Introduction to Decision Trees

Matt Richey

4/9/2020

Introduction

Let's take a tour of decision trees, first with regression and then with classification.

Regression Trees

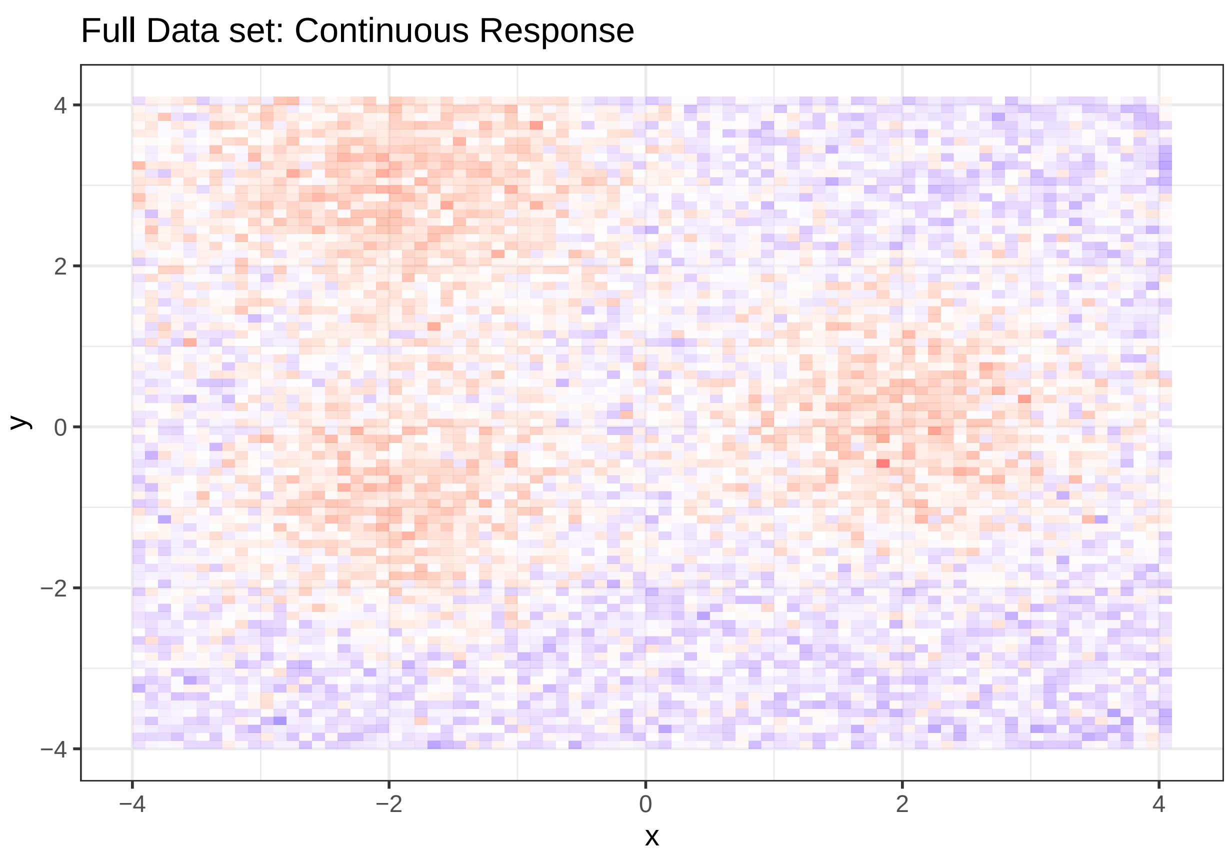
We'll start by looking at a simple synthetic example built on the structure below. The data set contains a continuous response variable (z) and two predictors (x and y).

```
dataDir <- "/Users/richeym/Dropbox/COURSES/ADM/ADM_S20/CODE/Chap08"
file <- "RectData_Quant.csv"
regData.df <- read.csv(file.path(dataDir,file))
dim(regData.df)
```

```
## [1] 6561    3
```

What does this data look like?

```
xvals <- with(regData.df,unique(x))
del <- xvals[2]-xvals[1]
mpt <- with(regData.df,mean(z))
data.gg <-
  regData.df %>%
  ggplot()+
  geom_rect(aes(xmin=x,xmax=x+del,ymin=y,ymax=y+del,fill=z),alpha=.5)+
  scale_fill_gradient2(low="blue",mid="white",high="red",midpoint=mpt)+
  ##scale_x_continuous(breaks=0:10)+
  ##scale_y_continuous(breaks=0:10)+
  labs(title= "Full Data set: Continuous Response",
       x="x",
       y="y")+
  guides(fill=FALSE)+
  theme_bw()
data.gg
```



Now we use a regression tree to model this data using x and y as the predictors and z (continuous) as the response.

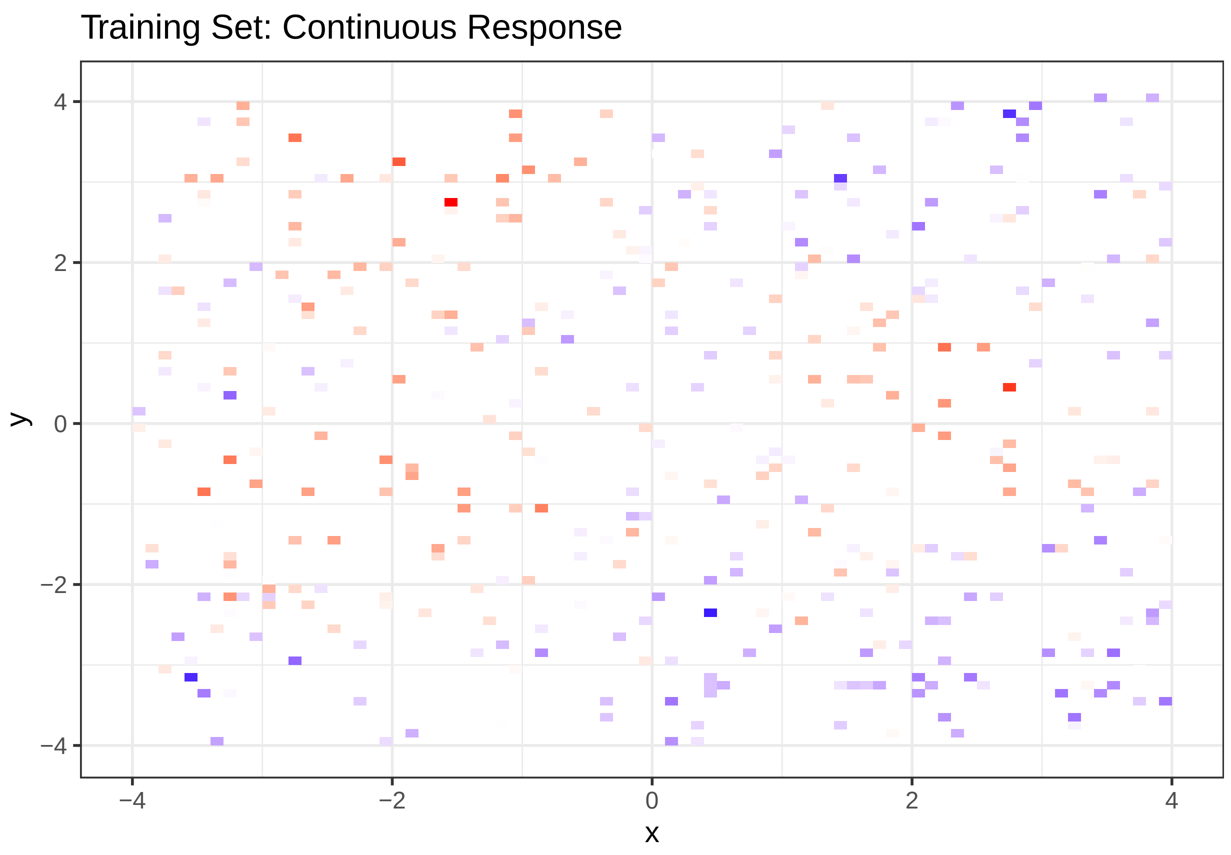
To start, build some test and training data by subsetting the large

```
N <- nrow(regData.df)
n <- 400
train <- sort(sample(1:N,n,rep=F))
test <- sample(1:N,n,rep=F)
##get rid of overlaps
both <- intersect(train,test)
train <- setdiff(train,both)
test <- setdiff(test,both)
##ok...
train.df <- regData.df[train,]
test.df <- regData.df[test,]
```

What do the training data look like?

```
train.gg <-
  train.df %>%
  ggplot()+
  geom_rect(aes(xmin=x,xmax=x+del,ymin=y,ymax=y+del,fill=z))+
  scale_fill_gradient2(low="blue",mid="white",high="red",midpoint=mpt)+
  ##scale_x_continuous(breaks=0:10)+
  ##scale_y_continuous(breaks=0:10)+
```

```r
  labs(title= "Training Set: Continuous Response",
       x="x",
       y="y")+
  guides(fill=FALSE)+
  theme_bw()
train.gg
```



The goal is to build a model using a regression tree that can predict the response value at any point.

Here comes our first tree model. Note the "control" option. This is where we specify the depth of the tree. There are three variables to consider: mindev, minsize, and mincut. Generally speaking, the smaller each of these is, the larger the tree becomes.

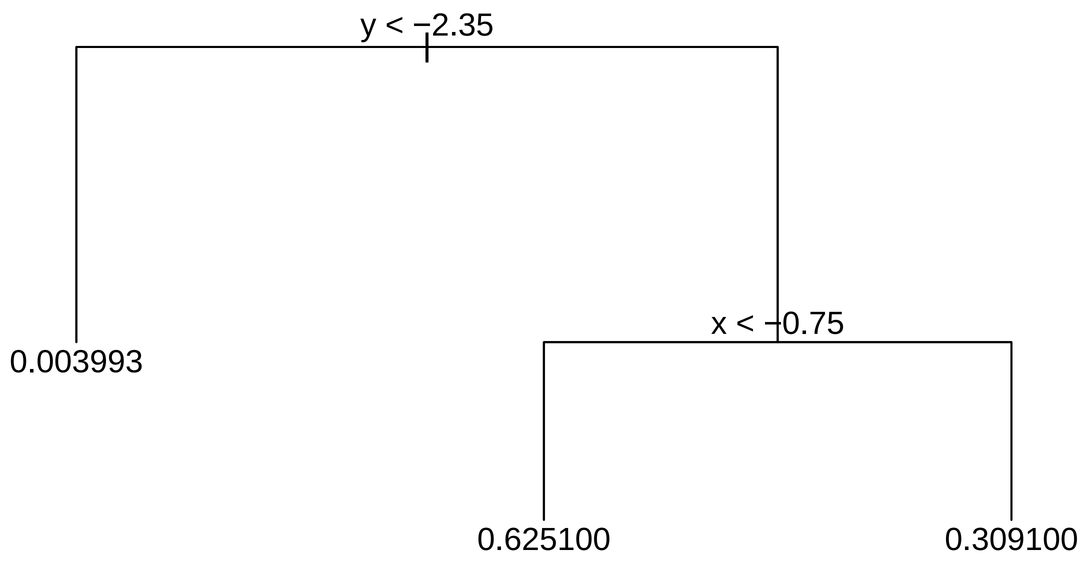
In particular, "mindev" controls the decrease in deviance from one node to its children. A mindev of, say, 0.01 means that we will stop growing the tree at a node if if the decrease in deviance is less than 0.01. The parameter "minsize" refers to the minimum size of node (in observations).

```r
mod.tree <- tree(z ~ x + y,
                 data=train.df,
                 control=tree.control(nrow(train.df),
                                      mindev=0.05,
                                      minsize=10))
```

Here's a way to look at the tree. It's not great, but it works.

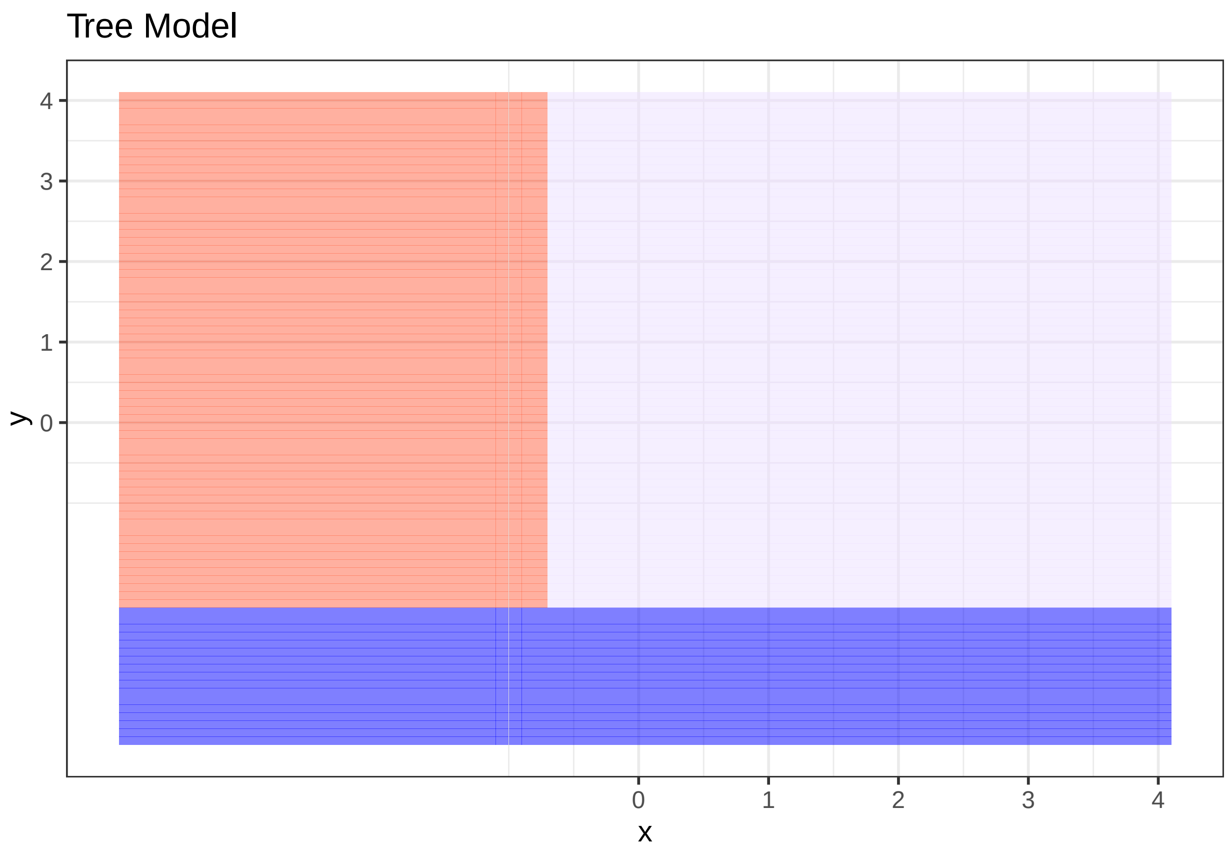
Note: the braces are an annoying workaround to solve a plotting error in RMarkdown.

```
{plot(mod.tree);text(mod.tree)}
```



Let's look at this on the original grid.

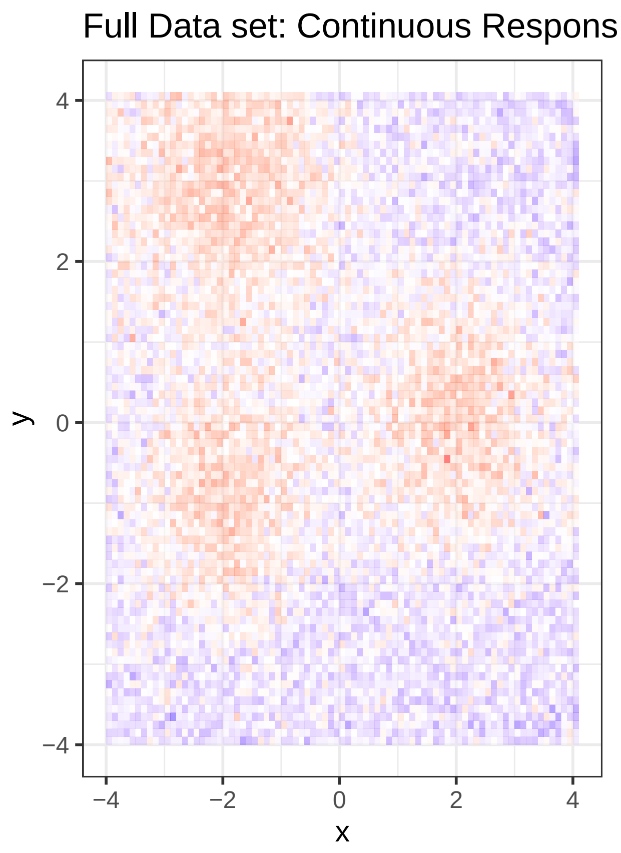
```
regData.df$pred <- predict(mod.tree,newdata=regData.df)
treeModel.gg  <-
  regData.df %>%
  ggplot()+
  geom_rect(aes(xmin=x,xmax=x+del,ymin=y,ymax=y+del,fill=pred),alpha=.5)+
  scale_fill_gradient2(low="blue",mid="white",high="red",midpoint = mpt)+
  scale_x_continuous(breaks=0:10)+
  scale_y_continuous(breaks=0:10)+
  labs(title= "Tree Model",
       x="x",
       y="y")+
  guides(fill=FALSE)+
  theme_bw()
treeModel.gg
```

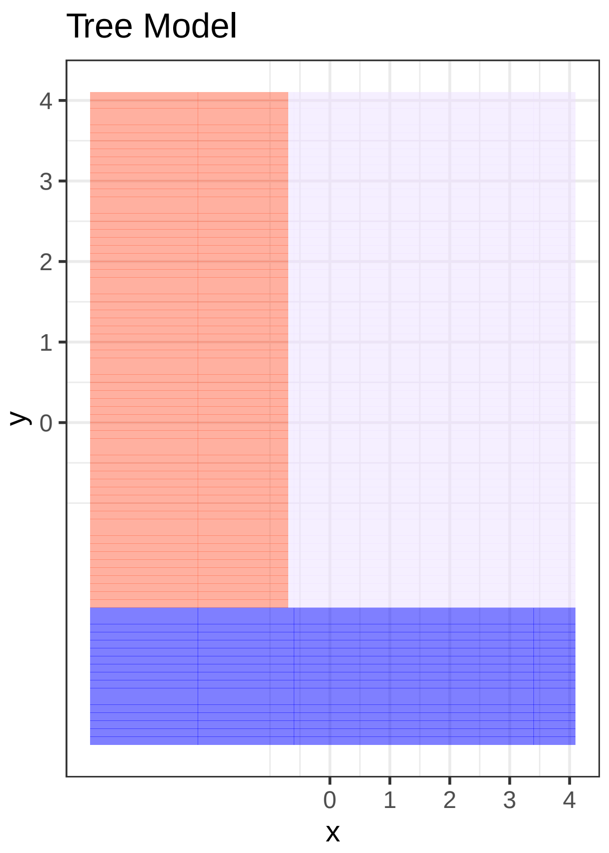


Make sure you can see how this picture compares to the tree structure.

How does this compare with the underlying structure the created the data?

```r
library(gridExtra) ## for grid.arrange
grid.arrange(data.gg + guides(fill=FALSE),
             treeModel.gg,nrow=1)
```

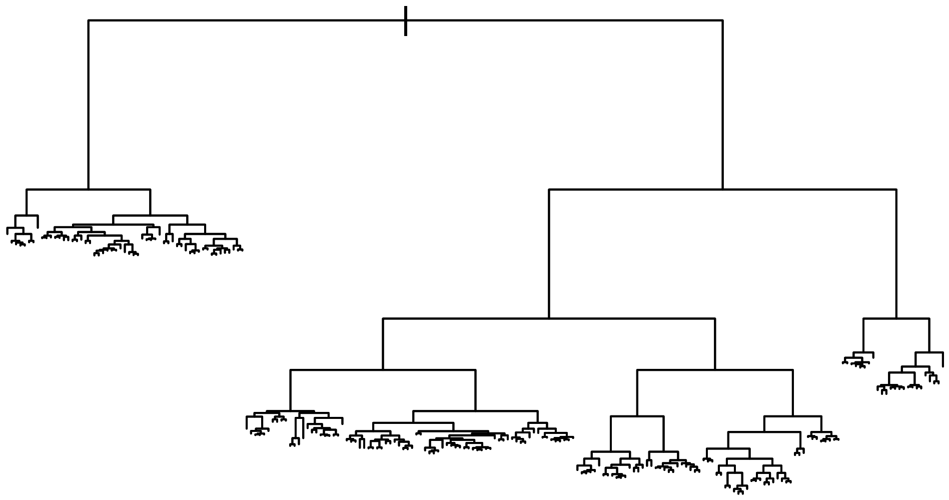




It's trying. . . Go head and repeat this with a different value of mindev.

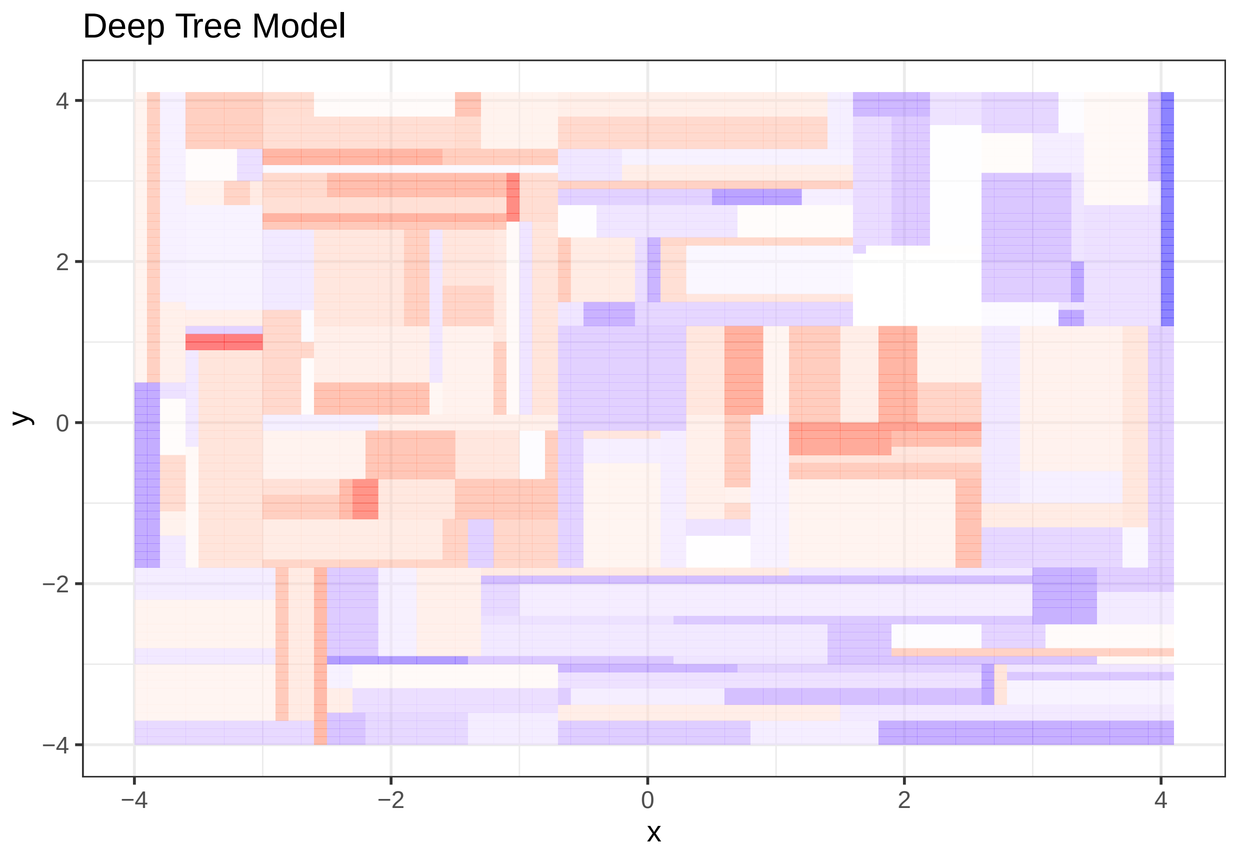
Here's a deep tree built with a very small mindev.

```
mod.tree <- tree(z ~ x + y,
                 data=train.df,
                 control=tree.control(nrow(train.df),
                                      ## small mindev, large tree
                                      mindev=0.0005,
                                      ## keep this small too
                                      minsize=2))
plot(mod.tree)
```



That's a lot of tree! What does the resulting model look like in the plane?

```
regData.df$pred <- predict(mod.tree,newdata=regData.df)
regData.df %>%
  ggplot()+
  geom_rect(aes(xmin=x,xmax=x+del,ymin=y,ymax=y+del,fill=pred),alpha=.5)+
  scale_fill_gradient2(low="blue",mid="white",high="red",midpoint=mpt)+
  labs(title= "Deep Tree Model",
       x="x",
       y="y")+
  guides(fill=FALSE)+
  theme_bw()
```



Looks like an overfitted model.

Cross-validating and optimal trees

Of course, we just guessed at a tree size. Here's an attempt (and not a bad one) at cross-validating to find the proper tree size. The key is that R has a function that will "prune" a tree back to a particular size. In this case, size is determined by the number of leaves.

Plan: Start with building a really deep tree using the training data. Go through the subtrees of all possible sizes and evaluate each on the test data. Pick the best one based on the MSE.

We can get the tree information from the tree's frame

```
names(mod.tree)
```

```
## [1] "frame"   "where"   "terms"   "call"    "y"       "weights"
```

```
## Extract  the frame
tree.frame <- mod.tree$frame
## Here's how to get the node count.
(numLeaves <- sum(tree.frame$var=="<leaf>"))
```

```
## [1] 216
```

We have a tree with 216 leaves. This is a large number of leaves and almost surely overfits the data. To correct, we will "prune" the tree looking for a smaller one that doesn't overfit.

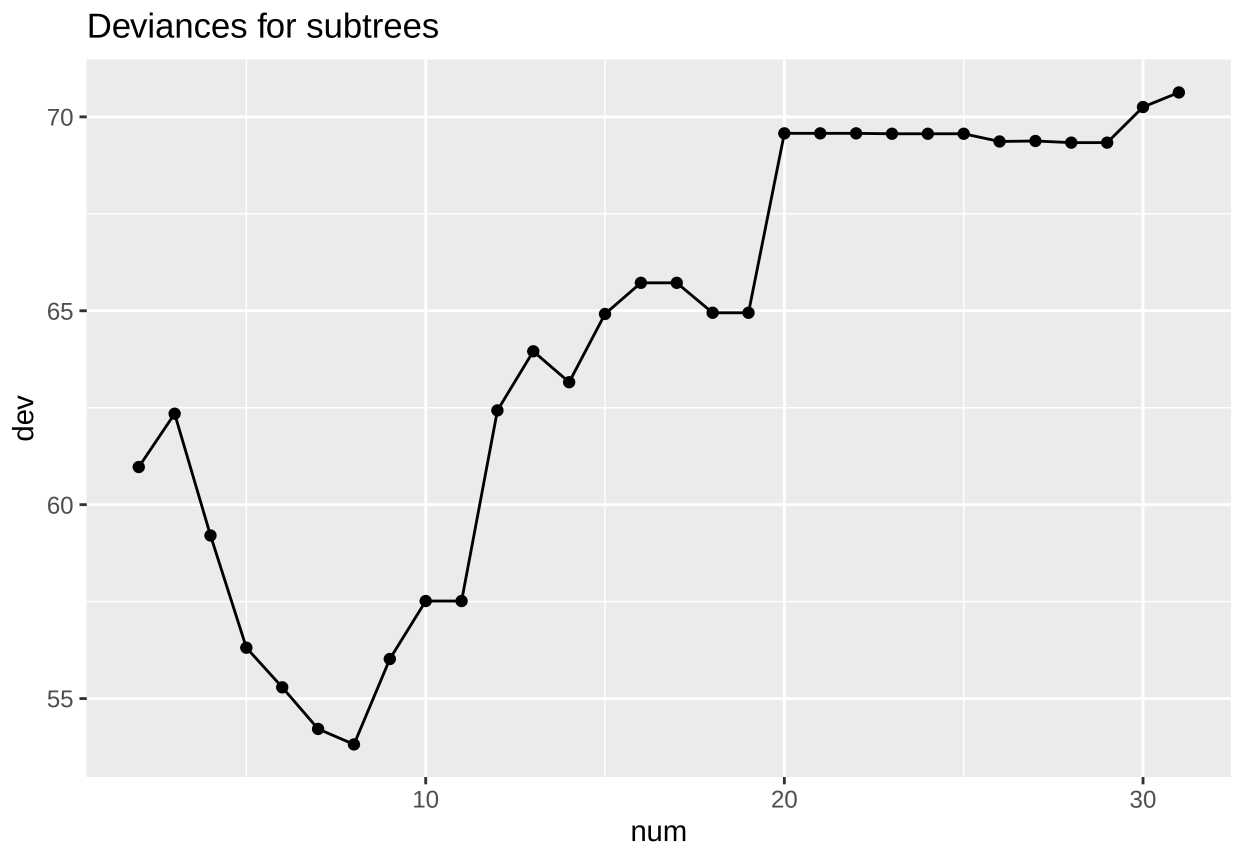
Now the usual loop. Note the pruning step. We will talk more about this shortly. At each stage we will keep track of the Deviance, essentially the MSE without the M.

```r
devs <- numeric(numLeaves)
for(num in 2:numLeaves){
  ## extract best subtree with num Leaves
  mod.prune <- prune.tree(mod.tree,best=num) ## pruning to a size!
  ## make the predictions
  preds <- predict(mod.prune,newdata=test.df)
  ## compute the deviance
  devs[num] <- with(test.df,sum((z-preds)^2))
}
devs <- devs[-1]
```

What does the plot of the Deviances look like?

Note: a quick glance at the devs shows that the values increase quite quickly after the first 20 or so

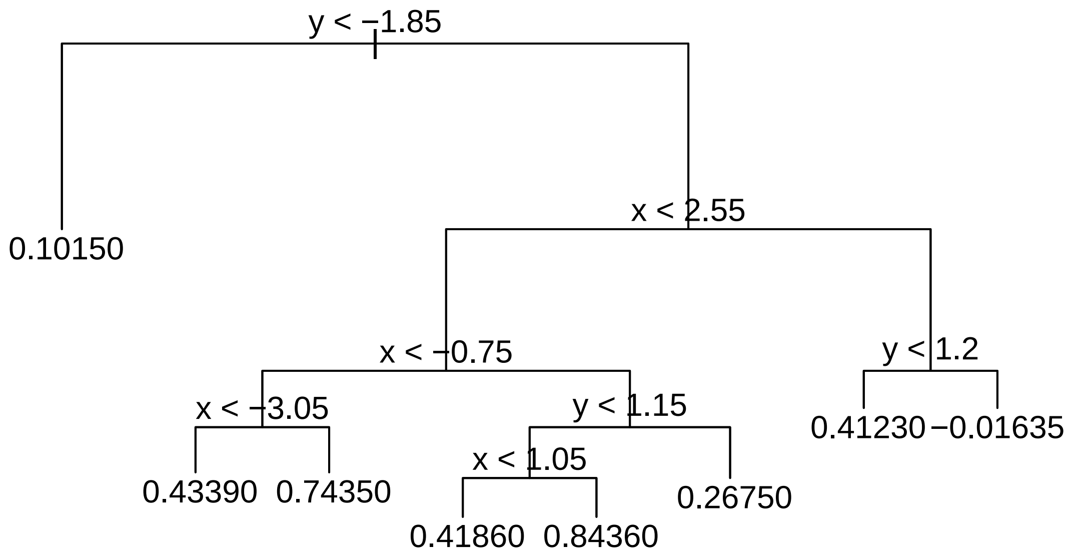
```r
maxNum <- 30

data.frame(num=2:(maxNum+1),dev=devs[1:maxNum]) %>%
  ggplot()+
  geom_point(aes(num,dev))+
  geom_line(aes(num,dev))+
  labs(title="Deviances for subtrees")
```



It looks as if the best performance occurs when the tree has about 10 leaves. maybe fewer.

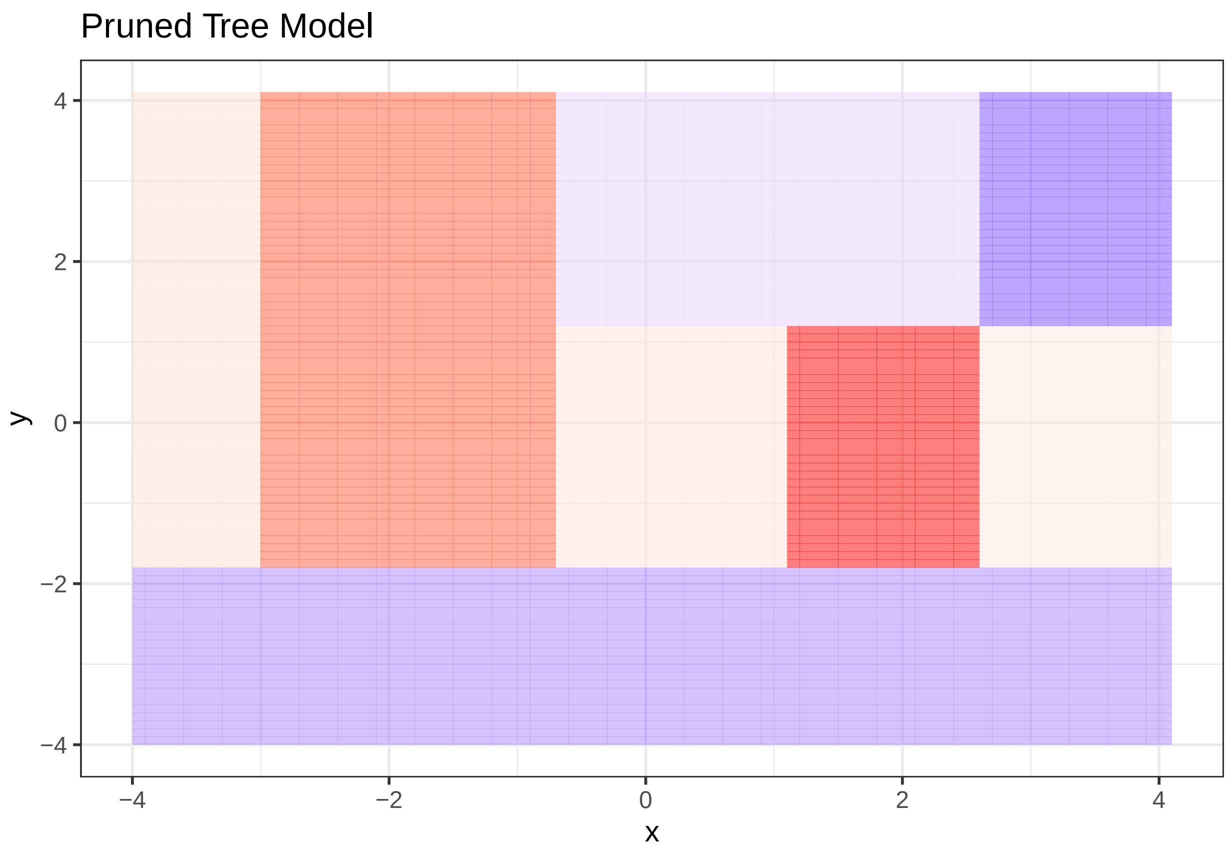
The prune.tree function will “prune” the tree back to a tree with exactly 10-15 leaves

```r
bestNum <- which.min(devs)+1
mod.prune <- prune.tree(mod.tree,best=bestNum) ## pruning to a size!
{plot(mod.prune);text(mod.prune)}
```



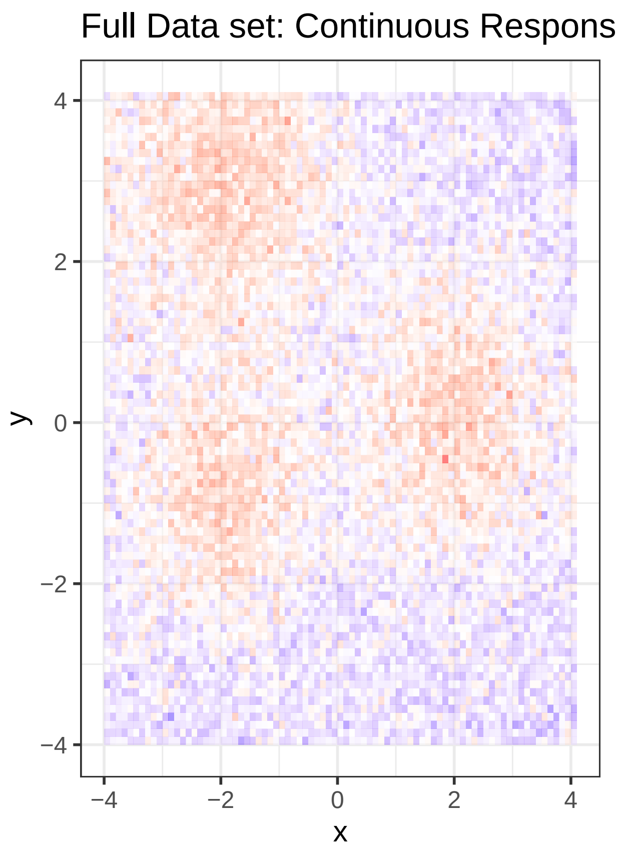
How does this look on the grid?

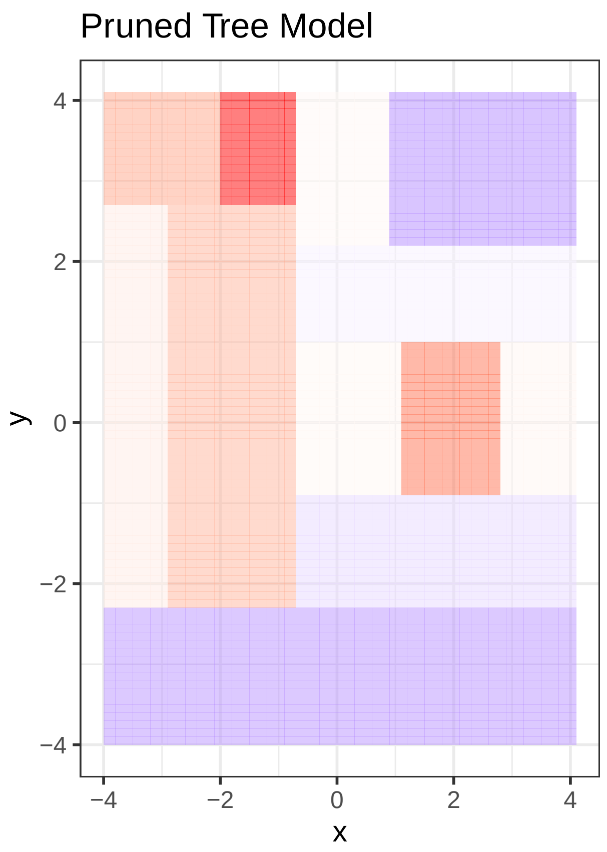
```r
regData.df$pred <- predict(mod.prune,newdata=regData.df)
##
pruned.gg <- regData.df %>%
  ggplot()+
  geom_rect(aes(xmin=x,xmax=x+del,ymin=y,ymax=y+del,fill=pred),alpha=.5)+
  scale_fill_gradient2(low="blue",mid="white",high="red",midpoint=mpt)+
  labs(title= "Pruned Tree Model",
       x="x",
       y="y")+
  guides(fill=FALSE)+
  theme_bw()
pruned.gg
```



So how does this look compared with the original structure?

```r
grid.arrange(data.gg,pruned.gg,nrow=1)
```





This looks as if it's really pulling out some detail

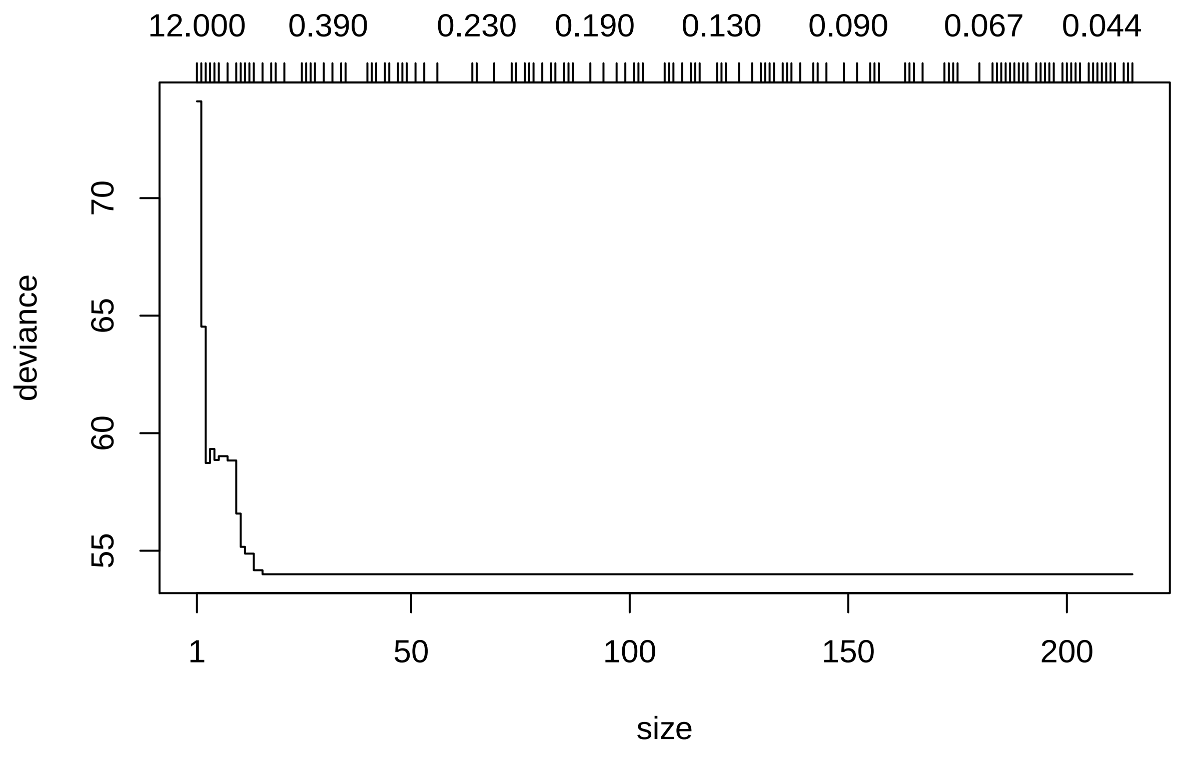
As with penalized regression, there are built-in ways to estimate an optimal model using cross-validation. Cross-validation with trees is slightly tricky. The process is described in the ISLR text (Algorithm 8.1). The key step is the "Cost Complexity" of any tree T. Let $|T|$ indicate the the number of leaves of the tree. Then, for any $\alpha > 0$, define

$$C_\alpha = \sum_{m=1}^{|T|} \text{Dev}_m + \alpha|T|$$

where Dev_m is the deviance of the m^{th} leaf. As with penalized regression, this formulation sets up a competition between to desirable features of a model. We want the total Deviance to be small but not at the expense of a large (protentially over-fitted) tree. On the other hand, we like small trees, but not at the expennse of a small (probably under-fitted) deviance. It can be shown that for any starter tree T_0 and any $\alpha > 0$, there is a sub-tree obtained by "pruning" which will minimize C_α. Then we cross-validate over α to find the optimal subtree.

All of this is thankfully done for us. To start, use the cv.tree on our large tree (mod.tree) function to cross-validated over subtrees of all sizes

```
tree.cv <- cv.tree(mod.tree)
plot(tree.cv)
```



Note that the deviances are minimized at about the same place as our exploration above revealed. The values are different because here we are cross-validating on the training data set.

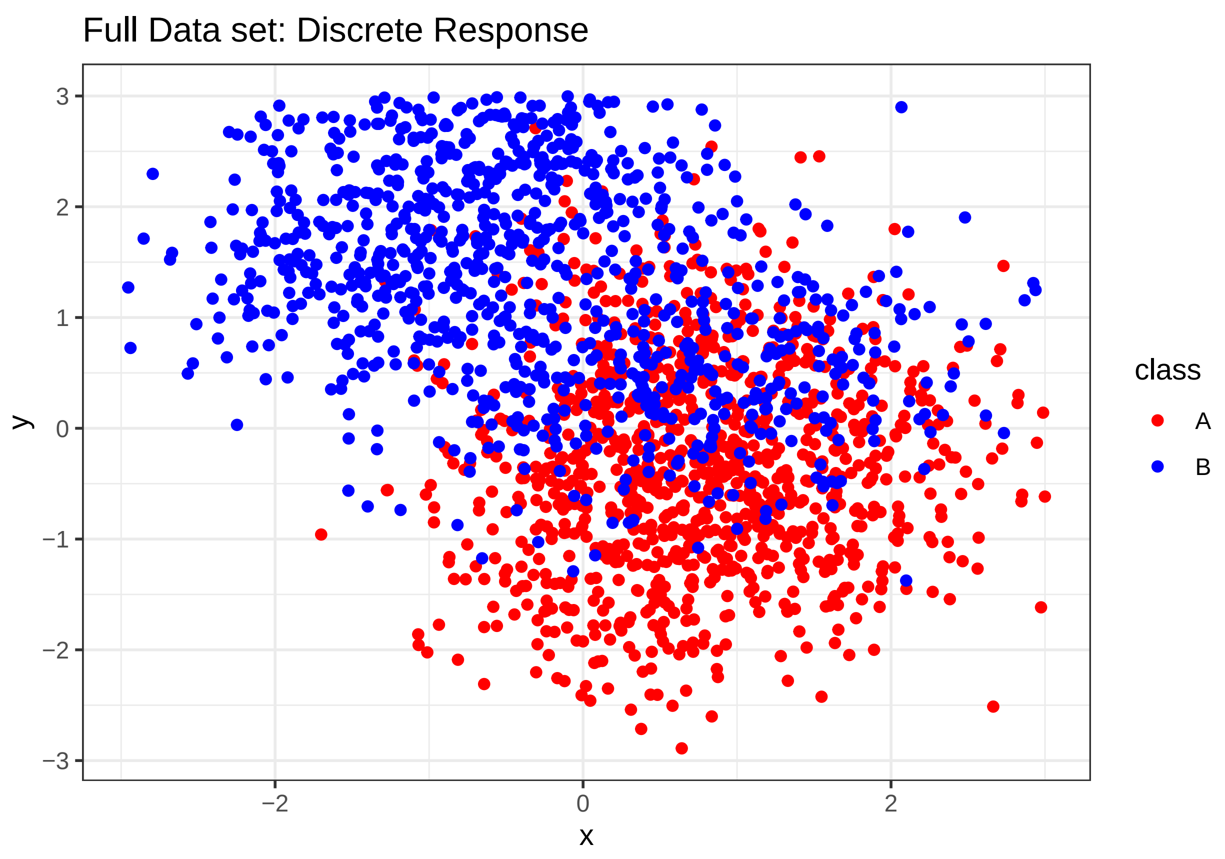
Classification Tree

Now let's look at a synthetic classification data set.

```
dataDir <- "/Users/richeym/Dropbox/COURSES/ADM/ADM_S20/CODE/Chap08"
file <- "Classify2D_Data3.csv"
classData.df <- read.csv(file.path(dataDir,file))
```

Here's a quick look at the data.

```
xvals <- with(classData.df,unique(x))
del <- xvals[2]-xvals[1]
cols <- c("red","blue")
data.gg <-
  classData.df %>%
  ggplot()+
  geom_point(aes(x,y,color=class))+
  ##geom_rect(aes(xmin=x,xmax=x+del,ymin=y,ymax=y+del,fill=class),alpha=.5)+
  scale_color_manual(values=cols)+
  labs(title= "Full Data set: Discrete Response",
       x="x",
       y="y")+
  guides(fill=FALSE)+
  theme_bw()
data.gg
```



We can proceed the same way as before. Keep in mind, that we will make split decisions based on minimizing the Gini score (to increase node purity) at each stage.

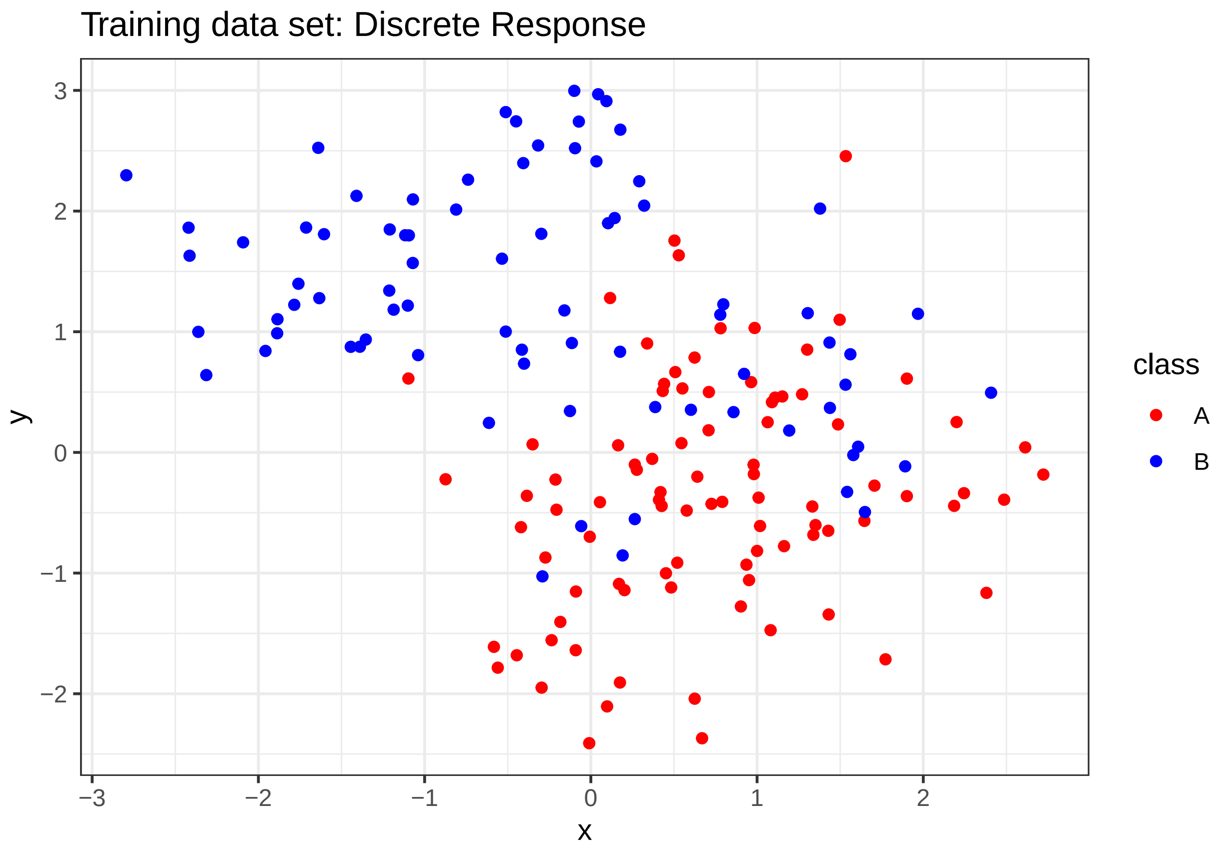
To start, build some test and training data by subsetting the large

```
N <- nrow(classData.df)
n <- 200
train <- sort(sample(1:N,n,rep=F))
test <- sample(1:N,n,rep=F)
both <- intersect(train,test)
train <- setdiff(train,both)
test <- setdiff(test,both)
train.df <- classData.df[train,]
test.df <- classData.df[test,]
```

And a graph. . . .

```
train.gg <-
  train.df %>%
   ggplot()+
  geom_point(aes(x,y,color=class))+
  ##geom_rect(aes(xmin=x,xmax=x+del,ymin=y,ymax=y+del,fill=class),alpha=.5)+
```

```r
  scale_color_manual(values=cols)+
  labs(title= "Training data set: Discrete Response",
       x="x",
       y="y")+
  guides(fill=FALSE)+
  theme_bw()
train.gg
```



Building a tree is pretty easy. The tree function recognizes that this is a classification.

```r
mod.tree <- tree(class ~ x + y,
                 data=train.df,
                 control=tree.control(nrow(train.df),
                                      mindev=0.0001,
                                      minsize=2))
```

The summary:

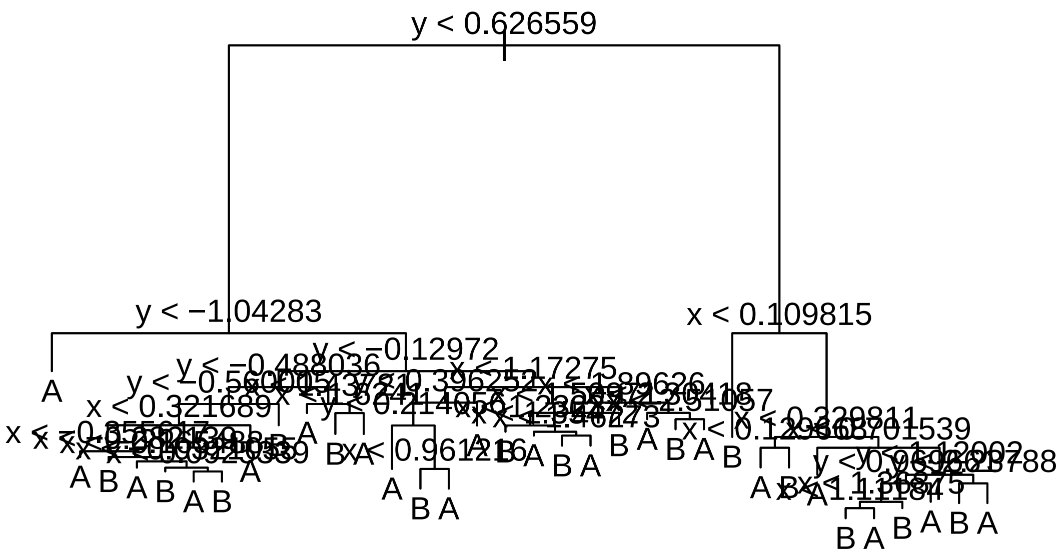
```r
summary(mod.tree)
```

```
##
## Classification tree:
## tree(formula = class ~ x + y, data = train.df, control = tree.control(nrow(train.df),
##     mindev = 1e-04, minsize = 2))
## Number of terminal nodes:  34
## Residual mean deviance:  0 = 0 / 135
## Misclassification error rate: 0 = 0 / 169
```

Note that the misclassification error rate is zero!

How's it look?

```
{plot(mod.tree);text(mod.tree)}
```



Pretty deep tree, that's why it has an error rate of zero on the training data.

How did this perform on the test data?

```
test.df$pred <- predict(mod.tree,
                        newdata=test.df,
                        type="class")
with(test.df,table(class,pred))
```

```
##      pred
## class  A  B
##     A 74 26
##     B 23 46
```

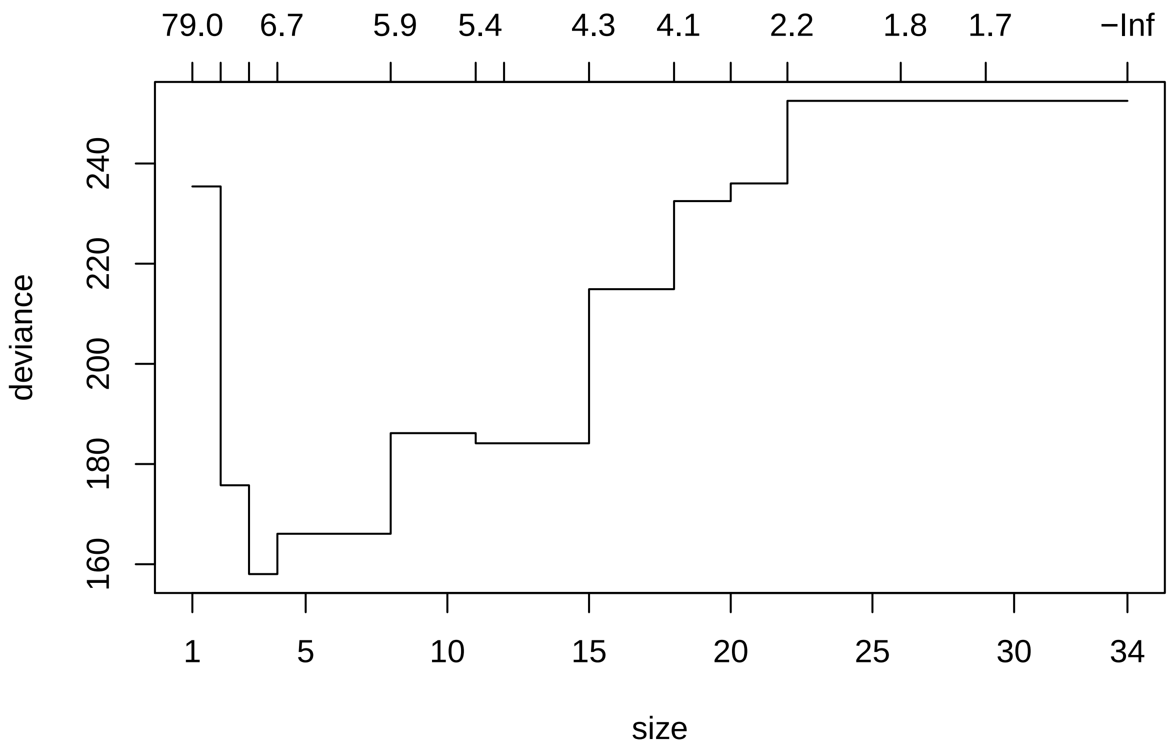
```
(err.tree <- with(test.df,mean(class != pred)))
```

```
## [1] 0.2899408
```

Ok, is this a reasonable error rate? Can we do better? Use cross-validation to find out.

The tree above, mod.tree, looks deep enough to use in the cross-validation.

```r
mod.cv <- cv.tree(mod.tree)
plot(mod.cv)
```



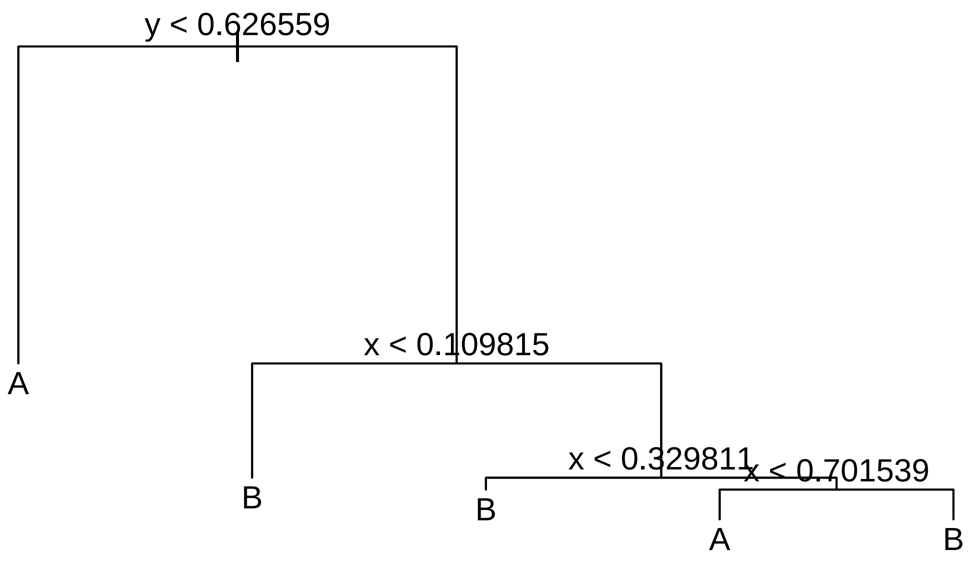
Looks like 4-5 is the best size.

```r
with(mod.cv,cbind(rev(size),rev(dev))) ##reverse the lists
```

```
##       [,1]     [,2]
##  [1,]    1 235.4188
##  [2,]    2 175.7600
##  [3,]    3 158.0311
##  [4,]    4 166.0789
##  [5,]    8 186.1680
##  [6,]   11 184.1404
##  [7,]   12 184.1404
##  [8,]   15 214.9090
##  [9,]   18 232.4957
## [10,]   20 236.0204
## [11,]   22 252.5022
## [12,]   26 252.5022
## [13,]   29 252.5022
## [14,]   34 252.5022
```

Ok, this is a pretty simple tree

```
mod.prune <- prune.misclass(mod.tree,best=5)
{plot(mod.prune);text(mod.prune)}
```



How does the pruned tree perform on the test data?

```
test.df$pred <- predict(mod.prune,
                        newdata=test.df,
                        type="class")
with(test.df,table(class,pred))
```

```
##      pred
## class  A  B
##     A 85 15
##     B 29 40
```

```
(err.prune <- with(test.df,mean(class != pred)))
```

```
## [1] 0.260355
```

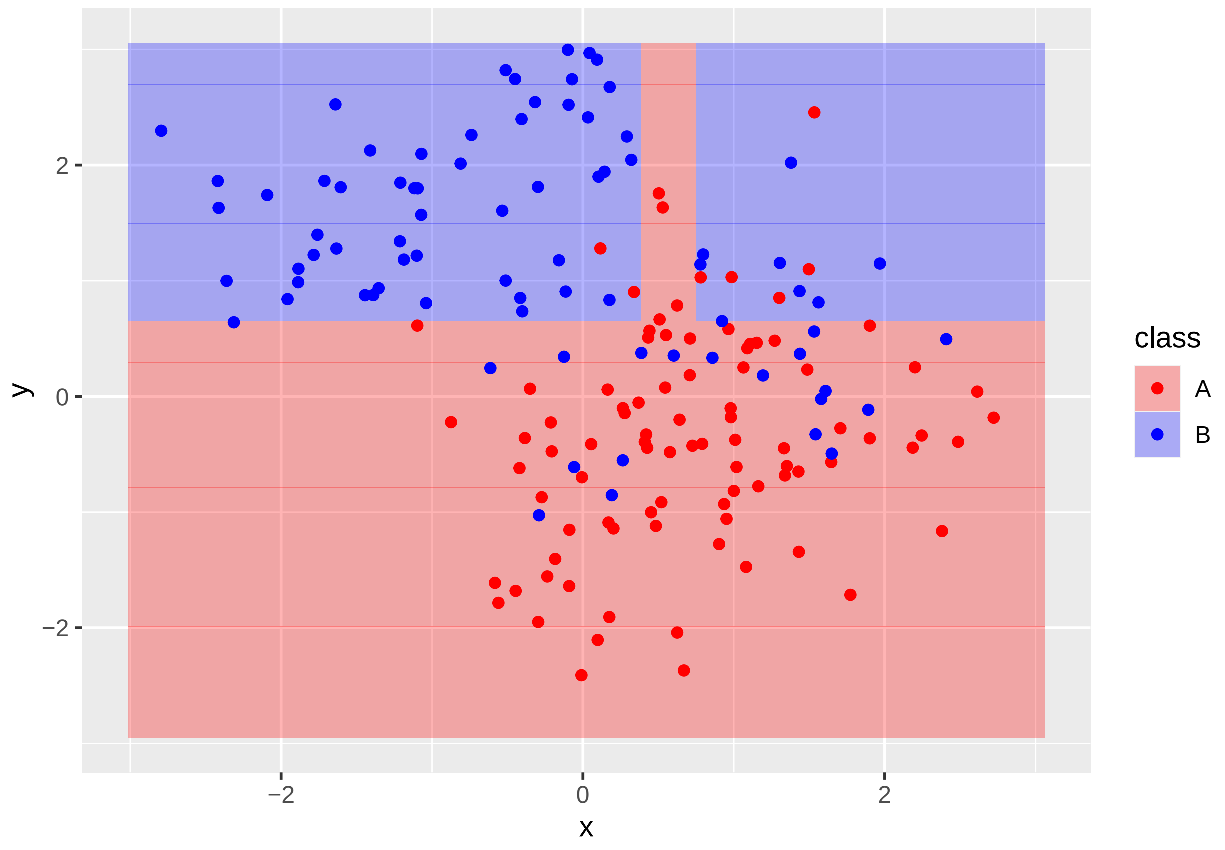
```
c(err.tree,err.prune)
```

```
## [1] 0.2899408 0.2603550
```

We can look at the optimal model on a grid (it's pretty obvious what it looks like)

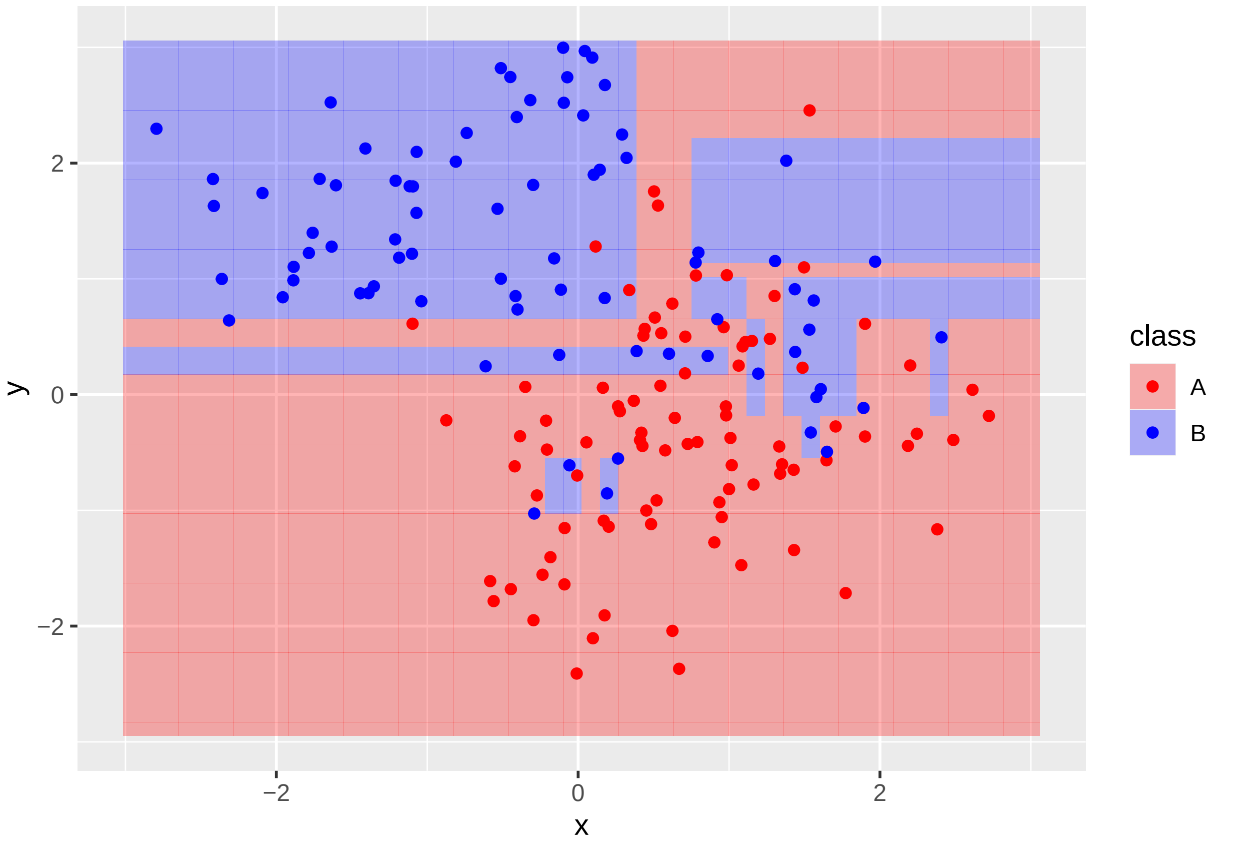
```
## build a grid
xRng <- with(classData.df,range(x))
yRng <- with(classData.df,range(y))
gridSize <- 50
grid.df <- data.frame(expand.grid(x=seq(xRng[1],xRng[2],length=gridSize),
                                  y=seq(yRng[1],yRng[2],length=gridSize)))
grid.df$class <- predict(mod.prune,newdata=grid.df,type="class")
```

```
grid.df$class <- predict(mod.prune,newdata=grid.df,type="class")
grid.df %>%
  ggplot()+
  geom_tile(aes(x,y,fill=class),alpha=.3)+
  scale_fill_manual(values=cols)+
  geom_point(data=train.df,aes(x,y,color=class))+
  scale_color_manual(values=cols)
```



As a contrast, this is how the full tree model looks.

```
grid.df$class <- predict(mod.tree,newdata=grid.df,type="class")
grid.df %>%
  ggplot()+
  geom_tile(aes(x,y,fill=class),alpha=.3)+
  scale_fill_manual(values=cols)+
  geom_point(data=train.df,aes(x,y,color=class))+
  scale_color_manual(values=cols)
```



New Training data.

Let's recreate the training data. We are interested in what changes we go through the process again.

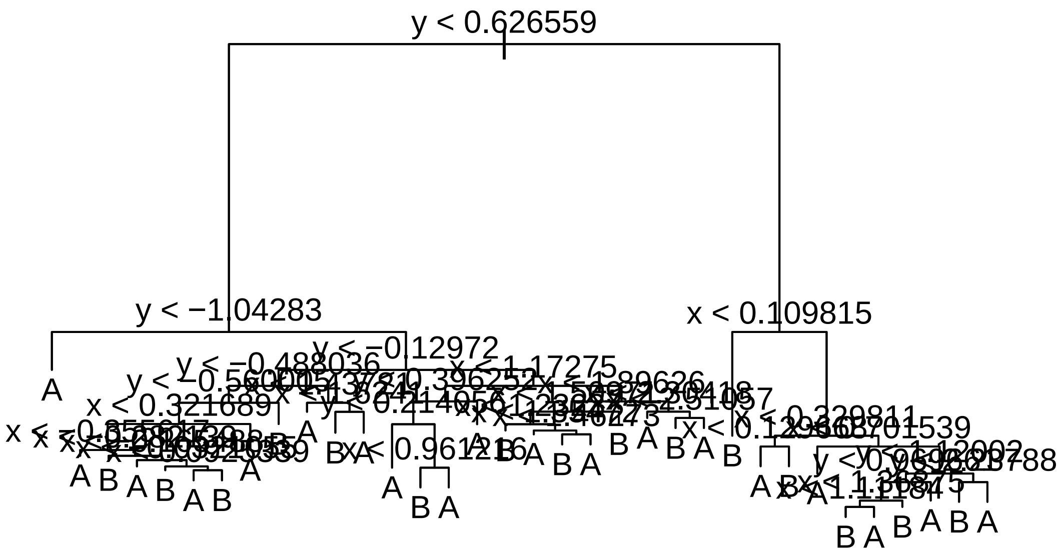
```
N <- nrow(classData.df)
n <- 200
train <- sort(sample(1:N,n,rep=F))
test <- sample(1:N,n,rep=F)
both <- intersect(train,test)
train <- setdiff(train,both)
test <- setdiff(test,both)
train.df <- classData.df[train,]
test.df <- classData.df[test,]
```

A new tree...

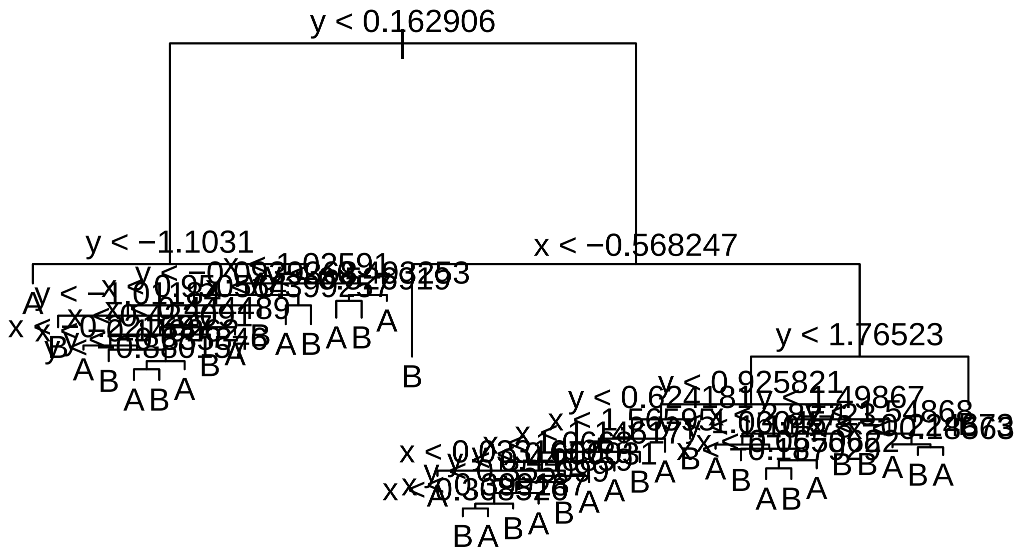
```
mod.tree2 <- tree(class ~ x + y,
                  data=train.df,
                  control=tree.control(nrow(train.df),
                                       mindev=0.0001,
                                       minsize=2))
```

Here's what both look like. . . .

```
{plot(mod.tree);text(mod.tree)}
```

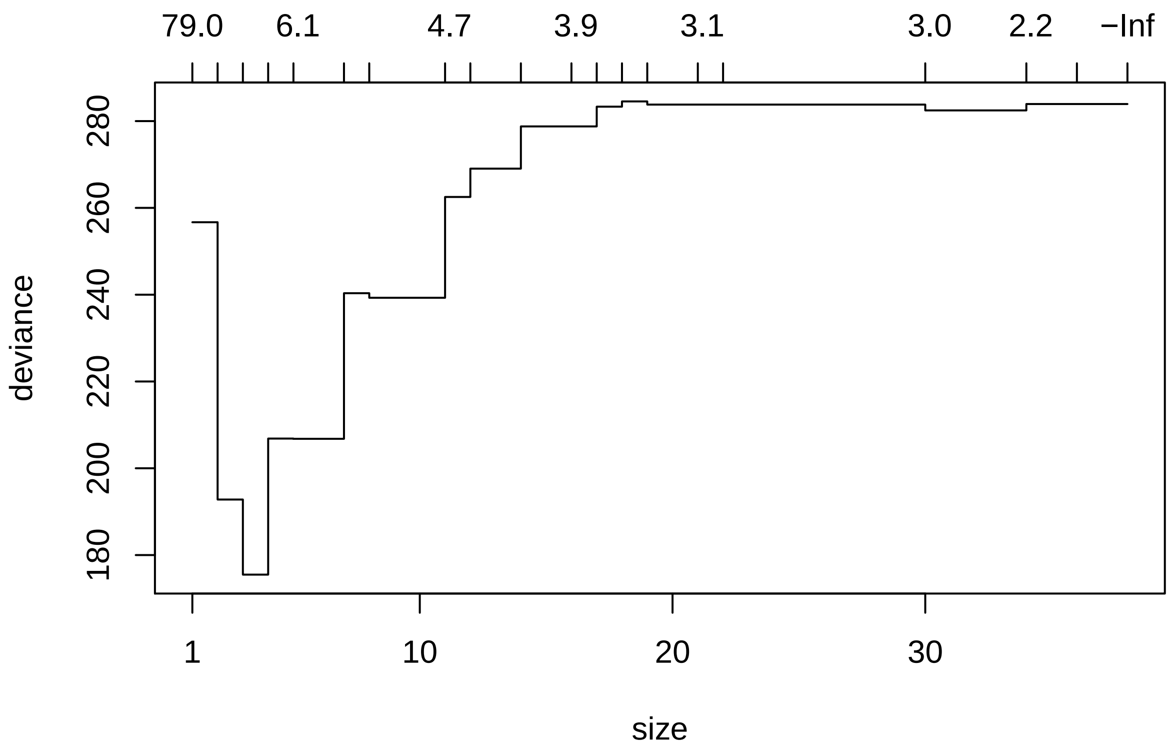


```
{plot(mod.tree2);text(mod.tree2)}
```



Cross-validate and prune the second tree.

```r
mod.cv2 <- cv.tree(mod.tree2)
plot(mod.cv2)
```



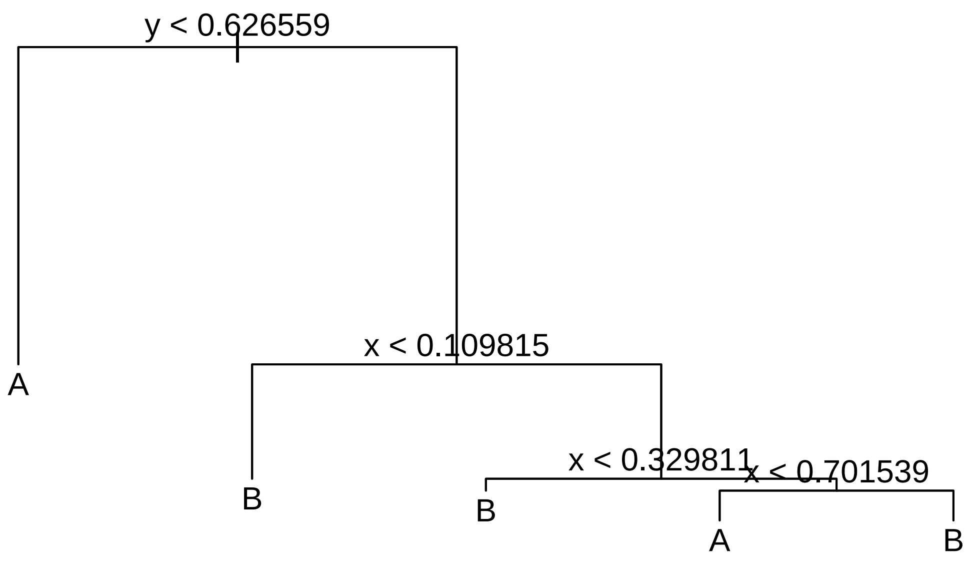
```r
with(mod.cv2,cbind(rev(size),rev(dev)))
```

```
##       [,1]     [,2]
##  [1,]    1 256.6903
##  [2,]    2 192.7978
##  [3,]    3 175.4947
##  [4,]    4 206.8467
##  [5,]    5 206.7919
##  [6,]    7 240.3389
##  [7,]    8 239.2867
##  [8,]   11 262.5148
##  [9,]   12 269.0466
## [10,]   14 278.7736
## [11,]   16 278.7736
## [12,]   17 283.3206
## [13,]   18 284.5328
## [14,]   19 283.8061
## [15,]   21 283.8061
## [16,]   22 283.8061
## [17,]   30 282.4602
## [18,]   34 283.9300
## [19,]   36 283.9300
## [20,]   38 283.9300
```

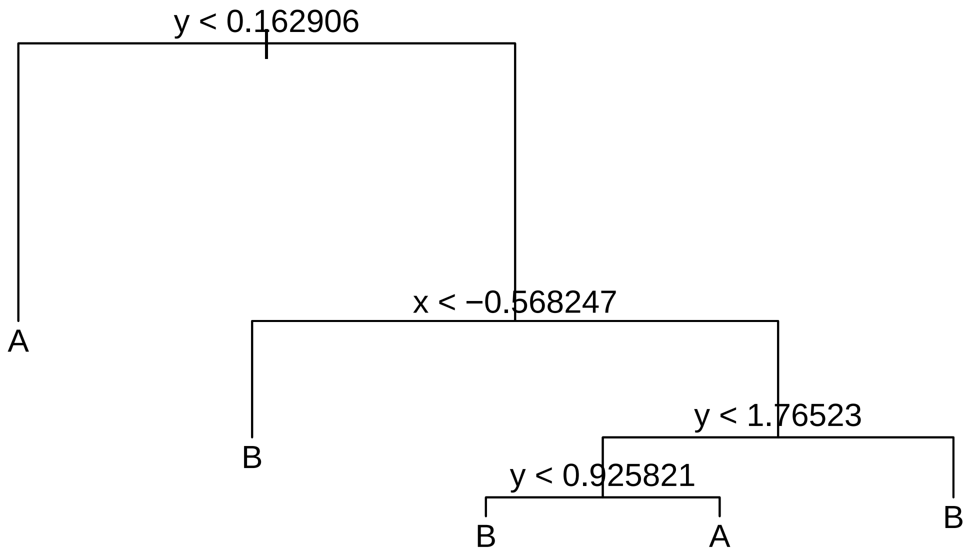
```r
best <- with(mod.cv2,which.min(rev(dev)))
mod.prune2 <- prune.misclass(mod.tree2,best=best)
```

How do the pruned trees compare?

```r
{plot(mod.prune);text(mod.prune)}
```



```r
{plot(mod.prune2);text(mod.prune2)}
```



These are quite different from each other. The size is similar, but the individual decisions nodes are not.

Conclusions

Decision trees, for either regression or classification, are easily implemented and interpreted. However, these trees are very "fragile" in the sense that minor changes to the original data set can lead to very different tree structures.

Next we will look at how we can use this tree variablitity to build a powerful predictive tool.

Assignment 1

KNN and Decision Trees are similar in some ways. For each of the two data sets above, compare how an optimal decision tree performs relative to an optimal KNN model. Are there reasons you would prefer one predictive modeling technique over the other?

Assignment 2

Use regression trees to model the 2012 County Election Data we used before. In this case, the response variable (VoterProp) is quantitative. How do the results compare with the results of the models from the Feature Selection dicussion? Which do you think is the best model?

Replace the VoterProp variable with a binary variable, HighTurnOut, indicating whether VoterProp is greater than 0.5. Use a classification tree to predict HighTurnOut.