Yes or No? Classifying Term Deposit Subscriptions using Random Forest

Aiden Liu

Variable Description

-age

-job - type of job (categorical:‘admin.’,‘blue-collar’,‘entrepreneur’,‘housemaid’,‘management’,‘retired’,‘self-employed’,‘services’,‘student’,‘technician’,‘unemployed’,‘unknown’)

-marital - marital status (categorical: ‘divorced’,‘married’,‘single’,‘unknown’; note: ‘divorced’ means divorced or widowed)

-education - (categorical: ‘basic.4y’,‘basic.6y’,‘basic.9y’,‘high.school’,‘illiterate’,‘professional.course’,‘university.degree’,‘unknown’)

-default - has credit in default?

-balance - average yearly balance

-housing - has housing loan?

-loan - has personal loan?

-contact - contact communication type (categorical: ‘cellular’,‘telephone’)

-day_of_week - last contact day of the week

-month - last contact month of year (categorical: ‘jan’, ‘feb’, ‘mar’, ..., ‘nov’, ‘dec’)

-duration - last contact duration, in seconds (numeric)

-day - date of the month

-campaign - number of contacts performed during this campaign and for this client (numeric, includes last contact)

-pdays - number of days that passed by after the client was last contacted from a previous campaign (numeric; -1 means client was not previously contacted)

-previous - number of contacts performed before this campaign and for this client

-poutcome - outcome of the previous marketing campaign (categorical: ‘failure’,‘nonexistent’,‘success’)

-y - has the client subscribed a term deposit?

Libraries

```
library(readr)
library(dplyr)
library(rpart)
library(rpart.plot)
library(randomForest)
library(party)
library(caret)
library(caTools)
library(pROC)
```

Reading in the data

```
bank.data = read_delim("bank.csv", delim = ";",
    escape_double = FALSE, trim_ws = TRUE)
```

Inspecting the class of each variable

```
sapply(bank.data, class)
```

```
##         age         job     marital   education     default     balance 
##   "numeric" "character" "character" "character" "character"   "numeric" 
##     housing        loan     contact         day       month    duration 
## "character" "character" "character"   "numeric" "character"   "numeric" 
##    campaign       pdays    previous    poutcome           y 
##   "numeric"   "numeric"   "numeric" "character" "character"
```

Adjusting all binary variables into factors

```
bank.data$default = as.factor(bank.data$default)
bank.data$housing = as.factor(bank.data$housing)
bank.data$loan = as.factor(bank.data$loan)
bank.data$y = as.factor(bank.data$y)
```

Checking for any NA values in the dataset

```
any(is.na(bank.data)) # Finding if there are any NA values in our dataset
```

```
## [1] FALSE
```

Upon inspection, there are unknown inputs across different columns in the dataset.

Some inputs are unknown. Therefore I will remove them since we have a relatively large number of observations

```
adj.bank.data = subset(bank.data, contact!= "unknown")
adj.bank.data = subset(adj.bank.data, job!= "unknown")
adj.bank.data = subset(adj.bank.data, marital!= "unknown")
adj.bank.data = subset(adj.bank.data, education!= "unknown")
adj.bank.data = subset(adj.bank.data, loan!= "unknown")
adj.bank.data = subset(adj.bank.data, month!= "unknown")
```

A large number of the outcomes of the previous marketing campaign are unknown which isn't very useful in comparative analysis across levels.

```
print(sum(bank.data$poutcome == "unknown")) # Finding the number of outcomes
of previous marketing campaign which are unknown

## [1] 3705
```

Remove unnecessary columns

```
adj.bank.data$poutcome = NULL
adj.bank.data$day = NULL # Removing date of the month
adj.bank.data$month = NULL
```

A large number of previous number of days that passed by after the client was last contacted from a previous campaign are -1, indicating most of the clients have not been contacted before. Therefore, I will remove pdays.

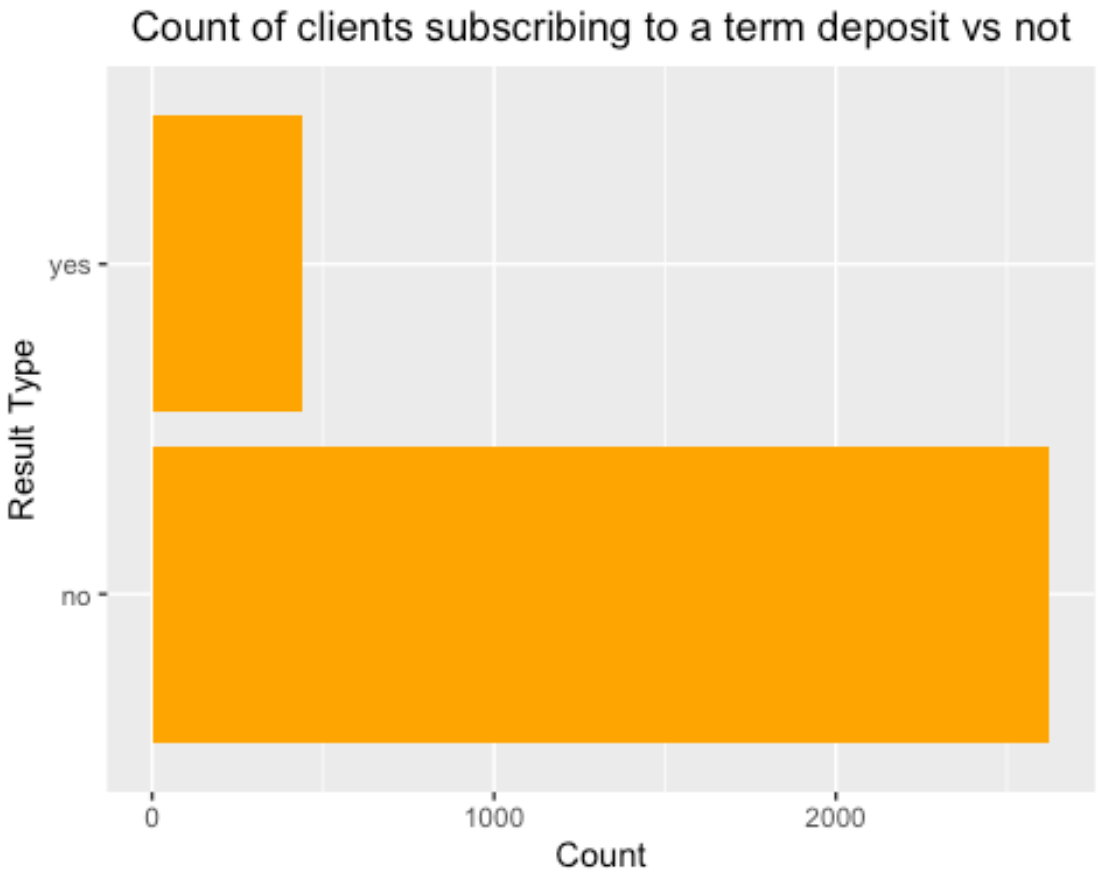
```
print(sum(bank.data$pdays == -1))

## [1] 3705

# Remove pdays column
adj.bank.data$pdays = NULL
```

Is the dataset imbalanced?

```
ggplot(data = adj.bank.data, mapping = aes(y=y))+
  geom_bar(fill="orange")+
  labs(x="Count", y="Result Type", title="Count of clients subscribing to a
term deposit vs not")+
    theme(plot.title = element_text(hjust = 0.5))
```



Massively imbalanced dataset where there more negative case is the majority class, and the positive case is the minority class.

Upsampling the minority class (yes to subscribing) to create a balanced dataset

```
balanced_data = upSample(x = adj.bank.data[, -which(names(adj.bank.data) ==
"y")],
                         y = adj.bank.data$y)
colnames(balanced_data)[colnames(balanced_data) == "Class"] = "y"
```

Double checking the dataset is balanced

```
table(balanced_data$y)

##
##   no  yes
## 2626 2626
```

Splitting the data into training and test sets

```
# Splitting data in train and test data
split = sample.split(balanced_data$y, SplitRatio = 0.7) # 70-30 split for
training vs test set
```

```r
train.data = subset(balanced_data, split == "TRUE") # Subsetting dataset into training set
test.data = subset(balanced_data, split == "FALSE") # Subsetting dataset into test set
test.y = test.data$y
```

Double checking the response is now balanced between positive and negative class.

```r
table(train.data$y)
```

```
##
##   no  yes
## 1838 1838
```

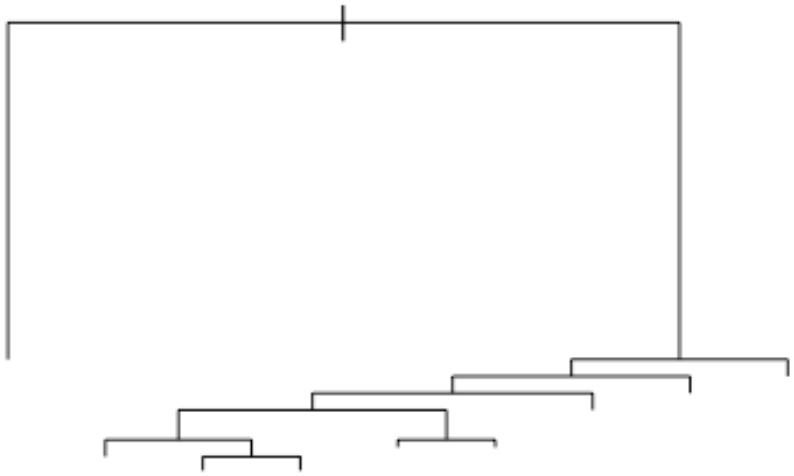
Fitting a decision tree with all possible variables

```r
tree_model = rpart(y ~ ., data = train.data, method = "class")

print(tree_model) # Inspecting the model
```

```
## n= 3676
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 3676 1838 no (0.5000000 0.5000000)
##     2) duration< 211.5 1427  266 no (0.8135950 0.1864050) *
##     3) duration>=211.5 2249  677 yes (0.3010227 0.6989773)
##       6) duration< 557.5 1465  566 yes (0.3863481 0.6136519)
##        12) previous< 0.5 892  432 yes (0.4843049 0.5156951)
##          24) age< 60.5 811  388 no (0.5215783 0.4784217)
##            48) duration< 349.5 428  157 no (0.6331776 0.3668224)
##              96) job=blue-collar,entrepreneur,housemaid,retired,self-
employed,services 143   23 no (0.8391608 0.1608392) *
##              97) job=admin.,management,student,technician,unemployed 285
134 no (0.5298246 0.4701754)
##               194) campaign>=1.5 168   57 no (0.6607143 0.3392857) *
##               195) campaign< 1.5 117   40 yes (0.3418803 0.6581197) *
##            49) duration>=349.5 383  152 yes (0.3968668 0.6031332)
##              98) balance>=5550.5 20    0 no (1.0000000 0.0000000) *
##              99) balance< 5550.5 363  132 yes (0.3636364 0.6363636) *
##          25) age>=60.5 81    9 yes (0.1111111 0.8888889) *
##        13) previous>=0.5 573  134 yes (0.2338569 0.7661431) *
##       7) duration>=557.5 784  111 yes (0.1415816 0.8584184) *
```

```r
plot(tree_model) # Visualising the tree
```



Creating predictions and confusion matrix based on the decision tree

```
predictions = predict(tree_model, newdata = test.data, type = "class") #
Predictions

conf_matrix = confusionMatrix(predictions, test.y) # Building Confusion
matrix
print(conf_matrix$byClass)
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.7550761            0.8058376            0.7954545
##       Neg Pred Value            Precision               Recall
##            0.7669082            0.7954545            0.7550761
##                   F1           Prevalence       Detection Rate
##            0.7747396            0.5000000            0.3775381
## Detection Prevalence    Balanced Accuracy
##            0.4746193            0.7804569
```

Not amazing predictive statistics from the decision tree, but we can do better.

How about a Random Forest method? Random Forest delivers highly accurate predictions even with large datasets, effectively handles missing data without sacrificing accuracy,

eliminates the need for normalisation or standardisation, and reduces the risk of overfitting by combining multiple decision trees.

```
rf.model = randomForest(y~., data = train.data, importance = TRUE)
```

Inspecting the random forest model

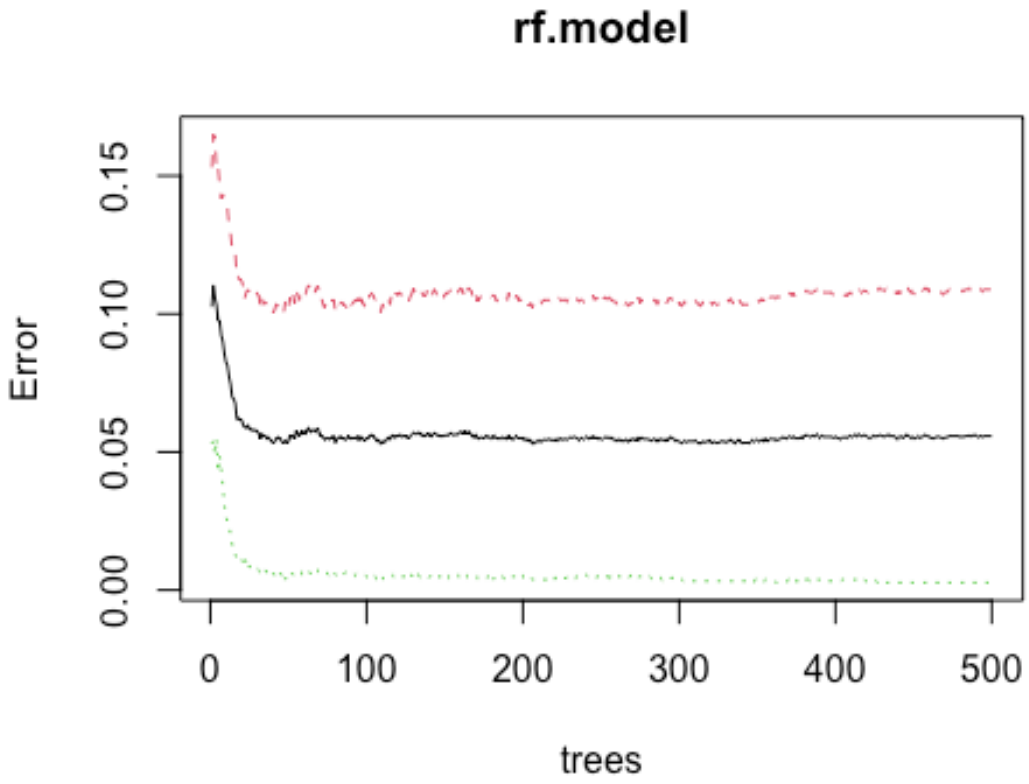
```
print(rf.model)
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = train.data, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 5.58%
## Confusion matrix:
##       no  yes class.error
## no  1638  200 0.108813928
## yes    5 1833 0.002720348
```

The OBB error rate suggests that the model performs well, as the error rate is relatively low.

Looking at the error rates

```
plot(rf.model)
```



The black line in the plot represents the overall OOB (Out-of-Bag) error rate, while the red dashed line corresponds to the error rate for the negative class The green dashed line indicates the error rate for the positive class As the number of trees increases, the error rates stabilise, showing that the random forest has enough trees to generalise effectively. The overall OOB error, represented by the black line, stabilises around 5%, which aligns with the previously observed result of 5.41%. For the class-specific error rates, the error for the "no" class, shown by the red dashed line, is higher and stabilises around 10%, reflecting the higher class error seen in the confusion matrix. On the other hand, the error for the "yes" class, depicted by the green dashed line, is significantly lower, stabilising near 0.6%, which is consistent with its strong predictive performance.

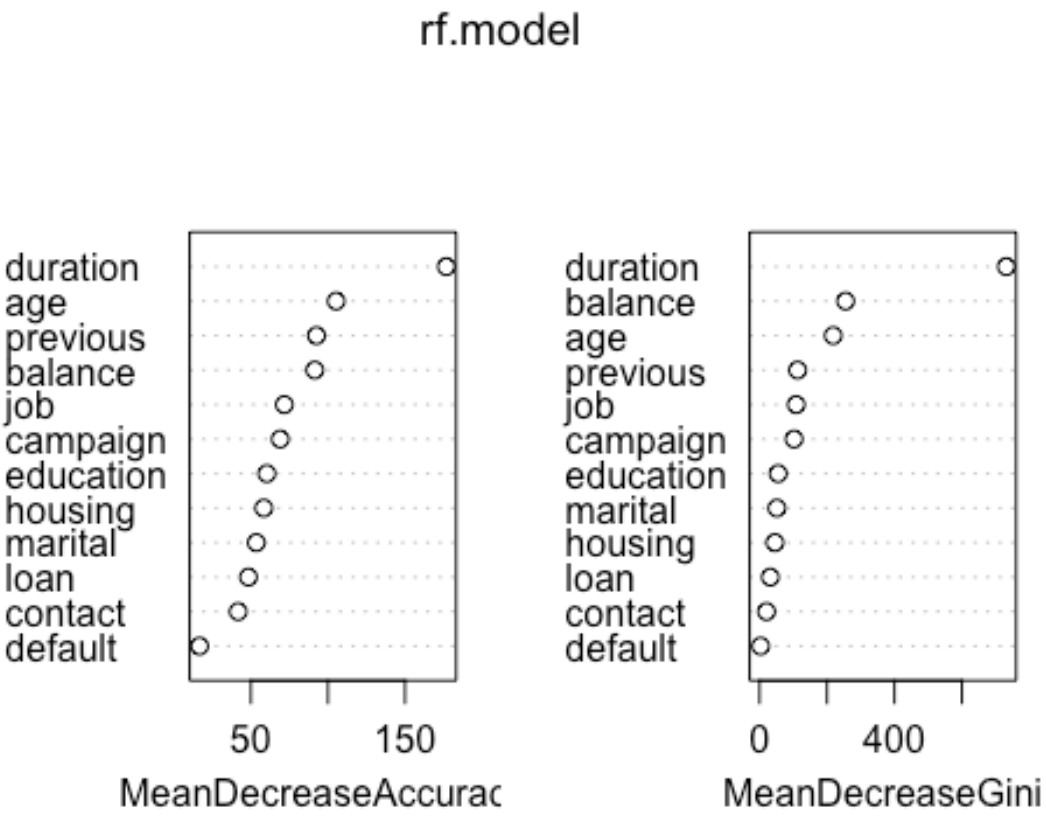
Checking the important features

```
# Feature importance
importance(rf.model)
```

```
##                   no       yes MeanDecreaseAccuracy MeanDecreaseGini
## age        11.552721 110.90072            105.31570       218.367843
## job         3.351322  72.07434             71.69306       108.682558
## marital    -1.200544  54.60265             53.51494        51.316769
## education   9.734797  60.78738             60.34932        55.141780
## default    -1.402382  17.89350             16.69411         3.226432
```

```
## balance      5.346546  93.33236              91.47393        255.049320
## housing     23.071357  57.96925              58.38078         45.626048
## loan         8.301759  48.56057              48.39979         31.951236
## contact      2.910859  43.11277              41.73910         20.580632
## duration   103.434282 181.45752             176.76255        731.449827
## campaign    10.391093  70.01819              69.01995        102.358559
## previous    26.507701  94.11920              92.55393        112.926906
```

```
varImpPlot(rf.model)
```



duration is the most critical variable, as excluding it would lead to the largest drop in accuracy, followed by age and balance (average of the two plots). Similarly, duration is the most important variable when it comes to contributing to the homogeneity of the nodes and leaves in the decision trees.

The bottom 3 variables in both plots: default, contact, and loan are the least important when it comes to a drop in accuracy and the Gini.

Therefore, I'll refit the model and omit default, contact, and loan.

```
new.train = train.data
# new training data without 'default', 'contact', and 'loan'.
new.train$default = NULL
new.train$contact = NULL
```

```r
new.train$loan = NULL

new.test = test.data
# new test data without 'default', 'contact', and 'loan'.
new.test$default = NULL
new.test$contact = NULL
new.test$loan = NULL

# adjusted Random Forest model without default, contact, and loan
adj.rf.model = randomForest(y~., data = new.train, importance=TRUE)
```

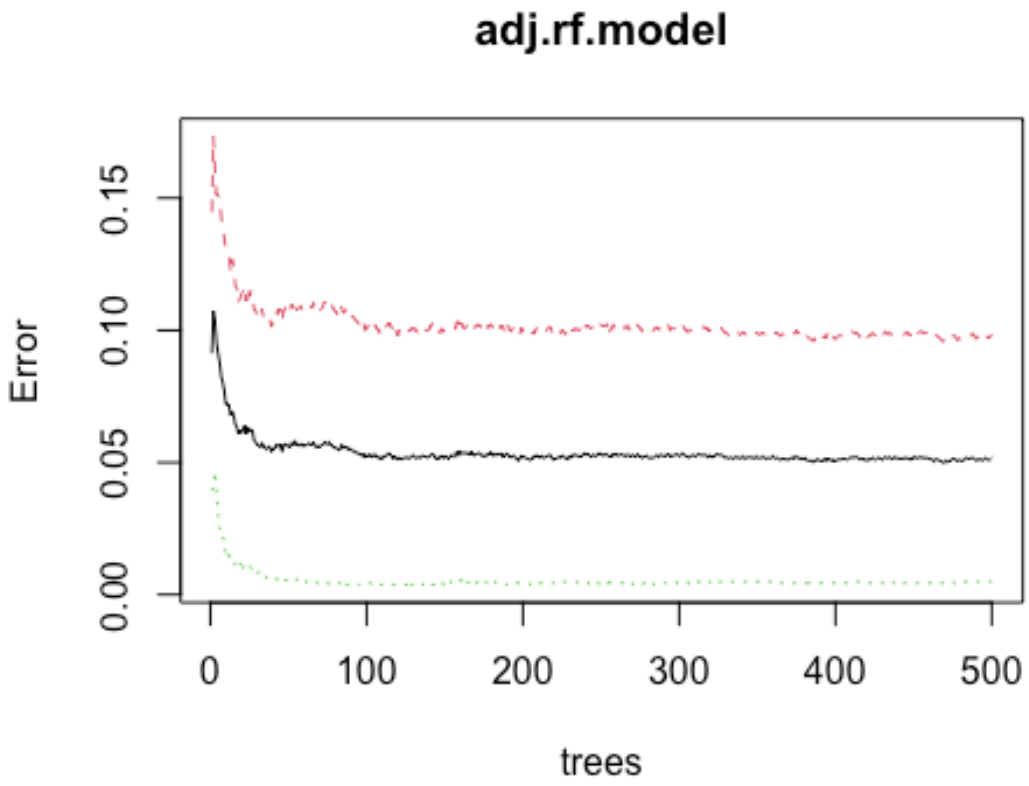
Inspecting the adjusted random forest model

```r
# Evaluate the model
print(adj.rf.model)

##
## Call:
##  randomForest(formula = y ~ ., data = new.train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 5.17%
## Confusion matrix:
##       no  yes class.error
## no  1657  181 0.098476605
## yes    9 1829 0.004896627
```

A lower OOB error rate indicates an improvement from the original model.

```r
plot(adj.rf.model)
```

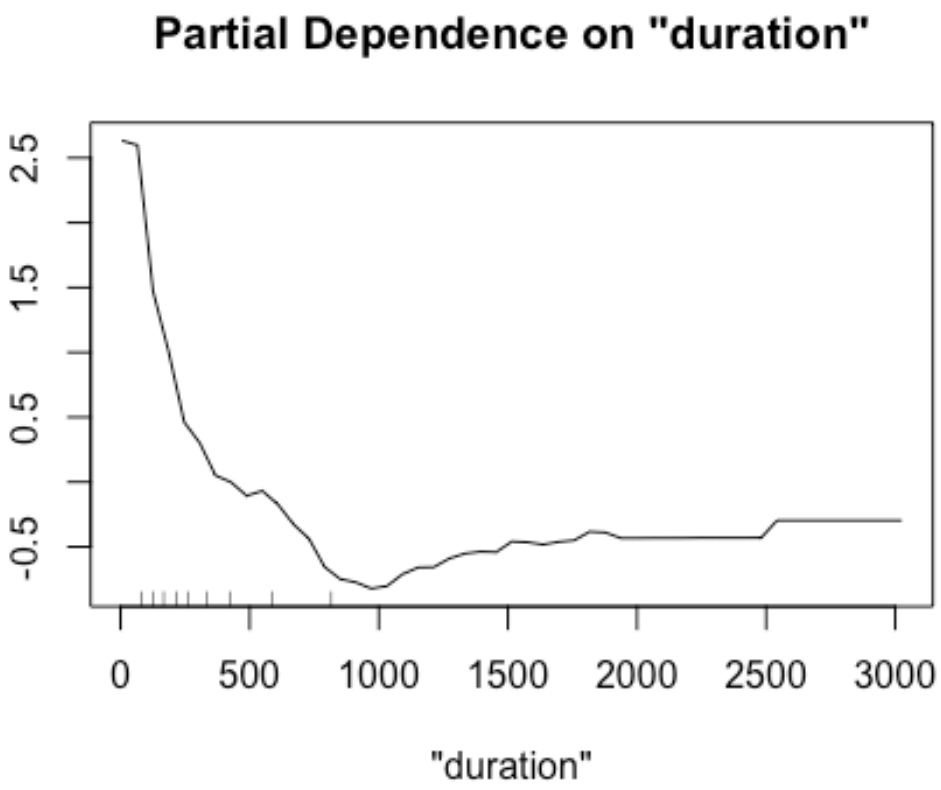


Similar story to the original model rf.model.

Let's inspect the relationship between the most important variables and the probability of a yes to subscribing to the term deposits.

Starting with duration

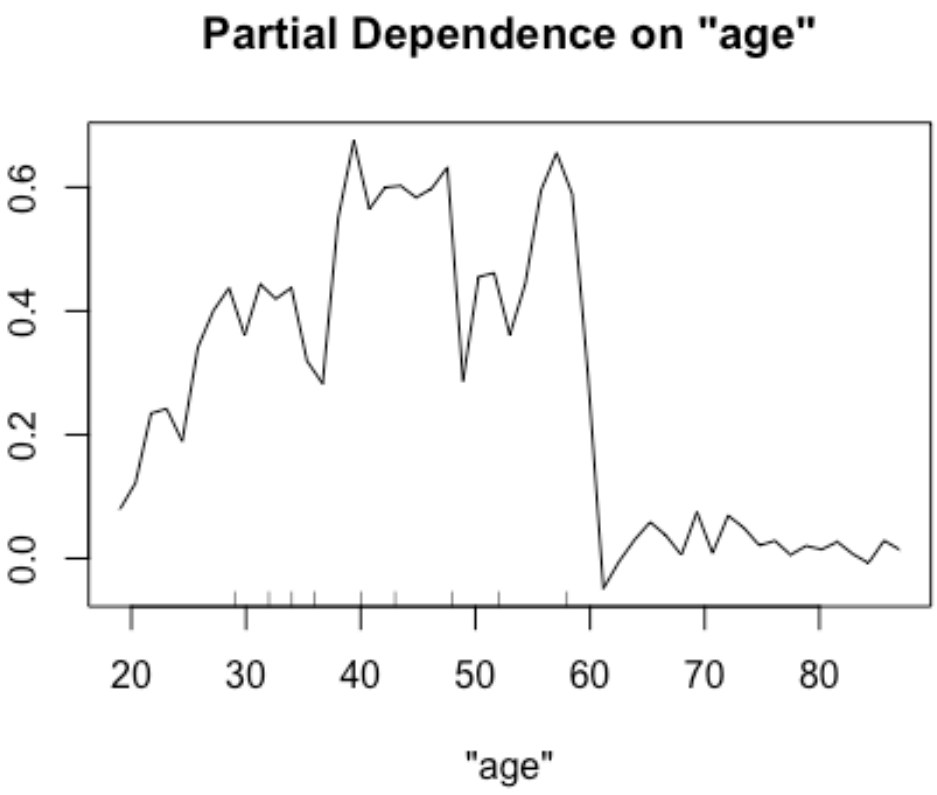
```
# Inspecting the relationship of the last contact duration
partialPlot(adj.rf.model, new.train, x.var = "duration")
```



As the length of the last contact duration increases, the probability of a subscription to the term deposit decreases. Key message: keep marketing short and sweet to minimise a 'no' to the term deposit subscription.

How about age?

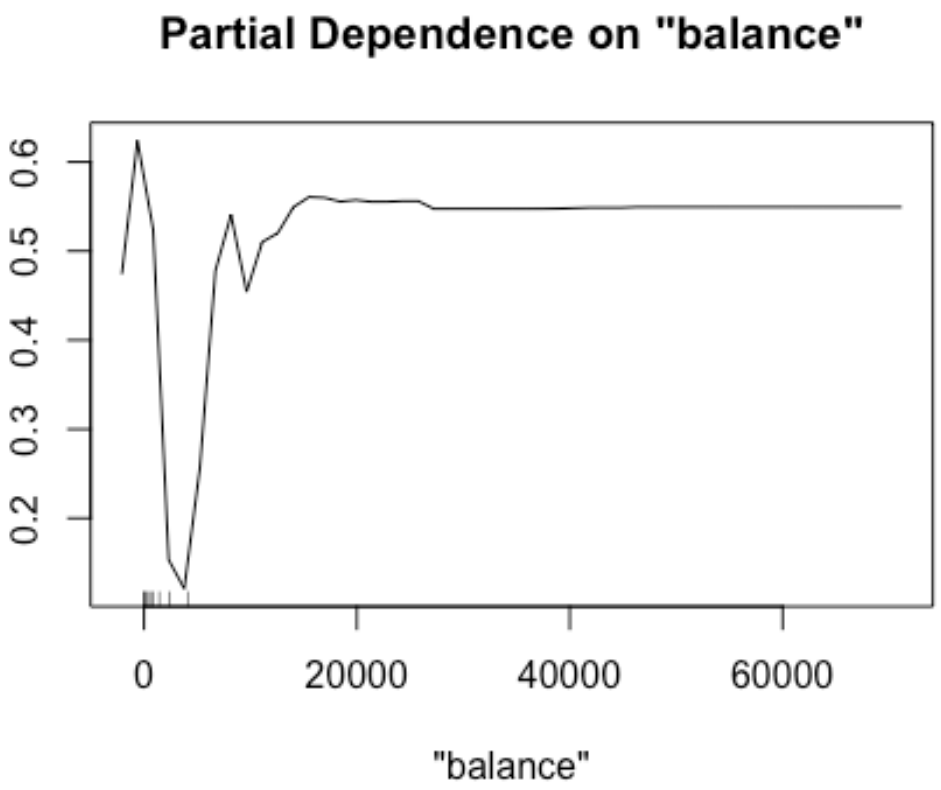
```
partialPlot(adj.rf.model, new.train, x.var = "age")
```



Upwards overall trend until just before 45, then there is a downwards trend up until 50. Moreover, from 50 to 60, there is an overall upwards trend with some dips in between. Just before age 60, there is a sharp downwards dip until just after age 60. Finally, there is a gradual rise post-60, then some indication of convergence. Overall, from ages 20 to 45, there is the highest probability of attaining a 'yes' while post-60 onwards is the lowest probability.

What about balance - average yearly balance?

```
# Inspecting the relationship of the last contact duration
partialPlot(adj.rf.model, new.train, x.var = "balance")
```



Looking from 0 onwards, there is an increasing trend as balance increases. In other words, the probability of a 'yes' to a subscription towards term deposits increases as balance increases.

Creating predictions and constructing the confusion matrix for the Random Forest model

```
# Creating predictions based on the fitted model and test data
predictions.rf = predict(adj.rf.model, newdata = new.test, type = "class")

conf_matrix.rf = confusionMatrix(predictions.rf, test.y) # Creating the
confusion matrix for the Random Forest model
print(conf_matrix.rf$byClass)
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.9060914            0.9936548            0.9930459
##       Neg Pred Value            Precision               Recall
##            0.9136523            0.9930459            0.9060914
##                   F1           Prevalence       Detection Rate
##            0.9475780            0.5000000            0.4530457
## Detection Prevalence    Balanced Accuracy
##            0.4562183            0.9498731
```

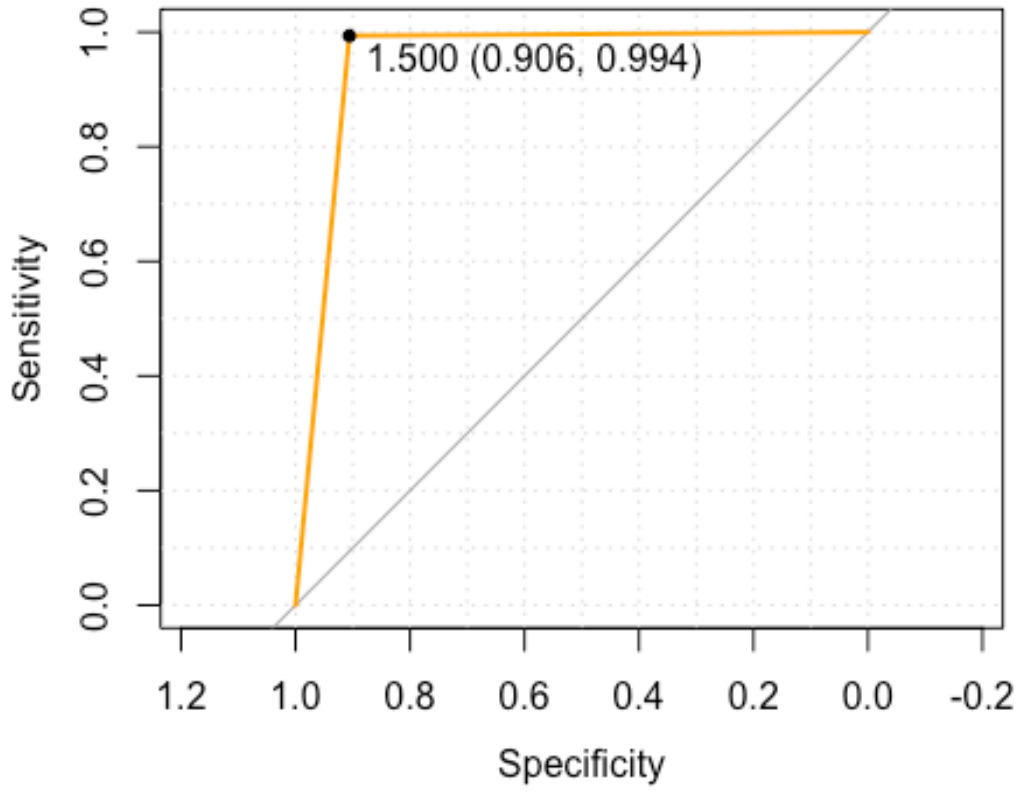
Good predictive model statistics overall.

```
# Estimate of AUC
roc_curve = roc(test.y, as.numeric(predictions.rf)) # ROC curve

## Setting levels: control = no, case = yes

## Setting direction: controls < cases

plot(roc_curve, grid=TRUE, col="orange",print.thres = "best") #Plot ROC curve
alongside the point that maximises both sensitivity and specificity
```



What's the AUC for the ROC curve?

```
print(auc(roc_curve))

## Area under the curve: 0.9499
```

The adjusted Random Forest model (adj.rf.model) is a great predictor of the binary classes. The model is good at distinguishing between a 'yes' or a 'no' in the response.

Final model

```
# Final model
print(adj.rf.model)
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = new.train, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##         OOB estimate of  error rate: 5.17%
## Confusion matrix:
##       no  yes class.error
## no  1657  181 0.098476605
## yes    9 1829 0.004896627
```

Actionable insight

The most important features of the marketing campaign is duration - last contact duration, in seconds, age, and balance - average yearly balance.

As per the PDP, the probability of a yes to the subscription to the term deposit decreases as the last contact duration increases. Continuing on the topic of PDP, with regard to ages; from ages 20 to 45, there is the highest probability of attaining a 'yes' while post-60 onwards is the lowest probability. There is an increasing trend as balance increases. In other words, the probability of a 'yes' to a subscription towards term deposits increases as balance increases.

Therefore there are 3 key takeaways from the PDP plots: - Keep the duration of the phone call short and sweet to promote likelihood of a 'yes' - Target ages groups 20 to 45 years old, and minimise efforts to post-60 year old clients - Call clients with sufficient and stable balances; Established clients should be prioritised.

Saving the model into a file for access

```
saveRDS(adj.rf.model, "bankmarketing_rf.rds")
```