HW 3

Lynn Check

9/24/2024

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).

$Var[X] = E[(X - E[X])^2]$

$Var[X] = E[X^2 - 2XE[X] + (E[X])^2]$ *Expanding the squared term

$Var[X] = E[X^2] - E[2E[X]] + E[(E[X])^2]]$

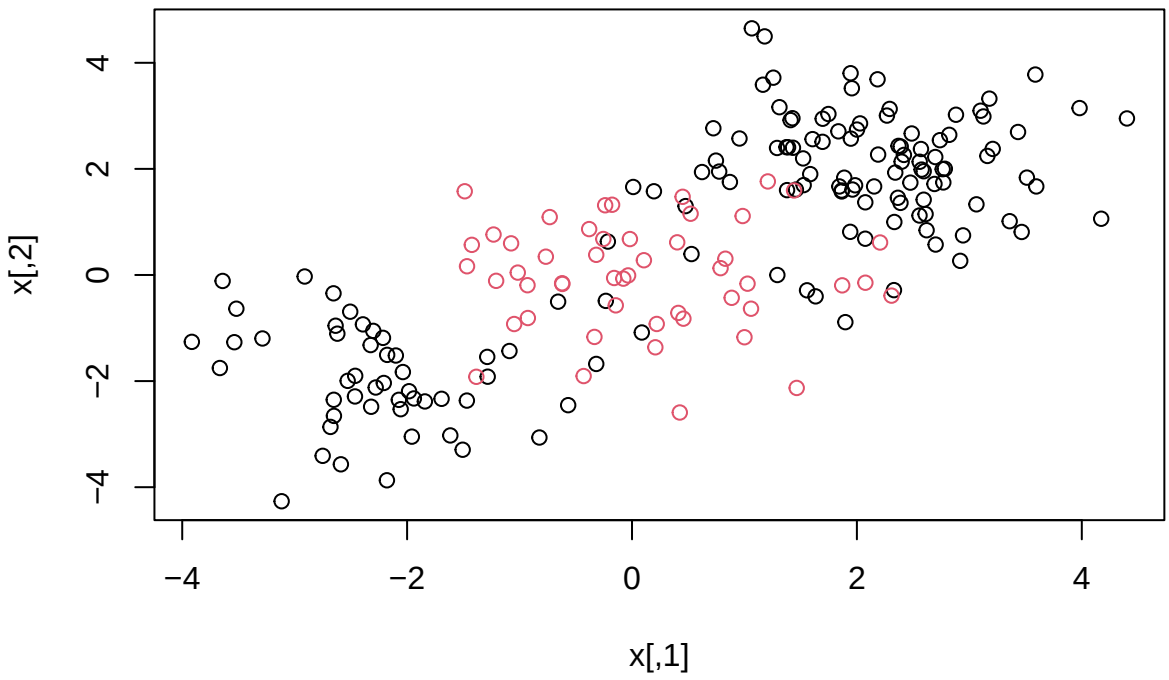
$Var[X] = E[X^2] - 2E[X]E[X] + E[(E[X])^2]]$ *Linearity of Expectation

$Var[X] = E[X^2] - 2(E[X])^2 + (E[X])^2$ *Simplified

$Var[X] = E[X^2] - (E[X])^2$ *Combined Like Terms

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)
```



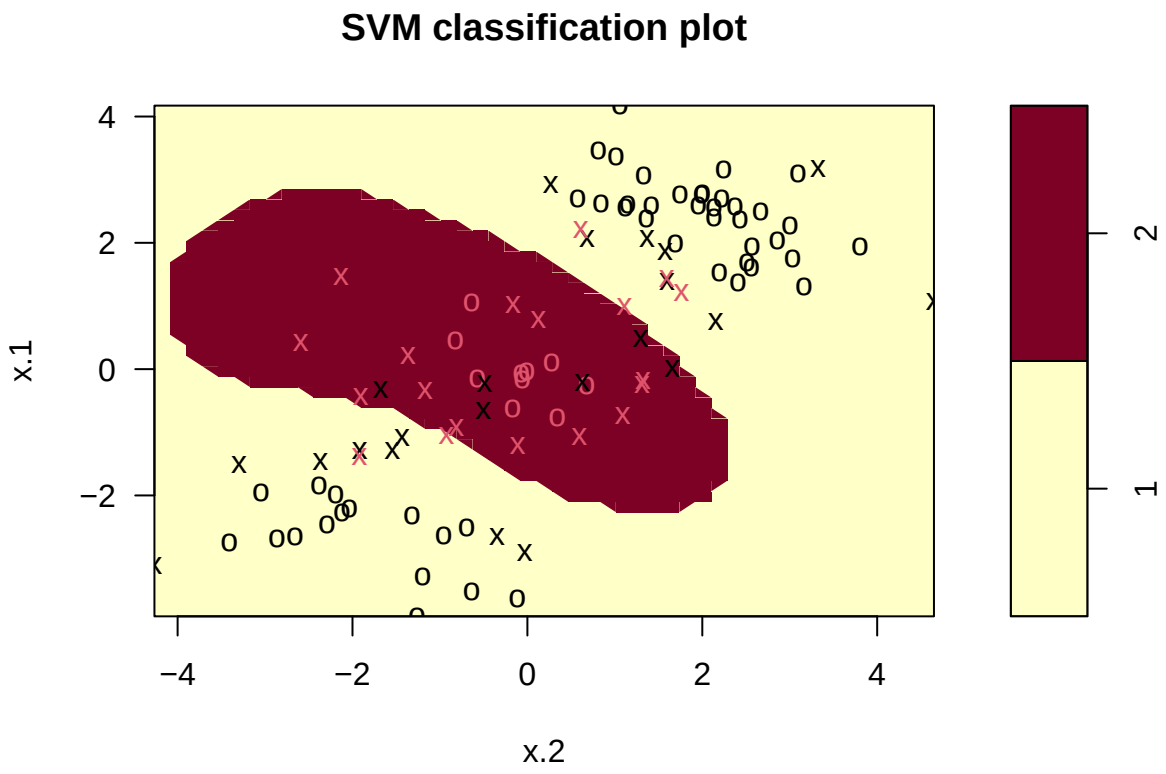
Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

```
set.seed(1)
training_indices = sample(200, 100)

# Creating Training Partition
# selecting rows from matrix x specified by training_indices including all columns
training = data.frame(x = x[training_indices,], # [x,] is calling on rows and column = x is a 2D object
                      y = as.factor(y[training_indices]) # y is a vector = 1D object hence we don't nee
                      )

# Creating Testing Partition
# selecting rows from matrix x that are NOT in training_indices including all columns
testing = data.frame(x = x[-training_indices,],
                     y = as.factor(y[-training_indices])
                     )

# Fitting an SVM on training data with radial kernal
svmfit = svm(y ~ ., data = dat[training_indices,], kernal = "radial", gamma = 1, cost = 1)
```

```r
# Printing SVM
print(svmfit)
```

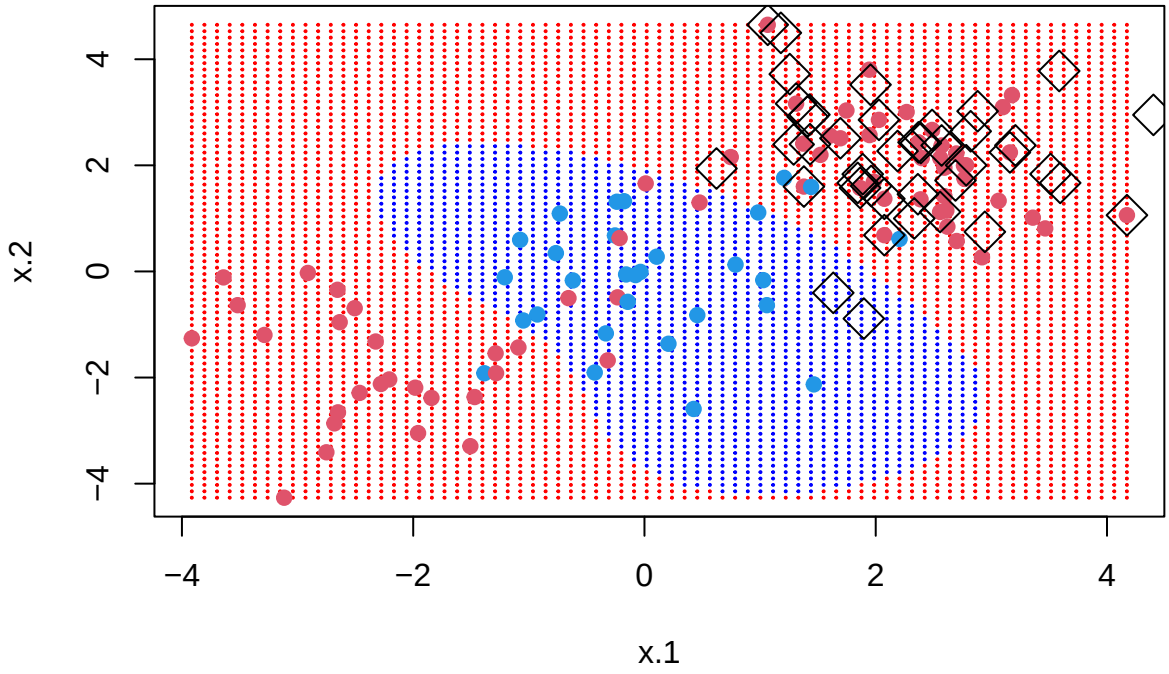
```
##
## Call:
## svm(formula = y ~ ., data = dat[training_indices, ], kernal = "radial",
##     gamma = 1, cost = 1)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##
## Number of Support Vectors:  41
```

```r
# Plotting SVM on training data
plot(svmfit, dat[training_indices,])
```




```r
#grid
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)
}
xgrid = make.grid(x[training_indices,])

#overlaying prediction on the grid
ygrid = predict(svmfit, xgrid)
#pch = 20 is smaller bullet style circle
plot(xgrid, col = c("red","blue")[as.numeric(ygrid)], pch = 20, cex = .2)
#pch = 19 is filled circle
points(x[training_indices,], col = y[training_indices]*2, pch = 19)
#diamonds around Support vectors
points(x[svmfit$index,], pch = 5, cex = 2)
```



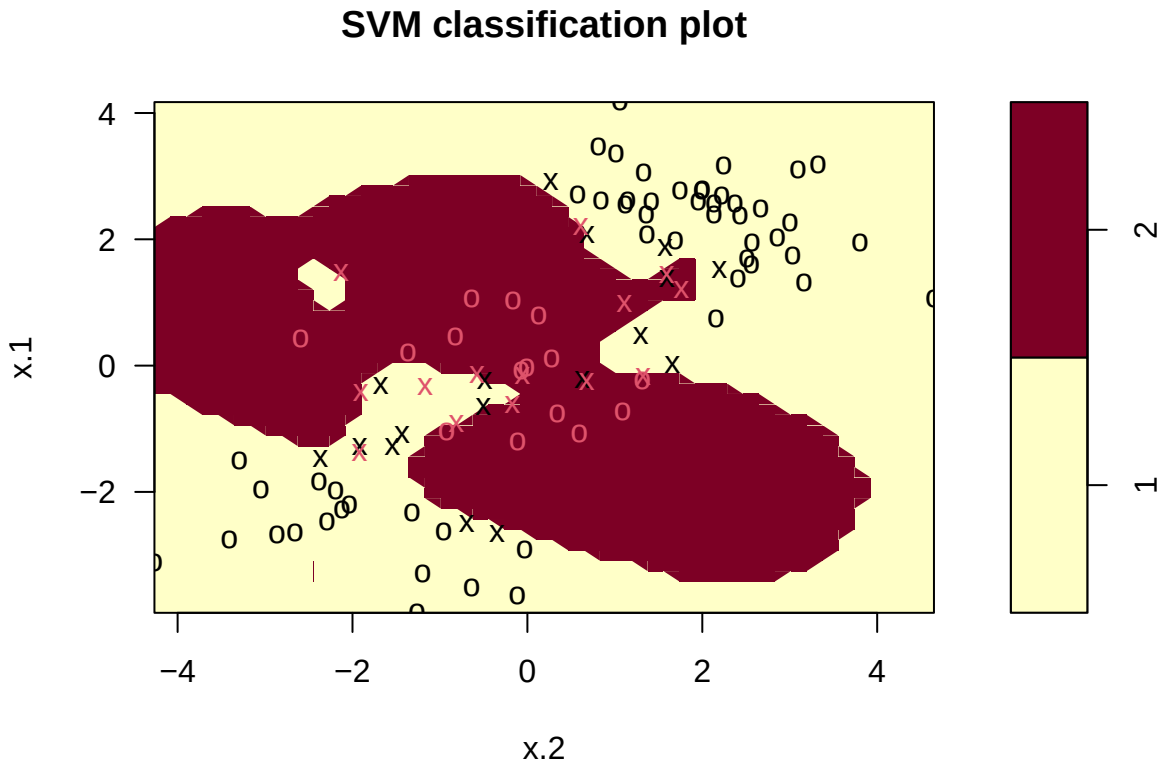
Notice that the above decision boundary is decidedly non-linear. It seems to perform reasonably well, but there are indeed some misclassifications. Let's see if increasing the cost [1] helps our classification error rate. Refit the svm with the radial kernel, $\gamma = 1$, and a cost of 10000. Plot this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

```
# Refitting SVM with cost = 10000
svmfit = svm(y ~ ., data = training, kernal = "radial", gamma = 1, cost = 10000)
print(svmfit)
```

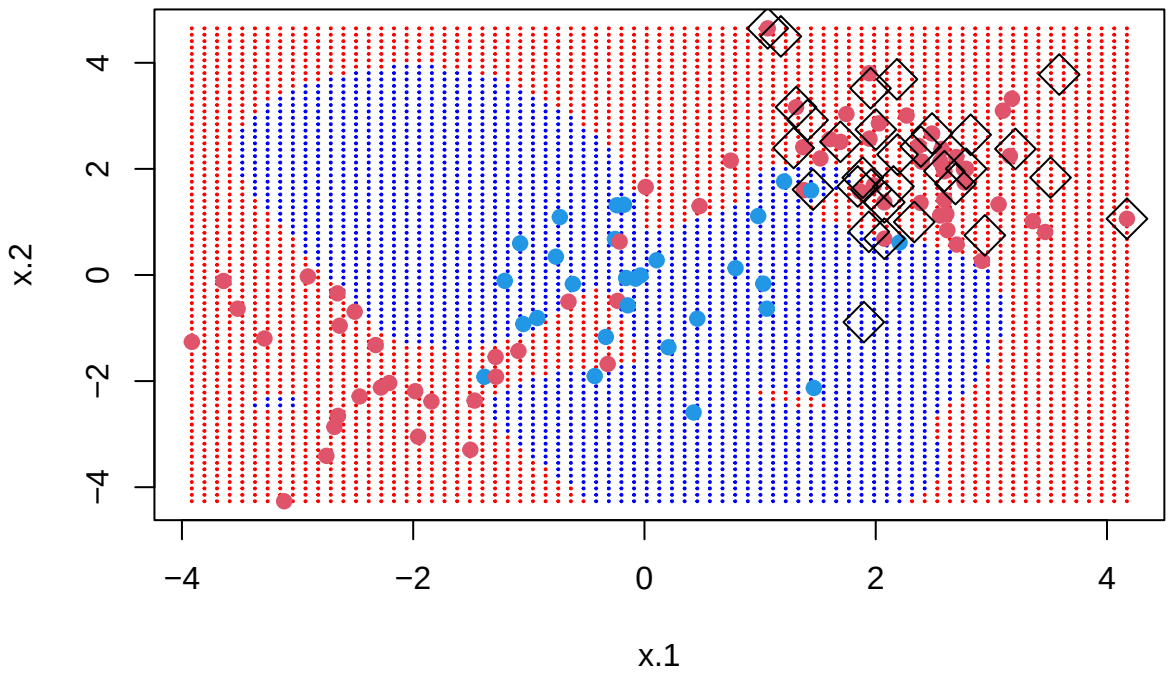
[1] Remember this is a parameter that decides how smooth your decision boundary should be

```
##
## Call:
## svm(formula = y ~ ., data = training, kernal = "radial", gamma = 1,
##     cost = 10000)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  10000
##
## Number of Support Vectors:  31
```

```
xgrid = make.grid(x[training_indices,])

plot(svmfit, dat[training_indices,])
```




```
#overlaying prediction on the grid
ygrid = predict(svmfit, xgrid)
#pch = 20 is smaller bullet style circle
plot(xgrid, col = c("red","blue")[as.numeric(ygrid)], pch = 20, cex = .2)
#pch = 19 is filled circle
points(x[training_indices,], col = y[training_indices]*2, pch = 19)
#diamonds around Support vectors
points(x[svmfit$index,], pch = 5, cex = 2)
```



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

Even though we are better capturing the training data, with such a model, there is the danger of overfitting. The model is overfitted and the danger of overfitting a model is decreasing the generalizability of the model. The model will likely be incapable to make accurate predictions on the testing set due to it being overly fitted to the noise that exist in the training set.

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 67 12
##    2  2 19
```

From the produced confusion matrix, there are 67 observations correctly predicted as Class 1 and 19 observations correctly predicted as Class 2. There are some disparities seen in the classifications presented in

the confusion matrix. There is a greater number of misclassified observations for Class 1 than there are for Class 2, which reveals that the model performs relatively well on Class 2. There is $((12/79) * 100) = 15.2\%$ of Class 1 incorrectly classified while Class 2 $((2/21) * 100) = 9.5\%$ were misclassified.

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.

```
sum(dat[training_indices,3] == 2)/100
```

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

```
table(as.factor(y[training_indices]))
```

```
##
##  1  2
## 71 29
```

Of the training set, 29% is Class 2 which is decently close to the underlying proportion of Class 2 in the data as a whole. While disparities can arise from a non-representative training set, this disparity seems to be less about the imbalance between the training and testing set and more representative of the overfitted decision boundary. An irregular decision boundary can cause even a representative training set to yield a bias model.

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)
tune.out <- tune(svm, y~., data = dat[training_indices,], kernel = "radial",
                 ranges = list(cost = c(0.1, 1, 10, 100, 1000), gamma = c(0.5,1,2,3,4)))
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
```

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.

From this confusion matrix presented, the misclassification rate for Class 1 has significantly decreased from 15.2% to $(100(7/79)) = 8.86\%$. Therefore, it is indicating the model improved from question 2 and it is not as overly fitted. The decision boundary was highly irregular before, so it is likely that in this scenerio, the decision boundary has improved and is less irregular. However, in both cases, Class 1 has a higher misclassification rate than Class 2, largely due to the imbalance present in the classes of the original data.

Let's turn now to decision trees.

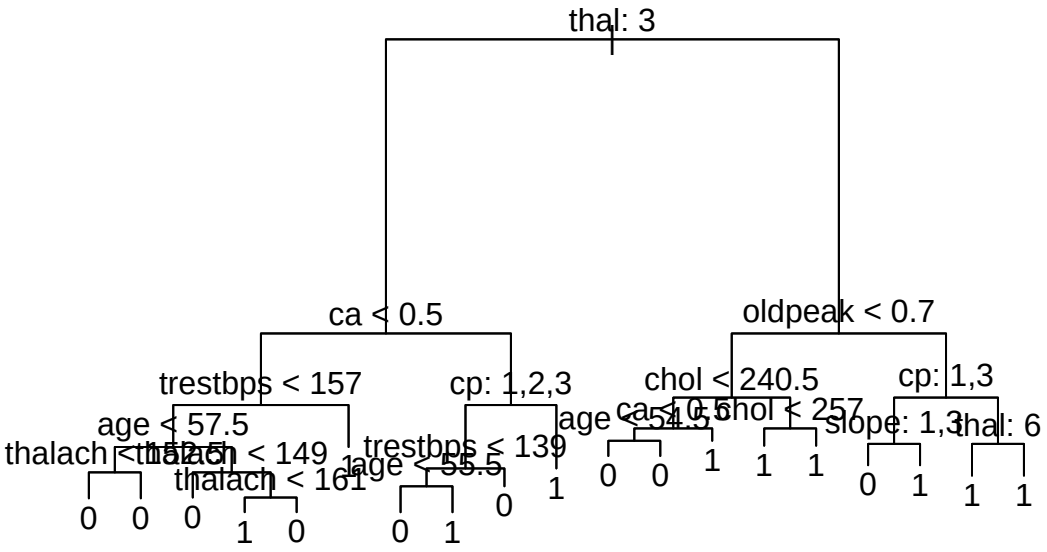
```
library(kmed)
data(heart)
library(tree)
```

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.

```
for (i in 1:length(heart$class)) {
  if (heart$class[i] > 0){
    heart$class[i] = 1
  }
}

heart$class = as.factor(heart$class)
```

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=sample(1:nrow(heart), 240)

tree.heart = tree(class~., heart, subset=train)
plot(tree.heart)
text(tree.heart, pretty=0)
```



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.heart, heart[-train,], type="class")
with(heart[-train,], table(tree.pred, class))
```

```
##          class
## tree.pred  0  1
##         0 28  3
##         1  8 18
```

```
1-(28+18)/57
```

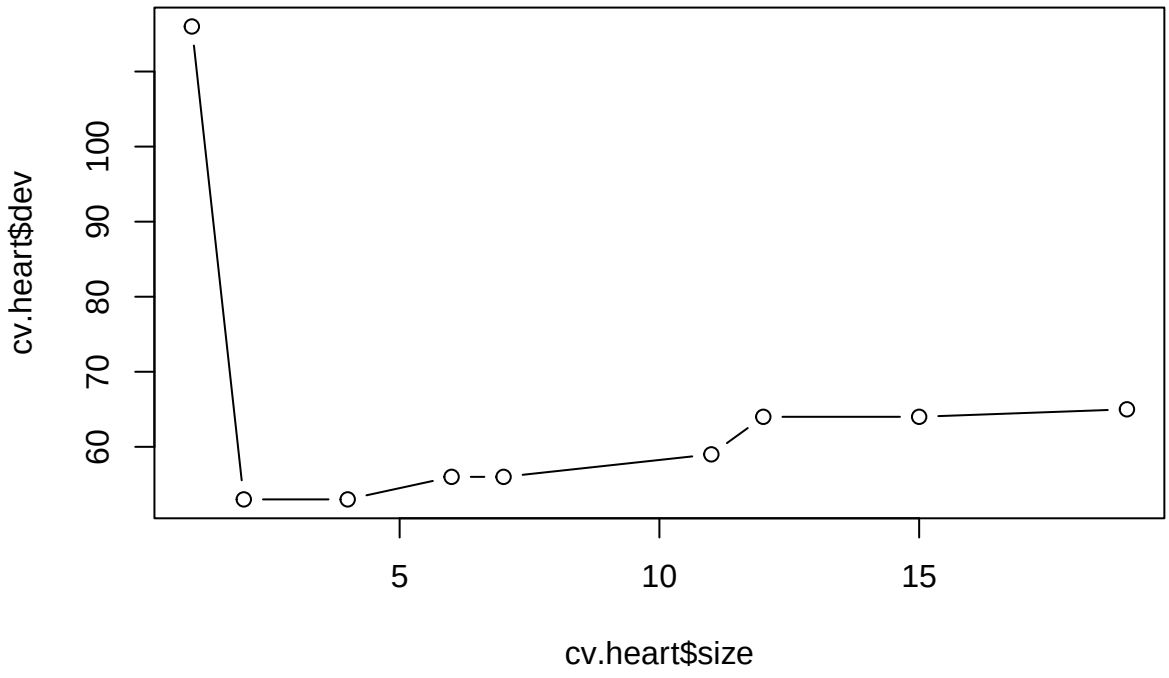
```
## [1] 0.1929825
```

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.

```r
set.seed(101)
cv.heart = cv.tree(tree.heart, FUN = prune.misclass)
cv.heart
```

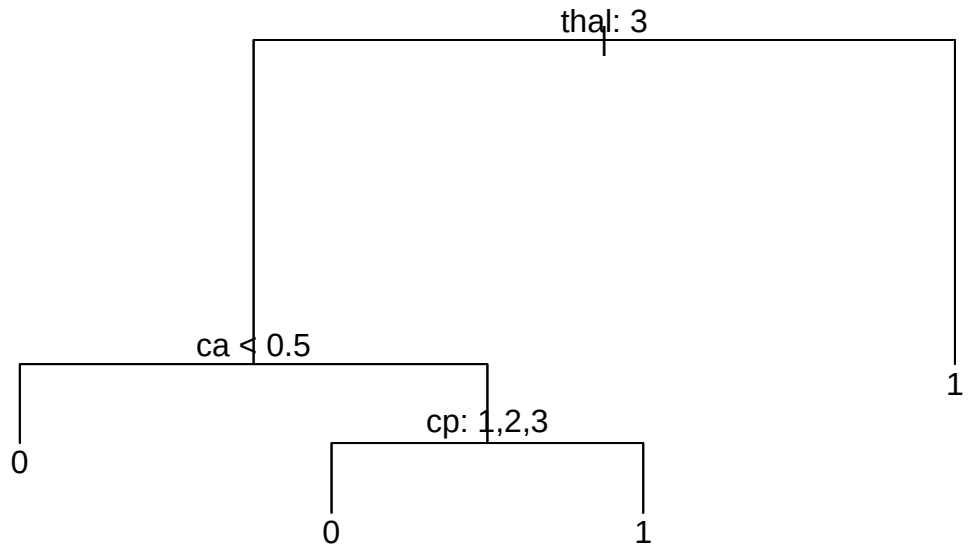
```
## $size
## [1] 19 15 12 11  7  6  4  2  1
##
## $dev
## [1]  65  64  64  59  56  56  53  53 116
##
## $k
## [1]       -Inf  0.0000000  0.6666667  1.0000000  1.5000000  2.0000000  3.5000000
## [8]  5.5000000 63.0000000
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```r
plot(cv.heart$size, cv.heart$dev, type = "b")
```




```r
prune.heart = prune.misclass(tree.heart, best =3)
```

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




```
tree.pred = predict(prune.heart, heart[-train,], type="class")
with(heart[-train,], table(tree.pred, class))
```

```
##          class
## tree.pred  0  1
##         0 26  4
##         1 10 17
```

```
1-((26+17)/57)
```

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

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

In pruning the tree above, we are sacrificing the classification accuracy by a marginal amount. By doing so, however, we have significantly acheived a more readable and interpretable decision tree with thal being the most important and influencial variable. The order of the remaining variables following thal are ca and cp.

Discuss the ways a decision tree could manifest algorithmic bias.

There are many ways that a decision tree could manifest algorithmic bias. For example, if the dataset that we are working with is not representative of the entire population, the the decision tree will learn patterns that'll favor the classes that are over represented. Another example is presence of selection bias such as removing relevant observations could cause the model to make predictions from inappropriate variables. As seen in this homework, imbalance of class distribution could lead to overfitting, manifesting algorithmic bias.