BIOST 546 - HW4

John Schoof

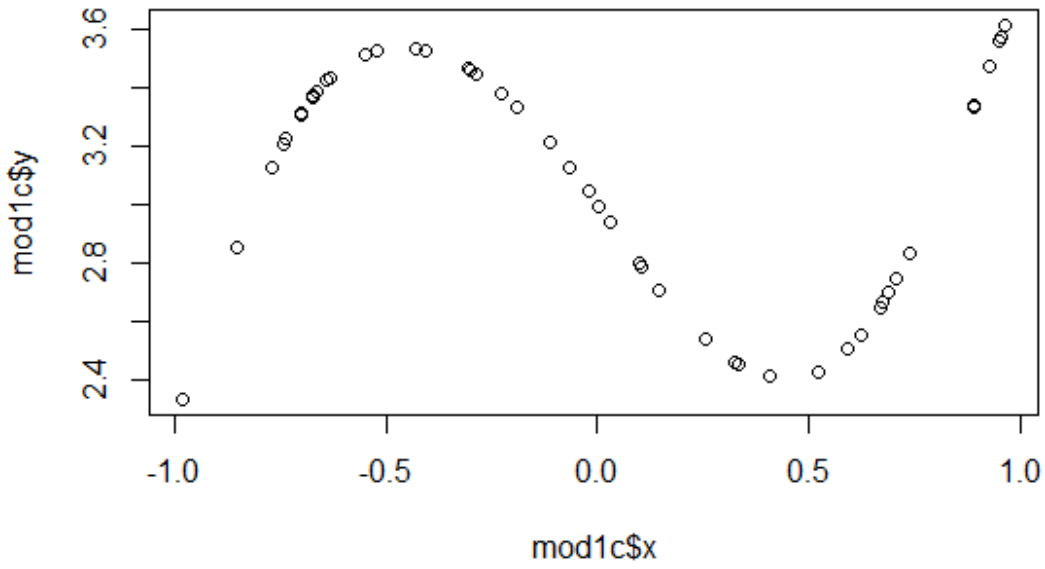
3/4/2021

Question 1: Nonlinear Regression

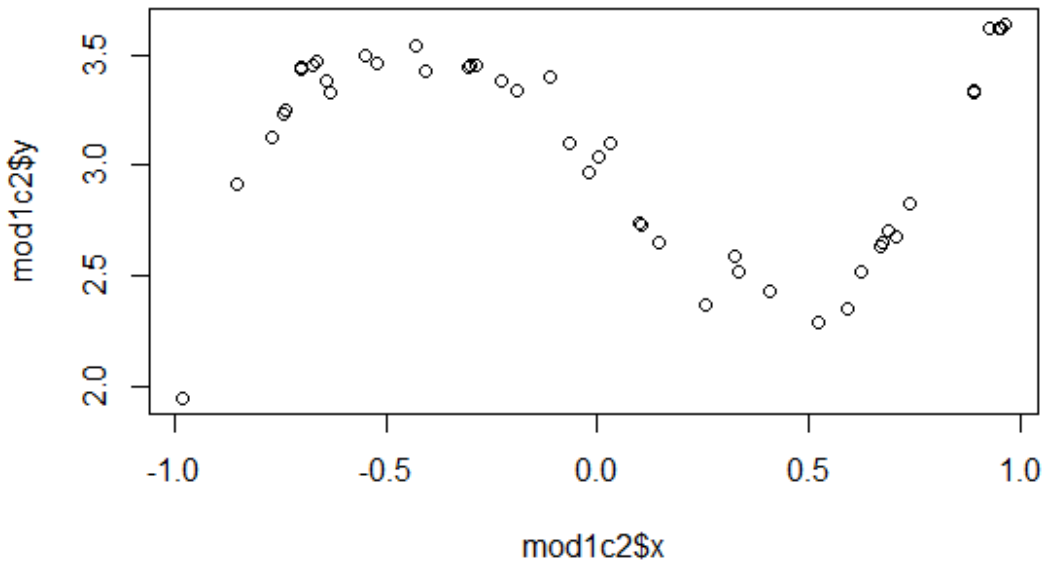
```
## generate data
  set.seed(2)
  x <- runif(50, -1, 1)
  noise <- rnorm(50, 0, 0.1)

## generate response vector
  y <- 3 - 2*x + 3*x^3 + noise
```

1a-b. The above two code chunks gnerate a set of random data, with predictor x with values between zero and one, and response variable y that is a result of the above polynomial function and a noise variable.

```
## smoothing spline models
  mod1c<-smooth.spline(x, y,lambda = 1e-3)
  plot(mod1c)
```

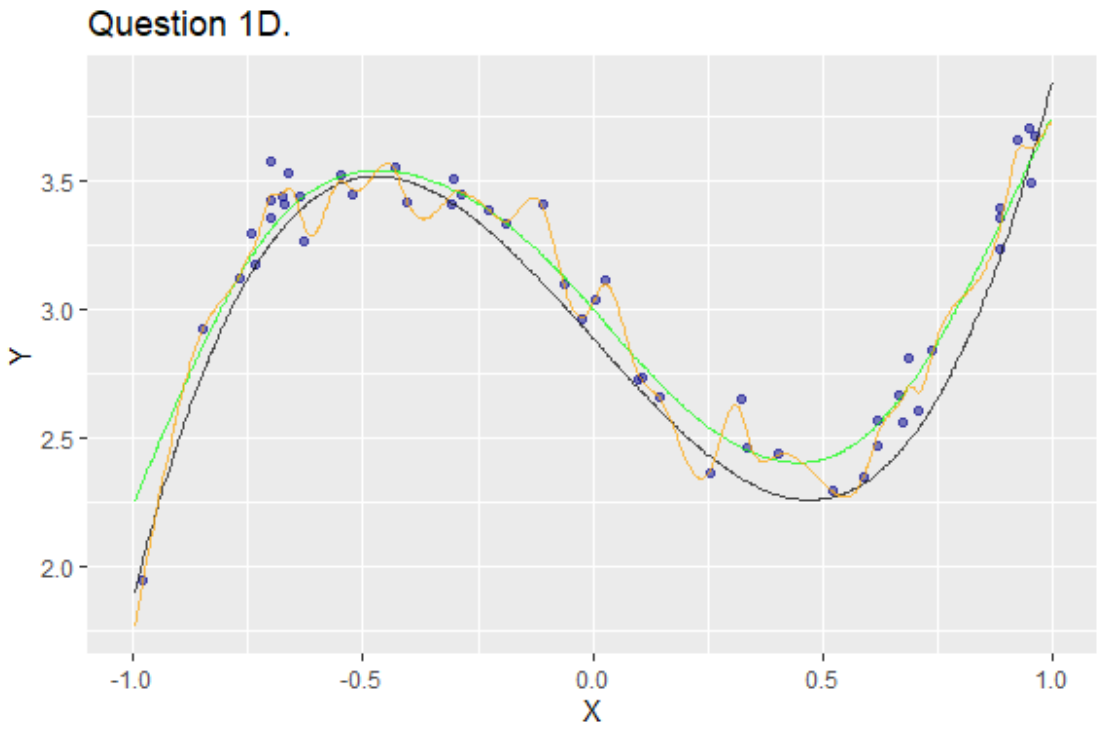



```
  mod1c2<-smooth.spline(x, y,lambda = 1e-7)
  plot(mod1c2)
```



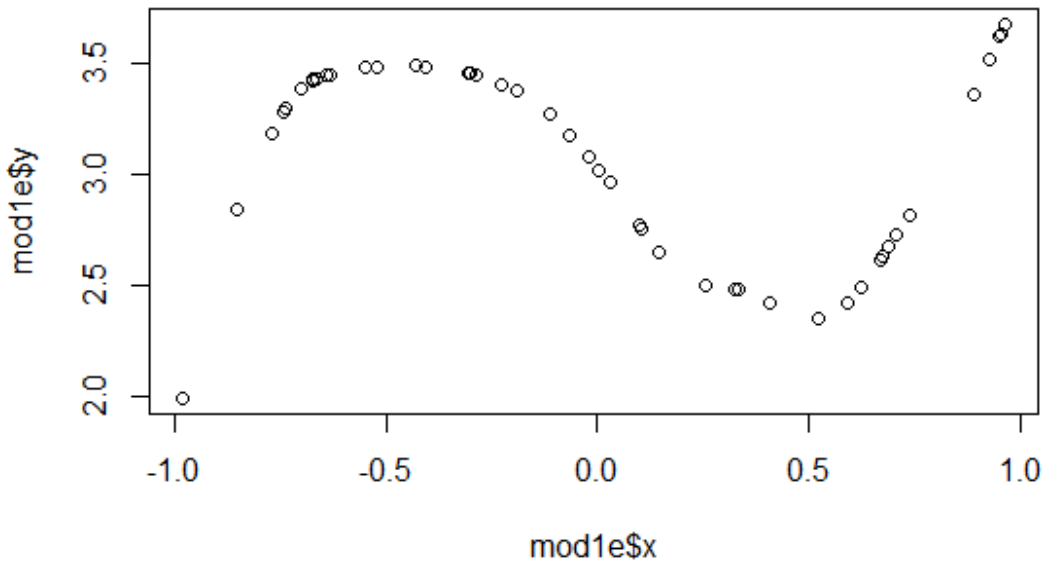
1c. The above two plots show the fitted smoothed spline models of the polynomial function defined in 1b with different levels of lambda. The plots are fit using lambda values of 1e^-3 and 1e^-7, respectively.

```
## smooth spline plot
  x.dense <- runif(n = 1000, -1, 1)
  y.dense = 3 - 2*x.dense + 3*(x.dense^3) + (rnorm(1, 0, 0.1))

  pred.mod1c <- predict(mod1c, x.dense)
  #plot(pred.mod1c)

  pred.mod1c2 <- predict(mod1c2, x.dense)
  #plot(pred.mod1c2)

  ggplot()+
    geom_point(aes(x = x, y = y), color = "darkblue", alpha = 0.5) +
    geom_line(aes(x = x.dense, y = y.dense), color = "black", alpha = 0.7) +
    geom_line(aes(x = pred.mod1c$x, y = pred.mod1c$y), color = "green", alpha
= 0.7) +
    geom_line(aes(x = pred.mod1c2$x, y = pred.mod1c2$y), color = "orange", al
pha = 0.7) +
    labs(x = "X", y = "Y", title ="Question 1D.")
```



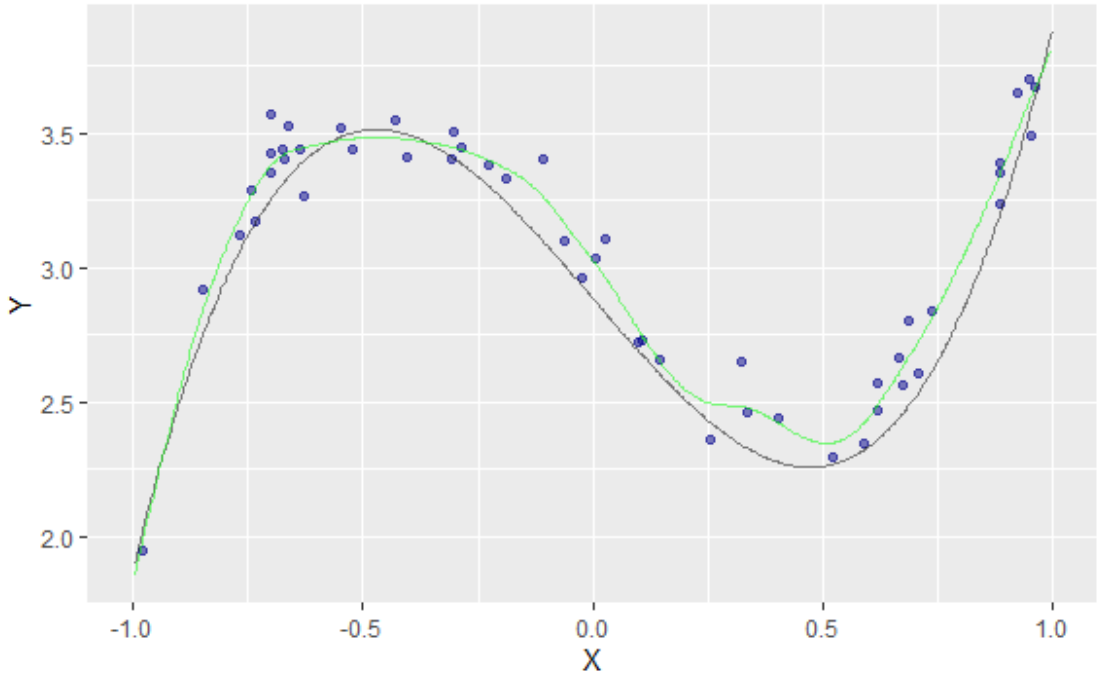
1d. The above plot shows the different functions we have seen thus far. The smooth black line is the true function f(x). The dark blue points are the 50 observations generated in 1a and 1b. The smooth green line is the spline model with a lambda value of 1e^-3, and the orange line is the spline model with a lambda value of 1e^-7.

```
## smooth spline cv
  mod1e <-smooth.spline(x = x, y = y,cv=TRUE)
  plot(mod1e)
```



1e. The above plot shows the smoothed spline model of the same function from 1a and 1b, but we now use cross validation to select the the value of lambda that minimizes the prediction error.

```
## smooth spline plot
  pred.mod1e <- predict(mod1e, x.dense)
  #plot(pred.mod1e)

  ggplot()+
    geom_point(aes(x = x, y = y), color = "darkblue", alpha = 0.5) +
    geom_line(aes(x = x.dense, y = y.dense), color = "black", alpha = 0.5) +
    geom_line(aes(x = pred.mod1e$x, y = pred.mod1e$y), color = "green", alpha
= 0.5) +
    labs(x = "X", y = "Y", title ="")
```



1f. The above plot is similar to that of 1d. The smooth black line is still the true function f(x), and the dark blue points are still the 50 observations generated in 1a and 1b. However, in this plot, the smooth green line is the spline model with the optimal lambda value, identified using cross validation.

```
# create empty vectors
boot.pred1y <- vector(mode = "integer", length = 1000L)
boot.pred2y <- vector(mode = "integer", length = 1000L)

##bind x and y
data.xy <- as.data.frame(cbind(x, y))

## bootstrap 1000 datasets
for (i in 1:1e3) {
      sample.x <- sample(1:50, size = length(x), replace= TRUE)

      sample.data <- data.xy[sample.x, ]

      boot.spline1 <- smooth.spline(x = sample.data$x, y = sample.data$y, lam
bda = 1e-3)
      boot.spline2 <- smooth.spline(x = sample.data$x, y = sample.data$y, lam
bda = 1e-7)
      boot.pred1 <- predict(boot.spline1, 0)
      boot.pred2 <- predict(boot.spline2, 0)
      boot.pred1y[i] <- boot.pred1$y
      boot.pred2y[i] <- boot.pred2$y
}
```

```
## calculate variance
  var.mod1 <- var(boot.pred1y)
  var.mod2 <- var(boot.pred2y)
```

1g. In the above chunk of code I wrote a function that creates 1000 bootstrapped datasets by sampling from the 50 generated observations with replacement. The output of that function is two vectors, the outcome variable y values for when x = 0 for the model with lambda = 1e^-3 and 1e^-7. The variance of these values are 9.5782^{-4} and 0.0041, respectively. The model with the smaller lambda value has a higher variance. This makes sense because there is less of a penalty for over fitting the data and the smaller lambda model will perform worse when it sees new data sets. This worse performance means it has a higher variance.

Question 2: Tree Models

```
## load data
  load("heart.RData")
  data <- full
  glimpse(data)
```

```
## Rows: 371
## Columns: 13
## $ Age      <dbl> 63, 67, 67, 37, 41, 56, 62, 57, 63, 53, 57, 56, 56, 44, 5
2...
## $ Gender   <fct> 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0,
1...
## $ CP       <fct> 1, 4, 4, 3, 2, 2, 4, 4, 4, 4, 4, 2, 3, 2, 3, 3, 2, 4, 3,
2...
## $ Trestbps <dbl> 145, 160, 120, 130, 130, 120, 140, 120, 130, 140, 140, 14
0...
## $ Chol     <dbl> 233, 286, 229, 250, 204, 236, 268, 354, 254, 203, 192, 29
4...
## $ FBS      <fct> 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0...
## $ RestECG  <fct> 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 2, 2, 0, 0, 0, 0, 0, 0,
0...
## $ Thalach  <dbl> 150, 108, 129, 187, 172, 178, 160, 163, 147, 155, 148, 15
3...
## $ Exang    <fct> 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0...
## $ Oldpeak  <dbl> 2.3, 1.5, 2.6, 3.5, 1.4, 0.8, 3.6, 0.6, 1.4, 3.1, 0.4, 1.
3...
## $ Slope    <fct> 3, 2, 2, 3, 1, 1, 3, 1, 2, 3, 2, 2, 2, 1, 1, 1, 3, 1, 1,
1...
## $ Thal     <fct> 6, 3, 7, 3, 3, 3, 3, 3, 7, 7, 6, 3, 6, 7, 7, 3, 7, 3, 3,
3...
## $ Disease  <fct> No Disease, Heart Disease, Heart Disease, No Disease, No
D...
```

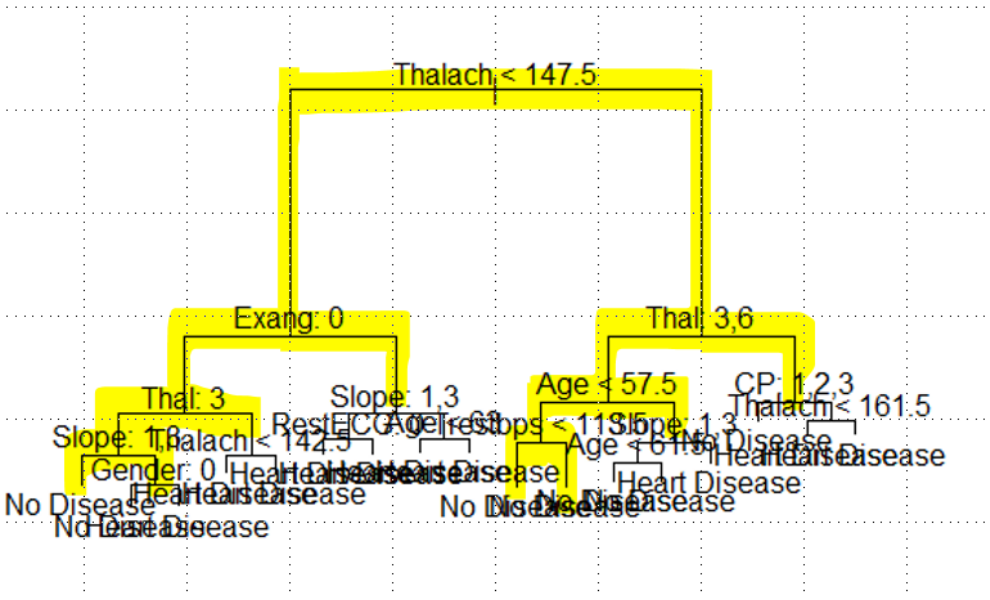
```
  table(data$Disease)
```

```
##
##    No Disease Heart Disease
##           171           200
```

```
## split sample
  set.seed(2)
  train_id = sample(nrow(data), 200)
  train = data[train_id,]
  test  = data[-train_id,]
```

2a. The Heart Disease Data Set has a sample size of 371, 12 predictor variables, and the outcome variable is defined by two classes, “Heart Disease” and “No Heart Disease.” The “Heart Disease” class has 200 observations and the “No Heart Disease” class has 171 observations. In the above code chunk I also split my sample into a training set of 200 observations and a test set of 171 observations.

```
## overgrown tree
tree.2b <- tree(Disease~., train)
summary(tree.2b)

##
## Classification tree:
## tree(formula = Disease ~ ., data = train)
## Variables actually used in tree construction:
## [1] "Thalach"  "Exang"    "Thal"     "Slope"    "Gender"   "RestECG"  "Age
"
## [8] "Trestbps" "CP"
## Number of terminal nodes:  17
## Residual mean deviance:  0.471 = 86.2 / 183
## Misclassification error rate: 0.11 = 22 / 200

plot(tree.2b)
text(tree.2b, pretty = 0)
```

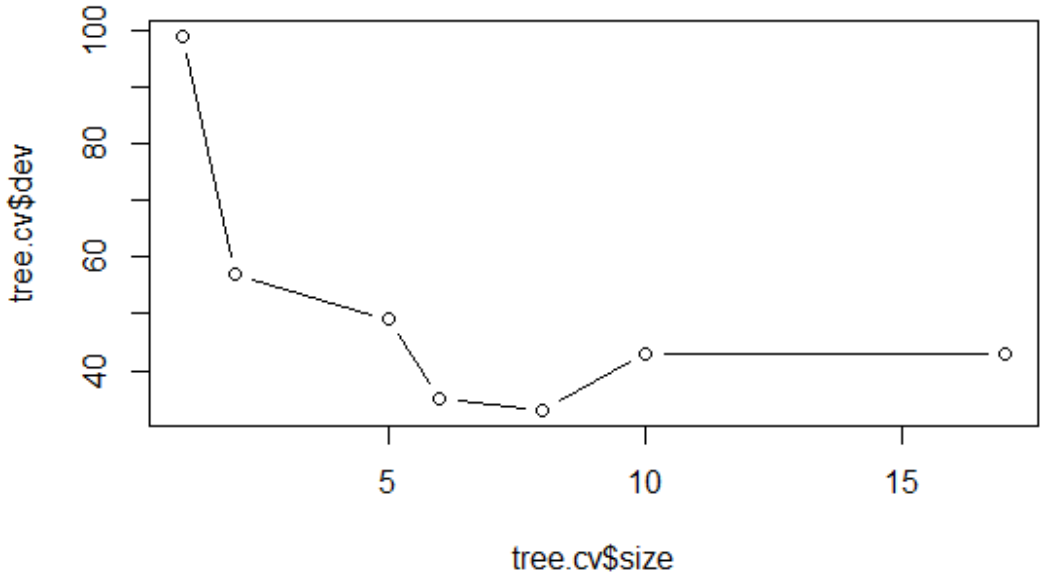


2b. The above model fits the overgrown decision tree and the plot shows the overgrown decision tree.

```
## training set
## confusion matrix (true positive rate and false positive rate)
  (tree.train.table <- table(tree.2b$y, train$Disease))

##
##                 No Disease Heart Disease
##   No Disease            96             0
##   Heart Disease          0           104

## misclassification error (proportion of test set that I predicted correctly
)
  (tree.train.accuracy <- mean(tree.2b$y == train$Disease))

## [1] 1

## test set
  pred.2c <- predict(tree.2b, type = "class", newdata = test)
## confusion matrix (true positive rate and false positive rate)
  (tree.test.table <- table(pred.2c, test$Disease))

##
## pred.2c         No Disease Heart Disease
##   No Disease            56            21
##   Heart Disease         19            75

## misclassification error (proportion of test set that I predicted correctly
)
  (tree.test.accuracy <- mean(pred.2c == test$Disease))

## [1] 0.7661
```

2c. Above are the confusion matrices and the prediction accuracy for the training and test set, respectively. The overgrown tree performs perfectly on the training set and has a prediction accuracy of 76.6% on the test set. This suggests that there is a high level of overfitting. This is what we would expect because the overgrown tree is exactly fitting the training set.

```
set.seed(2)
tree.cv <- cv.tree(tree.2b, FUN=prune.misclass)

## plot of
plot(tree.cv$size, tree.cv$dev, type="b")
```




```
tree.cv$size
```

```
## [1] 17 10  8  6  5  2  1
```

```
## tree plot
prune.tree.2b <- prune.tree(tree.2b, best = 8)
#plot(prune.tree.2b)
#text(prune.tree.2b, pretty=0)
```

2b. The above model prunes the overgrown tree using cross validation. The first plot shows the misclassification error on the tree size. The second plot shows the optimal subtree highlighted over the overgrown decision tree. The optimal subtree has a size of 8 branches.

```
## training set
## confusion matrix (true positive rate and false positive rate)
  (prune.train.table <- table(prune.tree.2b$y, train$Disease))
```

```
##
##                 No Disease Heart Disease
##   No Disease            96             0
##   Heart Disease          0           104
```

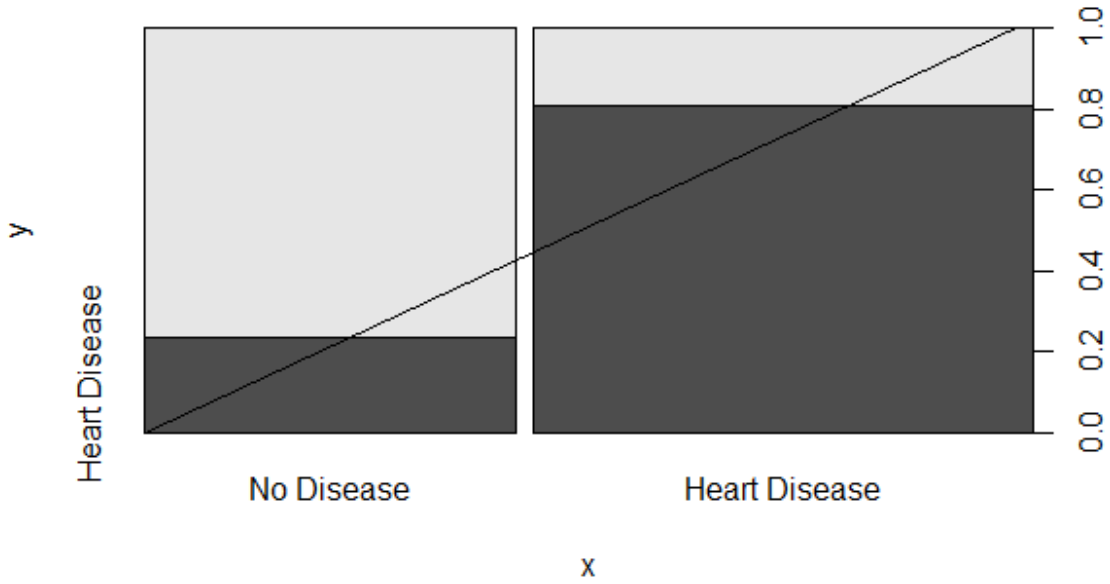
```
## misclassification error (proportion of test set that I predicted correctly
)
  (prune.train.accuracy <- mean(prune.tree.2b$y == train$Disease))
```

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

```
## test set
  prune.pred.2c <- predict(prune.tree.2b, type = "class", newdata = test)
## confusion matrix (true positive rate and false positive rate)
  (prune.test.table <- table(prune.pred.2c, test$Disease))

##
## prune.pred.2c   No Disease Heart Disease
##   No Disease            47            15
##   Heart Disease         28            81

## misclassification error (proportion of test set that I predicted correctly
)
  (prune.test.accuracy <- mean(prune.pred.2c == test$Disease))

## [1] 0.7485
```

2e. Above are the confusion matrices and the prediction accuracy for the training and test set, respectively. The optimal subtree performs perfectly on the training set and has a prediction accuracy of 74.9% on the test set, worse than the overgrown tree. This still suggests that there is a high level of overfitting.

```
## bagged model
set.seed(2)
mod2f.train <- randomForest(Disease~.,data=train, importance=TRUE)
#mod2f.train

mod2f.test <- predict(mod2f.train, newdata=test)
#mod2f.test

Disease.test <- test[,"Disease"]
plot(mod2f.test, Disease.test)
abline(0,1)
```




```
## training set
## confusion matrix (true positive rate and false positive rate)
  (bag.train.table <- table(mod2f.train$y, train$Disease))

##
##                 No Disease Heart Disease
##   No Disease            96             0
##   Heart Disease          0           104

## misclassification error (proportion of test set that I predicted correctly
)
  (bag.train.accuracy <- mean(mod2f.train$y == train$Disease))

## [1] 1

## test set
## confusion matrix (true positive rate and false positive rate)
  (bag.test.table <- table(mod2f.test, test$Disease))

##
## mod2f.test      No Disease Heart Disease
##   No Disease            56            17
##   Heart Disease         19            79

## misclassification error (proportion of test set that I predicted correctly
)
  (bag.test.accuracy <- mean(mod2f.test == test$Disease))

## [1] 0.7895
```

2f. Above I apply bagging to improve the performance of the pruned tree. The optimal subtree performs perfectly on the training set after bagging and has a prediction accuracy of 79.0% on the test set. This is the best performance that we have seen thus far in our tree models. .

```
set.seed(2)
rf.train <-randomForest(Disease~., data = train, importance=TRUE, mtry = 12/3
)
rf.test <-predict(rf.train, newdata = test)

## training set
## confusion matrix (true positive rate and false positive rate)
  (rf.train.table <- table(rf.train$y, train$Disease))

##
##                 No Disease Heart Disease
##   No Disease            96             0
##   Heart Disease          0           104

## misclassification error (proportion of test set that I predicted correctly
)
  (rf.train.accuracy <- mean(rf.train$y == train$Disease))

## [1] 1

## test set
## confusion matrix (true positive rate and false positive rate)
  (rf.test.table <- table(rf.test, test$Disease))

##
## rf.test         No Disease Heart Disease
##   No Disease            56            17
##   Heart Disease         19            79

## misclassification error (proportion of test set that I predicted correctly
)
  (rf.test.accuracy <- mean(rf.test == test$Disease))

## [1] 0.7895
```

2g. The random forest model that I fit above has exactly the same prediction accuracy on the training and test sets as the model with bagging in 2f.

2h. The bagged model and the random forest model randomly select smaller trees, so if we do not set the seed the final tree will be made up of a slightly different combination of smaller trees each time.

```
## boosted model
set.seed(2)

train$Disease.new[train$Disease == "No Disease"] <- 0
```

```r
train$Disease.new[train$Disease == "Heart Disease"] <- 1

test$Disease.new[test$Disease == "No Disease"] <- 0
test$Disease.new[test$Disease == "Heart Disease"] <- 1

boost.train <- gbm(Disease.new~.,
                   data = train,
                   distribution = "bernoulli",
                   n.trees = 500,
                   interaction.depth = 2,
                   shrinkage = 0.1,
                   )

boost.test <- predict(boost.train,
                      newdata = test,
                      type = "response",
                      n.trees = 500)

predict.class <- boost.test > 0.5

## training set
  boost.train$fit[boost.train$fit<0] <- 0
  boost.train$fit[boost.train$fit>0] <- 1
## confusion matrix (true positive rate and false positive rate)
  (boost.train.table <- table(boost.train$fit, train$Disease.new))

##
##       0   1
##   0  96   0
##   1   0 104

## misclassification error (proportion of test set that I predicted correctly
)
  (boost.train.accuracy <- mean(boost.train$fit == train$Disease.new))

## [1] 1

## test set
## confusion matrix (true positive rate and false positive rate)
  (boost.test.table <- table(predict.class, test$Disease))

##
## predict.class No Disease Heart Disease
##         FALSE         75             0
##         TRUE           0            96

## misclassification error (proportion of test set that I predicted correctly
)
  (boost.test.accuracy <- mean(predict.class == test$Disease))

## [1] 0
```

2g. After boosting the prediction accuracy is 100% for both the training and test set. I expected that the test prediction accuracy would be highest for the boosted model, however I did not expect it to be 100%. I am concerned that there is an error here.