Bank Marketing

Final EDX project of Harvard Data Science project Marketing

Javad Sahraei

4/1/2021

Contents

Introduction

This is the final project of professional course of data science powered by HarvardX. The topic I chose is "Bank Marketing" that the final goal is to predict which customer will accept the term deposit or not? For this, First, we will explore the data to get some insight from, and then we will use a supervised classification model.

Explaratory Data Analysis

Before doing anything, we need to know about our data. First, we will download the data and prepare it, then we will do exploratory data analysis.

Data preparation

The data source is provided by UCI (Center for Machine Learning and Intelligent Systems). Below chunk of code will download the data.

```
# Loading the data
temp <- tempfile()
url <- "https://archive.ics.uci.edu/ml/machine-learning-databases/00222/bank.zip"

download.file(url = url,
              destfile = temp)


data <- readr::read_delim(unz(temp, "bank-full.csv"), delim =";")
```

Let's have a glance at data:

```
glimpse(data)
```

```
## Rows: 45,211
## Columns: 17
## $ age       <dbl> 58, 44, 33, 47, 33, 35, 28, 42, 58, 43, 41, 29, 53, 58, 57, ~
## $ job       <chr> "management", "technician", "entrepreneur", "blue-collar", "~
## $ marital   <chr> "married", "single", "married", "married", "single", "marrie~
## $ education <chr> "tertiary", "secondary", "secondary", "unknown", "unknown", ~
## $ default   <chr> "no", "no", "no", "no", "no", "no", "no", "yes", "no", "no",~
## $ balance   <dbl> 2143, 29, 2, 1506, 1, 231, 447, 2, 121, 593, 270, 390, 6, 71~
## $ housing   <chr> "yes", "yes", "yes", "yes", "no", "yes", "yes", "yes", "yes"~
## $ loan      <chr> "no", "no", "yes", "no", "no", "no", "yes", "no", "no", "no"~
## $ contact   <chr> "unknown", "unknown", "unknown", "unknown", "unknown", "unkn~
## $ day       <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, ~
## $ month     <chr> "may", "may", "may", "may", "may", "may", "may", "may", "may~
## $ duration  <dbl> 261, 151, 76, 92, 198, 139, 217, 380, 50, 55, 222, 137, 517,~
## $ campaign  <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ pdays     <dbl> -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, ~
## $ previous  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ poutcome  <chr> "unknown", "unknown", "unknown", "unknown", "unknown", "unkn~
## $ y         <chr> "no", "no", "no", "no", "no", "no", "no", "no", "no", "no", ~
```

It seems that some of the columns are like factors but they are not truly defined, so below codes will change their type:

```
data$job = as.factor(data$job) # Type of job (Categorical)
data$marital = as.factor(data$marital)  # Maritial state (Categorical)
```

```r
data$education = as.factor(data$education)  # Education level
data$default = as.factor(data$default)   # Has credit in default? (Yes/No/Unknown)
data$housing = as.factor(data$housing)  # Has house loan? (Yes/No)
data$loan = as.factor(data$loan) # Has personal loan? (Yes/No/Unknown)
data$contact = as.factor(data$contact) # Type of contact (celluar, telephone)
data$month = as.factor(data$month) # Last month of contact
data$poutcome = as.factor(data$poutcome)  # outcome of the previous marketing campaign
data$y = as.factor(data$y)  # has the client subscribed a term deposit? (Target)
```

As a summary of what a column means, below you may find a descriptive summary:

- age: Age of the customer (Numeric)
- job: Job of the customer (Categorical)
- maritial: Maritial state of the customer (Categorical)
- education: Education level of the customer (Categorical)
- default: Does the customer has credit by default? (Categorical)
- balance: The amount of customer money. (Numeric)
- housing: Does the customer has house loan? (Categorical)
- loan: Does the customer has personal loan? (Categorical)
- contact: contact communication type (Categorical)
- day: last contact day of month (Categorical)
- month: last contact month of year (Categorical)
- duration: last contact duration, in seconds (Numeric)
- campaign: number of contacts performed during this campaign and for this client (Numeric)
- pdays: number of days that passed by after the client was last contacted from a previous campaign (Numeric)
- previous: number of contacts performed before this campaign and for this client (Numeric)
- poutcome: outcome of the previous marketing campaign (Categorical)
- y: has the client subscribed a term deposit? (Binary) *Predict variable*

Making test and train data sets

Now, we need to split data into two groups: One for model training (called **Train set**) and one for testing the final answers (called **Test set**). For doing this we will employ Caret package in R.

Insights from data

In this part, we will use the data sets to get some insightful information about the data.

Distribution of the customers by age

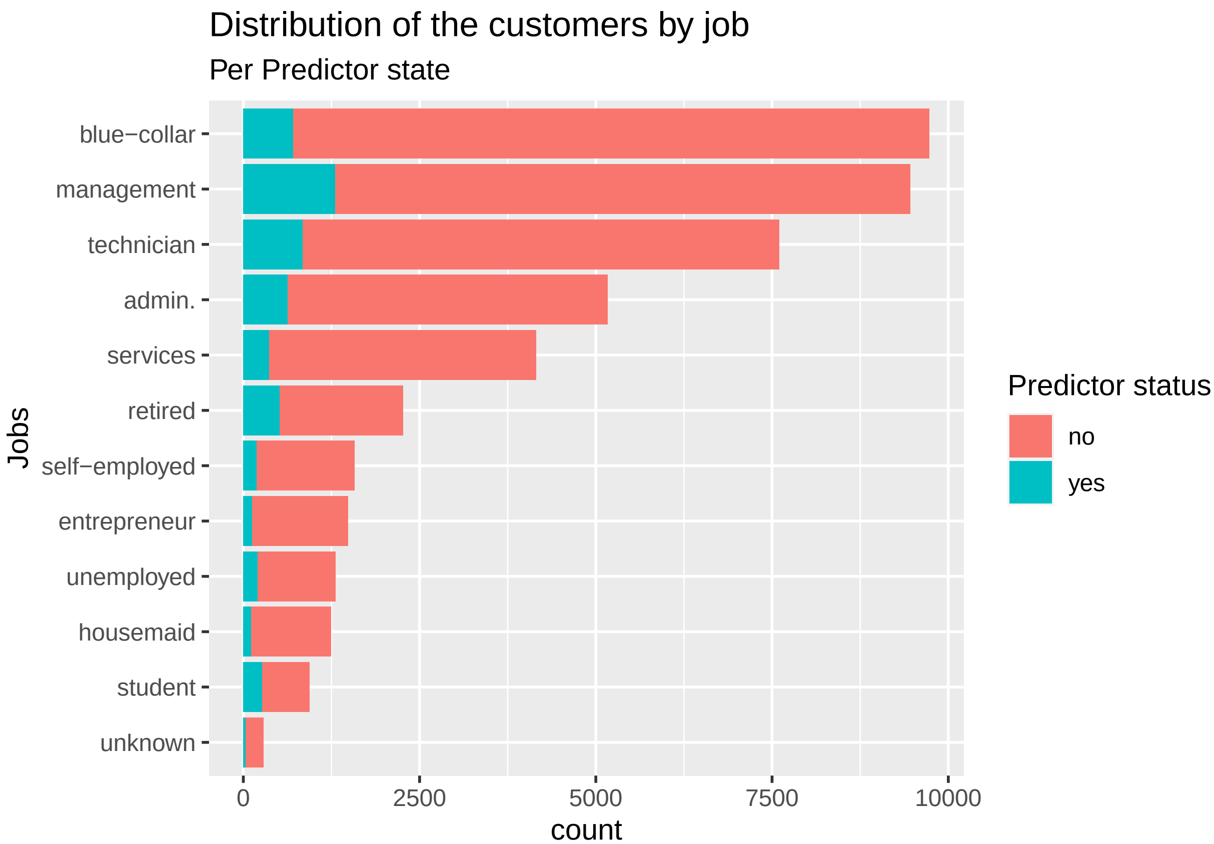
Below we will see the distribution of the customers by age and by their response to the offer.

It seems the chance of very old or very young customers to accept the offer is more than the customers in between.

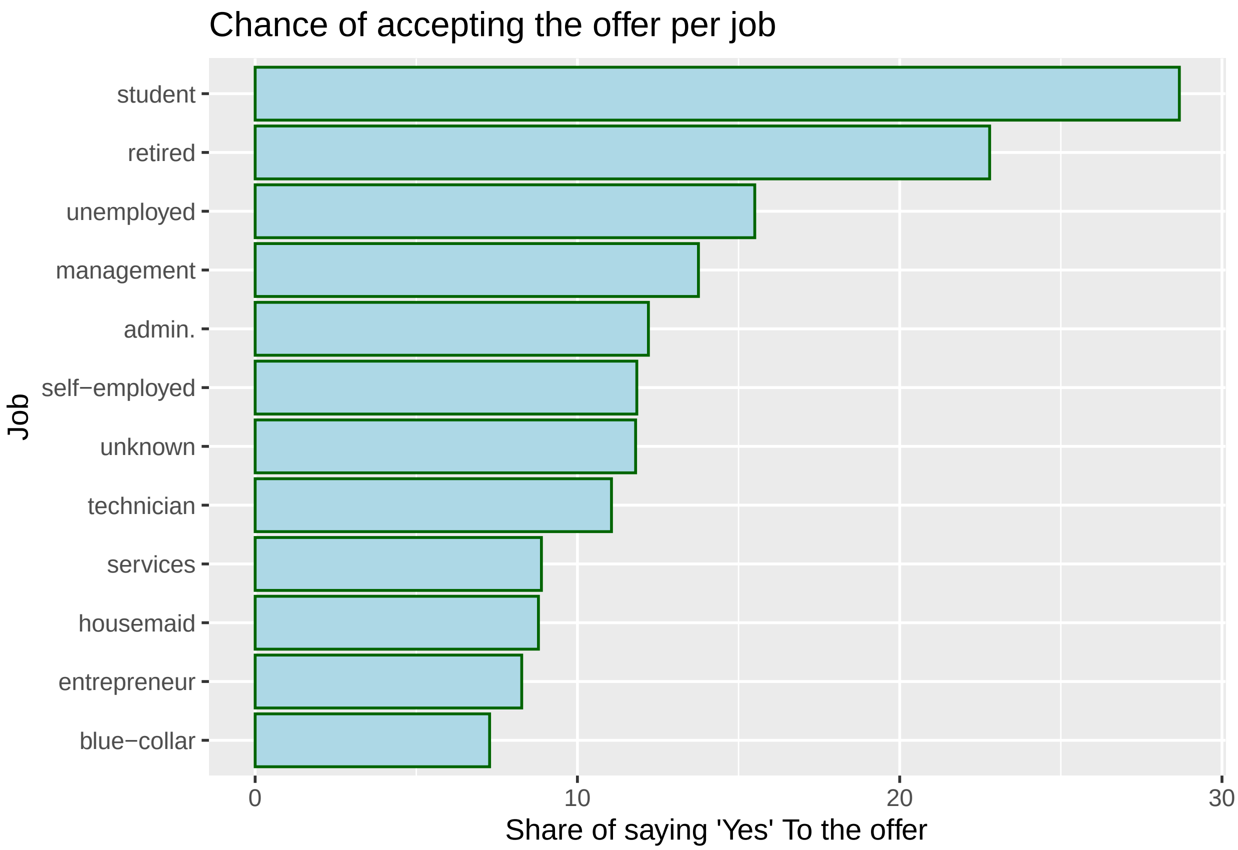
Customer's jobs

Let's look at the jobs and it's distribution over the offer.

```
## `summarise()` has grouped output by 'job'. You can override using the `.groups` argument.
```

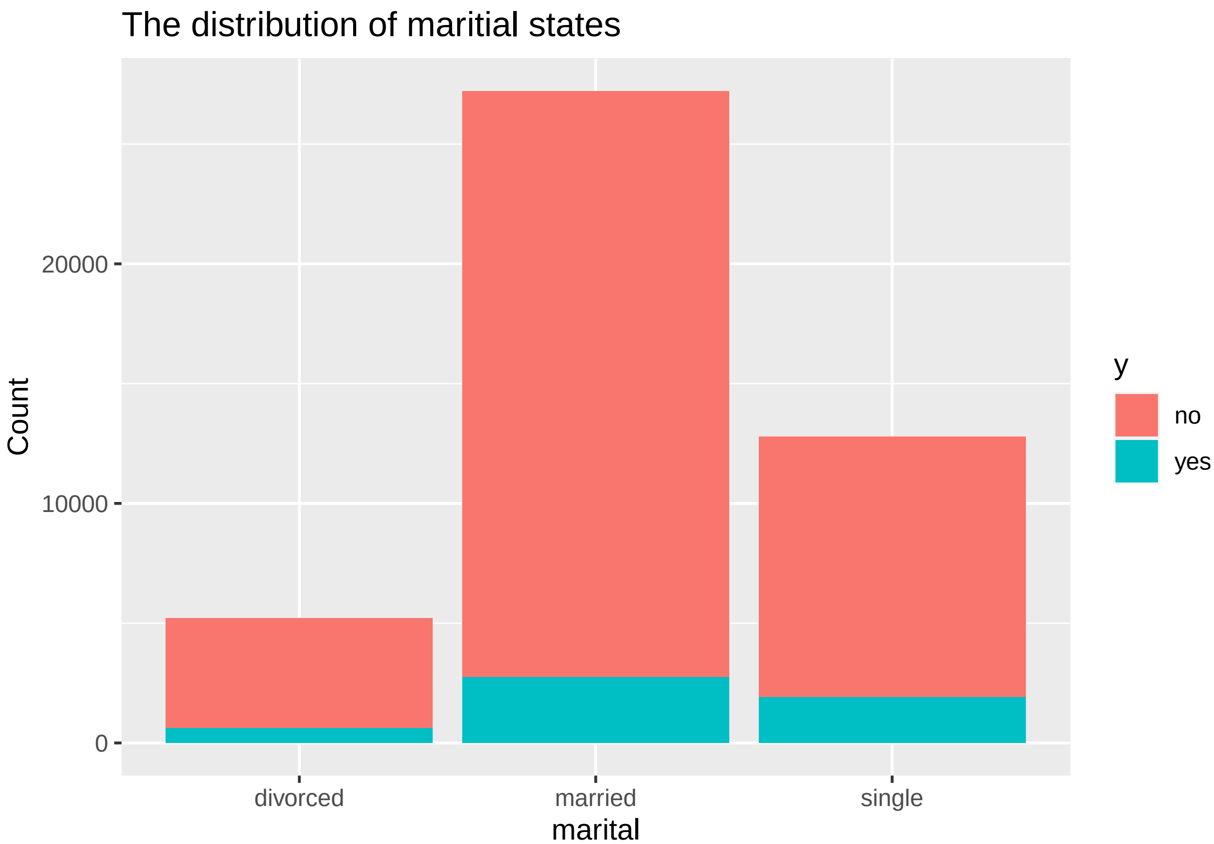



```r
data %>% filter(y == "yes") %>%
  group_by(job) %>%
  summarise(yes = n()) %>%
  left_join(y =
              data %>% group_by(job) %>%
              summarise(total = n()),
            by = "job") %>%
  mutate(Share = yes * 100 / total) %>%
  select(job, Share) %>%
  arrange(desc(Share)) %>%
  ggplot(aes(x = reorder(job, Share), y = Share)) +
  geom_bar(stat = "identity",color = "darkgreen",
           fill = "lightblue") +
  coord_flip() +
  labs(x = "Job",
       y = "Share of saying 'Yes' To the offer",
       title = "Chance of accepting the offer per job")
```



Maritial state

Let's see the distribution of maritial state and it's relationship with accepting the offer.

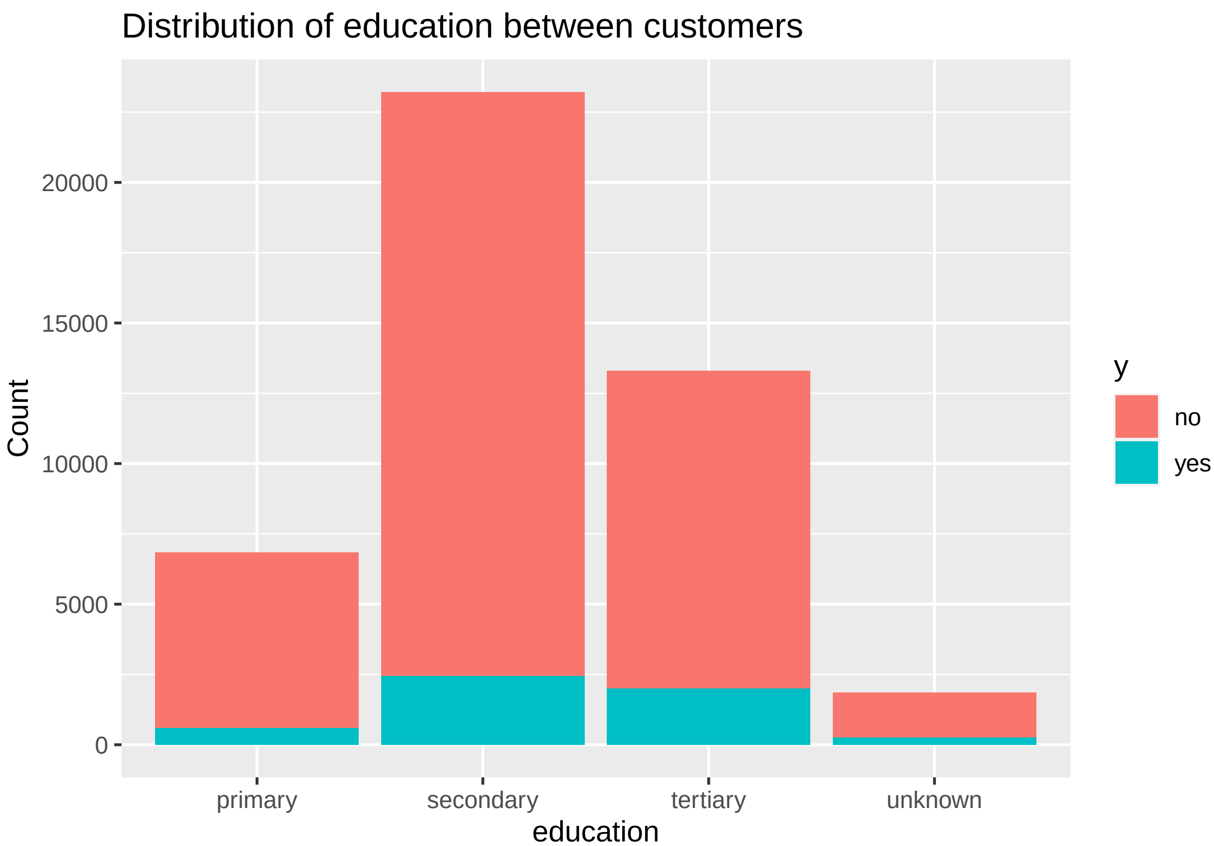


It seems that the maritial state is not important for predicting the final answer.

Education of the customers

Below charts shows how educated are the customers and what are the potential relationships between education and accepting the offer.

```
data %>%
  ggplot(aes(x = education, fill = y)) +
  geom_bar()+
  labs(title = "Distribution of education between customers",
       y = "Count")
```



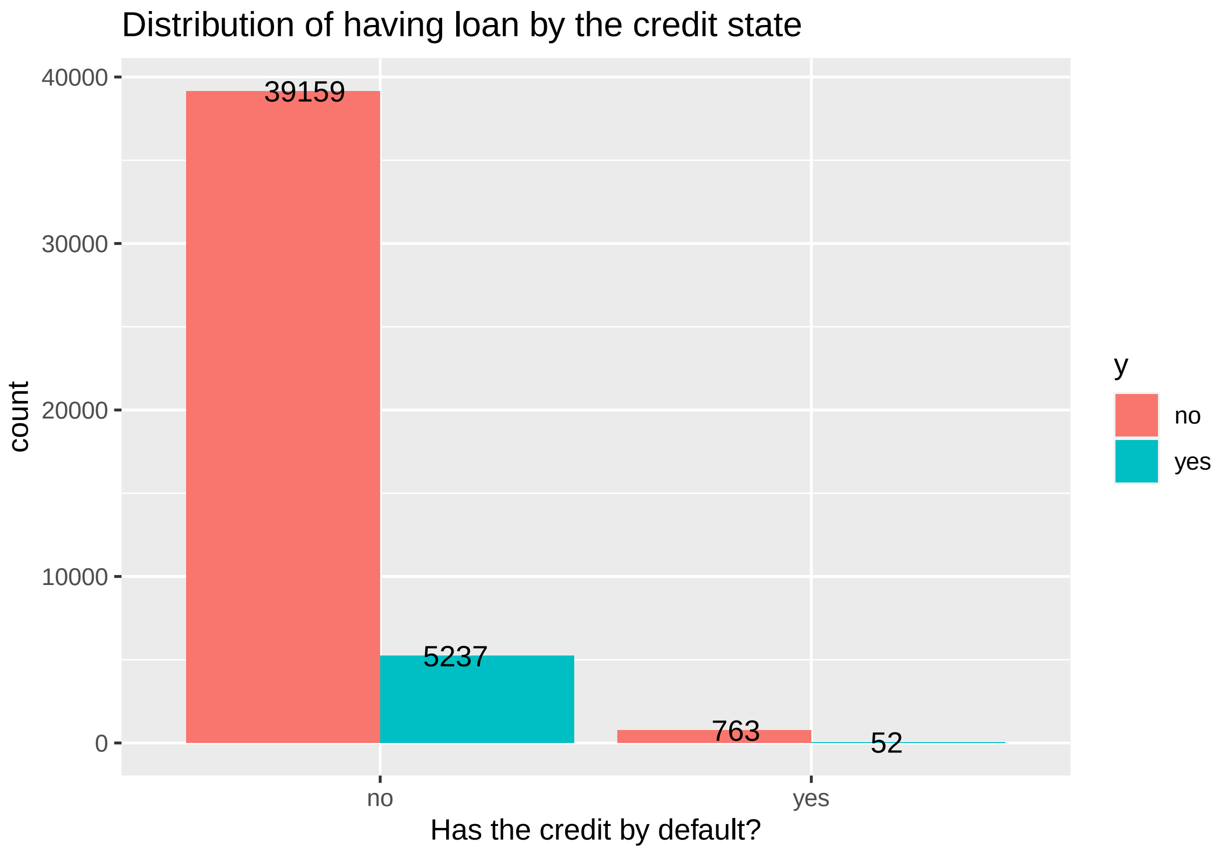

```
data %>%
  ggplot(aes(x = education, fill = y)) +
  geom_bar(position = "fill")+
  labs(title = "Chance of accepting the offer between customers",
       y = "Count")
```

It seems that there's some difference in chance of accepting the offer by different education level.

Having credit by default

Can having the credit affect on decision making? Let's see.

```
data %>%
  ggplot(aes(x = default, fill = y ))+
  geom_bar(position = "dodge")+
  labs(x = "Has the credit by default?",
       title = "Distribution of having loan by the credit state")+
  geom_text(aes(label = ..count..),
            stat = "count",
            position = position_dodge(width = 0.7))
```



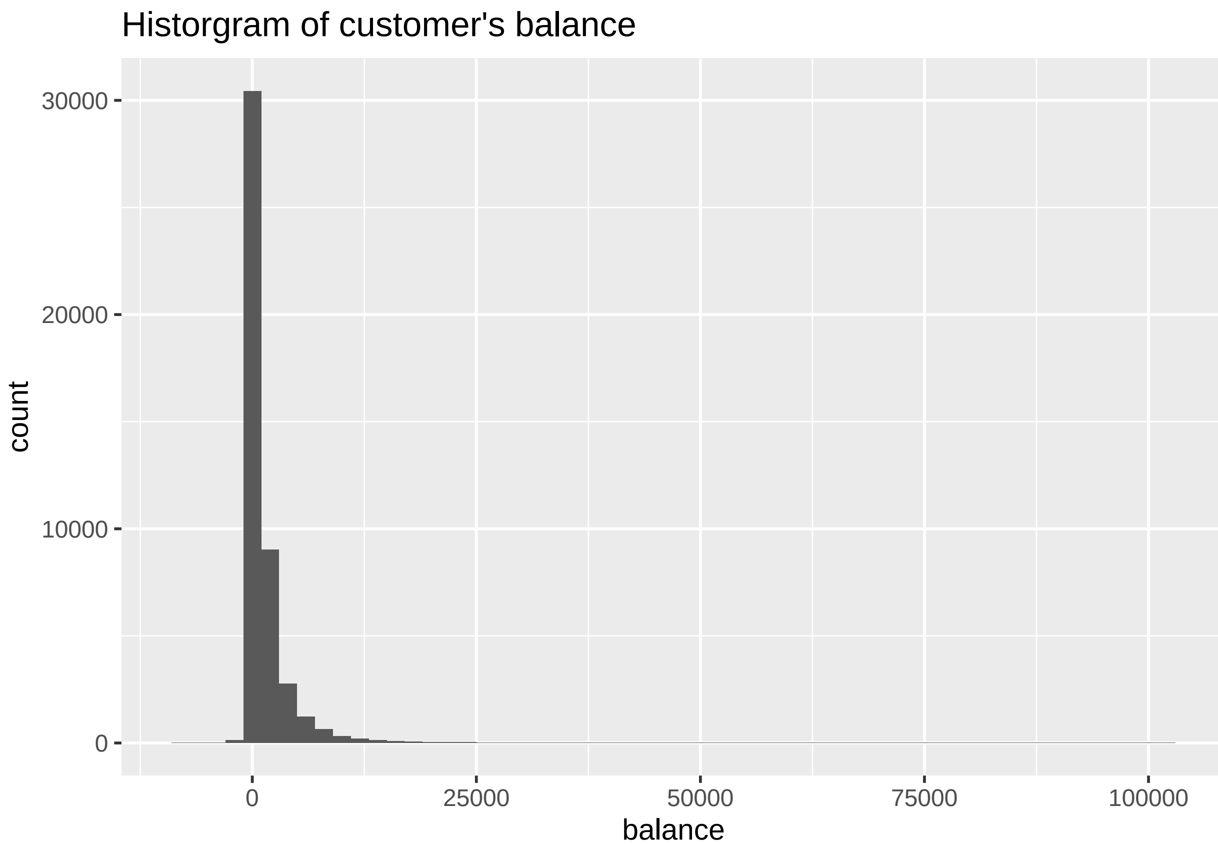

```
data %>%
  ggplot(aes(x = default, fill = y ))+
  geom_bar(position = "fill")+
  labs(x = "Has the credit by default?",
       title = "Chance of accepting the offer by having the credit")
```

The majority of customers do not have credit by default and it doesn't seem to have any effect on decision making.

Customer's balance

Let's see the distribution of money of the customers and see if there's any relationship between it and decision making.

```
data %>%
  ggplot(aes(x = balance))+
  geom_histogram(binwidth = 2000)+
  labs(title = "Historgram of customer's balance")
```



Let's take a look at some more detail of customer's money:

```
data$balance %>% summary()
```

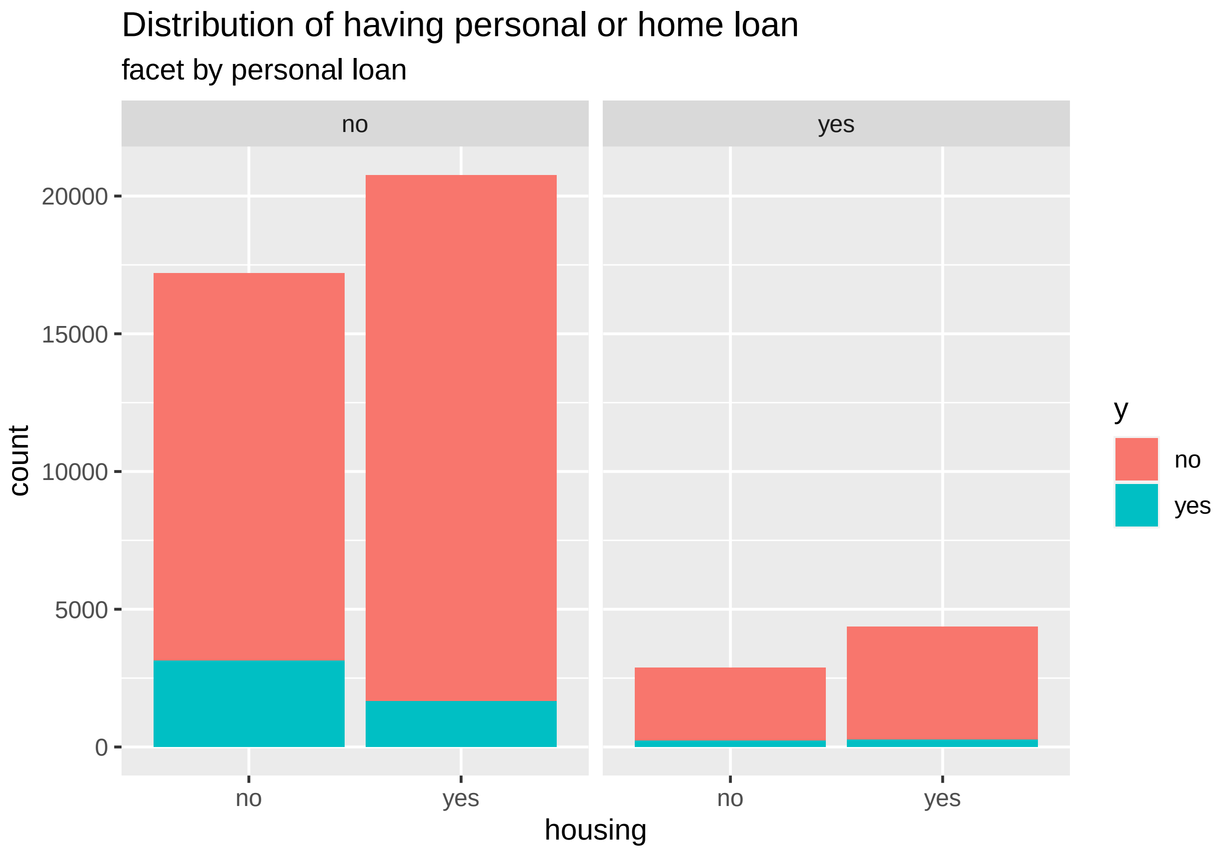
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -8019      72     448    1362    1428  102127
```

So the most of the people (75% of them) have less than 1400$.

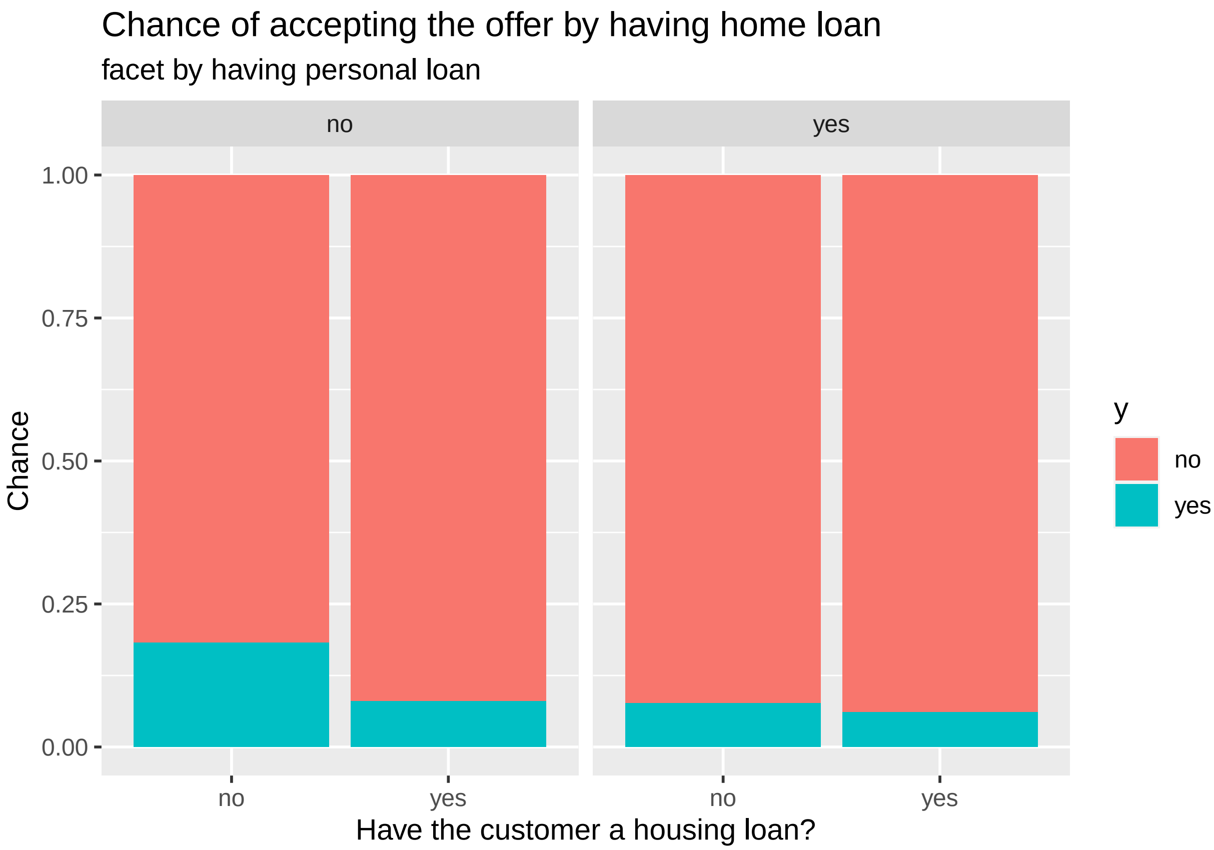
Housing and personal loan

Let's take a look at how many of the customers have personal or home loan:

```
data %>%
  ggplot(aes(x = housing, fill = y))+
  geom_bar()+
  facet_wrap(~ loan)+
  labs(title = "Distribution of having personal or home loan",
       subtitle = "facet by personal loan")
```




```r
data %>%
  ggplot(aes(x = housing, fill = y))+
  geom_bar(position = "fill")+
  facet_wrap(~ loan)+
  labs(title = "Chance of accepting the offer by having home loan",
       subtitle = "facet by having personal loan",
       x = "Have the customer a housing loan?",
       y = "Chance")
```

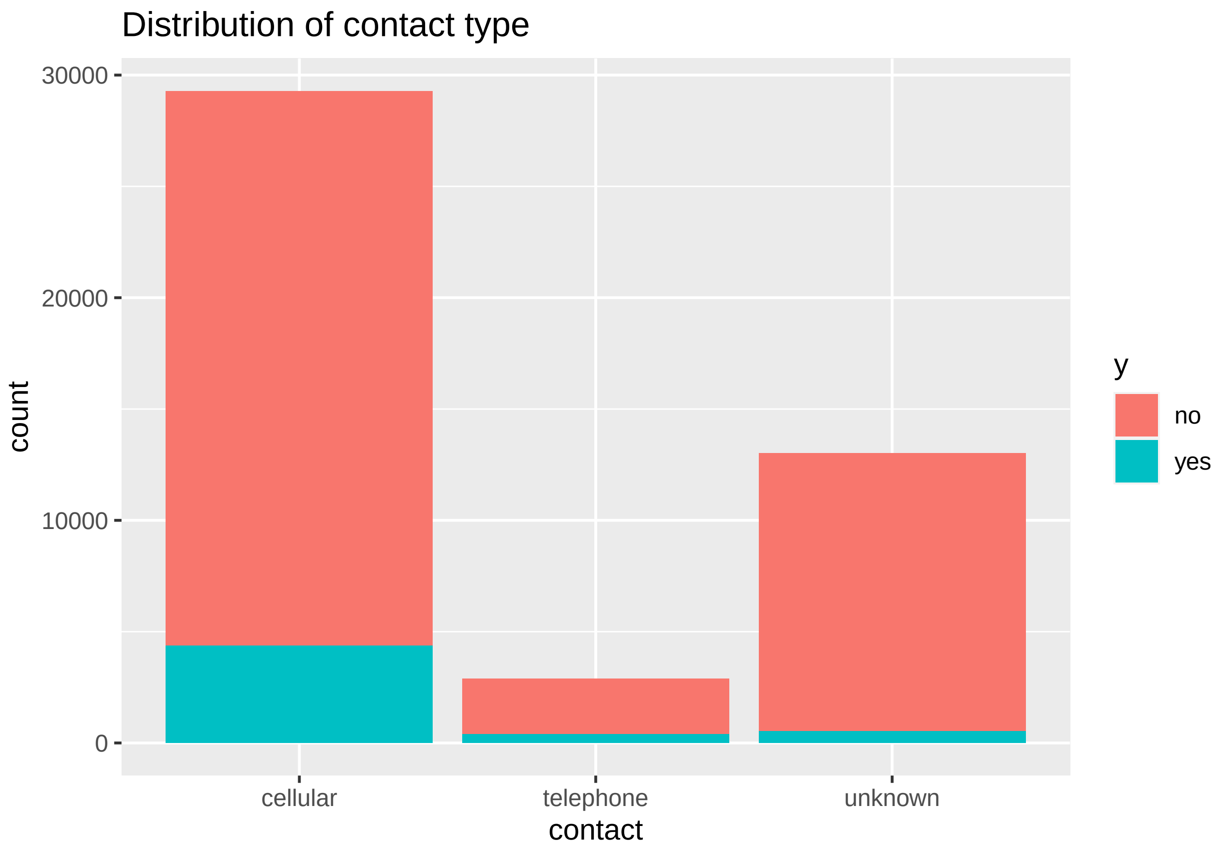


Generally, the chance of accepting the offer rises if the customer do not have any loan.

Contact type

Below we will see what are the preferred communication channels and how they are related to the accepting offers.

```
data %>%
  ggplot(aes(x = contact, fill = y))+
  geom_bar()+
  labs(title = "Distribution of contact type")
```



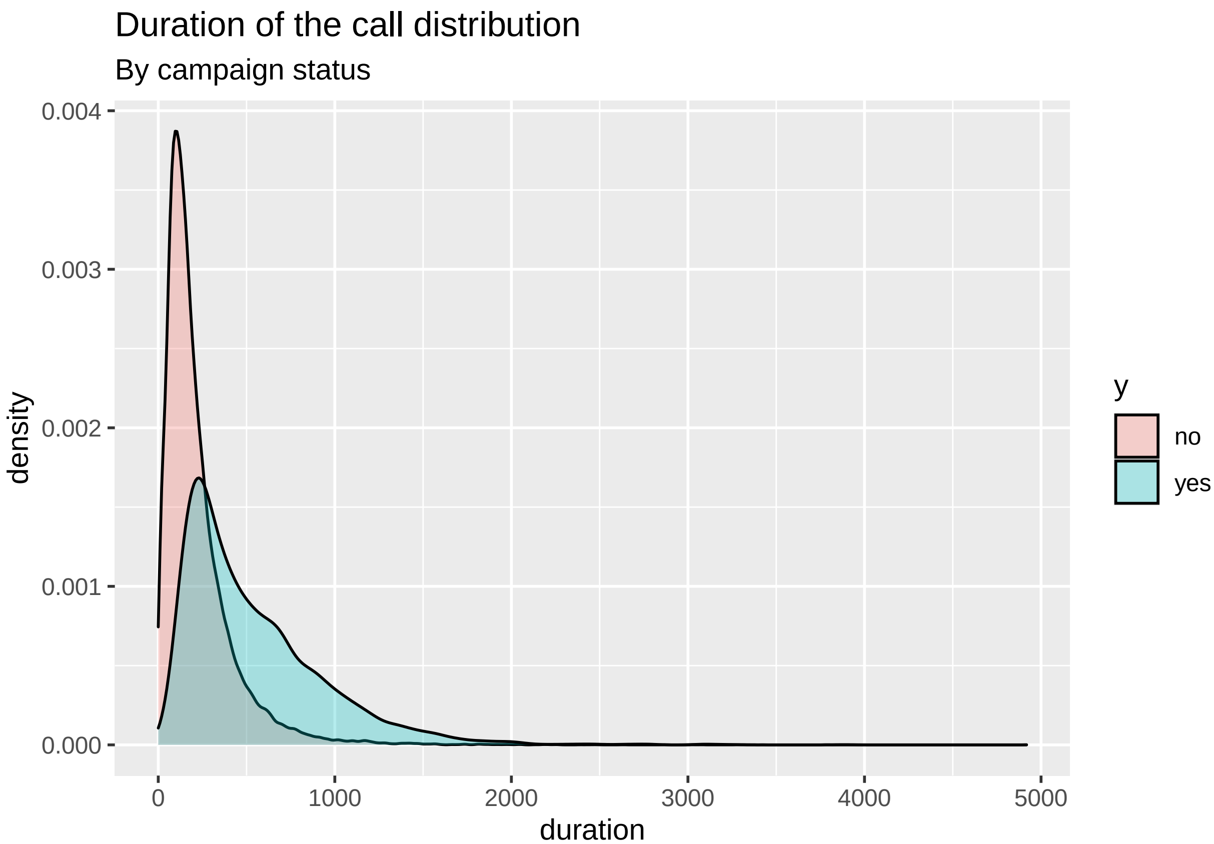

```
data %>%
  ggplot(aes(x = contact, fill = y))+
  geom_bar(position = "fill")+
  labs(title = "The chance of accepting the offer by contact type")
```

It seems that there's no relationship between contact tyoe and decision making.

Duration of the call

Let's see if duration of the call can affect on the decison making?

```
data %>%
  ggplot(aes(x = duration, fill = y))+
  geom_density(alpha = 0.3)+
  labs(title = "Duration of the call distribution",
       subtitle = "By campaign status")
```




```
# A glance at duration
data$duration %>% summary()
```

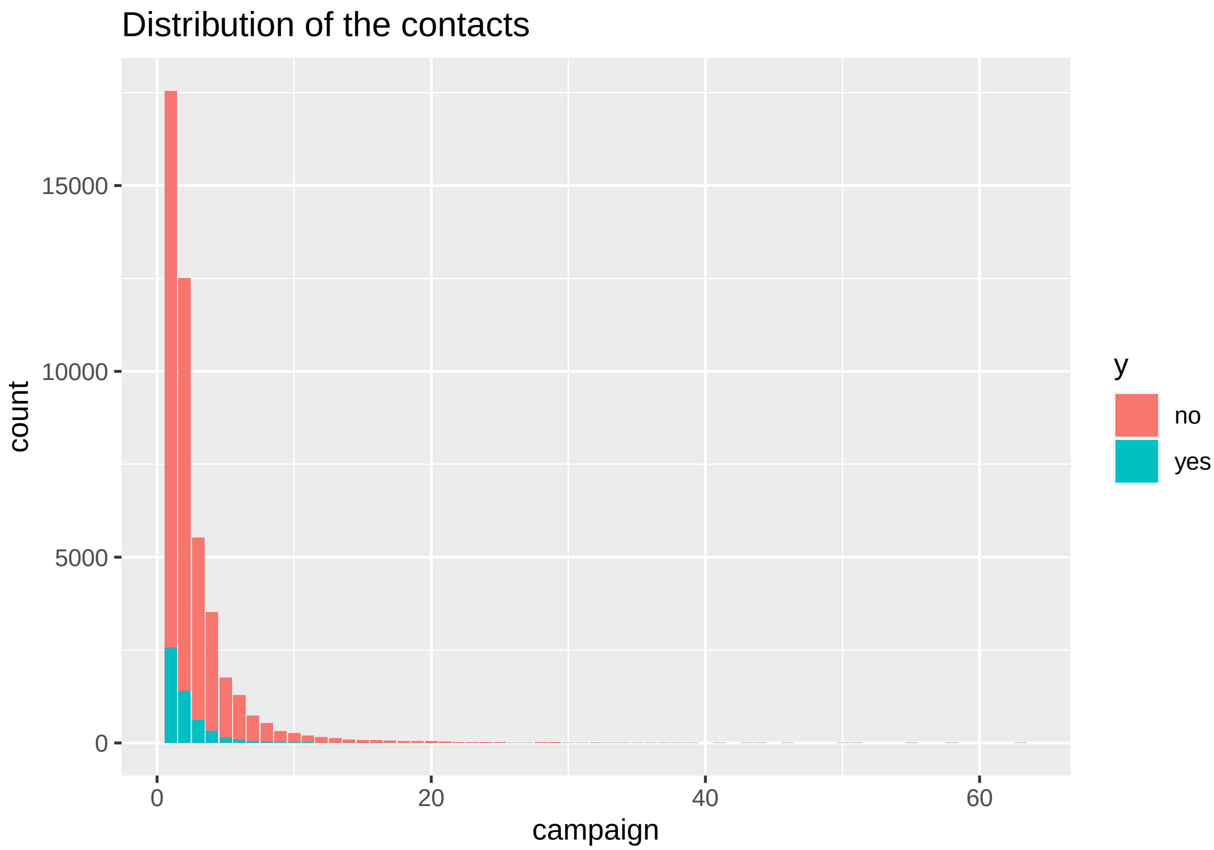
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0   103.0   180.0   258.2   319.0  4918.0
```

It seems if the call is too short, there's a very low chance for accepting the offer. And if the call lasts for more than a fixed time, the chance will raise.

Number of contacts per campaign

Let's see how number of contacts can affect on the results:

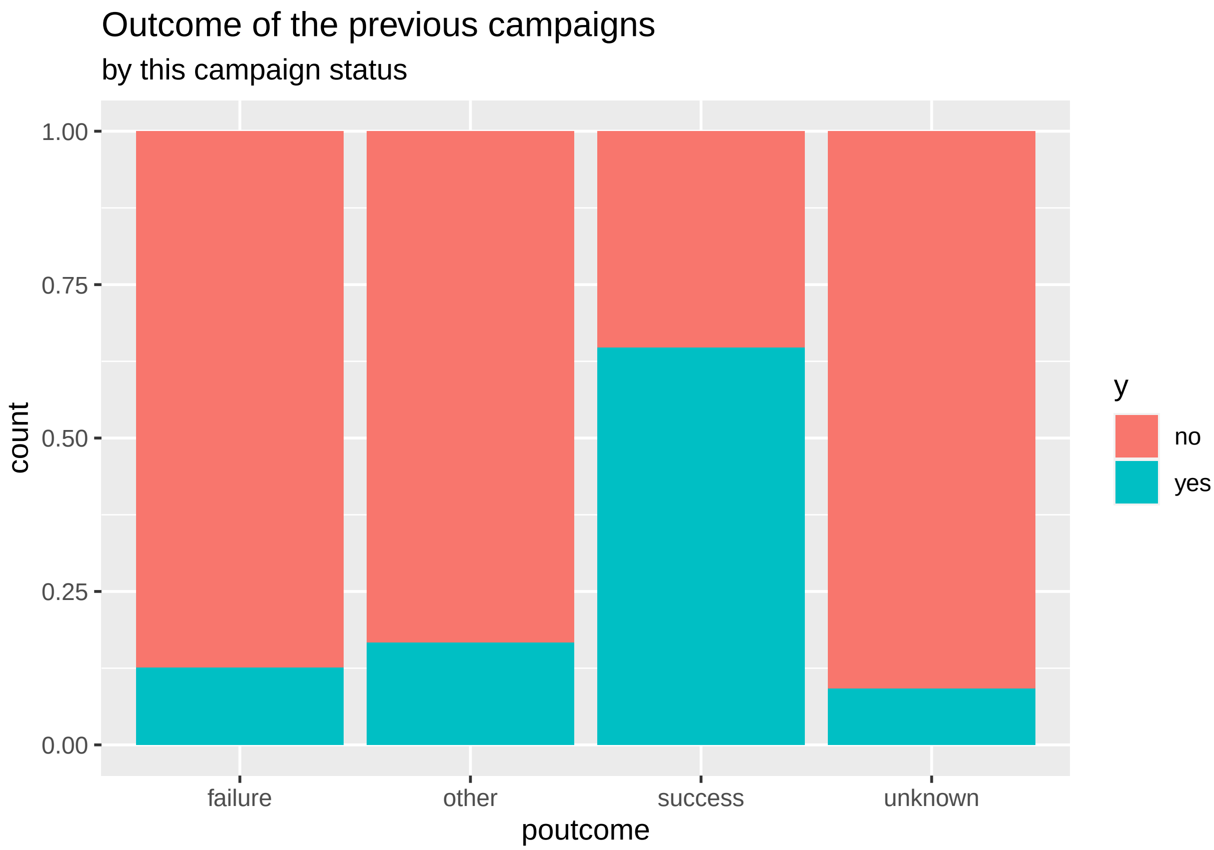
```
data %>%
  ggplot(aes(x = campaign, fill = y))+
  geom_bar(stat = "count")+
  labs(title = "Distribution of the contacts")
```



Outcome of the previous campaign

Below chart shows the distribution of last campaign outcome.

```
data %>%
  ggplot(aes(x = poutcome, fill = y))+
  geom_bar(position = "fill"
  )+
  labs(title = "Outcome of the previous campaigns",
       subtitle = "by this campaign status")
```



As shown above, the chance of accepting the offer for those customers who have accepted the previous one is very high.

Modelling

In this part, we will start building some machine learning techniques to predict which client will accept the offer or not. Four algorithems will be applied on the data as below:

- Decision tree
- Random Forest
- Naive Bayes

The mean misclassification error rate (mmce) performance measure is used during finetuning. In addition, paired t-test and confusion matrix are used to evaluate classifier's performance.

Sampling the data

In this part we will split the data into 70% (for training) and 30% (for test).

```
# Sampling data for test and train sets
library(mlr)
```

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

```
## Warning message: 'mlr' is in 'maintenance-only' mode since July 2019.
## Future development will only happen in 'mlr3'
## (<https://mlr3.mlr-org.com>). Due to the focus on 'mlr3' there might be
## uncaught bugs meanwhile in {mlr} - please consider switching.
```

```
##
```

```
## Attaching package: 'mlr'

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

```
#70% of the dataset
set.seed(1234)
smp_size <- floor(0.7*nrow(data))
set.seed(123)
train_index <- sample(seq(nrow(data)), size = smp_size)

#Assign to the train and test datasets
train <- data[train_index, ]
test <- data[-train_index, ]
```

Finetuning of hyperparameters

For the hyperparameter finetuning process, a 5-fold cross validation resampling strategy is applied.

```
# Configure classification task
classif.task <- makeClassifTask(data = train, target = 'y', id = 'bank')
```

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

```
# Configure tune control search and a 5-CV stratified sampling
ctrl  <- makeTuneControlGrid()
rdesc <- makeResampleDesc("CV", iters = 5L, stratify = TRUE)
```

The object classif.task has summarised key point information we have in our training dataset. Our target feature is y containing binary responses with respectively their distribution "No" (27938) and "Yes" (3709).

Decision Tree

Two arguments set for Decision Tree are "minsplit" and "maxdepth". Suggested by getParamSet(), we respectively set "maxdepth" and "minspit" the set of sequence from 1 to 30. We find that the optimal results for Decision Tree are when maxdepth=28; minsplit=28 : mmce.test.mean=0.0989351.

From the plot, we can see that it is until 2-3 iterations that the mmce drops and stablises.

```
# Configure learners with probability type
learner1 <- makeLearner('classif.rpart', predict.type = 'prob')

# Obtain parameters available for fine-tuning
getParamSet(learner1)
```

```
##                    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     -
```

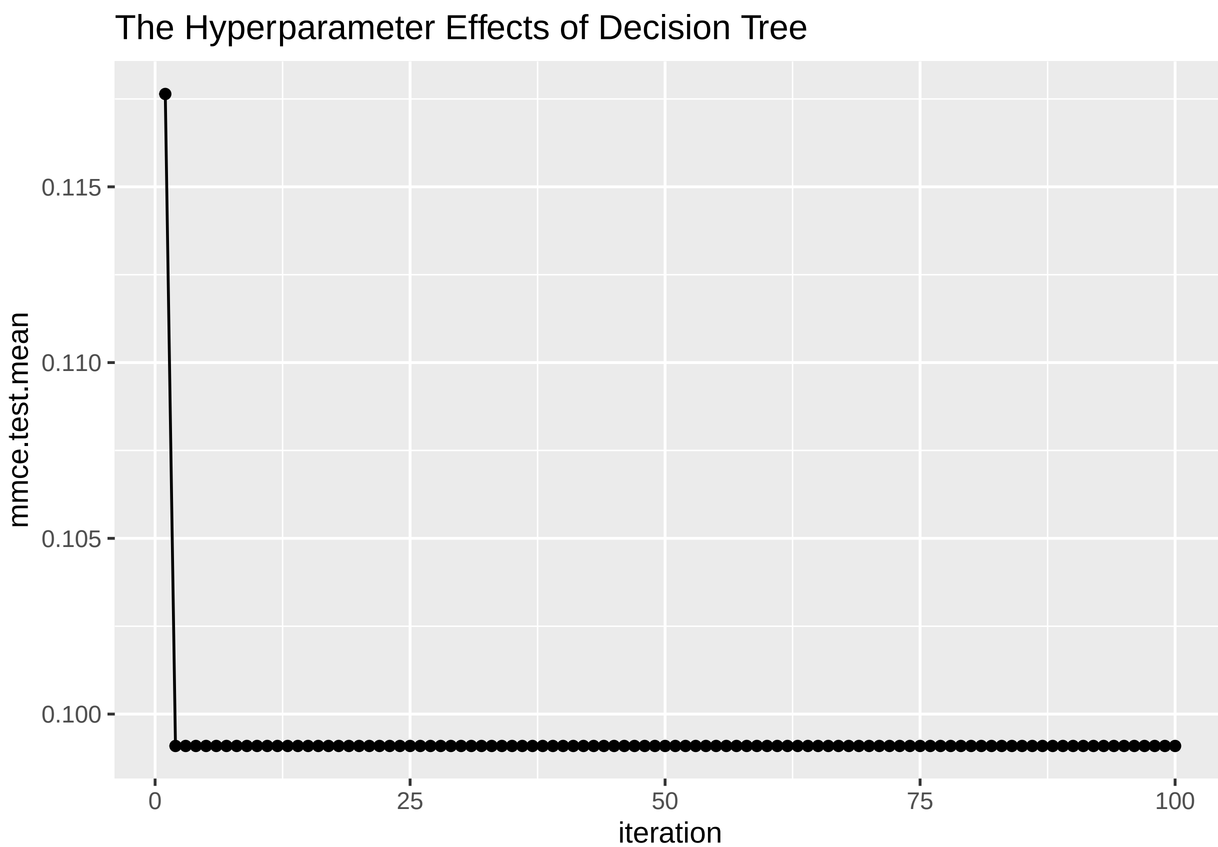
```
## parms              untyped   -    -         -   -     TRUE      -
```

```r
# Make Param Set
ps1 <- makeParamSet(
  makeDiscreteParam('maxdepth', values = c(seq(1,30,3))),
  makeDiscreteParam('minsplit', values = c(seq(1,30,3))))

# Configure tune Params settings
tunedLearner1_tuneparams <- tuneParams(learner = learner1,
                            task = classif.task,
                            resampling = rdesc,
                            par.set = ps1,
                            control = ctrl,
                            show.info =FALSE
                            )


# Getting the hyper parameter effects:
learner1_effect <- generateHyperParsEffectData(tunedLearner1_tuneparams)

#Plot the effect
plotHyperParsEffect(learner1_effect, x = "iteration", y = "mmce.test.mean", plot.type = "line") +
ggtitle("The Hyperparameter Effects of Decision Tree")
```




```r
# Making the tuned model:
tunedLearner1 <- setHyperPars(learner1, par.vals = tunedLearner1_tuneparams$x)

# Train the tune wrappers
```

```r
tunedMod1  <- train(tunedLearner1, classif.task)

# Predict on training data
tunedPred1 <- predict(tunedMod1, classif.task)
```

Random Forest

We fine-tune the number of features randomly sampled as candidates at each split (i.e. mtry). For a classification problem, learned that mtry is the square root of p where p is the number of descriptive features available in the dataset. In our case, square root of 13 is 3.6. Hence, we experimented the set of mtry = 2, 3, 4, 5, 6, which does not fall out of the range given by getParamSet. As for ntree argument, we set a sequence of values ranging from 10 to 100. The result is mtry=3; ntree=100 : mmce.test.mean=0.0987772.

The plot shows that mmce last drops from iteration 17 and stabilses thereafter.

```r
# Configure learners with probability type
learner2 <- makeLearner('classif.randomForest', predict.type = 'prob')

# Obtain parameters available for fine-tuning
getParamSet(learner2)
```

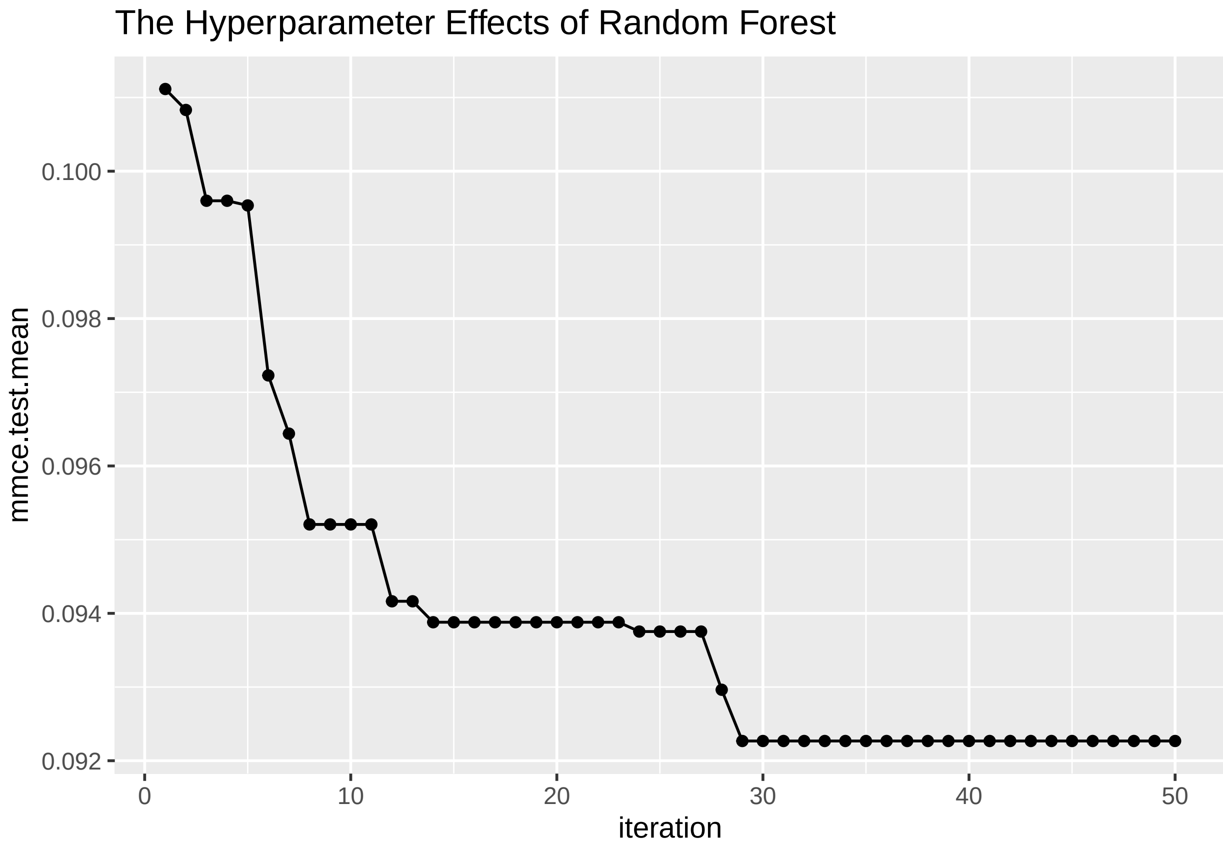
```
##                      Type  len   Def   Constr Req Tunable Trafo
## ntree             integer    -   500 1 to Inf   -    TRUE     -
## mtry              integer    -     - 1 to Inf   -    TRUE     -
## replace           logical    -  TRUE        -   -    TRUE     -
## classwt     numericvector <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     -
```

```r
# Make Param Set
ps2 <- makeParamSet(
  makeDiscreteParam('mtry', values = c(2,3,4,5,6)),
  makeDiscreteParam('ntree', values = c(seq(10,100,10))
  ))

# Configure tune Params settings
tunedLearner2_tuneparams <- tuneParams(learner = learner2,
                            task = classif.task,
                            resampling = rdesc,
                            par.set = ps2,
                            control = ctrl,
                            show.info = FALSE
                           )
```

```r
# Getting the hyper parameter effects:
learner2_effect <- generateHyperParsEffectData(tunedLearner2_tuneparams)

#Plot the effect
plotHyperParsEffect(learner2_effect, x = "iteration", y = "mmce.test.mean", plot.type = "line") + ggtitl
```




```r
# Making the tuned model:
tunedLearner2 <- setHyperPars(learner2, par.vals = tunedLearner2_tuneparams$x)

# Train the tune wrappers
tunedMod2  <- train(tunedLearner2, classif.task)

# Predict on training data
tunedPred2 <- predict(tunedMod2, classif.task)
```

Naive Bayes

We made attempts to tune on the laplace. By using the optimal kernel, we ran a grid search from 0 to 25. The optimal output was laplace=0 : mmce.test.mean=0.1163778.

The plot shows that the mmce drops to the lowest point at around second iteration and then remains stable.

```r
# Configure learners with probability type
learner4 <- makeLearner('classif.naiveBayes',predict.type = 'prob')

# Obtain parameters available for fine-tuning
getParamSet(learner4)
```

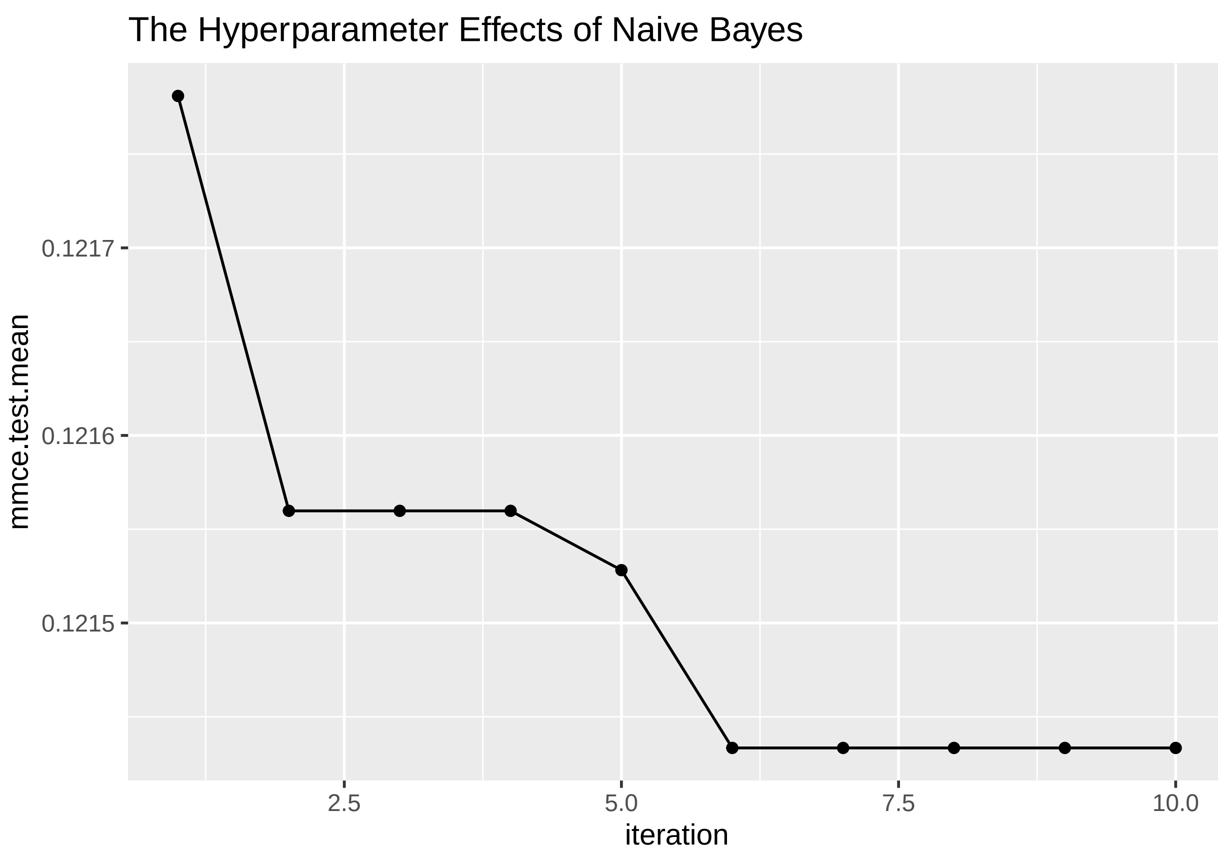
```
##             Type len Def   Constr Req Tunable Trafo
## laplace numeric   -   0 0 to Inf   -    TRUE     -
```

```r
# Make Param Set
ps4 <-makeParamSet(makeNumericParam("laplace", lower = 0, upper = 25))

# Configure tune Params settings
tunedLearner4_tuneparams <- tuneParams(learner = learner4,
                                       task = classif.task,
                                       resampling = rdesc,
                                       par.set = ps4,
                                       control = ctrl,
                                       show.info = FALSE)

# Getting the hyper parameter effects:
learner4_effect <- generateHyperParsEffectData(tunedLearner4_tuneparams)

#Plot the effect
plotHyperParsEffect(learner4_effect, x = "iteration", y = "mmce.test.mean", plot.type = "line") +  ggti
```




```r
# Making the tuned model:
tunedLearner4 <- setHyperPars(learner4, par.vals = tunedLearner4_tuneparams$x)

# Train the tune wrappers
tunedMod4  <- train(tunedLearner4, classif.task)

# Predict on training data
tunedPred4 <- predict(tunedMod4, classif.task)
```

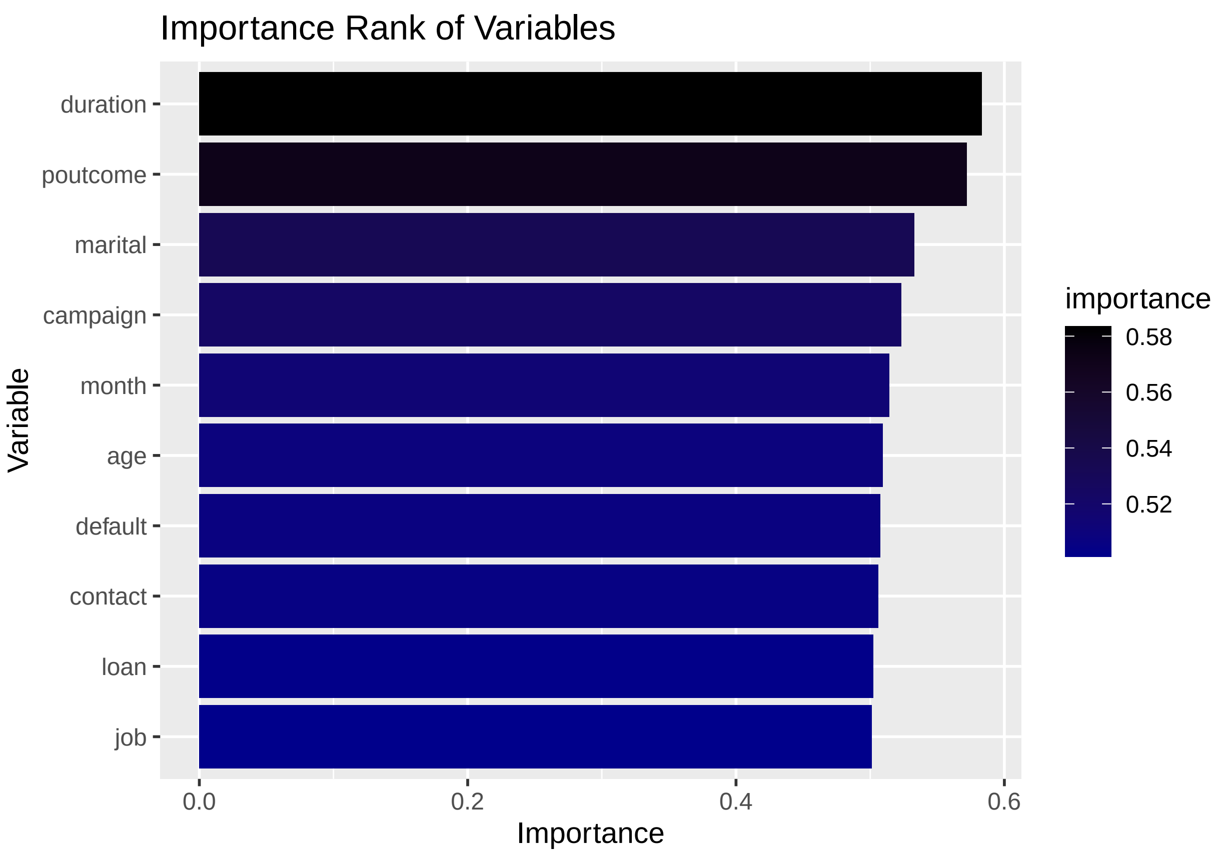
Decision Tree with feature selection

We have tried feature selection available in the SPSA package on decision to see if the performance could be better when there are fewer features.

```
# Run SPSA on tuned Decision Tree learner
learner1_FS <- spFSR:: spFeatureSelection(classif.task, wrapper = tunedLearner1,
                                           measure = mmce, num.features.selected = 0, show.info = FA

# Get the best models from feature selection
spsaModel <- learner1_FS$best.model

#Plot the important variables
spFSR::plotImportance(learner1_FS)
```



For the model predicting using Feature Selection, the best model has 10 descriptive features, 35 iterations and best measure value of 0.09862.The importance of the features is also plotted, which indicates that poutcome is the most important variable.

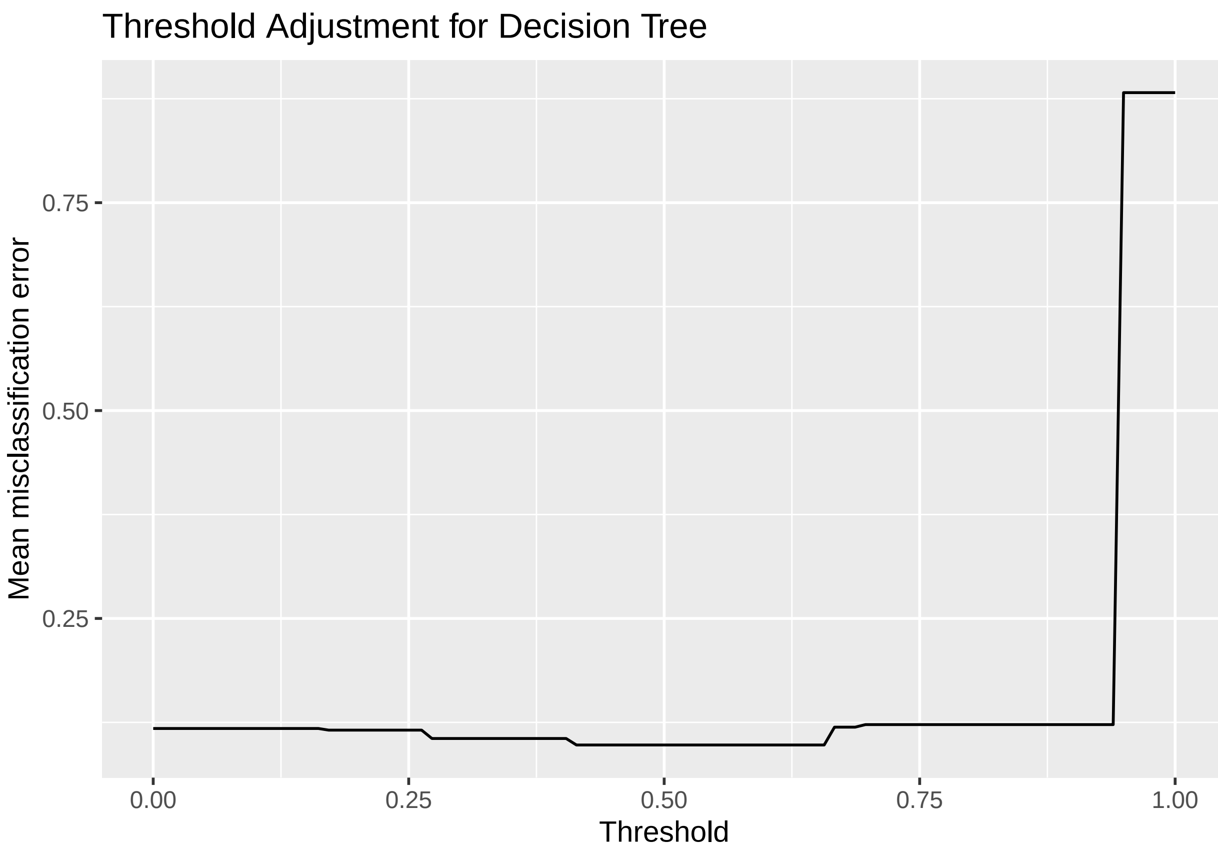
Treshold adjustment

The following plots depict the value of mmce vs. the range of probability thresholds. The thresholds are approximately 0.424, 0.646, and 0.162 for Decision Tree, Random Forest, and Naive Bayes classifiers respectively. These thresholds are used to determine the probability of an individual subscribing to the deposit term.

Decision Tree

```
# Generate data on threshold vs. performance(s) and
d1 <- generateThreshVsPerfData(tunedPred1, measures = list(mmce))

# Plot the threshold adjustment
plotThreshVsPerf(d1) + labs(title = 'Threshold Adjustment for Decision Tree', x = 'Threshold')
```

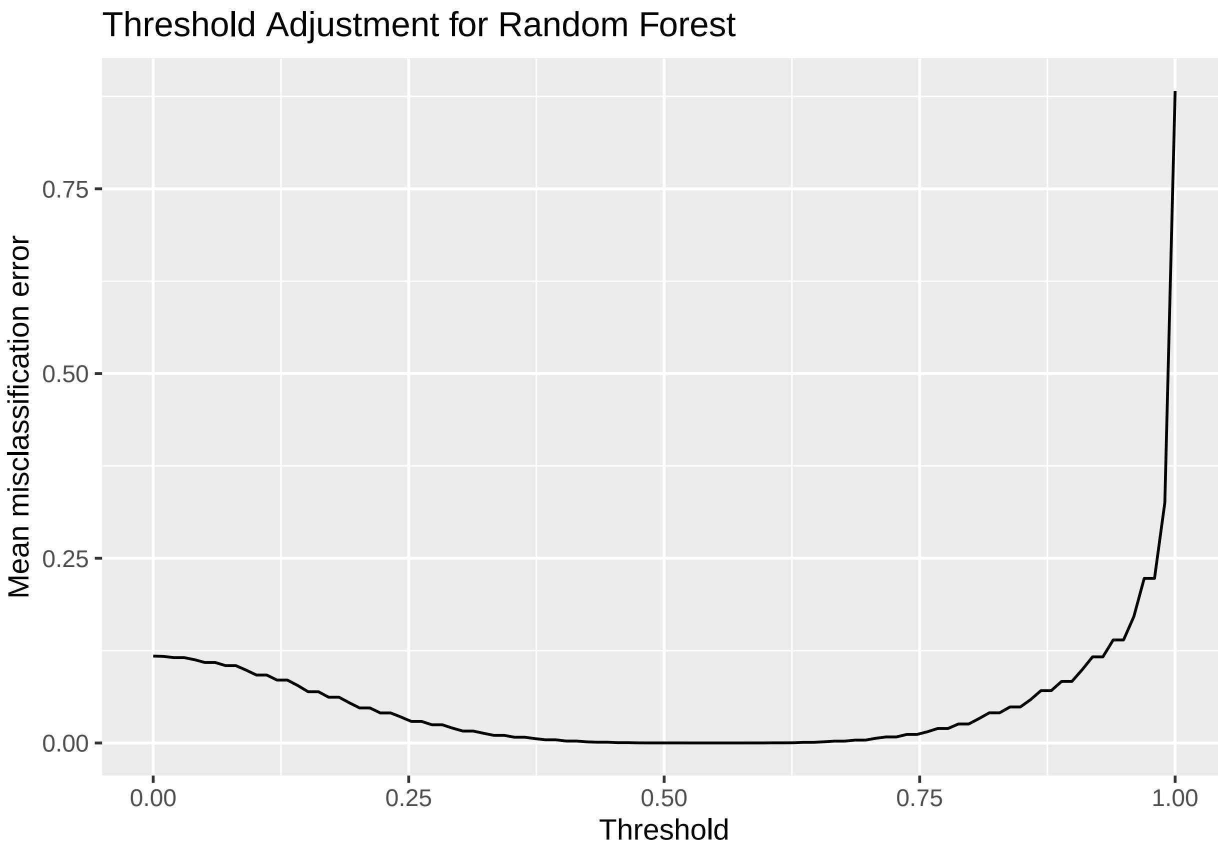



```
# Get threshold value
threshold1 <- d1$data$threshold[ which.min(d1$data$mmce) ]
```

Random Forest

```
# Generate data on threshold vs. performance(s) and
d2 <- generateThreshVsPerfData(tunedPred2, measures = list(mmce))

# Plot the threshold adjustment
plotThreshVsPerf(d2) + labs(title = 'Threshold Adjustment for Random Forest', x = 'Threshold')
```

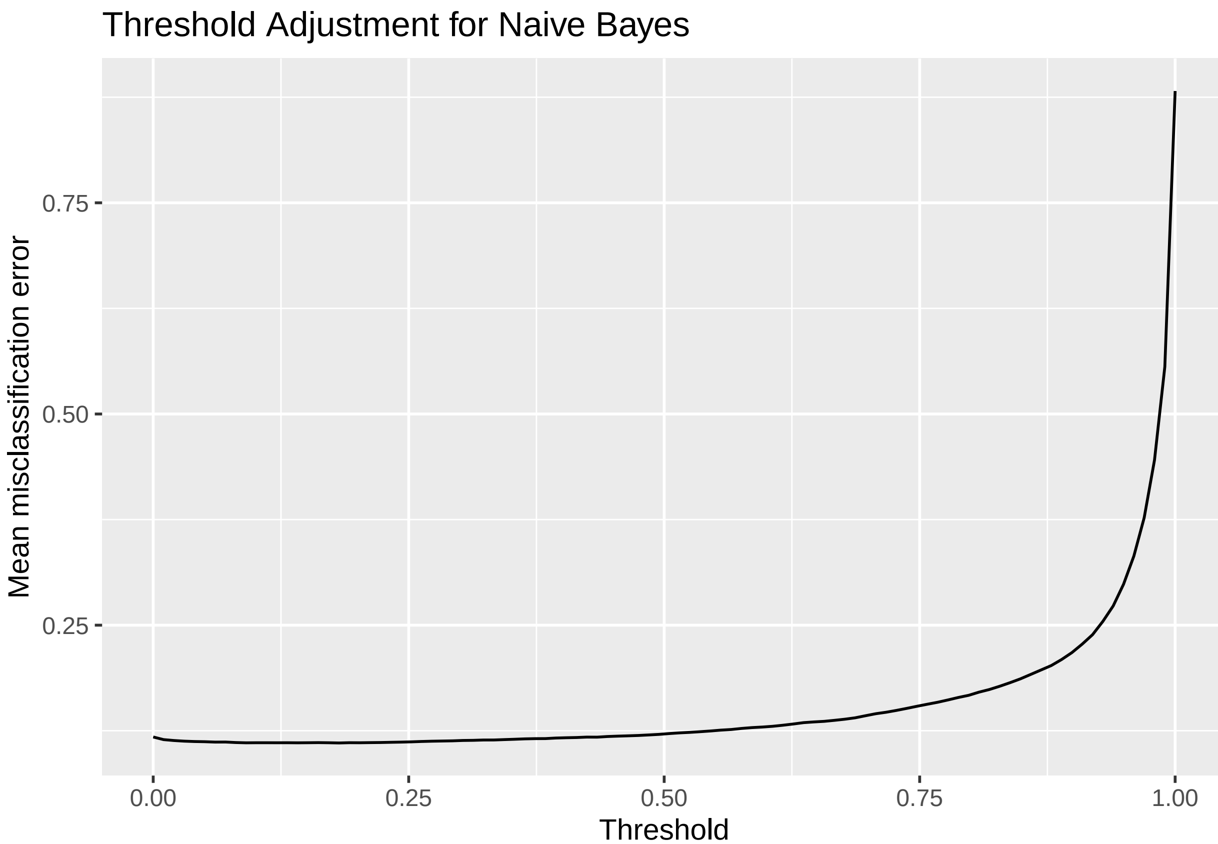



```
# Get threshold value
threshold2 <- d2$data$threshold[ which.min(d2$data$mmce) ]
```

Naive Bayes

```
# Generate data on threshold vs. performance(s)
d4 <- generateThreshVsPerfData(tunedPred4, measures = list(mmce))

# Plot the threshold adjustment
plotThreshVsPerf(d4) + labs(title = 'Threshold Adjustment for Naive Bayes', x = 'Threshold')
```




```r
# Get threshold value
threshold4 <- d4$data$threshold[ which.min(d4$data$mmce) ]
```

Analyze performance

We would use tuned wrapper models and optimal thresholds from previous sections to make predictions on the test data.

The performance measures used to evaluate the model are:

- AUC: Area Under The Curve. The higher, the better model.
- mmce: Misclassification error rate. The lower, the better model.

```r
# Decision Tree
testPred1 <- predict(tunedMod1, newdata = test)
```

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

```r
testPred1 <- setThreshold(testPred1, threshold1 )
# Random Forest
testPred2 <- predict(tunedMod2, newdata = test)
```

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

```r
testPred2 <- setThreshold(testPred2, threshold2 )
```

```r
# Naive Bayes
testPred4 <- predict(tunedMod4, newdata = test)
```

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

```r
testPred4 <- setThreshold(testPred4, threshold4 )
# Decision Tree with Feature Selection
testPred1_FS <- predict(spsaModel, newdata = test)
```

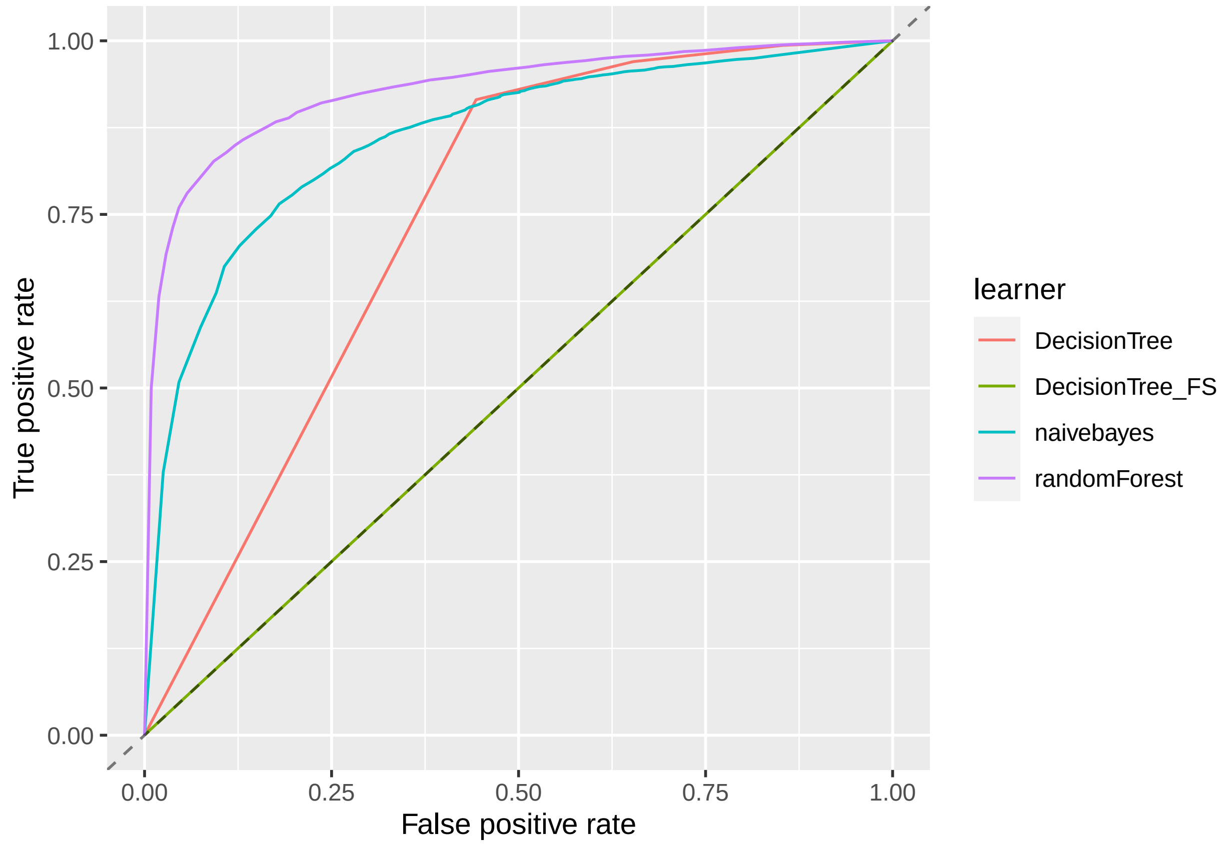
```
## Warning in predict.WrappedModel(spsaModel, newdata = test): Provided data for
## prediction is not a pure data.frame but from class tbl_df, hence it will be
## converted.
```

AUC

AUC curves

```r
#Comparing Decision Tree performance with Random Forest and Naive Bayes by using `plotROCCurves` plots
compare1 <- generateThreshVsPerfData(list(DecisionTree = testPred1, randomForest = testPred2, naivebayes
                                     measures = list(fpr, tpr))
```

```r
plotROCCurves(compare1)
```



The x-axis represents the False positive rate while the y-axis is the True positive rate. The 45-degree dotted line represents the uninformative line in the ROC curve. It is to say that the closer the curve comes to the line, the less accurate the model, meanwhile, the curve which is more sided to the upper left is more fit.

In our context, it is clear that the Decision Tree with Feature Selection lies near to the curve which represents an inappropriate fit. That means it is not any better than the random guess. On the other hand, the purple curve, which is Random Forest, is the farthest to the dotted line and closest to the top left.

However, the visualisation is only for reference purpose, it should be confirmed with the test.

Paired t-test

We continue to fit the optimised models on the test data. Since cross validation itself is a random process, we have performed pairwise t-tests to determine if any difference between the performance of any two classifiers is statistically significant. First, 5-fold stratified cross-validation is performed on each best model (without any repetitions). Second,paired t-test is conducted for the AUC score between the following model combinations:

** Decision Tree and Random Forest

** Decision Tree and Naïve Bayes

** Random Forest and Naïve Bayes

The Benchmark() function is exercised which allows us to compare different learning algorithms across one or more tasks on a given resampling strategy.

```
# Configure classification task for test data
classif.task_test <- makeClassifTask(data = test, target = 'y', id = 'bank')
```

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

```
#Perform benchmark on each learner
bmr <- benchmark(learners = list(
  makeLearner('classif.rpart', predict.type = 'prob'),
  makeLearner('classif.randomForest', predict.type = 'prob'),
  makeLearner('classif.naiveBayes',predict.type = 'prob')
), classif.task_test, rdesc, measures = auc)
```

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

## Resampling: cross-validation

## Measures:             auc

## [Resample] iter 1:    0.8045305

## [Resample] iter 2:    0.8072930

## [Resample] iter 3:    0.7945787

## [Resample] iter 4:    0.7863217

## [Resample] iter 5:    0.8011612

##

## Aggregated Result: auc.test.mean=0.7987770

##

## Task: bank, Learner: classif.randomForest

## Resampling: cross-validation

## Measures:             auc

## [Resample] iter 1:    0.9222027
```

```
## [Resample] iter 2:    0.9259784

## [Resample] iter 3:    0.9227010

## [Resample] iter 4:    0.9254180

## [Resample] iter 5:    0.9352892

##

## Aggregated Result: auc.test.mean=0.9263179

##

## Task: bank, Learner: classif.naiveBayes

## Resampling: cross-validation

## Measures:             auc

## [Resample] iter 1:    0.8560476

## [Resample] iter 2:    0.8620487

## [Resample] iter 3:    0.8462873

## [Resample] iter 4:    0.8607325

## [Resample] iter 5:    0.8452863

##

## Aggregated Result: auc.test.mean=0.8540805

##
```

```
#Get the overal performance
performance <- getBMRPerformances(bmr, as.df = TRUE)
```

```
#Subset the data frame for Decision Tree
performance_rpart <- performance[c(1:5),]
#Subset the data frame for Random Forest
performance_rf <- performance[c(6:10),]
#Subset the data frame for Naive Bayes
performance_nb <- performance[c(11:15),]
# t-test for Decision Tree and Random Forest
t.test(performance_rpart$auc, performance_rf$auc, paired = TRUE, alternative = "two.sided")
```

```
##
##  Paired t-test
##
## data:  performance_rpart$auc and performance_rf$auc
## t = -30.355, df = 4, p-value = 7.016e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1392064 -0.1158754
## sample estimates:
## mean of the differences
##              -0.1275409
```

```
# t-test for Decision Tree and Naive Bayes
t.test(performance_rpart$auc, performance_nb$auc, paired = TRUE, alternative = "two.sided")
```

```
##
##  Paired t-test
##
## data:  performance_rpart$auc and performance_nb$auc
## t = -10.871, df = 4, p-value = 0.0004064
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.06942800 -0.04117895
## sample estimates:
## mean of the differences
##             -0.05530347
```

```
# t-test for Random Forest and Naive Bayes
t.test(performance_rf$auc, performance_nb$auc, paired = TRUE, alternative = "two.sided")
```

```
##
##  Paired t-test
##
## data:  performance_rf$auc and performance_nb$auc
## t = 14.504, df = 4, p-value = 0.0001314
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  0.05840944 0.08606533
## sample estimates:
## mean of the differences
##              0.07223739
```

The null hypothesis for the test is that both algorithms perform equally well on the dataset. With p values smaller than 5% level of significance, we reject the null hypothesis signifying difference in model performance. This concludes that at 95% CI level, Random Forest is statistically the best model in this competition (in terms of AUC) when compared on the test data.

Confusion matrix

Decision Tree

```
# Calculate the confusion matrix for Decision Tree
calculateConfusionMatrix( testPred1,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     no        yes       -err.-
##   no     0.97/0.92 0.03/0.40 0.03
##   yes    0.65/0.08 0.35/0.60 0.65
##   -err.-      0.08      0.40 0.10
##
##
## Absolute confusion matrix:
##         predicted
## true        no yes -err.-
##   no     11638 360    360
##   yes     1023 543   1023
##   -err.-  1023 360   1383
```

```
performance(testPred1, measures = list(f1, tpr, tnr, fpr, fnr, mmce))
```

```
##        f1       tpr       tnr       fpr       fnr      mmce
## 0.9439150 0.9699950 0.3467433 0.6532567 0.0300050 0.1019611
```

Random Forest

```r
# Calculate the confusion matrix for Random Forest
calculateConfusionMatrix( testPred2,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     no        yes       -err.-
##   no     0.96/0.94 0.04/0.39 0.04
##   yes    0.46/0.06 0.54/0.61 0.46
##   -err.-      0.06      0.39 0.09
##
##
## Absolute confusion matrix:
##         predicted
## true        no yes -err.-
##   no     11468 530    530
##   yes      720 846    720
##   -err.-   720 530   1250
```

```r
performance(testPred2, measures = list(f1, tpr, tnr, fpr, fnr, mmce))
```

```
##         f1        tpr        tnr        fpr        fnr       mmce
## 0.94831721 0.95582597 0.54022989 0.45977011 0.04417403 0.09215571
```

Naive Bayes

```r
# Calculate the confusion matrix for Naive Bayes
calculateConfusionMatrix( testPred4,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     no        yes       -err.-
##   no     0.96/0.92 0.04/0.49 0.04
##   yes    0.65/0.08 0.35/0.51 0.65
##   -err.-      0.08      0.49 0.11
##
##
## Absolute confusion matrix:
##         predicted
## true        no yes -err.-
##   no     11474 524    524
##   yes     1016 550   1016
##   -err.-  1016 524   1540
```

```r
performance(testPred4, measures = list(f1, tpr, tnr, fpr, fnr, mmce))
```

```
##         f1        tpr        tnr        fpr        fnr       mmce
## 0.93711205 0.95632605 0.35121328 0.64878672 0.04367395 0.11353583
```

Decision tree with feature selection

```
# Calculate the confusion matrix for Decision Tree with Feature Selection
calculateConfusionMatrix(testPred1_FS,relative = TRUE)
```

```
## Relative confusion matrix (normalized by row/column):
##         predicted
## true     no          yes         -err.-
##   no     3e-04/1.00 1e+00/0.88 1.00
##   yes    0e+00/0.00 1e+00/0.12 0.00
##   -err.-       0.00       0.88 0.88
##
##
## Absolute confusion matrix:
##         predicted
## true     no   yes -err.-
##   no      3 11995  11995
##   yes     0  1566      0
##   -err.-  0 11995  11995
```

```
performance( testPred1_FS )
```

```
##      mmce
## 0.8843262
```

The total number of errors for single (true and predicted) classes is displayed in the -err.- row and column respectively. All the tuned classifiers accurately predict the clients who did not subscribe to term deposit. Decision Tree has the least mmce statistic, which shows it is more efficient than others.

Conclusion

Both the AUC and Cross-Validation method yields different outcomes. while AUC produces Random Forest to be the most efficient one based on the area covered under the curve, the canfusion matrix drifts in favour of decision tree providing better statistics for Precision, Recall , F1 and MMCE. It is also noticed that the decision tree is sensitive to number of features selected for modelling as reducing the number of descriptive features to 10 reduces the performance of the model with lower scores for Precision, Recall and other parameters. For this reason, working with full features is preferable over selected features for this dataset. In the end, it can be affirmed that decision tree outperforms Random Forest and Naive Bayes in terms of cross-validation parameters and we decided to settle on this as the one final and best model.