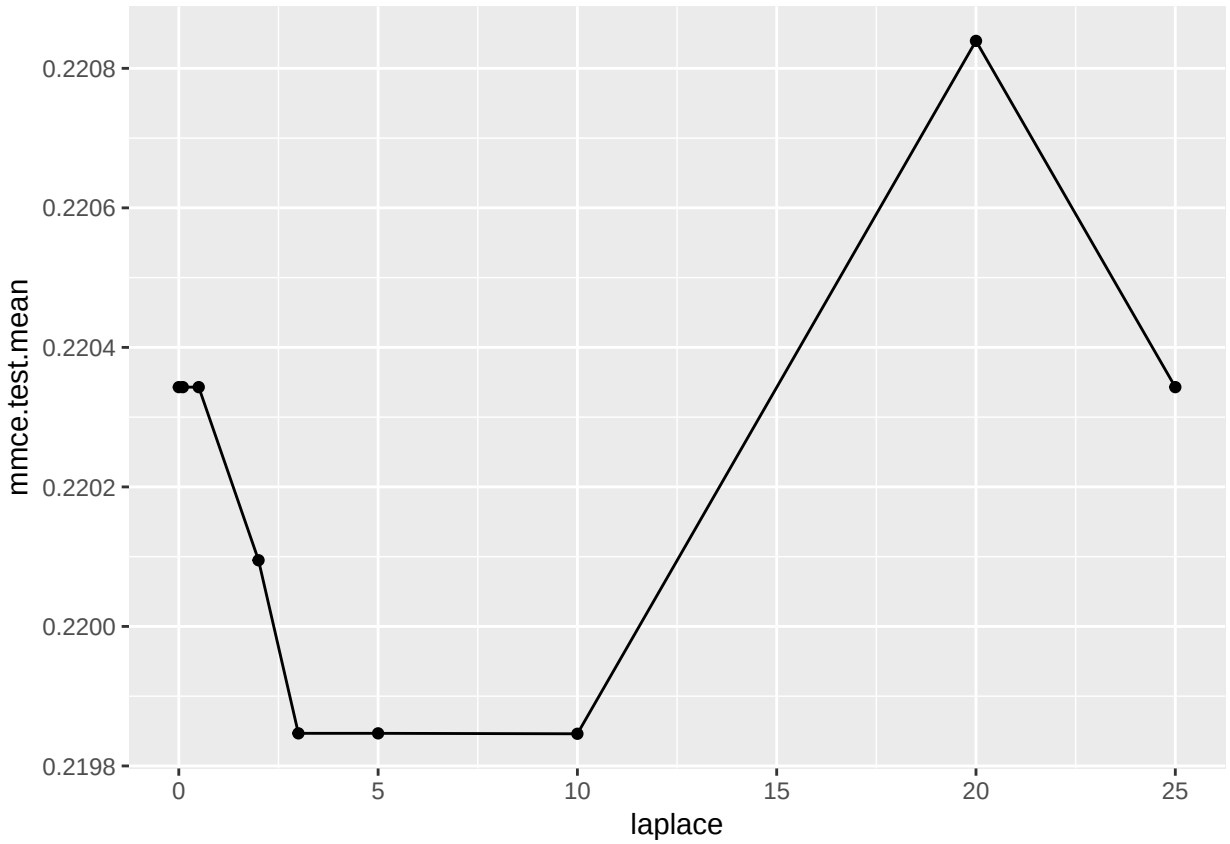
```
res$x
```

```
## $laplace
## [1] 10
```

```
res$y
```

```
## mmce.test.mean
##      0.2198461
```

```
tunedata <- generateHyperParsEffectData(res)
plotHyperParsEffect(tunedata, x = 'laplace', y = 'mmce.test.mean', plot.type = 'line')
```

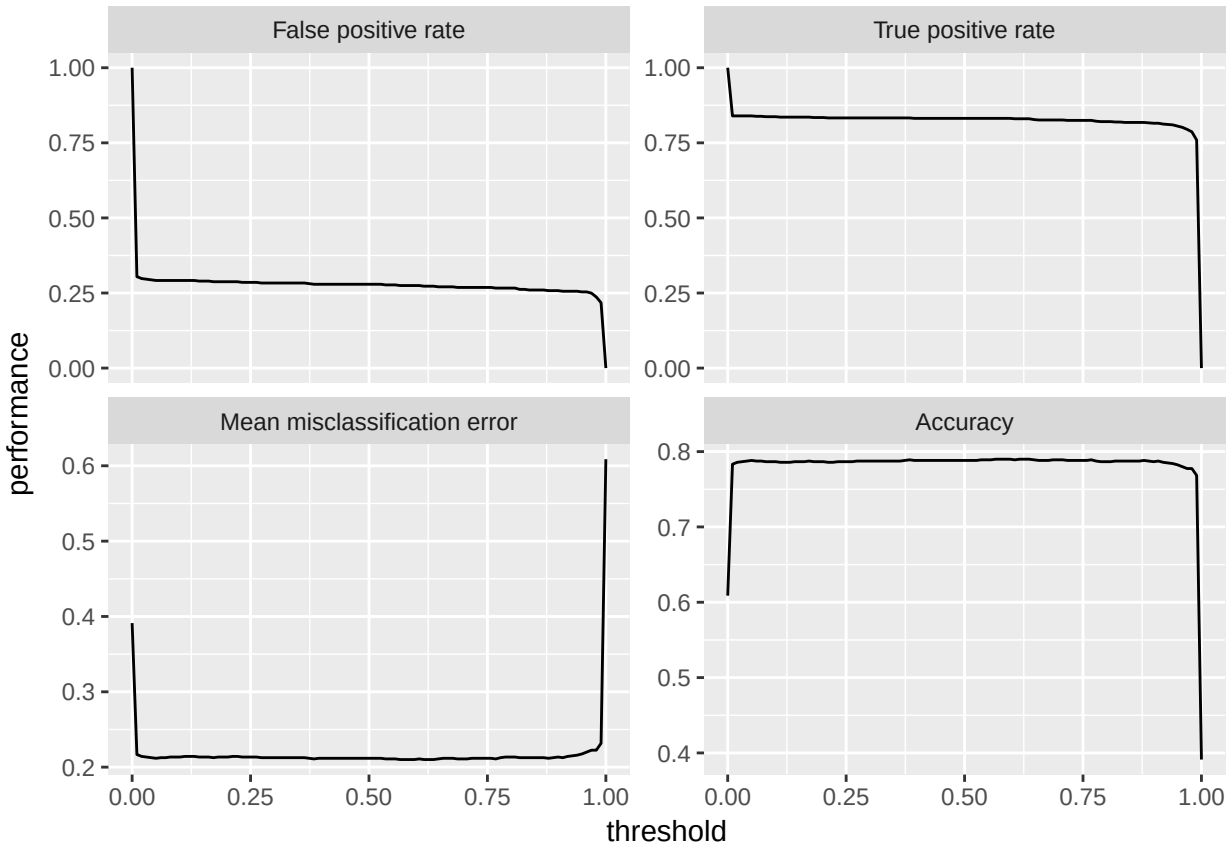


Tuned learners

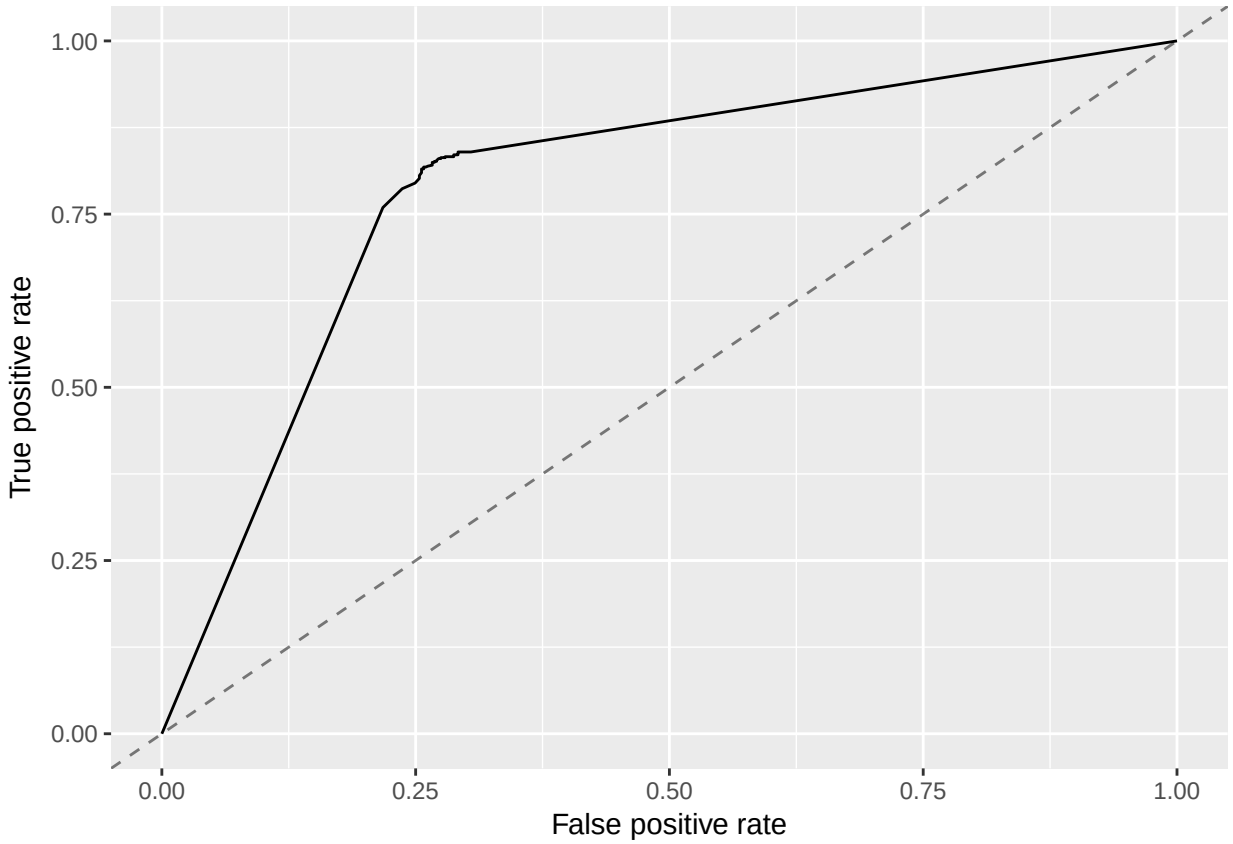
```
tunedlearners <- setHyperPars(makeLearner('classif.naiveBayes'), par.vals = res$x)

tunedlearners1 <- makeTuneWrapper(naivebayes.lrn, rdesc, mmce, ps, ctrl, show.info = F)
tunedmod <- train(tunedlearners1, classif.task, subset = train.set)

tunedpred <- predict(tunedmod, task = classif.task, subset = test.set)

performance(tunedpred, measures = list(fpr, fnr, mmce, acc))
```

```
##       fpr       fnr      mmce       acc
## 0.2790698 0.1684783 0.2117452 0.7882548
```

```
df = generateThreshVsPerfData(tunedpred, measures = list(fpr, tpr, mmce, acc))
plotThreshVsPerf(df)
```




```
plotROCCurves(df)
```




```
calculateConfusionMatrix(tunedpred, relative = TRUE, sums = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##          predicted
## true      No         Yes        -err.-     -n-
##   No      0.83/0.82 0.17/0.27 0.17       744
##   Yes     0.28/0.18 0.72/0.73 0.28       465
##   -err.-       0.18      0.27 0.21       <NA>
##   -n-     736        473        <NA>      2821
##
##
## Absolute confusion matrix:
##          No Yes -err.-  -n-
## No      612 124    124  736
## Yes     132 341    132  473
## -err.- 132 124    256   NA
## -n-    744 465     NA 2821
```

Decision Tree

To understand how variables lead to the conclusion of an individual's credit default risk, a decision tree classification is applied. The analysis shows that based on the given variables, the decision tree classification predicted that 789 observations will not default and 420 observation will default out of the test set. The performance in the prediction shows that the fpr, fnr and mmce are 33.62%, 14.4% and 21.92%, respectively. The average class accuracy is 78.08%. This indicates that, given the variables, the accuracy of making correct predictions on each class is 78.08%. The ROC is supported by showing that the curve has a tendency closer

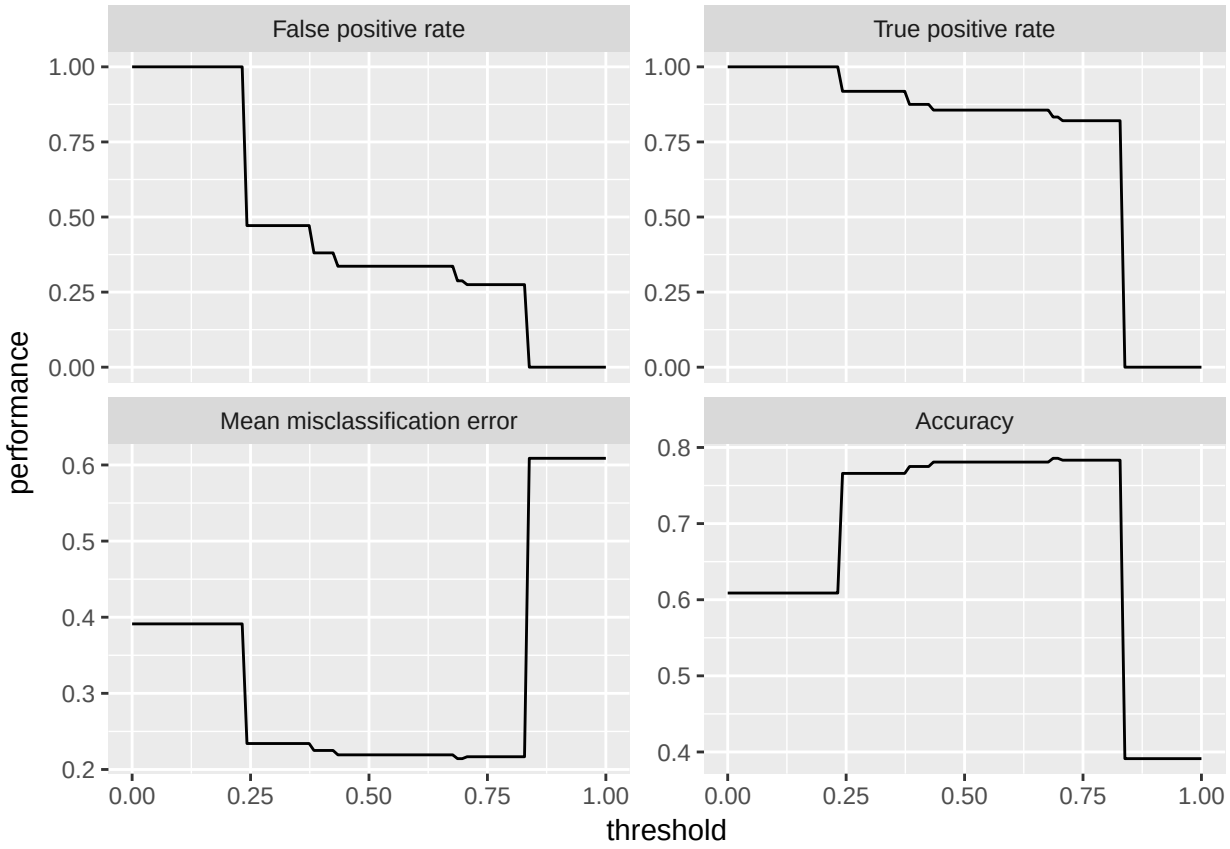
to the upper left corner. The figure shows that the tree is split based on the variables; May, August and September repayment, amount paid in May and gender.

```
#decision trees model
treemod <- train(tree.lrn, classif.task, subset = train.set)
treepred = predict(treemod, task = classif.task, subset = test.set)
```

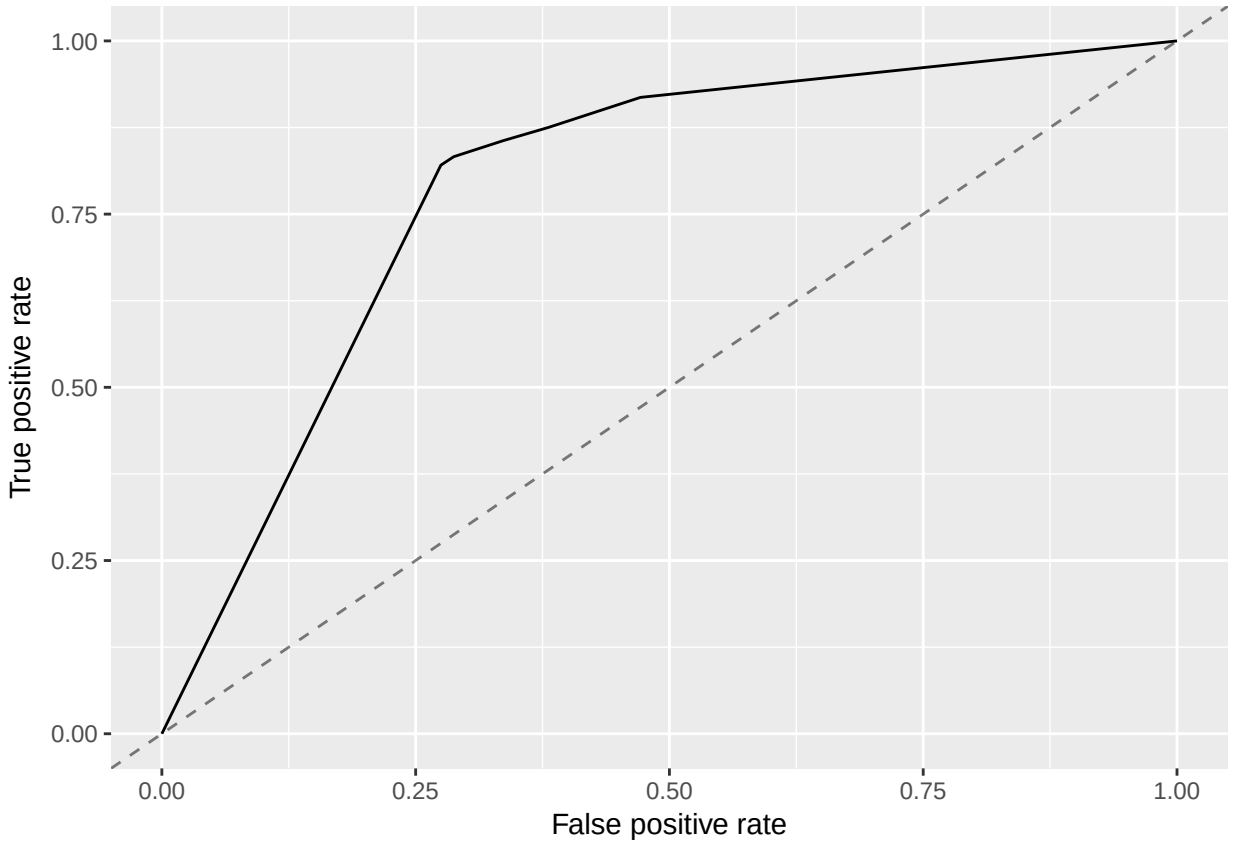
```
#prediction performance
performance(treepred, measures = list(fpr, fnr, mmce, acc))
```

```
##       fpr       fnr      mmce       acc
## 0.3361522 0.1440217 0.2191894 0.7808106
```

```
#plot for performance
treedf = generateThreshVsPerfData(treepred, measures = list(fpr, tpr, mmce, acc))
plotThreshVsPerf(treedf)
```




```
#plot of ROC curves
plotROCCurves(treedf)
```




```
#confusion matrix
calculateConfusionMatrix(treepred, relative = TRUE, sums = TRUE)
```

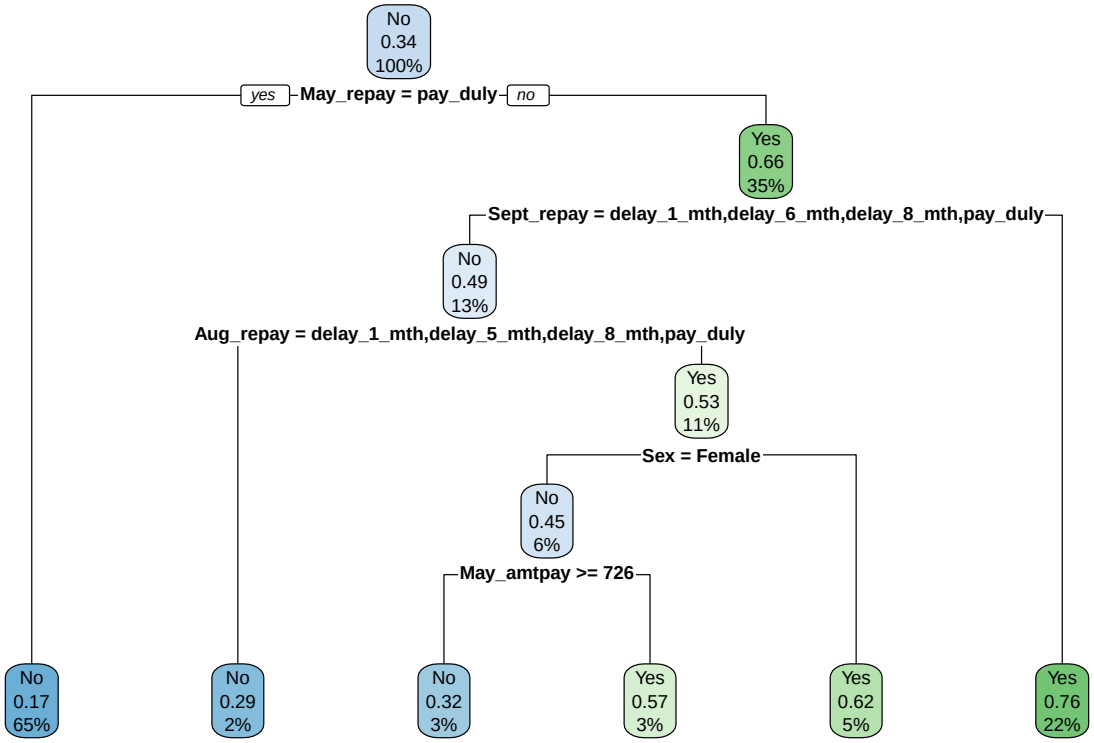
```
## Relative confusion matrix (normalized by row/column):
##          predicted
## true      No         Yes        -err.-     -n-
##   No      0.86/0.80 0.14/0.25 0.14       789
##   Yes     0.34/0.20 0.66/0.75 0.34       420
##   -err.-       0.20       0.25 0.22       <NA>
##   -n-     736        473        <NA>       2821
##
##
## Absolute confusion matrix:
##          No Yes -err.-  -n-
## No      630 106    106  736
## Yes     159 314    159  473
## -err.- 159 106    265   NA
## -n-    789 420     NA 2821
```

```
#summary of tree models
tree <- getLearnerModel(treemod)
tree
```

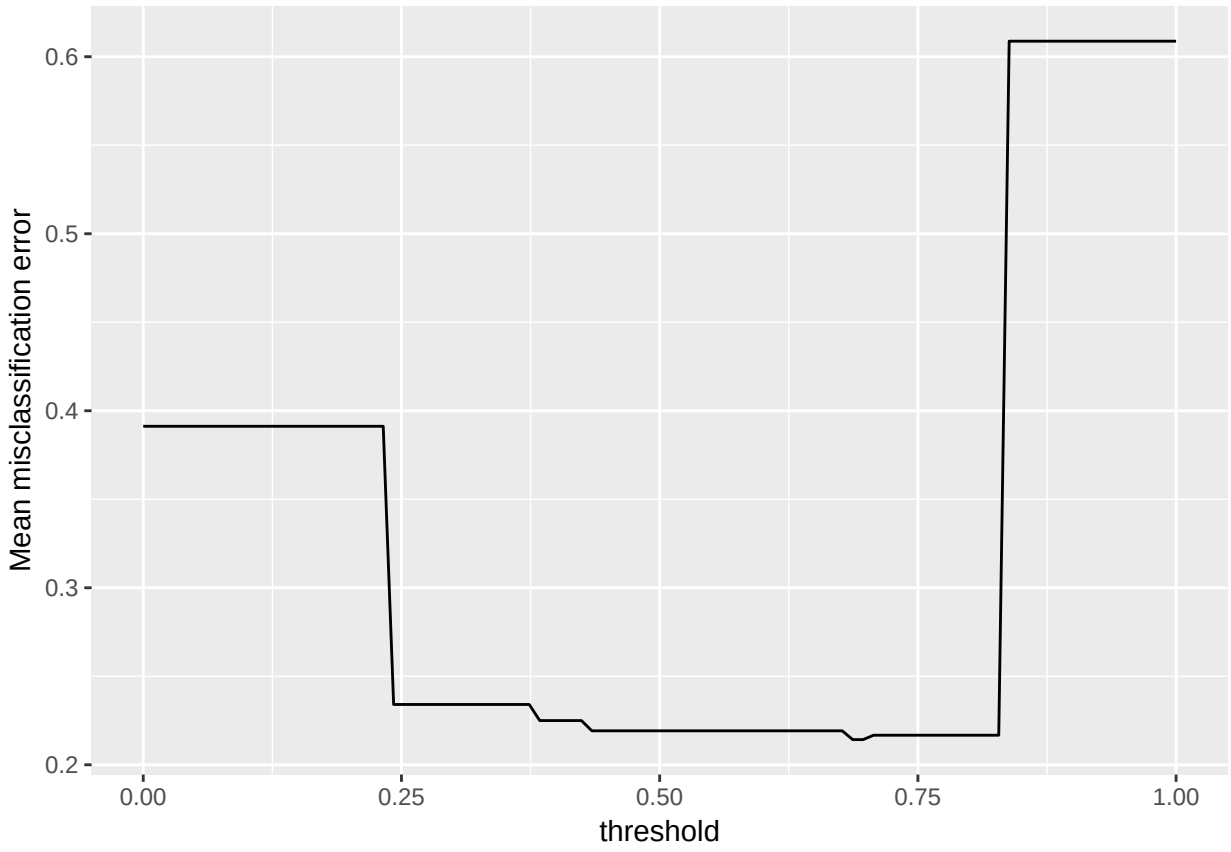
```
## n= 2821
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
```

```
##  1) root 2821 963 No (0.6586317 0.3413683)
##    2) May_repay=pay_duly 1832 311 No (0.8302402 0.1697598) *
##    3) May_repay=delay_2_mth,delay_3_mth,delay_4_mth,delay_5_mth,delay_6_mth,delay_7_mth,delay_8_mth 9
##      6) Sept_repay=delay_1_mth,delay_6_mth,delay_8_mth,pay_duly 371 183 No (0.5067385 0.4932615)
##       12) Aug_repay=delay_1_mth,delay_5_mth,delay_8_mth,pay_duly 58  17 No (0.7068966 0.2931034) *
##       13) Aug_repay=delay_2_mth,delay_3_mth,delay_4_mth,delay_7_mth 313 147 Yes (0.4696486 0.5303514)
##         26) Sex=Female 163  73 No (0.5521472 0.4478528)
##           52) May_amtpay>=725.5 79  25 No (0.6835443 0.3164557) *
##           53) May_amtpay< 725.5 84  36 Yes (0.4285714 0.5714286) *
##         27) Sex=Male 150  57 Yes (0.3800000 0.6200000) *
##      7) Sept_repay=delay_2_mth,delay_3_mth,delay_4_mth,delay_5_mth,delay_7_mth 618 149 Yes (0.241100
```

```
#visualising decision  trees
rpart.plot(tree)
```




```
#plot of mmce
d <- generateThreshVsPerfData(treepred, measures = mmce)
plotThreshVsPerf(d)
```



Decision tree Tuning

To improve decision tree's performance, hyperparameter tuning is appling on decision max depth and and complexity. These are used to control the size of the decision tree and to select optimal tree size (Kelleher, Mac Namee, & D'Arcy, 2015). The value of max depth is set between 1 and 10 and the upper and lower value for cp is set to 0.1 and 0.01.

The result of the parameter fine-tuning shows that the desired max depth and complexity parameter is nine and 0.02 with an mmce test mean of 22.6%. A new prediction on the test set after fusing the parameter with learner shows that there is a small improvement on the model. The fpr, fnr and mmce is reduce to 27.48%, 17.93% and 21.67%, respectively. The average class accuracy increased from 78.08% to 78.33%, which is small improvement. The confusion matrix shows that 734 observations will not default and 475 observations will default. In addition, the decision tree plot shows that only one variable, May repayment, is an important variable to split the tree. However, the result should be interpreted carefully as a majority of the variables are excluded, which may be a sign of overfitting.

```
set.seed(12345)
#set parameter
ps <- makeParamSet(
  makeDiscreteParam('maxdepth', values = c(1,2,3,4,5,6,7,8,9,10)),
  makeNumericLearnerParam('cp', lower = 0.01, upper = 0.1)
)

#set control grid
ctrl <- makeTuneControlGrid()
#10-fold cross validation with stratify sampling
rdesc <- makeResampleDesc('CV', iters = 10L, stratify = TRUE)
```

```
#fuse the tuning
res <- tuneParams('classif.rpart', task = classif.task, resampling = rdesc, par.set = ps, control = ctrl

#result of tune
res
```

```
## Tune result:
## Op. pars: maxdepth=9; cp=0.02
## mmce.test.mean=0.2258104
```

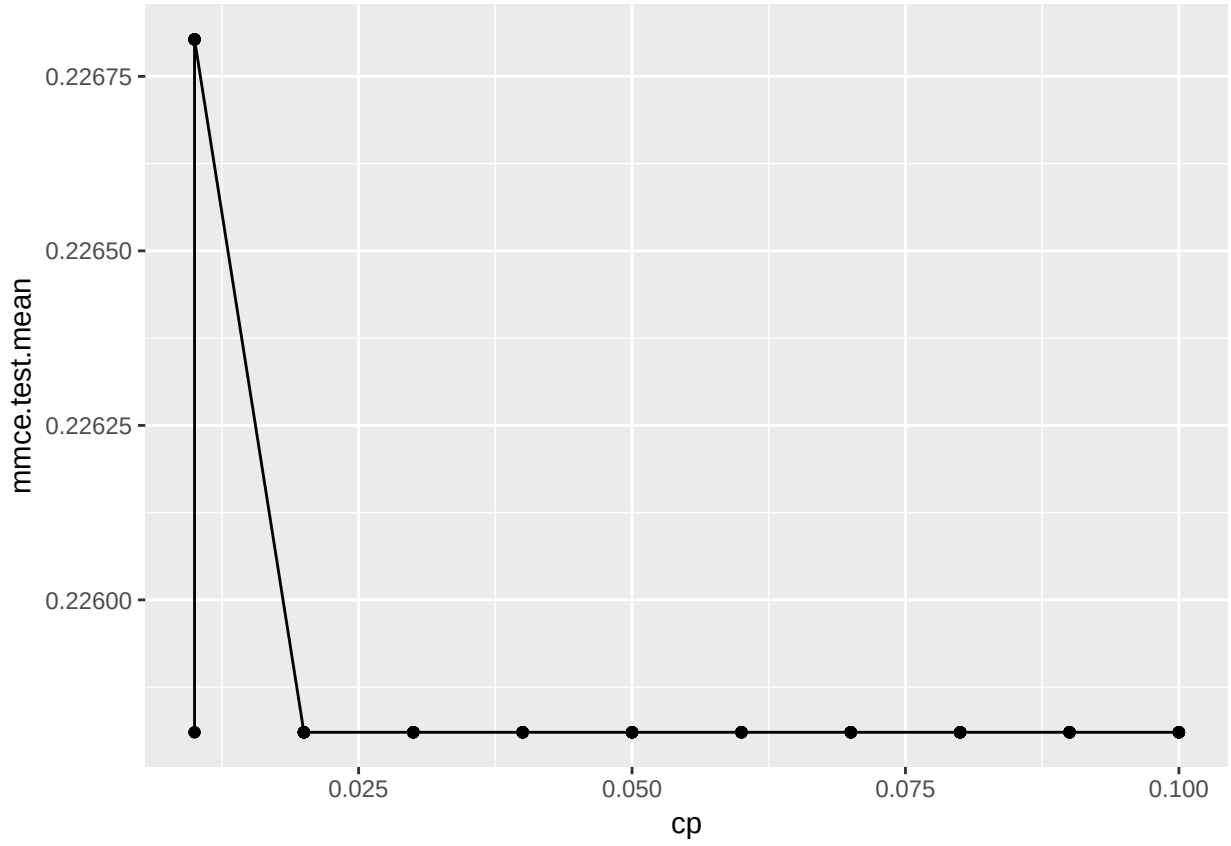
```
res$x
```

```
## $maxdepth
## [1] 9
##
## $cp
## [1] 0.02
```

```
res$y
```

```
## mmce.test.mean
##      0.2258104
```

```
tunedata1 <- generateHyperParsEffectData(res)
plotHyperParsEffect(tunedata1, x = 'cp', y = 'mmce.test.mean', plot.type = 'line')
```




```
tunedlearners.tree <- setHyperPars(makeLearner('classif.rpart'), par.vals = res$x)
tunedlearners.tree1 <- makeTuneWrapper(tree.lrn, rdesc, mmce, ps, ctrl, show.info = F)
tunedmod.tree <- train(tunedlearners.tree, classif.task, subset = train.set)
```

```
tunedpred.tree <- predict(tunedmod.tree, task = classif.task, subset = test.set)

performance(tunedpred.tree, measures = list(fpr, fnr, mmce, acc))
```

```
##       fpr       fnr      mmce       acc
## 0.2748414 0.1793478 0.2167080 0.7832920
```

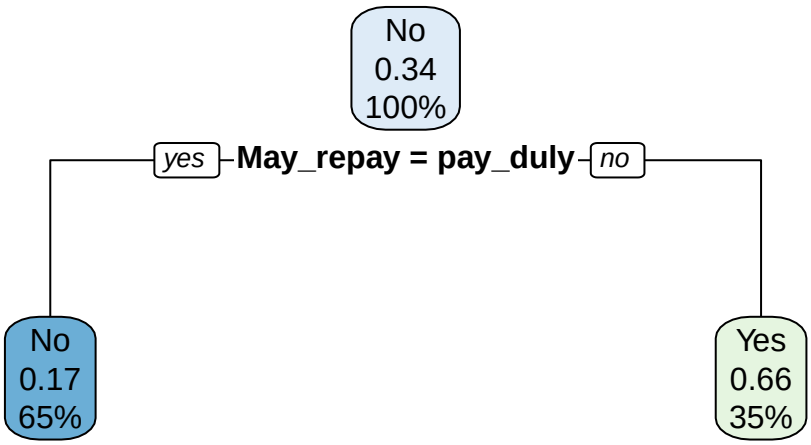
```
calculateConfusionMatrix(tunedpred.tree, relative = TRUE, sums = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##          predicted
## true     No        Yes       -err.-    -n-
##   No     0.82/0.82 0.18/0.28 0.18      734
##   Yes    0.27/0.18 0.73/0.72 0.27      475
##   -err.-      0.18      0.28 0.22      <NA>
##   -n-    736       473       <NA>      2821
##
##
## Absolute confusion matrix:
##          No Yes -err.-  -n-
## No     604 132    132  736
## Yes    130 343    130  473
## -err.- 130 132    262   NA
## -n-    734 475     NA 2821
```

```
tree <- getLearnerModel(tunedmod.tree)
tree
```

```
## n= 2821
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 2821 963 No (0.6586317 0.3413683)
##   2) May_repay=pay_duly 1832 311 No (0.8302402 0.1697598) *
##   3) May_repay=delay_2_mth,delay_3_mth,delay_4_mth,delay_5_mth,delay_6_mth,delay_7_mth,delay_8_mth 98
```

```
rpart.plot(tree)
```



K Nearest Neighbour Classifier (KNN)

The k-nearest neighbor classifier (k-NN) is another classification method that is used to study the similarity of the features and predict its class outcome. The initial training shows that the performance of the k-NN has a false positive rate and false negative rate of 41.43% and 17.66%. In addition, the mmce of the prediction is 26.96% with an average class accuracy of 73.04%. Compared to the classification method above, the k-NN has the lowest average class accuracy. The confusion matrix shows that the k-NN was able to correctly classify 606 observations to the default category. This is the lowest of the compared classification methods.

```
knn.model = train(knn.lrn, classif.task, subset = train.set)
knn.pred = predict(knn.model, task = classif.task, subset = test.set)

performance(knn.pred, measures = list(fpr, fnr, mmce, acc))
```

```
##       fpr       fnr      mmce       acc
## 0.4143763 0.1766304 0.2696443 0.7303557
```

```
calculateConfusionMatrix(knn.pred,relative = TRUE, sums = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##          predicted
## true      No         Yes        -err.-     -n-
##   No      0.82/0.76  0.18/0.32  0.18       802
##   Yes     0.41/0.24  0.59/0.68  0.41       407
##   -err.-       0.24       0.32  0.27       <NA>
##   -n-     736        473        <NA>       2821
##
##
## Absolute confusion matrix:
##          No Yes -err.-  -n-
## No      606 130    130  736
## Yes     196 277    196  473
## -err.- 196 130    326   NA
## -n-     802 407     NA 2821
```

KNN Tuning

To increase the performance of the k-NN, hyperparameter tuning is conducted by using cross-validation and sample stratifying methods on the dataset. To gain the optimal number of k, the algorithms are trained to study the value of k from 1 to 30. The optimal number of k is 20 with the mmce of 22.53%, as presented in the plot below. Later, the tuned parameter is fused with learner to re-train the model. The tuned model shows that the false positive rate and false negative rate have reduced to 36.36% and 14.54%. Furthermore, mmce decreased to 23.08% and average class accuracy increased to 76.23%.

```
set.seed(12345)
ps = makeParamSet(
  makeIntegerLearnerParam(id = 'k', lower = 1L, upper = 20L, default = 1L)
)

ctrl <- makeTuneControlGrid()

rdesc <- makeResampleDesc('CV', iters = 10, stratify = T)

res <- tuneParams('classif.kknn', task = classif.task, resampling = rdesc, par.set =  ps, control = ctrl
```

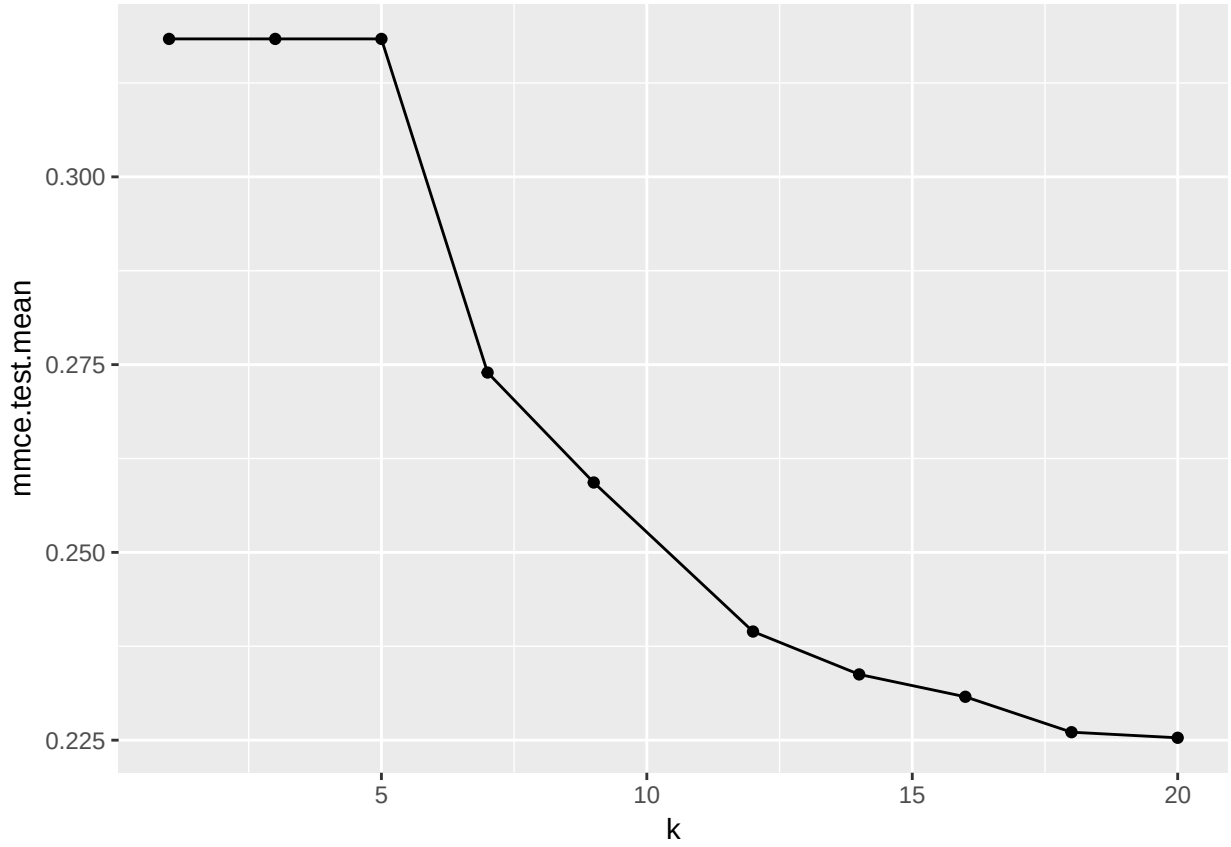
```
res$x
```

```
## $k
## [1] 20
```

```
res$y
```

```
## mmce.test.mean
##      0.2253159
```

```
tunedataknn <- generateHyperParsEffectData(res)
plotHyperParsEffect(tunedataknn, x = 'k', y = 'mmce.test.mean', plot.type = 'line')
```




```
tunedlearners.knn <- setHyperPars(makeLearner('classif.kknn'), par.vals = res$x)
tunedlearners.knn1 <- makeTuneWrapper(knn.lrn, rdesc, mmce, ps, ctrl, show.info = F)
tunedmod.knn <- train(tunedlearners.knn1, classif.task, subset = train.set)
tunedpred.knn <- predict(tunedmod.knn, task = classif.task, subset = test.set)

performance(tunedpred.knn, measures = list(fpr, fnr, mmce, acc))
```

```
##       fpr       fnr      mmce       acc
## 0.3636364 0.1453804 0.2307692 0.7692308
```

```
calculateConfusionMatrix(tunedpred.knn, relative = TRUE, sums = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     No        Yes       -err.-    -n-
##   No     0.85/0.79 0.15/0.26 0.15      801
```

```
##   Yes    0.36/0.21 0.64/0.74 0.36      408
##   -err.-      0.21      0.26 0.23      <NA>
##   -n-    736       473       <NA>      2821
##
##
## Absolute confusion matrix:
##         No Yes -err.-  -n-
## No     629 107    107  736
## Yes    172 301    172  473
## -err.- 172 107    279   NA
## -n-    801 408     NA 2821
```

Feature Selection

Although the performance of the classification methods above have high accuracy, it is important for us to understand which features are identified as the most important factors for future decision making. Therefore, the feature selection method is applied using chi.squared. The analysis shows that the most important feature is individual past repayment status. Using this result, a better classification method can be implemented.

```
#fv2 = generateFilterValuesData(classif.task, method = "chi.squared")
#plotFilterValues(fv2)
```

C5.0 Decision Tree

The C5.0 decision tree is an extension of the ID3 algorithm (Witten, Frank, Hall, & Pal, 2016). The performance of the C5.0 decision tree shows that it has a false positive rate and false negative rate of 34.88% and 14.13%. The mmce is around 22.25% and average class accuracy is 77.75%. In addition to the accuracy of the data, it shows that it is able to capture the important features based on the feature selection when compared to previous decision tree methods. As presented below in the summary of the decision tree, the main split of the tree is based on the latest payment, which is the September repayment, then August, May and April repayments.

```
c50.mod <- train(c50.lrn,classif.task, subset = train.set)
c50.pred <- predict(c50.mod, classif.task, subset = test.set)

performance(c50.pred, measures = list(fpr, fnr, mmce, acc))
```

```
##       fpr       fnr      mmce       acc
## 0.3488372 0.1413043 0.2224979 0.7775021
```

```
tree <- getLearnerModel(c50.mod)
summary(tree)
```

```
##
## Call:
## C5.0.default(x = d$data, y = d$target, weights = .weights, control = ctrl)
##
##
## C5.0 [Release 2.07 GPL Edition]      Sat Jun  9 11:41:22 2018
## -------------------------------
##
## Class specified by attribute `outcome'
```

```
##
## Read 2821 cases (25 attributes) from undefined.data
##
## Decision tree:
##
## Sept_repay in {delay_2_mth,delay_3_mth,delay_4_mth,delay_5_mth,delay_7_mth}:
## :...Aug_repay in {delay_1_mth,delay_2_mth,delay_3_mth,delay_4_mth,delay_5_mth,
## :   :             delay_6_mth,delay_7_mth,delay_8_mth}: Yes (635/155)
## :   Aug_repay = pay_duly: No (42/18)
## Sept_repay in {delay_1_mth,delay_6_mth,delay_8_mth,pay_duly}:
## :...May_repay = delay_8_mth: No (0)
##     May_repay in {delay_6_mth,pay_duly}:
##     :...Apr_repay in {delay_4_mth,delay_6_mth,delay_7_mth,
##     :   :             delay_8_mth}: No (0)
##     :   Apr_repay = delay_3_mth: Yes (5/1)
##     :   Apr_repay in {delay_2_mth,delay_5_mth,pay_duly}:
##     :   :...Aug_repay in {delay_1_mth,delay_2_mth,delay_4_mth,delay_5_mth,
##     :       :             delay_6_mth,delay_7_mth,delay_8_mth,
##     :       :             pay_duly}: No (1757/270)
##     :       Aug_repay = delay_3_mth: Yes (12/4)
##     May_repay in {delay_2_mth,delay_3_mth,delay_4_mth,delay_5_mth,delay_7_mth}:
##     :...Sex = Female:
##         :...Aug_repay in {delay_1_mth,delay_4_mth,delay_5_mth,delay_6_mth,
##         :   :             delay_8_mth,pay_duly}: No (38/9)
##         :   Aug_repay = delay_7_mth: Yes (7/3)
##         :   Aug_repay = delay_2_mth:
##         :   :...Limit_balance <= 50000: Yes (55/21)
##         :   :   Limit_balance > 50000: No (78/25)
##         :   Aug_repay = delay_3_mth:
##         :   :...Aug_amtpay <= 800: No (12/3)
##         :       Aug_amtpay > 800: Yes (6)
##         Sex = Male:
##         :...Apr_repay in {delay_4_mth,delay_5_mth,delay_6_mth,
##             :             pay_duly}: No (19/5)
##             Apr_repay in {delay_7_mth,delay_8_mth}: Yes (1)
##             Apr_repay = delay_2_mth:
##             :...Jun_repay in {delay_1_mth,delay_3_mth,delay_4_mth,delay_5_mth,
##             :   :             delay_6_mth,delay_7_mth,delay_8_mth,
##             :   :             pay_duly}: Yes (18/5)
##             :   Jun_repay = delay_2_mth:
##             :   :...Limit_balance <= 20000: No (28/10)
##             :       Limit_balance > 20000: Yes (88/27)
##             Apr_repay = delay_3_mth:
##             :...May_repay in {delay_2_mth,delay_3_mth,delay_5_mth,
##                 :             delay_7_mth}: Yes (5)
##                 May_repay = delay_4_mth:
##                 :...Age <= 36: No (9/2)
##                     Age > 36: Yes (6/1)
##
##
## Evaluation on training data (2821 cases):
##
##      Decision Tree
##    ----------------
```

```
##    Size      Errors
##
##      19  559(19.8%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##    1641   217    (a): class No
##     342   621    (b): class Yes
##
##
##  Attribute usage:
##
##  100.00% Sept_repay
##   93.65% Aug_repay
##   76.00% May_repay
##   69.05% Apr_repay
##   13.12% Sex
##    8.83% Limit_balance
##    4.75% Jun_repay
##    0.64% Aug_amtpay
##    0.53% Age
##
##
## Time: 0.0 secs
```

Conclusion

In short, four different classification methods have been used. Based on the analysis, all the classification methods have an average of 77% class accuracy. Using the naive Bayes classification method as a benchmark for the other classification methods shows that all the other classification methods are 'fitted'. However, from a business perspective, the C5.0 decision tree is the best method to identify which customer will default their credit, because it is able to capture all the important features that are actually related to predicting the target variables. By comparison,rpart decision tree shows that it has a tendency of overfitting. As the example shows in the graph of decision tree after the parameter tuning, only one variable is important to determine whether a person will default their credit or not, which is the variable of May repayment. However, this does not make sense for business purposes, where it is just looking at one variable and making a conclusion on individual credibility. Furthermore, using k-NN method it does not explicitly compute decision boundaries. It is not practical for business decision making. Therefore, the C50 decision tree and rpart decision tree before parameter tuning is the best method for this project.

References

Altman, N. (1992). An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. The American Statistician, 46(3), 175. http://dx.doi.org/10.2307/2685209

Bischl B, Lang M, Kotthoff L, Schiffner J, Richter J, Studerus E, Casalicchio G and Jones Z (2016). "mlr: Machine Learning in R."*Journal of Machine Learning Research*, *17*(170), pp. 1-5. <URL: http://jmlr.org/papers/v17/15-066.html>.

Freedman, D. A. (2009). Statistical models: theory and practice. cambridge university press.

Witten, I. H., Frank, E., Hall, M. A., & Pal, C. J. (2016). Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann.

Kelleher, J. D., Mac Namee, B., & D'Arcy, A. (2015). Fundamentals of machine learning for predictive data analytics: algorithms, worked examples, and case studies. MIT Press.

Metz CE (1978) Basic principles of ROC analysis. Seminars in Nuclear Medicine 8:283-298.

Russell, S. J., & Norvig, P. (2016). Artificial intelligence: a modern approach. Malaysia; Pearson Education Limited,.

Simon Urbanek (2017). rJava: Low-Level R to Java Interface. R package version 0.9-9. https://CRAN.R-project.org/package=rJava

Stephen Milborrow (2017). rpart.plot: Plot 'rpart' Models: An Enhanced Version of 'plot.rpart'. R package version 2.1.2. https://CRAN.R-project.org/package=rpart.plot

Yeh, I., & Lien, C. (2009). The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. Expert Systems With Applications, 36(2), 2473-2480. http://dx.doi.org/10.1016/j.eswa.2007.12.020

Zweig MH, Campbell G (1993) Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine. Clinical Chemistry 39:561-577