Data Mining Project

Prateek Kumar

April 24, 2019

Contents

```
set.seed(42)
```

1. Classification of Age based on Social Media Usage

For this problem, we classify the age of an individual (“High School" or “Adult") based on their social media app usage.

Reading the csv file

```
Data <- read.csv(file="syp-16-data.csv", header=TRUE)
```

The data was collected via a survey, with respondents consisting of the Women in Computing Sciences Summer Youth Program participants and female faculty at Michigan Tech. There are 60 responses with 26 Adults and 34 HS respondents.

(a) Create a small multiples plot showing the number of respondents who use or do not use each app grouped by age. Consider making grouped bar plots.

Firstly, we must aggregate the data

```
x=aggregate(.~Adult, Data, sum)
x=t(x)
x=x[-1,]


#We here aggregate and arrange our data for plotting grouped bar plot.
c1 = rep(names(Data[,1:19]),2)
c2 = c("Adult","HS")
c2 = sort(rep(c2,19))
c3=c( as.numeric(x[,1]),as.numeric(x[,2]))

tab = cbind(c1,c2,c3)
tab = as.data.frame(tab)
```

Now the grouped bar plot showing the number of respondents who use or do not use each app grouped by age

```
library(ggplot2)
p <- ggplot(as.data.frame(tab) , aes(fill=c2, y=c3, x=c1)) + geom_bar(position="dodge",
stat="identity")
p + labs(title = "Respondents v/s App Usage", fill = "Respondents") +
  xlab("Different Social Medias") + ylab("Number of respondents who use the app")
```

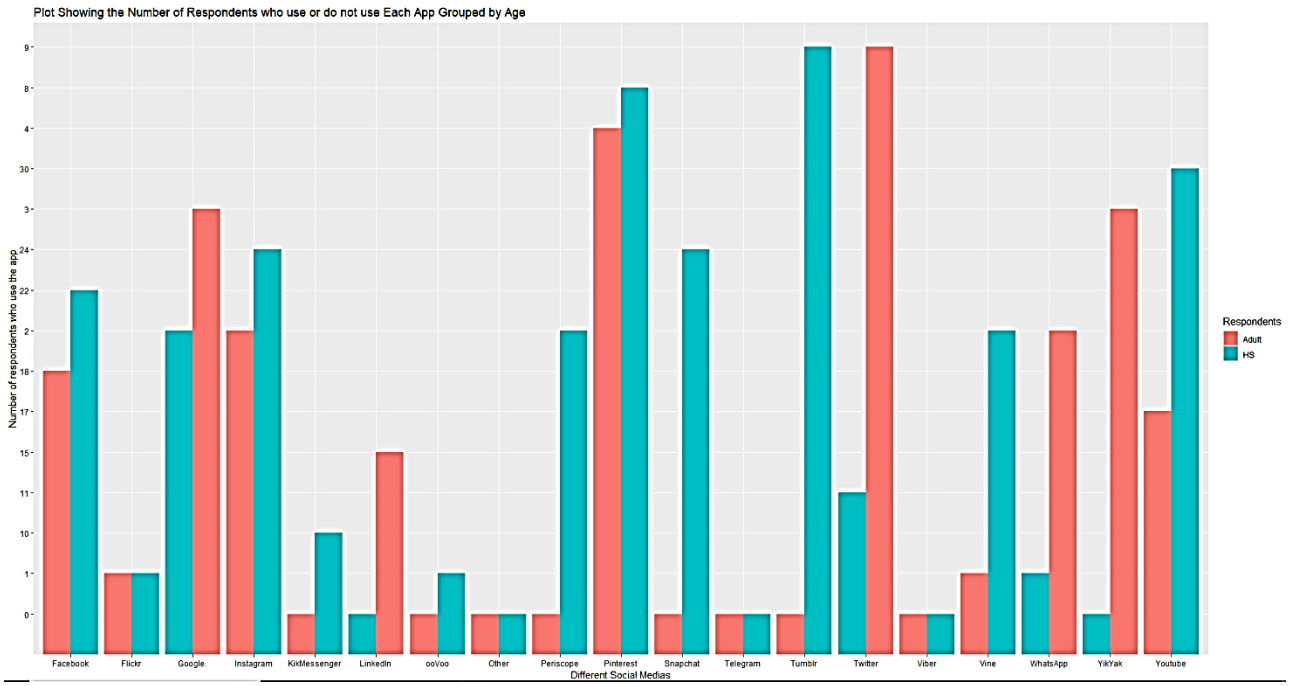

Figure 1: Grouped barplot

(b) From the figures in (a), which of the apps would you expect to find at the root of a deicision tree?

When we look at the plot we see that KikMessenger, LinkedIn, ooVoo, Periscope, Snapchat, Tumblr and YikYak have the highest purity, i.e. these apps have just one kind of respondents so any one amongst them will be the root of the decision tree.

(c) Construct a decision tree using this data set. Does the variable at the root of the tree match the intuition from part (b)?

```
library(rpart) #Loading the rpart library
dt.model <- rpart(Adult ~ ., data=Data) #Creating the decision tree
dt.model
```

```
## n= 60
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 60 26 HS (0.4333333 0.5666667)
##   2) Snapchat< 0.5 36 10 Adult (0.7222222 0.2777778)
##     4) Instagram< 0.5 28  4 Adult (0.8571429 0.1428571) *
##     5) Instagram>=0.5 8  2 HS (0.2500000 0.7500000) *
##   3) Snapchat>=0.5 24  0 HS (0.0000000 1.0000000) *
```

```
#post(dt.model, filename='')
```

```
rpart.plot(dt.model, box.palette="RdBu", shadow.col="gray", nn=TRUE)
```

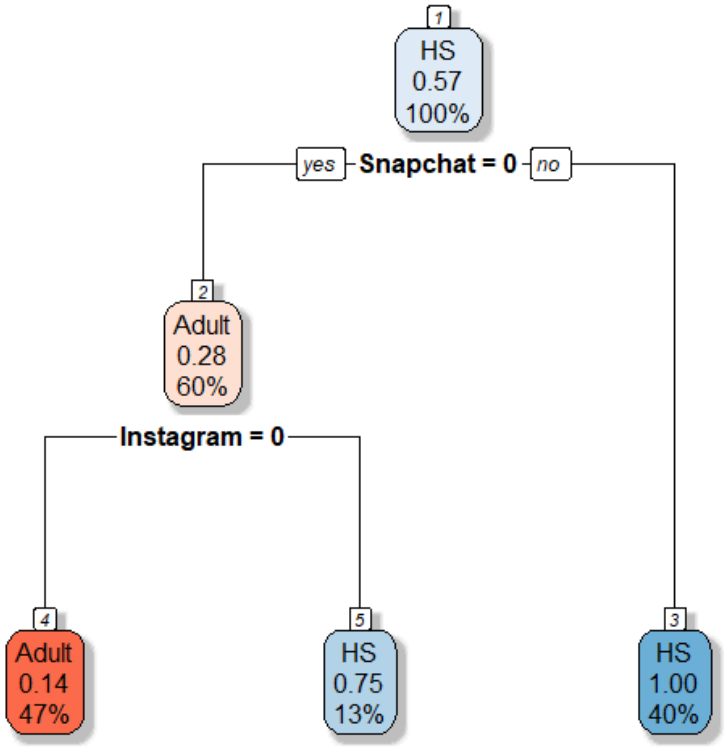

Figure 2: Decision Tree

The root of the decision tree comes out to be snapchat which was one of our expected apps. So, the variable at the root of the tree somewhat matched our intuition.

2. Classification of Spam: Trees

(a) Load in the spam data. You should not include the following columns in the classification task: isuid, id, domain, spampct, category, and cappct.

```
spam <- read.csv(file="spam.csv", header=TRUE)
spam <- subset( spam, select = -c(isuid, id, domain, spampct, category, cappct ) )
#spam<-spam[complete.cases(spam), ]
```

(b) Split the data into a training and test set with an 80/20 split of the data.

```
dim(spam)
```

```
## [1] 2171   15
```

```
indexes = sample(1:nrow(spam), size=0.2*nrow(spam))
# Split data
test = spam[indexes,]
dim(test)

## [1] 434  15

train = spam[-indexes,]
dim(train)

## [1] 1737   15
```

(c) Construct a classification tree to predict spam on the training data.

```
library(rpart)
dt.model <- rpart(spam ~ ., data=train)
dt.model

## n= 1737
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1737 563 no (0.67587795 0.32412205)
##     2) box=yes 894  21 no (0.97651007 0.02348993) *
##     3) box=no 843 301 yes (0.35705813 0.64294187)
##       6) local=yes 159   8 no (0.94968553 0.05031447) *
##       7) local=no 684 150 yes (0.21929825 0.78070175)
##        14) large.text=no 401 143 yes (0.35660848 0.64339152)
##          28) name=name 248 117 yes (0.47177419 0.52822581)
##            56) time.of.day>=5.5 191  87 no (0.54450262 0.45549738)
##             112) digits< 0.5 155  60 no (0.61290323 0.38709677)
##               224) day.of.week=Mon,Thu,Wed 95  26 no (0.72631579 0.27368421) *
##               225) day.of.week=Fri,Sat,Sun,Tue 60  26 yes (0.43333333 0.56666667) *
##             113) digits>=0.5 36   9 yes (0.25000000 0.75000000) *
##            57) time.of.day< 5.5 57  13 yes (0.22807018 0.77192982) *
##          29) name=empty,single 153  26 yes (0.16993464 0.83006536) *
##        15) large.text=yes 283   7 yes (0.02473498 0.97526502) *
```

(d) Describe the tree that is constructed (print or plot the tree). How many terminal leaves does the tree have? What is the total number of nodes in the tree?

```
#post(dt.model, filename='')
rpart.plot(dt.model, box.palette="RdBu", shadow.col="gray", nn=TRUE)
```

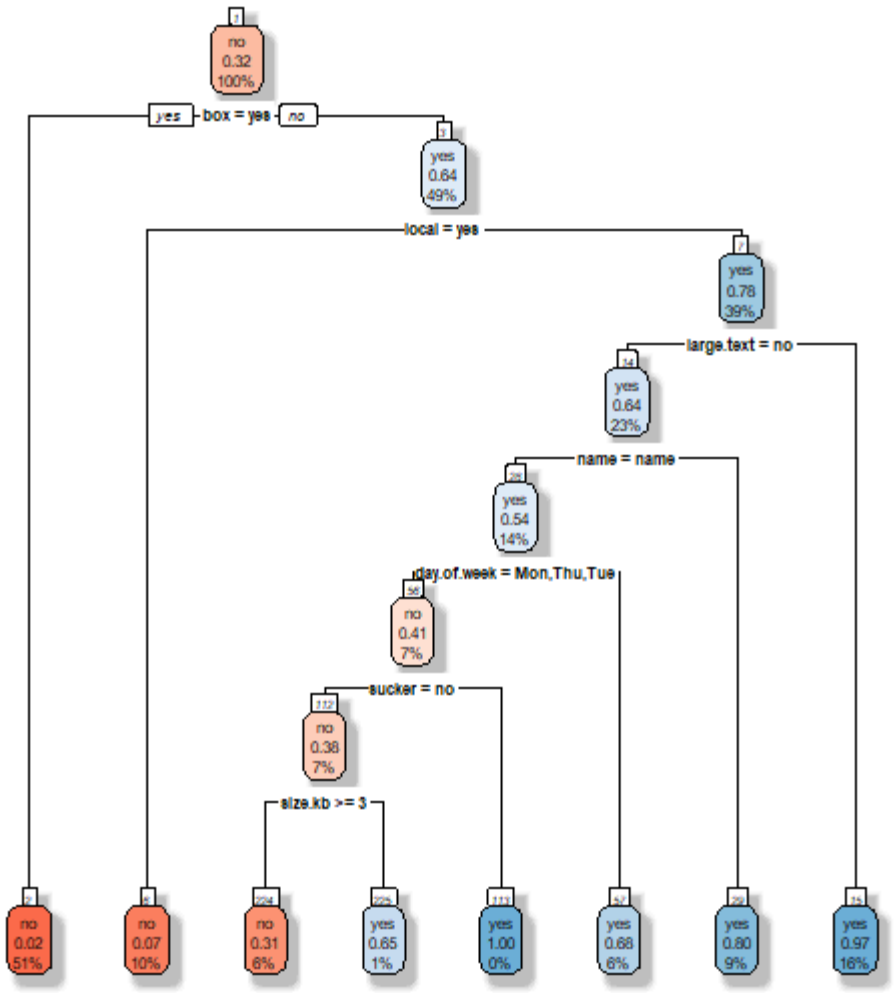

Figure 3: Decision Tree Spam

In the above tree we have 7 decision nodes: box, local, large.text, name, day.of.week, sucker, size.kb and 8 leaves.

(e) Estimate the performance of the decision tree on the training set and the testing set. Report accuracy, error rate, and AUC(Area Under Curve) using a threshold of 0.5.

```
pred <- predict(dt.model, test) #predicting over test data


#changing the data so that we can calculate the accuracy
vec <- c()
for (i in 1:nrow(pred))
  {
  if(pred[i,1]>pred[i,2]){
    vec <- c(vec,'no')
  }else{
    vec <- c(vec,'yes')
  }
}
```

```
vec1 <- factor(as.numeric(test$spam))
vec_new <- c()
for (j in 1:length(vec)){
  if(vec[j]=='yes'){
    vec_new <- c(vec_new,2)
  }else{
    vec_new <- c(vec_new,1)
  }
}
vec_new <- as.factor(vec_new)


#calculating accuracy, error and AUC
confMat <- table(vec1,vec_new)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.91"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.09"

library(pROC)

vec1 <- as.numeric(vec1)
vec_new <- as.numeric(vec_new)
roc_obj <- roc(vec1,vec_new)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.9"
```

(f) Try pruning the tree, explore 2 other sized tree and report the classification performance in either case.

First Tree with 10% post-pruning

```
tree1 <- prune(dt.model, cp = 0.1)
plot(tree1)
text(tree1)
```

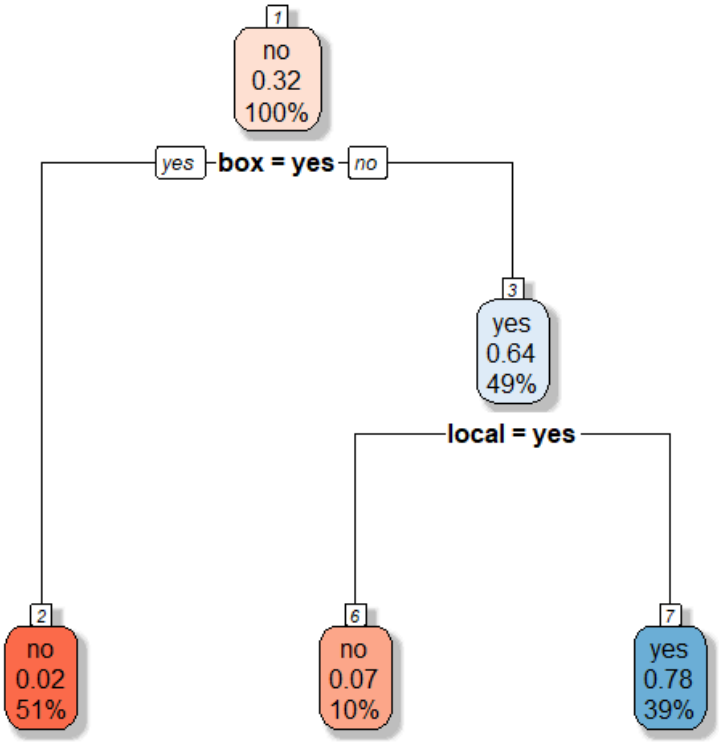

Figure 4: Tree after 10% post-pruning

```
pred <- predict(tree1, test)


#changing the data so that we can calculate the accuracy
vec <- c()
for (i in 1:nrow(pred))
  {
  if(pred[i,1]>pred[i,2]){
    vec <- c(vec,'no')
  }else{
    vec <- c(vec,'yes')
  }
}

vec1 <- factor(as.numeric(test$spam))
vec_new <- c()
for (j in 1:length(vec)){
  if(vec[j]=='yes'){
    vec_new <- c(vec_new,2)
  }else{
    vec_new <- c(vec_new,1)
  }
}
vec_new <- as.factor(vec_new)


#calculating accuracy, error and AUC
confMat <- table(vec1,vec_new)
```

```
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.9"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.1"

library(pROC)
vec1 <- as.numeric(vec1)
vec_new <- as.numeric(vec_new)
roc_obj <- roc(vec1,vec_new)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.91"
```

Second Tree with 30% post-pruning

```
tree2 <- prune(dt.model, cp = 0.3)
plot(tree2)
text(tree2)
```

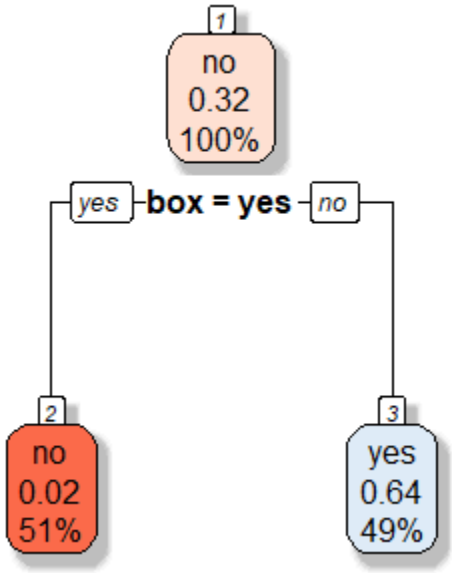

Figure 5: Tree after 30% post-pruning

```
pred <- predict(tree2, test)


#changing the data so that we can calculate the accuracy
vec <- c()
for (i in 1:nrow(pred))
  {
  if(pred[i,1]>pred[i,2]){
    vec <- c(vec,'no')
```

```
  }else{
    vec <- c(vec,'yes')
  }
}

vec1 <- factor(as.numeric(test$spam))
vec_new <- c()
for (j in 1:length(vec)){
  if(vec[j]=='yes'){
    vec_new <- c(vec_new,2)
  }else{
    vec_new <- c(vec_new,1)
  }
}
vec_new <- as.factor(vec_new)


#calculating accuracy, error and AUC
confMat <- table(vec1,vec_new)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.8"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.2"

library(pROC)
vec1 <- as.numeric(vec1)
vec_new <- as.numeric(vec_new)
roc_obj <- roc(vec1,vec_new)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.84"
```

3. Classification of Music Popularity

(a) Load in the music data. You should not use the artist or song title and IDs in the prediction along with the confidence variables.

```
music <- read.csv(file="music.csv", header=TRUE)
music <- music[,-c(1,2,3,4,5,7,10,12),drop=FALSE]

dim(music)

## [1] 7574   31
```

```
indexes = sample(1:nrow(music), size=0.2*nrow(music))
# Split data
test = music[indexes,]
dim(test)

## [1] 1514   31

train = music[-indexes,]
dim(train)

## [1] 6060   31
```

(b) Prepare the data for a 10-fold cross-validation. Ensure that each split of the data has a balanced distribution of class labels.

```
library("caret")

kfold <- trainControl(method  = "cv", number  = 10)
```

(c) Use kNN to predict whether a song is a hit. Estimate the generalization performance over the 10-folds, calculate and report the accuracy, error, and AUC performance on the testing data. Show these results for three values of k = 1, 3, 5, 7, 9.

For k=1

```
library("e1071")

# Applying knn

fit <- train(as.factor(Top10) ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = c(1)),
             trControl  = kfold,
             metric     = "Accuracy",
             data       = train)


#calculating accuracy, error and AUC
pred <- predict(fit, test)
confMat <- table(pred,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.78"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.22"
```

```
library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.57"
```

For k=3

```
library("e1071")


# Applying knn
fit <- train(as.factor(Top10) ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = c(3)),
             trControl  = kfold,
             metric     = "Accuracy",
             data       = train)


#calculating accuracy, error and AUC
pred <- predict(fit, test)
confMat <- table(pred,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.81"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.19"
```

```
library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.58"
```

For k=5

```
library("e1071")
```

```
# Applying knn
fit <- train(as.factor(Top10) ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = c(5)),
             trControl  = kfold,
             metric     = "Accuracy",
             data       = train)


#calculating accuracy, error and AUC
pred <- predict(fit, test)
confMat <- table(pred,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.84"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.16"
```

```
library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.65"
```

For k=7

```
library("e1071")


# Applying knn
fit <- train(as.factor(Top10) ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = c(7)),
             trControl  = kfold,
             metric     = "Accuracy",
             data       = train)


#calculating accuracy, error and AUC
pred <- predict(fit, test)
confMat <- table(pred,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.83"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.17"
```

```
library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.61"
```

For k=9

```
library("e1071")


# Applying knn
fit <- train(as.factor(Top10) ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = c(9)),
             trControl  = kfold,
             metric     = "Accuracy",
             data       = train)


#calculating accuracy, error and AUC
pred <- predict(fit, test)
confMat <- table(pred,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.84"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.16"
```

```
library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.64"
```

(d) Use decision trees to predict whether a song is a hit. Estimate the generalization performance over the 10-folds, calculate and report the accuracy, error, and AUC performance on the testing data. Show the results for two different sized decision trees (consider different amounts of pruning).

```
library("e1071")


#Decision Tree with 10 fold cross validation
fit <- train(as.factor(Top10) ~ .,
             method     = "rpart",
             trControl  = kfold,
             metric     = "Accuracy",
             data       = train)


#Plotting the Decision Tree
rpart.plot(fit$finalModel, box.palette="RdBu", shadow.col="gray", nn=TRUE)
```

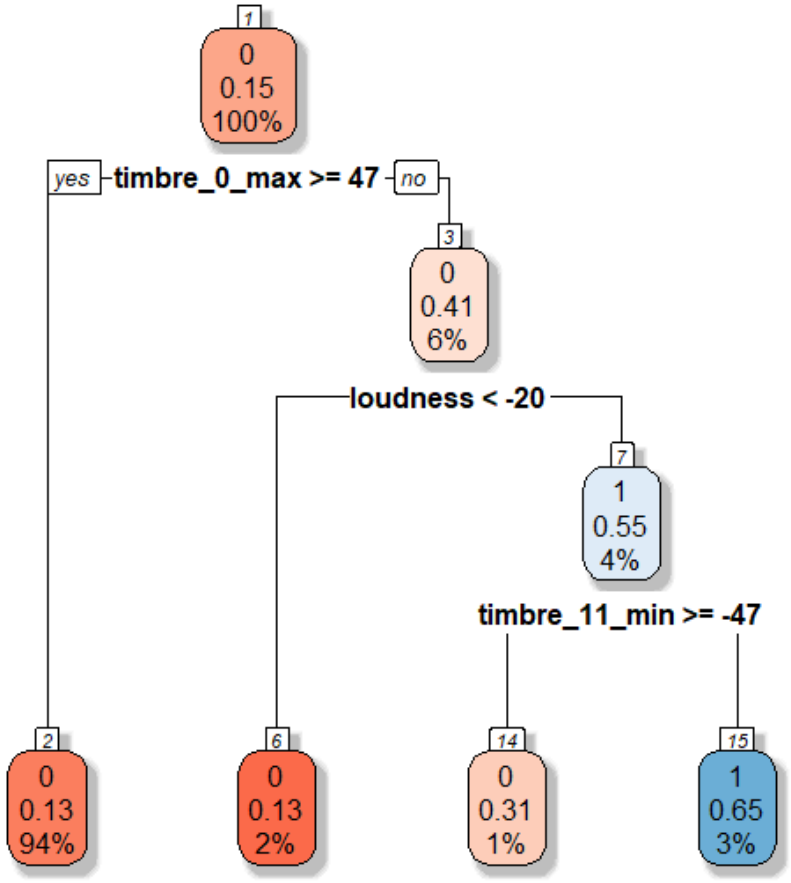

Figure 6: Music Decision Tree with 10 fold cross validation.

```
pred <- predict(fit, test)
confMat <- table(pred,test$Top10)
```

```
#calculating accuracy, error and AUC

accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.85"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.15"

library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.72"
```

2 other trees. First Tree with 1% post-pruning

```
dt <- fit$finalModel
tree1 <- prune(dt, cp = 0.01)

pred <- predict(tree1, test)
vec <- c()
for (i in 1:nrow(pred))
  {
  if(pred[i,1]>pred[i,2]){
    vec <- c(vec,0)
  }else{
    vec <- c(vec,1)
  }
}


#calculating accuracy, error and AUC
confMat <- table(vec,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.85"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.15"

library(pROC)
vec_n1 <- as.numeric(as.character(unlist(vec)))
vec_n2 <- as.numeric(as.character(unlist(test$Top10)))
```

```
roc_obj <- roc(vec_n1,vec_n2)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.75"
```

Second Tree with 2% post-pruning

```
tree1 <- prune(dt, cp = 0.02)

pred <- predict(tree1, test)
vec <- c()
for (i in 1:nrow(pred))
  {
  if(pred[i,1]>pred[i,2]){
    vec <- c(vec,0)
  }else{
    vec <- c(vec,1)
  }
}


#calculating accuracy, error and AUC
confMat <- table(vec,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.85"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.15"
```

```
library(pROC)
vec_n1 <- as.numeric(as.character(unlist(vec)))
vec_n2 <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(vec_n1,vec_n2)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.75"
```

(e) Use a Naive Bayes classifier to predict whether a song is a hit. Calculate and report the accuracy, error, and AUC performance on the testing data.

```
library(e1071)

#using Naïve Bayes Classifier
nb_model <- naiveBayes(as.factor(Top10) ~., data = train)
pred <- predict(nb_model, test)
```

```
confMat <- table(pred,test$Top10)


#calculating accuracy, error and AUC

accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.74"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.26"

library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.61"
```

(f) Discuss whether the selection of the negative samples included in the data set may infuence the results.

Increasing the size of the negative training set (with a constant quantity of positives), will lead to decrease in recall and improve the precision, although no significant effect on AUC values will be observed.

4. Classification of Music Popularity

(a) Load in the music data. You should not use the artist or song title and IDs in the prediction along with the confidence variables.

```
music <- read.csv(file="music.csv", header=TRUE)

#removing the columns not used for classification
music <- music[,-c(1,2,3,4,5,7,10,12),drop=FALSE]

dim(music)

## [1] 7574   31

indexes = sample(1:nrow(music), size=0.2*nrow(music))
# Split data
test = music[indexes,]
dim(test)
```

```
## [1] 1514   31

train = music[-indexes,]
dim(train)

## [1] 6060   31
```

(b) Prepare the data for a 10-fold cross-validation. Ensure that each split of the data has a balanced distribution of class labels.

```
library("caret")
kfold <- trainControl(method  = "cv", number  = 10)
```

(f) Use Random Forests to predict whether a song is a hit. Calculate and report the accuracy, error, and AUC performance on the testing data (10-fold c.v.).

```
library("e1071")
library("ranger")

#using Random Forest with 10 fold cross validation

fit <- train(as.factor(Top10) ~ .,
             method     = "ranger",
             trControl  = kfold,
             metric     = "Accuracy",
             data       = train)


#calculating accuracy, error and AUC
pred <- predict(fit, test)
confMat <- table(pred,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.85"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.15"

library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.83"
```

(g) Learn a support vector machine (SVM) with a RBF kernel to predict whether the song is a hit. Consider at least the following values for cost: 0.01, 0.1, 1, 10, 100. Calculate and report the accuracy, error, and AUC performance on the testing data for the best model found.

```
library("e1071")

#using SVM with 10 fold cross validation
tune_out <- tune.svm(x=train[,-31],y=train[,31],cost=c(0.01, 0.1, 1, 10,
100),kernel="radial",data = train)


#print best values of cost
tune_out$best.parameters$cost

## [1] 1

#build model
svm_model <- svm(Top10~ ., data=train, method="C-classification",
kernel="radial",cost=tune_out$best.parameters$cost)


#test set predictions
pred_svm <-predict(svm_model,test)

pred_svm1 <- c()
for (j in 1:length(pred_svm)){
  if(abs(pred_svm[j]-0) < abs(pred_svm[j]-1)){
    pred_svm1 <- c(pred_svm1,0)
  }else{
    pred_svm1 <- c(pred_svm1,1)
  }
}


#calculating accuracy, error and AUC
confMat <- table(pred_svm1,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.85"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.15"

library(pROC)
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- multiclass.roc(pred_svm1, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.83"
```

(i) Re-run the analysis in (c) using a nested cross-validation to determine the best value of k and to determine the best parameters of the SVM.

```
library("e1071")


#rerunning knn with nested k values
fit <- train(as.factor(Top10) ~ .,
             method     = "knn",
             tuneGrid   = expand.grid(k = c(1,2,3,4,5,6,7,8,9,10)),
             trControl  = kfold,
             metric     = "Accuracy",
             data       = train)


#calculating accuracy, error and AUC
pred <- predict(fit, test)
confMat <- table(pred,test$Top10)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.83"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.17"

library(pROC)
pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$Top10)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.63"

#best parameter for k and for SVM

print(paste0('Best k value: ',fit$bestTune))

## [1] "Best k value: 9"

print(paste0('Best c value for SVM: ',tune_out$best.parameters$cost))

## [1] "Best c value for SVM: 1"
```

5. Classification of Spam

(a) Load in the spam data. You should not include the following columns in the classification task: isuid, id, domain, spampct, category, and cappct.

```
spam <- read.csv(file="spam.csv", header=TRUE)
spam <- subset( spam, select = -c(isuid, id, domain, spampct, category, cappct ) )
spam<-spam[complete.cases(spam), ]
```

(b) Split the data into a training and test set with an 80/20 split of the data.

```
dim(spam)
```

```
## [1] 2171   15
```

```
indexes = sample(1:nrow(spam), size=0.2*nrow(spam))
# Split data
test = spam[indexes,]
dim(test)
```

```
## [1] 434  15
```

```
train = spam[-indexes,]
dim(train)
```

```
## [1] 1737   15
```

(c) Use a Naive Bayes classifier to predict spam. Calculate and report the accuracy, error, and AUC performance on the testing data.

```
library(e1071)

#using Naïve Bayes Classifier
nb_model <- naiveBayes(as.factor(spam) ~., data = train)
pred <- predict(nb_model, test)

levels(pred) <- c("2", "1")
levels(test$spam) <- c("2", "1")


#calculating accuracy, error and AUC
confMat <- table(pred,test$spam)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.9"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.1"
```

```
library(pROC)

pred <- as.numeric(as.character(unlist(pred)))
tval <- as.numeric(as.character(unlist(test$spam)))

roc_obj <- roc(pred,tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.88"
```

(d) Learn a decision tree to predict spam. Calculate and report the accuracy, error, and AUC performance on the testing data.

```
library(rpart)

#learning the decision tree
dt.model <- rpart(spam ~ ., data=train)

pred1 <- predict(dt.model, test)


#calculating the decision tree predictions
vec <- c()
for (i in 1:nrow(pred1))
{
  if(pred1[i,1]>pred1[i,2]){
    vec <- c(vec,'no')
  }else{
    vec <- c(vec,'yes')
  }
}

vec1 <- factor(as.numeric(test$spam))
vec_new <- c()
for (j in 1:length(vec)){
  if(vec[j]=='yes'){
    vec_new <- c(vec_new,2)
  }else{
    vec_new <- c(vec_new,1)
  }
}
vec_new <- as.factor(vec_new)


#calculating accuracy, error and AUC
confMat <- table(vec1,vec_new)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.91"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.09"
```

```
library(pROC)
vec1 <- as.numeric(vec1)
vec_new <- as.numeric(vec_new)
roc_obj <- roc(vec1,vec_new)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.9"
```

(e) Use Random Forests to predict spam. Calculate and report the accuracy, error, and AUC performance on the testing data.

```
library("e1071")
library("ranger")
library("caret")


#using Random Forest
fit <- train(as.factor(spam) ~ .,
             method     = "ranger",
             metric     = "Accuracy",
             data       = train)

pred2 <- predict(fit, test)

levels(pred2) <- c("2", "1")
levels(test$spam) <- c("2", "1")


#calculating accuracy, error and AUC
confMat <- table(pred2,test$spam)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.92"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.08"
```

```
library(pROC)
pred2 <- as.numeric(as.character(unlist(pred2)))
tval <- as.numeric(as.character(unlist(test$spam)))

roc_obj <- roc(pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.88"
```

(f) Learn a support vector machine (SVM) with a RBF kernel to predict spam. Use cross-validation to pick the best parameters for the kernel and SVM on the training set. Consider at least the following values for cost: {0:01; 0:1; 1; 10; 100} Calculate and report the accuracy, error, and AUC performance on the testing data for the best model found.

```
library("e1071")


#tuning the levels as numbers for training data
levels(train$day.of.week) <- c("1", "2", "3", "4", "5", "6", "7")
levels(train$box) <- c("1", "2")
levels(train$local) <- c("1", "2")
levels(train$name) <- c("1", "2", "3")
levels(train$credit) <- c("1", "2")
levels(train$sucker) <- c("1", "2")
levels(train$porn) <- c("1", "2")
levels(train$chain) <- c("1", "2")
levels(train$username) <- c("1", "2")
levels(train$large.text) <- c("1", "2")
levels(train$spam) <- c("1", "2")

indx <- sapply(train, is.factor)
train[indx] <- lapply(train[indx], function(x) as.numeric(levels(x))[x])

#applying svm
tune_out <- tune.svm(x=train[,-15],y=train[,15],cost=c(0.01, 0.1, 1, 10,
100),kernel="radial",data = train)

#build model with best values of cost
svm_model <- svm(spam~ ., data=train, method="C-classification",
kernel="radial",cost=tune_out$best.parameters$cost)
#test set predictions


#tuning the levels as numbers for testing data
levels(test$day.of.week) <- c("1", "2", "3", "4", "5", "6", "7")
levels(test$box) <- c("1", "2")
levels(test$local) <- c("1", "2")
levels(test$name) <- c("1", "2", "3")
levels(test$credit) <- c("1", "2")
levels(test$sucker) <- c("1", "2")
levels(test$porn) <- c("1", "2")
levels(test$chain) <- c("1", "2")
levels(test$username) <- c("1", "2")
levels(test$large.text) <- c("1", "2")
levels(test$spam) <- c("1", "2")

indx <- sapply(test, is.factor)
test[indx] <- lapply(test[indx], function(x) as.numeric(levels(x))[x])

pred3 <-predict(svm_model,test)
```

```
pred3_new <- c()
for (j in 1:length(pred3)){
  if(abs(pred3[j]-1) < abs(pred3[j]-2)){
    pred3_new <- c(pred3_new,1)
  }else{
    pred3_new <- c(pred3_new,2)
  }
}


#calculating accuracy, error and AUC
confMat <- table(pred3_new,test$spam)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.92"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.08"

library(pROC)
#pred3 <- as.numeric(as.character(unlist(pred3)))
tval <- as.numeric(as.character(unlist(test$spam)))

roc_obj <- multiclass.roc(pred3_new, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.91"
```

(g) Examine which samples are mis-classified by some of the above models (construct a matrix with a column of predictions for each method: Naive Bayes, Decision Trees, SVMs). Create an ensemble predictor that takes the majority vote of the three models. Calculate and report the accuracy, error, and AUC performance on the testing data.

```
#combining all the accuracies into a data frame
com_pred <- cbind(pred,vec_new,pred3_new)
com_pred <- as.data.frame(com_pred)
names(com_pred) <- c("NB","DT","SVM")


#Creating the ensemble prediction of the three classifiers
fin_pred <- c()
for (i in 1:nrow(com_pred))
  {
  one_c <- 0
  two_c <- 0
  for (j in 1:length(com_pred[i,]))
```

```
    {
    if(com_pred[i,j] == 1)
    {
      one_c <- one_c + 1
    }
    else
    {
    two_c <- two_c + 1
    }
  }
  if(one_c > two_c)
  {
    fin_pred <- c(fin_pred,1)
  }
  else
  {
    fin_pred <- c(fin_pred,2)
  }

}


#calculating accuracy, error and AUC
confMat <- table(fin_pred,test$spam)
accuracy <- sum(diag(confMat))/sum(confMat)
print(paste0('Accuracy: ',round(accuracy,2)))
```

```
## [1] "Accuracy: 0.91"
```

```
print(paste0('Error Rate: ',round((1-accuracy),2)))
```

```
## [1] "Error Rate: 0.09"
```

```
library(pROC)

tval <- as.numeric(as.character(unlist(test$spam)))

roc_obj <- multiclass.roc(fin_pred, tval)
print(paste0('AUC: ',round(auc(roc_obj),2)))
```

```
## [1] "AUC: 0.9"
```

(h) (8 points) Use bagging to fit an ensemble of 100 trees to the training data. Report the error rate on the testing data.

```
library(adabag)

library(ipred)

library(rpart)
```

```
#Bagging to fit 100 trees

bag <- bagging(spam ~.,data=train,mfinal=100)

bag_pred <- predict(bag,newdata=test)


#calculating the error rate
pred_bag <- c()
for (j in 1:length(bag_pred)){
  if(abs(bag_pred[j]-1) < abs(bag_pred[j]-2)){
    pred_bag <- c(pred_bag,1)
  }else{
    pred_bag <- c(pred_bag,2)
  }
}

confMat <- table(pred_bag,test$spam)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.09"
```

(i) Use boosting to fit an ensemble of 100 trees to the training data. Report the error rate on the testing data.

```
library(adabag);
library(ipred);
library(rpart);

train$spam <- as.factor(train$spam)


#Boosting to fit 100 trees
boost <- boosting(spam ~., data=train, mfinal=100)


#calculating the error rate
boost_pred <- predict.boosting(boost,newdata=test)
confMat <- boost_pred$confusion

print(paste0('Error: ',round(boost_pred$error,2)))

## [1] "Error: 0.08"
```

6. Naive Bayes Classification

(a) Split the data into training and testing data using a 75/25 split.

```
fruit <- read.table("fruit.txt", sep=",")
fruit <- as.data.frame(fruit)
names(fruit) <- c('Type', 'Weight', 'Height', 'Width')

dim(fruit)

## [1] 54  4

indexes = sample(1:nrow(fruit), size=0.25*nrow(fruit))
# Split data
test = fruit[indexes,]
dim(test)

## [1] 13  4

train = fruit[-indexes,]
dim(train)

## [1] 41  4
```

(b) Estimate the probabilities needed for Naive Bayes Classification using Laplace smoothing

```
# we calculate laplace probabilities for k=1
den <- nrow(train)
den <- den + 3 # because we have 3 fruits type here so domain here is 3

#Prior Probabilities
p_t1 <- (sum(train$Type == 1) + 1)/den
p_t2 <- (sum(train$Type == 2) + 1)/den
p_t3 <- (sum(train$Type == 3) + 1)/den

#filtering out the 3 fruits table
c1_tab = train[train$Type == 1, ]
c2_tab = train[train$Type == 2, ]
c3_tab = train[train$Type == 3, ]

########### Conditional probabilities for apple
#Weight
den <- nrow(c1_tab)
den <- den + 2 # because we have 2 weight type here so domain here is 2
p_w0_t1 = (sum(c1_tab$Weight == 0) + 1)/den
p_w1_t1 = (sum(c1_tab$Weight == 1) + 1)/den

#height
den <- nrow(c1_tab)
den <- den + 3 # because we have 3 height type here so domain here is 3
p_h0_t1 = (sum(c1_tab$Height == 0) + 1)/den
p_h1_t1 = (sum(c1_tab$Height == 1) + 1)/den
```

```
p_h2_t1 = (sum(c1_tab$Height == 2) + 1)/den

#width
den <- nrow(c1_tab)
den <- den + 3 # because we have 3 width type here so domain here is 3
p_wd0_t1 = (sum(c1_tab$Width == 0) + 1)/den
p_wd1_t1 = (sum(c1_tab$Width == 1) + 1)/den
p_wd2_t1 = (sum(c1_tab$Width == 2) + 1)/den

########### Conditional probabilities for orange
#Weight
den <- nrow(c2_tab)
den <- den + 2 # because we have 2 weight type here so domain here is 2
p_w0_t2 = (sum(c2_tab$Weight == 0) + 1)/den
p_w1_t2 = (sum(c2_tab$Weight == 1) + 1)/den

#height
den <- nrow(c2_tab)
den <- den + 3 # because we have 3 height type here so domain here is 3
p_h0_t2 = (sum(c2_tab$Height == 0) + 1)/den
p_h1_t2 = (sum(c2_tab$Height == 1) + 1)/den
p_h2_t2 = (sum(c2_tab$Height == 2) + 1)/den

#width
den <- nrow(c2_tab)
den <- den + 3 # because we have 3 width type here so domain here is 3
p_wd0_t2 = (sum(c2_tab$Width == 0) + 1)/den
p_wd1_t2 = (sum(c2_tab$Width == 1) + 1)/den
p_wd2_t2 = (sum(c2_tab$Width == 2) + 1)/den

########### Conditional probabilities for lemon
#Weight
den <- nrow(c3_tab)
den <- den + 2 # because we have 2 weight type here so domain here is 2
p_w0_t3 = (sum(c3_tab$Weight == 0) + 1)/den
p_w1_t3 = (sum(c3_tab$Weight == 1) + 1)/den

#height
den <- nrow(c3_tab)
den <- den + 3 # because we have 3 height type here so domain here is 3
p_h0_t3 = (sum(c3_tab$Height == 0) + 1)/den
p_h1_t3 = (sum(c3_tab$Height == 1) + 1)/den
p_h2_t3 = (sum(c3_tab$Height == 2) + 1)/den

#width
den <- nrow(c3_tab)
den <- den + 3 # because we have 3 width type here so domain here is 3
p_wd0_t3 = (sum(c3_tab$Width == 0) + 1)/den
p_wd1_t3 = (sum(c3_tab$Width == 1) + 1)/den
p_wd2_t3 = (sum(c3_tab$Width == 2) + 1)/den
```

(c) Report the predicted class on the test samples using the estimated parameters. Do these calculation using code/functions that you create and the probabilities estimated in part (b);

```
#test for apple
c_app <- c()

for (i in 1:nrow(test))
{
  w = 0
  h = 0
  wd = 0
  val1 = 0

  if(test[i,]$Weight == 0)
  {
    w = p_w0_t1
  }
  if(test[i,]$Weight == 1)
  {
    w = p_w1_t1
  }

  if(test[i,]$Height == 0)
  {
    h = p_h0_t1
  }
  if(test[i,]$Height == 1)
  {
    h = p_h1_t1
  }
  if(test[i,]$Height == 2)
  {
    h = p_h2_t1
  }

  if(test[i,]$Width == 0)
  {
    wd = p_wd0_t1
  }
  if(test[i,]$Width == 1)
  {
    wd = p_wd1_t1
  }
  if(test[i,]$Width == 2)
  {
    wd = p_wd2_t1
  }
    #Naive Bayes
  val1 = w * h * wd * p_t1

  c_app <- c(c_app, val1)
}
```

```r
#test for orange
c_org <- c()

for (i in 1:nrow(test))
{
  w = 0
  h = 0
  wd = 0
  val1 = 0
  if(test[i,]$Weight == 0)
  {
    w = p_w0_t2
  }
  else if(test[i,]$Weight == 1)
  {
    w = p_w1_t2
  }

  if(test[i,]$Height == 0)
  {
    h = p_h0_t2
  }
  else if(test[i,]$Height == 1)
  {
    h = p_h1_t2
  }
  else if(test[i,]$Height == 2)
  {
    h = p_h2_t2
  }

  if(test[i,]$Width == 0)
  {
    wd = p_wd0_t2
  }
  else if(test[i,]$Width == 1)
  {
    wd = p_wd1_t2
  }
  else if(test[i,]$Width == 2)
  {
    wd = p_wd2_t2
  }
  #Naive Bayes
  val1 = w * h * wd * p_t2

  c_org <- c(c_org, val1)
}

#test for lemon
c_lem <- c()
```

```
for (i in 1:nrow(test))
{
  w = 0
  h = 0
  wd = 0
  val1 = 0
  if(test[i,]$Weight == 0)
  {
    w = p_w0_t3
  }
  else if(test[i,]$Weight == 1)
  {
    w = p_w1_t3
  }

  if(test[i,]$Height == 0)
  {
    h = p_h0_t3
  }
  else if(test[i,]$Height == 1)
  {
    h = p_h1_t3
  }
  else if(test[i,]$Height == 2)
  {
    h = p_h2_t3
  }

  if(test[i,]$Width == 0)
  {
    wd = p_wd0_t3
  }
  else if(test[i,]$Width == 1)
  {
    wd = p_wd1_t3
  }
  else if(test[i,]$Width == 2)
  {
    wd = p_wd2_t3
  }
    #Naive Bayes
  val1 = w * h * wd * p_t2

  c_lem <- c(c_lem, val1)
}


#combining all probabilities after Naïve Bayes into a data frame
df_acc <- data.frame(c_app, c_org, c_lem)
```

```
c_acc_1 <- c()


#calculating the test result
for (i in 1:nrow(df_acc))
{
  a = df_acc[i,1]
  b = df_acc[i,2]
  c = df_acc[i,3]
  val = 0

  if(a >= b && a >= c)
  {
    val = 1
  }
  if(b >= a && b >= c)
  {
    val = 2
  }
  if(c >= a && c >= b)
  {
      val = 3
  }

  c_acc_1 <- c(c_acc_1,val)
}

#calculating accuracy, error and AUC
confMat <- table(c_acc_1,test$Type)
accuracy <- sum(diag(confMat))/sum(confMat)
print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.46"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.54"

library(pROC)

roc_obj <- multiclass.roc(c_acc_1,test$Type)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.9"
```

(d) Confirm the results above using a Naive Bayes classifier available a function or package.

```
library(e1071)

#using Naïve Bayes Classifier
nb_model <- naiveBayes(as.factor(Type) ~., data = train)
```

```
pred <- predict(nb_model, test)


#calculating accuracy, error and AUC
confMat <- table(pred,test$Type)
accuracy <- sum(diag(confMat))/sum(confMat)

print(paste0('Accuracy: ',round(accuracy,2)))

## [1] "Accuracy: 0.46"

print(paste0('Error Rate: ',round((1-accuracy),2)))

## [1] "Error Rate: 0.54"

library(pROC)

pred_6 <- as.numeric(as.character(unlist(pred)))

roc_obj <- multiclass.roc(pred_6,test$Type)
print(paste0('AUC: ',round(auc(roc_obj),2)))

## [1] "AUC: 0.9"
```

We see that both of our results match.

(e) Repeat the evaluation using a 75/25 split 10 times. Report the accuracy, sensitivity, and specificity for each of the 10 repetitions and averaged over the 10 repetitions.

```
c_acc_6 <- c()

df_sen <- data.frame(Apple = numeric(0), Orange = numeric(0), Lemon = numeric(0))
df_spe <- data.frame(Apple = numeric(0), Orange = numeric(0), Lemon = numeric(0))

#running 10 times
for(i in 1:10)
{
  print(paste0("Split Number", i))

  indexes = sample(1:nrow(fruit), size=0.25*nrow(fruit))
  # Split data
  test = fruit[indexes,]
  train = fruit[-indexes,]


  nb_model <- naiveBayes(as.factor(Type) ~., data = train)
  pred <- predict(nb_model, test)

  confMat <- table(pred,test$Type)
  accuracy <- sum(diag(confMat))/sum(confMat)
  print(paste0('Accuracy: ',round(accuracy,2)))
```

```
  c_acc_6 <- c(c_acc_6, accuracy)

  c_mat <- confusionMatrix(confMat)

  df_6 <- as.data.frame(c_mat$byClass)
  print("Sensitivity by each class(Apple,Orange,Lemon)")
  print(df_6$Sensitivity)

  df_sen <- rbind(df_sen, df_6$Sensitivity)

  print("Specificity by each class(Apple,Orange,Lemon)")
  print(df_6$Specificity)

  df_spe <- rbind(df_spe, df_6$Specificity)

  print("_____________________________")
}
```

```
## [1] "Split Number1"
## [1] "Accuracy: 0.38"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.5000000 0.1428571 1.0000000
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 0.7777778 0.8333333 0.5454545
## [1] "_____________________________"
## [1] "Split Number2"
## [1] "Accuracy: 0.62"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.3333333 0.2500000 1.0000000
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 0.7000000 0.7777778 1.0000000
## [1] "_____________________________"
## [1] "Split Number3"
## [1] "Accuracy: 0.85"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.8000000 0.6666667 1.0000000
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 0.875 0.900 1.000
## [1] "_____________________________"
## [1] "Split Number4"
## [1] "Accuracy: 0.62"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 1.0000000 0.2857143 1.0000000
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 0.8 1.0 0.7
## [1] "_____________________________"
## [1] "Split Number5"
## [1] "Accuracy: 0.69"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.6 0.6 1.0
## [1] "Specificity by each class(Apple,Orange,Lemon)"
```

```
## [1] 1.000 0.875 0.700
## [1] "_____________________________"
## [1] "Split Number6"
## [1] "Accuracy: 0.54"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.5 0.2 1.0
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 0.6666667 1.0000000 0.6666667
## [1] "_____________________________"
## [1] "Split Number7"
## [1] "Accuracy: 0.69"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.8 0.4 1.0
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 0.750 0.875 0.900
## [1] "_____________________________"
## [1] "Split Number8"
## [1] "Accuracy: 0.77"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.8000000 0.3333333 1.0000000
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 1.00 0.90 0.75
## [1] "_____________________________"
## [1] "Split Number9"
## [1] "Accuracy: 0.62"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.6 0.4 1.0
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 1.000 0.875 0.600
## [1] "_____________________________"
## [1] "Split Number10"
## [1] "Accuracy: 0.69"
## [1] "Sensitivity by each class(Apple,Orange,Lemon)"
## [1] 0.6000000 0.6666667 1.0000000
## [1] "Specificity by each class(Apple,Orange,Lemon)"
## [1] 0.7500000 0.7142857 1.0000000
## [1] "_____________________________"
```

```
print(paste0('Average Accuracy: ',round(sum(c_acc_6)/length(c_acc_6),2)))
```

```
## [1] "Average Accuracy: 0.65"
```

```
colnames(df_sen) <- c("Apple","orange","Lemon")
colnames(df_spe) <- c("Apple","orange","Lemon")
```

```
print("Average Sensitivity:")
```

```
## [1] "Average Sensitivity:"
```

```
print(colMeans(df_sen))
```

```
##     Apple    orange     Lemon
## 0.6533333 0.3945238 1.0000000
```

```r
print("Average Specificity:")
```

```
## [1] "Average Specificity:"
```

```r
print(colMeans(df_spe))
```

```
##     Apple    orange     Lemon
## 0.8319444 0.8750397 0.7862121
```