Predicting Individual Good or Bad Credit Risk

Machine Learning - Applied Project - Phase 2

Saurabh Mallik (s3623575) & Dilip Chandra (s3574580)

Introduction

The purpose of the project is to build classifiers which will help in predicting whether or not an individual has good or bad credit risk from German Credit Dataset, which was sourced from UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)). The project is segregated into two distinct stages. In stage one we created classifiers that helped the machine learning process to provide prediction. Stage one comprised of data pre-processing, exploration and visualisation. In stage two, we focus on model building, model fitting and evaluations of binary-classifiers.

The report is further divided into the following parts:
1. Methodology: explains what types of models were used and what parameters were set for machine learning algorithms. 2. Classifiers: Here we further discuss the detailed performance of each classifier for followed by discussion about fine-tuning process for the classifiers used. 3. Evaluation: Evaluate machine learning algorithms used prediction by use of resampling method and compare the results of the classifiers 4. Summary: We summarise the report and comment on the fit of each model.

Methodology

We use the classification approach for data modelling in this report. Classification approach uses supervised learning methods which develops the computer to learn from the data set followed by using this learning to classify new projections. We have considered three classifiers for this report - K Nearest Neighbour, Random Forest and Naive Bayes. Naive Bayes has been considered as the baseline classifier. For fine-tuning process, predication thresholds were adjusted by training each classifier to make probability predictions. Next, we split the data into training & testing data. This is done to apply different models on the training data and later test the accuracy of the predictions using the testing data. Test size of 0.7 refers to an 70-30 split of the original data into training & testing data respectively. Each set resembled the full data by having the same proportion of target classes. To refine the performance and to smoothen any imbalance class of the target feature, we perform a five-folded cross-validation stratified sampling on each classifier. To test the accuracy of each of the classifier on the test set and report the performance we rely on Confusion Matrix, mmce, Average classification accuracy rate.

Hyperparameter Tune-Fining

Naive-Bayes

Naive Bayes can often outperform more sophisticated classification methods. This classifier uses technique base in Bayesian theorem which is highly suitable when the dimensionality of the inputs is high. We have used Laplacian smoothing parameter to mitigate any zero probabilities as predictions. We used values that ranged from 0 to 30. The value of Laplacian parameter was 3.741 with a mean test error of 0.243 for Jobtype and 0.2453 for Age.

Random Forest

Random forest classifier is used to creates a set of decision trees from randomly selected sample data set from the actual data set. A single decision tree may be prone to noise, but aggregate of many decision trees would reduce the noise effect. It then aggregates the votes from different decision trees to decide the final class of the test object. This report we have considered 'mtry = sqrt(p)' where P refers to the number of descriptive features. For our project, p = 22, hence root of p is 4.69 Therefore, we experimented mtry = 3, 4, and 5. Number of trees to grow remains at the default value. The result was 4 with a mean test error of 0.263 for Jobtype and 0.28 for Age.

K-Nearest Neighbour

K Nearest Neighbour algorithm uses label points in the data set and these labelled points are used to learn how to label other data points. In simple words, to label a new point, if the new point is similar to its neighbour, then is labelled the same. We consider the K factor to tune hyper parameters and the optimal thresholds to confirm is the number of neighbours that are similar which can range from 2 to 20 hypnotically. The outcome was 20 with a mean test error of 0.263 for Jobtype and 0.26 for age.

Threshold adjustment using JobType as descriptive variable

```r
library(readr)
library(ggplot2)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
library(mlr)
```

```
## Loading required package: ParamHelpers
```

```r
library(tidyverse)
```

```
## ── Attaching packages ────────────────────────────────────────────────────────
──────────────────────────────────────────────────── tidyverse 1.2.1 ──
```

```
## ✔ tibble  3.0.1     ✔ purrr   0.3.4
## ✔ tidyr   0.8.1     ✔ stringr 1.3.1
## ✔ tibble  3.0.1     ✔ forcats 0.3.0
```

```
## ── Conflicts ─────────────────────────────────────────────────────────────────
────────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
```

```
library(GGally)
```

```
##
## Attaching package: 'GGally'
```

```
## The following object is masked from 'package:dplyr':
##
##     nasa
```

```
library(cowplot)
```

```
##
## Attaching package: 'cowplot'
```

```
## The following object is masked from 'package:ggplot2':
##
##     ggsave
```

Importing dataset.

```
credit <- read_csv("credit.csv")
```

```
## Parsed with column specification:
## cols(
##   .default = col_character(),
##   duration = col_integer(),
##   credit_amount = col_integer(),
##   installment_commitment = col_integer(),
##   residence_since = col_integer(),
##   age = col_integer(),
##   existing_credits = col_integer(),
##   num_dependents = col_integer()
## )
```

```
## See spec(...) for full column specifications.
```

```
names(credit) <- c('Checking', 'Duration', 'CreditHistory', 'Purpose', 'CreditAmount'
,
                    'Saving', 'Employment', 'Installment', 'Status', 'OtherParties',
'Residence',
                    'Asset', 'Age', 'OtherPlans', 'Housing', 'ExistingCredits', 'JobT
ype', 'Dependants',
                   'Phone', 'Foreign', 'CreditRisk')
credit[, sapply(credit, is.character )] <- sapply( credit[, sapply(credit, is.charact
er )], trimws)

credit <- credit %>%
         mutate(Purpose1 = ifelse( Purpose %in% c('domestic appliance', 'furniture/eq
uipment', 'radio/tv', 'repairs',                                     'retraining'),
'Household',
                                   ifelse( Purpose %in% c('new car', 'used car'), 'Ca
rs',
                                           ifelse( grepl('business', Purpose), 'Busin
ess',
                                                   ifelse( grepl('education', Purpos
e), 'Education',
                                                           ifelse( grepl('other', Pur
pose), 'Other', Purpose))))),
                Status1 = ifelse( Status != 'female div/dep/mar', 'Male', 'Female')
          )
credit[, sapply( credit, is.character )] <- lapply( credit[, sapply( credit, is.chara
cter )], factor)
```

Setting the training and test data.

```
set.seed(1234)
training_index <- sample(nrow(credit)*0.70)
test_index     <- setdiff(seq(1:nrow(credit)), training_index )
```

```
training_data  <- credit[training_index, ]
test_data      <- credit[test_index, ]
```

```
task <- makeClassifTask(data = training_data, target = 'CreditRisk', id = 'JobType')
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Provided data is not a pure data.frame but from class tbl_df, hence
## it will be converted.
```

Configuring learners with probability type for Naive Bayes, Random Forest and KKNN.

```
learner1 <- makeLearner('classif.naiveBayes', predict.type = 'prob')
learner2 <- makeLearner('classif.randomForest', predict.type = 'prob')
learner3 <- makeLearner('classif.kknn', predict.type = 'prob')
```

```
## Loading required package: kknn
```

For naiveBayes, we fine-tune Laplacian

```
ps1 <- makeParamSet(
  makeNumericParam('laplace', lower = 0, upper = 30)
)
```

For randomForest, we can fine-tune mtry i.e mumber of variables randomly sampled as candidates at each split. Following Breiman, L. (2001), Random Forests, Machine Learning 45(1), 5-32, we can try mtry = 3, 4, 5 as mtry = sqrt(p) where p = 13

```
ps2 <- makeParamSet(
  makeDiscreteParam('mtry', values = c(3,4,5))
)
```

For kknn, we can fine-tune k = 2 to 20

```
ps3 <- makeParamSet(
  makeDiscreteParam('k', values = seq(2, 20, by = 1))
)
```

Next, we configure tuning control search and a 5-CV version of stratified sampling

```
ctrl  <- makeTuneControlGrid()
rdesc <- makeResampleDesc("CV", iters = 5L, stratify = TRUE)


tunedLearner1 <- makeTuneWrapper(learner1, rdesc, mmce, ps1, ctrl)
tunedLearner2 <- makeTuneWrapper(learner2, rdesc, mmce, ps2, ctrl)
tunedLearner3 <- makeTuneWrapper(learner3, rdesc, mmce, ps3, ctrl)


tunedMod1  <- train(tunedLearner1, task)
```

```
## [Tune] Started tuning learner classif.naiveBayes for parameter set:
```

```
##             Type len Def  Constr Req Tunable Trafo
## laplace numeric   -   - 0 to 30   -    TRUE     -
```

```
## With control class: TuneControlGrid
```

```
## Imputation value: 1
```

```
## [Tune-x] 1: laplace=0
```

```
## [Tune-y] 1: mmce.test.mean=0.2770466; time: 0.0 min
```

```
## [Tune-x] 2: laplace=3.33
```

```
## [Tune-y] 2: mmce.test.mean=0.2770773; time: 0.0 min
```

```
## [Tune-x] 3: laplace=6.67
## [Tune-y] 3: mmce.test.mean=0.2784752; time: 0.0 min
```

```
## [Tune-x] 4: laplace=10
```

```
## [Tune-y] 4: mmce.test.mean=0.2813121; time: 0.0 min
```

```
## [Tune-x] 5: laplace=13.3
```

```
## [Tune-y] 5: mmce.test.mean=0.2813121; time: 0.0 min
```

```
## [Tune-x] 6: laplace=16.7
```

```
## [Tune-y] 6: mmce.test.mean=0.2856184; time: 0.0 min
```

```
## [Tune-x] 7: laplace=20
```

```
## [Tune-y] 7: mmce.test.mean=0.2899041; time: 0.0 min
```

```
## [Tune-x] 8: laplace=23.3
```

```
## [Tune-y] 8: mmce.test.mean=0.2870775; time: 0.0 min
```

```
## [Tune-x] 9: laplace=26.7
```

```
## [Tune-y] 9: mmce.test.mean=0.2899142; time: 0.0 min
```

```
## [Tune-x] 10: laplace=30
```

```
## [Tune-y] 10: mmce.test.mean=0.2870571; time: 0.0 min
```

```
## [Tune] Result: laplace=0 : mmce.test.mean=0.2770466
```

```
tunedMod2  <- train(tunedLearner2, task)
```

```
## [Tune] Started tuning learner classif.randomForest for parameter set:
```

```
##          Type len Def Constr Req Tunable Trafo
## mtry discrete   -   -  3,4,5   -    TRUE     -
```

```
## With control class: TuneControlGrid
```

```
## Imputation value: 1
## [Tune-x] 1: mtry=3
```

```
## [Tune-y] 1: mmce.test.mean=0.2342137; time: 0.1 min
```

```
## [Tune-x] 2: mtry=4
```

```
## [Tune-y] 2: mmce.test.mean=0.2399792; time: 0.1 min
```

```
## [Tune-x] 3: mtry=5
```

```
## [Tune-y] 3: mmce.test.mean=0.2528472; time: 0.1 min
```

```
## [Tune] Result: mtry=3 : mmce.test.mean=0.2342137
```

```
tunedMod3  <- train(tunedLearner3, task)
```

```
## [Tune] Started tuning learner classif.kknn for parameter set:
```

```
##       Type len Def                                   Constr Req Tunable Trafo
## k discrete   -   - 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,...   -    TRUE     -
```

```
## With control class: TuneControlGrid
```

```
## Imputation value: 1
```

```
## [Tune-x] 1: k=2
```

```
## [Tune-y] 1: mmce.test.mean=0.3086488; time: 0.0 min
```

```
## [Tune-x] 2: k=3
```

```
## [Tune-y] 2: mmce.test.mean=0.3086488; time: 0.0 min
```

```
## [Tune-x] 3: k=4
```

```
## [Tune-y] 3: mmce.test.mean=0.3086488; time: 0.0 min
```

```
## [Tune-x] 4: k=5
```

```
## [Tune-y] 4: mmce.test.mean=0.2901077; time: 0.0 min
```

```
## [Tune-x] 5: k=6
## [Tune-y] 5: mmce.test.mean=0.2858015; time: 0.0 min
```

```
## [Tune-x] 6: k=7
```

```
## [Tune-y] 6: mmce.test.mean=0.2858015; time: 0.0 min
```

```
## [Tune-x] 7: k=8
```

```
## [Tune-y] 7: mmce.test.mean=0.2700870; time: 0.0 min
```

```
## [Tune-x] 8: k=9
```

```
## [Tune-y] 8: mmce.test.mean=0.2672502; time: 0.0 min
```

```
## [Tune-x] 9: k=10
```

```
## [Tune-y] 9: mmce.test.mean=0.2615256; time: 0.0 min
```

```
## [Tune-x] 10: k=11
```

```
## [Tune-y] 10: mmce.test.mean=0.2615256; time: 0.0 min
```

```
## [Tune-x] 11: k=12
```

```
## [Tune-y] 11: mmce.test.mean=0.2586786; time: 0.0 min
```

```
## [Tune-x] 12: k=13
```

```
## [Tune-y] 12: mmce.test.mean=0.2515150; time: 0.0 min
```

```
## [Tune-x] 13: k=14
```

```
## [Tune-y] 13: mmce.test.mean=0.2500864; time: 0.0 min
```

```
## [Tune-x] 14: k=15
```

```
## [Tune-y] 14: mmce.test.mean=0.2543620; time: 0.0 min
```

```
## [Tune-x] 15: k=16
```

```
## [Tune-y] 15: mmce.test.mean=0.2529232; time: 0.0 min
## [Tune-x] 16: k=17
```

```
## [Tune-y] 16: mmce.test.mean=0.2571988; time: 0.0 min
```

```
## [Tune-x] 17: k=18
```

```
## [Tune-y] 17: mmce.test.mean=0.2557906; time: 0.0 min
```

```
## [Tune-x] 18: k=19
```

```
## [Tune-y] 18: mmce.test.mean=0.2557906; time: 0.0 min
```

```
## [Tune-x] 19: k=20
```

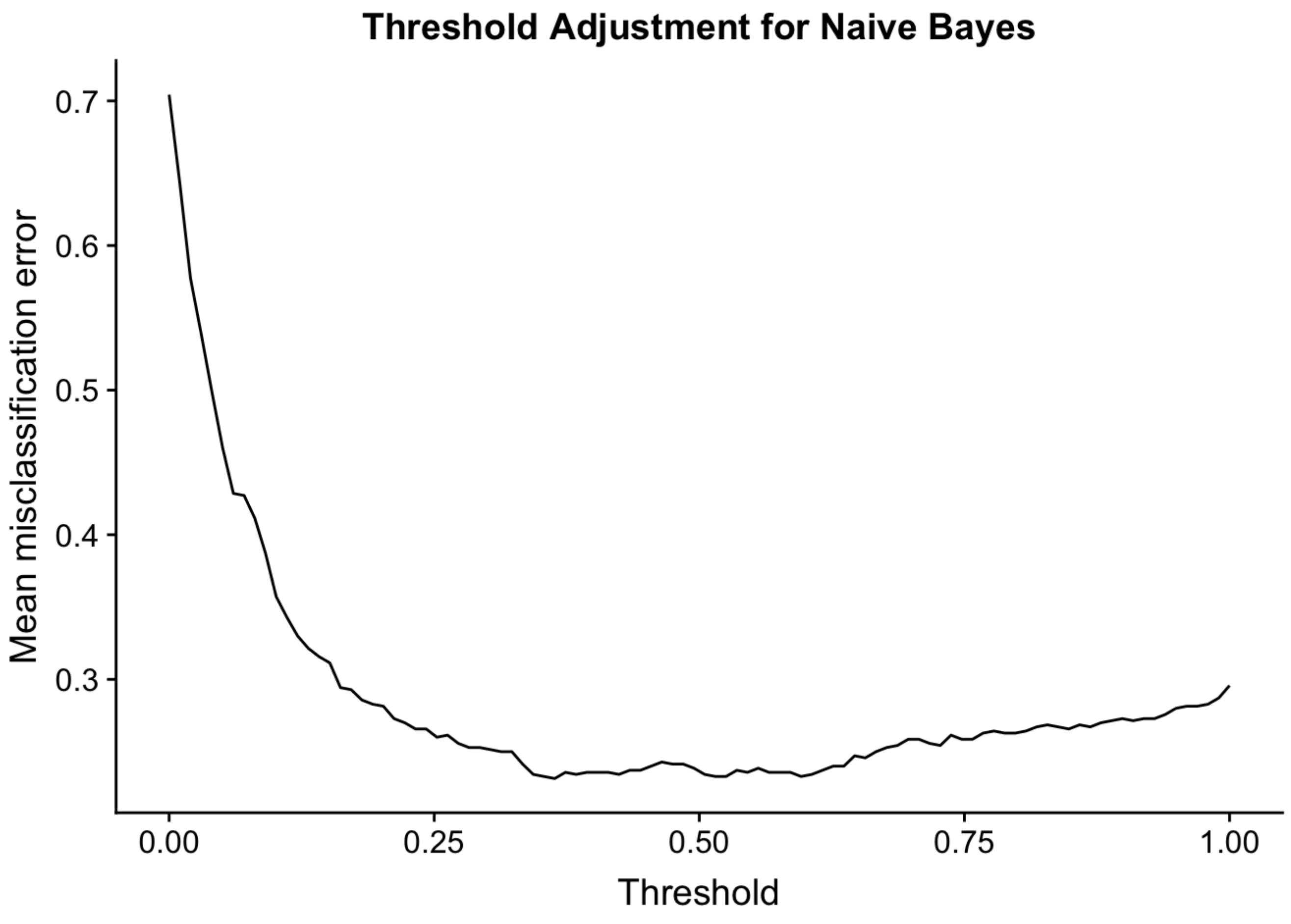
```
## [Tune-y] 19: mmce.test.mean=0.2572090; time: 0.0 min
```

```
## [Tune] Result: k=14 : mmce.test.mean=0.2500864
```

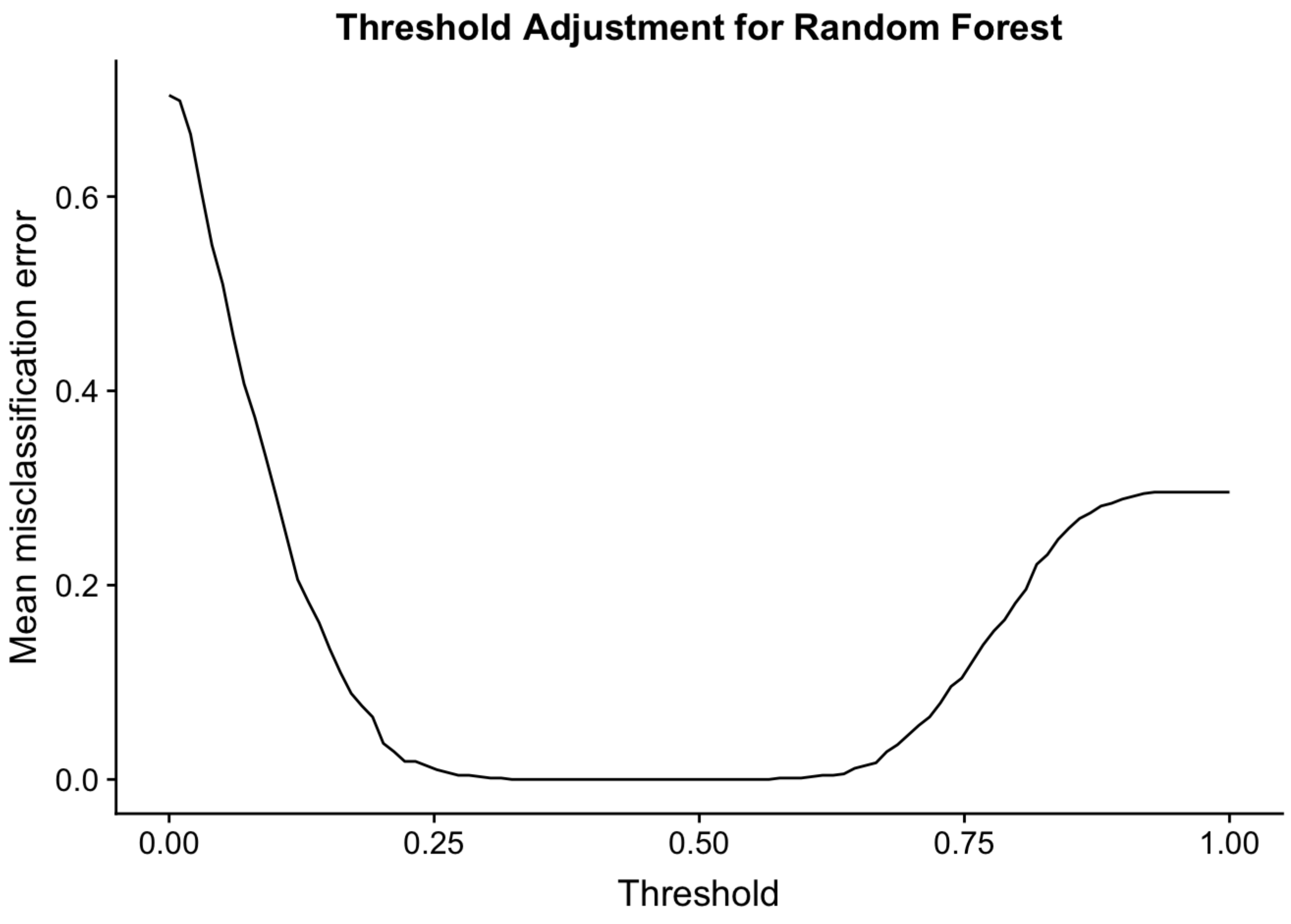
```
tunedPred1 <- predict(tunedMod1, task)
tunedPred2 <- predict(tunedMod2, task)
tunedPred3 <- predict(tunedMod3, task)


d1 <- generateThreshVsPerfData(tunedPred1, measures = list(mmce))
d2 <- generateThreshVsPerfData(tunedPred2, measures = list(mmce))
d3 <- generateThreshVsPerfData(tunedPred3, measures = list(mmce))
```

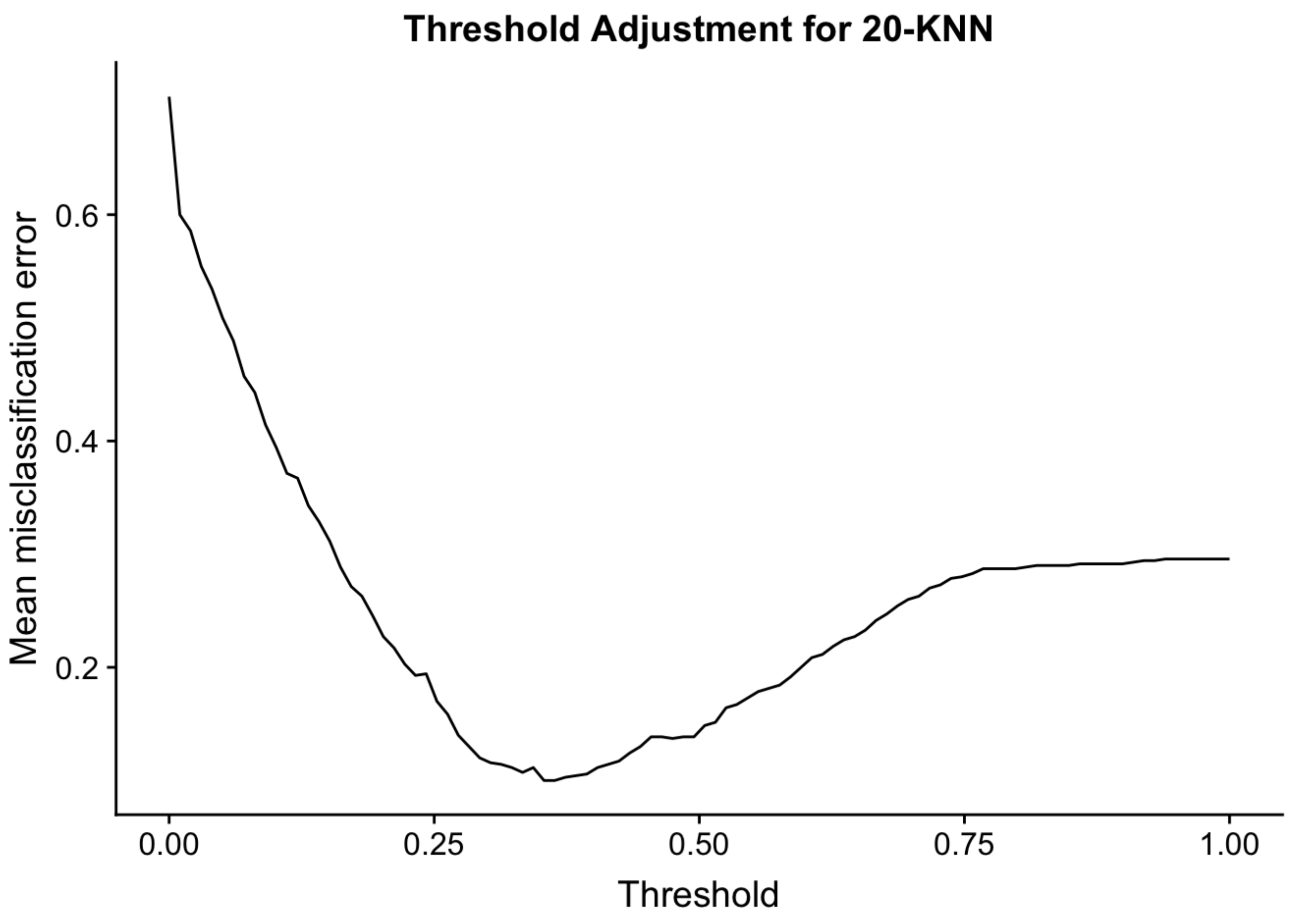
```
mlr::plotThreshVsPerf(d1) + ggplot2::labs(title = 'Threshold Adjustment for Naive Bay
es', x = 'Threshold')
```



```
mlr::plotThreshVsPerf(d2) + ggplot2::labs(title = 'Threshold Adjustment for Random Fo
rest', x = 'Threshold')
```



```
mlr::plotThreshVsPerf(d3) + ggplot2::labs(title = 'Threshold Adjustment for 20-KNN',
 x = 'Threshold')
```



The following plots depict the value of mmce vs. the range of probability thresholds. The thresholds were approximately 0.55, 0.65, and 0.35 for NB, RF, and 20-KNN classifiers respectively. These thresholds were used to determine the probability of an individual defaulting or not.

Evaluation of classifiers

```
threshold1 <- d1$data$threshold[ which.min(d1$data$mmce) ]
threshold2 <- d2$data$threshold[ which.min(d2$data$mmce) ]
threshold3 <- d3$data$threshold[ which.min(d3$data$mmce) ]



testPred1 <- predict(tunedMod1, newdata = test_data)
```

```
## Warning in predict.WrappedModel(tunedMod1, newdata = test_data): Provided data
## for prediction is not a pure data.frame but from class tbl_df, hence it will be
## converted.
```

```
testPred2 <- predict(tunedMod2, newdata = test_data)
```

```
## Warning in predict.WrappedModel(tunedMod2, newdata = test_data): Provided data
## for prediction is not a pure data.frame but from class tbl_df, hence it will be
## converted.
```

```
testPred3 <- predict(tunedMod3, newdata = test_data)
```

```
## Warning in predict.WrappedModel(tunedMod3, newdata = test_data): Provided data
## for prediction is not a pure data.frame but from class tbl_df, hence it will be
## converted.
```

```
testPred1 <- setThreshold(testPred1, threshold1 )
testPred2 <- setThreshold(testPred2, threshold2 )
testPred3 <- setThreshold(testPred3, threshold3 )
```

By making use of the obtained parameters and threshold levels, we calculated the confusion matrix for each classifier.

The confusion matrix of Naive Bayes classifer is as follow:

```
calculateConfusionMatrix( testPred1,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     bad       good      -err.-
##   bad    0.69/0.59 0.31/0.15 0.31
##   good   0.21/0.41 0.79/0.85 0.21
##   -err.-      0.41      0.15 0.24
## 
## 
## Absolute confusion matrix:
##         predicted
## true     bad good -err.-
##   bad     64   29     29
##   good    44  163     44
##   -err.-  44   29     73
```

Classification accuracy for Naive Bayes is 75.66%, which shows a good performance.

The confusion matrix of Random Forest classifer is as follows

```
calculateConfusionMatrix( testPred2,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     bad       good      -err.-
##   bad    0.73/0.56 0.27/0.14 0.27
##   good   0.26/0.44 0.74/0.86 0.26
##   -err.-      0.44      0.14 0.26
## 
## 
## Absolute confusion matrix:
##         predicted
## true     bad good -err.-
##   bad     68   25     25
##   good    54  153     54
##   -err.-  54   25     79
```

Classification accuracy for Random Forest is 73.66%, which shows a good performance.

The confusion matrix of 20-KNN classifer is as follows

```
calculateConfusionMatrix( testPred3,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     bad       good      -err.-
##   bad    0.56/0.63 0.44/0.19 0.44
##   good   0.14/0.37 0.86/0.81 0.14
##   -err.-      0.37      0.19 0.24
##
##
## Absolute confusion matrix:
##         predicted
## true     bad good -err.-
##   bad     52   41     41
##   good    30  177     30
##   -err.-  30   41     71
```

Classification accuracy for 20-kNN is 76.33%, which shows a good performance.

All classifiers accurately predicted individual having good credit risk, but not bad credit risk. The class accuracy difference was substantial. Based on class accuracy and mmce, we concluded that the 20-KNN classifer was the best model using id = JobType.

Threshold adjustment using Age as descriptive variable

```
task2 <- makeClassifTask(data = training_data, target = 'CreditRisk', id = 'Age')
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Provided data is not a pure data.frame but from class tbl_df, hence
## it will be converted.
```

Configuring learners with probability type for Naive Bayes, Random Forest and KKNN.

```
learner4 <- makeLearner('classif.naiveBayes', predict.type = 'prob')
learner5 <- makeLearner('classif.randomForest', predict.type = 'prob')
learner6 <- makeLearner('classif.kknn', predict.type = 'prob')
```

For naiveBayes, we fine-tune Laplacian

```
ps4 <- makeParamSet(
  makeNumericParam('laplace', lower = 0, upper = 30)
)
```

For randomForest, we can fine-tune mtry i.e mumber of variables randomly sampled as candidates at each split. Following Breiman, L. (2001), Random Forests, Machine Learning 45(1), 5-32, we can try mtry = 3, 4, 5 as mtry = sqrt(p) where p = 13

```
ps5 <- makeParamSet(
  makeDiscreteParam('mtry', values = c(3,4,5))
)
```

For kknn, we can fine-tune k = 2 to 20

```
ps6 <- makeParamSet(
  makeDiscreteParam('k', values = seq(2, 20, by = 1))
)
```

Next, we configure tuning control search and a 5-CV version of stratified sampling

```
ctrl2  <- makeTuneControlGrid()
rdesc2 <- makeResampleDesc("CV", iters = 5L, stratify = TRUE)


tunedLearner4 <- makeTuneWrapper(learner4, rdesc2, mmce, ps4, ctrl2)
tunedLearner5 <- makeTuneWrapper(learner5, rdesc2, mmce, ps5, ctrl2)
tunedLearner6 <- makeTuneWrapper(learner6, rdesc2, mmce, ps6, ctrl2)


tunedMod4  <- train(tunedLearner4, task2)
```

```
## [Tune] Started tuning learner classif.naiveBayes for parameter set:
```

```
##            Type len Def  Constr Req Tunable Trafo
## laplace numeric   -   - 0 to 30   -    TRUE     -
```

```
## With control class: TuneControlGrid
```

```
## Imputation value: 1
```

```
## [Tune-x] 1: laplace=0
```

```
## [Tune-y] 1: mmce.test.mean=0.2684955; time: 0.0 min
```

```
## [Tune-x] 2: laplace=3.33
```

```
## [Tune-y] 2: mmce.test.mean=0.2741587; time: 0.0 min
```

```
## [Tune-x] 3: laplace=6.67
```

```
## [Tune-y] 3: mmce.test.mean=0.2827712; time: 0.0 min
```

```
## [Tune-x] 4: laplace=10
```

```
## [Tune-y] 4: mmce.test.mean=0.2841999; time: 0.0 min
```

```
## [Tune-x] 5: laplace=13.3
## [Tune-y] 5: mmce.test.mean=0.2942306; time: 0.0 min
```

```
## [Tune-x] 6: laplace=16.7
```

```
## [Tune-y] 6: mmce.test.mean=0.2956693; time: 0.0 min
```

```
## [Tune-x] 7: laplace=20
```

```
## [Tune-y] 7: mmce.test.mean=0.2913939; time: 0.0 min
```

```
## [Tune-x] 8: laplace=23.3
```

```
## [Tune-y] 8: mmce.test.mean=0.2956797; time: 0.0 min
```

```
## [Tune-x] 9: laplace=26.7
```

```
## [Tune-y] 9: mmce.test.mean=0.2913837; time: 0.0 min
```

```
## [Tune-x] 10: laplace=30
```

```
## [Tune-y] 10: mmce.test.mean=0.2885266; time: 0.0 min
```

```
## [Tune] Result: laplace=0 : mmce.test.mean=0.2684955
```

```
tunedMod5  <- train(tunedLearner5, task2)
```

```
## [Tune] Started tuning learner classif.randomForest for parameter set:
```

```
##          Type len Def Constr Req Tunable Trafo
## mtry discrete   -   -  3,4,5   -    TRUE     -
```

```
## With control class: TuneControlGrid
```

```
## Imputation value: 1
```

```
## [Tune-x] 1: mtry=3
```

```
## [Tune-y] 1: mmce.test.mean=0.2499132; time: 0.1 min
```

```
## [Tune-x] 2: mtry=4
```

```
## [Tune-y] 2: mmce.test.mean=0.2613214; time: 0.1 min
## [Tune-x] 3: mtry=5
```

```
## [Tune-y] 3: mmce.test.mean=0.2527091; time: 0.1 min
```

```
## [Tune] Result: mtry=3 : mmce.test.mean=0.2499132
```

```
tunedMod6  <- train(tunedLearner6, task2)
```

```
## [Tune] Started tuning learner classif.kknn for parameter set:
```

```
##       Type len Def                                   Constr Req Tunable Trafo
## k discrete   -   - 2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,...   -    TRUE     -
```

```
## With control class: TuneControlGrid
```

```
## Imputation value: 1
```

```
## [Tune-x] 1: k=2
```

```
## [Tune-y] 1: mmce.test.mean=0.2970835; time: 0.0 min
```

```
## [Tune-x] 2: k=3
```

```
## [Tune-y] 2: mmce.test.mean=0.2970835; time: 0.0 min
```

```
## [Tune-x] 3: k=4
```

```
## [Tune-y] 3: mmce.test.mean=0.2970835; time: 0.0 min
```

```
## [Tune-x] 4: k=5
```

```
## [Tune-y] 4: mmce.test.mean=0.2643269; time: 0.0 min
```

```
## [Tune-x] 5: k=6
```

```
## [Tune-y] 5: mmce.test.mean=0.2543166; time: 0.0 min
```

```
## [Tune-x] 6: k=7
```

```
## [Tune-y] 6: mmce.test.mean=0.2543166; time: 0.0 min
```

```
## [Tune-x] 7: k=8
## [Tune-y] 7: mmce.test.mean=0.2514084; time: 0.0 min
```

```
## [Tune-x] 8: k=9
```

```
## [Tune-y] 8: mmce.test.mean=0.2514288; time: 0.0 min
```

```
## [Tune-x] 9: k=10
```

```
## [Tune-y] 9: mmce.test.mean=0.2485305; time: 0.0 min
```

```
## [Tune-x] 10: k=11
```

```
## [Tune-y] 10: mmce.test.mean=0.2542347; time: 0.0 min
```

```
## [Tune-x] 11: k=12
```

```
## [Tune-y] 11: mmce.test.mean=0.2556735; time: 0.0 min
```

```
## [Tune-x] 12: k=13
```

```
## [Tune-y] 12: mmce.test.mean=0.2499488; time: 0.0 min
```

```
## [Tune-x] 13: k=14
```

```
## [Tune-y] 13: mmce.test.mean=0.2542246; time: 0.0 min
```

```
## [Tune-x] 14: k=15
```

```
## [Tune-y] 14: mmce.test.mean=0.2556838; time: 0.0 min
```

```
## [Tune-x] 15: k=16
```

```
## [Tune-y] 15: mmce.test.mean=0.2556941; time: 0.0 min
```

```
## [Tune-x] 16: k=17
```

```
## [Tune-y] 16: mmce.test.mean=0.2556737; time: 0.0 min
```

```
## [Tune-x] 17: k=18
```

```
## [Tune-y] 17: mmce.test.mean=0.2599697; time: 0.0 min
## [Tune-x] 18: k=19
```

```
## [Tune-y] 18: mmce.test.mean=0.2542551; time: 0.0 min
```

```
## [Tune-x] 19: k=20
```

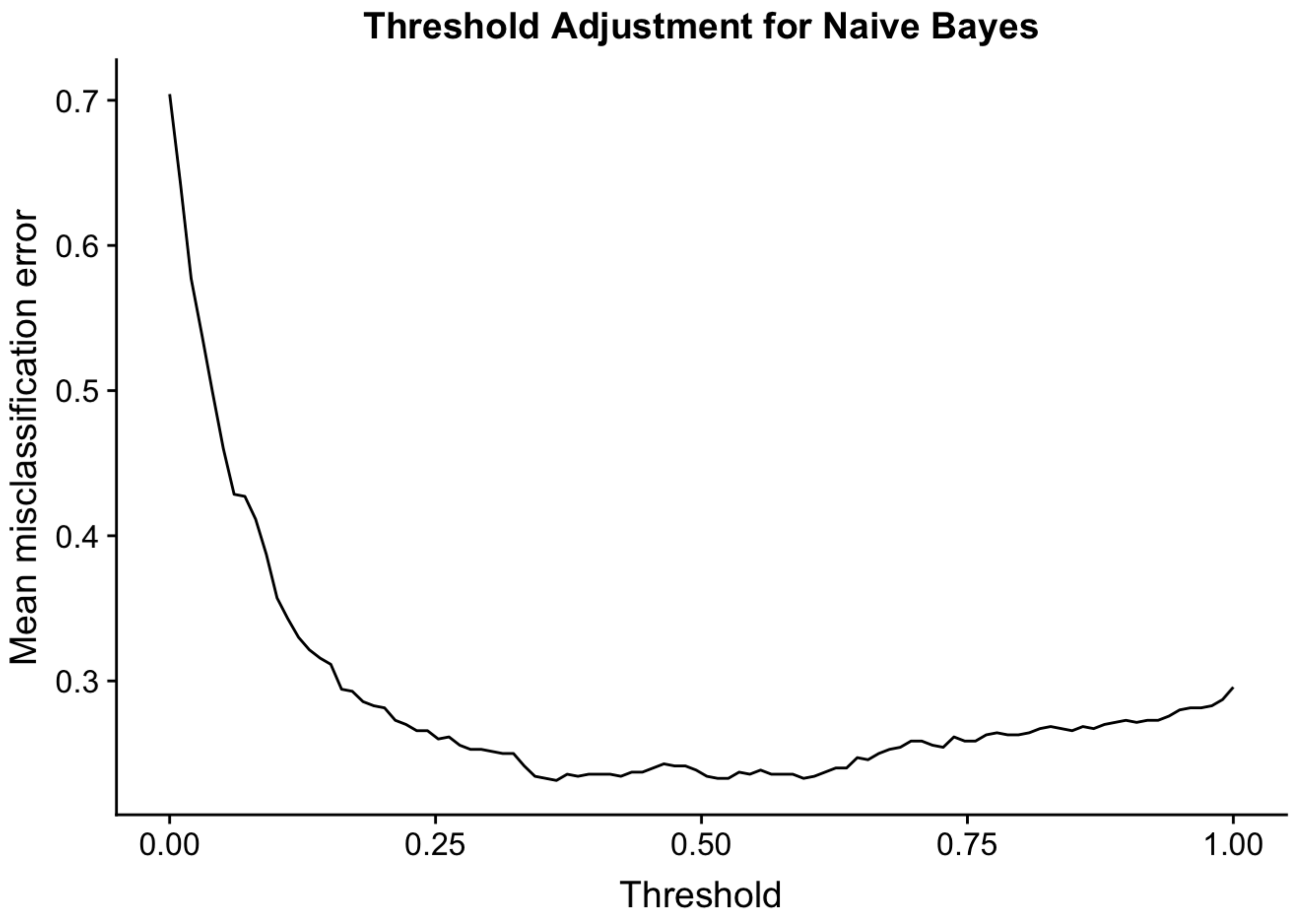
```
## [Tune-y] 19: mmce.test.mean=0.2585308; time: 0.0 min
```

```
## [Tune] Result: k=10 : mmce.test.mean=0.2485305
```

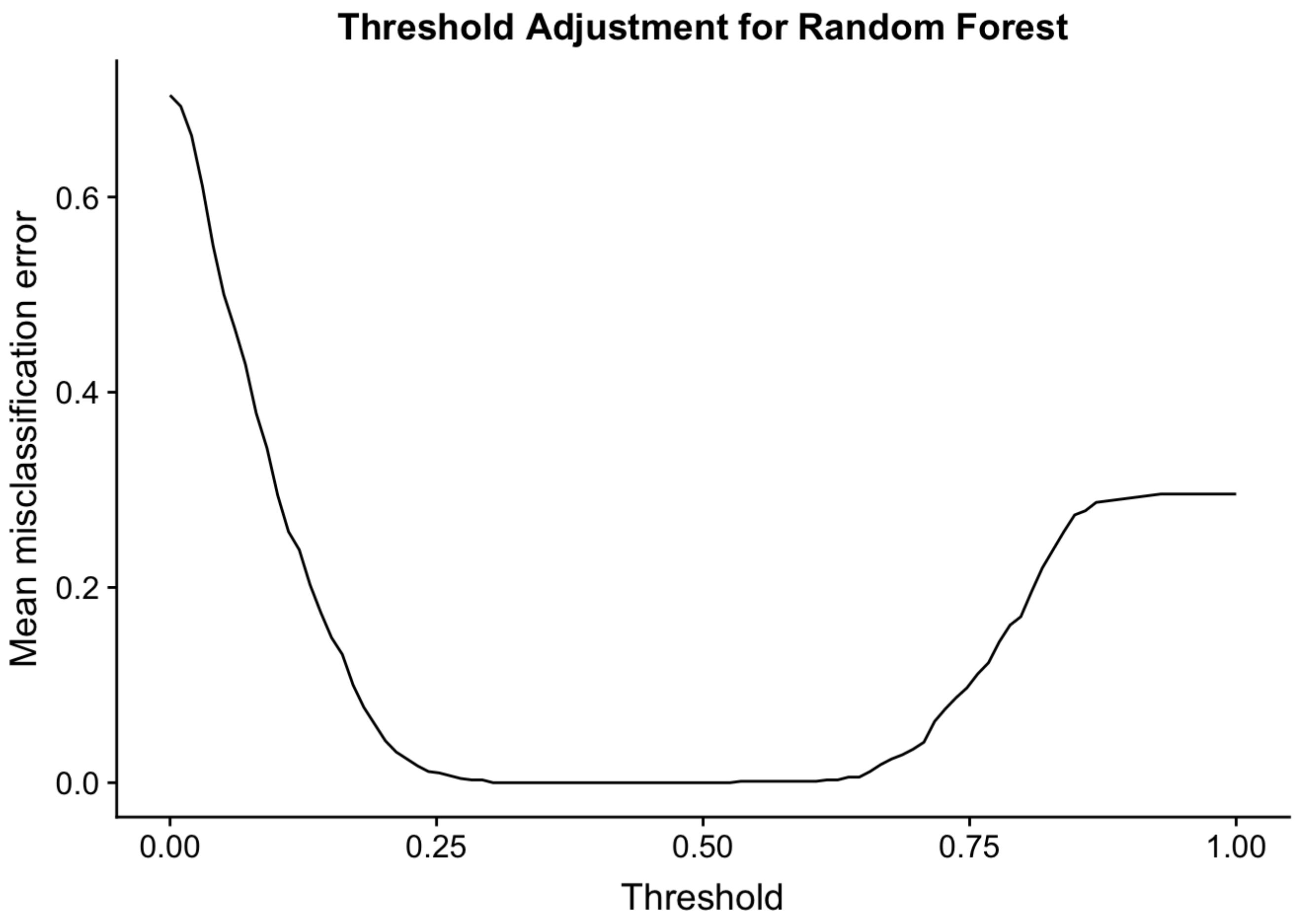
```
tunedPred4 <- predict(tunedMod4, task2)
tunedPred5 <- predict(tunedMod5, task2)
tunedPred6 <- predict(tunedMod6, task2)


d4 <- generateThreshVsPerfData(tunedPred4, measures = list(mmce))
d5 <- generateThreshVsPerfData(tunedPred5, measures = list(mmce))
d6 <- generateThreshVsPerfData(tunedPred6, measures = list(mmce))
```

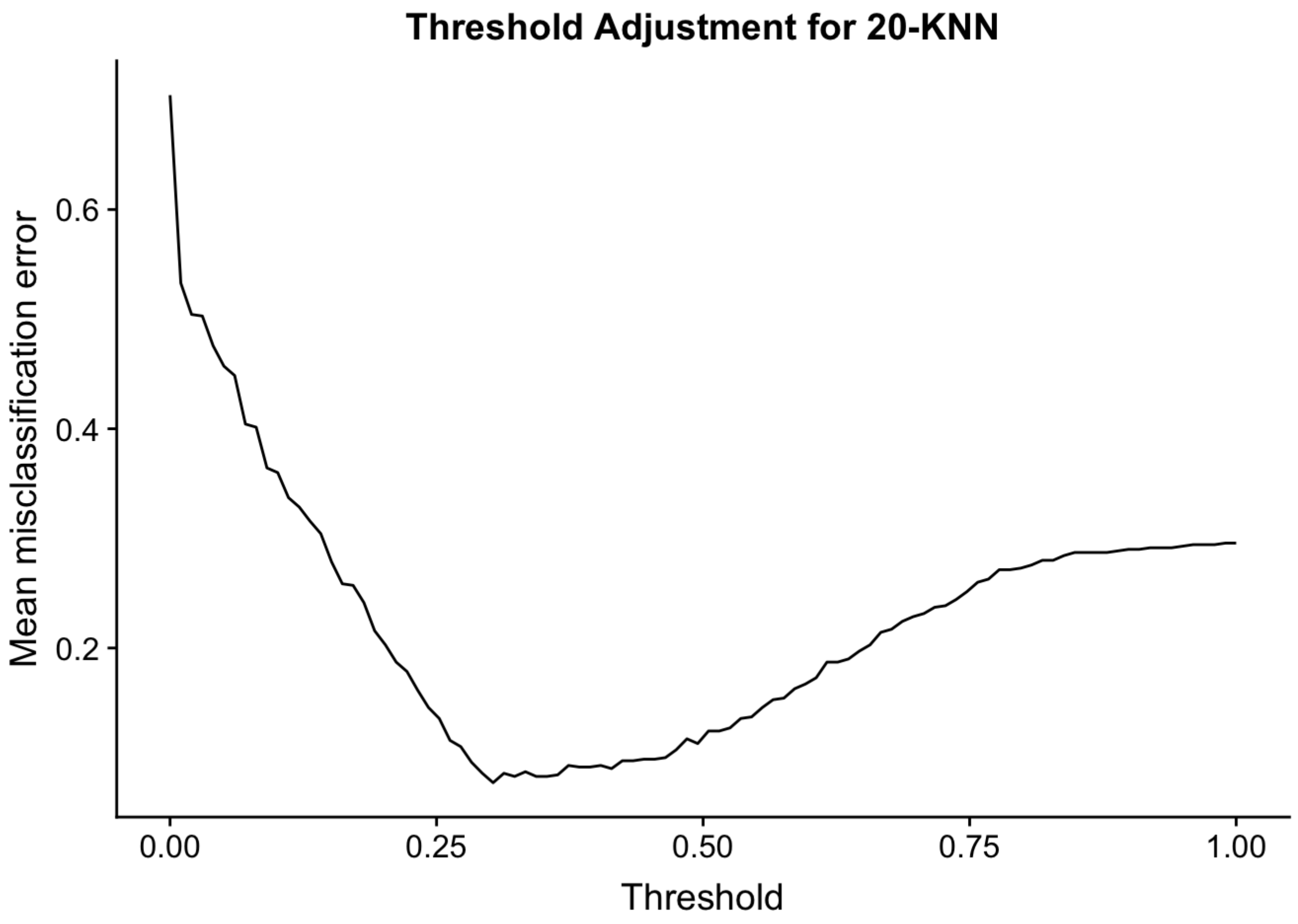
```
mlr::plotThreshVsPerf(d4) + ggplot2::labs(title = 'Threshold Adjustment for Naive Bay
es', x = 'Threshold')
```



```
mlr::plotThreshVsPerf(d5) + ggplot2::labs(title = 'Threshold Adjustment for Random Fo
rest', x = 'Threshold')
```



```
mlr::plotThreshVsPerf(d6) + ggplot2::labs(title = 'Threshold Adjustment for 20-KNN',
 x = 'Threshold')
```



The following plots depict the value of mmce vs. the range of probability thresholds. The thresholds were approximately 0.50, 0.62, and 0.3 for NB, RF, and 20-KNN classifiers respectively. These thresholds were used to determine the probability of an individual defaulting or not.

Evaluation of classifiers

```
threshold4 <- d4$data$threshold[ which.min(d4$data$mmce) ]
threshold5 <- d5$data$threshold[ which.min(d5$data$mmce) ]
threshold6 <- d6$data$threshold[ which.min(d6$data$mmce) ]

testPred4 <- predict(tunedMod4, newdata = test_data)
```

```
## Warning in predict.WrappedModel(tunedMod4, newdata = test_data): Provided data
## for prediction is not a pure data.frame but from class tbl_df, hence it will be
## converted.
```

```
testPred5 <- predict(tunedMod5, newdata = test_data)
```

```
## Warning in predict.WrappedModel(tunedMod5, newdata = test_data): Provided data
## for prediction is not a pure data.frame but from class tbl_df, hence it will be
## converted.
```

```
testPred6 <- predict(tunedMod6, newdata = test_data)
```

```
## Warning in predict.WrappedModel(tunedMod6, newdata = test_data): Provided data
## for prediction is not a pure data.frame but from class tbl_df, hence it will be
## converted.
```

```
testPred4 <- setThreshold(testPred4, threshold4 )
testPred5 <- setThreshold(testPred5, threshold5 )
testPred6 <- setThreshold(testPred6, threshold6 )
```

By making use of the obtained parameters and threshold levels, we calculated the confusion matrix for each classifier.

The confusion matrix of Naive Bayes classifer is as follow:

```
calculateConfusionMatrix( testPred4,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     bad       good      -err.-
##   bad    0.69/0.59 0.31/0.15 0.31
##   good   0.21/0.41 0.79/0.85 0.21
##   -err.-      0.41      0.15 0.24
##
##
## Absolute confusion matrix:
##         predicted
## true     bad good -err.-
##   bad     64   29     29
##   good    44  163     44
##   -err.-  44   29     73
```

Classification accuracy for Naive Bayes is 75.66%, which shows a good performance.

The confusion matrix of Random Forest classifer is as follows

```
calculateConfusionMatrix( testPred5,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     bad       good      -err.-
##   bad    0.76/0.53 0.24/0.13 0.24
##   good   0.30/0.47 0.70/0.87 0.30
##   -err.-      0.47      0.13 0.28
##
##
## Absolute confusion matrix:
##         predicted
## true     bad good -err.-
##   bad     71   22     22
##   good    62  145     62
##   -err.-  62   22     84
```

Classification accuracy for Random Forest is 73.33%, which shows a good performance.

The confusion matrix of 20-KNN classifer is as follows

```
calculateConfusionMatrix( testPred6,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     bad       good      -err.-
##   bad    0.62/0.57 0.38/0.18 0.38
##   good   0.21/0.43 0.79/0.82 0.21
##   -err.-      0.43      0.18 0.26
##
##
## Absolute confusion matrix:
##         predicted
## true     bad good -err.-
##   bad     58   35     35
##   good    43  164     43
##   -err.-  43   35     78
```

Classification accuracy for 20-kNN is 74%, which shows a good performance.

All classifiers accurately predicted individual having good credit risk, but not bad credit risk. The class accuracy difference was substantial. Based on class accuracy and mmce, we concluded that the Naive Bayes classifer was the best model using id = Age.

Discussion

The above section shows that all three of our classifiers performed accurately for good credit risk, however they performed poorly for bad credit risk despite of stratified sampling. This implies the prevalence of imbalance class problem. In order to get more effective results, a good approach could be a cost-sensitive classification where we could have allocated more cost to true positive groups i.e. the correctly predicted bad credit risk class. Another alternative would be under- or oversampling to adjust the class balance, despite the risk of inducing biases.

The Naive Bayes model goes with the assumption that the descriptive features to follow normality, which is not necessarily true. The solution could be a transformation on numeric features.

Based mmce, the Random Forest classifer underperformed the KNN and Naive Bayes classifier. This highlights the Random Forest classifier might not be appropriate given there were many binary features in the data. The Naive Bayes outperformed other models.

Conclusion

Among the three classifiers: - for JobType as ID - the 20-KNN - and for Age as ID - the Naive Bayes produced the best performance in predicting individuals having good credit risk.

We split the data into training and test sets. Via a stratified sampling, we determined the optimal value of the selected hyperparameter of each classifier and the probability threshold. Despite this, the imbalance class issue still persisted and therefore reduced the class accuracy of the bad credit risk.

For future work, we proposed to consider cost-sensitive classification and under/over-sampling methods to mitigate the class imbalance.

References

“Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies”, John D. Kelleher, Brian Mac Namee, and Aoife D’Arcy, 1st Edition, MIT Press, 2015. “Predictive Analytics”, Eric Siegel, Wiley, 2016. “R Cookbook: Proven Recipes for Data Analysis, Statistics, and Graphics”, Paul Teetor, 1st Edition, O’reilly Cookbooks, 2011.