HW 3

Student Name

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).

$$\text{Var}(x) = E\left[(x - E(x))^2\right]$$

$$= E\left[x^2 - 2 \cdot x \cdot E(x) + E(x)^2\right]$$

$$= E(x^2) - E\left[2 \cdot x \cdot E(x)\right] + E\left[E(x)^2\right]$$

$$= E(x^2) - 2 \cdot E\left[x \cdot E(x)\right] + E(x)^2$$

$$= E(x^2) - 2 \cdot E(x) \cdot E\left[E(x)\right] + E(x)^2$$

$$= E(x^2) - 2 \cdot E(x)^2 + E(x)^2$$

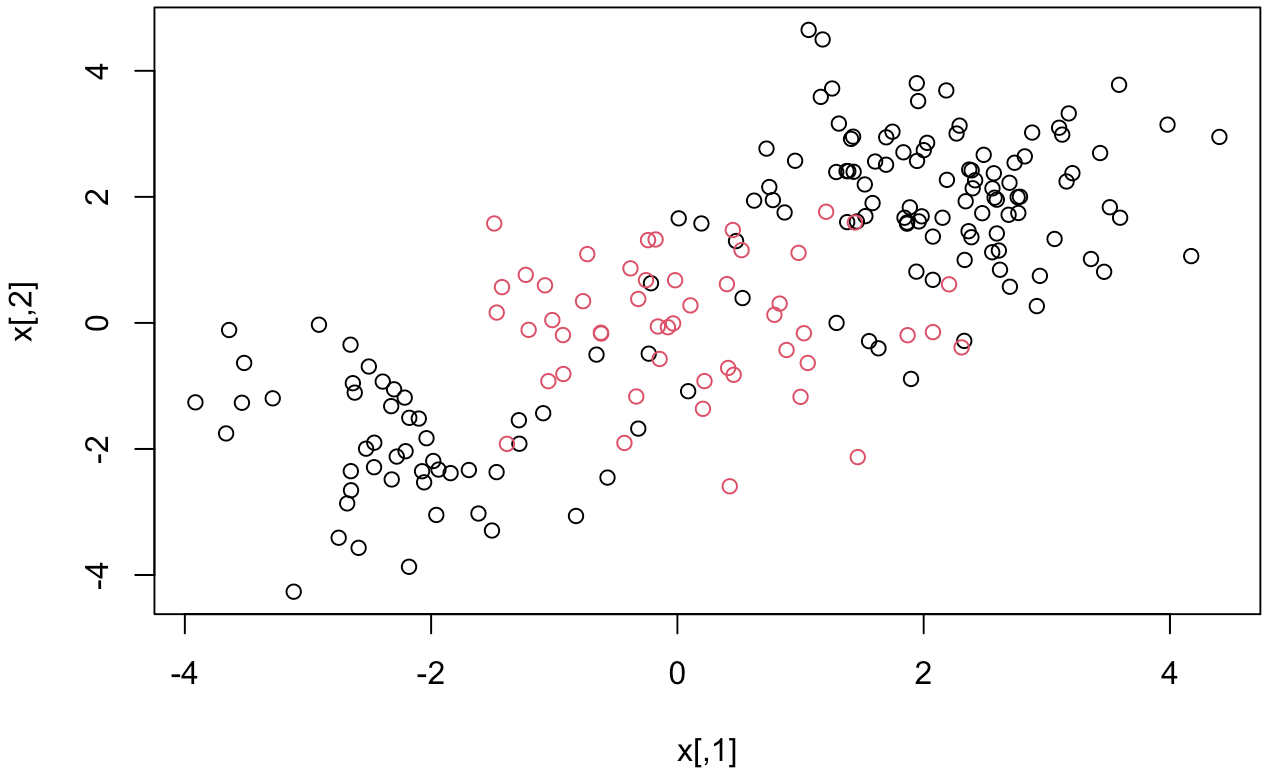
$$= E(x^2) - E(x)^2$$

We used linearity of expectations and that $E(x)$ is a constant here to separate the terms inside the expectation.

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.

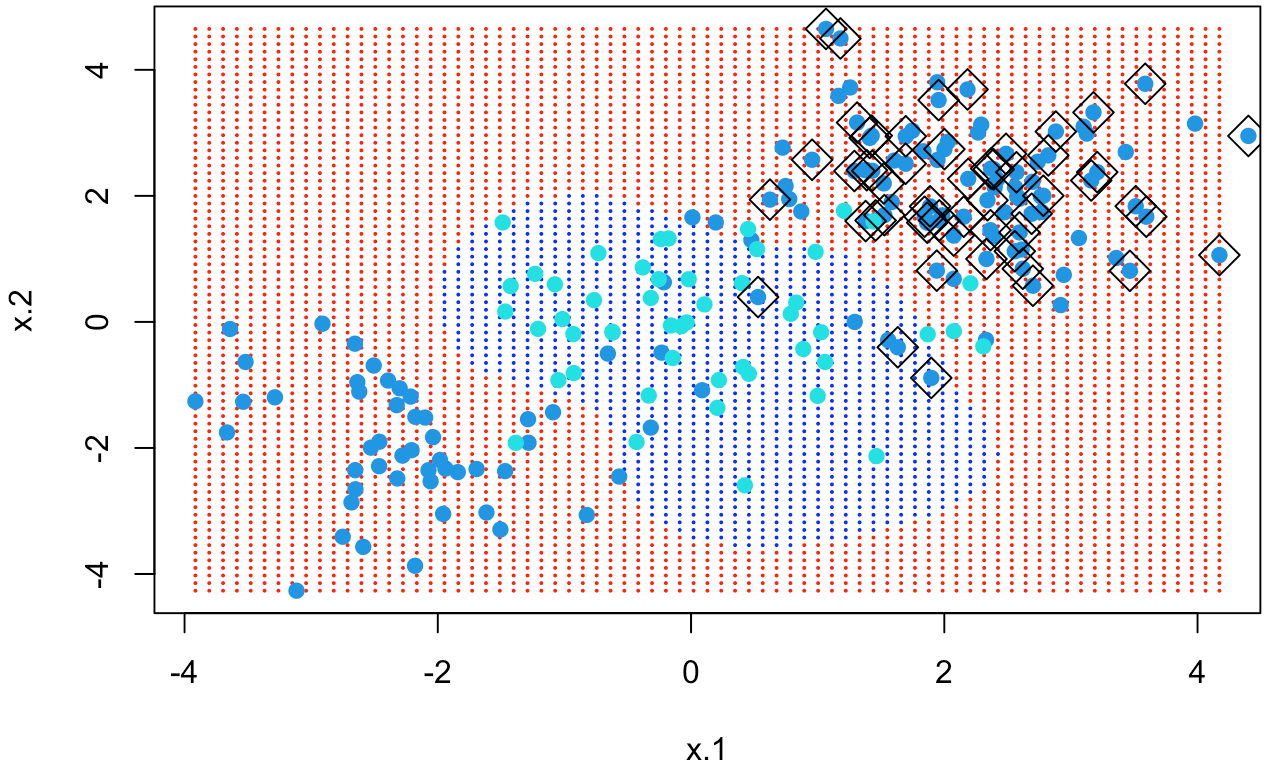
```
library(e1071)
set.seed(1)
x=matrix(rnorm(200*2),ncol=2)
x[1:100,]=x[1:100,]+2
x[101:150,]=x[101:150,]-2
y=c(rep(1,150),rep(2,50))
dat=data.frame(x=x,y=as.factor(y))
plot(x, col=y)
```



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.

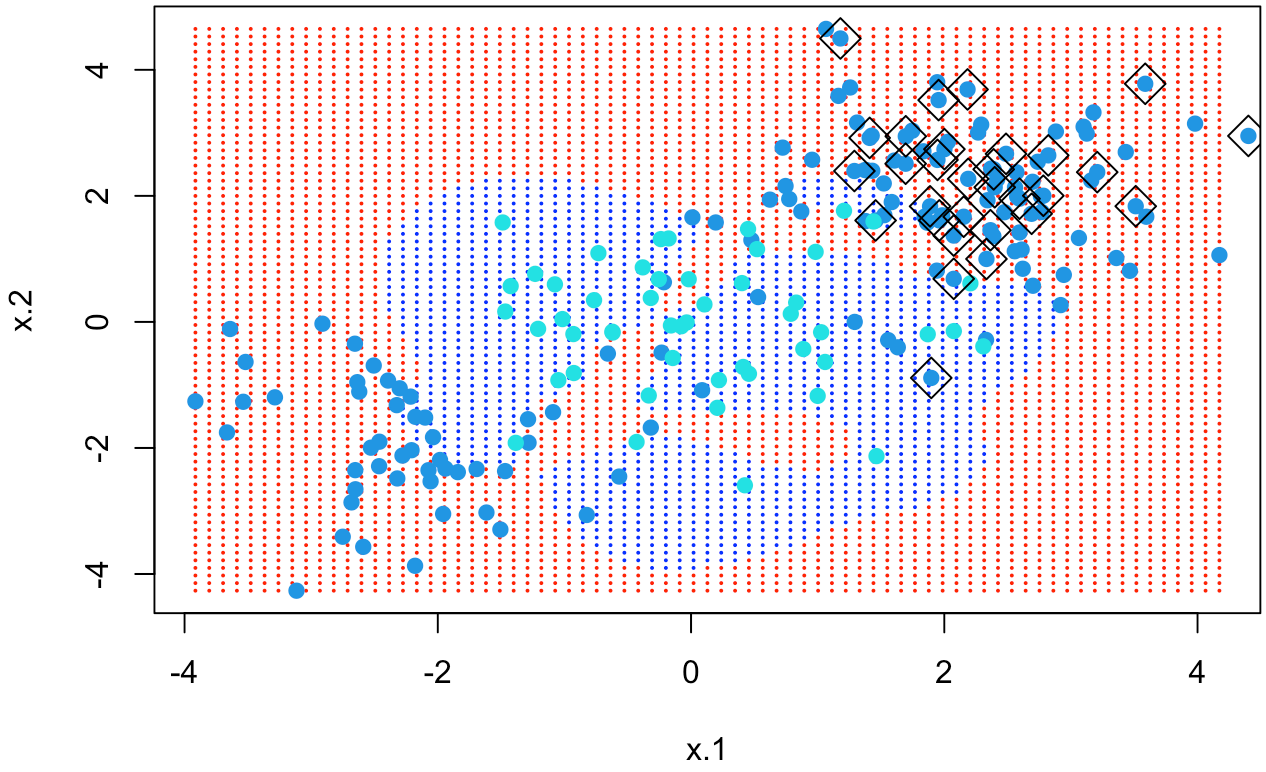
```
set.seed(1)

# Dividing the data
training_indices <- sample(1:nrow(dat), 100)
training_data <- dat[training_indices, ]
test_data <- dat[-training_indices, ]

# SVM
svmfit <- svm(y ~ ., data = training_data, kernel = "radial", gamma=1, cost=1, scale=FALSE)

# make grid function from earlier
make.grid <- function(x, n = 75) {
  grange <- apply(x, 2, range)
  x1 <- seq(from = grange[1,1], to = grange[2,1], length = n)
  x2 <- seq(from = grange[1,2], to = grange[2,2], length = n)
  expand.grid(x.1 = x1, x.2 = x2)
}

# 1:2 since that only selects x1 and x2, I dont think we're supposed to include y?
xgrid <- make.grid(training_data[, 1:2])

ygrid <- predict(svmfit, xgrid)

# Plotting the graph
plot(xgrid, col = c("red","blue")[as.numeric(ygrid)], pch = 20, cex = .2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index,], pch = 5, cex = 2)
```



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.

```
svmfit <- svm(y ~ ., data = training_data, kernel = "radial", gamma=1, cost=10000, scale=FALSE)

# 1:2 since that only selects x1 and x2, I dont think we're supposed to include y?
xgrid <- make.grid(training_data[, 1:2])

ygrid <- predict(svmfit, xgrid)

# Plotting the graph
plot(xgrid, col = c("red","blue")[as.numeric(ygrid)], pch = 20, cex = .2)
points(x, col = y + 3, pch = 19)
points(x[svmfit$index,], pch = 5, cex = 2)
```



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.

-> In this graph, we do see that we are able to capture the training data a lot better as opposed to the previous graph. However, this can become quite risky when trying to generalize this model. If we introduce new points to the data or try to use this model on a new dataset altogether, it is much more likely to misclassify the data points as it is so rigidly trained with the training data with such a high cost.

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[-training_indices,"y"], pred=predict(svmfit, newdata=dat[-training_indices,]))
```

```
##     pred
## true  1  2
##    1 62 17
##    2  3 18
```

-> There is a disparity in our classification results. We see that the model incorrectly classifies many of the Class 1 instances as Class 2 with an accuracy of 78% (17 out of the 79 true Class 1 labels were misclassified as Class 2). Its accuracy for Class 2 is a little better, where it correctly identified 18 out of 21 true Class 2 labels, which gives it an accuracy of 85%. Overall, the accuracy of the model is 80%, classifying exactly 80 entries correctly out of 100.

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.

```
training_class2_prop <- mean(training_data$y == 2)

full_class2_prop <- mean(dat$y == 2)

print(training_class2_prop)
```

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

```
print(full_class2_prop)
```

```
## [1] 0.25
```

-> The training data has a 0.29 proportion of Class 2 labels while the full data has a 0.25 proportion for Class 2. This small over representation in the training data is likely due to random variation. This difference does not seem to be the likely cause for the disparity that we see in the classification since the training data does seem to accurately represent the full data. There could be other factors at hand such as the features being used for classification as well as the parameters for SVM (the kernel, gamma value, and cost).

2.5

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` .

```
set.seed(1)

cost_values <- c(0.1, 1, 10, 100, 1000)
gamma_values <- c(0.5, 1, 2, 3, 4)

# Create a list of parameters to tune
tune_parameters <- list(
  cost = cost_values,
  gamma = gamma_values
)

tune.out <- tune(svm, y~., data = training_data, kernel="radial", ranges = tune_parameters, tunecontrol =  tune.control(c
ross = 10))

print(summary(tune.out))
```

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

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

2.6

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.

-> We have made significant improvements in the models accuracy, jumping up from 80% to 92% in overall accuracy. The accuracy for class one increased to 91% (72/79) and the accuracy for class two increased to 95% (20/21). The new parameters have improved performance. However, we do need to be a little bit careful and keep in mind overfitting. We want to be able to generalize the model so that it performs well on on scene data.

3

Let's turn now to decision trees.

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

head(heart)
```

```
##   age   sex cp trestbps chol   fbs restecg thalach exang oldpeak slope ca thal
## 1  63  TRUE  1      145  233  TRUE       2     150 FALSE     2.3     3  0    6
## 2  67  TRUE  4      160  286 FALSE       2     108  TRUE     1.5     2  3    3
## 3  67  TRUE  4      120  229 FALSE       2     129  TRUE     2.6     2  2    7
## 4  37  TRUE  3      130  250 FALSE       0     187 FALSE     3.5     3  0    3
## 5  41 FALSE  2      130  204 FALSE       2     172 FALSE     1.4     1  0    3
## 6  56  TRUE  2      120  236 FALSE       0     178 FALSE     0.8     1  0    3
##   class
## 1     0
## 2     2
## 3     1
## 4     0
## 5     0
## 6     0
```

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.

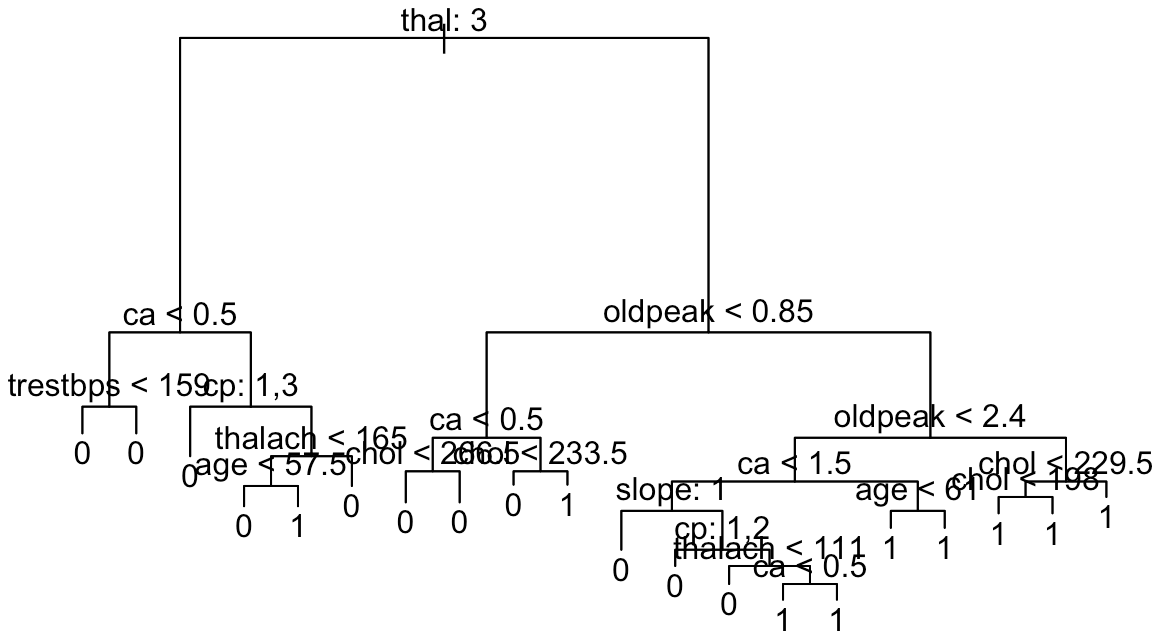
```
heart$disease_binary <- as.factor(ifelse(heart$class >= 2, 1, 0))

head(heart, 20)
```

```
##    age   sex cp trestbps chol   fbs restecg thalach exang oldpeak slope ca thal
## 1   63  TRUE  1      145  233  TRUE       2     150 FALSE     2.3     3  0    6
## 2   67  TRUE  4      160  286 FALSE       2     108  TRUE     1.5     2  3    3
## 3   67  TRUE  4      120  229 FALSE       2     129  TRUE     2.6     2  2    7
## 4   37  TRUE  3      130  250 FALSE       0     187 FALSE     3.5     3  0    3
## 5   41 FALSE  2      130  204 FALSE       2     172 FALSE     1.4     1  0    3
## 6   56  TRUE  2      120  236 FALSE       0     178 FALSE     0.8     1  0    3
## 7   62 FALSE  4      140  268 FALSE       2     160 FALSE     3.6     3  2    3
## 8   57 FALSE  4      120  354 FALSE       0     163  TRUE     0.6     1  0    3
## 9   63  TRUE  4      130  254 FALSE       2     147 FALSE     1.4     2  1    7
## 10  53  TRUE  4      140  203  TRUE       2     155  TRUE     3.1     3  0    7
## 11  57  TRUE  4      140  192 FALSE       0     148 FALSE     0.4     2  0    6
## 12  56 FALSE  2      140  294 FALSE       2     153 FALSE     1.3     2  0    3
## 13  56  TRUE  3      130  256  TRUE       2     142  TRUE     0.6     2  1    6
## 14  44  TRUE  2      120  263 FALSE       0     173 FALSE     0.0     1  0    7
## 15  52  TRUE  3      172  199  TRUE       0     162 FALSE     0.5     1  0    7
## 16  57  TRUE  3      150  168 FALSE       0     174 FALSE     1.6     1  0    3
## 17  48  TRUE  2      110  229 FALSE       0     168 FALSE     1.0     3  0    7
## 18  54  TRUE  4      140  239 FALSE       0     160 FALSE     1.2     1  0    3
## 19  48 FALSE  3      130  275 FALSE       0     139 FALSE     0.2     1  0    3
## 20  49  TRUE  2      130  266 FALSE       0     171 FALSE     0.6     1  0    3
##    class disease_binary
## 1      0              0
## 2      2              1
## 3      1              0
## 4      0              0
## 5      0              0
## 6      0              0
## 7      3              1
## 8      0              0
## 9      2              1
## 10     1              0
## 11     0              0
## 12     0              0
## 13     2              1
## 14     0              0
## 15     0              0
## 16     0              0
## 17     1              0
## 18     0              0
## 19     0              0
## 20     0              0
```

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)
train_indices <- sample(1:nrow(heart), 240)
heart_train <- heart[train_indices, ]
heart_test <- heart[-train_indices, ]

tree_model <- tree(disease_binary ~ .-class, data = heart_train)

plot(tree_model)
text(tree_model, 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(tree_model, heart_test, type = "class")
with(heart_test, table(tree_pred, disease_binary))
```

```
##          disease_binary
## tree_pred  0  1
##         0 34  5
##         1  9  9
```

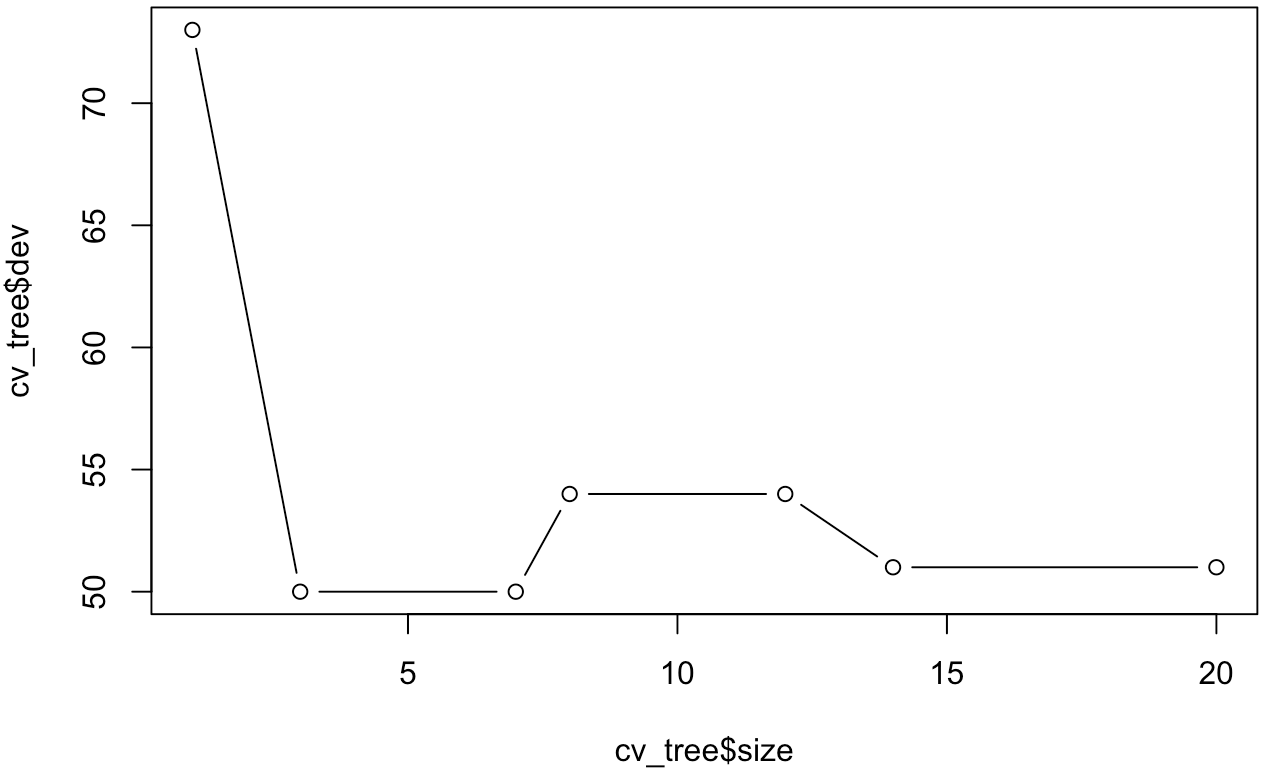
```
error_rate <- mean(tree_pred != heart_test$disease_binary)
print(error_rate)
```

```
## [1] 0.245614
```

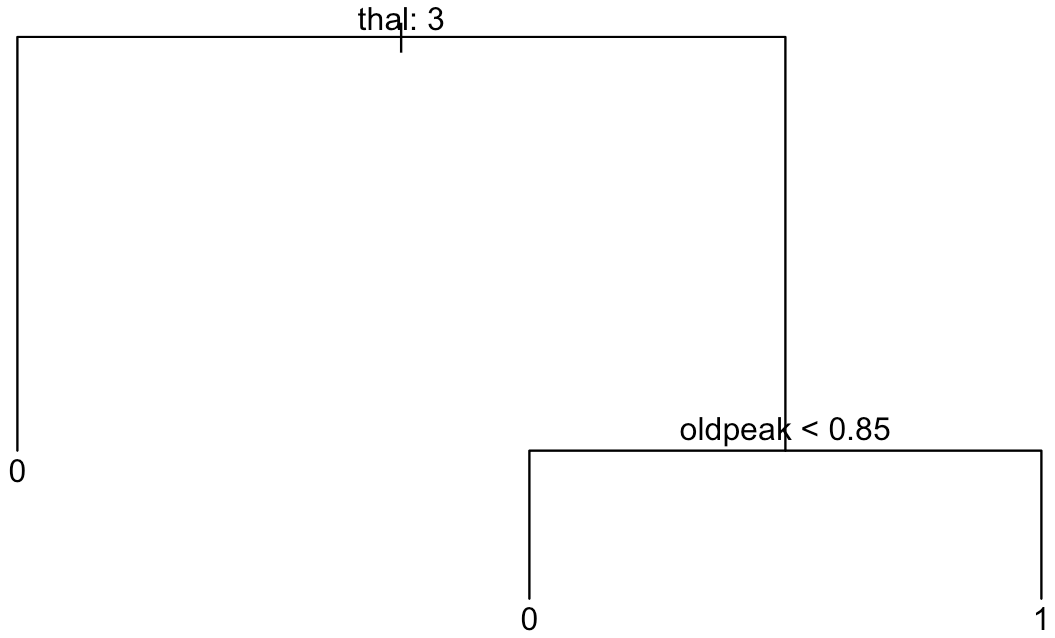
3.4

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_tree <- cv.tree(tree_model, FUN = prune.misclass)

plot(cv_tree$size, cv_tree$dev, type = "b")
```




```
pruned_tree <- prune.misclass(tree_model, best = 3)

plot(pruned_tree)
text(pruned_tree, pretty = 0)
```




```
tree_pred_pruned <- predict(pruned_tree, heart_test, type = "class")

with(heart_test, table(tree_pred_pruned, disease_binary))
```

```
##                disease_binary
## tree_pred_pruned  0  1
##                0 36  7
##                1  7  7
```

```
pruned_error_rate <- mean(tree_pred_pruned != heart_test$disease_binary)
print(pruned_error_rate)
```

```
## [1] 0.245614
```

3.5

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

-> The pruned tree has much lesser splits, making it very easy to read and increasing interpretability significantly. Despite this reduction in the number of splits, the accuracy remains about the same for this tree at 24.5%. This shows that even though lost a few levels of classification conditions, we can maintain the same accuracy with a bit more heterogeneity in the leaves.

3.6

Discuss the ways a decision tree could manifest algorithmic bias.

-> One of the biggest ways we can have algorithmic bias is by having biased training data. No matter how good the classification parameters get, if the training data itself is biased, then the results will be skewed towards similar results. Additionally, we can have some specific variables that have a high correlation with specific outputs without directly affecting the output. If this is the case in the data, then this specific variable might be used to create one of the initial splits (therefore signifying more importance) despite the fact that this variable did not influence the outcome variable at all. Lastly, looking for higher accuracy during training can lead to overfitting, which would make our model not generalizable for newer data.

---

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