Problem Sheet 3 write-up

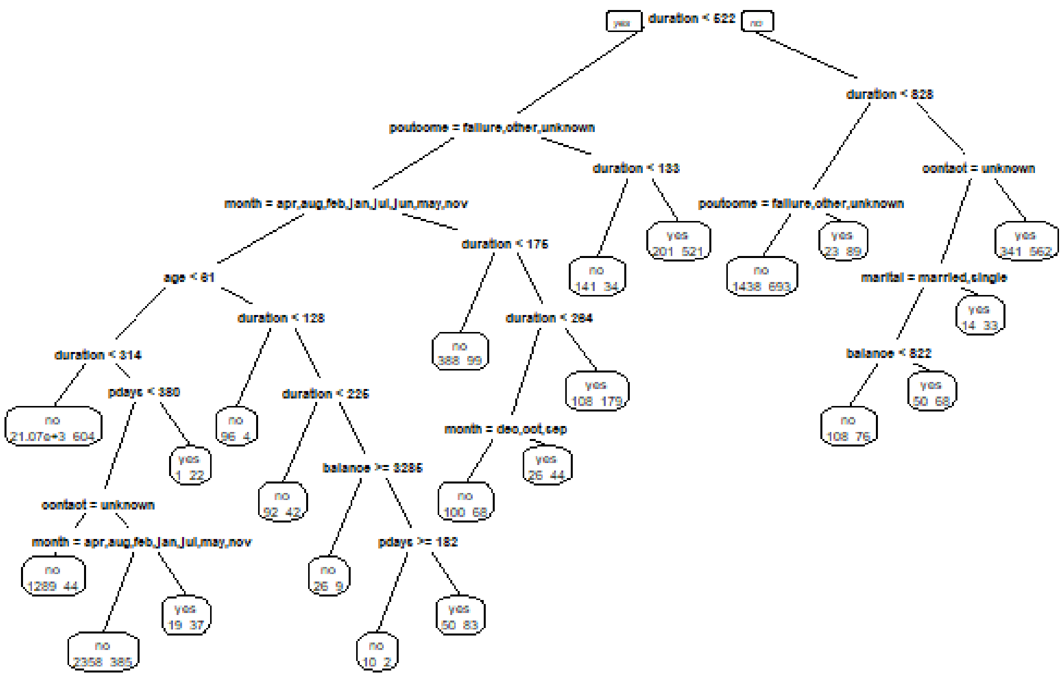
(part a) Data is imported and delimited using R's inbuilt data importer. I also used the column class option to make the categorical data, "categories".

```
#setting the seed
set.seed(42)

#use 70% of dataset as training set and remaining 30% as testing set
data = sort(sample(nrow(bank2), nrow(bank2)*0.7))
train=bank2[data,]
test=bank2[-data,]
#checking for duplicate data
bank2=distinct(bank2)
```

(part b) See the corresponding code below.

```
#build the initial tree
tree = rpart(as.factor(y) ~ ., data=train, control=rpart.control(cp=.002))

#checking the details of our tree
printcp(tree)
#finding the best cp such that error is minimsied
best = tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]

#produce a pruned tree based on the best cp value
pruned_tree = prune(tree, cp=best)

#plot the pruned tree
prp(pruned_tree,
    faclen=0, #use full names for factor labels
    extra=1, #display number of obs. for each terminal node
    roundint=T, #don't round to integers in output
    digits=5) #display 5 decimal places in output
```

This generates the following tree.



From this tree, we can observe that balance, contact, duration, marital, month, pdays and poutcome were the only variables used in the tree construction. The most important variable seems to be duration with poutcome and month also appearing to be important by how many times they have been used in our tree.

(part c) Using the training data, we can work out a misclassification error.

```
#predicting using our classification tree using training data
predict = predict(tree,type="class")
#building a confusion matrix
confusion_matrix = table(train$y, predict)
confusion_matrix
#working out the resulting error
error.train = sum(diag(confusion_matrix))/sum(confusion_matrix)
1-error.train
```

This returns the following output.

```
> #predicting using our classification tree using training data
> predict = predict(tree,type="class")
> #building a confusion matrix
> confusion_matrix = table(train$y, predict)
> confusion_matrix
     predict
         no   yes
  no  27116   833
  yes  2060  1638
> #working out the resulting error
> error.train = sum(diag(confusion_matrix))/sum(confusion_matrix)
> 1-error.train
[1] 0.09141467
> 
```

We can observe an error rate of 9.1414% (4dp) when we predict using the training data.

```
#predicting using our classification tree using test data
predict.test = predict(tree,test,type="class")
#building a confusion matrix for our test data
confusion_matrix.test = table(predict.test,test$y)
confusion_matrix.test
#working out the test error
error.test = sum(diag(confusion_matrix.test))/sum(confusion_matrix.test)
1-error.test
```

We then predict a misclassification error using the test data. This returns the following.

```
> #predicting using our classification tree using test data
> predict.test = predict(tree,test,type="class")
> #building a confusion matrix for our test data
> confusion_matrix.test = table(predict.test,test$y)
> confusion_matrix.test

predict.test    no   yes
         no  11571   920
         yes   402   671
> #working out the test error
> error.test = sum(diag(confusion_matrix.test))/sum(confusion_matrix.test)
> 1-error.test
[1] 0.09746387
> 
```

Using our test data, we can observe a misclassification error of 9.7463% (4dp). This is slightly higher than our training misclassification error.

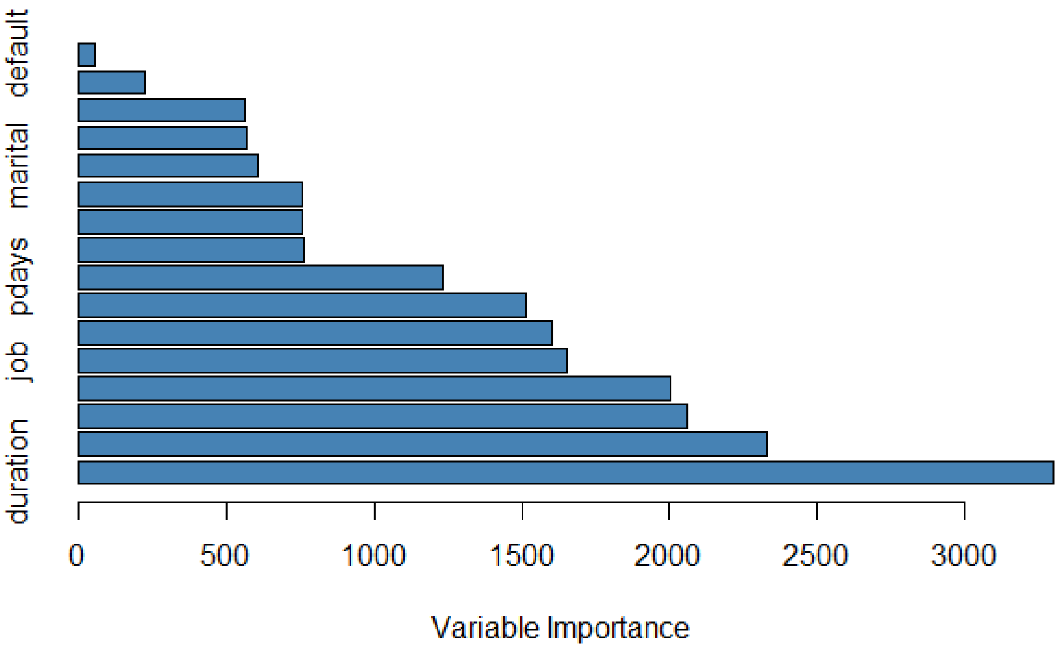
(part d) Observe the following code.

```
#fitting our bagging model
bag = bagging(
  formula = as.factor(y)~.,
  data = train,
  nbagg = 20,
  coob = TRUE,
  control = rpart.control(minsplit = 2, cp = 0))
bag

#predicting using our bagging approach
predict.bag = predict(bag,test,type="class")
confusion_matrix.bagtest = table(predict.bag,test$y)
confusion_matrix.bagtest
error.bagtest = sum(diag(confusion_matrix.bagtest))/sum(confusion_matri
1-error.bagtest
#checking the importance of each variable
#calculate variable importance
VI <- data.frame(var=names(bank2[,-1]), imp=varImp(bag))

#sort variable importance descending
VI_plot <- VI[order(VI$Overall, decreasing=TRUE),]

#visualize variable importance with horizontal bar plot
barplot(VI_plot$Overall,
        names.arg=rownames(VI_plot),
        horiz=TRUE,
        col='steelblue',
        xlab='Variable Importance')
```

We get the following confusion matrix.

```
> #predicting using our bagging approach
> predict.bag = predict(bag,test,type="class")
> confusion_matrix.bagtest = table(predict.bag,test$y)
> confusion_matrix.bagtest

predict.bag    no   yes
        no  11471   800
        yes   502   791
> error.bagtest = sum(diag(confusion_matrix.bagtest))/sum(confusion_matrix.bagtest)
> 1-error.bagtest
[1] 0.09598938
> #checking the importance of each variable
> #calculate variable importance
> VI <- data.frame(var=names(bank2[,-1]), imp=varImp(bag))
>
```

We can also observe that our bagging approach has a test error of 9.5989% (4dp). We can also observe the following plot which shows variable importance.



We can observe from this that duration is by far the most important variable and default is one of the least important variables. Job, pdays and marital are some examples of the middle-ranked variables in terms of importance.

(part e) Observe the code for building random forests.

```
#fitting random forest to the train dataset
model <- randomForest(formula = as.factor(y)~.,data =train,ntree = 500)

predict.rf = predict(model, test,type = "class")
cm.rf = table(predict.rf, test$y)
error.rf = sum(diag(cm.rf))/sum(cm.rf)
1-error.rf
```

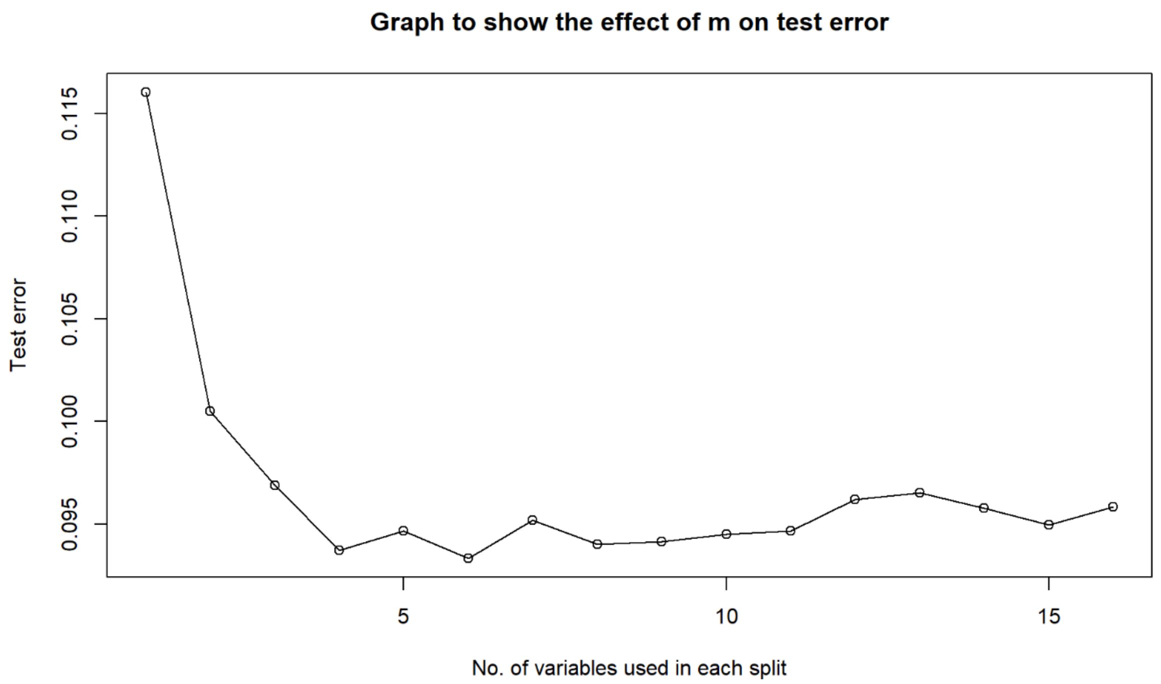
This returns the following test error rate of 9.3188% (4dp)

```
> predict.rf = predict(model, test,type = "class")
> cm.rf = table(predict.rf, test$y)
> error.rf = sum(diag(cm.rf))/sum(cm.rf)
> 1-error.rf
[1] 0.09318785
```

We can next consider the effect of the number of variables considered at each split against the test error by the following code.

```
#plot the test error by number of variables
mtry = 1:16
df = data.frame()
# Loop over each value of mtry and store result in a data frame
for (i in mtry) {
  rf1=randomForest(as.factor(y)~.,data =train,ntree = 100,mtry=i)
  predict.i = predict(rf1,test)
  error.rf = mean(predict.i!=test$y)
  result = data.frame(mtry=i,error=error.rf)
  df = rbind(df,result)
}
df

plot(df$mtry,df$error,type="o",
     xlab="No. of variables used in each split", ylab = "Test error"
     ,main="Graph to show the effect of m on test error")
```

This outputs the following.



This shows that the optimal number of variables used in a spit is m=6. There is a sharp decrease in test error from m=1 to m=4 and we can observe there is small change in error between m=4 and m=6. Then, from m=6 onwards, we can see that the test error slowly increases.

(part f) Overall, we can observe that the random forests test error is the lowest making it the better classification method for this data. We can then observe that bagging gives the next best test error and that classification trees are the worst at classifying. It should therefore be argued that random forests is the best classification method for this data.