Bank Marketing Data- Machine Learning Project Phase 2

Piyush Bhatt

9 November 2019

```
knitr::opts_chunk$set(echo = TRUE)
```

Libraries

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.1     v purrr   0.3.4
## v tibble  3.0.1     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(reshape2) # manipulate data structure
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
library(dplyr)
library(spFSR)
```

```
## Loading required package: mlr
```

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

```
## 'mlr' is in maintenance mode since July 2019. Future development
## efforts will go into its successor 'mlr3' (<https://mlr3.mlr-org.com>).
```

```
## Loading required package: parallelMap
## Loading required package: parallel
```

```
## Loading required package: tictoc
```

```
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
## 
## Attaching package: 'randomForest'
```

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

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

```
library(kknn)
library(plotrix)
library(rpart)
library(rlang)
```

```
## 
## Attaching package: 'rlang'
```

```
## The following objects are masked from 'package:purrr':
## 
##     %@%, as_function, flatten, flatten_chr, flatten_dbl, flatten_int,
##     flatten_lgl, flatten_raw, invoke, list_along, modify, prepend,
##     splice
```

```
library(ggvis)
```

```
## 
## Attaching package: 'ggvis'
```

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

```
library(plyr)
```

```
## ------------------------------------------------------------------------------
```

```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)
```

```
## ------------------------------------------------------------------------------
```

```
## 
## Attaching package: 'plyr'
```

```
## The following objects are masked from 'package:dplyr':
## 
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
## The following object is masked from 'package:purrr':
## 
##     compact
```

```
library(rJava)
library(FSelector)
library(forcats)
library(readr)
library(randomForestSRC)
```

```
## 
##  randomForestSRC 2.9.3 
## 
##  Type rfsrc.news() to see new features, changes, and bug fixes.
## 
```

```
## 
## Attaching package: 'randomForestSRC'
```

```
## The following objects are masked from 'package:mlr':
## 
##     impute, subsample
```

```
## The following object is masked from 'package:purrr':
## 
##     partial
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:kknn':
##
##     contr.dummy
```

```
## The following object is masked from 'package:mlr':
##
##     train
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(mlbench)
library(ggplot2)
library(Hmisc)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: Formula
```

```
##
## Attaching package: 'Hmisc'
```

```
## The following object is masked from 'package:randomForestSRC':
##
##     impute
```

```
## The following objects are masked from 'package:plyr':
##
##     is.discrete, summarize
```

```
## The following object is masked from 'package:mlr':
##
##     impute
## The following objects are masked from 'package:dplyr':
## 
##     src, summarize
```

```
## The following objects are masked from 'package:base':
## 
##     format.pval, units
```

```
library(mosaic)
```

```
## Loading required package: ggformula
```

```
## Loading required package: ggstance
```

```
## 
## Attaching package: 'ggstance'
```

```
## The following objects are masked from 'package:ggplot2':
## 
##     geom_errorbarh, GeomErrorbarh
```

```
## 
## New to ggformula?  Try the tutorials: 
## 	learnr::run_tutorial("introduction", package = "ggformula")
## 	learnr::run_tutorial("refining", package = "ggformula")
```

```
## Loading required package: mosaicData
```

```
## Loading required package: Matrix
```

```
## 
## Attaching package: 'Matrix'
```

```
## The following object is masked from 'package:ggvis':
## 
##     band
```

```
## The following objects are masked from 'package:tidyr':
## 
##     expand, pack, unpack
```

```
## Registered S3 method overwritten by 'mosaic':
##   method                           from   
##   fortify.SpatialPolygonsDataFrame ggplot2
```

```
##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by t
his.
##
## Note: If you use the Matrix package, be sure to load it BEFORE loading mosaic.
##
## Have you tried the ggformula package for your plots?
```

```
##
## Attaching package: 'mosaic'
```

```
## The following object is masked from 'package:Matrix':
##
##     mean
```

```
## The following object is masked from 'package:caret':
##
##     dotPlot
```

```
## The following object is masked from 'package:plyr':
##
##     count
```

```
## The following objects are masked from 'package:ggvis':
##
##     prop, props
```

```
## The following object is masked from 'package:plotrix':
##
##     rescale
```

```
## The following object is masked from 'package:mlr':
##
##     resample
```

```
## The following objects are masked from 'package:dplyr':
##
##     count, do, tally
```

```
## The following object is masked from 'package:purrr':
##
##     cross
```

```
## The following object is masked from 'package:ggplot2':
##
##     stat
## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median, prop.test,
##     quantile, sd, t.test, var
```

```
## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

```
library(knitr)
```

Phase 2

```
rm(list = ls())
```

Data Prep and exploration

```
bank_data <- read.csv("C:/Users/bhatt/Desktop/Resume And Imp Documents/RMIT University/Semest
er 3/Machine Learning/Phase-2/bank/bank-full.csv",
                        sep = ";", stringsAsFactors = TRUE)
# Data Exploration
bank_data <- subset(bank_data, bank_data$poutcome != "other")

bank_data$education <- plyr::revalue(bank_data$education, c("unknown" = "other"))
bank_data$job <- plyr::revalue(bank_data$job, c("unknown" = "other"))
# Check missing value in Numeric columns
num_var <- select_if(bank_data, is.numeric)
colSums(sapply(num_var, is.na))
```

```
##      age  balance      day duration campaign    pdays previous
##        0        0        0        0        0        0        0
```

```
# Check missing values in Categorical Columns
cat_var <- select_if(bank_data, is.factor)
colSums(sapply(cat_var, is.na))
```

```
##       job   marital education   default   housing      loan   contact     month
##         0         0         0         0         0         0         0         0
##  poutcome         y
##         0         0
```

```
# Summarize the numerical variables
summary(num_var)
```

```
##       age           balance            day           duration     
##  Min.   :18.00   Min.   : -8019   Min.   : 1.00   Min.   :   0.0  
##  1st Qu.:33.00   1st Qu.:    70   1st Qu.: 8.00   1st Qu.: 103.0  
##  Median :39.00   Median :   443   Median :16.00   Median : 180.0  
##  Mean   :40.99   Mean   :  1357   Mean   :15.86   Mean   : 258.3  
##  3rd Qu.:48.00   3rd Qu.:  1417   3rd Qu.:21.00   3rd Qu.: 318.0  
##  Max.   :95.00   Max.   :102127   Max.   :31.00   Max.   :4918.0  
##     campaign          pdays           previous      
##  Min.   : 1.000   Min.   : -1.00   Min.   : 0.0000  
##  1st Qu.: 1.000   1st Qu.: -1.00   1st Qu.: 0.0000  
##  Median : 2.000   Median : -1.00   Median : 0.0000  
##  Mean   : 2.777   Mean   : 32.16   Mean   : 0.4349  
##  3rd Qu.: 3.000   3rd Qu.: -1.00   3rd Qu.: 0.0000  
##  Max.   :63.000   Max.   :871.00   Max.   :55.0000  
```

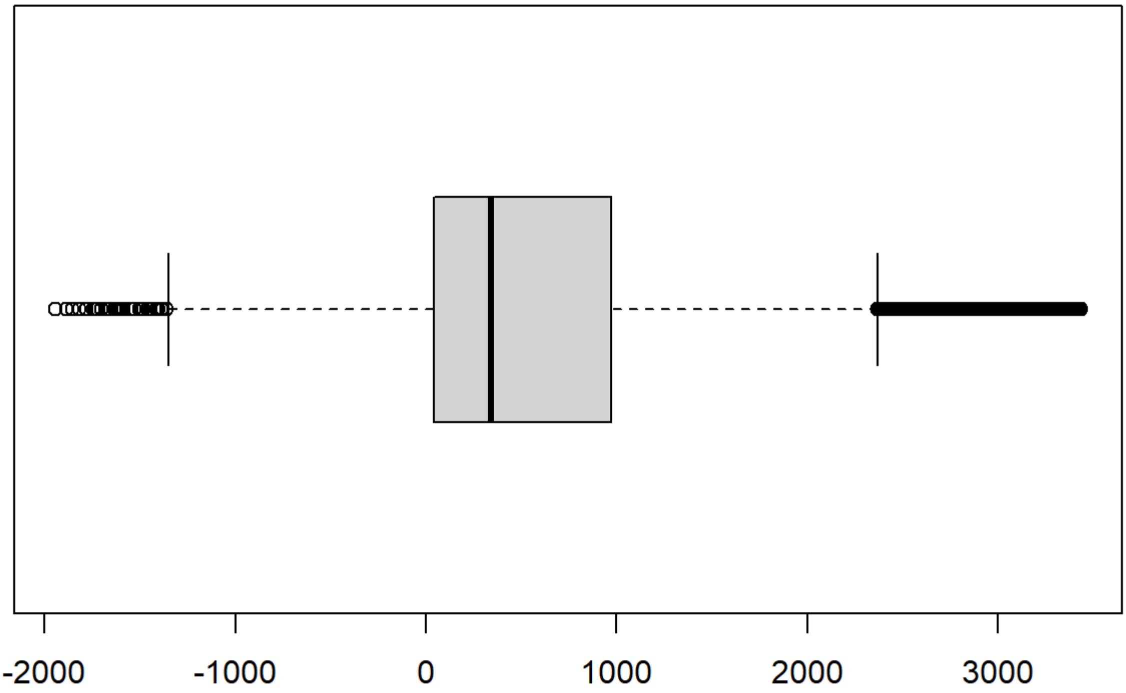
```
# Summarize the categorical variables
summary(cat_var$poutcome)
```

```
## failure   other success unknown 
##    4901       0    1511   36959 
```

```
# Explore the target variable
table(bank_data$y)
```

```
## 
##    no   yes 
## 38389  4982 
```

```
# Visualize the balance to check the outliers and remove them if any
outliers <- boxplot(bank_data$balance, horizontal = TRUE, plot = FALSE)$out
bank_data <- bank_data[-which(bank_data$balance %in% outliers),]
boxplot(bank_data$balance, horizontal = TRUE)
```

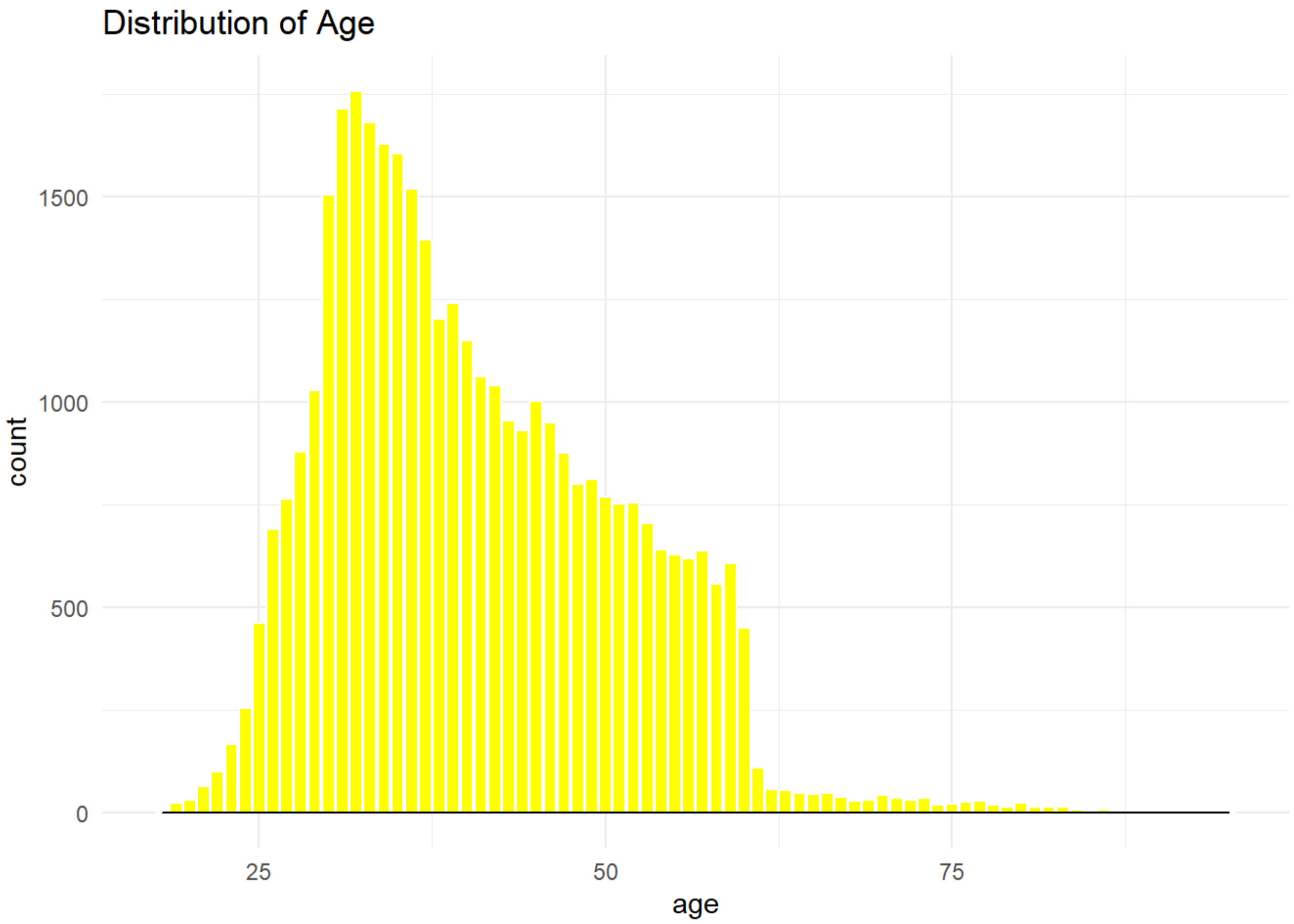


```
# Remove the column contact as it has no impact on target variable y
bank_data$contact <- NULL
# Keep records which has call duration of more than 5 seconds
bank_data <- subset.data.frame(bank_data, bank_data$duration > 5)
# Drop the records for customer with education as other
bank_data <- subset(bank_data, bank_data$education != "other")
cat_var <- select_if(bank_data, is.factor)
summary(cat_var)
```

```
##           job           marital          education     default     housing
##  blue-collar:8157   divorced: 4395   primary  : 5930   no :36429   no :16170
##  management :7586   married :22304   secondary:20267   yes:  753   yes:21012
##  technician :6364   single  :10483   tertiary :10985
##  admin.     :4352                    other    :    0
##  services   :3554
##  retired    :1746
##  (Other)    :5423
##   loan           month         poutcome       y
##  no :30725   may    :11659   failure: 4169   no :33113
##  yes: 6457   jul    : 6143   other  :    0   yes: 4069
##              aug    : 5343   success: 1218
##              jun    : 4340   unknown:31795
##              nov    : 2817
##              apr    : 2274
##              (Other): 4606
```

```
# Rename the y variable as target
names(bank_data)[length(bank_data)] <- "Target"

# Data Exploration
# Distribution of age
p <- ggplot(bank_data, aes(x = age))
p + geom_bar(color = "white",
             fill  = "yellow") + theme_minimal() + labs(title = "Distribution of Age") +
  geom_density()
```



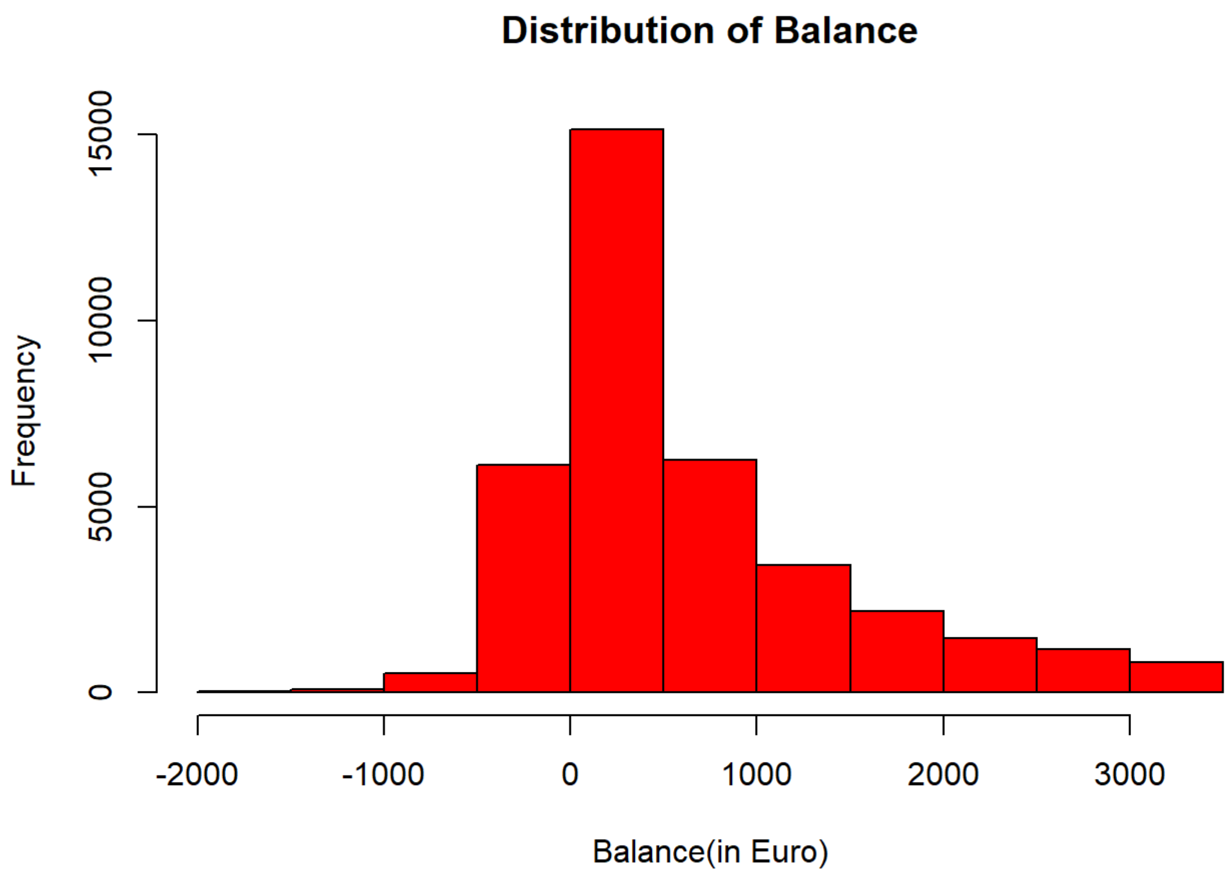
```
# Distribution of Balance
hist(bank_data$balance, fill = "red", col = "red",
     main = "Distribution of Balance",
     xlab = "Balance(in Euro)")
```

```
## Warning in plot.window(xlim, ylim, "", ...): "fill" is not a graphical parameter
```

```
## Warning in title(main = main, sub = sub, xlab = xlab, ylab = ylab, ...): "fill"
## is not a graphical parameter
```

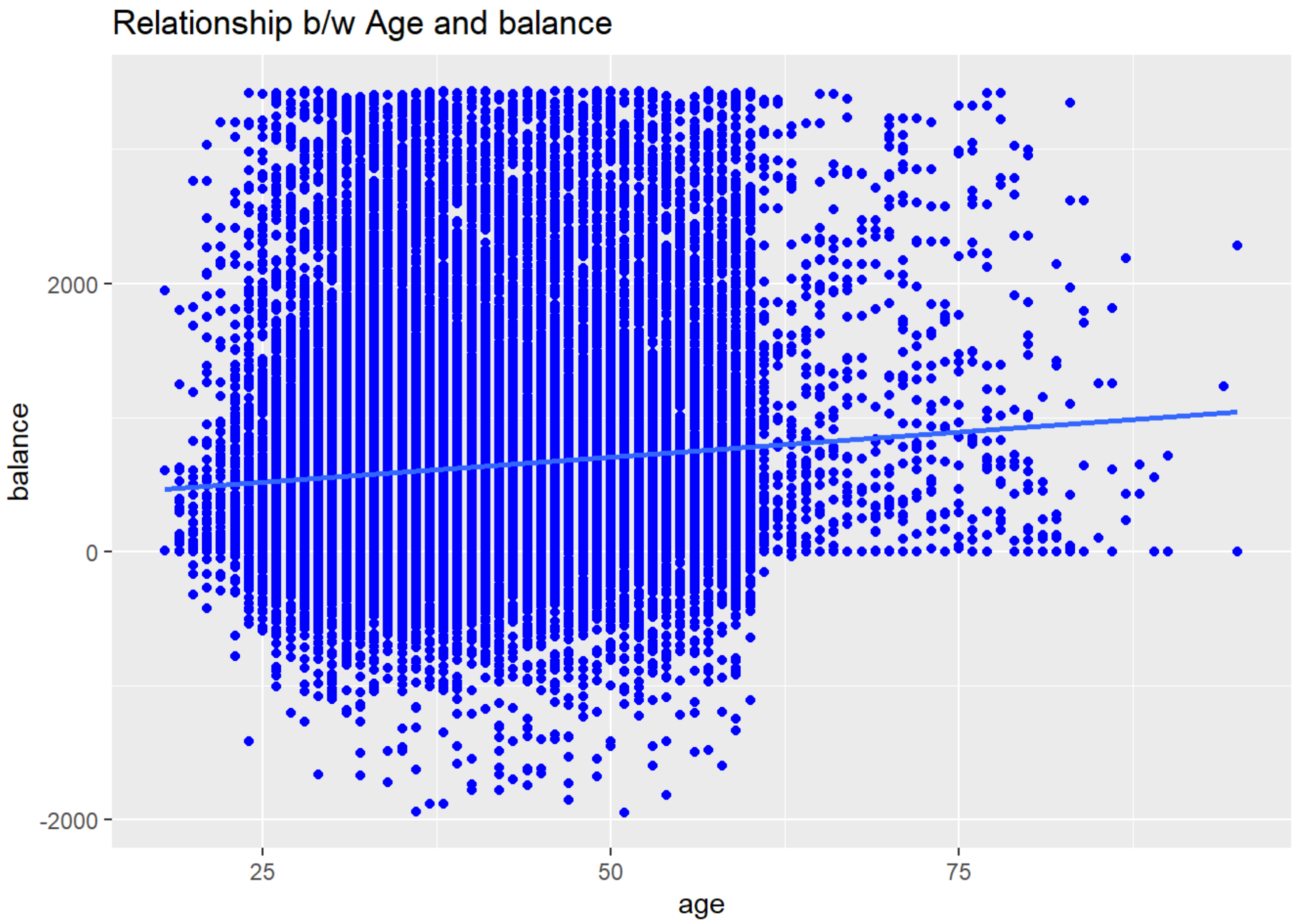
```
## Warning in axis(1, ...): "fill" is not a graphical parameter
```

```
## Warning in axis(2, ...): "fill" is not a graphical parameter
```

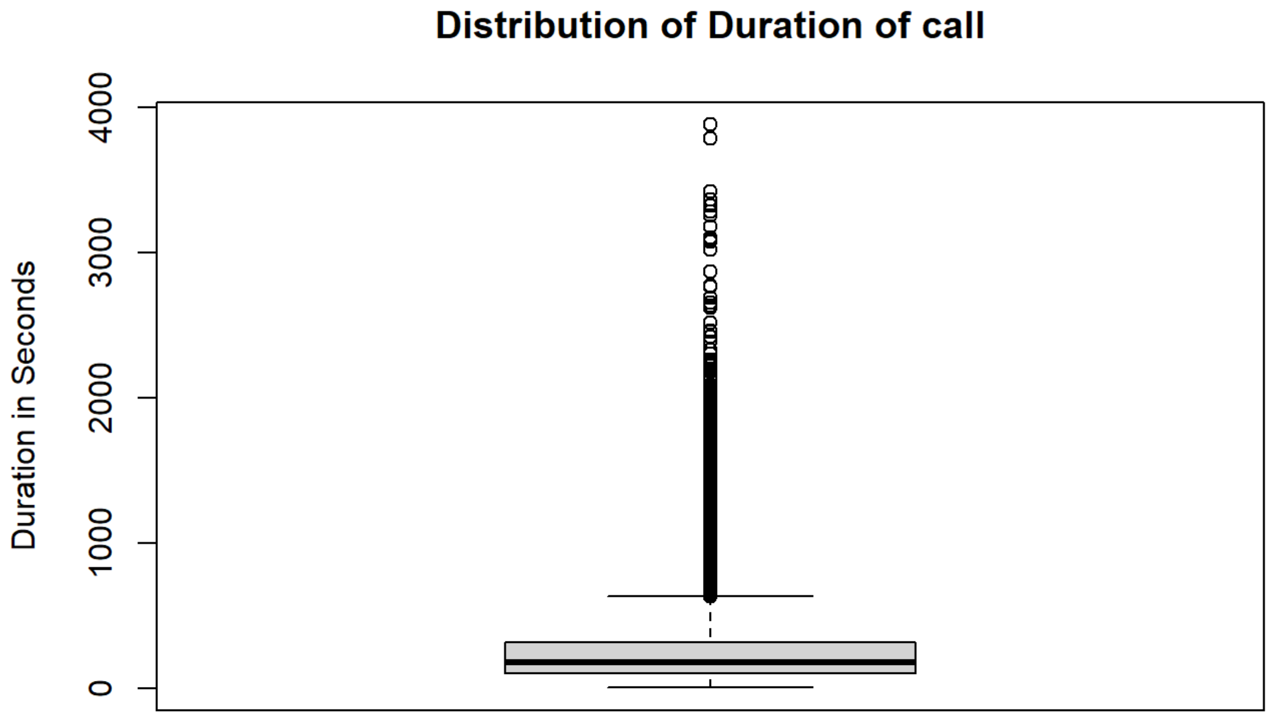


```
# Relationship between age and balance
d <- ggplot(bank_data, aes(x = age, y = balance))
d +  geom_point(color = "blue") + labs(title = "Relationship b/w Age and balance") +
  geom_smooth(method = "lm", se = F)
```

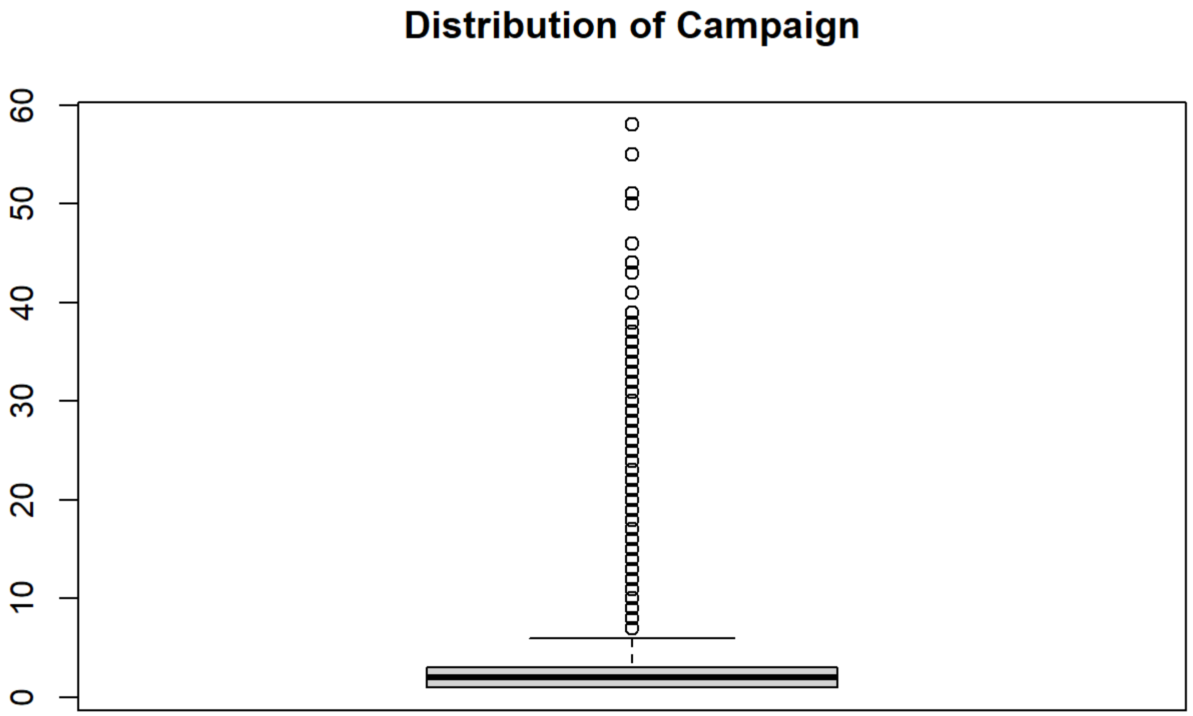
```
## `geom_smooth()` using formula 'y ~ x'
```



```
# Visualization of duration and campaign
boxplot(bank_data$duration, main = "Distribution of Duration of call",
        ylab = "Duration in Seconds")
```



```
boxplot(bank_data$campaign, main = "Distribution of Campaign")
```

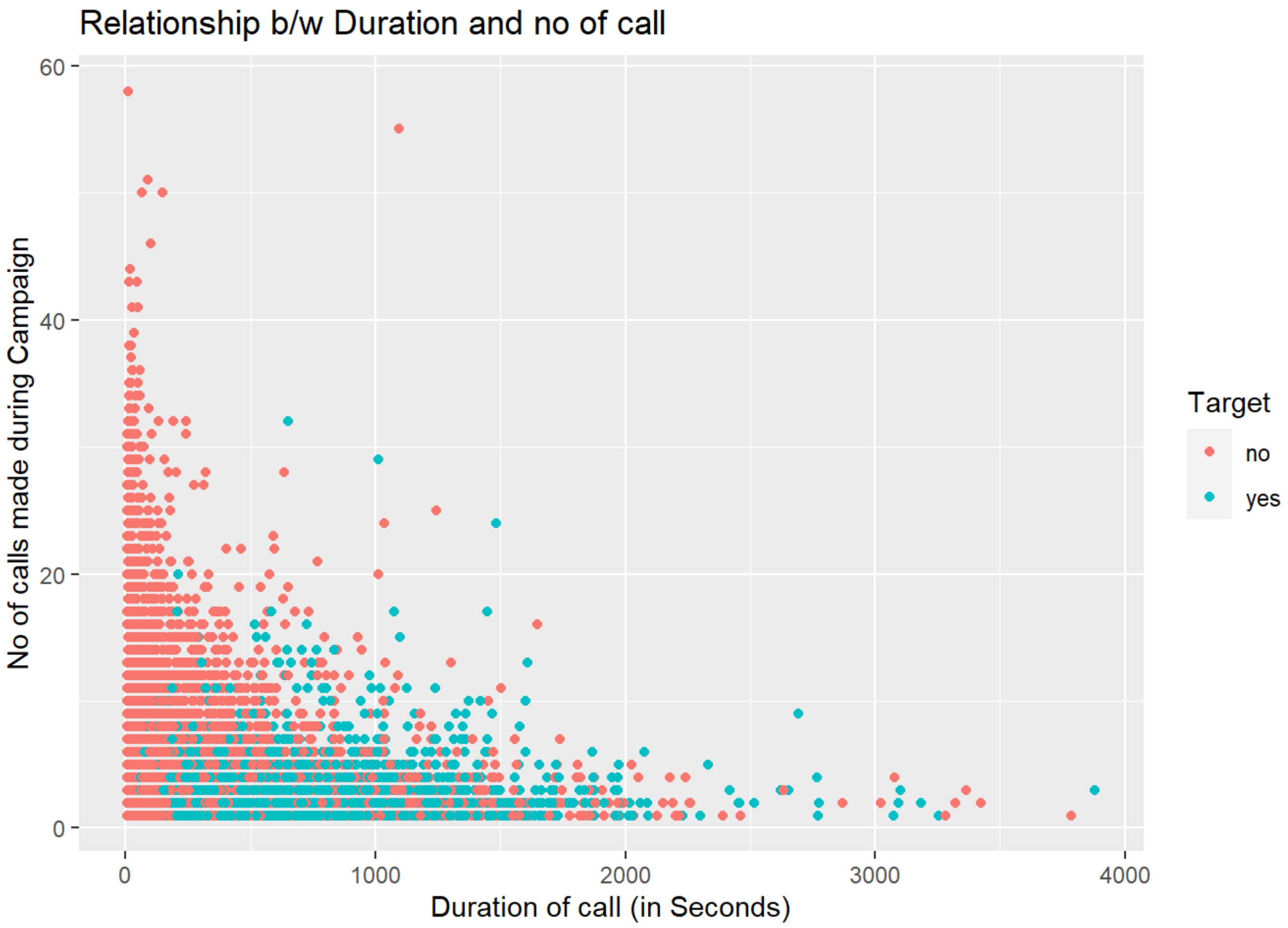


```
# Relationship b/w duration and campaign with response rate

ggplot(bank_data, aes(x = bank_data$duration, y = bank_data$campaign)) +
  geom_point(aes(col = Target)) + labs(title = "Relationship b/w Duration and no of call",
                                       x = "Duration of call (in Seconds)",
                                       y = "No of calls made during Campaign")
```

```
## Warning: Use of `bank_data$duration` is discouraged. Use `duration` instead.
```

```
## Warning: Use of `bank_data$campaign` is discouraged. Use `campaign` instead.
```



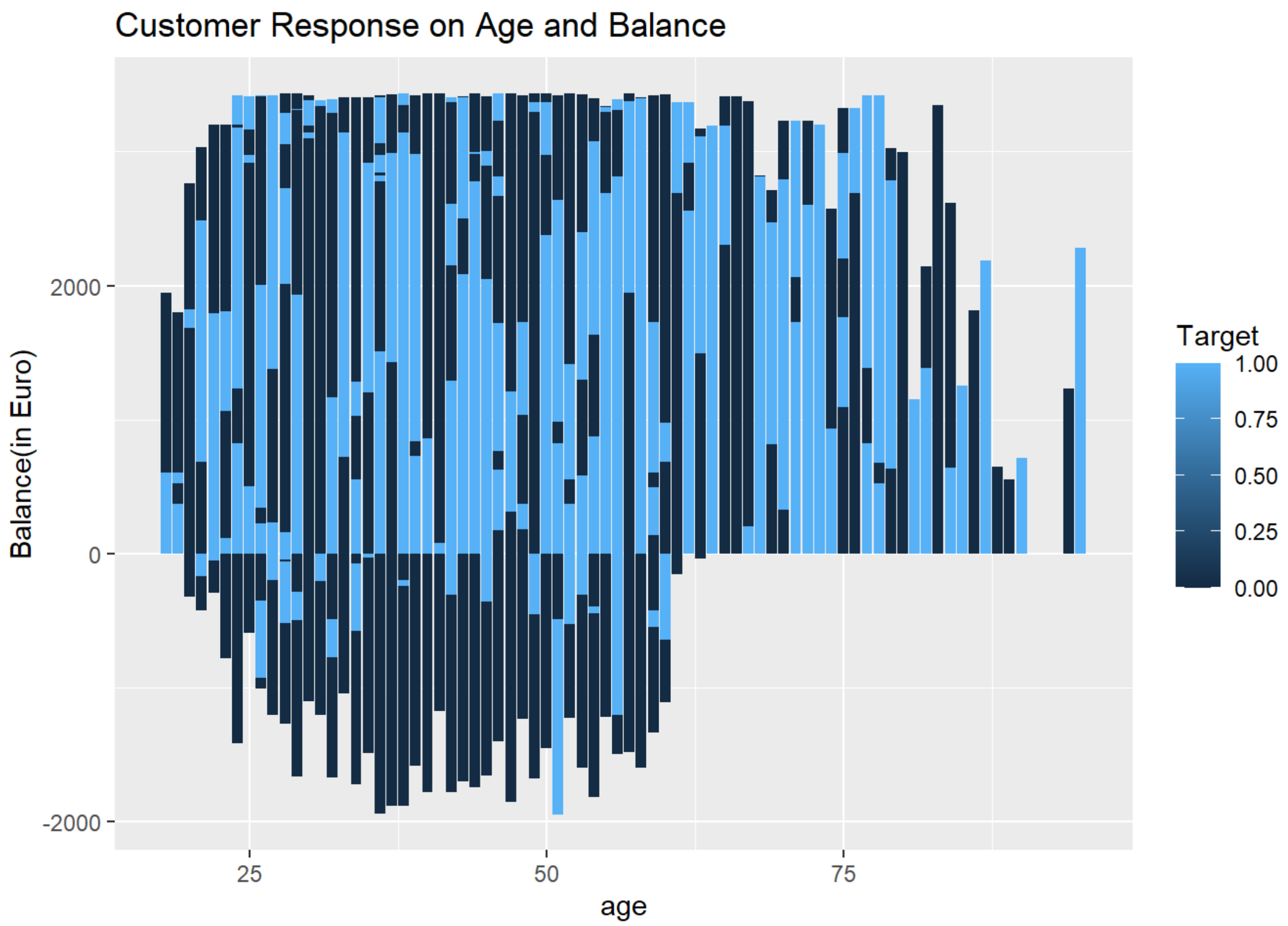
```
bank_data$Target <- ifelse(bank_data$Target == "yes", 1,0)
table(bank_data$job)
```

```
## 
##        admin.   blue-collar  entrepreneur     housemaid    management 
##          4352          8157          1225          1044          7586 
##       retired self-employed      services       student    technician 
##          1746          1303          3554           628          6364 
##    unemployed         other 
##          1086           137
```

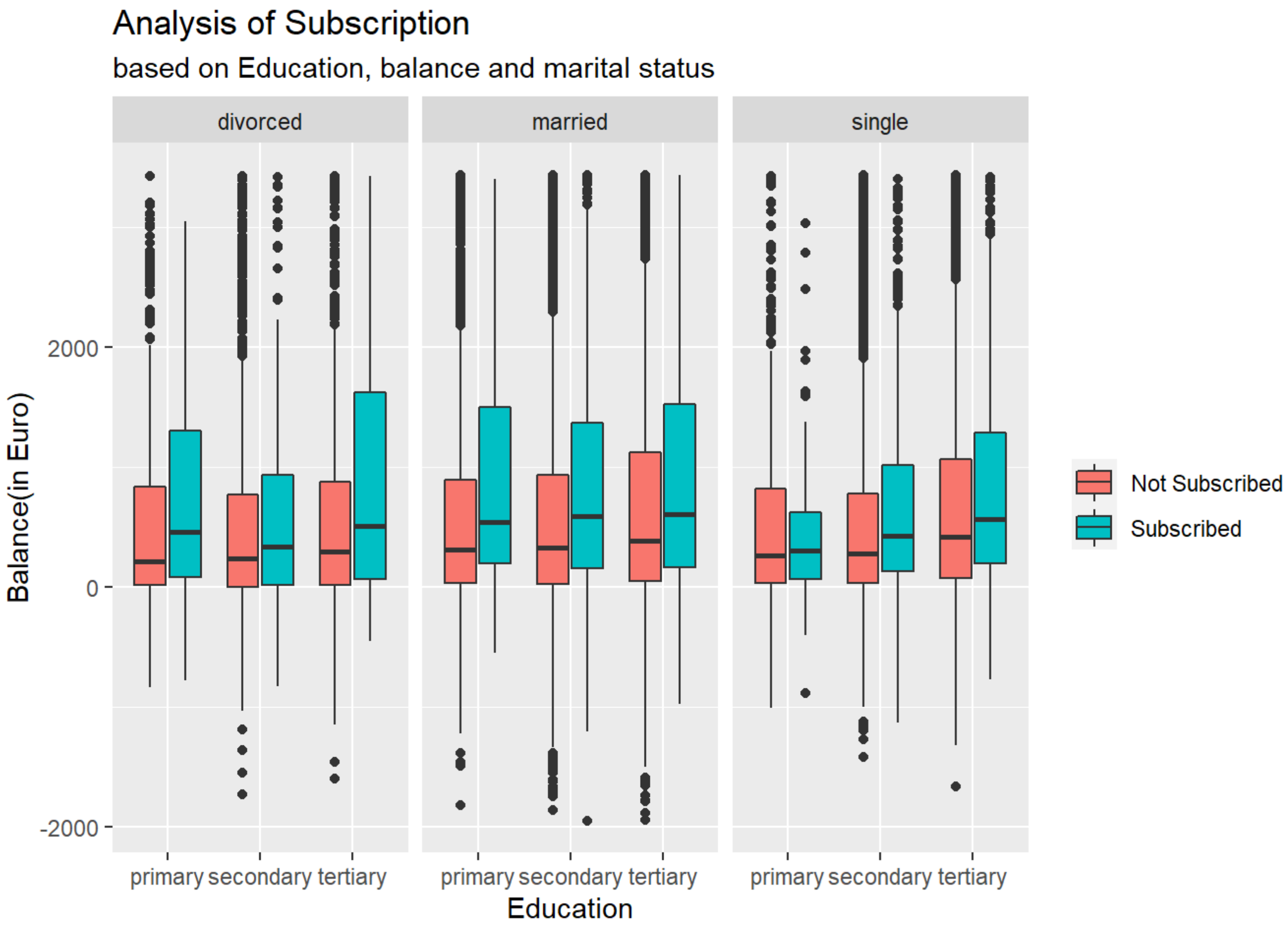
```
bank_data$job <- gsub("admin.","admin",bank_data$job, fixed = TRUE)
bank_data$job <- gsub("blue-collar","blue_collar",bank_data$job, fixed = TRUE)
bank_data$job <- gsub("self-employed","self_employed",bank_data$job, fixed = TRUE)
sub_data <- bank_data[,c("age","balance","duration", "campaign", "pdays","previous", "Target"
)]
bank_data$job <- as.factor(bank_data$job)


# Visualise target variable with respect to different different predictors
ggplot(data = bank_data, aes(x = age, y = bank_data$balance, fill = Target)) +
  geom_bar(stat = "identity", position = position_dodge()) +
  labs(title = "Customer Response on Age and Balance",
       y = "Balance(in Euro)") + scale_color_manual(values = c("#999999","##56B4E9"))
```

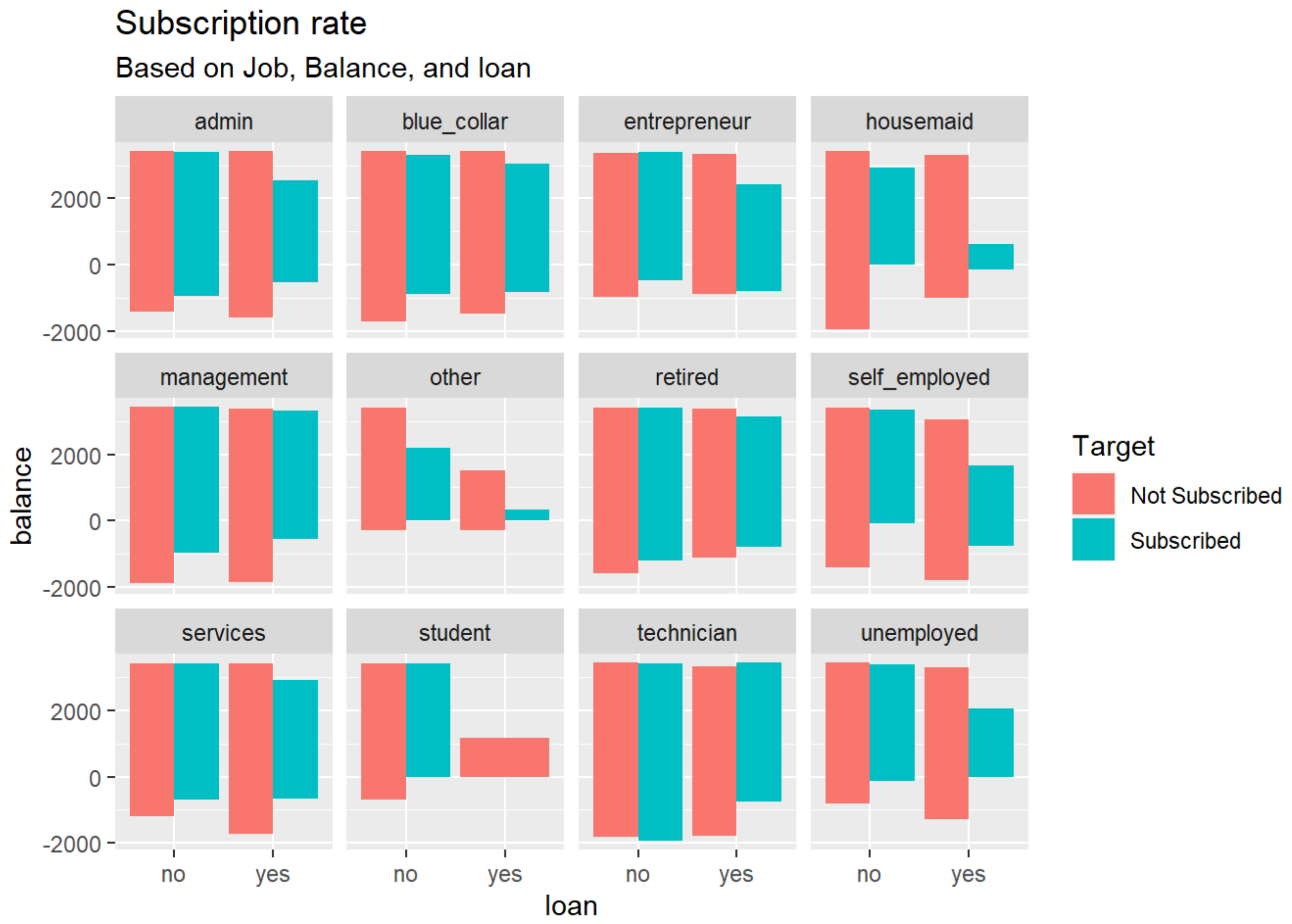
```
## Warning: Use of `bank_data$balance` is discouraged. Use `balance` instead.
```



```
bank_data$Target <- ifelse(bank_data$Target == 1, "Subscribed","Not Subscribed")
e <- ggplot(data = bank_data, aes(x = education, y = balance, fill = Target))
e + geom_boxplot() + labs(y = "Balance(in Euro)",
                          x = "Education",
                          title = "Analysis of Subscription ",
                          subtitle = "based on Education, balance and marital status") +
  theme(legend.title = element_blank()) + facet_wrap( ~ marital)
```



```
f <- ggplot(bank_data, aes(x = loan, y = balance, fill = Target))
f + geom_bar(stat = "identity", position = position_dodge()) +
  labs(title = "Subscription rate",
       subtitle = "Based on Job, Balance, and loan") + facet_wrap(~ job)
```



## Phase 2 codes ### Scaling of Numerical Variable

```
colnames(num_var)
```

```
## [1] "age"       "balance"   "day"       "duration"  "campaign"  "pdays"     "previous"
```

```
y <- bank_data[,c(16)]
bank_data <- bank_data[,c(1:ncol(bank_data)-1)]
bank_data_norm <- normalizeFeatures(bank_data, method = "standardize",
                                    cols = colnames(num_var))
```

One hot encoding of categorical variables

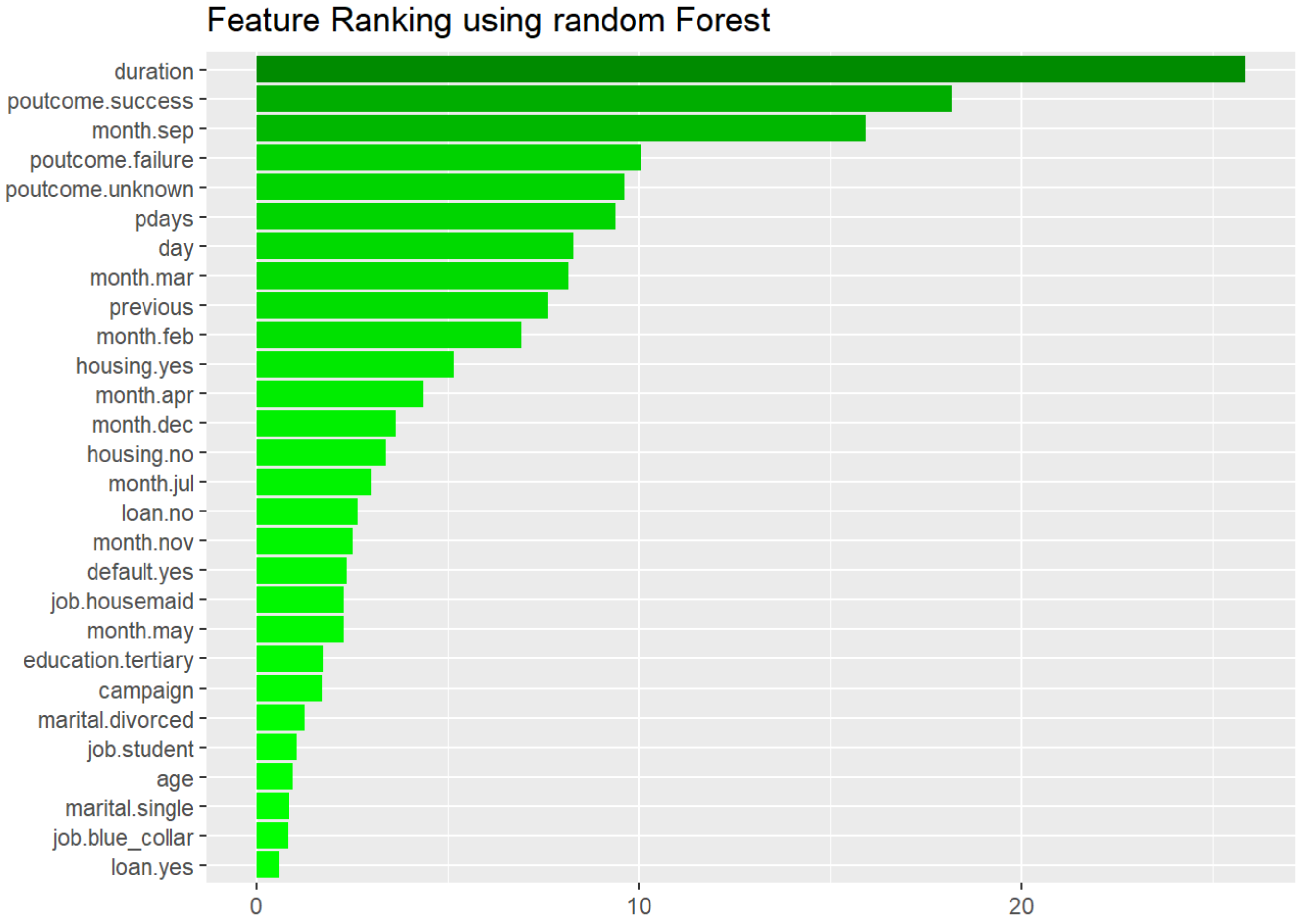
```
bank_data_encode <- dummyVars("~.", data = bank_data_norm)
bank_data_encode <- data.frame(predict(bank_data_encode, newdata = bank_data_norm))
bank_data_encode <- cbind(bank_data_encode, y)
colnames(bank_data_encode)[ncol(bank_data_encode)] <- c("Target")
colnames(bank_data_encode)
```

```
##  [1] "age"                  "job.admin"           "job.blue_collar"    
##  [4] "job.entrepreneur"     "job.housemaid"       "job.management"     
##  [7] "job.other"            "job.retired"         "job.self_employed"  
## [10] "job.services"         "job.student"         "job.technician"     
## [13] "job.unemployed"       "marital.divorced"    "marital.married"    
## [16] "marital.single"       "education.primary"   "education.secondary"
## [19] "education.tertiary"  "education.other"     "default.no"         
## [22] "default.yes"          "balance"             "housing.no"         
## [25] "housing.yes"          "loan.no"             "loan.yes"           
## [28] "day"                  "month.apr"           "month.aug"          
## [31] "month.dec"            "month.feb"           "month.jan"          
## [34] "month.jul"            "month.jun"           "month.mar"          
## [37] "month.may"            "month.nov"           "month.oct"          
## [40] "month.sep"            "duration"            "campaign"           
## [43] "pdays"                "previous"            "poutcome.failure"   
## [46] "poutcome.other"       "poutcome.success"    "poutcome.unknown"   
## [49] "Target"
```

```
bank_data_encode <- bank_data_encode[sample(nrow(bank_data_encode),500),]
```

Feature Selection and Ranking

```
weights <- random.forest.importance(Target ~ ., data = bank_data_encode,
                                    importance.type = 1)
rank <- data.frame(features = rownames(weights),
                   importance = weights$attr_importance)
p1 <- ggplot(data = rank[which(rank$importance > 0.5),],
             aes(x = reorder(features,
                             importance),
                             y = importance,
                             fill = importance)) + coord_flip() +
            geom_bar(stat = "identity", width = 0.25) +
            geom_col(position = "dodge") +
            scale_fill_gradient(low = "green",
                                high = "green4",
                                space = "Lab",
                                guide = FALSE) +
  labs(title = "Feature Ranking using random Forest",
       x = NULL,
       y = NULL)
p1
```



Clearly the most important feature is Duration, follwed by month.mar, day, age and so on.

Balancing dataset

```
x <- data.frame(100*round(table(bank_data_encode$Target)/length(bank_data_encode$Target),2))
x
```

```
##             Var1 Freq
## 1 Not Subscribed   88
## 2     Subscribed   12
```

Since our dataset has 89% of Not subscribed data, which suggests that our datset is highly imbalanced and it will have severe effect on the accuracy of our models. So we will balance our data by using synthetic data generation apporach.

```
library(ROSE)
```

```
## Loaded ROSE 0.0-3
```

```
bank_data_balanced <- ROSE(Target ~ ., data = bank_data_encode, seed = 1)$data
table(bank_data_balanced$Target)
```

```
## 
## Not Subscribed     Subscribed 
##            270            230
```

```
data.frame(100*round(table(bank_data_balanced$Target)/length(bank_data_balanced$Target),2))
```

```
##             Var1 Freq
## 1 Not Subscribed   54
## 2     Subscribed   46
```

Now the data has 50 50 ratio for Not subscribed and subscribed outcome.

Data Splicing

We will split our data in train and test dataset in the ratio of 80-20. i.e 80% of the data will be used to train our models and rest 20% will be used to test the model acurracy.

```
# Split the data into test and train in 80 20 ratio
set.seed(101)
sample <- sample.int(n = nrow(bank_data_balanced),
                     size = floor(0.80 * nrow(bank_data_balanced)), replace = F)
train <- bank_data_balanced[sample,]
test <- bank_data_balanced[-sample,]
```

Classification task

```
task_train <- makeClassifTask(data = train, target = "Target")
task_test <- makeClassifTask(data = test, target = "Target")
```

Check task

```
task_train
```

```
## Supervised task: train
## Type: classif
## Target: Target
## Observations: 400
## Features:
##    numerics     factors     ordered functionals
##          48           0           0           0
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Has coordinates: FALSE
## Classes: 2
## Not Subscribed     Subscribed
##            213            187
## Positive class: Not Subscribed
```

task_train gives us a picture of training data but with a problem of considering NOt Subscribed as Positive class, so let's convert the positive class to subscribed.

```
task_train <- makeClassifTask(data = train,target = "Target",positive = "Subscribed")
task_train
```

```
## Supervised task: train
## Type: classif
## Target: Target
## Observations: 400
## Features:
##    numerics     factors     ordered functionals 
##          48           0           0           0 
## Missings: FALSE
## Has weights: FALSE
## Has blocking: FALSE
## Has coordinates: FALSE
## Classes: 2
## Not Subscribed     Subscribed 
##            213            187 
## Positive class: Subscribed
```

Set up 10 fold cross validation

```
# 10 fold cross Cross Validation
cv <- makeResampleDesc("CV", iters = 5L)
```

-- --

---- ## Decision Tree:

```
getParamSet("classif.rpart")
```

```
##                    Type len  Def   Constr Req Tunable Trafo
## minsplit        integer   -   20 1 to Inf   -    TRUE     -
## minbucket       integer   -    - 1 to Inf   -    TRUE     -
## cp              numeric   - 0.01   0 to 1   -    TRUE     -
## maxcompete      integer   -    4 0 to Inf   -    TRUE     -
## maxsurrogate    integer   -    5 0 to Inf   -    TRUE     -
## usesurrogate   discrete   -    2    0,1,2   -    TRUE     -
## surrogatestyle discrete   -    0      0,1   -    TRUE     -
## maxdepth        integer   -   30  1 to 30   -    TRUE     -
## xval            integer   -   10 0 to Inf   -   FALSE     -
## parms           untyped   -    -        -   -    TRUE     -
```

```
# Learner Tree
bank_dt.learn <- makeLearner("classif.rpart", predict.type = "prob", fix.factors.prediction = 
TRUE)
bank_dt.learn
```

```
## Learner classif.rpart from package rpart
## Type: classif
## Name: Decision Tree; Short name: rpart
## Class: classif.rpart
## Properties: twoclass,multiclass,missings,numerics,factors,ordered,prob,weights,featimp
## Predict-Type: prob
## Hyperparameters: xval=0
```

```
dt_control <- makeTuneControlGrid()
```

"MinSplit: number of observation in a node for split to take place" "Minbucket: min number of observation" cp : complexity parametr should be set low to avoid overfitting **-- -----** **--** ## K- Nearest Neighbour(KNN)

```
getParamSet("classif.kknn")
```

```
##               Type len     Def                                  Constr Req
## k          integer   -       7                                1 to Inf   -
## distance   numeric   -       2                                0 to Inf   -
## kernel   discrete   - optimal rectangular,triangular,epanechnikov,b...   -
## scale     logical   -    TRUE                                       -   -
##          Tunable Trafo
## k           TRUE     -
## distance    TRUE     -
## kernel      TRUE     -
## scale       TRUE     -
```

```
bank_knn.learn <- makeLearner("classif.kknn", predict.type = "prob", fix.factors.prediction = 
TRUE)
bank_knn.learn
```

```
## Learner classif.kknn from package kknn
## Type: classif
## Name: k-Nearest Neighbor; Short name: kknn
## Class: classif.kknn
## Properties: twoclass,multiclass,numerics,factors,prob
## Predict-Type: prob
## Hyperparameters:
```

-- **-- ----** ## Random forest

```
getParamSet("classif.randomForest")
```

```
##                      Type  len   Def   Constr Req Tunable Trafo
## ntree             integer    -   500 1 to Inf   -    TRUE     -
## mtry              integer    -     - 1 to Inf   -    TRUE     -
## replace           logical    -  TRUE        -   -    TRUE     -
## classwt     numericvector <NA>     - 0 to Inf   -    TRUE     -
## cutoff      numericvector <NA>     -   0 to 1   -    TRUE     -
## strata            untyped    -     -        -   -   FALSE     -
## sampsize    integervector <NA>     - 1 to Inf   -    TRUE     -
## nodesize          integer    -     1 1 to Inf   -    TRUE     -
## maxnodes          integer    -     - 1 to Inf   -    TRUE     -
## importance        logical    - FALSE        -   -    TRUE     -
## localImp          logical    - FALSE        -   -    TRUE     -
## proximity         logical    - FALSE        -   -   FALSE     -
## oob.prox          logical    -     -        -   Y   FALSE     -
## norm.votes        logical    -  TRUE        -   -   FALSE     -
## do.trace          logical    - FALSE        -   -   FALSE     -
## keep.forest       logical    -  TRUE        -   -   FALSE     -
## keep.inbag        logical    - FALSE        -   -   FALSE     -
```

```
bank_rf.learn <- makeLearner("classif.randomForest", predict.type = "prob", fix.factors.predi
ction = TRUE)
bank_rf.learn$par.vals <- list(importance = TRUE)
bank_rf.learn
```

```
## Learner classif.randomForest from package randomForest
## Type: classif
## Name: Random Forest; Short name: rf
## Class: classif.randomForest
## Properties: twoclass,multiclass,numerics,factors,ordered,prob,class.weights,oobpreds,feati
mp
## Predict-Type: prob
## Hyperparameters: importance=TRUE
```

-- --
---- ## Decision Tree

```
bank_dt.tree  <- mlr::train(bank_dt.learn, task_train)
bank_dt.tree
```

```
## Model for learner.id=classif.rpart; learner.class=classif.rpart
## Trained on: task.id = train; obs = 400; features = 48
## Hyperparameters: xval=0
```

KNN Model Model

```
bank_mod.knn <- mlr::train(bank_knn.learn, task_train)
bank_mod.knn
```

```
## Model for learner.id=classif.kknn; learner.class=classif.kknn
## Trained on: task.id = train; obs = 400; features = 48
## Hyperparameters:
```

Random Forest

```
bank.mod.rf <- mlr::train(bank_rf.learn, task_train)
bank.mod.rf
```

```
## Model for learner.id=classif.randomForest; learner.class=classif.randomForest
## Trained on: task.id = train; obs = 400; features = 48
## Hyperparameters: importance=TRUE
```

5 fold cross validation

```
checks <- list(mmce, tpr, fnr, fpr)
bench <- benchmark(learners = list(bank_dt.learn,
                                   bank_rf.learn,
                                   bank_knn.learn), task_train, cv, checks)
```

```
## Task: train, Learner: classif.rpart
```

```
## Resampling: cross-validation
```

```
## Measures:             mmce      tpr       fnr       fpr
```

```
## [Resample] iter 1:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
## [Resample] iter 2:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
## [Resample] iter 3:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
## [Resample] iter 4:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
## [Resample] iter 5:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
##
```

```
## Aggregated Result: mmce.test.mean=0.0000000,tpr.test.mean=1.0000000,fnr.test.mean=0.000000
0,fpr.test.mean=0.0000000
```

```
##
```

```
## Task: train, Learner: classif.randomForest
```

```
## Resampling: cross-validation
```

```
## Measures:             mmce      tpr       fnr       fpr
## [Resample] iter 1:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
## [Resample] iter 2:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
## [Resample] iter 3:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
## [Resample] iter 4:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
## [Resample] iter 5:    0.0000000 1.0000000 0.0000000 0.0000000
```

```
##
```

```
## Aggregated Result: mmce.test.mean=0.0000000,tpr.test.mean=1.0000000,fnr.test.mean=0.0000000,fpr.test.mean=0.0000000
```

```
##
```

```
## Task: train, Learner: classif.kknn
```

```
## Resampling: cross-validation
```

```
## Measures:             mmce      tpr       fnr       fpr
```

```
## [Resample] iter 1:    0.2000000 0.7560976 0.2439024 0.1538462
```

```
## [Resample] iter 2:    0.2000000 0.7142857 0.2857143 0.1333333
```

```
## [Resample] iter 3:    0.0875000 0.8437500 0.1562500 0.0416667
```

```
## [Resample] iter 4:    0.1625000 0.7941176 0.2058824 0.1304348
```

```
## [Resample] iter 5:    0.1250000 0.8666667 0.1333333 0.1142857
```

```
##
```

```
## Aggregated Result: mmce.test.mean=0.1550000,tpr.test.mean=0.7949835,fnr.test.mean=0.2050165,fpr.test.mean=0.1147133
```

```
##
```

```
getBMRAggrPerformances(bench)
```

```
## $train
## $train$classif.rpart
## mmce.test.mean  tpr.test.mean  fnr.test.mean  fpr.test.mean 
##              0              1              0              0 
## 
## $train$classif.randomForest
## mmce.test.mean  tpr.test.mean  fnr.test.mean  fpr.test.mean 
##              0              1              0              0 
## 
## $train$classif.kknn
## mmce.test.mean  tpr.test.mean  fnr.test.mean  fpr.test.mean 
##      0.1550000      0.7949835      0.2050165      0.1147133
```

Decision tree

```
dt_par <- makeParamSet(
  makeDiscreteParam("cp", values = seq(0,0.002,0.0005)),
  makeIntegerParam("minsplit", lower = 2, upper = 10),
  makeDiscreteParam("maxdepth", values = c(10,20,30))
)
ctrl <- makeTuneControlRandom(maxit = 5)
bank_dt_tune <- makeTuneWrapper(bank_dt.learn, cv, mmce, dt_par, ctrl)
print(bank_dt_tune)
```

```
## Learner classif.rpart.tuned from package rpart
## Type: classif
## Name: ; Short name: 
## Class: TuneWrapper
## Properties: numerics,factors,ordered,missings,weights,prob,twoclass,multiclass,featimp
## Predict-Type: prob
## Hyperparameters: xval=0
```

```
bank_dt.trn <- mlr::train(bank_dt_tune, task_train)
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=0; minsplit=6; maxdepth=20
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
## [Tune-x] 2: cp=0; minsplit=8; maxdepth=30
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: cp=0.002; minsplit=9; maxdepth=30
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: cp=5e-04; minsplit=4; maxdepth=30
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: cp=0; minsplit=4; maxdepth=20
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: cp=0; minsplit=8; maxdepth=30 : mmce.test.mean=0.0000000
```

```
bank_dt.trn
```

```
## Model for learner.id=classif.rpart.tuned; learner.class=TuneWrapper
## Trained on: task.id = train; obs = 400; features = 48
## Hyperparameters: xval=0
```

HYper Fine tuning ## KNN

```
knn_par <- makeParamSet(
  makeIntegerParam("k", lower = 2, upper = 10),
  makeDiscreteParam("kernel",values = c("rectangular", "optimal"))
  )
ctrl <- makeTuneControlRandom(maxit = 5)
bank.knn.tune <- makeTuneWrapper(bank_knn.learn, cv, mmce, knn_par, ctrl)

print(bank.knn.tune)
```

```
## Learner classif.kknn.tuned from package kknn
## Type: classif
## Name: ; Short name:
## Class: TuneWrapper
## Properties: numerics,factors,prob,twoclass,multiclass
## Predict-Type: prob
## Hyperparameters:
```

```
bank.knn.mod <- mlr::train(bank.knn.tune, task_train)
```

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

```
##            Type len Def              Constr Req Tunable Trafo
## k       integer   -   -             2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=3; kernel=optimal
```

```
## [Tune-y] 1: mmce.test.mean=0.1475000; time: 0.0 min
```

```
## [Tune-x] 2: k=2; kernel=optimal
```

```
## [Tune-y] 2: mmce.test.mean=0.1475000; time: 0.0 min
```

```
## [Tune-x] 3: k=7; kernel=rectangular
```

```
## [Tune-y] 3: mmce.test.mean=0.2150000; time: 0.0 min
```

```
## [Tune-x] 4: k=8; kernel=optimal
```

```
## [Tune-y] 4: mmce.test.mean=0.1600000; time: 0.0 min
```

```
## [Tune-x] 5: k=2; kernel=optimal
```

```
## [Tune-y] 5: mmce.test.mean=0.1475000; time: 0.0 min
```

```
## [Tune] Result: k=2; kernel=optimal : mmce.test.mean=0.1475000
```

```
bank.knn.mod
```

```
## Model for learner.id=classif.kknn.tuned; learner.class=TuneWrapper
## Trained on: task.id = train; obs = 400; features = 48
## Hyperparameters:
```

Random Forest

```
rf_par <- makeParamSet(
  makeDiscreteParam("ntree", values = c(100,200, 300, 400, 500))
)
bank_rf_tune <- makeTuneWrapper(bank_rf.learn, cv, mmce, rf_par,ctrl)
print(bank_rf_tune)
```

```
## Learner classif.randomForest.tuned from package randomForest
## Type: classif
## Name: ; Short name:
## Class: TuneWrapper
## Properties: numerics,factors,ordered,prob,twoclass,multiclass,class.weights,featimp,oobpre
ds
## Predict-Type: prob
## Hyperparameters: importance=TRUE
```

```
bank.rf.mod.t <- mlr::train(bank_rf_tune, task_train)
```

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

```
##           Type len Def              Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=300
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=300
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=500
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=300
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=200
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=500 : mmce.test.mean=0.0000000
```

```
bank.rf.mod.t
```

```
## Model for learner.id=classif.randomForest.tuned; learner.class=TuneWrapper
## Trained on: task.id = train; obs = 400; features = 48
## Hyperparameters: importance=TRUE
```

Benchmarking

```
bench1 <- benchmark(tasks = task_train,
                    learners = list(bank_dt_tune,bank.knn.tune,bank_rf_tune))
```

```
## Task: train, Learner: classif.rpart.tuned
```

```
## Resampling: cross-validation
```

```
## Measures:             mmce
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=0.001; minsplit=5; maxdepth=30
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: cp=0.001; minsplit=4; maxdepth=30
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: cp=0.0015; minsplit=10; maxdepth=20
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: cp=0; minsplit=4; maxdepth=30
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: cp=0; minsplit=9; maxdepth=20
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: cp=0.0015; minsplit=10; maxdepth=20 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 1:    0.0000000
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=5e-04; minsplit=2; maxdepth=10
```

```
## [Tune-y] 1: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 2: cp=5e-04; minsplit=8; maxdepth=30
```

```
## [Tune-y] 2: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 3: cp=0.002; minsplit=7; maxdepth=20
```

```
## [Tune-y] 3: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 4: cp=0.001; minsplit=2; maxdepth=20
```

```
## [Tune-y] 4: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 5: cp=0.002; minsplit=8; maxdepth=10
```

```
## [Tune-y] 5: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune] Result: cp=0.002; minsplit=7; maxdepth=20 : mmce.test.mean=0.0027778
```

```
## [Resample] iter 2:    0.0000000
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=0.001; minsplit=4; maxdepth=10
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: cp=0.002; minsplit=7; maxdepth=30
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: cp=0.002; minsplit=2; maxdepth=10
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: cp=0.002; minsplit=10; maxdepth=20
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: cp=0.0015; minsplit=5; maxdepth=30
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: cp=0.002; minsplit=2; maxdepth=10 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 3:    0.0000000
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

```
## Imputation value: 1
## [Tune-x] 1: cp=5e-04; minsplit=8; maxdepth=20
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: cp=0; minsplit=7; maxdepth=20
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: cp=0.002; minsplit=7; maxdepth=10
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: cp=0; minsplit=7; maxdepth=20
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: cp=0.001; minsplit=4; maxdepth=10
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: cp=0.001; minsplit=4; maxdepth=10 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 4:    0.0000000
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=0; minsplit=10; maxdepth=30
```

```
## [Tune-y] 1: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 2: cp=5e-04; minsplit=9; maxdepth=10
```

```
## [Tune-y] 2: mmce.test.mean=0.0027778; time: 0.0 min
## [Tune-x] 3: cp=0.0015; minsplit=7; maxdepth=30
```

```
## [Tune-y] 3: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 4: cp=0.0015; minsplit=4; maxdepth=20
```

```
## [Tune-y] 4: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 5: cp=5e-04; minsplit=9; maxdepth=20
```

```
## [Tune-y] 5: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune] Result: cp=0; minsplit=10; maxdepth=30 : mmce.test.mean=0.0027778
```

```
## [Resample] iter 5:    0.0000000
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=5e-04; minsplit=6; maxdepth=10
```

```
## [Tune-y] 1: mmce.test.mean=0.0055556; time: 0.0 min
```

```
## [Tune-x] 2: cp=0.0015; minsplit=4; maxdepth=20
```

```
## [Tune-y] 2: mmce.test.mean=0.0055556; time: 0.0 min
```

```
## [Tune-x] 3: cp=0.0015; minsplit=10; maxdepth=10
```

```
## [Tune-y] 3: mmce.test.mean=0.0055556; time: 0.0 min
```

```
## [Tune-x] 4: cp=0; minsplit=5; maxdepth=30
```

```
## [Tune-y] 4: mmce.test.mean=0.0055556; time: 0.0 min
## [Tune-x] 5: cp=5e-04; minsplit=2; maxdepth=20
```

```
## [Tune-y] 5: mmce.test.mean=0.0055556; time: 0.0 min
```

```
## [Tune] Result: cp=0.0015; minsplit=4; maxdepth=20 : mmce.test.mean=0.0055556
```

```
## [Resample] iter 6:    0.0000000
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=0.001; minsplit=10; maxdepth=30
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: cp=0; minsplit=8; maxdepth=30
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: cp=0.0015; minsplit=10; maxdepth=20
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: cp=0.0015; minsplit=7; maxdepth=10
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: cp=0; minsplit=9; maxdepth=30
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: cp=0; minsplit=8; maxdepth=30 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 7:    0.0000000
## [Tune] Started tuning learner classif.rpart for parameter set:
```

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=0; minsplit=2; maxdepth=10
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: cp=0.0015; minsplit=3; maxdepth=10
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: cp=5e-04; minsplit=3; maxdepth=30
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: cp=0.001; minsplit=3; maxdepth=30
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: cp=0.0015; minsplit=9; maxdepth=30
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: cp=0.0015; minsplit=9; maxdepth=30 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 8:    0.0000000
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
## Imputation value: 1
```

```
## [Tune-x] 1: cp=5e-04; minsplit=6; maxdepth=10
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: cp=0.0015; minsplit=6; maxdepth=30
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: cp=0; minsplit=9; maxdepth=10
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: cp=0.0015; minsplit=4; maxdepth=10
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: cp=5e-04; minsplit=2; maxdepth=10
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: cp=0.0015; minsplit=6; maxdepth=30 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 9:    0.0000000
```

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

```
##              Type len Def                     Constr Req Tunable Trafo
## cp       discrete   -   - 0,5e-04,0.001,0.0015,0.002   -    TRUE     -
## minsplit  integer   -   -                    2 to 10   -    TRUE     -
## maxdepth discrete   -   -                   10,20,30   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: cp=0; minsplit=7; maxdepth=10
```

```
## [Tune-y] 1: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 2: cp=5e-04; minsplit=2; maxdepth=30
## [Tune-y] 2: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 3: cp=5e-04; minsplit=10; maxdepth=30
```

```
## [Tune-y] 3: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 4: cp=0.002; minsplit=9; maxdepth=30
```

```
## [Tune-y] 4: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune-x] 5: cp=0; minsplit=3; maxdepth=30
```

```
## [Tune-y] 5: mmce.test.mean=0.0027778; time: 0.0 min
```

```
## [Tune] Result: cp=5e-04; minsplit=2; maxdepth=30 : mmce.test.mean=0.0027778
```

```
## [Resample] iter 10:   0.0000000
```

```
##
```

```
## Aggregated Result: mmce.test.mean=0.0000000
```

```
##
```

```
## Task: train, Learner: classif.kknn.tuned
```

```
## Resampling: cross-validation
```

```
## Measures:             mmce
```

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

```
##            Type len Def              Constr Req Tunable Trafo
## k       integer   -   -             2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=2; kernel=optimal
## [Tune-y] 1: mmce.test.mean=0.1583333; time: 0.0 min
```

```
## [Tune-x] 2: k=10; kernel=rectangular
```

```
## [Tune-y] 2: mmce.test.mean=0.2361111; time: 0.0 min
```

```
## [Tune-x] 3: k=7; kernel=optimal
```

```
## [Tune-y] 3: mmce.test.mean=0.1638889; time: 0.0 min
```

```
## [Tune-x] 4: k=4; kernel=optimal
```

```
## [Tune-y] 4: mmce.test.mean=0.1583333; time: 0.0 min
```

```
## [Tune-x] 5: k=7; kernel=optimal
```

```
## [Tune-y] 5: mmce.test.mean=0.1638889; time: 0.0 min
```

```
## [Tune] Result: k=4; kernel=optimal : mmce.test.mean=0.1583333
```

```
## [Resample] iter 1:    0.2000000
```

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

```
##            Type len Def              Constr Req Tunable Trafo
## k       integer   -   -             2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=3; kernel=rectangular
```

```
## [Tune-y] 1: mmce.test.mean=0.1638889; time: 0.0 min
```

```
## [Tune-x] 2: k=6; kernel=optimal
```

```
## [Tune-y] 2: mmce.test.mean=0.1666667; time: 0.0 min
```

```
## [Tune-x] 3: k=9; kernel=optimal
## [Tune-y] 3: mmce.test.mean=0.1611111; time: 0.0 min
```

```
## [Tune-x] 4: k=8; kernel=rectangular
```

```
## [Tune-y] 4: mmce.test.mean=0.2416667; time: 0.0 min
```

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

```
## [Tune-y] 5: mmce.test.mean=0.1666667; time: 0.0 min
```

```
## [Tune] Result: k=9; kernel=optimal : mmce.test.mean=0.1611111
```

```
## [Resample] iter 2:    0.1250000
```

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

```
##            Type len Def               Constr Req Tunable Trafo
## k       integer   -   -              2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=5; kernel=rectangular
```

```
## [Tune-y] 1: mmce.test.mean=0.2027778; time: 0.0 min
```

```
## [Tune-x] 2: k=5; kernel=rectangular
```

```
## [Tune-y] 2: mmce.test.mean=0.2027778; time: 0.0 min
```

```
## [Tune-x] 3: k=5; kernel=rectangular
```

```
## [Tune-y] 3: mmce.test.mean=0.2027778; time: 0.0 min
```

```
## [Tune-x] 4: k=4; kernel=rectangular
```

```
## [Tune-y] 4: mmce.test.mean=0.2055556; time: 0.0 min
```

```
## [Tune-x] 5: k=10; kernel=optimal
## [Tune-y] 5: mmce.test.mean=0.1583333; time: 0.0 min
```

```
## [Tune] Result: k=10; kernel=optimal : mmce.test.mean=0.1583333
```

```
## [Resample] iter 3:    0.1250000
```

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

```
##            Type len Def               Constr Req Tunable Trafo
## k       integer   -   -              2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=5; kernel=rectangular
```

```
## [Tune-y] 1: mmce.test.mean=0.2000000; time: 0.0 min
```

```
## [Tune-x] 2: k=7; kernel=rectangular
```

```
## [Tune-y] 2: mmce.test.mean=0.2111111; time: 0.0 min
```

```
## [Tune-x] 3: k=5; kernel=optimal
```

```
## [Tune-y] 3: mmce.test.mean=0.1833333; time: 0.0 min
```

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

```
## [Tune-y] 4: mmce.test.mean=0.1833333; time: 0.0 min
```

```
## [Tune-x] 5: k=5; kernel=rectangular
```

```
## [Tune-y] 5: mmce.test.mean=0.2000000; time: 0.0 min
```

```
## [Tune] Result: k=5; kernel=optimal : mmce.test.mean=0.1833333
```

```
## [Resample] iter 4:    0.1250000
```

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

```
##            Type len Def              Constr Req Tunable Trafo
## k       integer   -   -             2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

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

```
## [Tune-y] 1: mmce.test.mean=0.2138889; time: 0.0 min
```

```
## [Tune-x] 2: k=7; kernel=optimal
```

```
## [Tune-y] 2: mmce.test.mean=0.2055556; time: 0.0 min
```

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

```
## [Tune-y] 3: mmce.test.mean=0.1916667; time: 0.0 min
```

```
## [Tune-x] 4: k=6; kernel=rectangular
```

```
## [Tune-y] 4: mmce.test.mean=0.2416667; time: 0.0 min
```

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

```
## [Tune-y] 5: mmce.test.mean=0.2055556; time: 0.0 min
```

```
## [Tune] Result: k=2; kernel=optimal : mmce.test.mean=0.1916667
```

```
## [Resample] iter 5:    0.1000000
```

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

```
##            Type len Def              Constr Req Tunable Trafo
## k       integer   -   -             2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=3; kernel=optimal
## [Tune-y] 1: mmce.test.mean=0.1722222; time: 0.0 min
```

```
## [Tune-x] 2: k=7; kernel=rectangular
```

```
## [Tune-y] 2: mmce.test.mean=0.2138889; time: 0.0 min
```

```
## [Tune-x] 3: k=10; kernel=rectangular
```

```
## [Tune-y] 3: mmce.test.mean=0.2444444; time: 0.0 min
```

```
## [Tune-x] 4: k=2; kernel=rectangular
```

```
## [Tune-y] 4: mmce.test.mean=0.2166667; time: 0.0 min
```

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

```
## [Tune-y] 5: mmce.test.mean=0.2083333; time: 0.0 min
```

```
## [Tune] Result: k=3; kernel=optimal : mmce.test.mean=0.1722222
```

```
## [Resample] iter 6:    0.1750000
```

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

```
##             Type len Def              Constr Req Tunable Trafo
## k        integer   -   -             2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=10; kernel=rectangular
```

```
## [Tune-y] 1: mmce.test.mean=0.2722222; time: 0.0 min
```

```
## [Tune-x] 2: k=5; kernel=rectangular
```

```
## [Tune-y] 2: mmce.test.mean=0.2166667; time: 0.0 min
```

```
## [Tune-x] 3: k=9; kernel=rectangular
## [Tune-y] 3: mmce.test.mean=0.2472222; time: 0.0 min
```

```
## [Tune-x] 4: k=6; kernel=optimal
```

```
## [Tune-y] 4: mmce.test.mean=0.1638889; time: 0.0 min
```

```
## [Tune-x] 5: k=5; kernel=optimal
```

```
## [Tune-y] 5: mmce.test.mean=0.1500000; time: 0.0 min
```

```
## [Tune] Result: k=5; kernel=optimal : mmce.test.mean=0.1500000
```

```
## [Resample] iter 7:    0.1250000
```

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

```
##            Type len Def              Constr Req Tunable Trafo
## k       integer   -   -             2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=7; kernel=rectangular
```

```
## [Tune-y] 1: mmce.test.mean=0.2111111; time: 0.0 min
```

```
## [Tune-x] 2: k=6; kernel=optimal
```

```
## [Tune-y] 2: mmce.test.mean=0.1472222; time: 0.0 min
```

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

```
## [Tune-y] 3: mmce.test.mean=0.2055556; time: 0.0 min
```

```
## [Tune-x] 4: k=2; kernel=optimal
```

```
## [Tune-y] 4: mmce.test.mean=0.1472222; time: 0.0 min
```

```
## [Tune-x] 5: k=7; kernel=optimal
## [Tune-y] 5: mmce.test.mean=0.1527778; time: 0.0 min
```

```
## [Tune] Result: k=6; kernel=optimal : mmce.test.mean=0.1472222
```

```
## [Resample] iter 8:    0.2250000
```

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

```
##            Type len Def               Constr Req Tunable Trafo
## k       integer   -   -              2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=8; kernel=rectangular
```

```
## [Tune-y] 1: mmce.test.mean=0.2666667; time: 0.0 min
```

```
## [Tune-x] 2: k=8; kernel=rectangular
```

```
## [Tune-y] 2: mmce.test.mean=0.2361111; time: 0.0 min
```

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

```
## [Tune-y] 3: mmce.test.mean=0.1972222; time: 0.0 min
```

```
## [Tune-x] 4: k=10; kernel=rectangular
```

```
## [Tune-y] 4: mmce.test.mean=0.2527778; time: 0.0 min
```

```
## [Tune-x] 5: k=8; kernel=rectangular
```

```
## [Tune-y] 5: mmce.test.mean=0.2416667; time: 0.0 min
```

```
## [Tune] Result: k=4; kernel=rectangular : mmce.test.mean=0.1972222
```

```
## [Resample] iter 9:    0.3500000
```

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

```
##             Type len Def              Constr Req Tunable Trafo
## k        integer   -   -             2 to 10   -    TRUE     -
## kernel discrete   -   - rectangular,optimal   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: k=7; kernel=optimal
```

```
## [Tune-y] 1: mmce.test.mean=0.1583333; time: 0.0 min
```

```
## [Tune-x] 2: k=6; kernel=optimal
```

```
## [Tune-y] 2: mmce.test.mean=0.1611111; time: 0.0 min
```

```
## [Tune-x] 3: k=8; kernel=optimal
```

```
## [Tune-y] 3: mmce.test.mean=0.1694444; time: 0.0 min
```

```
## [Tune-x] 4: k=8; kernel=optimal
```

```
## [Tune-y] 4: mmce.test.mean=0.1694444; time: 0.0 min
```

```
## [Tune-x] 5: k=8; kernel=rectangular
```

```
## [Tune-y] 5: mmce.test.mean=0.2500000; time: 0.0 min
```

```
## [Tune] Result: k=7; kernel=optimal : mmce.test.mean=0.1583333
```

```
## [Resample] iter 10:   0.0500000
```

```
##
```

```
## Aggregated Result: mmce.test.mean=0.1600000
```

```
##
```

```
## Task: train, Learner: classif.randomForest.tuned
```

```
## Resampling: cross-validation
## Measures:             mmce
```

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

```
##           Type len Def              Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=100
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=200
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=100
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=100
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=300
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=100 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 1:    0.0000000
```

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

```
##           Type len Def              Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
## Imputation value: 1
```

```
## [Tune-x] 1: ntree=200
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=100
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=200
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=400
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=300
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=200 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 2:    0.0000000
```

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

```
##           Type len Def              Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=400
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=300
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
## [Tune-x] 3: ntree=300
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=200
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=100
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=300 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 3:    0.0000000
```

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

```
##           Type len Def              Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=500
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=400
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=300
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=500
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=100
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=500 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 4:    0.0000000
```

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

```
##           Type len Def               Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=200
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=500
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=500
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=100
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=400
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=100 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 5:    0.0000000
```

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

```
##           Type len Def               Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=300
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=300
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=400
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=200
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=100
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=400 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 6:    0.0000000
```

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

```
##           Type len Def               Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=400
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=400
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=200
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=300
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=100
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=400 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 7:    0.0000000
```

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

```
##           Type len Def              Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=100
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=100
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=300
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
## [Tune-x] 4: ntree=500
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=300
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=100 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 8:    0.0000000
```

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

```
##           Type len Def              Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=100
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=500
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=400
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=300
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=400
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=100 : mmce.test.mean=0.0000000
## [Resample] iter 9:    0.0000000
```

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

```
##           Type len Def              Constr Req Tunable Trafo
## ntree discrete   -   - 100,200,300,400,500   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

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

```
## [Tune-x] 1: ntree=200
```

```
## [Tune-y] 1: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 2: ntree=500
```

```
## [Tune-y] 2: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 3: ntree=200
```

```
## [Tune-y] 3: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 4: ntree=500
```

```
## [Tune-y] 4: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune-x] 5: ntree=300
```

```
## [Tune-y] 5: mmce.test.mean=0.0000000; time: 0.0 min
```

```
## [Tune] Result: ntree=500 : mmce.test.mean=0.0000000
```

```
## [Resample] iter 10:   0.0000000
```

```
##
```

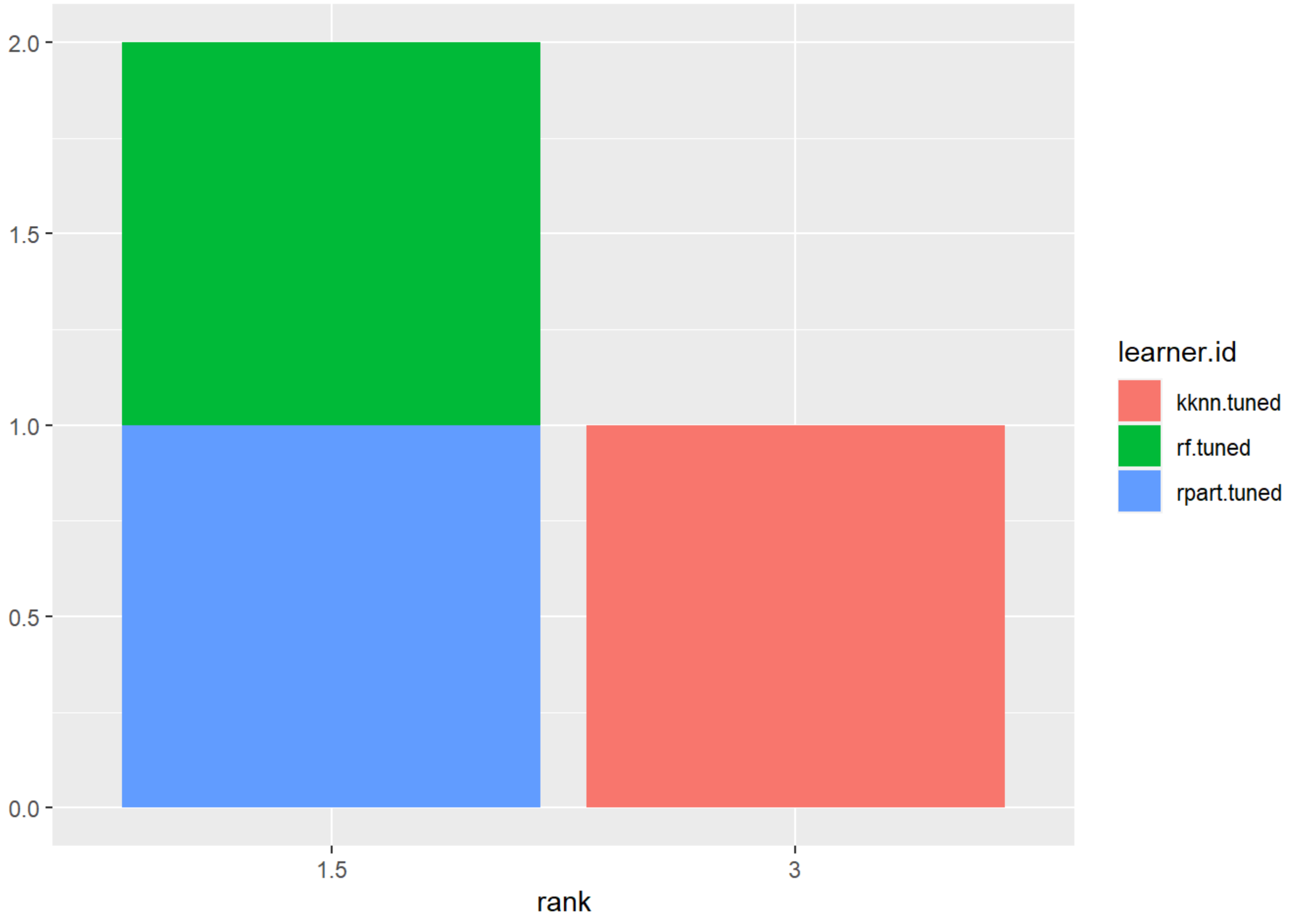
```
## Aggregated Result: mmce.test.mean=0.0000000
```

```
##
```

```
getBMRAggrPerformances(bench1)
```

```
## $train
## $train$classif.rpart.tuned
## mmce.test.mean
##              0
##
## $train$classif.kknn.tuned
## mmce.test.mean
##           0.16
##
## $train$classif.randomForest.tuned
## mmce.test.mean
##              0
```

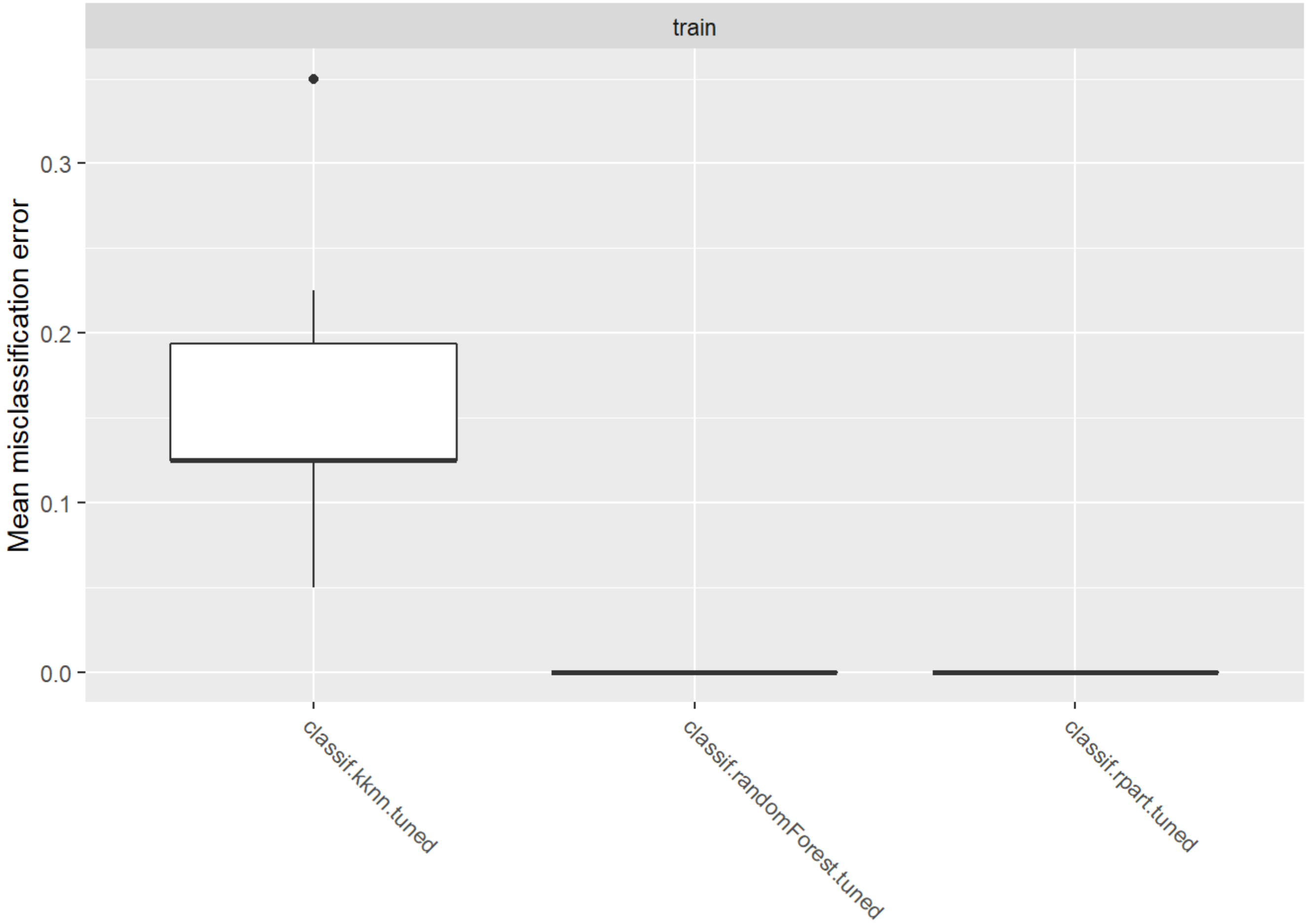
```
plotBMRRanksAsBarChart(bench1)
```



get all the predcictions

Decision Tree

```
plotBMRBoxplots(bench1)
```



## Model Stats

```
prediction <- getBMRPredictions(bench1)
dtree_predict <- prediction$train$classif.rpart.tuned
calculateConfusionMatrix(dtree_predict)
```

```
##                  predicted
## true              Not Subscribed Subscribed -err.-
##   Not Subscribed             213          0      0
##   Subscribed                   0        187      0
##   -err.-                       0          0      0
```

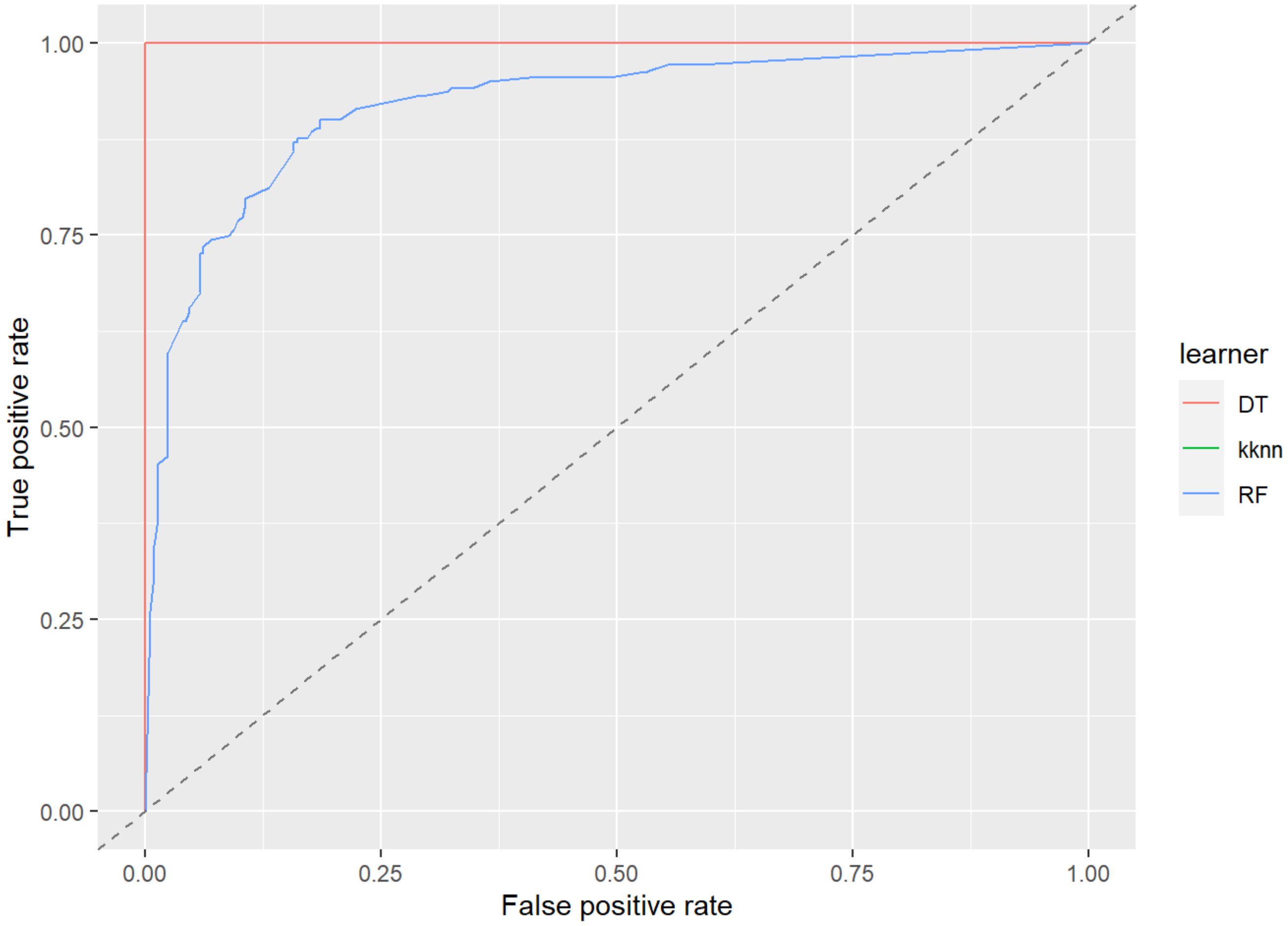
```
prediction <- getBMRPredictions(bench1)
rf_predict <- prediction$train$classif.kknn.tuned
calculateConfusionMatrix(rf_predict)
```

```
##                  predicted
## true              Not Subscribed Subscribed -err.-
##   Not Subscribed             182         31     31
##   Subscribed                  33        154     33
##   -err.-                      33         31     64
```

```
prediction <- getBMRPredictions(bench1)
knn_predict <- prediction$train$classif.kknn.tuned
calculateConfusionMatrix(knn_predict)
```

```
##                   predicted
## true               Not Subscribed Subscribed -err.-
##    Not Subscribed             182         31     31
##    Subscribed                  33        154     33
##    -err.-                      33         31     64
```

```
cur <- generateThreshVsPerfData(list(kknn = knn_predict, DT = dtree_predict, RF = rf_predic
t),
                                measures = list(fpr, tpr))
plotROCCurves(cur)
```


True positive rate
False positive rate
learner
DT
kknn
RF