Data Analytics C3T2

Objectives

One of the objectives of the survey was to find out which of two brands of computers our customers prefer.

I would like you to run and optimize at least two different decision tree classification methods in R - C5.0 and RandomForest

Importing Data & Libraries

```
require(pacman)
```

```
## Loading required package: pacman
```

```
pacman:: p_load(pacman, dplyr, GGally, ggplot2, ggrepel, patchwork, gifski, ggforce, ggthemes, maps, sf
```

```
CR <- import("CompleteResponses.csv")
SI <- import("SurveyIncomplete.csv")
```

EDA

```
names(CR)
```

```
## [1] "salary"  "age"     "elevel"  "car"     "zipcode" "credit"  "brand"
```

```
str(CR)
```

```
## 'data.frame':    9898 obs. of  7 variables:
##  $ salary : num  119807 106880 78021 63690 50874 ...
##  $ age    : int  45 63 23 51 20 56 24 62 29 41 ...
##  $ elevel : int  0 1 0 3 3 3 4 3 4 1 ...
##  $ car    : int  14 11 15 6 14 14 8 3 17 5 ...
##  $ zipcode: int  4 6 2 5 4 3 5 0 0 4 ...
##  $ credit : num  442038 45007 48795 40889 352951 ...
##  $ brand  : int  0 1 0 1 0 1 1 1 0 1 ...
```

```
#Changing brand to Categories and Assigning Brand Names
CR$brand <- as.factor(CR$brand)
levels(CR$brand) <- c("Acer", "Sony")
```

Data Split

```r
set.seed(107)

inTrain <- createDataPartition(y = CR$brand, p = .75, list = FALSE)
training <- CR[ inTrain,]
testing <- CR[-inTrain,]
```

Modeling

```r
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 3, classProbs = TRUE)

cl <- makeCluster(6)
registerDoParallel(cl)
GBM_Fit1 <- train(brand ~ ., data = training, method = "gbm", tuneLength = 1, trControl = ctrl, metric=
GBM_Fit2 <- train(brand ~ ., data = training, method = "gbm", tuneLength = 2, trControl = ctrl, metric=
GBM_Fit3 <- train(brand ~ ., data = training, method = "gbm", tuneLength = 3, trControl = ctrl, metric=

RF_Fit1 <- train(brand~., data = training, method = "rf", tuneLength = 1, trControl=ctrl, metric="Accura
RF_Fit2 <- train(brand~., data = training, method = "rf", tuneLength = 2, trControl=ctrl, metric="Accura
RF_Fit3 <- train(brand~., data = training, method = "rf", tuneLength = 3, trControl=ctrl, metric="Accura

RF_Fit4 <- train(brand~., data = training, method = "rf", tuneLength = 1, metric="Accuracy", trControl=
stopCluster(cl)
```

GBM Results

By running `GBM_Fit#` on R. Since I specified a two Class Summary (A specialized function for 2 class data to measure performance) in the control parameters, the command returns the area under the ROC curve. ROC takes into account the Rate of True Positives and the Rate of False Positives, an ROC of 1.0 means 100% accurate predictions.

Changing the interaction depth allows for greater accuracy in the case of GBM. Also, by running `GBM_Fit3` I get the results from the other two as well.

```r
GBM_Fit3
```

```
## Stochastic Gradient Boosting
##
## 7424 samples
##    6 predictor
##    2 classes: 'Acer', 'Sony'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 6681, 6681, 6681, 6681, 6682, 6683, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7271866  0.4245586
##   1                  100      0.7255701  0.4198603
##   1                  150      0.7260646  0.4203436
```

```
##   2                  50      0.8210322  0.6250693
##   2                 100      0.8829461  0.7564531
##   2                 150      0.9071027  0.8047286
##   3                  50      0.8774700  0.7468892
##   3                 100      0.9007271  0.7927668
##   3                 150      0.9176981  0.8264262
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
```

Random Forest Results

Using this Random Forest command, it yields ROC numbers which don't change between iterations due to the nature of Random Forest. Furthermore, between the three iterations, feature importance scores dont seem to change that much.

```
RF_Fit1
```

```
## Random Forest
##
## 7424 samples
##    6 predictor
##    2 classes: 'Acer', 'Sony'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 6682, 6681, 6683, 6681, 6682, 6681, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9157691  0.8214673
##
## Tuning parameter 'mtry' was held constant at a value of 2
```

```
RF_Fit2
```

```
## Random Forest
##
## 7424 samples
##    6 predictor
##    2 classes: 'Acer', 'Sony'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 6683, 6682, 6681, 6682, 6681, 6681, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
```

```
##   2     0.9150950  0.8198739
##   6     0.9124015  0.8137178
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
RF_Fit3
```

```
## Random Forest
##
## 7424 samples
##    6 predictor
##    2 classes: 'Acer', 'Sony'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 6681, 6681, 6681, 6683, 6682, 6681, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##   2     0.9159481  0.8217146
##   4     0.9155890  0.8206959
##   6     0.9136579  0.8165360
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 2.
```

```
I1 = varImp(RF_Fit1, scale=TRUE)
I2 = varImp(RF_Fit2, scale=TRUE)
I3 = varImp(RF_Fit3, scale=TRUE)
I1
```

```
## rf variable importance
##
##         Overall
## salary  100.000
## age      47.334
## credit   12.595
## car       4.602
## zipcode   2.194
## elevel    0.000
```

```
I2
```

```
## rf variable importance
##
##         Overall
## salary  100.000
## age      49.926
## credit   13.043
## car       4.719
## zipcode   2.101
## elevel    0.000
```

```r
I3
```

```
## rf variable importance
##
##         Overall
## salary  100.000
## age      46.015
## credit   12.812
## car       4.640
## zipcode   2.123
## elevel    0.000
```

However, during my research I encountered a different way of modeling random forest using `randomForest()` and creating some really nice visualizations.

Different Random Forest Modelling

After some research, I found the "`randomForest()` command which can be used alongside ggplot to generate similar results and display them. By simply calling on the function, i get a confusion matrix of the values:

```r
model <- randomForest(brand ~ ., data = CR, proximity = TRUE)
model
```

```
##
## Call:
##  randomForest(formula = brand ~ ., data = CR, proximity = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 7.78%
## Confusion matrix:
##      Acer Sony class.error
## Acer 3393  351   0.0937500
## Sony  419 5735   0.0680858
```

GGplot and Random Forest

The ggplot is based on **`err.rate matrix`**, a matrix calculated when constructing the model using `randomForest()`. It contains columns for the OOB (out of bag) error rate, Acer error rate, Sony error rate (how frequent those two get missed classified). Each row of the matrix represents the error rate after certain iterations of the random forest, so first row is the error rates after making the first tree, the 50th row shows the error rates after making the 50th tree and so on.

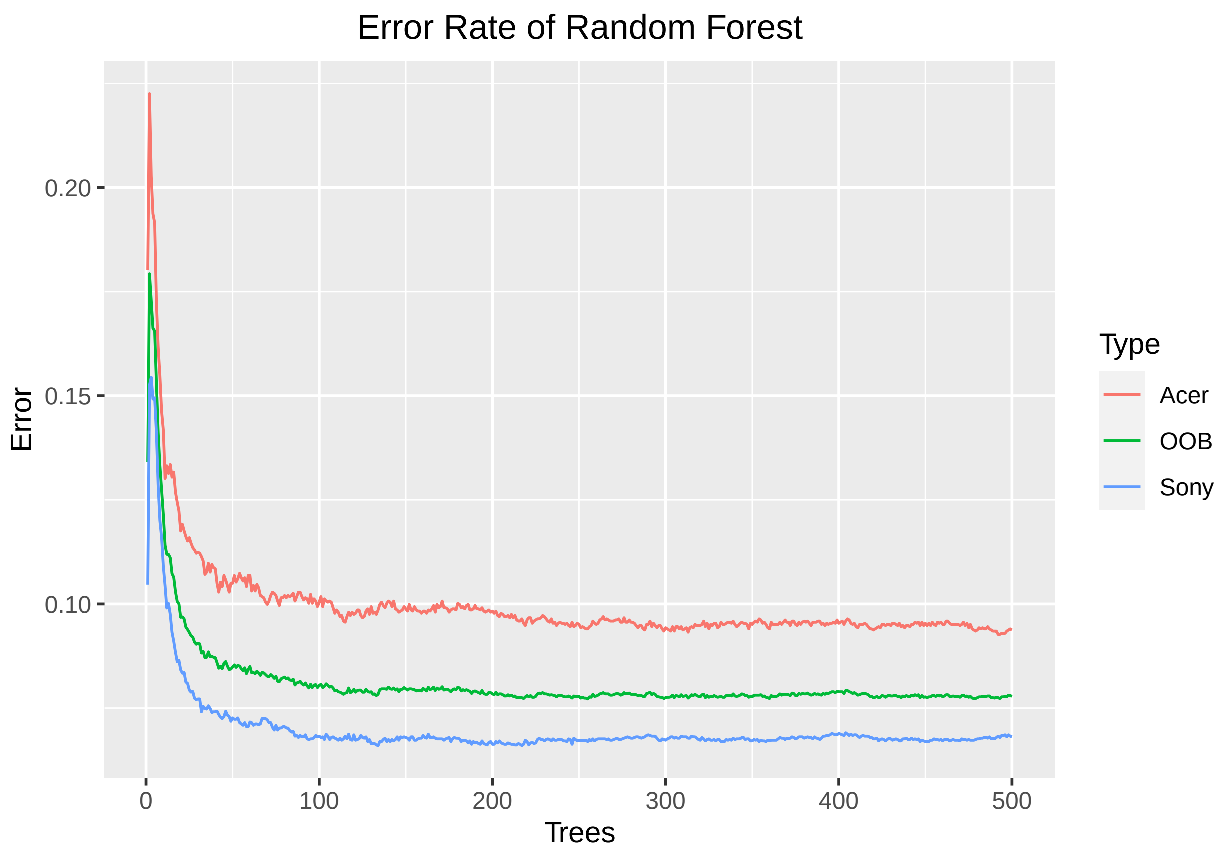
oob.error.data is created to transform the data into something ggplot can understand and plot.

Then using `ggplot` I can graph the matrix and evaluate whether the number of trees I selected are enough to stabilize the error rates.

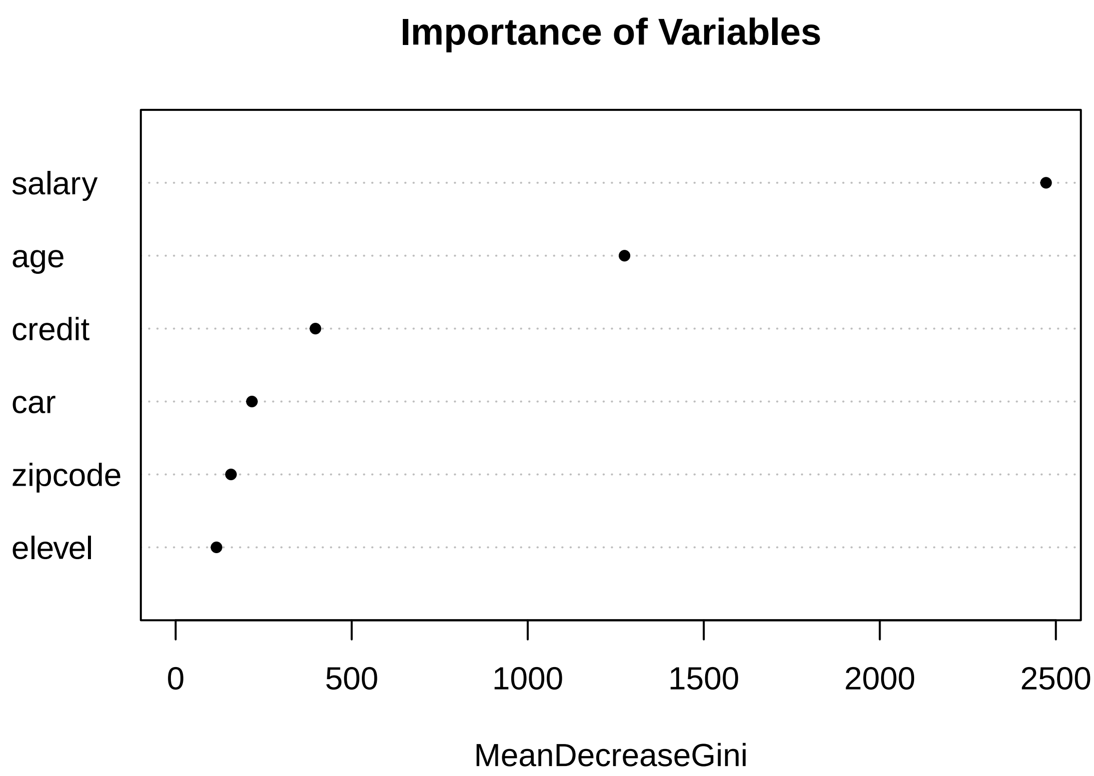
```r
head(model$err.rate)
```

```
##              OOB      Acer      Sony
## [1,] 0.1341234 0.1802486 0.1046358
## [2,] 0.1793115 0.2225623 0.1525561
## [3,] 0.1728412 0.2027601 0.1544486
## [4,] 0.1662063 0.1937107 0.1492131
## [5,] 0.1655436 0.1915269 0.1495005
## [6,] 0.1530491 0.1724724 0.1410882
```

```r
oob.error.data <- data.frame(
  Trees=rep(1:nrow(model$err.rate), times=3),
  Type=rep(c("OOB", "Acer", "Sony"), each=nrow(model$err.rate)),
  Error=c(model$err.rate[,"OOB"],
    model$err.rate[,"Acer"],
    model$err.rate[,"Sony"]))



ggplot(oob.error.data, aes(Trees, Error)) +
  geom_line(aes(color=Type)) +
  labs( title = "Error Rate of Random Forest") +
  theme(plot.title = element_text(hjust = 0.5))
```




```r
varImpPlot(model, pch = 20, main = "Importance of Variables")
```



After about 200 trees, the errors rates seem to stabilize and so using 500 trees to estimate is more than enough.

```
#oob.values <- vector(length=5)
#for (i in 1:5){
#  temp.model <- randomForest(brand ~ ., data=CR, mtry=i, ntree=1000)
#  oob.values[i] <- temp.model$err.rate[nrow(temp.model$err.rate),1]
#}

#oob.values
```

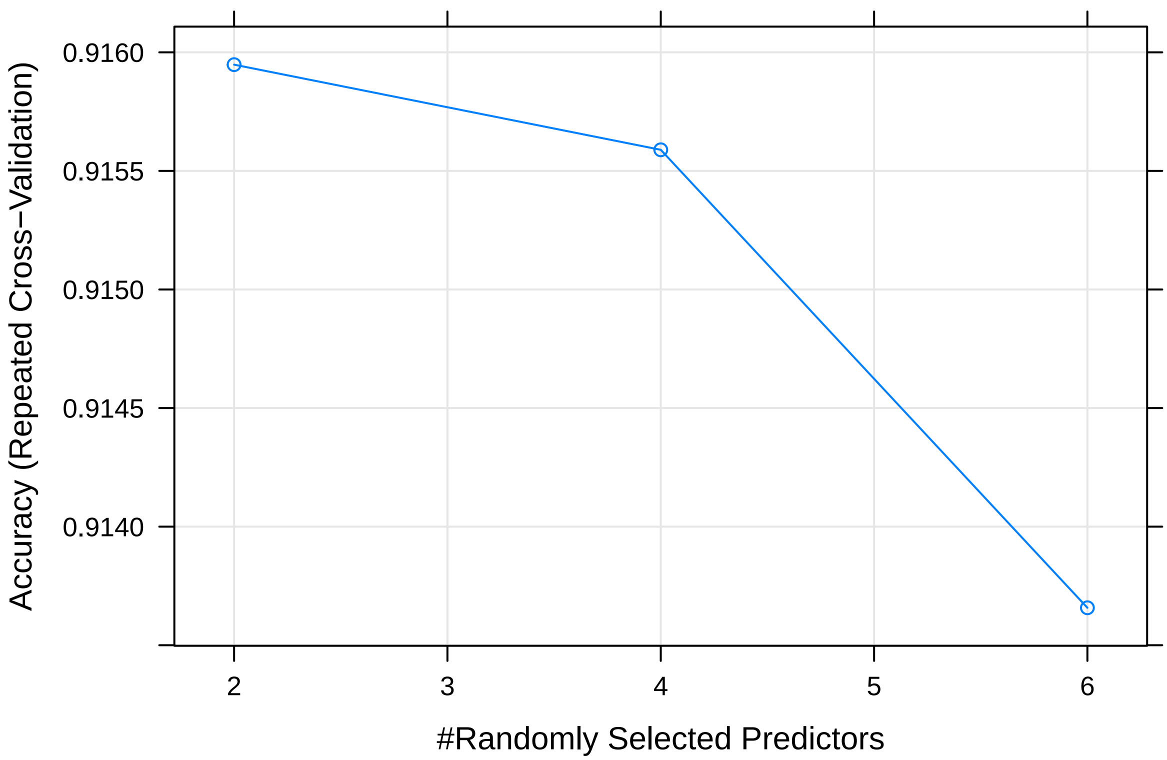
This checks whether the default of 2 variables checked at each split is the most optimal solution for this.

```
#distance.matrix <- dist(1-model$proximity)


#mds.stuff <- cmdscale(distance.matrix, eig=TRUE, x.ret=TRUE)

#mds.var.per <- round(mds.stuff$eig/sum(mds.stuff$eig)*100, 1)

#mds.values <- mds.stuff$points
#mds.data <- data.frame(Sample=rownames(mds.values),
#                       X=mds.values[,1],
#                       Y=mds.values[,2],
#                       Status=CR$brand)
```

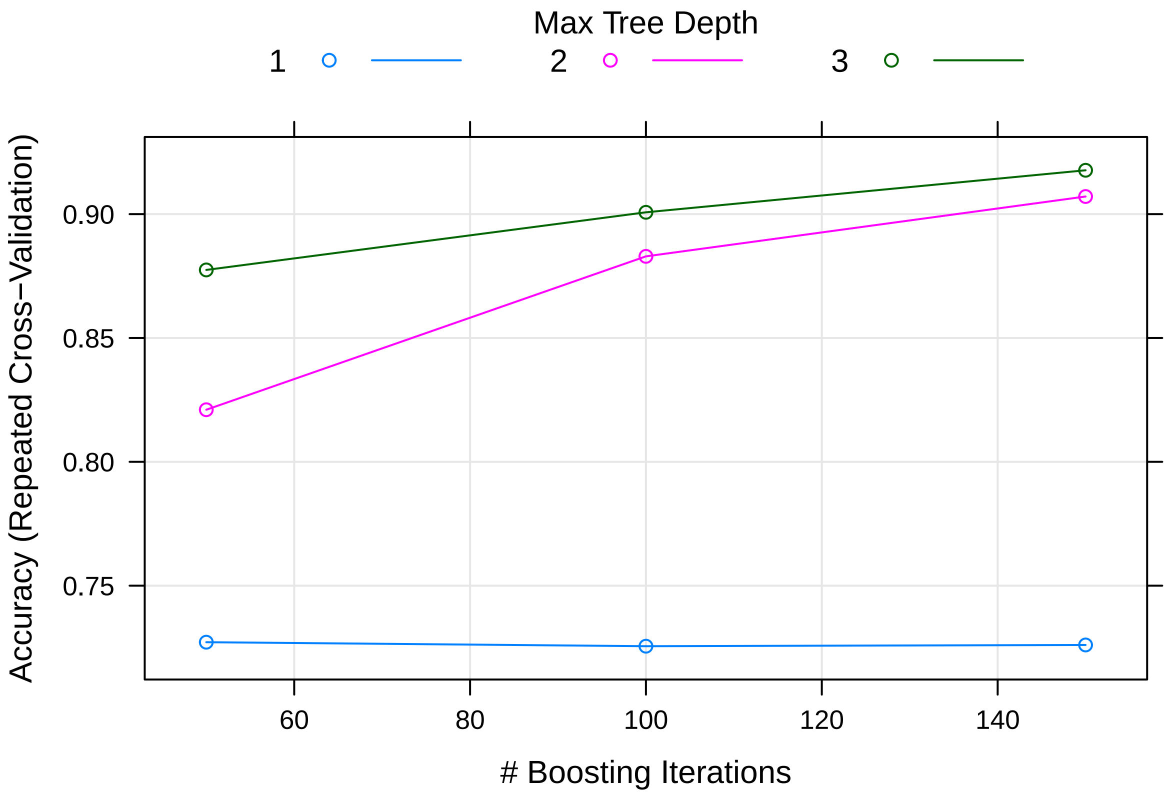
```
#ggplot(mds.data, aes(X,Y, label = Sample)) +
#  geom_text(aes(color=Status))+
#  theme_bw()+
#  xlab(paste("MDS1 - ", mds.var.per[1], "%", sep = ""))+
#  ylab(paste("MDS1 - ", mds.var.per[2], "%", sep = ""))+
#  ggtitle("MDS Plot using (1 - Random Forest Proximities)")
```

Predictions

```
RFpred <- predict(model, newdata = testing)
GBMpred <- predict(GBM_Fit3, newdata = testing)


plot(RF_Fit3)
```




```
plot(GBM_Fit3)
```




```
SI$brand <- as.factor(SI$brand)
levels(SI$brand) <- c("Acer", "Sony")

RFpred <- predict(model, newdata = SI)
GBMpred <- predict(GBM_Fit3, newdata = SI)

postResample(RFpred, SI$brand)
```

```
##   Accuracy      Kappa
## 0.39100000 0.01292757
```

```
postResample(GBMpred, SI$brand)
```

```
##  Accuracy     Kappa
## 0.4030000 0.0124046
```

```
summary(SI$brand)
```

```
## Acer Sony
## 4937   63
```

```
summary(RFpred)
```

```
## Acer Sony
## 1900 3100
```

```
summary(GBMpred)
```

```
## Acer Sony
## 1964 3036
```