am5696_HW_2

Aryan Mishra

2/24/2022

```
set.seed(1)
library(caret)

## Loading required package: ggplot2

## Loading required package: lattice

library(plyr)
library(dplyr)

##
## Attaching package: 'dplyr'

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

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

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

library(C50)
library(tree)
library(randomForest)

## randomForest 4.7-1

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

##
## Attaching package: 'randomForest'

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

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

```
library(gbm)

## Loaded gbm 2.1.8

library(caret)

library(kernlab)

##
## Attaching package: 'kernlab'

## The following object is masked from 'package:ggplot2':
##
##     alpha

setwd('/Users/aryan/Desktop/MSBA_Spring_22/MRKT_9653_ML/Homework_2')
```

Downloading the Data and Printing Summary (Q1)

```
data <-
read.csv("/Users/aryan/Desktop/MSBA_Spring_22/MRKT_9653_ML/Homework_2/bank-
additional/bank-additional-full.csv", header=TRUE, sep=";")

cols_to_remove <-  c(9,10,11,20)
colnames(data[,cols_to_remove])

## [1] "month"       "day_of_week" "duration"    "nr.employed"

data_reduced <- data[,-which(names(data) %in%
colnames(data[,cols_to_remove]))]
```

We remove duration since it highly affects the output target (if duration = 0, then y = ‘no’). Furthermore, duration is not known before a call is performed. Also, after the end of the call, y is obviously known. Therefore, duration is redundant when it comes to predicting y.

From a modeling perspective, month and day_of_week have multiple categories, which can be problematic for tree-based methods that search through all features/categories for each split. However, more importantly, from a business perspective, the day of the week and month are not really useful variables to identify customers who would subscribe to the product since they usually don’t have an effect on the firm’s ability to attract potential customers. Even if they did have an effect, it is extremely hard to control these factors if the firm wishes to attract new customers. Finally, nr.employed also is a variable that is not really useful at all from a commercial perspective. The number of employees of the banking institution has little to no effect on the success of the current marketing campaign.

```
print(summary(data_reduced))
```

```
##       age            job              marital          education        
##  Min.   :17.00   Length:41188       Length:41188       Length:41188      
##  1st Qu.:32.00   Class :character   Class :character   Class :character  
##  Median :38.00   Mode  :character   Mode  :character   Mode  :character  
##  Mean   :40.02                                                           
##  3rd Qu.:47.00                                                           
##  Max.   :98.00                                                           
##    default            housing              loan             contact         
##  Length:41188       Length:41188       Length:41188       Length:41188      
##  Class :character   Class :character   Class :character   Class :character  
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character  
##                                                                           
##                                                                           
##                                                                           
##     campaign          pdays          previous       poutcome        
##  Min.   : 1.000   Min.   :  0.0   Min.   :0.000   Length:41188      
##  1st Qu.: 1.000   1st Qu.:999.0   1st Qu.:0.000   Class :character  
##  Median : 2.000   Median :999.0   Median :0.000   Mode  :character  
##  Mean   : 2.568   Mean   :962.5   Mean   :0.173                     
##  3rd Qu.: 3.000   3rd Qu.:999.0   3rd Qu.:0.000                     
##  Max.   :56.000   Max.   :999.0   Max.   :7.000                     
##   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.:1.344  
##  Median : 1.10000   Median :93.75   Median :-41.8   Median :4.857  
##  Mean   : 0.08189   Mean   :93.58   Mean   :-40.5   Mean   :3.621  
##  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            
##  Length:41188      
##  Class :character  
##  Mode  :character  
##                    
##                    
##                    
```

Looking at the summary of the imported data (excluding the variables we removed), we notice some interesting insights. First of all, the average (mean) age of the bank customer is 40, implying most of the clients of this institution are in their middle ages and thus, are more likely to have a relatively high level of income compared to some of the younger customers. The mean 'campaign' is 2.568, indicating that on average, the bank performed multiple contacts during the current campaign for a particular client. The median number of pdays, including the first and third quartiles, interestingly is 999, which implies almost all the clients were not previously contacted during a prior campaign. This shows us that the bank is mostly targeting new customers (which makes sense). Other interesting insights include the consumer confidence index, which has an average number of -40.5. The consumer confidence index tells us how optimistic people are about the economy and their ability to find jobs. A mean score of -40.5 indicates at the time the bank was conducting this marketing campaign, the consumers were less optimistic than the benchmark CCI of 100 set in the reference period (most likely 1985). Interestingly, the average consumer price index, which is a measure of inflation, is at 93.58, indicating a 6.42% decrease in the price of the market basket compared to the reference period. Furthermore, the median euribor3m rate of 4.857, which coupled with the rest of the economic indicators (low CCI and low inflation) hints that this marketing campaign was most likely conducted during 2008-2010, when the financial crisis hit the world (I later confirmed that this data was collected from May 2008 to November 2010). I hypothesize that this might have a major impact on the effectiveness of the marketing campaign, especially given the possibility that it was conducted during the height of the global financial crisis.

Data Pre-Processing (Q2)

```
data_reduced[data_reduced == "unknown"] <- NA #Removing NAs
data_reduced <- na.omit(data_reduced)

data_reduced["job"][data_reduced["job"] != "unemployed"] <- "employed"
data_reduced["marital"][data_reduced["marital"] == "divorced"] <- "single"
for(i in 1 : nrow(data_reduced)){
     if (data_reduced$education[i] == 'illiterate'){
         data_reduced$education[i] = 0
     } else if (data_reduced$education[i] == 'basic.4y'){
         data_reduced$education[i] = 1
     } else if (data_reduced$education[i] == 'basic.6y'){
         data_reduced$education[i] = 2
     } else if (data_reduced$education[i] == 'basic.9y'){
         data_reduced$education[i] = 3
     } else if (data_reduced$education[i] == 'high.school'){
        data_reduced$education[i] = 4
     } else if (data_reduced$education[i] == 'professional.course'){
         data_reduced$education[i] = 5
     }
       else if (data_reduced$education[i] == 'university.degree'){
           data_reduced$education[i] = 6
     }
}
data_reduced$education<-as.numeric(data_reduced$education)
data_reduced["poutcome"][data_reduced["poutcome"] == "failure"] <-
"nonexistent"
```

Train-Test Split & Transforming Characters into Factors (Q3)

```
set.seed(1)
train <- sample(1:nrow(data_reduced), nrow(data_reduced) / 2)
data_reduced.test <- data_reduced[-train, ]
data_reduced[sapply(data_reduced, is.character)] <-
lapply(data_reduced[sapply(data_reduced, is.character)],
                                                           as.factor)
```

```
data_reduced.test[sapply(data_reduced.test, is.character)] <-
lapply(data_reduced.test[sapply(data_reduced.test, is.character)],as.factor)
```

Let's verify if we successfully transformed the characters into factors.

```
str(data_reduced)

## '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/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ housing       : Factor w/ 2 levels "no","yes": 1 2 1 1 1 2 2 2 1 2 ...
##  $ loan          : Factor w/ 2 levels "no","yes": 1 1 1 2 1 1 1 1 2 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/ 2 levels "nonexistent",..: 1 1 1 1 1 1 1 1 1
1 ...
##  $ 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 ...
##  - attr(*, "na.action")= 'omit' Named int [1:10700] 2 6 8 11 16 18 20 22
27 28 ...
##   ..- attr(*, "names")= chr [1:10700] "2" "6" "8" "11" ...

str(data_reduced.test)

## 'data.frame':    15244 obs. of  17 variables:
##  $ age           : int  56 56 59 24 29 57 35 35 50 41 ...
##  $ 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 2 2 2 1 1
1 2 ...
##  $ education     : num  1 4 5 5 4 1 2 2 3 4 ...
##  $ default       : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
##  $ housing       : Factor w/ 2 levels "no","yes": 1 1 1 2 1 2 2 2 2 2 ...
##  $ loan          : Factor w/ 2 levels "no","yes": 1 2 1 1 2 1 1 1 2 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/ 2 levels "nonexistent",..: 1 1 1 1 1 1 1 1 1
1 ...
##  $ 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 ...
##  - attr(*, "na.action")= 'omit' Named int [1:10700] 2 6 8 11 16 18 20 22
27 28 ...
##   ..- attr(*, "names")= chr [1:10700] "2" "6" "8" "11" ...
```

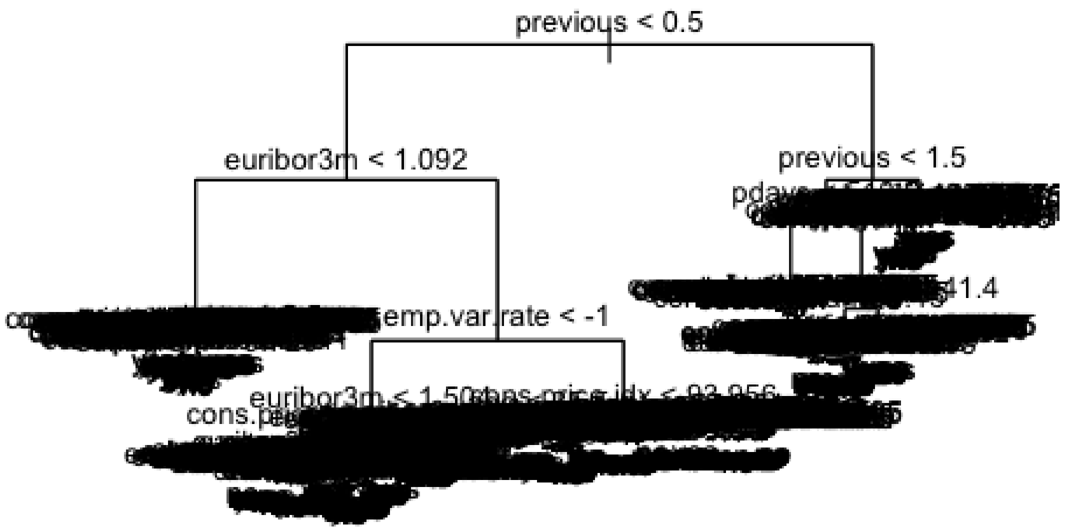
Success!

Decision Tree (Q4)

```
set.seed(1)
tree.data_reduced_gini <- tree(y ~., data_reduced, subset=train, split =
'gini') #Fitting the Tree with the Gini splitting criterion.
tree.pred_gini <- predict(tree.data_reduced_gini, data_reduced.test,
type="class") #Testing it on test set.

summary(tree.data_reduced_gini)

##
## Classification tree:
## tree(formula = y ~ ., data = data_reduced, subset = train, split = "gini")
## Variables actually used in tree construction:
##  [1] "previous"       "euribor3m"      "campaign"       "emp.var.rate"
##  [5] "cons.price.idx" "age"            "education"      "marital"
##  [9] "contact"        "housing"        "cons.conf.idx"  "loan"
## [13] "pdays"          "poutcome"
## Number of terminal nodes:  1256
## Residual mean deviance:  0.4276 = 5981 / 13990
## Misclassification error rate: 0.09505 = 1449 / 15244
```

We see that the training error rate is 9.505%. The residual mean deviance is 0.4276. Note that a small deviance indicates that our tree provides a good fit to the (training) data. Let us now display the tree structure using the plot() and text() functions.

```
plot(tree.data_reduced_gini)
text(tree.data_reduced_gini, pretty = 0, cex=0.8) #Pretty ugly tree I know
```



To properly evaluate the performance of the decision tree, we must estimate the test error instead of simply using only the training error. The following code block shows us the confusion matrix from which we can estimate the test error.

```
confusionMatrix(tree.pred_gini, data_reduced$y[-train])

## Confusion Matrix and Statistics
##
##           Reference
## Prediction    no   yes
##        no  12744  1348
##        yes   592   560
##
##                Accuracy : 0.8727
##                  95% CI : (0.8673, 0.878)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 0.7872
##
##                   Kappa : 0.3
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9556
##             Specificity : 0.2935
##          Pos Pred Value : 0.9043
##          Neg Pred Value : 0.4861
##              Prevalence : 0.8748
##          Detection Rate : 0.8360
##    Detection Prevalence : 0.9244
##       Balanced Accuracy : 0.6246
##
##        'Positive' Class : no
##
```

As seen above, building the tree on the training set and evaluating its performance on the test set leads to correct predictions for around 87.27% of the locations in the test data set. However, we have to be careful and instead look at the Balanced Accuracy as the proper metric since we have an imbalanced dataset (more on why is explained in Q7). The balanced accuracy is in fact 62.46 %, considerably lower than 87.27%.

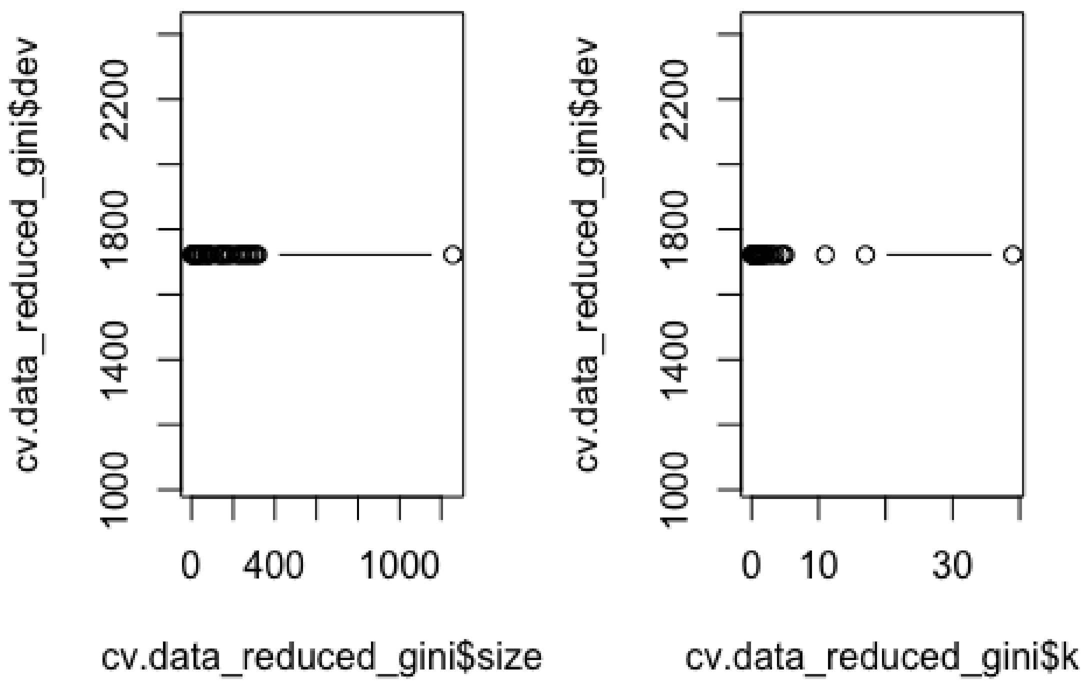
Let us now check whether pruning the tree might lead to improved results. For this, we will have to perform cross-validation in order to determine the optimal level of tree complexity. Cost complexity will be used to select a sequence of trees for consideration.

```
set.seed(3)
cv.data_reduced_gini <- cv.tree(tree.data_reduced_gini, FUN = prune.misclass)
names(cv.data_reduced_gini)

## [1] "size"   "dev"    "k"      "method"

cv.data_reduced_gini

## $size
##  [1] 1256  319  309  301  283  273  264  248  241  226  219  189  180  165
155
## [16]  146  136  129   97   85   81   75   71   55   49   39   34   30   25
23
## [31]   20   13    9    7    4    1
##
## $dev
##  [1] 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722
1722
## [16] 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722 1722
1722
## [31] 1722 1722 1722 1722 1722 1722
##
## $k
##  [1]       -Inf  0.0000000  0.1000000  0.1250000  0.1666667  0.2000000
##  [7]  0.2222222  0.2500000  0.2857143  0.3333333  0.4285714  0.5000000
## [13]  0.5555556  0.6666667  0.7000000  0.7777778  0.8000000  0.8571429
## [19]  1.0000000  1.0833333  1.2500000  1.3333333  1.5000000  1.7500000
## [25]  1.8333333  1.9000000  2.0000000  2.7500000  3.0000000  3.5000000
## [31]  4.3333333  4.7142857  5.0000000 11.0000000 17.0000000 39.0000000
##
```

```
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

“dev” corresponds to the cross validation error rate in this instance. Interestingly, we see that the cross-validation error rate is the same across all trees with different number of terminal nodes. We can verify this by plotting the error rate as a function of both size and k.

```
par(mfrow = c(1, 2))
plot(cv.data_reduced_gini$size, cv.data_reduced_gini$dev,type = "b")
plot(cv.data_reduced_gini$k, cv.data_reduced_gini$dev,type = "b")
```



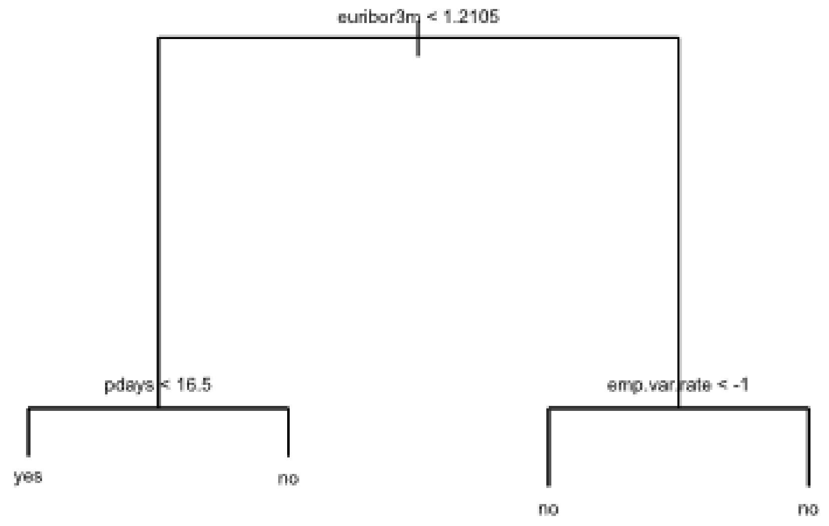
Now that we have trained and tested this decision tree, we will now perform the same steps, but this time use ‘deviance’ as the splitting criterion to examine any possible differences.

```
set.seed(1)
tree.data_reduced_deviance <- tree(y ~., data_reduced, subset=train, split =
'deviance')
tree.pred_deviance <- predict(tree.data_reduced_deviance, data_reduced.test,
type="class")
summary(tree.data_reduced_deviance)
```

```
##
## Classification tree:
## tree(formula = y ~ ., data = data_reduced, subset = train, split =
"deviance")
## Variables actually used in tree construction:
## [1] "euribor3m"    "pdays"        "emp.var.rate"
## Number of terminal nodes:  4
## Residual mean deviance:  0.6244 = 9516 / 15240
## Misclassification error rate: 0.1124 = 1714 / 15244
```

We see that the training error rate is 11.24%, which is higher than the decision tree where we used the gini criterion to split the nodes (9.505%). The residual mean deviance is 0.6244, which is also higher than the previous case, indicating that the tree with the ‘deviance’ split is a worse fit than the tree with the ‘gini’ split to the training data. However, to truly see if the first decision tree is better than the second (and not just overfitting the data), we need to evaluate the test error. Let us first display the tree structure using the plot() and text() functions.

```
plot(tree.data_reduced_deviance)
text(tree.data_reduced_deviance, pretty = 0, cex = .5) #Much cleaner and less
complex
```



In order to properly evaluate the performance of the decision tree, we must estimate the test error instead of simply using only the training error. The following code block shows us the confusion matrix from which we can estimate the test error.

```
confusionMatrix(tree.pred_deviance, data_reduced$y[-train])

## Confusion Matrix and Statistics
## 
##              Reference
## Prediction     no   yes
##        no   13189  1579
##        yes    147   329
## 
##                Accuracy : 0.8868
##                  95% CI : (0.8816, 0.8918)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 3.36e-06
## 
##                   Kappa : 0.2379
## 
##  Mcnemar's Test P-Value : < 2.2e-16
## 
##             Sensitivity : 0.9890
##             Specificity : 0.1724
##          Pos Pred Value : 0.8931
##          Neg Pred Value : 0.6912
##              Prevalence : 0.8748
##          Detection Rate : 0.8652
##    Detection Prevalence : 0.9688
##       Balanced Accuracy : 0.5807
## 
##        'Positive' Class : no
## 
```

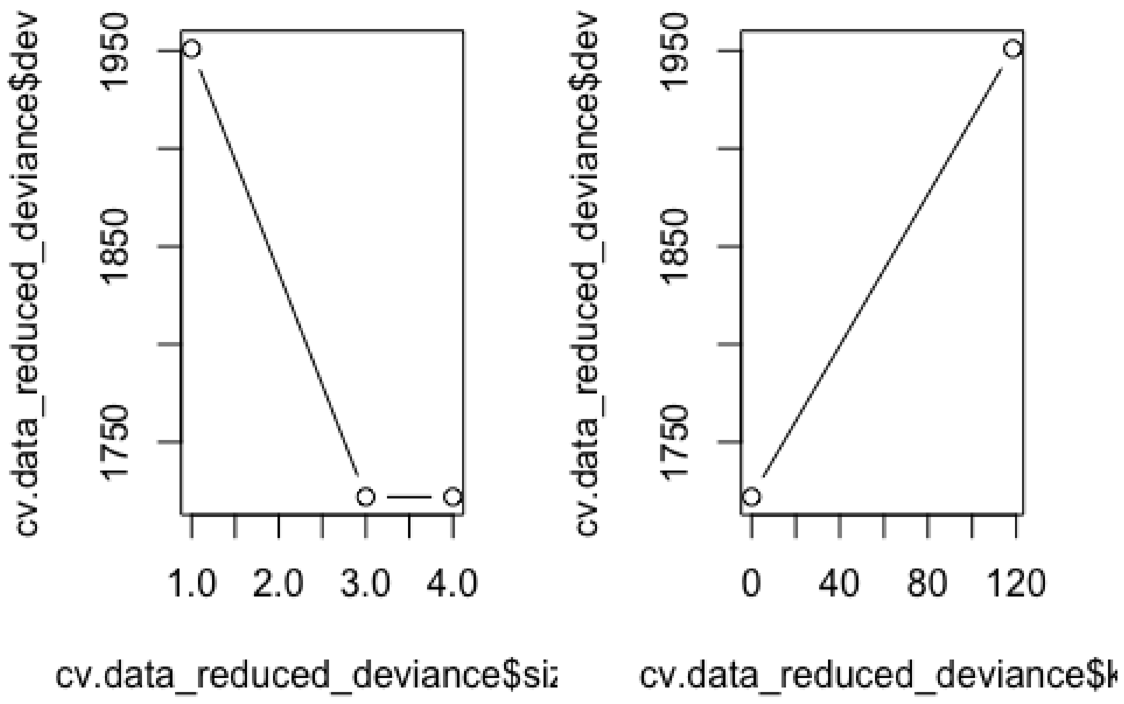
As seen above, building the second decision tree (where 'deviance' is the splitting criterion) on the training set and evaluating its performance on the test set leads to correct predictions for around 88.7% of the locations in the test data set. However, the balanced accuracy (which is the more meaningful metric) is lower, at 58.07%, indicating that the tree with the "deviance" splitting criterion is worse than the one with the "gini" criterion. Let us now consider whether pruning the tree might lead to improved results. For this, we will have to perform cross-validation in order to determine the optimal level of tree complexity. Cost complexity will be used to select a sequence of trees for consideration.

```
set.seed(3)
cv.data_reduced_deviance <- cv.tree(tree.data_reduced_deviance, FUN =
prune.misclass)
names(cv.data_reduced_deviance)

## [1] "size"   "dev"    "k"      "method"
```

```
cv.data_reduced_deviance

## $size
## [1] 4 3 1
##
## $dev
## [1] 1722 1722 1951
##
## $k
## [1]  -Inf   0.0 118.5
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

As seen above, the cross validation error rate is lowest using a decision tree of size 4 or 3. We can also verify this with a plot.

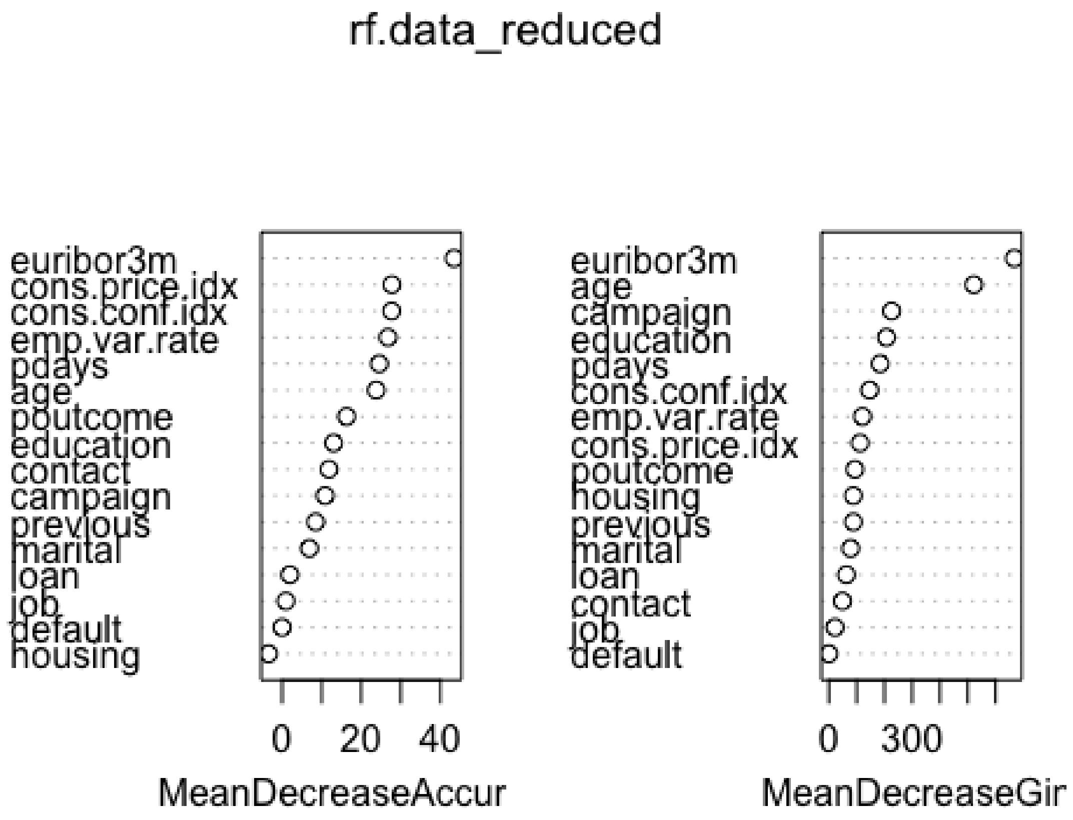
```
par(mfrow = c(1, 2))
plot(cv.data_reduced_deviance$size, cv.data_reduced_deviance$dev,type = "b")
plot(cv.data_reduced_deviance$k, cv.data_reduced_deviance$dev,type = "b")
```



Since the default tree we fitted with the deviance as the splitting criterion already had a size of 4 (4 terminal nodes), thereby leading to the specification with the lowest cross-validation rate, we will not perform any further configurations to the decision tree model.

Fitting Random Forest (Q5)

```r
set.seed(1)
rf.data_reduced <- randomForest(y~., data = data_reduced, subset = train,
mtry = 5, importance = TRUE)
varImpPlot(rf.data_reduced)
```

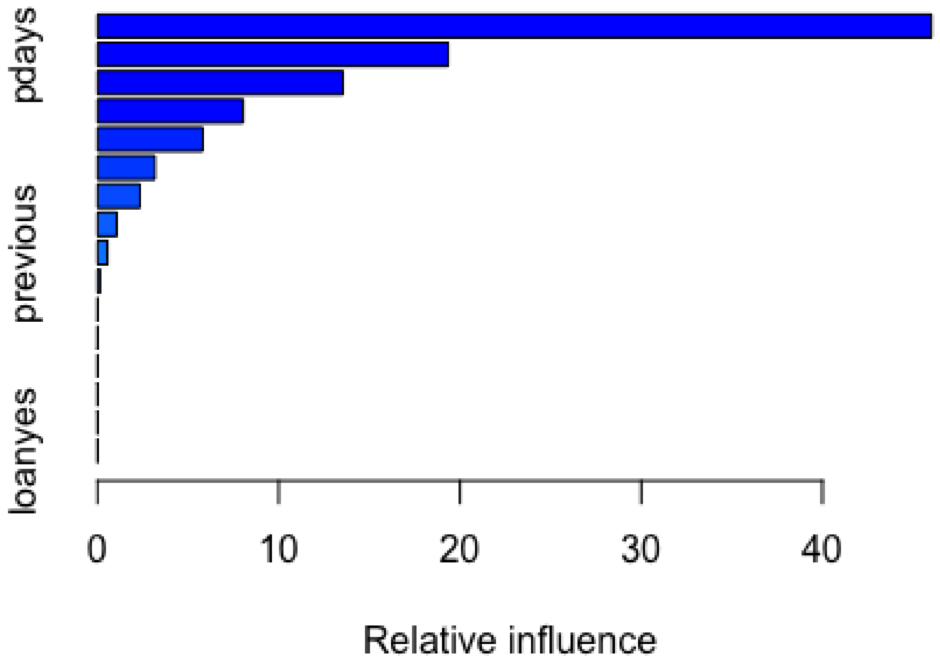


As we can see above, we can measure the variable importance using Mean Decrease Accuracy, and Mean Decrease Gini. The former is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model. The latter is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees. The higher the value of mean decrease accuracy or mean decrease gini score , higher the importance of the variable in the model. In the plot shown above, we can clearly see that 'euribor3m' is the most important variable. This is unsurprising since the bank is indeed advertising bank term deposits, and consumers will generally look at the interest rate to see if this product will indeed be an attractive option. If the euribor3m rate is relatively high, this will trickle down to banks offering higher interest rates in the future, which will eventually be beneficial for the consumers.

Fitting Gradient Boosted Trees (Q6)

While we have used the gbm library in class to fit a gradient boosted tree model, this time we will use the caret library to fit, test, and evaluate our model. This is because the caret library is a lot more convenient when it comes to machine learning algorithms and makes it easier to evaluate model performance.

```
tc = trainControl(method = "repeatedcv", number = 10)
boost.data_reduced = train(y ~., data=data_reduced[train, ], method="gbm",
trControl=tc, verbose=F)
summary(boost.data_reduced)
```



```
##                                 var    rel.inf
## euribor3m                 euribor3m 46.0146548
## pdays                         pdays 19.3477925
## emp.var.rate           emp.var.rate 13.5488500
## cons.conf.idx         cons.conf.idx  8.0275601
## poutcomesuccess     poutcomesuccess  5.8153306
## cons.price.idx       cons.price.idx  3.1654487
## age                             age  2.3388553
## contacttelephone contacttelephone  1.0525366
## previous                   previous  0.5493091
## campaign                   campaign  0.1396623
## jobunemployed         jobunemployed  0.0000000
## maritalsingle         maritalsingle  0.0000000
## education                 education  0.0000000
## defaultyes               defaultyes  0.0000000
## housingyes               housingyes  0.0000000
## loanyes                     loanyes  0.0000000
```

As seen above, our gradient boosted tree model also considers the euribor3m feature as the most important feature. Interestingly, it considers pdays as the second most important feature, which was also ranked pretty high in our random forest model.

Testing Models & Comparing Performances (Q7)

As we hinted earlier, when comparing performances across models, we must be very careful. Specifically, since our data is heavily imbalanced, accuracy is no longer a viable metric. This is because of the "accuracy paradox". The accuracy paradox is the paradoxical finding that accuracy is not a good metric for predictive models when classifying in predictive analytics. This is because a simple model may have a high level of accuracy but be too crude to be useful. For example, if the incidence of category A is dominant, being found in 99% of cases, then predicting that every case is category A will have an accuracy of 99%. This is why we instead will use "Balanced Accuracy". Balanced accuracy is based on two more commonly used metrics: sensitivity (also known as true positive rate or recall) and specificity (also known as true negative rate, or 1 – false-positive rate). These metrics are more useful when we have imbalanced data sets. Balanced Accuracy is actually simply the arithmetic mean of Sensitivity and Specificity.

Taking this into account, we will now evaluate the performance of our models.

Performance of Random Forest

```
set.seed(1)
yhat.rf = predict(rf.data_reduced,newdata=data_reduced[-train,]) #Testing RF
Model on New Data
confusionMatrix(yhat.rf, data_reduced$y[-train])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    no   yes
##        no  12951  1351
##        yes   385   557
##
##                Accuracy : 0.8861
##                  95% CI : (0.881, 0.8911)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 1.065e-05
##
```

```
##                  Kappa : 0.3359
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9711
##             Specificity : 0.2919
##          Pos Pred Value : 0.9055
##          Neg Pred Value : 0.5913
##              Prevalence : 0.8748
##          Detection Rate : 0.8496
##    Detection Prevalence : 0.9382
##       Balanced Accuracy : 0.6315
##
##        'Positive' Class : no
##
```

As we can see, the (balanced) accuracy of the random forest is 63.15%, which is pretty good, and in fact, better than our decision tree models, which had (balanced) accuracies of 62.46% and 58.07%.

Performance of Gradient Boosted Trees

```
set.seed(1)
yhat.boost = predict(boost.data_reduced, data_reduced[-train, ])
confusionMatrix(yhat.boost, data_reduced$y[-train])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    no   yes
##        no  13174  1554
##        yes   162   354
##
##                Accuracy : 0.8874
##                  95% CI : (0.8823, 0.8924)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 9.95e-07
##
##                   Kappa : 0.2522
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9879
##             Specificity : 0.1855
##          Pos Pred Value : 0.8945
##          Neg Pred Value : 0.6860
##              Prevalence : 0.8748
##          Detection Rate : 0.8642
##    Detection Prevalence : 0.9662
##       Balanced Accuracy : 0.5867
```

```
##
##        'Positive' Class : no
##
```

While at first glance it may seem that our Gradient Boosted Trees model performed better than our RF model, with a 88.7% accuracy, in fact it performed worse since the balanced accuracy is 58.67%. While usually boosted trees yield superior results, they are also more susceptible to overfitting, and are also harder to tune, which might primarily explain why our gradient boosted tree model performed worse. Indeed, Random Forest models are easier to tune with fewer hyper-parameters, and given that RF models involve the development of independent decision trees on different samples in the data, they are less susceptible to over-fitting.

As a reminder, let us also check the performance of our decision trees (as seen in Q4).

Performance of Decision Tree (with Gini Split Criterion)

```
set.seed(1)
confusionMatrix(tree.pred_gini, data_reduced$y[-train])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    no   yes
##        no  12744  1348
##        yes   592   560
##
##                Accuracy : 0.8727
##                  95% CI : (0.8673, 0.878)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 0.7872
##
##                   Kappa : 0.3
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9556
##             Specificity : 0.2935
##          Pos Pred Value : 0.9043
##          Neg Pred Value : 0.4861
##              Prevalence : 0.8748
##          Detection Rate : 0.8360
##    Detection Prevalence : 0.9244
##       Balanced Accuracy : 0.6246
##
##        'Positive' Class : no
##
```

Performance of Decision Tree (with Deviance Split Criterion)

```r
set.seed(1)
confusionMatrix(tree.pred_deviance, data_reduced$y[-train])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    no   yes
##        no  13189  1579
##        yes   147   329
##
##                Accuracy : 0.8868
##                  95% CI : (0.8816, 0.8918)
##     No Information Rate : 0.8748
##     P-Value [Acc > NIR] : 3.36e-06
##
##                   Kappa : 0.2379
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9890
##             Specificity : 0.1724
##          Pos Pred Value : 0.8931
##          Neg Pred Value : 0.6912
##              Prevalence : 0.8748
##          Detection Rate : 0.8652
##    Detection Prevalence : 0.9688
##       Balanced Accuracy : 0.5807
##
##        'Positive' Class : no
##
```

Looking at the two decision trees, we see that they perform worse than the RF model, which is not surprising due to the fact that RF models reduce overfitting by considering a random subset of features at each split and through bagging. However, the decision tree with the Gini splitting criterion has a higher balanced accuracy than the gradient boosted model. As explained before, this might be because the boosted model overfitted the data, and the fact that we did not properly tune the hyperparameters.

In conclusion, the RF model performed the best, followed by our decision tree model with the gini criterion, followed by gradient boosted tree, and finally the decision tree model with the deviance criterion.