HW 3

Ryan Dee

9/24/2024

1

Let $E[X] = \mu$. Show that $Var[X] := E[(X - E[X])^2] = E[X^2] - (E[X])^2$. Note, all you have to do is show the second equality (the first is our definition from class).

```
"E[X-E[X]^2] = E[X^2 - 2XE[X]+(E[X])^2] By Foil
            = E[X^2] - E[2XE[X]] + E[E[X]^2] Distribute the Expectation
            = E[X^2] - 2E[X]E[X] + E[X]^2->  Because E[x] is constant
            = E[X^2] - 2(E[X])^2 + E[X]^2  Multiplication of second term
            = E[X^2] - (E[X])^2           Combine like terms"
```

```
## [1] "E[X-E[X]^2] = E[X^2 - 2XE[X]+(E[X])^2] By Foil\n            = E[X^2] - E[2XE[X]] + E
[E[X]^2] Distribute the Expectation\n            = E[X^2] - 2E[X]E[X] + E[X]^2->  Because E
[x] is constant\n            = E[X^2] - 2(E[X])^2 + E[X]^2  Multiplication of second term\n
= E[X^2] - (E[X])^2           Combine like terms"
```

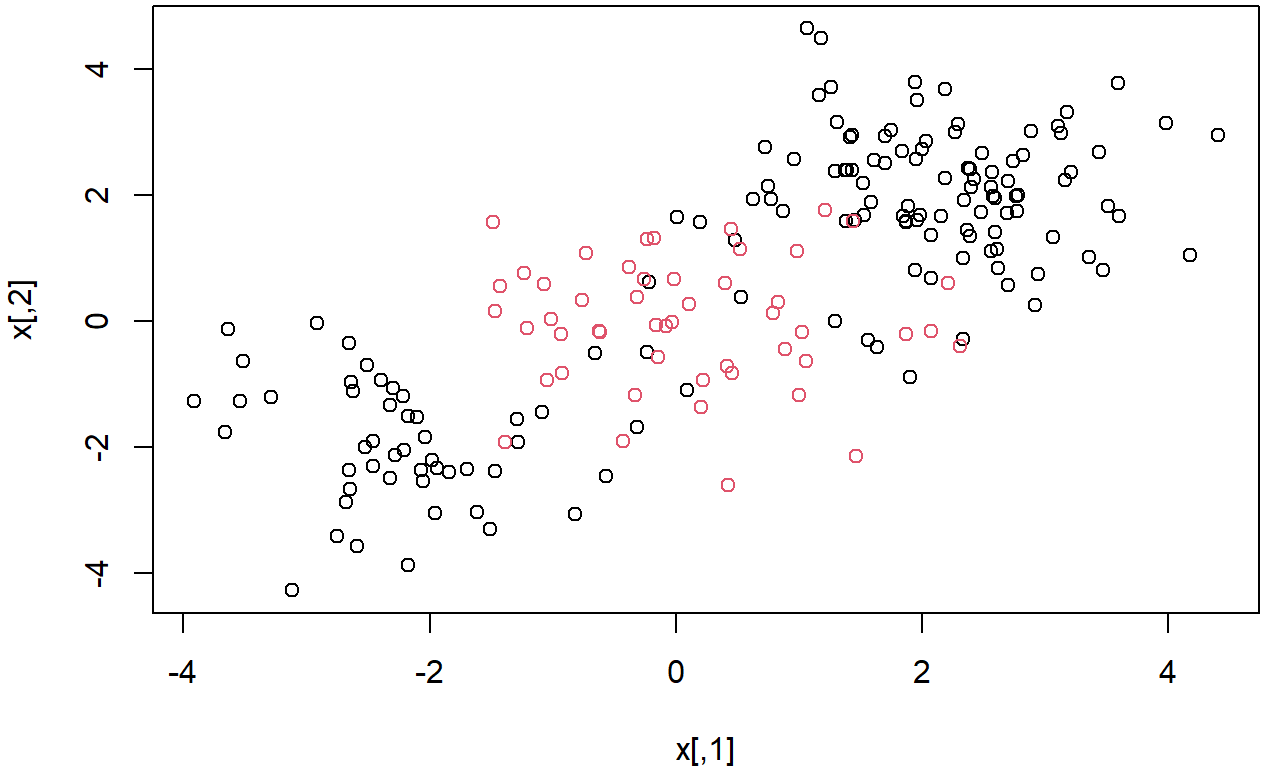
2

In the computational section of this homework, we will discuss support vector machines and tree-based methods. I will begin by simulating some data for you to use with SVM.

```
library(e1071)
```

```
## Warning: package 'e1071' was built under R version 4.2.3
```

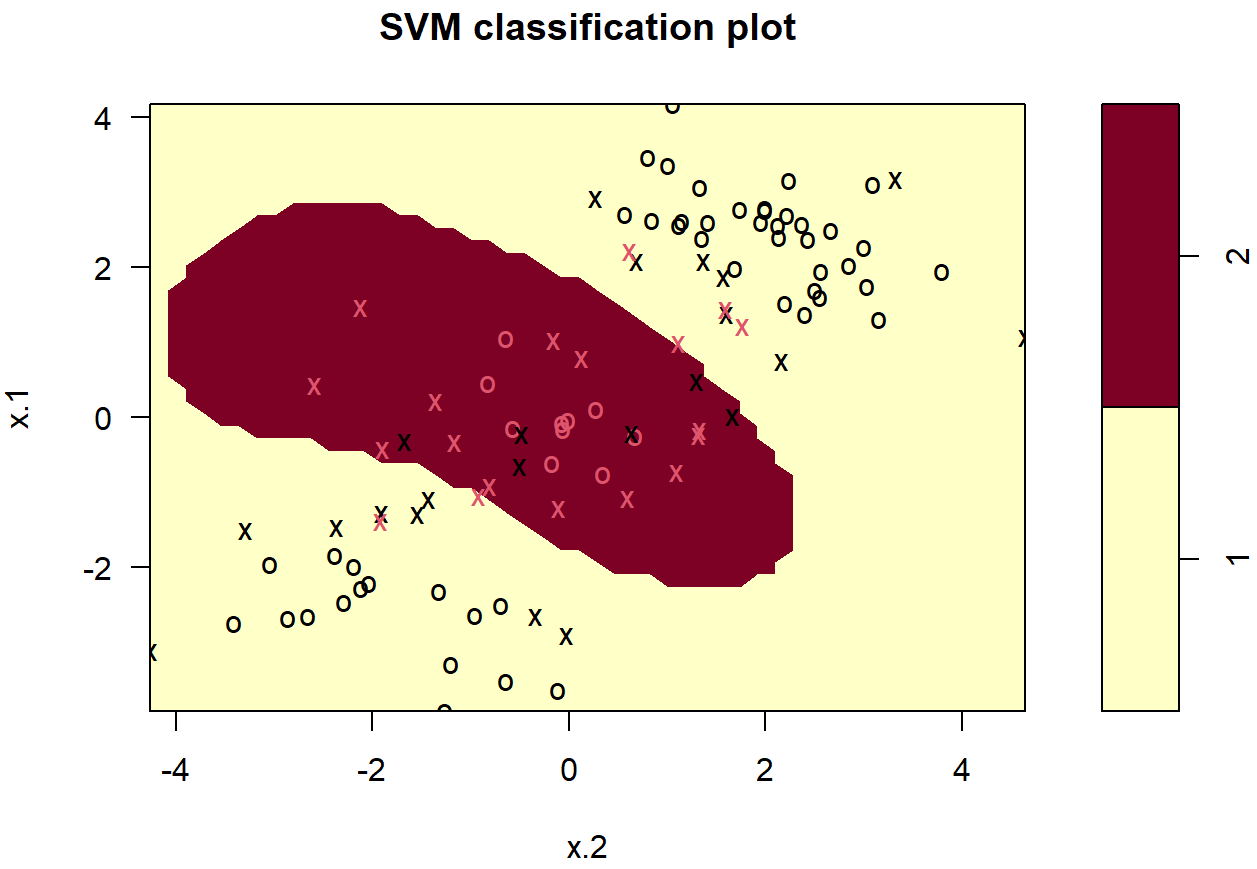
```
set.seed(1)
x=matrix(rnorm(200*2),ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)
```



2.1

Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

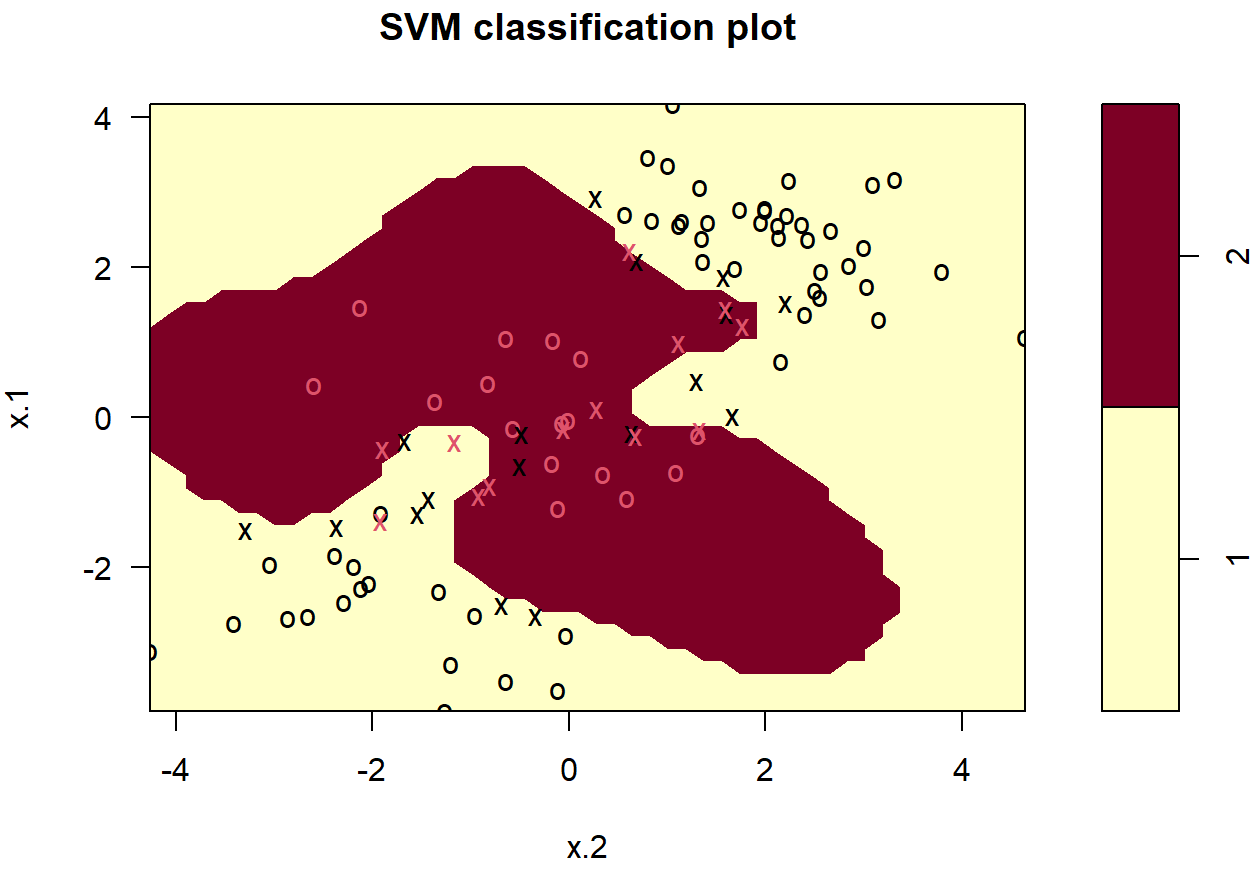
```
set.seed(1)

sample <- sample(1:nrow(dat), 0.5 * nrow(dat))
train <- dat[sample, ]
test <- dat[-sample, ]
svm_fit = svm(y~., data =train, kernel = "radial", cost =1, gamma= 1)
plot(svm_fit, train)
```



2.2

Notice that the above decision boundary is decidedly non-linear. It seems to perform reasonably well, but there are indeed some misclassifications. Let's see if increasing the cost [1] helps our classification error rate. Refit the svm with the radial kernel, $\gamma = 1$, and a cost of 10000. Plot this svm on the training data.

```
svm_fit2 = svm(y~., data =train, kernel = "radial", cost =1000, gamma = 1)
plot(svm_fit2, train)
```



2.3

It would appear that we are better capturing the training data, but comment on the dangers (if any exist), of such a model.

This runs the risk of overfitting as the test data may not fit very well compared to the training data

2.4

Create a confusion matrix by using this svm to predict on the current testing partition. Comment on the confusion matrix. Is there any disparity in our classification results?

```
#remove eval = FALSE in above
table(true=dat[-sample,"y"], pred=predict(svm_fit2, newdata=dat[-sample,]))
```

```
##      pred
## true  1  2
##    1 66 13
##    2  2 19
```

Yes, 1 seems to be predicted correctly much more than 2. ##

Is this disparity because of imbalance in the training/testing partition? Find the proportion of class `2` in your

training partition and see if it is broadly representative of the underlying 25% of class 2 in the data as a whole.

```
y_column <- train[['y']]
y_column_int <- (as.numeric(as.character(y_column)))
count_2 <- sum(y_column_int == 2)
count_2 / 100
```

```
## [1] 0.29
```

Student Response There does not seem to be a huge difference between the distribution of Y in the training data versus the data as a whole. There is only a difference of 4 between the 29 present in the training versus the 25 percent of the actual data ##

Let's try and balance the above to solutions via cross-validation. Using the `tune` function, pass in the training data, and a list of the following cost and γ values: {0.1, 1, 10, 100, 1000} and {0.5, 1,2,3,4}. Save the output of this function in a variable called `tune.out` .

```
tune_cost <- c(0.1,  1, 10, 100, 1000)
tune_gamma <- c(.5, 1,2,3,4)
set.seed(1)
tune.out <- tune(svm,
                 y ~ .,
                 data = train,
                 ranges = list(gamma = tune_gamma, cost = tune_cost))
summary(tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost
##    0.5    1
##
## - best performance: 0.12
##
## - Detailed performance results:
##    gamma  cost error dispersion
## 1    0.5 1e-01  0.28 0.15491933
## 2    1.0 1e-01  0.25 0.13540064
## 3    2.0 1e-01  0.28 0.14757296
## 4    3.0 1e-01  0.28 0.15491933
## 5    4.0 1e-01  0.29 0.14491377
## 6    0.5 1e+00  0.12 0.07888106
## 7    1.0 1e+00  0.14 0.09660918
## 8    2.0 1e+00  0.15 0.10801234
## 9    3.0 1e+00  0.15 0.10801234
## 10   4.0 1e+00  0.16 0.09660918
## 11   0.5 1e+01  0.15 0.10801234
## 12   1.0 1e+01  0.16 0.10749677
## 13   2.0 1e+01  0.19 0.15238839
## 14   3.0 1e+01  0.20 0.16329932
## 15   4.0 1e+01  0.18 0.13984118
## 16   0.5 1e+02  0.17 0.11595018
## 17   1.0 1e+02  0.21 0.15238839
## 18   2.0 1e+02  0.18 0.14757296
## 19   3.0 1e+02  0.20 0.13333333
## 20   4.0 1e+02  0.21 0.11972190
## 21   0.5 1e+03  0.23 0.14944341
## 22   1.0 1e+03  0.20 0.14142136
## 23   2.0 1e+03  0.23 0.12516656
## 24   3.0 1e+03  0.27 0.11595018
## 25   4.0 1e+03  0.31 0.15951315
```

I will take `tune.out` and use the best model according to error rate to test on our data. I will report a confusion matrix corresponding to the 100 predictions.

```
table(true=dat[-sample,"y"], pred=predict(tune.out$best.model, newdata=dat[-sample,]))
```

```
##     pred
## true  1  2
##    1 72  7
##    2  1 20
```

2.5

Comment on the confusion matrix. How have we improved upon the model in question 2 and what qualifications are still necessary for this improved model.
The confusion matrix seems to assert our new svm model better fits the testing data. The gamma and cost that we used initially are present in our tune function, so we know that our new gamma and our new cost variable will at least be as good as the ones we had before. However, we still need to qualify that ther may be a better polynomial kernel or different combinations that we have not used yet.

3

Let's turn now to decision trees.

```
library(kmed)
```

```
## Warning: package 'kmed' was built under R version 4.2.3
```

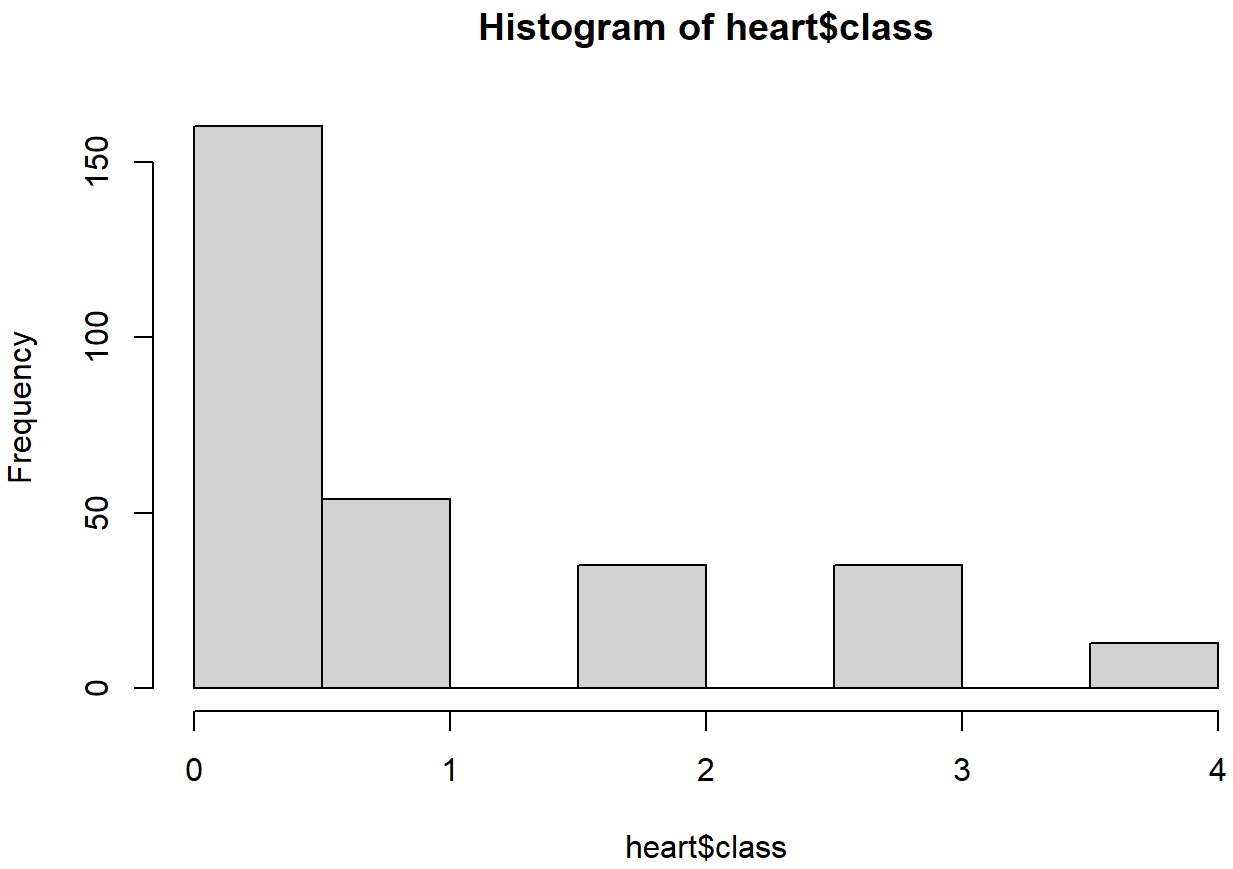
```
data(heart)
library(tree)
```

```
## Warning: package 'tree' was built under R version 4.2.3
```

3.1

The response variable is currently a categorical variable with four levels. Convert heart disease into binary categorical variable. Then, ensure that it is properly stored as a factor.

```
hist(heart$class)
```




```
heart$cp <- as.numeric(as.character(heart$cp))
heart$sex <- as.factor(heart$sex)
heart$fbs <- as.factor(heart$fbs)
heart$exang <- as.factor(heart$exang)
heart$class <- as.numeric(heart$class)
High <- ifelse(heart$cp <= 2, "low", "high")
High <- as.factor(High)
heart$high <- High
```

3.2

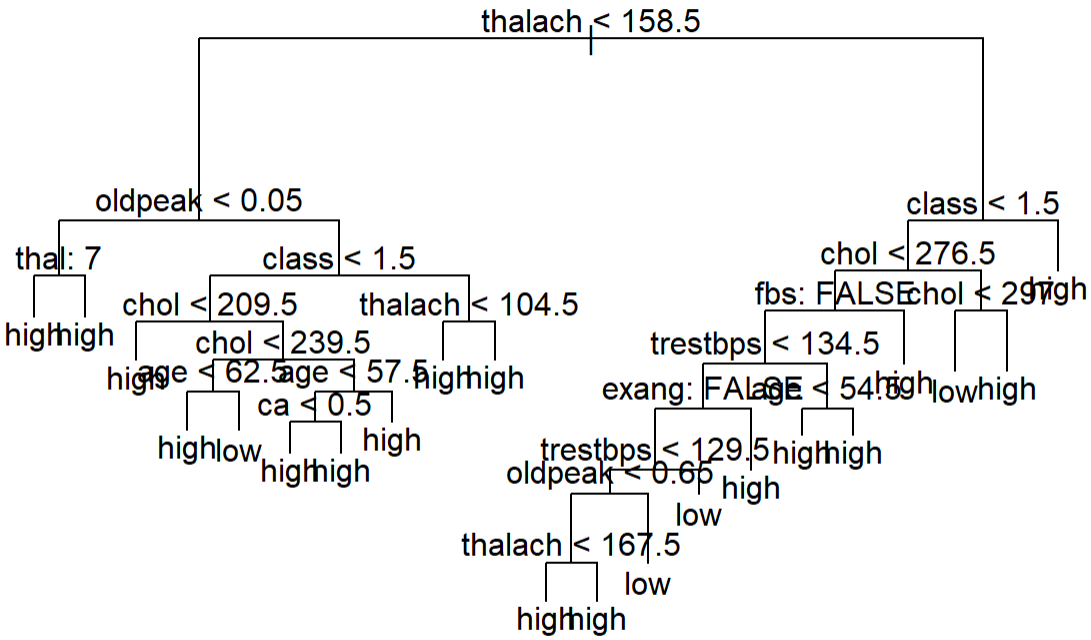
Train a classification tree on a 240 observation training subset (using the seed I have set for you). Plot the tree.

```
set.seed(101)
sample_t <- sample(1:nrow(heart), 0.81 * nrow(heart))
train_h <- heart[sample_t,]
test_h <- heart[-sample_t,]
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 4.2.3
```

```
## Loading required package: rpart
```

```
library(rpart)
heart.tree <- tree(high ~. -cp, data = heart, subset = sample_t)
par(xpd = NA) # otherwise on some devices the text is clipped

plot(heart.tree)
text(heart.tree, pretty = 0)
```



3.3

Use the trained model to classify the remaining testing points. Create a confusion matrix to evaluate performance. Report the classification error rate.

```
tree.pred <- predict(heart.tree, test_h, type = "class")
conf_matrix <- table(True = test_h$high, Predicted = tree.pred)
print(conf_matrix)
```

```
##       Predicted
## True   high low
##   high   37   3
##   low    16   1
```

```
38/(38+19)
```

```
## [1] 0.6666667
```

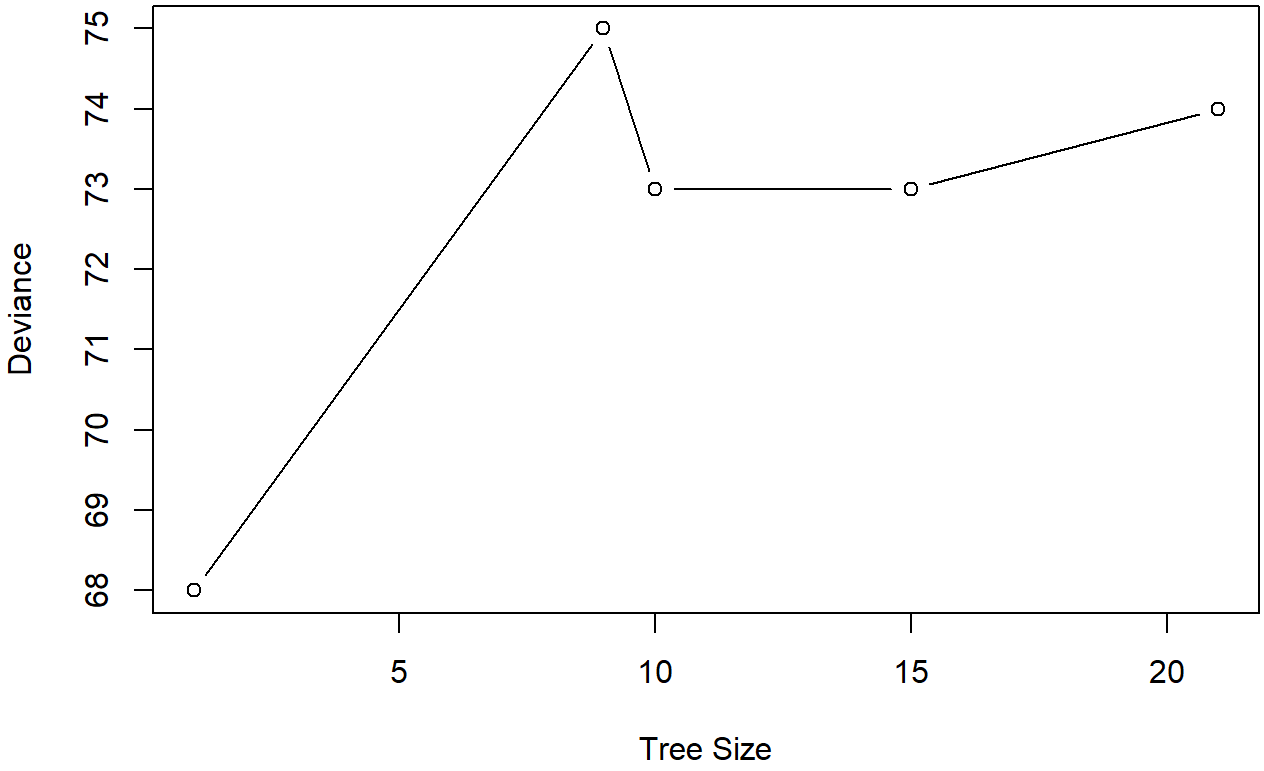
3.4

Above we have a fully grown (bushy) tree. Now, cross validate it using the `cv.tree` command. Specify cross validation to be done according to the misclassification rate. Choose an ideal number of splits, and plot this tree. Finally, use this pruned tree to test on the testing set. Report a confusion matrix and the misclassification rate.

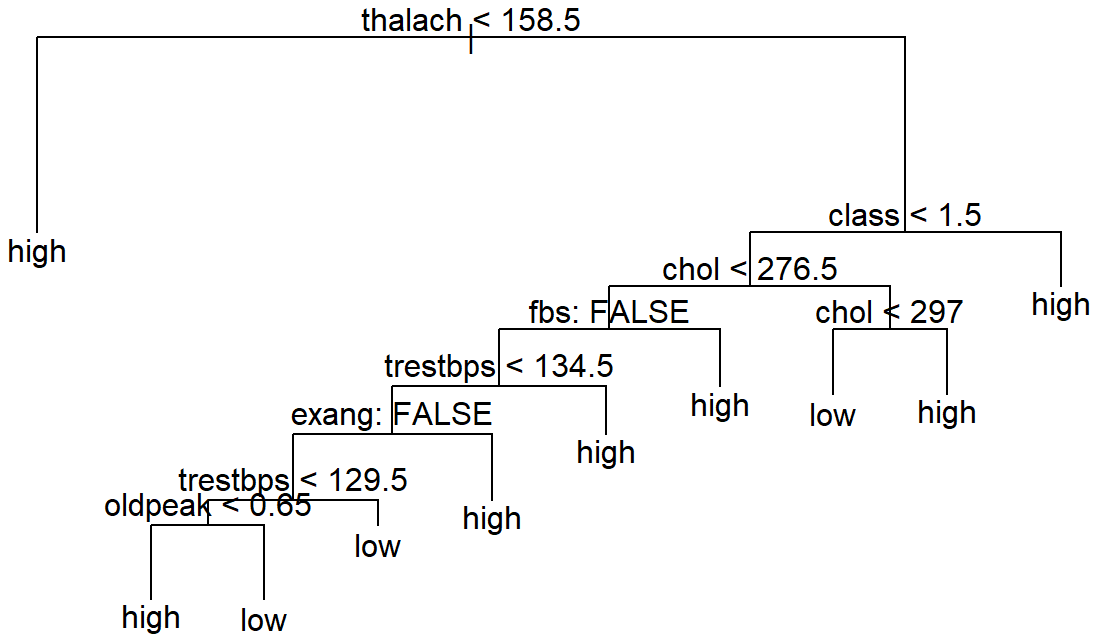
```
set.seed(101)
cv.heart <- cv.tree(heart.tree, FUN = prune.misclass)
print(cv.heart)
```

```
## $size
## [1] 21 15 10  9  1
##
## $dev
## [1] 74 73 73 75 68
##
## $k
## [1] -Inf 0.00 0.20 1.00 2.25
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
plot(cv.heart$size, cv.heart$dev, type = "b", xlab = "Tree Size", ylab = "Deviance")
```




```
prune.heart_tree <- prune.misclass(heart.tree, best = 10)
plot(prune.heart_tree)
text(prune.heart_tree, pretty=0)
```




```
tree.pred_2 <- predict(prune.heart_tree, newdata = test_h, type = "class")
conf_matrix <- table(True = test_h$high, Predicted = tree.pred_2)
print(conf_matrix)
```

```
##       Predicted
## True   high low
##   high   38   2
##   low    17   0
```

```
tree.pred <- predict(heart.tree, test_h, type = "class")
conf_matrix <- table(True = test_h$high, Predicted = tree.pred_2)
print(conf_matrix)
```

```
##       Predicted
## True   high low
##   high   38   2
##   low    17   0
```

```
38/(38+19)
```

```
## [1] 0.6666667
```

3.5

Discuss the trade-off in accuracy and interpretability in pruning the above tree.

Student Input In this instance there was no differnece between misclassification rate before and after pruning, this was able to help us classify without needing such a long tree, and makes the tree easier to use/work with. However, typically long overfit trees do not test well, and this new one while shorter and easier to interpret, may be more generalizeable.
##

Discuss the ways a decision tree could manifest algorithmic bias.
Decision trees mostly manifest algorithmic bias through fully grown trees which are not applicable and overfit the test data, or training data which does not reflect the true data.
Student Answer

---

1. Remember this is a parameter that decides how smooth your decision boundary should be↩