Trees, Random Forest, and Boosting: Applications in Marketing

Nick Kunz

3/5/2019

Introduction

This exercise briefly explores the topic of Marketing as it relates to Machine Learning. It broadly serves as an exploratory exercise, exclusively focusing on Classification Trees, Random Forest, and Boosting. The objective is to use these prediction methods to identify customers that have high purchase intent to direct marketing campaigns. The exercise assumes a conceptual understanding of Classification Trees, Random Forest, and Boosting. This work was completed in partial fulfillment for the course Machine Learning for Finance & Marketing at Columbia Business School in 2019 with Dr. Lentzas and is based on the paper by Moro, Cortez, and Rita, "A Data-Driven Approach to Predict the Success of Bank Telemarketing" Decision Support Systems, Elsevier, 62:22-31, June 2014.

Background

"A Portuguese retail bank uses its own contact-center to do direct marketing campaigns, mainly through phone calls (tele-marketing). Each campaign is managed in an integrated fashion and the results for all calls and clients within the campaign are gathered together, in a flat file report concerning only the data used to do the phone call. … In this context, in September of 2010, a research project was conducted to evaluate the efficiency and effectiveness of the telemarketing campaigns to sell long-term deposits. The primary goal was to achieve previously undiscovered valuable knowledge in order to redirect managers efforts to improve campaign results. … Since this project started being analyzed in detail in September of 2010, it meant that there were available reports for about three years of telemarketing campaigns … "

– Moro, Cortez, and Rita, "A Data-Driven Approach to Predict the Success of Bank Telemarketing", 2014

Requirements

First, we load the libraries 'tree', 'randomForest', and 'gbm'. The 'tree' library contains the 'tree( )' function, which is required to conduct Classification Tree regression. The 'randomForest' library contains the 'randomForest( )' function, which is required to conduct Random Forest. The library 'gbm' contains the 'gbm( )' function, which is required for Boosting.

```
## load libraries
library(tree) # classification trees
library(randomForest)  # random forest
library(gbm)  # boosted trees
```

Data

Here we load the data retrieved from the University of California - Irvine (UCI) Machine Learning Repository. The source tells us that the data contains customer information related to direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be 'yes' to subscribe or 'no' not to subscribe.

```
## load data
mkt_cmpgn_data = read.table("./bank-additional/bank-additional-full.csv",
                             sep  = ";",  # separator values
                             header = TRUE,  # create headers
                             check.names = TRUE,  # do not change header names
                             strip.white = TRUE,  # remove leading/trailing whitespace
                             stringsAsFactors = TRUE,  # strings as factor data type
                             na.strings = c("", "NA"))  # assign blank cells na
```

Data Inspection

Feature Names

After the data has been loaded, we briefly inspect the data frame by viewing the feature names by calling the 'names( )' function.

```
## view feature names
names(mkt_cmpgn_data)
```

```
##  [1] "age"            "job"            "marital"        "education"
##  [5] "default"        "housing"        "loan"           "contact"
##  [9] "month"          "day_of_week"    "duration"       "campaign"
## [13] "pdays"          "previous"       "poutcome"       "emp.var.rate"
## [17] "cons.price.idx" "cons.conf.idx"  "euribor3m"      "nr.employed"
## [21] "y"
```

Dimensions

Next, we view the number of observations and features contained within the data frame by calling the 'dim( )' function. Here we see that there are 41,188 observations and 21 features, which we know the names of from the previous section.

```
## view data frame dimension
dim(mkt_cmpgn_data)
```

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

Preview Observations

After, we preview the first few observations to get a more nuanced understanding of what is contained within the data frame by calling the 'head( )' function.

```
## preview observations
head(mkt_cmpgn_data, 5)
```

```
##   age       job marital   education default housing loan   contact month
## 1  56 housemaid married    basic.4y      no      no   no telephone   may
## 2  57  services married high.school unknown      no   no telephone   may
## 3  37  services married high.school      no     yes   no telephone   may
## 4  40    admin. married    basic.6y      no      no   no telephone   may
## 5  56  services married high.school      no      no  yes telephone   may
##   day_of_week duration campaign pdays previous    poutcome emp.var.rate
## 1         mon      261        1   999        0 nonexistent          1.1
## 2         mon      149        1   999        0 nonexistent          1.1
## 3         mon      226        1   999        0 nonexistent          1.1
## 4         mon      151        1   999        0 nonexistent          1.1
## 5         mon      307        1   999        0 nonexistent          1.1
```

```
##   cons.price.idx cons.conf.idx euribor3m nr.employed  y
## 1         93.994         -36.4     4.857        5191 no
## 2         93.994         -36.4     4.857        5191 no
## 3         93.994         -36.4     4.857        5191 no
## 4         93.994         -36.4     4.857        5191 no
## 5         93.994         -36.4     4.857        5191 no
```

Data Pre-Processing

Feature Selection

After inspecting the data (and the meta data), we remove the features 'duration', 'day_of_week', 'month' and 'nr.employed'. These features are removed, as they contain information collected after contact has already been made with the client and are highly correlated to the predicted outcome of whether the client subscribed for a term deposit. Conceptually, this is a "look ahead" into the future test set and is highly inappropriate in any prediction setting.

```
## remove feats
mkt_cmpgn_data$duration = NULL
mkt_cmpgn_data$day_of_week = NULL
mkt_cmpgn_data$month = NULL
mkt_cmpgn_data$nr.employed = NULL
```

Missingness

Here we replace the character string 'unknown' with NA. Subsequently, we remove those observations containing missing values by utilizing the 'na.omit( )' function. Then, we inspect the data frame dimension as an ad hoc measurement of missingness. Recall that the original data frame dimension contained 41,188 observations and 21 features. After removing the previous 4 features and observations containing NA's, the data frame now contains 30,488 observations and 17 features.

```
## replace 'unknown' char string in the data with na
mkt_cmpgn_data[mkt_cmpgn_data == "unknown" ] = NA

## remove observations containing na's
mkt_cmpgn_data = na.omit(mkt_cmpgn_data)

## view data frame dimension
dim(mkt_cmpgn_data)
```

```
## [1] 30488    17
```

Transformations

Binary

Consider that Trees based methods are best applied by ordered categorical values. Therefore, we reduce the 'job' feature, which contains multiple job titles to contain only two values, either employed or unemployed. Similarly, we apply the same transformation to the 'marital' feature, which contains multiple marital status, to either married or single.

```
## convert 'job' feat to char string data type for transformation
mkt_cmpgn_data$job = as.character(mkt_cmpgn_data$job)

## transform 'job' feat from multiple titles to two categories, either employed or unemployed
mkt_cmpgn_data$job[mkt_cmpgn_data$job != "unemployed"] = "employed"

## convert 'marital' feat to char string data type for transformation
```

```r
mkt_cmpgn_data$marital = as.character(mkt_cmpgn_data$marital)

## transform 'martial' feat from marital titles to two categories, either married or single
mkt_cmpgn_data$marital[mkt_cmpgn_data$marital != "married"] = "single"
```

Ordinal

In addition, we transform the 'education' feature to a hierarchical ordered numeric dummy feature, taking 6 increasing values commensurate with the levels of education observed. Here we assign the 'illiterate' values to 0, the 'basic.4y' to 1, 'basic.6y' to 2, 'basic.9y' to 3, 'high.school' to 4, 'professional.course' to 5, and 'university.degree' to 6. As a final step in this transformation, we convert the feature 'education' to numeric.

```r
## convert 'education' feat to char string data type for transformation
mkt_cmpgn_data$education = as.character(mkt_cmpgn_data$education)

## preview education levels
unique(mkt_cmpgn_data$education)
```

```
## [1] "basic.4y"            "high.school"         "basic.6y"
## [4] "professional.course" "basic.9y"            "university.degree"
## [7] "illiterate"
```

```r
## transform 'education' feat from char string to num ordinals
mkt_cmpgn_data$education[mkt_cmpgn_data$education == "illiterate"] = 0
mkt_cmpgn_data$education[mkt_cmpgn_data$education == "basic.4y"] = 1
mkt_cmpgn_data$education[mkt_cmpgn_data$education == "basic.6y"] = 2
mkt_cmpgn_data$education[mkt_cmpgn_data$education == "basic.9y"] = 3
mkt_cmpgn_data$education[mkt_cmpgn_data$education == "high.school"] = 4
mkt_cmpgn_data$education[mkt_cmpgn_data$education == "professional.course"] = 5
mkt_cmpgn_data$education[mkt_cmpgn_data$education == "university.degree"] = 6

## convert 'education' feat to num data type for analysis
mkt_cmpgn_data$education = as.numeric(mkt_cmpgn_data$education)
```

Data Types

Recall that the 'tree( )' function in R only takes numeric and factor data types as inputs. Therefore, we transform any and all character features in the data frame into factors. One way to achieve this is to utilize the 'as.factor( )' to manually transform the data type for each feature in the data frame. However, we know that as a default, R treats all features containing character strings as factors unless specified otherwise. We utilize this aspect of the programming language in this example to quickly convert all features containing character strings to factors by with the 'unclass( )' function and returning the result to the original data frame. After, we confirm that the data types have been converted correctly.

```r
## convert data types
mkt_cmpgn_data = as.data.frame(unclass(mkt_cmpgn_data))

## confirm data types
str(mkt_cmpgn_data)
```

```
## 'data.frame':    30488 obs. of  17 variables:
##  $ age           : int  56 37 40 56 59 24 25 25 29 57 ...
##  $ job           : Factor w/ 2 levels "employed","unemployed": 1 1 1 1 1 1 1 1 1 1 ...
##  $ marital       : Factor w/ 2 levels "married","single": 1 1 1 1 1 2 2 2 2 2 ...
##  $ education     : num  1 4 2 4 5 5 4 4 4 1 ...
##  $ default       : Factor w/ 3 levels "no","unknown",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ housing       : Factor w/ 3 levels "no","unknown",..: 1 3 1 1 1 3 3 3 1 3 ...
```

```
##  $ loan          : Factor w/ 3 levels "no","unknown",..: 1 1 1 3 1 1 1 1 3 1 ...
##  $ contact       : Factor w/ 2 levels "cellular","telephone": 2 2 2 2 2 2 2 2 2 2 ...
##  $ campaign      : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ pdays         : int  999 999 999 999 999 999 999 999 999 999 ...
##  $ previous      : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ poutcome      : Factor w/ 3 levels "failure","nonexistent",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ emp.var.rate  : num  1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 ...
##  $ cons.price.idx: num  94 94 94 94 94 ...
##  $ cons.conf.idx : num  -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 ...
##  $ euribor3m     : num  4.86 4.86 4.86 4.86 4.86 ...
##  $ y             : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

Descriptive Statistics

As a measure of analytical prudence, we provide brief descriptive statistics for the features contained within the data frame 'mkt_cmpgn_data'.

```
## descriptive stats
summary(mkt_cmpgn_data)
```

```
##       age                job            marital         education    
##  Min.   :17.00   employed  :29750   married:17492   Min.   :0.000  
##  1st Qu.:31.00   unemployed:  738   single :12996   1st Qu.:3.000  
##  Median :37.00                                      Median :4.000  
##  Mean   :39.03                                      Mean   :4.358  
##  3rd Qu.:45.00                                      3rd Qu.:6.000  
##  Max.   :95.00                                      Max.   :6.000  
##     default          housing           loan            contact     
##  no     :30485   no     :13967   no     :25720   cellular :20443  
##  unknown:     0   unknown:     0   unknown:     0   telephone:10045  
##  yes    :     3   yes    :16521   yes    : 4768                    
##                                                                    
##                                                                    
##                                                                    
##     campaign          pdays          previous            poutcome    
##  Min.   : 1.000   Min.   :  0.0   Min.   :0.0000   failure    : 3461  
##  1st Qu.: 1.000   1st Qu.:999.0   1st Qu.:0.0000   nonexistent:25836  
##  Median : 2.000   Median :999.0   Median :0.0000   success    : 1191  
##  Mean   : 2.521   Mean   :956.3   Mean   :0.1943                      
##  3rd Qu.: 3.000   3rd Qu.:999.0   3rd Qu.:0.0000                      
##  Max.   :43.000   Max.   :999.0   Max.   :7.0000                      
##   emp.var.rate      cons.price.idx  cons.conf.idx     euribor3m    
##  Min.   :-3.40000   Min.   :92.20   Min.   :-50.8   Min.   :0.634  
##  1st Qu.:-1.80000   1st Qu.:93.08   1st Qu.:-42.7   1st Qu.:1.313  
##  Median : 1.10000   Median :93.44   Median :-41.8   Median :4.856  
##  Mean   :-0.07151   Mean   :93.52   Mean   :-40.6   Mean   :3.460  
##  3rd Qu.: 1.40000   3rd Qu.:93.99   3rd Qu.:-36.4   3rd Qu.:4.961  
##  Max.   : 1.40000   Max.   :94.77   Max.   :-26.9   Max.   :5.045  
##    y        
##  no :26629  
##  yes: 3859  
##             
##             
##             
```

##

Data Splitting

Now that we have clean data, we split it into training and test sets. This allows us to compare and assess our results from Classification Trees, Random Forest, and Boosting. A prior step before we move forward with splitting our data into a training and test set, is specifying a random number with the function 'set.seed( )'. This allows our results to be reproducible for further analysis. After we begin at an established random number, we conduct splitting by calling the function 'sample( )'.

```
## random number
set.seed(1)

## training and test split
train = sample(1:nrow(mkt_cmpgn_data), nrow(mkt_cmpgn_data) / 2)  # create training set
test = mkt_cmpgn_data[-train, ]  # create test set
y.test = mkt_cmpgn_data$y[-train]  # create test y
```

Classification Trees

Training

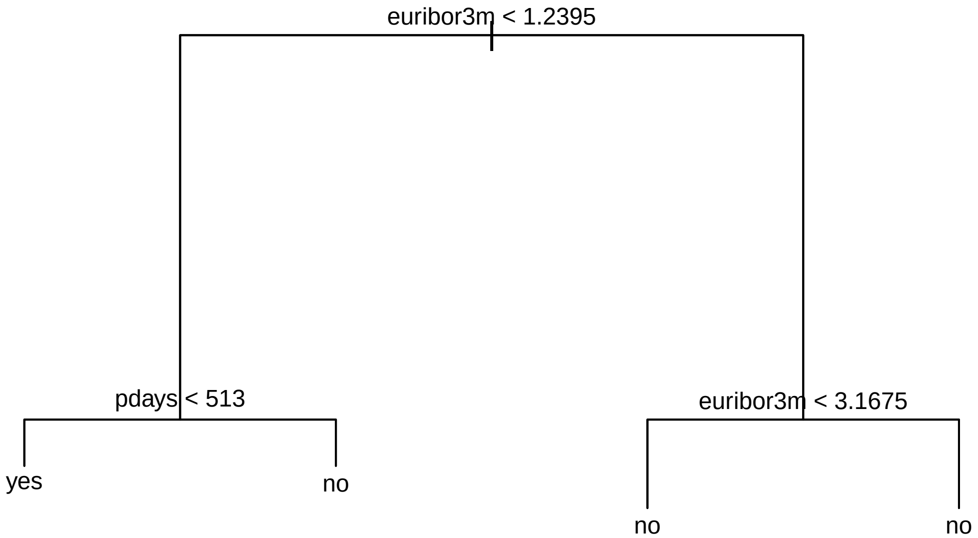
Here we begin to train the Classification Tree utilizing the 'tree( )' function. The function is simple, where the first argument contains the formula used in prediction (the dependent Y variable is the 'y' feature in data frame 'mkt_cmpg_data', the independent X variables are all other features, called by the period '.' before the tilde), the second argument specifies the data frame to call, and the third argument subsets 'mkt-cmpgn_data' to limit training only to the training set.

```
## tree training
tree = tree(y ~.,  # training formula
            data = mkt_cmpgn_data,  # data frame
            subset = train)  # limit training to training set
```

Feature Importance

To help better interpret the Classification Tree results, we plot and inspect the tree diagram. Interpreting the results, the most important feature in this classification is 'euribor3m', where the observations are split at 1.2395 (first split). The second most important feature is 'pdays', where the observations are split at 513 (second split). Also, we see the 'euribor3m' feature again (second split). However, the split occurs at a different level in the observations at 3.1675. Recall the goal of this exercise is to predict whether or not the client would subscribe for a bank term deposit. Observe the outcomes, 'yes' or 'no'. Notice that the results 'yes' (the result we are interested in) fall under the tree at less than 1.2395 in 'euribor3m' and of those, only on those which are also less than 513 in 'pdays' result in 'yes'. Meaning, the lower the average inter-bank interest rate at which European banks are prepared to lend to one another ('euribor3m') and the lower the number of days that passed by after the client was last contacted from a previous campaign ('pday') were the most influential indicators of marketing effectiveness according to the Classification Tree model. This is important to consider when fitting a the Classification Tree model and at the next step in prediction.

```
## tree plotting
plot(tree)
text(tree, cex = 0.75)
```



Prediction

After we trained the model, we utilize it for making predictions on the test set with the 'predict( )' function. Note that the 'type' argument is utilized in order to specify the prediction type. In this case, the type of prediction we're interested in is classification, indicated by the term "class".

```
## tree prediction
tree_predict = predict(tree,
                       newdata = test,  # test set
                       type = "class")  # classification
```

Performance Evaluation

Next, we evaluate the model's performance by calling the 'table( )' function and calculate the Test MSE. The results indicate a Test MSE of roughly 11.3%. Meaning, the model achieves a roughly 88.7% accuracy on whether or not the client would subscribe for a bank term deposit. Cool!

```
## tree prediction results
table(tree_predict, y.test)
```

```
##             y.test
## tree_predict    no   yes
##          no  13180  1561
##          yes   164   339
```

```
## calculate test mse
1 - (13180 + 339) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.1131593
```

```
## calculate accuracy
(13180 + 339) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.8868407
```

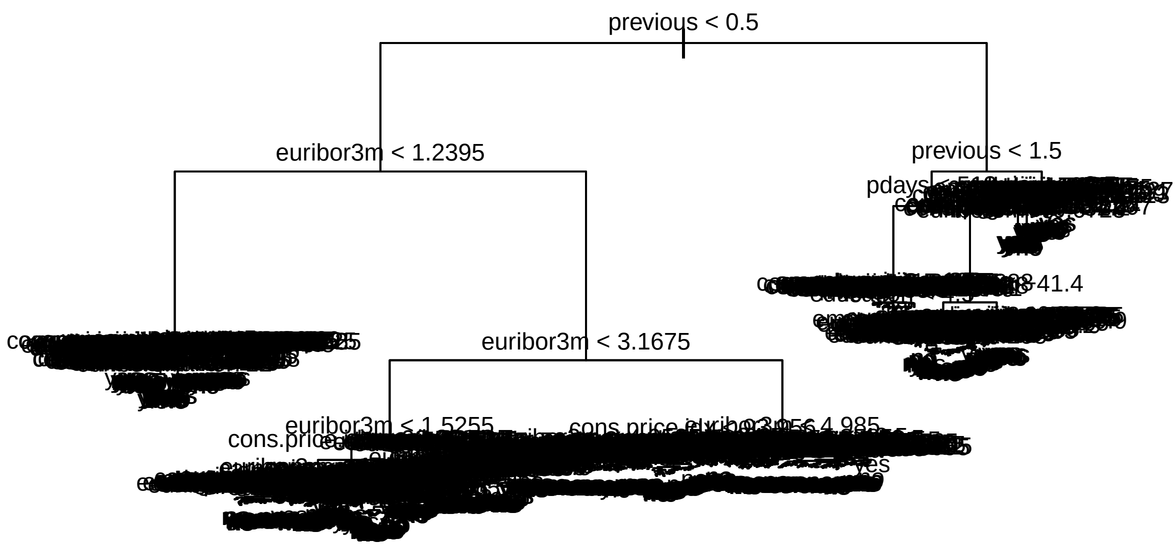
Alternatives

Gini

As an alternative tree splitting method, the following serves as an experiment to test if we can improve the model's predictive accuracy by simply specifying a different splitting criteria. The following succinctly demonstrates the repetition of all previous analysis. However, uses 'gini' as the splitting criteria. The results indicate a Test MSE of roughly 12.4%. Meaning, the model achieves a roughly 87.6% accuracy on whether or not the client would subscribe to a bank term deposit. In other words, a reduction in the predictive performance of the model.

```
## tree training - gini
tree_gini = tree(y ~.,  # training formula
                 data = mkt_cmpgn_data,  # data frame
                 subset = train,  # limit training to training set
                 split = "gini")  # splitting criteria

## tree plotting - gini
plot(tree_gini)
text(tree_gini, cex = 0.75)
```




```
## tree prediction - gini
tree_predict_gini = predict(tree_gini, test, type = "class")

## tree prediction results - gini
table(tree_predict_gini, y.test)
```

```
##                   y.test
## tree_predict_gini    no   yes
##                no 12764  1309
##               yes   580   591
```

```
## calculate test mse - gini
1 - (12764 + 591) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.1239176
```

```
## calculate accuracy - gini
(12764 + 591) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.8760824
```

Random Forest

Training

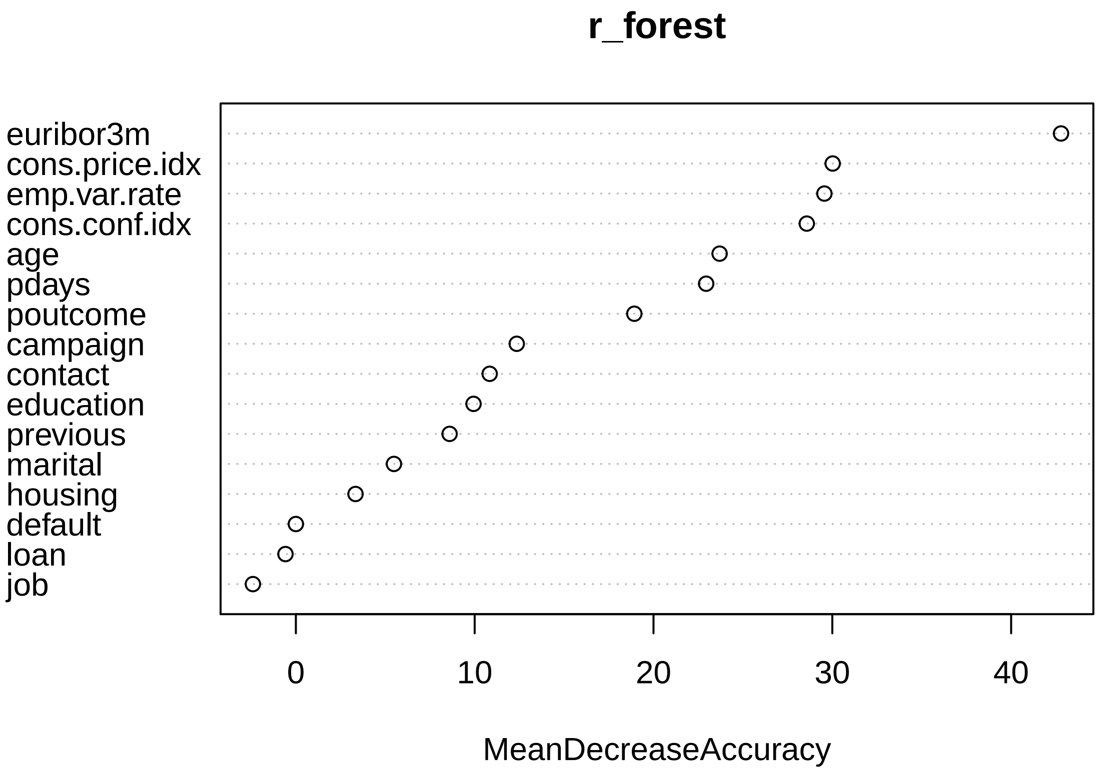
Here we begin to train Random Forest utilizing the 'randomForest( )' function. The function is simple, where the first argument contains the formula used in prediction (the dependent Y variable is the 'y' feature in data frame 'mkt_cmpg_data', the independent X variables are all other features, called by the period '.' before the tilde), the second argument specifies the data frame to call, and the third argument subsets 'mkt-cmpgn_data' to limit training only to the training set. The fourth argument specifies the tuning parameter 'm' where it is set at a generally accepted square root of 'p' or the number of features in the data frame 'mkt_cmpgn_data'. Lastly, in the 'importance' argument, we specify the output of the feature importance.

```
## random forest training
r_forest = randomForest(y ~.,  # training formula
                        data = mkt_cmpgn_data,  # data frame
                        subset = train,  # limit training to training set
                        mtry = sqrt(ncol(mkt_cmpgn_data)),  # tuning param 'm'
                        importance = TRUE)  # output feature importance
```

Feature Importance

To help better interpret the Random Forest training results, we create a table and a corresponding plot that displays each features influence on the model. Interpreting the results, the most important feature is 'euribor3m', similar to the Classification Tree model. However the second most important feature is 'cons.price.idx', which tells us something different than the Classification Tree model, where the second most important feature was 'pdays' the number of days that passed by after the client was last contacted from a previous campaign. Meaning, the average inter-bank interest rate at which European banks are prepared to lend to one another ('euribor3m') and consumer price index ('cons.price.idx') were the most influential indicators of marketing effectiveness according to the Random Forest model. The following plot is measured by the decrease in mean accuracy.

```
## plot random forest feature importance
varImpPlot(r_forest,
           type = 1,
           sort = TRUE,
           n.var = nrow(r_forest$importance))
```




```r
## preview random forest feature importance
importance(r_forest)
```

```
##                       no         yes MeanDecreaseAccuracy MeanDecreaseGini
## age            25.272470  -2.1048511           23.6977803    406.361635703
## job            -2.496049  -0.3821214           -2.4031117     21.331255912
## marital         9.828987  -7.5931037            5.4841014     62.985728675
## education       6.985848   7.3147942            9.9349928    170.092765015
## default         0.000000   0.0000000            0.0000000      0.001102574
## housing         4.172401  -0.6109898            3.3368841     71.509566806
## loan           -1.124318   0.9706323           -0.5814772     53.571622359
## contact         8.605375  23.5409696           10.8375719     48.842409169
## campaign       11.285645   4.6122490           12.3520294    179.246428327
## pdays          10.173465  26.3481012           22.9453287    173.546556432
## previous        8.752254  -0.9927725            8.5970801     61.183786508
## poutcome       15.993109  11.1995176           18.9277397    138.019527033
## emp.var.rate   27.598693  10.8126862           29.5568536    137.129182541
## cons.price.idx 29.438427 -11.8214282           30.0211083    129.874479398
## cons.conf.idx  27.758824  -8.9360724           28.5722001    142.079136436
## euribor3m      38.171192  11.4925029           42.7911910    576.146045867
```

Prediction

After we trained the model, we utilize it for making predictions on the test set with the ‘predict( )’ function. Note that the ‘type’ argument is utilized in order to specify the prediction type. Similarly to the previous examples, the type of prediction we’re interested in is classification, indicated by the term “class”.

```r
## random forest prediction
r_forest_predict = predict(r_forest,
                           newdata = test,  # test set
                           type = "class")  # classification
```

Performance Evaluation

Next, we evaluate the model's performance by calling the 'table( )' function and calculate the Test MSE. Our results indicate a Test MSE of roughly 11.6%. Meaning, the model achieves a roughly 88.4% accuracy on whether or not the client would subscribe for a bank term deposit. There was no improvement, but rather a marginal reduction in the predictive performance in this model when compared to Random Forest.

```
## random forest prediction results
table(r_forest_predict, y.test)
```

```
##                  y.test
## r_forest_predict    no   yes
##              no  12898  1320
##              yes   446   580
```

```
## calculate test mse
1 - (12898 + 580) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.1158489
```

```
## calculate accuracy
(12898 + 580) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.8841511
```

Boosting

Pre-Processing

Before we proceed with Boosting with the 'gbm( )' function, the algorithm requires that our Y prediction 'y' is numeric. Since our the classification results we are predicting is binary, either 'yes' or 'no', we transform those observations in the 'y' feature to either 1 or 0, respectively. We apply this transformation to all data sets.

```
## training set

## convert 'y' feat to char string data type for transformation
mkt_cmpgn_data$y = as.character(mkt_cmpgn_data$y)

## transform 'y' feat from char string (yes/no) to binary (1/0)
mkt_cmpgn_data$y[mkt_cmpgn_data$y == "yes"] = 1
mkt_cmpgn_data$y[mkt_cmpgn_data$y == "no"] = 0

## convert 'y' feat to num
mkt_cmpgn_data$y = as.numeric(mkt_cmpgn_data$y)

## test set

## convert 'y' feat to char string data type for transformation
test$y = as.character(test$y)

## transform 'y' feat from char string (yes/no) to binary (1/0)
test$y[test$y == "yes"] = 1
test$y[test$y == "no"] = 0

## convert 'y' feat to num
test$y = as.numeric(test$y)
```

```r
## test y

## convert 'y' feat to char string data type for transformation
y.test = as.character(y.test)

## transform 'y' feat from char string (yes/no) to binary (1/0)
y.test[y.test == "yes"] = 1
y.test[y.test == "no"] = 0

## convert 'y' feat to num
y.test = as.numeric(y.test)
```

Training

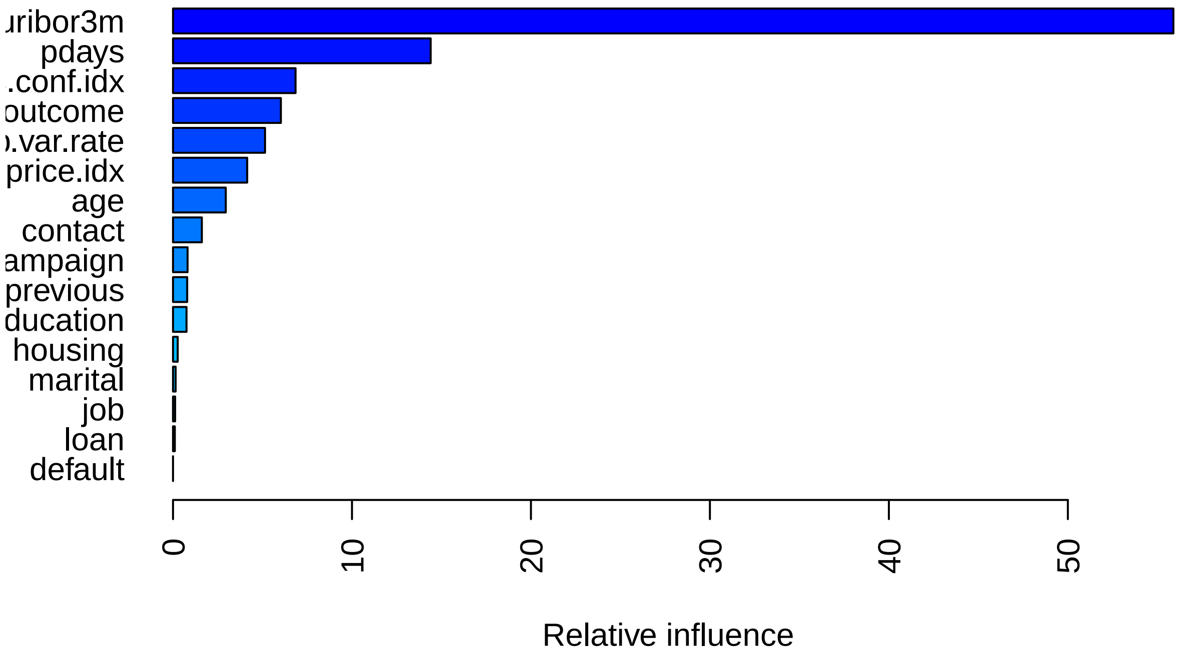
Here we begin training the Boosting model utilizing the 'gbm( )' function. The first argument contains the formula used in prediction (the dependent Y variable is the 'y' feature in data frame 'mkt_cmpg_data', the independent X variables are all other features, called by the period '.' before the tilde), the second argument specifies the data frame to call. The 'distribution' argument specified here as 'bernoulli', which is required for classification. If we were interested in regression, the argument would be more appropriately specified as 'gaussian', as a normal distribution. The argument 'n.trees' specifies the number of trees to create. In this case we somewhat arbitrarily choose 10,000. We do not test the optimal number of trees here, as it is outside of the scope of this study. However, it is a prudent number of trees and falls within the recommended range according the algorithms documentation. The argument 'interaction.depth' indicates the tree depth we establish at 3, which also falls within the recommended range. In our finally argument, we establish the tuning parameter lambda to slightly shrink our coefficient estimates, introducing bias, but in anticipation for decreased variance in the Test MSE.

```r
## boosting training
boost = gbm(y ~.,
            data = mkt_cmpgn_data[train, ],
            distribution = "bernoulli",  # bernoulli distribution for classification
            n.trees = 10000,  # num of trees
            interaction.depth = 3,  # tree depth
            shrinkage = 0.001)  # tuning param lambda
```

Feature Importance

To help better interpret the Boosting training results, we plot each features influence on the model from its corresponding table. Exhibited here are the relative influences of each feature. Interpreting the results, the most important feature is 'euribor3m', similar to the Classification Tree and Random Forest models. However, the second most important feature is 'pday', which tells us something different than the Random Forest model, where the second most important feature was the consumer price index ('cons.price.idx'). Yet, it tells us something similar to the Classification Tree model, where the second most important feature was also 'pday'. Meaning, the average inter-bank interest rate at which European banks are prepared to lend to one another ('euribor3m') and the number of days that passed by after the client was last contacted from a previous campaign ('pday') were the most influential indicators of marketing effectiveness according to the Boosting model. Perhaps there is something to be gleaned from the similarity in the two most important features from 2 out of the 3 models? The following plot is measured by the decrease in mean accuracy.

```r
## preview boosting feature importance
summary(boost,
        method = relative.influence,
        las = 2)
```




```
##                             var    rel.inf
## euribor3m           euribor3m 55.8980095
## pdays                   pdays 14.3965723
## cons.conf.idx   cons.conf.idx  6.8472419
## poutcome             poutcome  6.0269971
## emp.var.rate     emp.var.rate  5.1409076
## cons.price.idx cons.price.idx  4.1442113
## age                       age  2.9488841
## contact               contact  1.6112705
## campaign             campaign  0.8139977
## previous             previous  0.7913704
## education           education  0.7536652
## housing               housing  0.2621474
## marital               marital  0.1454201
## job                       job  0.1175776
## loan                     loan  0.1017273
## default               default  0.0000000
```

Prediction

After we trained the model, we utilize it for making predictions on the test set. Note that the 'type' argument is utilized in order to specify the prediction type. In this case, the type of prediction we're interested in is classification. However, unlike Classification Trees and Random Forest, where our prediction was a factor indicated by the term "class", we need to specify "response" as our prediction in numerical terms. Also, because of the application of Boosting, we specify the same number of trees utilized in fitting the model (10,000). As a final step in Boosting prediction, we round our results to either 1 or 0.

```
## boosting prediction
boost_predict = predict(boost,
                        newdata = test,
                        type = "response",
                        n.trees = 10000)

## round boosting predictions to binary (1/0)
```

```r
boost_predict = round(boost_predict)
```

Performance Evaluation

Next, we evaluate the model's performance by calling the 'table( )' function and calculate the Test MSE. The results indicate a Test MSE of roughly 11.2%. Meaning, the model achieves a roughly 88.8% accuracy on whether or not the client would subscribe for a bank term deposit. There was a marginal improvement in the predictive performance in this model when compared to both Classification Trees and Random Forest.

```r
## boosting prediction results
table(boost_predict, y.test)
```

```
##              y.test
## boost_predict     0     1
##             0 13083  1444
##             1   261   456
```

```r
## calculate test mse
1 - (13083 + 456) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.1118473
```

```r
## calculate accuracy
(13083 + 456) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.8881527
```

Results & Discussion

Examining the Test MSE results from Classification Trees, Random Forest, and Boosting, the model that achieved the lowest Test MSE (highest predictive accuracy) is the Boosting model. The Boosting model resulted in a Test MSE of roughly 11.2%. The Classification Tree model performed only slightly worse with the Test MSE of roughly 11.3%. The Random Forest model had similar performance, yet slightly worse than the rest with a Test MSE of roughly 11.6%. It is important to consider that these differences are marginal and the computational cost in achieving them should also be considered. Although the Boosting model achieved the highest predictive performance, it is generally more computationally expensive than the Classification Tree model. Calculating the computational cost is beyond the scope of this discussion. However, the speed at which these results cannot be dismissed; especially in another application, marketing or otherwise. As a recommendation for this application, Classification Trees might be appropriate. With only a marginal loss in predictive accuracy, Classification Trees achieve relative predictive quality and are computationally inexpensive when compared to Random Forest and Boosting.

```r
## calculate test mse - trees
1 - (13180 + 339) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.1131593
```

```r
## calculate test mse - random forest
1 - (12898 + 580) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.1158489
```

```r
## calculate test mse - boosting
1 - (13083 + 456) / (nrow(mkt_cmpgn_data) / 2)
```

```
## [1] 0.1118473
```

Conclusion

This exercise briefly explored the topic of Marketing as it relates to Machine Learning. The study tested the predictive accuracy of whether or not a client would subscribe for a bank term deposit for a Portuguese retail bank using direct telemarketing campaigns. It focused exclusively on Classification Trees, Random Forest, and Boosting. This study serves as an exploratory exercise with commonly applied machine learning methods. More information in this regard can be found in the text “An Introduction to Statistical Learning with Applications in R” by James, Witten, Hastie, and Tibshirani (2016). With regard to the content of the study, more information can be found in the text ”A Data-Driven Approach to Predict the Success of Bank Telemarketing" by Moro, Cortez, and Rita.