HW 3

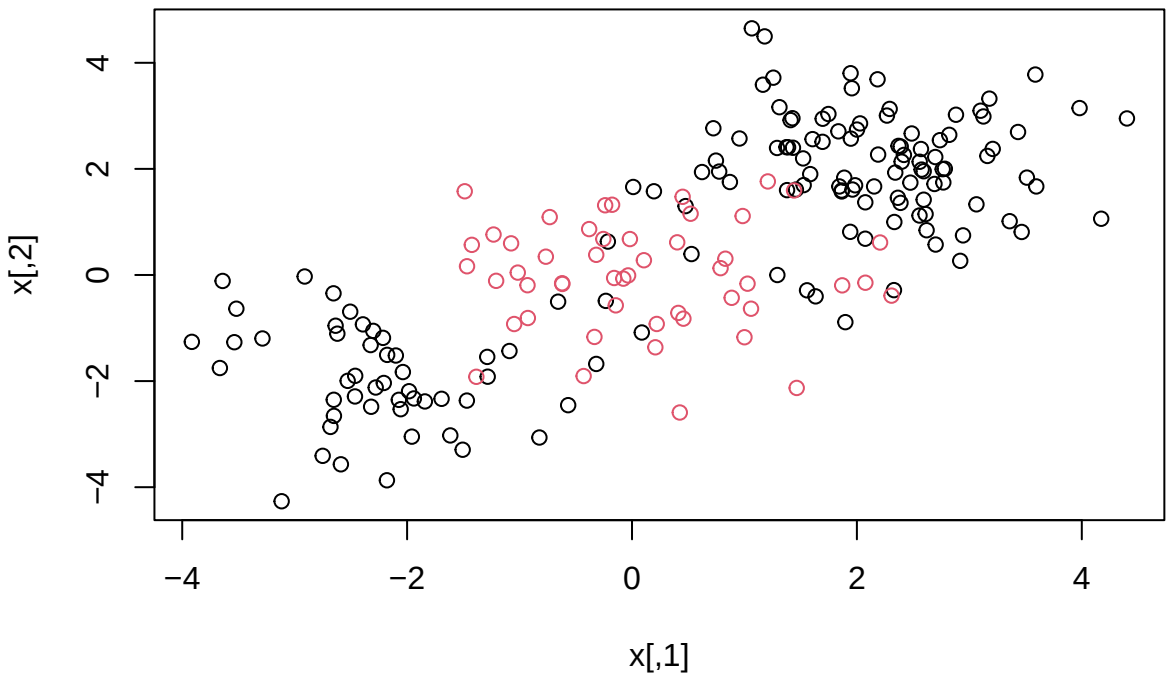
Keegan Burr

10/7/2024

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). ### My answer # First expand the squared term $X^2 - 2XE[X] + (E[X])^2$ # Add E outside all of it $E[X^2 - 2XE[X] + (E[X])^2]$ $E[X^2] - 2E[XE[X]] + E[(E[X])^2]$ # Simplify (Note: E[X] is a constant) $E[X^2] - 2(E[X])^2 + (E[X])^2$ $E[X^2] - (E[X])^2$ # So finally we would have $Var[X] := E[(X - E[X])^2] = E[X^2] - (E[X])^2$

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)
```

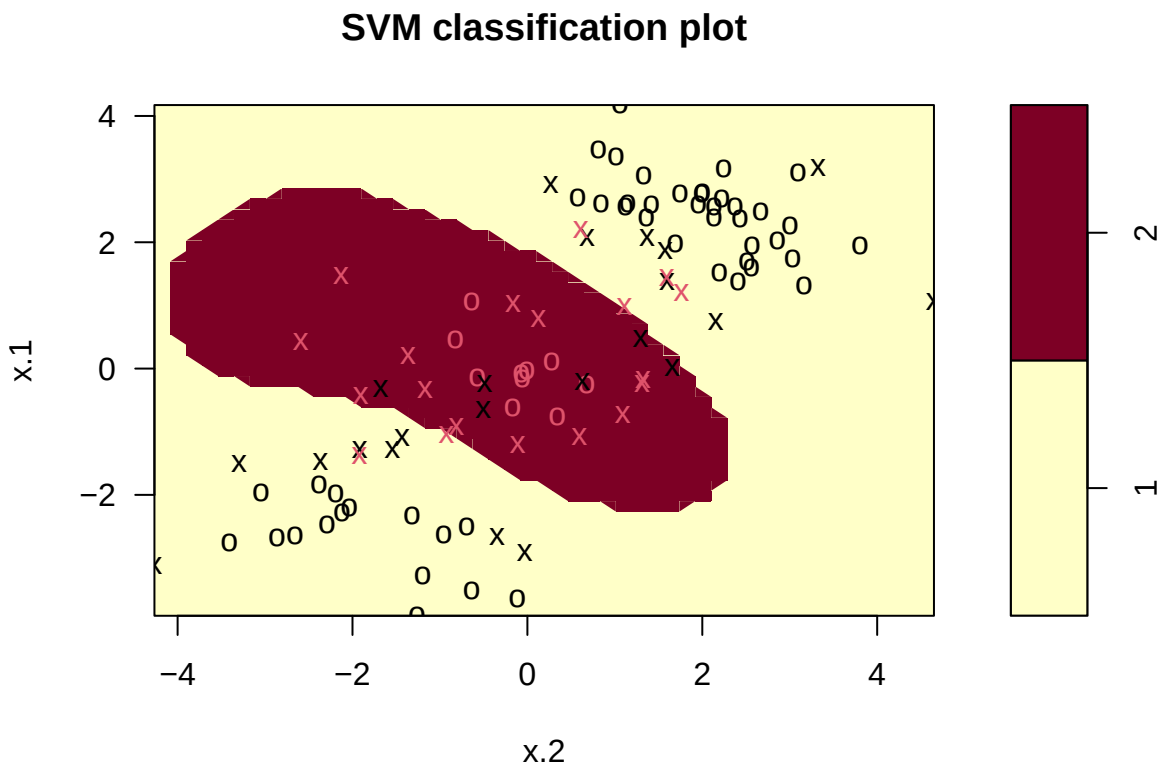


Quite clearly, the above data is not linearly separable. Create a training-testing partition with 100 random observations in the training partition. Fit an svm on this training data using the radial kernel, and tuning parameters $\gamma = 1$, cost $= 1$. Plot the svm on the training data.

```
set.seed(1)
train <- sample(200,100)
svmfit <- svm(y ~ ., data = dat[train,], kernel = "radial", gamma = 1, cost = 1)
print(svmfit)
```

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

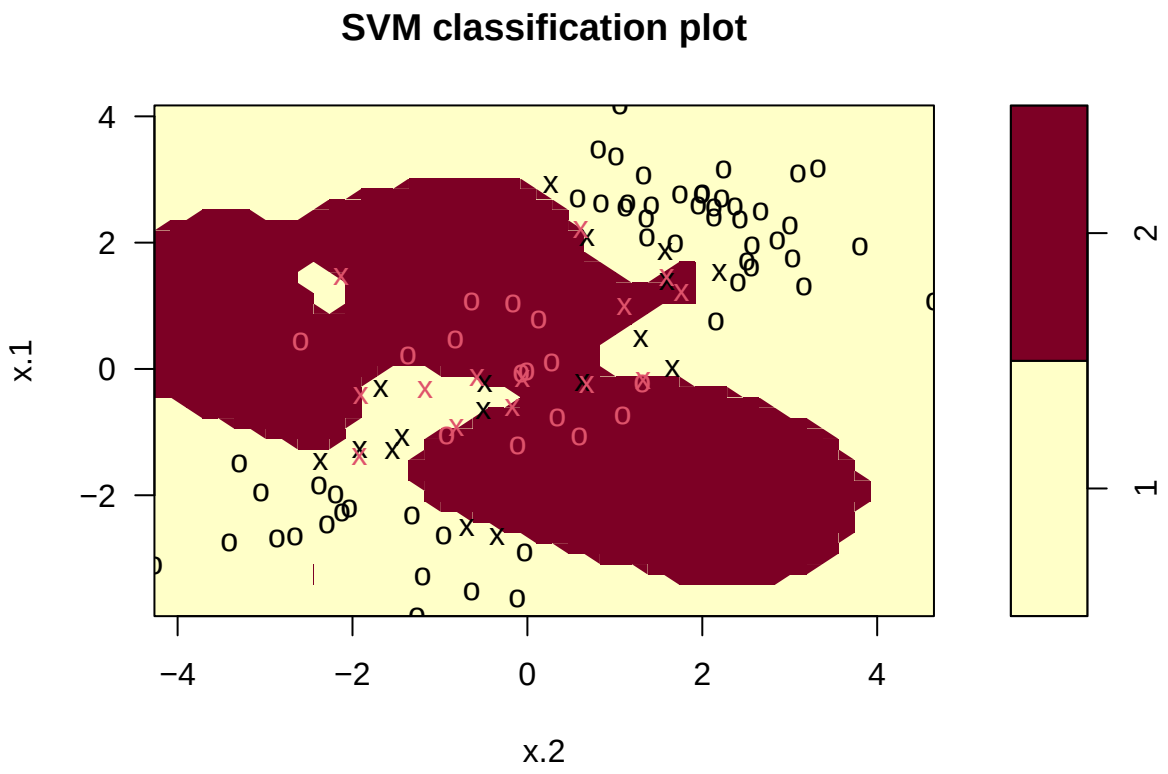
```
plot(svmfit, dat[train,])
```



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 = dat[train,], kernel = "radial", gamma = 1, cost = 10000)
plot(svmfit, dat[train,])
```

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



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

If we were to implement a high cost, it forces the model to attempt to correctly classify almost every training data point leading us to an overfit model that would capture small flucations in the data that maybe shouldn't. It is likely better to weaken the linearity constraint because there is a clear non-linear pattern, but weaken this constraint will lead to a trade-off particularly pertaining to overfitting as this would generalize the model to future data. The model may fit well to the current training data, but it's performance will likely decrease when it is implemented on new data.

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

```
##      pred
## true  1  2
##    1 67 12
##    2  2 19
```

In our confusion matrix above overall the model performs pretty well in classifying instances of 1, with 67 instances correct and only 12 misclassifcations. Similarly with instances of 2 the model correctly predicts 19 instances correct and only had 2 misclassifcations. I do worry slightly about the disparity in the classification rates as instance 1 has misclassification rate of ~15% and instance 2 of ~9%. Although not a huge difference, this misclassifcation could suggest the model is overfitting to the training data.

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.

```
# View table
dat[train,]
```

```
##             x.1          x.2 y
## 68   3.46555486  0.810886705 1
## 167 -0.25502703  0.678340177 2
## 129 -2.68166048 -2.864035954 1
## 162 -0.23864710  1.314002167 2
## 43   2.69696338  2.223480415 1
## 14  -0.21469989  0.629792122 1
## 187  1.46458731 -2.129360648 2
## 51   2.39810588  2.136221893 1
## 85   2.59394619  1.419385696 1
## 21   2.91897737  0.266781593 1
## 106 -0.23271273 -0.487787306 1
## 182  0.98389557  1.111431081 2
## 74   1.06590237  4.649166881 1
## 7    2.48742905  2.667066167 1
## 73   2.61072635  1.148142908 1
## 79   2.07434132  0.683754840 1
## 37   1.60571005  2.560820729 1
## 105 -2.65458464 -0.345854698 1
## 110 -0.31782392 -1.676993497 1
## 165 -0.61924305 -0.169318332 2
## 34   1.94619496  2.570507636 1
## 190 -0.92610950 -0.811170153 2
## 126 -1.28733369 -1.543864397 1
## 89   2.37001881  2.433702150 1
## 172  0.10580237  0.277914132 2
## 33   2.38767161  1.359518297 1
## 84   0.47643320  1.298768331 1
## 163  1.05848305 -0.635543001 2
## 70   4.17261167  1.060170673 1
## 188 -0.76608200  0.344845762 2
## 42   1.74663832  3.034107735 1
```

```
## 166  2.20610246  0.612218174 2
## 111 -2.63573645 -0.956387542 1
## 148 -1.98260438 -2.191278951 1
## 156 -1.07519230  0.596234109 2
## 20   2.59390132  1.955290863 1
## 44   2.55666320  1.121292387 1
## 121 -2.50595746 -0.692098480 1
## 87   3.06309984  1.331821393 1
## 176 -0.03472603 -0.008309014 2
## 173  0.45699881 -0.823081122 2
## 40   2.76317575  1.994655972 1
## 25   2.61982575  0.843427637 1
## 119 -1.50581167 -3.294140004 1
## 122 -0.65696117 -0.502958991 1
## 39   3.10002537  3.096777044 1
## 170  0.20753834 -1.363291256 2
## 134 -3.51839408 -0.635565071 1
## 24   0.01064830  1.659031420 1
## 195 -1.20808279 -0.110158762 2
## 130 -2.32427027 -1.320769226 1
## 45   1.31124431  3.162964556 1
## 146 -2.75081900 -3.408850456 1
## 22   2.78213630  2.002131860 1
## 115 -2.20738074 -2.035922423 1
## 104 -1.84197123 -2.383632106 1
## 161  0.42510038 -2.592327670 2
## 144 -2.46353040 -2.289499367 1
## 145 -3.11592011 -4.264889356 1
## 103 -2.91092165 -0.028662614 1
## 75   0.74636660  2.156011676 1
## 13   1.37875942  1.599753256 1
## 159 -1.38442685 -1.918909820 2
## 177  0.78763961  0.128855402 2
## 23   2.07456498  1.369699666 1
## 189 -0.43021175 -1.904955446 2
## 174 -0.07715294 -0.068840934 2
## 141 -3.91435943 -1.260410774 1
## 29   1.52184994  2.197193439 1
## 108 -1.08982577 -1.432779085 1
## 48   2.76853292  1.744329291 1
## 175 -0.33400084 -1.167662326 2
## 149 -3.28630053 -1.196716784 1
## 191 -0.17710396  1.324004321 2
## 31   3.35867955  1.014173300 1
## 102 -1.95788413 -3.047298149 1
## 17   1.98380974  1.691259431 1
## 186 -0.15875460 -0.056521425 2
## 133 -1.46850381 -2.367450756 1
## 197  1.44115771  1.592913754 2
## 83   3.17808700  3.324258630 1
## 118 -2.27911330 -2.121010111 1
## 114 -2.65069635 -2.655781852 1
## 90   2.26709879  3.005159218 1
## 150 -3.64060553 -0.112525537 1
```

```
## 107 -1.28329252 -1.917034266 1
## 64   2.02800216  2.857409778 1
## 94   2.70021365  0.573742658 1
## 179  1.02739244 -0.163910957 2
## 96   2.55848643  2.134447661 1
## 169 -0.14439960 -0.572542604 2
## 60   1.86494540  1.574732278 1
## 193 -0.73174817  1.091668956 2
## 93   3.16040262  2.244164924 1
## 180  1.20790840  1.763552003 2
## 10   1.69461161  2.510108423 1
## 1    1.37354619  2.409401840 1
## 196 -1.04798441 -0.924312773 2
## 59   2.56971963  2.374724407 1
## 26   1.94387126  3.803141908 1
```

```
# Determine in the 'y' column how many have class 2
sum(dat[train, "y"] == 2)
```

```
## [1] 29
```

29/100 is obviously 29%, which is pretty close to the underlying percentage. This leads me to believe that the disparity is likley not because of an imbalance between the training/testing data but likely because of overfitting.

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[train,], kernel = "radial", ranges = list(cost = c(0.1, 1, 10, 10
```

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

```
##     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.

So now it seems like we are not overfitting as badly and our model preforms a little better. Our misclassifications for both instances 1 and 2 reduced (for 1 it reduced from 12 to 7 and for 2 it reduced from 2 to 1), but still with our data imbalance, instances 1 are still misclassified more often than instances of 2.

Let's turn now to decision trees.

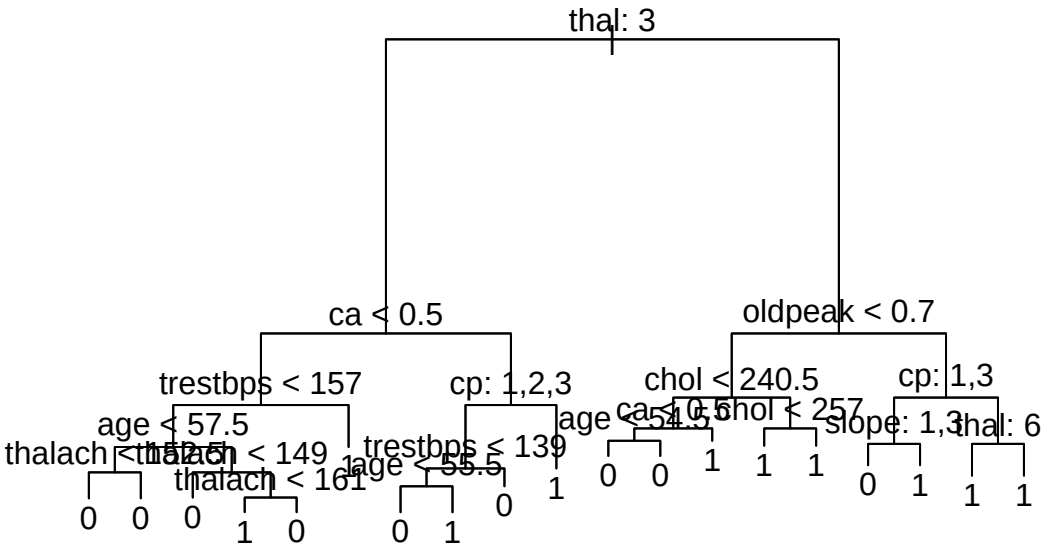
```
library(kmed)
data(heart)
library(tree)
```

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:nrow(heart)) {
  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_heart <- sample(1:nrow(heart), 240)
tree.heart <- tree(class~., heart, subset=train_heart)
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_heart,], type="class")
with(heart[-train_heart,], table(tree.pred, class))
```

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

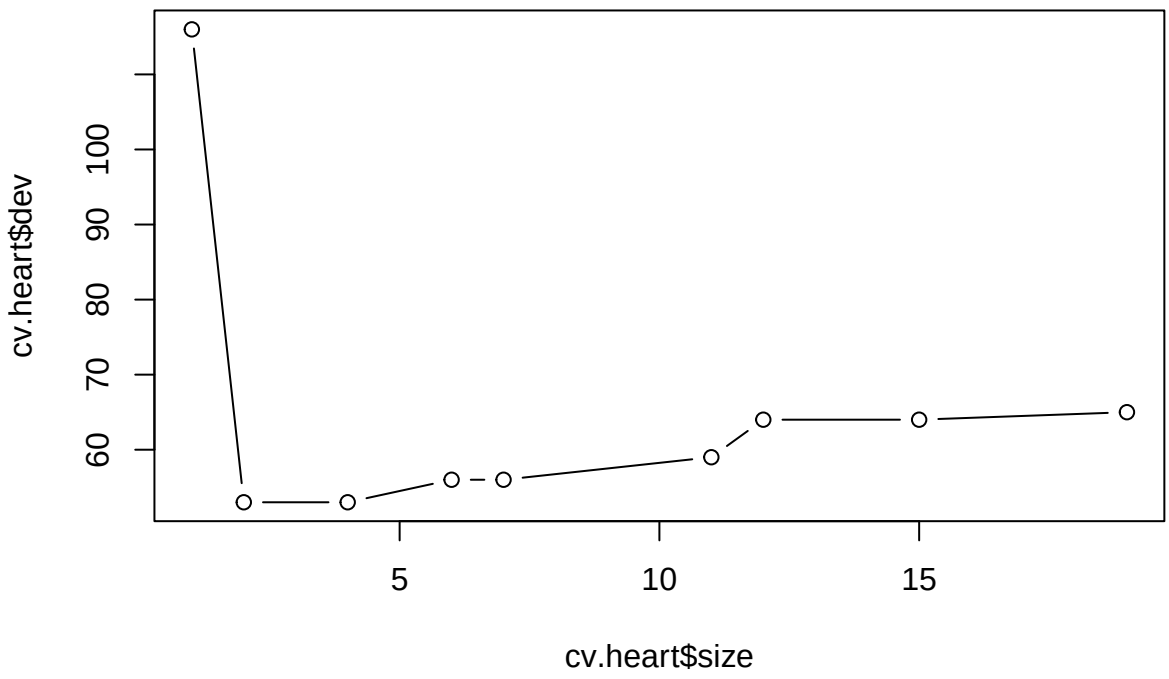
```
# So then using results
# 1-(28+18)/57 = ~.193
```

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.

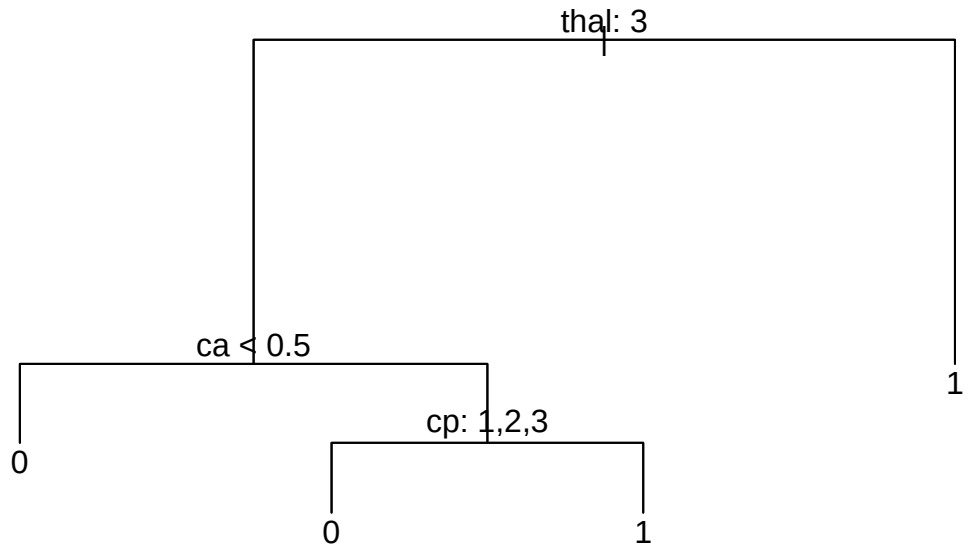
```r
set.seed(101)

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
# Take some fully grown bushy tree and prune it to the user specified number of splits (in this case 3
prune.heart = prune.misclass(tree.heart, best = 3)
plot(prune.heart)
text(prune.heart, pretty=0)
```




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

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

```r
# So then using results
# 1-((26+17)/57) = ~.246
```

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

By pruning the tree we are basically "collasping" branches together to make a less complex tree, but by doing this we often times give up some accuracy (in this case it was about 5%) in the classification rate for easier interpretability of the tree.

Discuss the ways a decision tree could manifest algorithmic bias.

A couple ways a decision tree could manifest algorithic bias are bias in the training/testing data and class imbalance (as we saw earlier in the homework). This could lead us to overfitting to biased data which would lead to poor/inaccurate results