Bank Marketing Data

Group 8

Load Data

```
#Read dataset
bank_df <- read_delim("bank-additional-full.csv", delim=";")
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_character(),
##   age = col_double(),
##   duration = col_double(),
##   campaign = col_double(),
##   pdays = col_double(),
##   previous = col_double(),
##   emp.var.rate = col_double(),
##   cons.price.idx = col_double(),
##   cons.conf.idx = col_double(),
##   euribor3m = col_double(),
##   nr.employed = col_double()
## )
## i Use `spec()` for the full column specifications.
```

```
#Assign category to all categorical variables

#2.job as category
bank_df$job <- as.factor(bank_df$job)

#3.marital status as category
bank_df$marital <- as.factor(bank_df$marital)

#4.education as category
bank_df$education <- as.factor(bank_df$education)

#5.credit default as category
bank_df$default <- as.factor(bank_df$default)

#6.housing loan as category
bank_df$housing <- as.factor(bank_df$housing)

#7.personal loan as category
bank_df$loan <- as.factor(bank_df$loan)

#8.contact communication type as category
bank_df$contact <- as.factor(bank_df$contact)

#9.last contact month of year as category
bank_df$month <- as.factor(bank_df$month)

#10.last contact day of the month as category
bank_df$day_of_week <- as.factor(bank_df$day_of_week)

#15.outcome of the previous marketing campaign as category
bank_df$poutcome <- as.factor(bank_df$poutcome)

#21.output y as binary factor
bank_df$y <- factor(bank_df$y, levels = c("no","yes"))

dim(bank_df)
```

```
## [1] 41188    21
```

Data preprocessing

```
bank_df %>%
  summarise_all(list(~sum(. == "unknown"))) %>%
  gather(key = "variable", value = "nr_unknown") %>%
  arrange(-nr_unknown)
```

```
## # A tibble: 21 x 2
##    variable    nr_unknown
##    <chr>            <int>
##  1 default           8597
##  2 education         1731
##  3 housing            990
##  4 loan               990
##  5 job                330
##  6 marital             80
##  7 age                  0
##  8 contact              0
##  9 month                0
## 10 day_of_week          0
## # ... with 11 more rows
```

```
# Analyse default
table(bank_df$default)
```

```
##
##      no unknown     yes
##   32588    8597       3
```

```
## This is not usable, too few "yes" to evaluate
```

analyse the unknown values

```
# setting default parameters for crosstables
# fun_crosstable = function(df, var1, var2){
#   # df: dataframe containing both columns to cross
#   # var1, var2: columns to cross together.
#   CrossTable(df$var1, df$var2,
#              prop.r = T,
#              prop.c = F,
#              prop.t = F,
#              prop.chisq = F,
#              dnn = c(var1, var2)) # dimension names
# }

#default
CrossTable(bank_df$default, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("default",
"y"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##              | y
##      default |        no |       yes | Row Total |
## -------------|-----------|-----------|-----------|
##           no |     28391 |      4197 |     32588 |
##              |     0.871 |     0.129 |     0.791 |
##              |     0.689 |     0.102 |           |
## -------------|-----------|-----------|-----------|
##      unknown |      8154 |       443 |      8597 |
##              |     0.948 |     0.052 |     0.209 |
##              |     0.198 |     0.011 |           |
## -------------|-----------|-----------|-----------|
##          yes |         3 |         0 |         3 |
##              |     1.000 |     0.000 |     0.000 |
##              |     0.000 |     0.000 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

```
table(bank_df$default)
```

```
##
##      no unknown     yes
##   32588    8597       3
```

```
# job
CrossTable(bank_df$job, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("job", "y"))
```

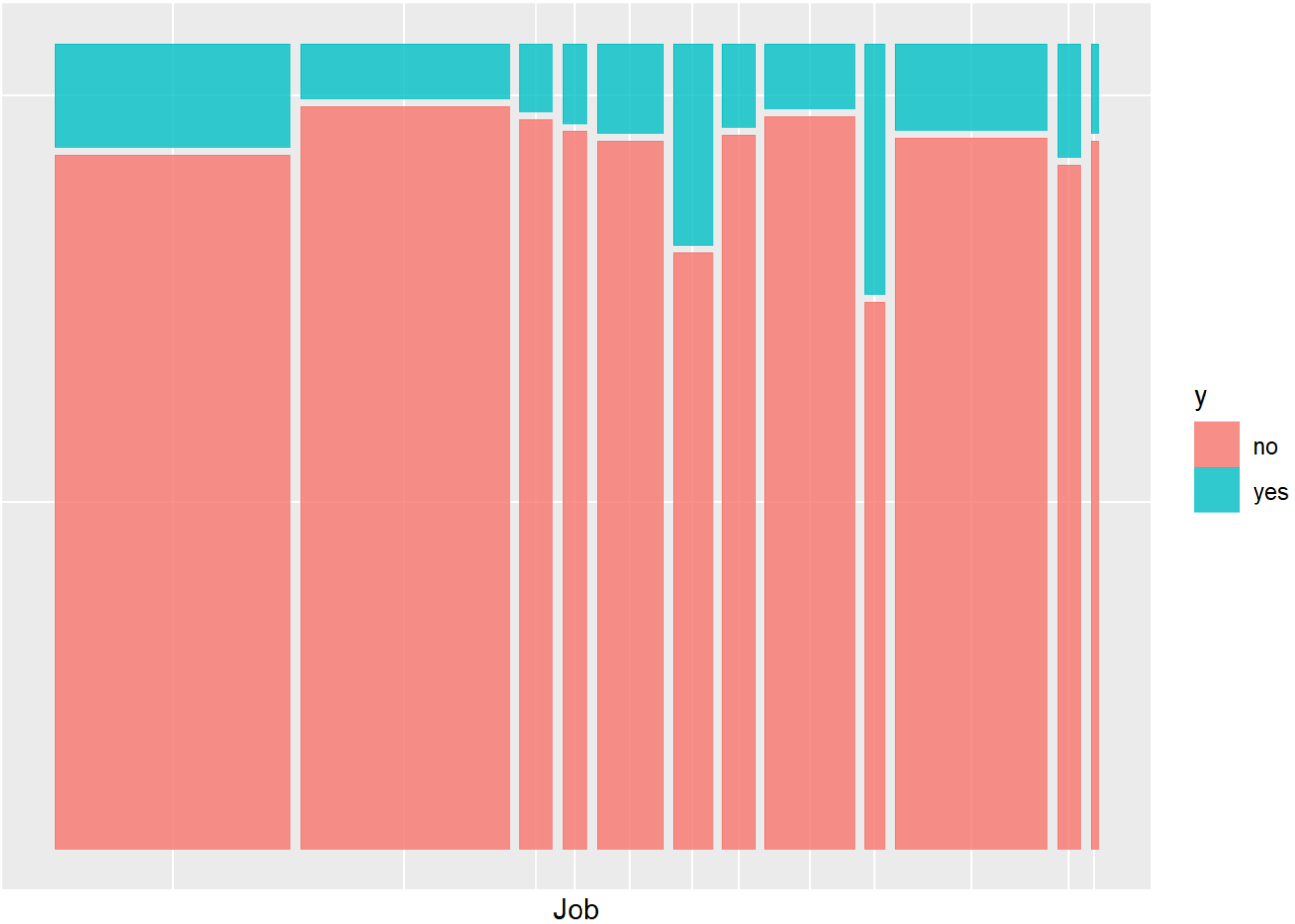
```
## 
## 
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
## 
## 
## Total Observations in Table:  41188 
## 
## 
##               | y 
##           job |        no |       yes | Row Total | 
## --------------|-----------|-----------|-----------|
##        admin. |      9070 |      1352 |     10422 | 
##               |     0.870 |     0.130 |     0.253 | 
##               |     0.220 |     0.033 |           | 
## --------------|-----------|-----------|-----------|
##   blue-collar |      8616 |       638 |      9254 | 
##               |     0.931 |     0.069 |     0.225 | 
##               |     0.209 |     0.015 |           | 
## --------------|-----------|-----------|-----------|
##  entrepreneur |      1332 |       124 |      1456 | 
##               |     0.915 |     0.085 |     0.035 | 
##               |     0.032 |     0.003 |           | 
## --------------|-----------|-----------|-----------|
##     housemaid |       954 |       106 |      1060 | 
##               |     0.900 |     0.100 |     0.026 | 
##               |     0.023 |     0.003 |           | 
## --------------|-----------|-----------|-----------|
##    management |      2596 |       328 |      2924 | 
##               |     0.888 |     0.112 |     0.071 | 
##               |     0.063 |     0.008 |           | 
## --------------|-----------|-----------|-----------|
##       retired |      1286 |       434 |      1720 | 
##               |     0.748 |     0.252 |     0.042 | 
##               |     0.031 |     0.011 |           | 
## --------------|-----------|-----------|-----------|
## self-employed |      1272 |       149 |      1421 | 
##               |     0.895 |     0.105 |     0.035 | 
##               |     0.031 |     0.004 |           | 
## --------------|-----------|-----------|-----------|
##      services |      3646 |       323 |      3969 | 
##               |     0.919 |     0.081 |     0.096 | 
##               |     0.089 |     0.008 |           | 
## --------------|-----------|-----------|-----------|
##       student |       600 |       275 |       875 | 
##               |     0.686 |     0.314 |     0.021 | 
##               |     0.015 |     0.007 |           | 
## --------------|-----------|-----------|-----------|
##    technician |      6013 |       730 |      6743 | 
##               |     0.892 |     0.108 |     0.164 | 
##               |     0.146 |     0.018 |           | 
## --------------|-----------|-----------|-----------|
##    unemployed |       870 |       144 |      1014 | 
```

```
##                |      0.858 |      0.142 |      0.025 |
##                |      0.021 |      0.003 |            |
## ---------------|------------|------------|------------|
##        unknown |        293 |         37 |        330 |
##                |      0.888 |      0.112 |      0.008 |
##                |      0.007 |      0.001 |            |
## ---------------|------------|------------|------------|
##   Column Total |      36548 |       4640 |      41188 |
## ---------------|------------|------------|------------|
##
##
```

```
table(bank_df$job)
```

```
##
##        admin.   blue-collar  entrepreneur     housemaid    management
##         10422          9254          1456          1060          2924
##       retired self-employed      services       student    technician
##          1720          1421          3969           875          6743
##    unemployed       unknown
##          1014           330
```

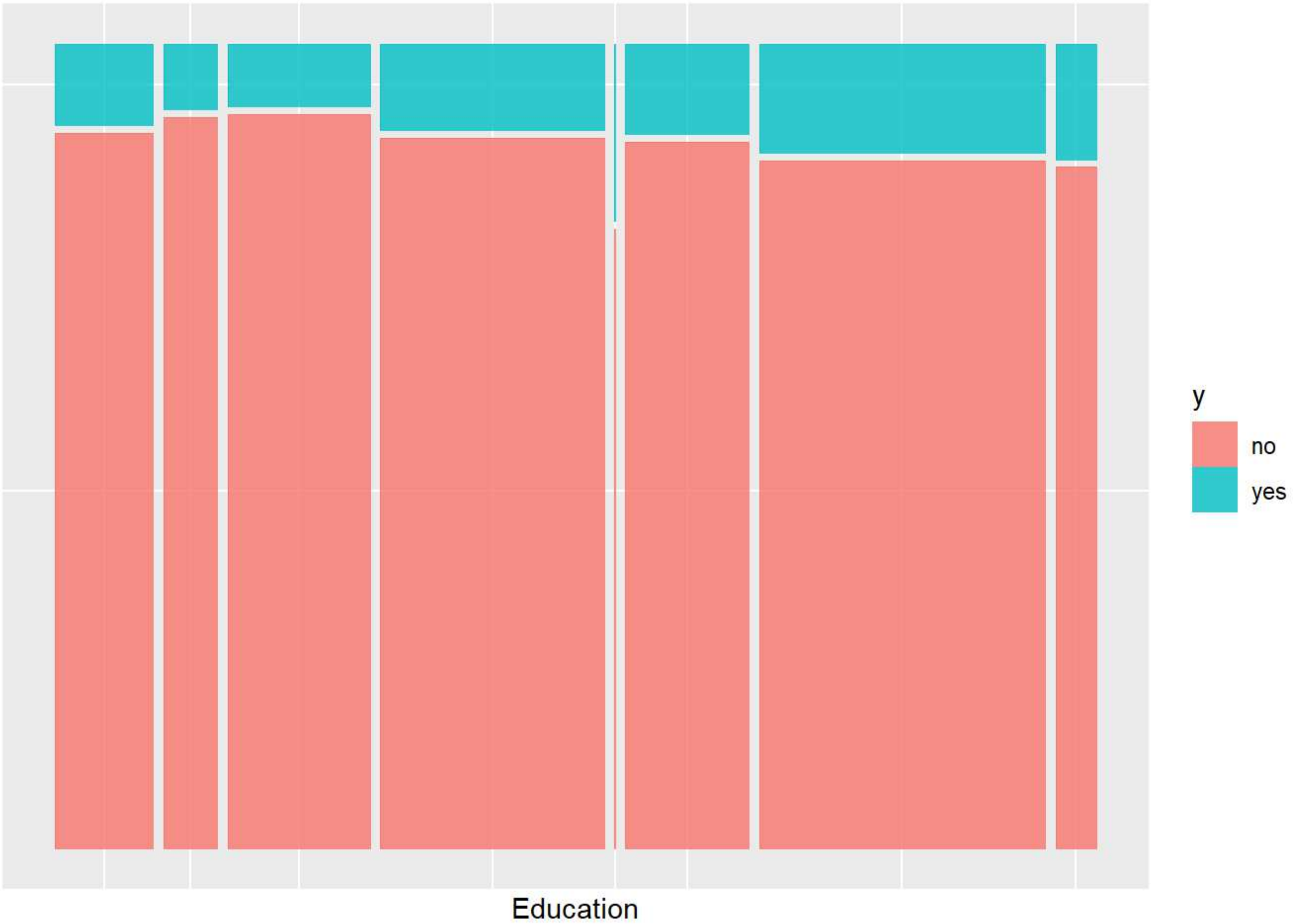
```
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, job), fill = y)) +
  #mosaic_theme +
  xlab("Job") +
  ylab(NULL)
```



```r
bank_df <- bank_df %>%
  mutate(job = recode(job, "unknown" = "unconventional"))


# marital
CrossTable(bank_df$marital, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("marital",
"y"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##              | y
##      marital |        no |       yes | Row Total |
## -------------|-----------|-----------|-----------|
##     divorced |      4136 |       476 |      4612 |
##              |     0.897 |     0.103 |     0.112 |
##              |     0.100 |     0.012 |           |
## -------------|-----------|-----------|-----------|
##      married |     22396 |      2532 |     24928 |
##              |     0.898 |     0.102 |     0.605 |
##              |     0.544 |     0.061 |           |
## -------------|-----------|-----------|-----------|
##       single |      9948 |      1620 |     11568 |
##              |     0.860 |     0.140 |     0.281 |
##              |     0.242 |     0.039 |           |
## -------------|-----------|-----------|-----------|
##      unknown |        68 |        12 |        80 |
##              |     0.850 |     0.150 |     0.002 |
##              |     0.002 |     0.000 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

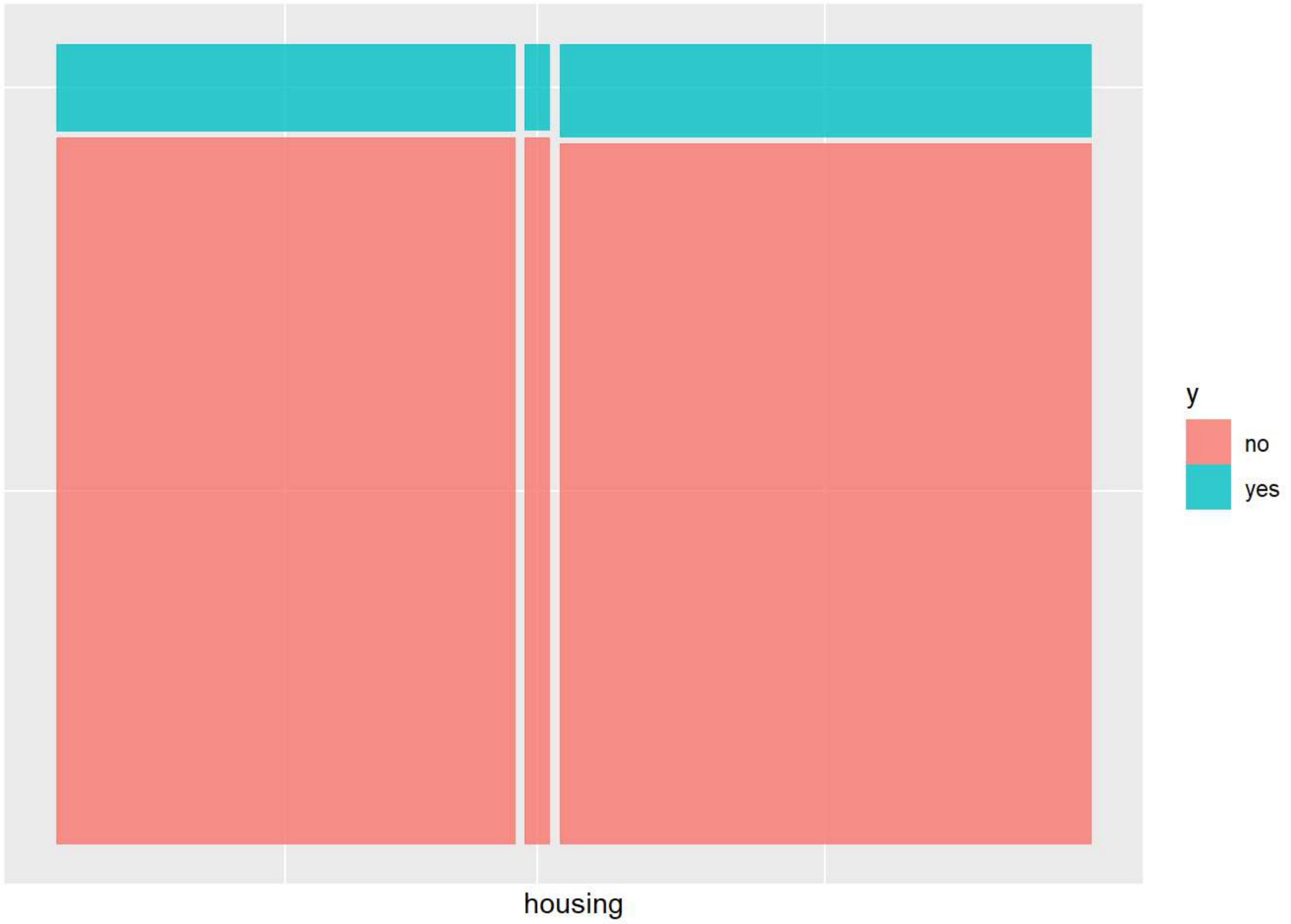
```
## can merge single+unknown, married+divorced since values are similar
bank_df = bank_df %>%
  mutate(marital = recode(marital, "unknown" = "single", "divorced"="married"))

# education
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, education),  fill = y)) +
  #mosaic_theme +
  xlab("Education") +
  ylab(NULL)
```




```
    ## recode unknown as univeristy degree because proportions are similar
bank_df = bank_df %>%
  mutate(education = recode(education, "unknown" = "university.degree"))

# housing
CrossTable(bank_df$housing, bank_df$y, prop.r = T, prop.c=F, prop.chisq=F, dnn = c("housing",
"y"))
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |           N / Row Total |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  41188
##
##
##              | y
##      housing |        no |       yes | Row Total |
## -------------|-----------|-----------|-----------|
##           no |     16596 |      2026 |     18622 |
##              |     0.891 |     0.109 |     0.452 |
##              |     0.403 |     0.049 |           |
## -------------|-----------|-----------|-----------|
##      unknown |       883 |       107 |       990 |
##              |     0.892 |     0.108 |     0.024 |
##              |     0.021 |     0.003 |           |
## -------------|-----------|-----------|-----------|
##          yes |     19069 |      2507 |     21576 |
##              |     0.884 |     0.116 |     0.524 |
##              |     0.463 |     0.061 |           |
## -------------|-----------|-----------|-----------|
## Column Total |     36548 |      4640 |     41188 |
## -------------|-----------|-----------|-----------|
##
##
```

```
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, housing),  fill = y)) +
  #mosaic_theme +
  xlab("housing") +
  ylab(NULL)
```



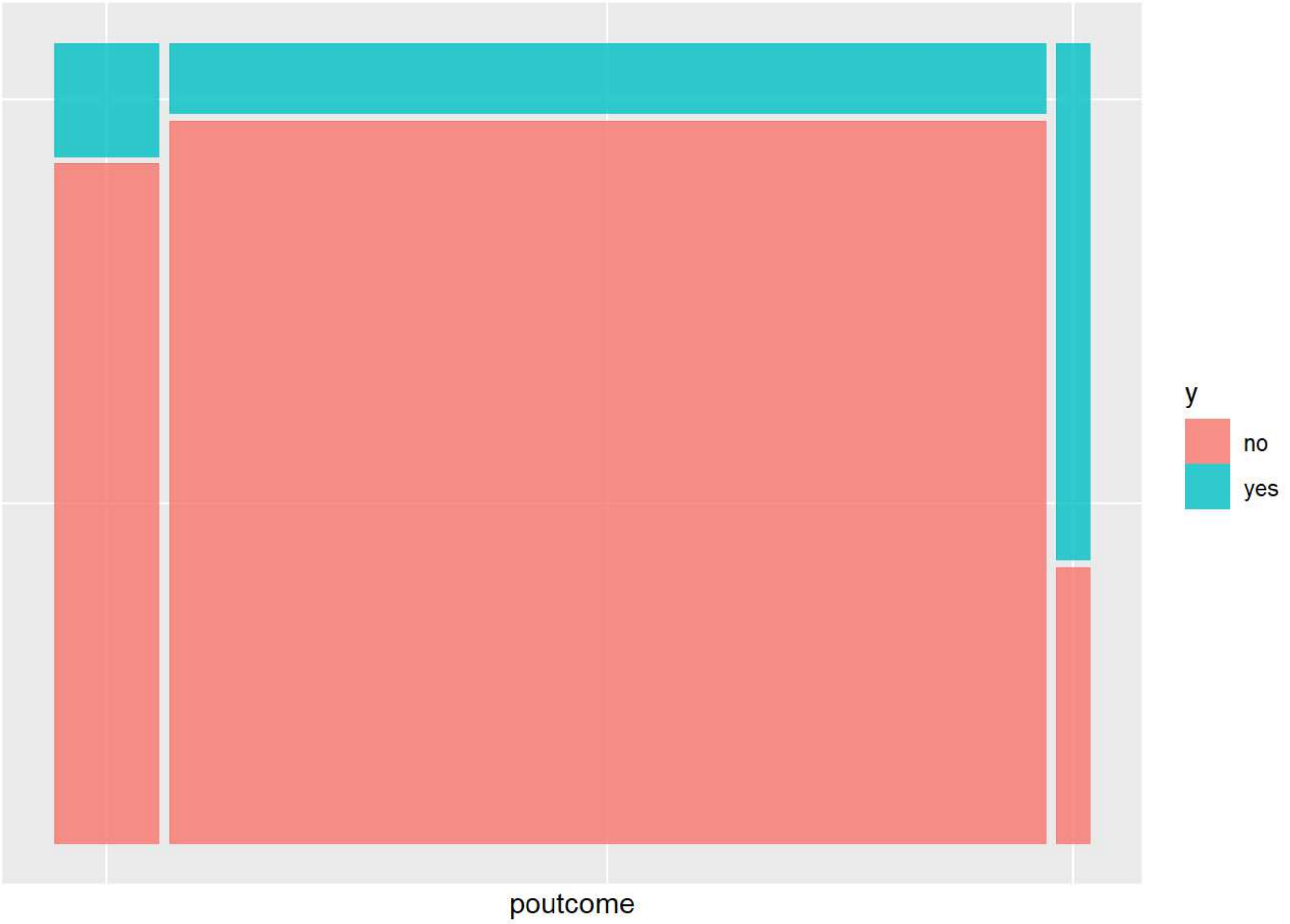

```
  ## the plot looks very similar, do chisquared test to see if there are differences
chisq.test(bank_df$housing, bank_df$y) # drop this column
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$housing and bank_df$y
## X-squared = 5.6845, df = 2, p-value = 0.05829
```

```
bank_df$housing <- NULL

# loan
chisq.test(bank_df$loan, bank_df$y) # drop col, pvalue >0.1
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$loan and bank_df$y
## X-squared = 1.094, df = 2, p-value = 0.5787
```

```
bank_df$loan <- NULL

# pdays
# poutcome
bank_df %>%
  ggplot() +
  geom_mosaic(aes(x = product(y, poutcome),  fill = y)) +
  #mosaic_theme +
  xlab("poutcome") +
  ylab(NULL)
```




```
bank_df = bank_df %>%
  mutate(past_dummyvar = recode(poutcome, "failure" = 0.5, "nonexistent"=0.2, "success"=1))
# combining previous and poutcome
bank_df$past_dummyvar1 = bank_df$past_dummyvar*(bank_df$previous+1)
chisq.test(bank_df$past_dummyvar1, bank_df$y)
```

```
## Warning in chisq.test(bank_df$past_dummyvar1, bank_df$y): Chi-squared
## approximation may be incorrect
```

```
##
##  Pearson's Chi-squared test
##
## data:  bank_df$past_dummyvar1 and bank_df$y
## X-squared = 4383.4, df = 11, p-value < 2.2e-16
```

```
bank_df$previous <-NULL
bank_df$poutcome <-NULL
bank_df$past_dummyvar <-NULL

bank_df = bank_df %>%
  mutate(pdays_dummy = if_else(pdays == 999, "0", "1")) %>%
  select(-pdays)
bank_df$pdays<-NULL

#resolve default, let yes become unknown
bank_df = bank_df %>%
  mutate(default = recode(default, "yes"="unknown"))


# dayofweek
bank_df = bank_df %>%
  mutate(day_of_week = recode(day_of_week, "mon"=1, "tue"=2,"wed"=3,"thu"=4,"fri"=5))

# age
bank_df = bank_df %>%
  mutate(age = if_else(
    age<20, 1, if_else(
      age<23, 2, if_else(
        age<26, 3, if_else(
          age<31, 4, if_else(
            age<41, 5, if_else(age<51, 6, 7)))))))

#dataset after preprocessing
dim(bank_df)
```

```
## [1] 41188    18
```

```
summary(bank_df)
```

```
##       age                  job           marital     
##  Min.   :1.000   admin.     :10422   married:29540  
##  1st Qu.:5.000   blue-collar: 9254   single :11648  
##  Median :5.000   technician : 6743                  
##  Mean   :5.367   services   : 3969                  
##  3rd Qu.:6.000   management : 2924                  
##  Max.   :7.000   retired    : 1720                  
##                  (Other)    : 6156                  
##                education        default           contact          month      
##  basic.4y           : 4176   no     :32588   cellular :26144   may    :13769  
##  basic.6y           : 2292   unknown: 8600   telephone:15044   jul    : 7174  
##  basic.9y           : 6045                                     aug    : 6178  
##  high.school        : 9515                                     jun    : 5318  
##  illiterate         :   18                                     nov    : 4101  
##  professional.course: 5243                                     apr    : 2632  
##  university.degree  :13899                                     (Other): 2016  
##   day_of_week      duration         campaign       emp.var.rate     
##  Min.   :1.00   Min.   :   0.0   Min.   : 1.000   Min.   :-3.40000  
##  1st Qu.:2.00   1st Qu.: 102.0   1st Qu.: 1.000   1st Qu.:-1.80000  
##  Median :3.00   Median : 180.0   Median : 2.000   Median : 1.10000  
##  Mean   :2.98   Mean   : 258.3   Mean   : 2.568   Mean   : 0.08189  
##  3rd Qu.:4.00   3rd Qu.: 319.0   3rd Qu.: 3.000   3rd Qu.: 1.40000  
##  Max.   :5.00   Max.   :4918.0   Max.   :56.000   Max.   : 1.40000  
##                                                                     
##  cons.price.idx  cons.conf.idx     euribor3m      nr.employed     y        
##  Min.   :92.20   Min.   :-50.8   Min.   :0.634   Min.   :4964   no :36548  
##  1st Qu.:93.08   1st Qu.:-42.7   1st Qu.:1.344   1st Qu.:5099   yes: 4640  
##  Median :93.75   Median :-41.8   Median :4.857   Median :5191              
##  Mean   :93.58   Mean   :-40.5   Mean   :3.621   Mean   :5167              
##  3rd Qu.:93.99   3rd Qu.:-36.4   3rd Qu.:4.961   3rd Qu.:5228              
##  Max.   :94.77   Max.   :-26.9   Max.   :5.045   Max.   :5228              
##                                                                            
##  past_dummyvar1   pdays_dummy       
##  Min.   :0.2000   Length:41188      
##  1st Qu.:0.2000   Class :character  
##  Median :0.2000   Mode  :character  
##  Mean   :0.3703                     
##  3rd Qu.:0.2000                     
##  Max.   :8.0000                     
## 
```

```
#Standardize the numeric features
num.ind <- sapply(bank_df, is.numeric)
bank_df.mean <- apply(bank_df[,num.ind], 2, mean)
bank_df.sd <- apply(bank_df[,num.ind], 2, sd)

bank_df.scaled <- bank_df

bank_df.scaled[,num.ind] <- scale(bank_df[,num.ind], center=bank_df.mean, scale=bank_df.sd)
```

```r
# splitting train and test
library(caTools)
set.seed(1)
smp_size <- floor(0.8*nrow(bank_df.scaled))
train_ind <- sample(seq_len(nrow(bank_df.scaled)), size = smp_size)
train <- bank_df.scaled[train_ind, ]
test <- bank_df.scaled[-train_ind, ]
```

KNN

```r
set.seed(8)
#use K-fold CV to find best trCtrl:
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 1) #5 fold CV repeated 1
 times

# knn.fit <- train(y ~., data = train, method = "knn",
#                   trControl=trctrl, tuneLength = 10) # tuneLength parameter tells the algori
thm to try different default values for the main parameter

knn.fit <- train(y ~., data = train, method = "knn",
                 trControl=trctrl) # tuneLength parameter tells the algorithm to try differen
t default values for the main parameter

# knn.fit <- train(y ~., data = train, method = "knn")#by default bootstrap is used to find t
uning parameter -> trCtrl
knn.fit
```

```
## k-Nearest Neighbors
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.8994540  0.4346305
##   7  0.9020639  0.4392127
##   9  0.9047954  0.4458783
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```r
#predict using test data
knn.pred <- predict(knn.fit, newdata = test)
#knn.pred

#confusion matrix
cm.knn <- table(knn.pred, test$y)
cm.knn
```

```
##
## knn.pred   no  yes
##      no  7076  521
##      yes  225  416
```

```
TP <- cm.knn[2,2]
TN <- cm.knn[1,1]
FP <- cm.knn[2,1]
FN <- cm.knn[1,2]

#FPR / Type I error
FPR.knn = FP/(FP+TN)
FPR.knn
```

```
## [1] 0.0308177
```

```
#FNR / Type II error
FNR.knn = FN/(FN+TP)
FNR.knn
```

```
## [1] 0.5560299
```

```
#Precision
precis.knn = TP/(TP+FP)
precis.knn
```

```
## [1] 0.648986
```

```
#Recall / sensitivity
recall.knn = TP/(TP+FN)
recall.knn
```

```
## [1] 0.4439701
```

```
#misclassification error
test.err.knn = 1-(sum(diag(cm.knn))/sum(cm.knn))
test.err.knn
```

```
## [1] 0.09055596
```

Logistic Regression

```
set.seed(8)
glm.fit <- glm(y ~., data = train, family = binomial)
summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -5.9280  -0.3025  -0.1891  -0.1390   3.2894
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)                  -2.750785   0.134765 -20.412  < 2e-16 ***
## age                          -0.055930   0.027213  -2.055 0.039852 *
## jobblue-collar               -0.246842   0.087127  -2.833 0.004609 **
## jobentrepreneur              -0.145059   0.138872  -1.045 0.296230
## jobhousemaid                  0.039898   0.162792   0.245 0.806388
## jobmanagement                -0.028385   0.094597  -0.300 0.764129
## jobretired                    0.352578   0.106563   3.309 0.000938 ***
## jobself-employed             -0.150559   0.132842  -1.133 0.257060
## jobservices                  -0.127794   0.093575  -1.366 0.172040
## jobstudent                    0.059646   0.124981   0.477 0.633192
## jobtechnician                 0.006576   0.078381   0.084 0.933134
## jobunemployed                -0.089351   0.145955  -0.612 0.540417
## jobunconventional            -0.043046   0.275330  -0.156 0.875763
## maritalsingle                -0.031908   0.057275  -0.557 0.577456
## educationbasic.6y             0.037662   0.136393   0.276 0.782449
## educationbasic.9y             0.045684   0.105675   0.432 0.665517
## educationhigh.school          0.055489   0.101846   0.545 0.585865
## educationilliterate           1.558755   0.740636   2.105 0.035325 *
## educationprofessional.course  0.101641   0.112486   0.904 0.366216
## educationuniversity.degree    0.151933   0.098805   1.538 0.124123
## defaultunknown               -0.295427   0.074573  -3.962 7.45e-05 ***
## contacttelephone             -0.630566   0.085678  -7.360 1.84e-13 ***
## monthaug                      0.720398   0.133252   5.406 6.43e-08 ***
## monthdec                      0.330156   0.229521   1.438 0.150305
## monthjul                      0.078329   0.106239   0.737 0.460947
## monthjun                     -0.472526   0.140064  -3.374 0.000742 ***
## monthmar                      1.945546   0.159750  12.179  < 2e-16 ***
## monthmay                     -0.536562   0.091073  -5.892 3.83e-09 ***
## monthnov                     -0.565715   0.134703  -4.200 2.67e-05 ***
## monthoct                      0.055455   0.170805   0.325 0.745431
## monthsep                      0.286532   0.199121   1.439 0.150154
## day_of_week                   0.033569   0.022596   1.486 0.137376
## duration                      1.193618   0.021334  55.950  < 2e-16 ***
## campaign                     -0.108741   0.035285  -3.082 0.002057 **
## emp.var.rate                 -2.620715   0.248940 -10.527  < 2e-16 ***
## cons.price.idx                1.175153   0.162518   7.231 4.80e-13 ***
## cons.conf.idx                 0.090009   0.039579   2.274 0.022956 *
## euribor3m                     0.668129   0.248996   2.683 0.007290 **
## nr.employed                   0.270856   0.250083   1.083 0.278780
## past_dummyvar1               -0.106698   0.027802  -3.838 0.000124 ***
## pdays_dummy1                  1.854235   0.131958  14.052  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 23162  on 32949  degrees of freedom
```

```
## Residual deviance: 13829  on 32909  degrees of freedom
## AIC: 13911
##
## Number of Fisher Scoring iterations: 6
```

```
#predict using test data
glm.prob <- predict(glm.fit, type = "response", newdata = test)

#check which one is 'Yes'
contrasts(test$y)#Yes = 1, Low = 0
```

```
##     yes
## no    0
## yes   1
```

```
glm.pred <- rep('no', nrow(test))
glm.pred[glm.prob > 0.5] <- 'yes' #yes = 1, no = 0

#confusion matrix
cm.reg = table(glm.pred, test$y)
cm.reg
```

```
##
## glm.pred   no  yes
##      no  7111  546
##      yes  190  391
```

```
TP <- cm.reg[2,2]
TN <- cm.reg[1,1]
FP <- cm.reg[2,1]
FN <- cm.reg[1,2]

#FPR / Type I error
FPR.reg = FP/(FP+TN)
FPR.reg
```

```
## [1] 0.02602383
```

```
#FNR / Type II error
FNR.reg = FN/(FN+TP)
FNR.reg
```

```
## [1] 0.5827108
```

```
#Precision
precis.reg = TP/(TP+FP)
precis.reg
```

```
## [1] 0.6729776
```

```
#Recall / sensitivity
recall.reg = TP/(TP+FN)
recall.reg
```

```
## [1] 0.4172892
```

```
#misclassification error
test.err.reg = 1-(sum(diag(cm.reg))/sum(cm.reg))
test.err.reg
```

```
## [1] 0.08934207
```

Decision Tree

```
# train.tree <- data.frame(train)
# test.tree <-data.frame(test)

#Decision Tree using rpart()
set.seed(8)
#use K-fold CV to find best trCtrl:
trctrl <- trainControl(method = "repeatedcv", number = 5, repeats = 1) #5 fold CV repeated 1
 times

cls.tree1 = train(y ~ ., data=train, method="rpart",
                  trControl=trctrl)# tuneLength parameter tells the algorithm to try differen
t default values for the main parameter

# cls.tree1 = train(y ~ ., data=train.tree, method="rpart")#by default bootstrap is used to f
ind tuning parameter -> trCtrl

cls.tree1
```

```
## CART
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.01876857  0.9085282  0.4758751
##   0.02106400  0.9058881  0.4220528
##   0.07061842  0.8964786  0.2597907
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.01876857.
```

```
#plot(cls.tree1$finalModel)
#text(cls.tree1$finalModel)

#predict using test data
tree.pred1 <- predict(cls.tree1, newdata = test)
#tree.pred1

#confusion matrix
cm.tree1 <- table(tree.pred1, test$y)
cm.tree1
```

```
##
## tree.pred1   no  yes
##        no  7145  612
##        yes  156  325
```

```
TP <- cm.tree1[2,2]
TN <- cm.tree1[1,1]
FP <- cm.tree1[2,1]
FN <- cm.tree1[1,2]

#FPR / Type I error
FPR.tree1 = FP/(FP+TN)
FPR.tree1
```

```
## [1] 0.02136694
```

```
#FNR / Type II error
FNR.tree1 = FN/(FN+TP)
FNR.tree1
```

```
## [1] 0.6531483
```

```
#Precision
precis.tree1 = TP/(TP+FP)
precis.tree1
```

```
## [1] 0.6756757
```

```
#Recall / sensitivity
recall.tree1 = TP/(TP+FN)
recall.tree1
```

```
## [1] 0.3468517
```

```
#misclassification error
test.err.tree1 = 1-(sum(diag(cm.tree1))/sum(cm.tree1))
test.err.tree1
```

```
## [1] 0.09322651
```

Random Forest

```
#Random forest with 500 bootstrapped trees
#p = 16
sqrt(16) # ntree = 4
```

```
## [1] 4
```

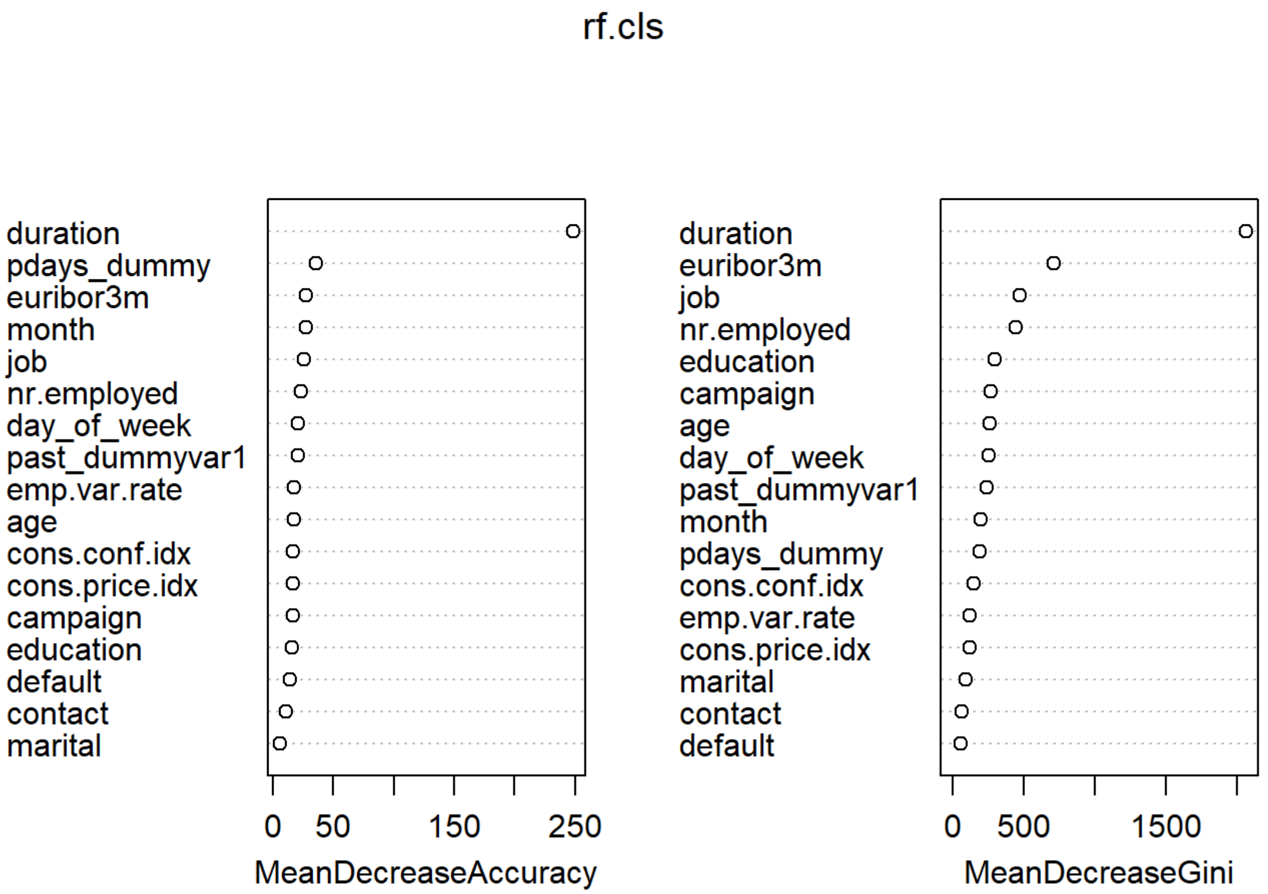
```
set.seed(8)
rf.cls <- randomForest(y ~ .,
                       data = train,
                       mtry = 4,
                       ntree = 500,
                       importance = TRUE)
rf.cls
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train, mtry = 4, ntree = 500,      importance = TRU
E)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 8.58%
## Confusion matrix:
##        no  yes class.error
## no  28159 1088   0.0372004
## yes  1739 1964   0.4696192
```

```
#ls(rf.cls)
importance(rf.cls)
```

```
##                        no          yes MeanDecreaseAccuracy MeanDecreaseGini
## age             17.5994852   2.6431157             17.17497        261.27791
## job             33.4598069  -8.3279418             25.53862        472.00539
## marital          8.4804094  -2.0770280              5.63656         90.42659
## education       18.2285426   0.2902509             15.85282        296.38671
## default          7.9851672   9.9466921             14.06632         57.38234
## contact          5.8083405  31.6780656             10.75700         65.39431
## month           26.1934824   5.7403113             27.21203        201.40406
## day_of_week     19.7355665   7.1906377             20.93292        256.55362
## duration       152.0502038 246.1841391            248.91491       2063.97786
## campaign         9.1802979  14.2948341             16.52478        270.40819
## emp.var.rate    16.4877524   7.0310918             17.40129        124.77060
## cons.price.idx  16.6662412  -3.0556567             16.77137        121.28844
## cons.conf.idx   15.9713846   3.4665630             16.86955        147.71796
## euribor3m       24.9134347  13.3710567             27.65883        709.60922
## nr.employed     19.1091313  23.1302593             23.16106        444.41888
## past_dummyvar1  10.4070327  24.5936232             20.68765        239.75494
## pdays_dummy      0.5297726  55.5552773             36.07905        194.92174
```

```
varImpPlot(rf.cls)
```




```
#predict using test data
rf.pred <- predict(rf.cls, newdata = test, type = "class")
#rf.pred

#confusion matrix
cm.rf <- table(rf.pred, test$y)
cm.rf
```

```
##
## rf.pred   no  yes
##     no  7035  425
##     yes  266  512
```

```
TP <- cm.rf[2,2]
TN <- cm.rf[1,1]
FP <- cm.rf[2,1]
FN <- cm.rf[1,2]

#FPR / Type I error
FPR.rf = FP/(FP+TN)
FPR.rf
```

```
## [1] 0.03643337
```

```
#FNR / Type II error
FNR.rf = FN/(FN+TP)
FNR.rf
```

```
## [1] 0.4535752
```

```
#Precision
precis.rf = TP/(TP+FP)
precis.rf
```

```
## [1] 0.6580977
```

```
#Recall / sensitivity
recall.rf = TP/(TP+FN)
recall.rf
```

```
## [1] 0.5464248
```

```
#misclassification error
test.err.rf = 1-(sum(diag(cm.rf))/sum(cm.rf))
test.err.rf
```

```
## [1] 0.08387958
```

Gradient Boosting

```
#Gradient boosting
set.seed(8)

#Use K-fold CV to find best trControl
fitControl <- trainControl(method = "repeatedcv",
                           number = 5,
                           repeats = 1) #5 folds repeated 1 times

gbm.fit <- train(y ~ ., data = train,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE)

# gbm.fit <- train(y ~ ., data = train,
#                   method = "gbm",
#                   verbose = FALSE) #by default bootstrap is used to find tuning parameter ->
trCtrl
gbm.fit
```

```
## Stochastic Gradient Boosting
##
## 32950 samples
##    17 predictor
##     2 classes: 'no', 'yes'
##
## No pre-processing
## Resampling: Cross-Validated (5 fold, repeated 1 times)
## Summary of sample sizes: 26361, 26359, 26360, 26359, 26361
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.9054326  0.3425904
##   1                  100      0.9085587  0.4064470
##   1                  150      0.9098333  0.4385889
##   2                   50      0.9091959  0.4340753
##   2                  100      0.9124735  0.4971532
##   2                  150      0.9133840  0.5092936
##   3                   50      0.9117148  0.4927134
##   3                  100      0.9135356  0.5149861
##   3                  150      0.9157512  0.5309642
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
#predict using test data
gbm.pred <- predict(gbm.fit, newdata = test)
#gbm.pred

#confusion matrix
cm.gbm <- table(gbm.pred, test$y)
cm.gbm
```

```
##
## gbm.pred   no  yes
##      no  7055  429
##      yes  246  508
```

```
TP <- cm.gbm[2,2]
TN <- cm.gbm[1,1]
FP <- cm.gbm[2,1]
FN <- cm.gbm[1,2]

#FPR / Type I error
FPR.gbm = FP/(FP+TN)
FPR.gbm
```

```
## [1] 0.03369401
```

```
#FNR / Type II error
FNR.gbm = FN/(FN+TP)
FNR.gbm
```

```
## [1] 0.4578442
```

```
#Precision
precis.gbm = TP/(TP+FP)
precis.gbm
```

```
## [1] 0.6737401
```

```
#Recall / sensitivity
recall.gbm = TP/(TP+FN)
recall.gbm
```

```
## [1] 0.5421558
```

```
#misclassification error
test.err.gbm = 1-(sum(diag(cm.gbm))/sum(cm.gbm))
test.err.gbm
```

```
## [1] 0.08193736
```

AdaBoost

```
#AdaBoost
set.seed(8)
x.trainA = model.matrix(data=train, y~.-1)
y.trainA = rep(1, nrow(train))
y.trainA [train$y=="no"]=-1 #for Adaboost

x.testA = model.matrix(data=test, y~.-1)
y.testA = rep(1, nrow(test))
y.testA [test$y=="no"]=-1 #for Adaboost

ada.cls <- adaboost(x.trainA, y.trainA, tree_depth=5, n_rounds=500)
ada.cls
```

```
## AdaBoost: tree_depth =  5  rounds =  500
##
##
##  In-sample confusion matrix:
##     yhat
## y       -1     1
##   -1 28415   832
##   1   1430  2273
```

```
#predict using test data
ada.pred <- predict(ada.cls, x.testA)
#ada.pred

#confusion matrix
cm.ada <- table(ada.pred, y.testA) #-1 is "no", 1 is "yes"
cm.ada
```

```
##         y.testA
## ada.pred   -1    1
##       -1 6998  435
##       1   303  502
```

```
TP <- cm.ada[2,2]
TN <- cm.ada[1,1]
FP <- cm.ada[2,1]
FN <- cm.ada[1,2]

#FPR / Type I error
FPR.ada = FP/(FP+TN)
FPR.ada
```

```
## [1] 0.04150116
```

```
#FNR / Type II error
FNR.ada = FN/(FN+TP)
FNR.ada
```

```
## [1] 0.4642476
```

```r
#Precision
precis.ada = TP/(TP+FP)
precis.ada
```

```
## [1] 0.6236025
```

```r
#Recall / sensitivity
recall.ada = TP/(TP+FN)
recall.ada
```

```
## [1] 0.5357524
```

```r
#misclassification error
test.err.ada = 1-(sum(diag(cm.ada))/sum(cm.ada))
test.err.ada
```

```
## [1] 0.08958485
```

XGBoost

```r
#XGBoost
set.seed(8)
x.trainXG =model.matrix(data=train,y~.-1)
y.trainXG = rep(1, nrow(train))
y.trainXG[train$y=="no"]=0 #for XGBoost

x.testXG = model.matrix(data=test, y~.-1)
y.testXG = rep(1, nrow(test))
y.testXG[test$y=="no"]=0 #for XGBoost

xgb.cls <- xgboost(data=x.trainXG,label=y.trainXG,max_depth=5,eta=0.01,nrounds=500,verbose=FA
LSE)
#xgb.cls <- xgboost(data=x.trainXG,label=y.trainXG,max_depth=10,nrounds=500,verbose=FALSE)
xgb.cls
```

```
## ##### xgb.Booster
## raw: 1 Mb
## call:
##   xgb.train(params = params, data = dtrain, nrounds = nrounds,
##     watchlist = watchlist, verbose = verbose, print_every_n = print_every_n,
##     early_stopping_rounds = early_stopping_rounds, maximize = maximize,
##     save_period = save_period, save_name = save_name, xgb_model = xgb_model,
##     callbacks = callbacks, max_depth = 5, eta = 0.01)
## params (as set within xgb.train):
##   max_depth = "5", eta = "0.01", validate_parameters = "1"
## xgb.attributes:
##   niter
## callbacks:
##   cb.evaluation.log()
## # of features: 41
## niter: 500
## nfeatures : 41
## evaluation_log:
##     iter train_rmse
##        1   0.496162
##        2   0.492358
## ---
##      499   0.225786
##      500   0.225769
```

```
#predict using test data
xgb.pred.prob<-predict(xgb.cls,x.testXG)

xgb.pred<-as.numeric(xgb.pred.prob>0.5) #convert to 0 ("no") or 1 ("yes")

#confusion matrix
cm.xgb<-table(xgb.pred,y.testXG) #0 is "no", 1 is "yes"
cm.xgb
```

```
##         y.testXG
## xgb.pred    0    1
##        0 7057  428
##        1  244  509
```

```
TP <- cm.xgb[2,2]
TN <- cm.xgb[1,1]
FP <- cm.xgb[2,1]
FN <- cm.xgb[1,2]

#FPR / Type I error
FPR.xgb = FP/(FP+TN)
FPR.xgb
```

```
## [1] 0.03342008
```

```
#FNR / Type II error
FNR.xgb = FN/(FN+TP)
FNR.xgb
```

```
## [1] 0.4567769
```

```
#Precision
precis.xgb = TP/(TP+FP)
precis.xgb
```

```
## [1] 0.6759628
```

```
#Recall / sensitivity
recall.xgb = TP/(TP+FN)
recall.xgb
```

```
## [1] 0.5432231
```

```
#misclassification error
test.err.xgb = 1-sum(diag(cm.xgb))/sum(cm.xgb)
test.err.xgb
```

```
## [1] 0.0815732
```

SVM with linear kernel

```
set.seed(8)
svm.fit <- svm(y~., data=train, kernel='linear', cost=1)
#summary(svm.fit)

#CV for tuning the cost parameter
set.seed(8)
tune.out1 <- tune(svm, y~.,
              data=train,
              kernel="linear",
              )

#tune.out1 <- tune(svm, y~.,
#               data=train,
#               kernel="linear",
#               ranges=list(cost=c(0.01, 0.1, 1, 10, 100)),tunecontrol=tune.control(cross=10))
summary(tune.out1)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09729894
```

```
svm.lin.best <- tune.out1$best.model
summary(svm.lin.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  6635
##
##  ( 3324 3311 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
lin.pred <- predict(svm.lin.best, test)

#confusion matrix
cm.lin <- table(lin.pred, test$y)
cm.lin
```

```
##
## lin.pred   no  yes
##      no  7146  646
##      yes  155  291
```

```
TP <- cm.lin[2,2]
TN <- cm.lin[1,1]
FP <- cm.lin[2,1]
FN <- cm.lin[1,2]

#FPR / Type I error
FPR.lin = FP/(FP+TN)
FPR.lin
```

```
## [1] 0.02122997
```

```
#FNR / Type II error
FNR.lin = FN/(FN+TP)
FNR.lin
```

```
## [1] 0.6894344
```

```
#Precision
precis.lin = TP/(TP+FP)
precis.lin
```

```
## [1] 0.6524664
```

```
#Recall / sensitivity
recall.lin = TP/(TP+FN)
recall.lin
```

```
## [1] 0.3105656
```

```
#misclassification error
test.err.lin = 1-(sum(diag(cm.lin))/sum(cm.lin))
test.err.lin
```

```
## [1] 0.09723234
```

SVM with polynomial kernel

```
set.seed(8)
tune.out2 <- tune(svm, y~.,
              data=train,
              kernel="polynomial",
              )

#tune.out2 <- tune(svm, y~.,
#              data=train,
#              kernel="polynomial",
#              ranges=list(cost=c(0.1,1,5,10,15,20,50,100),
#                          degree=c(2,3,4)))
summary(tune.out2)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09456753
```

```
svm.poly.best <- tune.out2$best.model
summary(svm.poly.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  6712
##
##  ( 3399 3313 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
poly.pred <- predict(svm.poly.best, test)

#confusion matrix
cm.poly <- table(poly.pred, test$y)
cm.poly
```

```
##
## poly.pred   no  yes
##       no  7214  691
##       yes   87  246
```

```
TP <- cm.poly[2,2]
TN <- cm.poly[1,1]
FP <- cm.poly[2,1]
FN <- cm.poly[1,2]

#FPR / Type I error
FPR.poly = FP/(FP+TN)
FPR.poly
```

```
## [1] 0.01191618
```

```
#FNR / Type II error
FNR.poly = FN/(FN+TP)
FNR.poly
```

```
## [1] 0.73746
```

```
#Precision
precis.poly = TP/(TP+FP)
precis.poly
```

```
## [1] 0.7387387
```

```
#Recall / sensitivity
recall.poly = TP/(TP+FN)
recall.poly
```

```
## [1] 0.26254
```

```
#misclassification error
test.err.poly = 1-(sum(diag(cm.poly))/sum(cm.poly))
test.err.poly
```

```
## [1] 0.0944404
```

SVM with rbf kernel

```
set.seed(8)
tune.out3 <- tune(svm, y~.,
              data=train,
              kernel="radial",)

#tune.out3 <- tune(svm, y~.,
#              data=train,
#              kernel="radial",
#              ranges=list(cost=c(0.1,1,5,10),
#                          gamma=c(0.01,0.1,1,5,10,100)))
summary(tune.out3)
```

```
##
## Error estimation of 'svm' using 10-fold cross validation: 0.09125948
```

```
svm.rbf.best <- tune.out3$best.model
summary(svm.rbf.best)
```

```
##
## Call:
## best.tune(method = svm, train.x = y ~ ., data = train, kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  6573
##
##  ( 3334 3239 )
##
##
## Number of Classes:  2
##
## Levels:
##  no yes
```

```
#predict using test data
rbf.pred <- predict(svm.rbf.best, test)

#confusion matrix
cm.rbf <- table(rbf.pred, test$y)
cm.rbf
```

```
##
## rbf.pred   no  yes
##      no  7161  614
##      yes  140  323
```

```
TP <- cm.rbf[2,2]
TN <- cm.rbf[1,1]
FP <- cm.rbf[2,1]
FN <- cm.rbf[1,2]

#FPR / Type I error
FPR.rbf = FP/(FP+TN)
FPR.rbf
```

```
## [1] 0.01917546
```

```
#FNR / Type II error
FNR.rbf = FN/(FN+TP)
FNR.rbf
```

```
## [1] 0.6552828
```

```
#Precision
precis.rbf = TP/(TP+FP)
precis.rbf
```

```
## [1] 0.6976242
```

```
#Recall / sensitivity
recall.rbf = TP/(TP+FN)
recall.rbf
```

```
## [1] 0.3447172
```

```
#misclassification error
test.err.rbf = 1-(sum(diag(cm.rbf))/sum(cm.rbf))
test.err.rbf
```

```
## [1] 0.09152707
```

Result Summary

```
options(digits = 3)
cl.err <- matrix(c(test.err.knn,FNR.knn,precis.knn,recall.knn,
                   test.err.reg,FNR.reg,precis.reg,recall.reg,
                   test.err.tree1,FNR.tree1,precis.tree1,recall.tree1,
                   test.err.rf,FNR.rf,precis.rf,recall.rf,
                   test.err.gbm,FNR.gbm,precis.gbm,recall.gbm,
                   test.err.ada,FNR.ada,precis.ada,recall.ada,
                   test.err.xgb,FNR.xgb,precis.xgb,recall.xgb,
                   test.err.lin,FNR.lin,precis.lin,recall.lin,
                   test.err.poly,FNR.poly,precis.poly,recall.poly,
                   test.err.rbf,FNR.rbf,precis.rbf,recall.rbf),
                   ncol=4, byrow=TRUE)
colnames(cl.err) <- c('misclass error','type-II error','precision','recall')
rownames(cl.err) <- c('KNN',
                      'Logistic regression',
                      'Decision tree with rpart',
                      'Random forest',
                      'Gradient boosting',
                      'Adaboost',
                      'XGBoost',
                      'SVM with linear kernel',
                      'SVM with polynomial kernel',
                      'SVM with radial kernel')
as.table(cl.err)
```

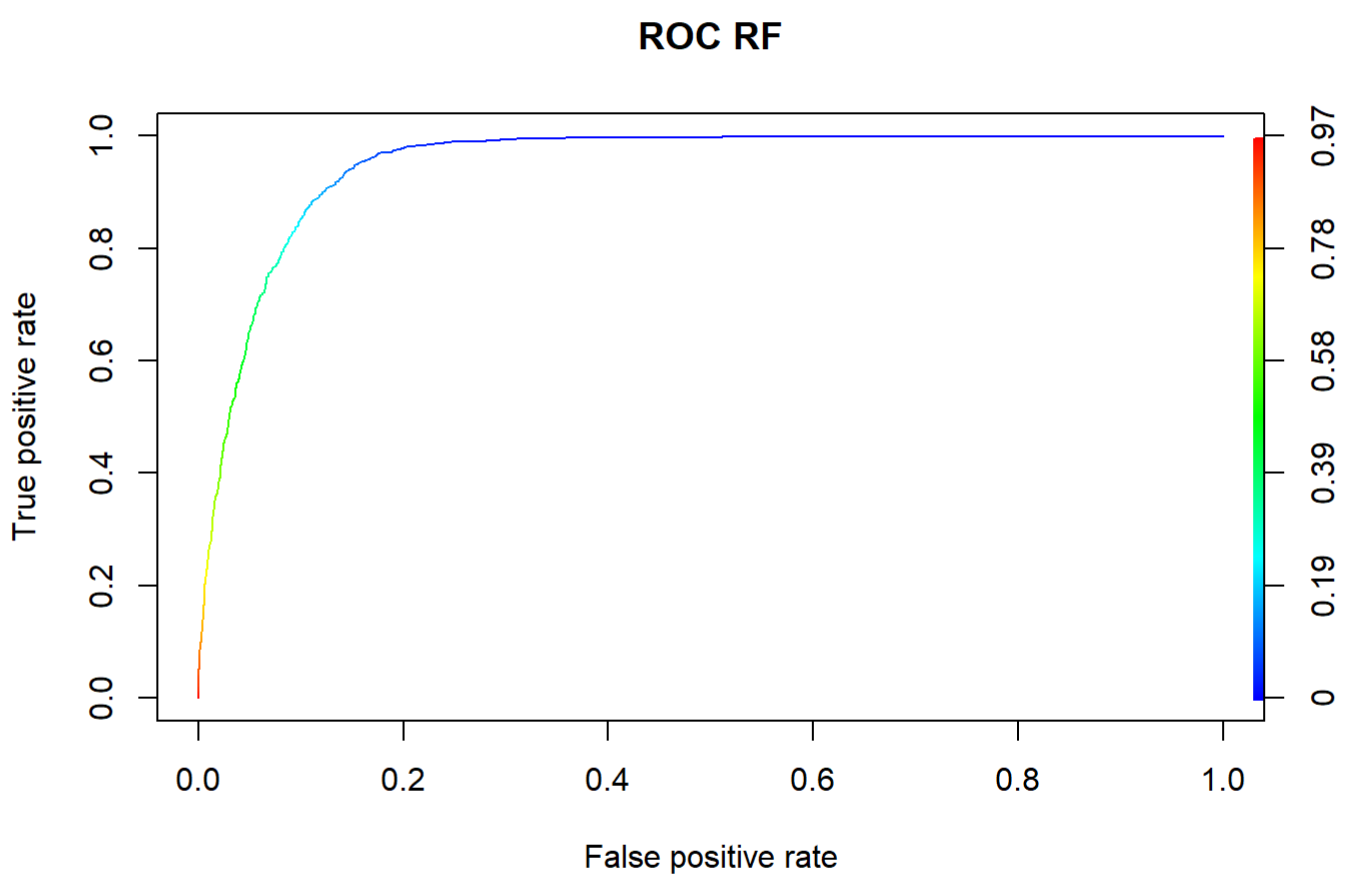
```
##                              misclass error type-II error precision recall
## KNN                                  0.0906        0.5560    0.6490 0.4440
## Logistic regression                  0.0893        0.5827    0.6730 0.4173
## Decision tree with rpart             0.0932        0.6531    0.6757 0.3469
## Random forest                        0.0839        0.4536    0.6581 0.5464
## Gradient boosting                    0.0819        0.4578    0.6737 0.5422
## Adaboost                             0.0896        0.4642    0.6236 0.5358
## XGBoost                              0.0816        0.4568    0.6760 0.5432
## SVM with linear kernel               0.0972        0.6894    0.6525 0.3106
## SVM with polynomial kernel           0.0944        0.7375    0.7387 0.2625
## SVM with radial kernel               0.0915        0.6553    0.6976 0.3447
```

Based on Type-II error comparison, best models are shortlisted: Random Forest, XGBoost, Adaboost, Gradient boosting.

ROC and AUC

Random Forest

```
#Prepare model for ROC curve
rf.pred <- predict(rf.cls, newdata = test, type = "prob")

forestpred = prediction(rf.pred[,2], test$y)

roc.perf.rf = performance(forestpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.rf, main='ROC RF', colorize=T)
```



```r
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.rf, forestpred))
roc.result = as.data.frame((opt.cut(roc.perf.rf, forestpred)))
roc.result
```

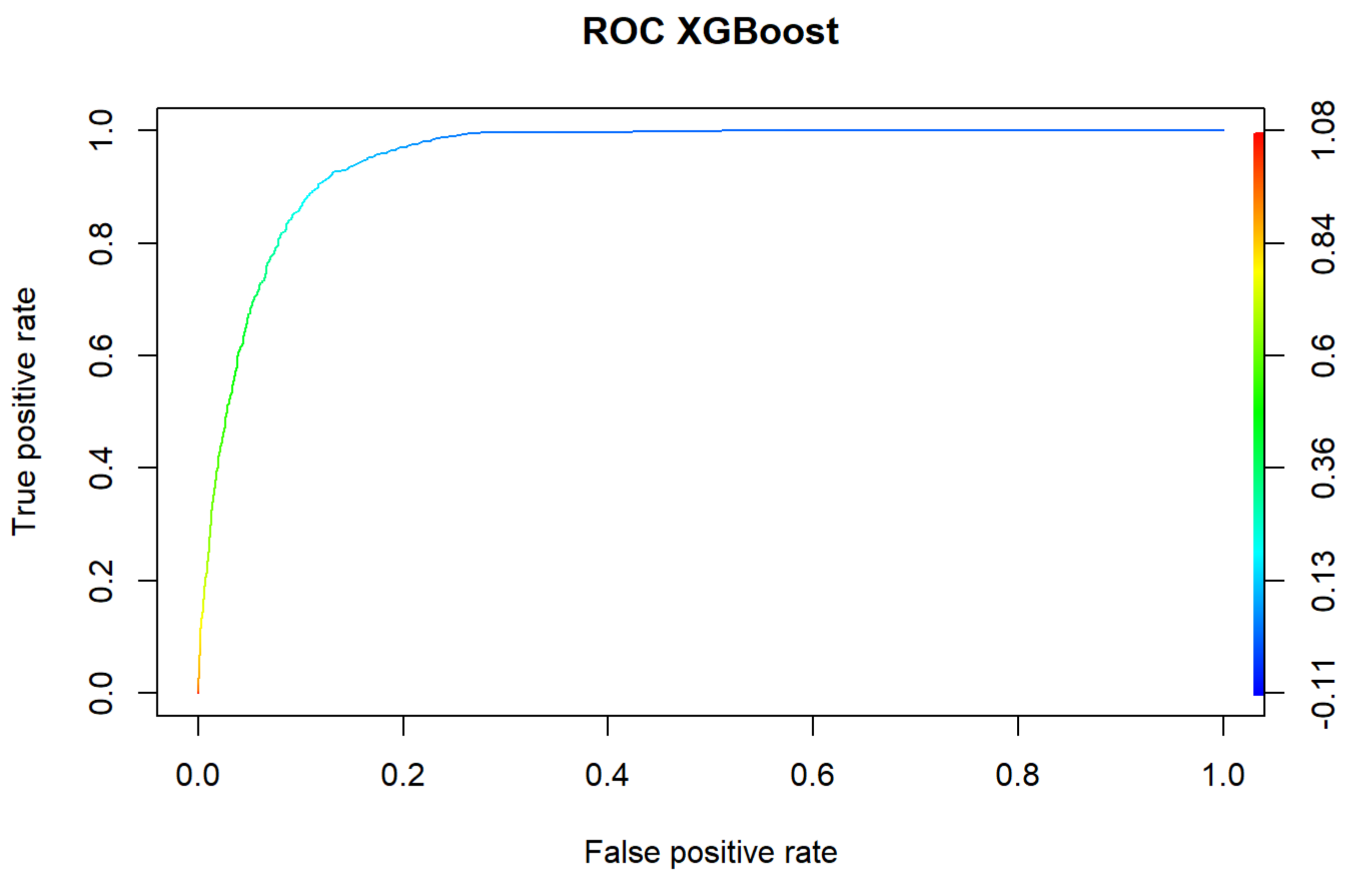
```
##                V1
## sensitivity 0.908
## specificity 0.875
## cutoff      0.154
```

```r
rf.sens = roc.result[1,]
rf.spec = roc.result[2,]
rf.cutoff = roc.result[3,]

auc.perf.rf = performance(forestpred, measure = 'auc')
auc.rf = auc.perf.rf@y.values
auc.rf
```

```
## [[1]]
## [1] 0.951
```

XGBoost

```r
#Prepare model for ROC curve
xgbpred = prediction(xgb.pred.prob, test$y)

roc.perf.xgb = performance(xgbpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.xgb, main='ROC XGBoost', colorize=T)
```




```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.xgb, xgbpred))
roc.result = as.data.frame((opt.cut(roc.perf.xgb, xgbpred)))
roc.result
```

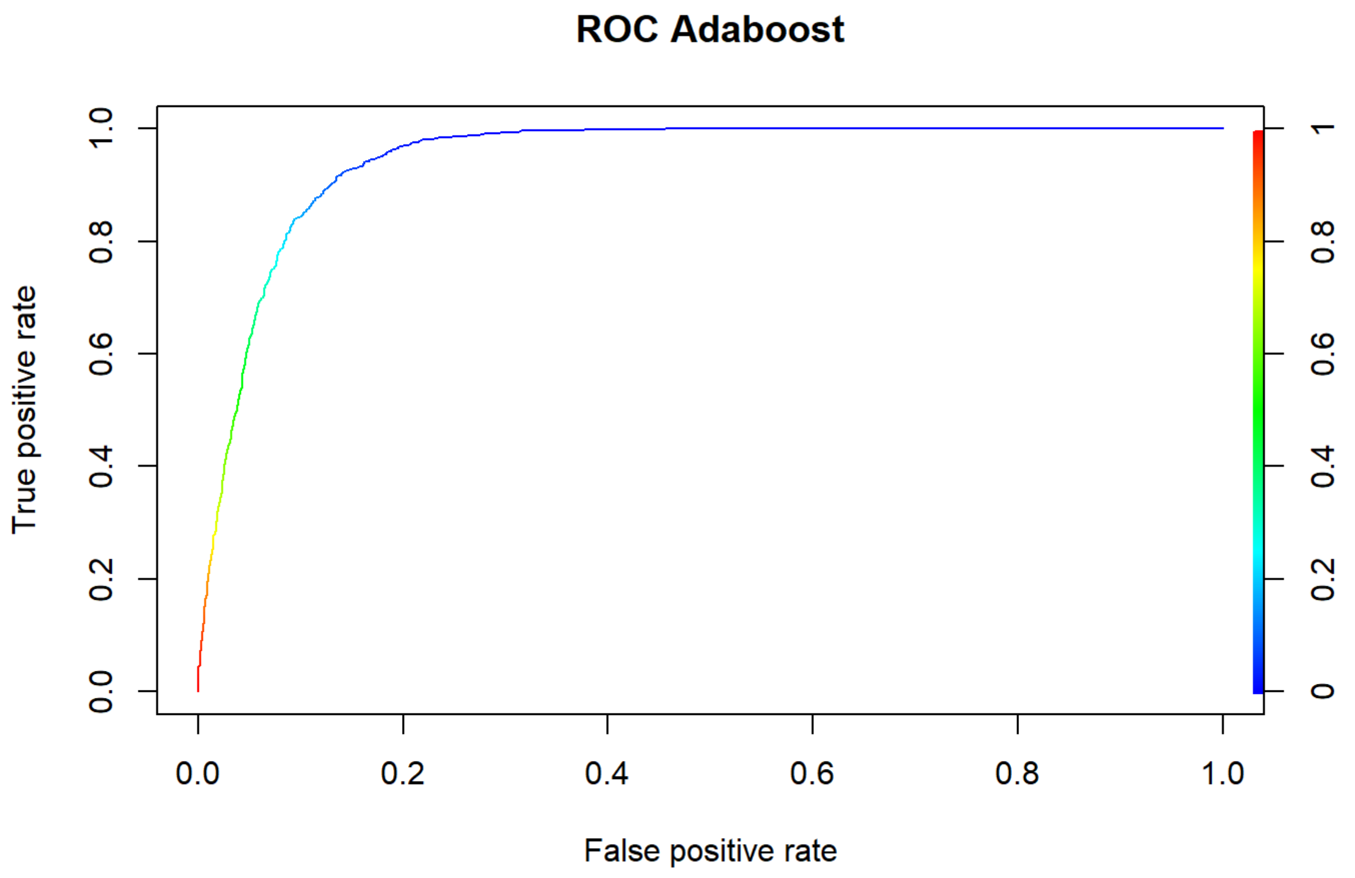
```
##                V1
## sensitivity 0.906
## specificity 0.883
## cutoff      0.166
```

```
xgb.sens = roc.result[1,]
xgb.spec = roc.result[2,]
xgb.cutoff = roc.result[3,]

auc.perf.xgb = performance(xgbpred, measure = 'auc')
auc.xgb = auc.perf.xgb@y.values
auc.xgb
```

```
## [[1]]
## [1] 0.953
```

Adaboost

```
#Prepare model for ROC curve
ada.pred <- predict(ada.cls, x.testA, type = "prob")

adapred = prediction(ada.pred, test$y)

roc.perf.ada = performance(adapred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.ada, main='ROC Adaboost', colorize=T)
```




```
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.ada, adapred))
roc.result = as.data.frame((opt.cut(roc.perf.ada, adapred)))
roc.result
```

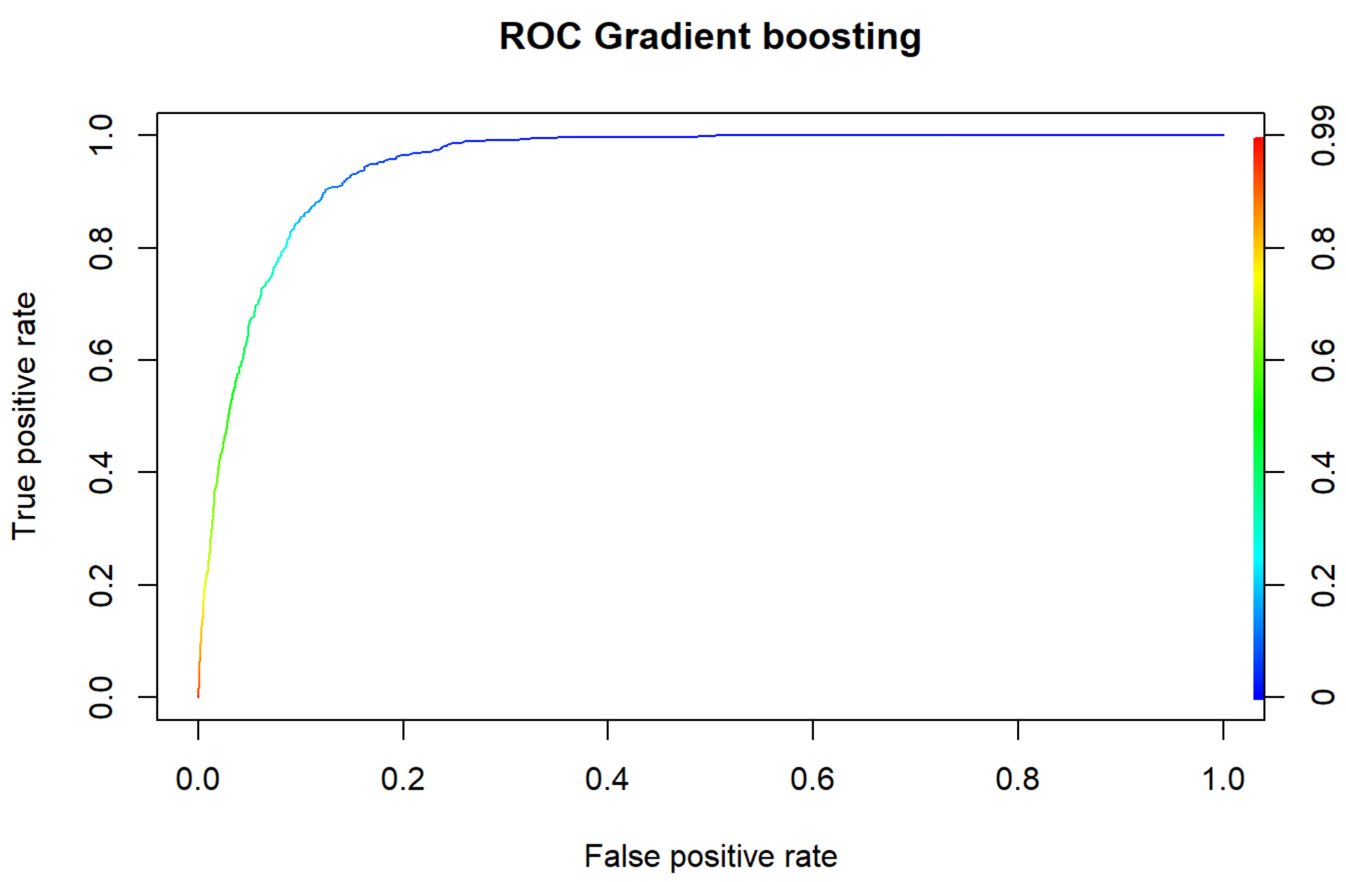
```
##                    V1
## sensitivity 0.9168
## specificity 0.8645
## cutoff      0.0876
```

```
ada.sens = roc.result[1,]
ada.spec = roc.result[2,]
ada.cutoff = roc.result[3,]

auc.perf.ada = performance(adapred, measure = 'auc')
auc.ada = auc.perf.ada@y.values
auc.ada
```

```
## [[1]]
## [1] 0.946
```

Gradient boosting

```
#Prepare model for ROC curve
gbm.pred <- predict (gbm.fit, test, type = "prob")

gbmpred = prediction(gbm.pred[,2], test$y)

roc.perf.gbm = performance(gbmpred, measure = "tpr", x.measure = "fpr")
plot(roc.perf.gbm, main='ROC Gradient boosting', colorize=T)
```



```r
#Optimal cutoff
opt.cut = function(perf, pred){
    cut.ind = mapply(FUN=function(x, y, p){
        d = (x - 0)^2 + (y-1)^2
        ind = which(d == min(d))
        c(sensitivity = y[[ind]], specificity = 1-x[[ind]],
            cutoff = p[[ind]])
    }, perf@x.values, perf@y.values, pred@cutoffs)
}
#print(opt.cut(roc.perf.gbm, gbmpred))
roc.result = as.data.frame((opt.cut(roc.perf.gbm, gbmpred)))
roc.result
```

```
##                V1
## sensitivity 0.904
## specificity 0.876
## cutoff      0.136
```

```r
gbm.sens = roc.result[1,]
gbm.spec = roc.result[2,]
gbm.cutoff = roc.result[3,]

auc.perf.gbm = performance(gbmpred, measure = 'auc')
auc.gbm = auc.perf.gbm@y.values
auc.gbm
```

```
## [[1]]
## [1] 0.949
```

AUC Summary

```r
options(digits = 3)
cl.err <- matrix(c(
                   test.err.rf, FNR.rf, rf.cutoff, (1-rf.sens), auc.rf,
                   test.err.xgb, FNR.xgb, xgb.cutoff, (1-xgb.sens), auc.xgb,
                   test.err.ada, FNR.ada, ada.cutoff, (1-ada.sens), auc.ada,
                   test.err.gbm, FNR.gbm, gbm.cutoff, (1-gbm.sens), auc.gbm
                   ),
                   ncol=5, byrow=TRUE)
colnames(cl.err) <- c('misclass err','Type-II err@0.5', 'cutoff', 'Type-II err@cutoff','AUC')
rownames(cl.err) <- c(
                      'RandomForest',
                      'XGBoost',
                      'Adaboost',
                      'Gradboost')
as.matrix(cl.err)
```

```
##              misclass err Type-II err@0.5 cutoff Type-II err@cutoff AUC
## RandomForest 0.0839       0.454           0.154  0.0918             0.951
## XGBoost      0.0816       0.457           0.166  0.0939             0.953
## Adaboost     0.0896       0.464           0.0876 0.0832             0.946
## Gradboost    0.0819       0.458           0.136  0.0961             0.949
```