3.1 Decision Trees

Jonathan De Los Santos

2/3/2021

Contents

1 Decision Tree Overview **2**

1.1 Tree Components . . . 2

1.2 Indices . . . 2

1.3 GINI Index . . . 2

1.4 Entropy . . . 3

1.5 Misclassification Measure (Classification Error) . . . 4

1.6 Splitting Method Comparison . . . 4

1.7 Post-Pruning . . . 5

2 C5.0 Tree Algorithm **5**

3 Cross Validating a Classification Tree **5**

4 CART Algorithm **6**

4.1 Data Prep . . . 6

4.2 Build the Model . . . 6

4.3 Classification Tree . . . 8

4.4 Pruned Tree . . . 9

4.5 Predict and Evaluate . . . 10

5 Rpart Algorithm **11**

5.1 Pruning Rpart . . . 12

5.2 Evaluating RPart . . . 13

6 Party Algorithm!! **14**

6.1 Evaluate your Party . . . 14

1 Decision Tree Overview

Decision trees are classifiers that use a tree structure to model the relationships among the variables and potential outcomes

- The primary challenge of decision trees is figuring out which feature/class to split on
- Purity: the degree to which a subset of examples contains only a single class/feature
- Pure: a subset composed of a single class

1.1 Tree Components

- Root node: the first group with all of the training data
- Decision or internal node: Require choices to be made on the attributes of the job
  - The data is split across branches that indicate the poential outcomes of the decision
- Terminal or leaf nodes: the final subgroups
- The key problem to evaluate is the effectiveness of the splitting
  $p(i : t)$
  i = feature or class t = node

1.2 Indices

1.3 GINI Index

- Measures *impurity*, or how dissimilar the class labels are for the data instances belonging to a common node

$$1 - \sum_{i=0}^{n}$$

- i - class
- t = node
- $p_i(t)$ is the relative frequency of class i at node t
- c is the total number of classes

Maximum: $1 - \frac{1}{n_c}$

- Records are equally distributed among all classes
- Least interesting information Minimum: (0)
- All records belong to one class
- Most interesting information

C1	0
C2	6
Gini=0.000	

C1	1
C2	5
Gini=0.278	

C1	3
C2	3
Gini=0.500	

1.3.1 Splitting

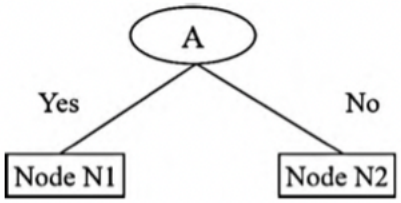
$GINI_{split} = \sum \frac{k}{i=1} \frac{n_i}{n} GINI(i)$ - n_i is the number of records at child i - n is the number of records at node p

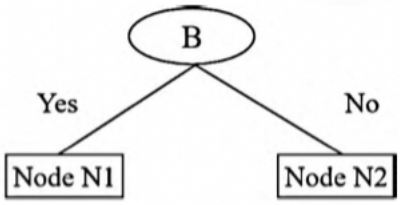
1.3.1.1 Splitting of Binary Attributes

Solve the Gini index for A

ttributes

	Parent
C1	6
C2	6
Gini = 0.500	





	N1	N2
C1	4	2
C2	3	3
Gini=?		

	N1	N2
C1	5	1
C2	2	4
Gini=0.333		

Gini for Node N1: $1 - (\frac{4}{7})^2 - (\frac{3}{7}^2) = 0.4898$

Gini for Node N2: $1 - (\frac{2}{5})^2 - (\frac{3}{5}^2) = 0.48$

Gini for Entire Split: $\frac{7}{12} * 0.4898 + \frac{5}{12} * 0.48 = 0.485$

1.4 Entropy

- Quantifies the randomness or disorder within a set of class values
- Sets with high entropy are very diverse
  - Gives us little information about other items that may also belong in the set
  - No apparent commonality
- The goal is to find splits that reduce entropy, or increase homogeneity within the groups
- For n classes, entropy ranges from 0 to $\log_2(n)$
  - Minimum value indicates that the sample is homogenous
  - Max value indicates the data are as diverse as possible

$$-\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t)$$

- c = Number of class levels
- p_i = Proportion of values in class level i

1.4.1 Information Gain

A measure of the change in homogeneity that would result from a split on each possible feature - Calculated as the difference between the entropy in the segment before the split and the partions resulting from the split

$InfoGain(F) = Entropy(S_1) - Entropy(S_2)$

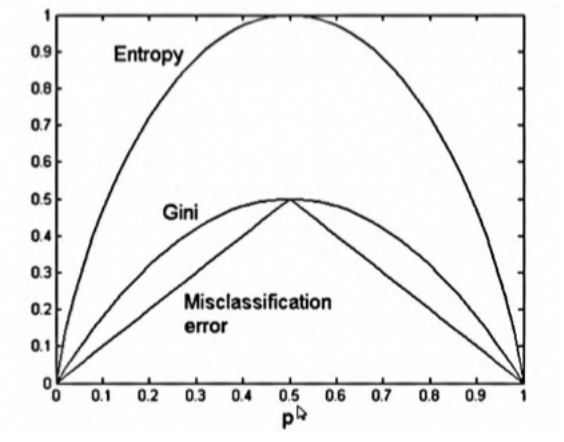
- F = Feature
- S_1 = The unsplit set
- S_2 = Partitions resulting from the split
  - This is divided into more than one partition
  - To consider the total entropy across all partitions, it weights each partition's entropy by the proportion of records falling into each one
  - See ML with R - Brett Lantz pp. 133

1.5 Misclassification Measure (Classification Error)

Classification Error $= 1 - \max_i[p_i(t)]$

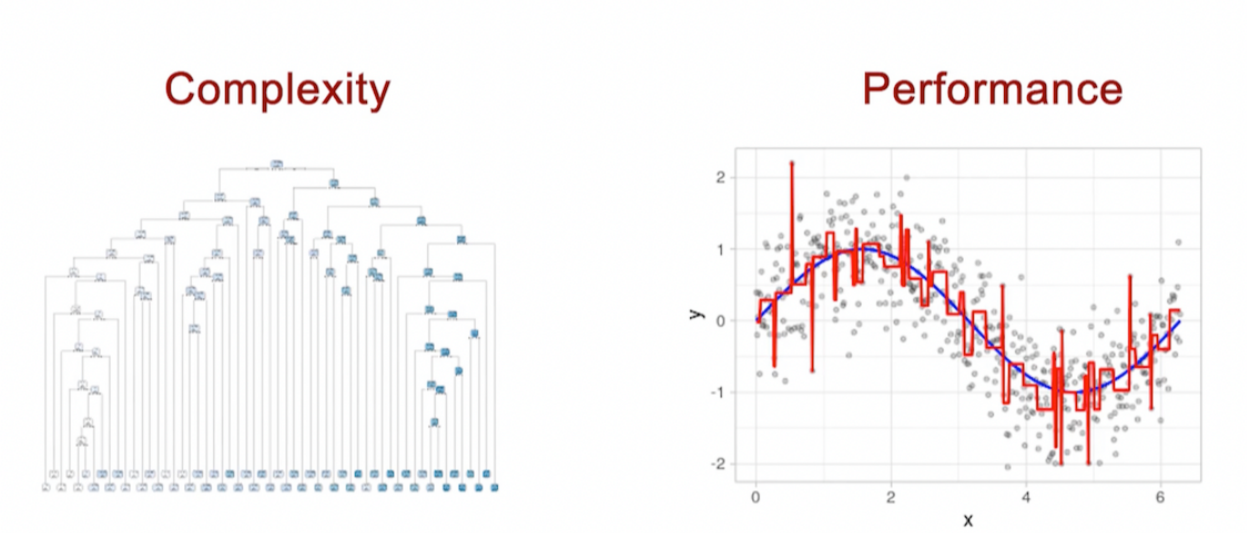
- Measures misclassification error made by a node
- Maximum $(1 - \frac{1}{n_c}$
  - When records are equally distributed among all classes
  - Least interesting
- Minimum (0.0)
  - When records all belong to one class
  - Most interesting information

1.6 Splitting Method Comparison



Pruning

If a decision tree grows too large, many of the decisions will be overly specific to the data - This causes overfitting, or poor generalization - **Pruning:** Reducing the size of a tree such that it generalizes better to new data



Early Stopping

Pre-pruning a tree by restricting it from splitting further once it reaches a certain number of decisions - Minimum terminal node size - May miss important patterns that would have been found in a larger tree

1.7 Post-Pruning

- Growing a tree too large then pruning leaf nodes back to find the optimal final tree
- Find a cost complexity parameter
  - $\min SSE + \alpha|T|$
  - I have no idea what this means

2 C5.0 Tree Algorithm

- Industry standard for producing decision trees
- Uses entropy, explained above
- Post-prunes by removing nodes and branches that have little effect on classification errors

3 Cross Validating a Classification Tree

- Two-fold method
  - Split training data into two partitions (folds)
  - Run the tree with one fold as train and the other as test, then flip the test/train label and run it again
  - Average the performance
- K-fold
  - Same concept but split into k number of folds
  - Each iteration, one fold is a test while the others are trains
  - Run the model k number of times so each fold is a test once

4 CART Algorithm

Classification and Regression Tree (CART) - Another tree algorithm - Who cares about specifics let's get goin

4.1 Data Prep

Data set: Heart health database - We are trying to predict alveolar hydatid disease (AHD) - Specifically we will use AHD.f which is factored

Fire up that caret package and create your data partitions babyyy

```
library(caret)
```

```
heart <- read.csv("Data Sets/3.0-Heart.csv")
set.seed(123)
heart <- heart[,-1]
partition <- createDataPartition(y = heart$AHD, p = 0.7, list = FALSE)
heart$AHD <- factor(heart$AHD)
#heart <- subset(heart, select = -c(AHD.f))

train<- heart[partition,]
test <- heart[-partition,]
str(train)
```

```
## 'data.frame':    213 obs. of  14 variables:
##  $ Age      : int  67 56 57 57 56 56 44 48 54 48 ...
##  $ Sex      : int  1 1 0 1 0 1 1 1 1 0 ...
##  $ ChestPain: chr  "asymptomatic" "nontypical" "asymptomatic" "asymptomatic" ...
##  $ RestBP   : int  160 120 120 140 140 130 120 110 140 130 ...
##  $ Chol     : int  286 236 354 192 294 256 263 229 239 275 ...
##  $ Fbs      : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ RestECG  : int  2 0 0 0 2 2 0 0 0 0 ...
##  $ MaxHR    : int  108 178 163 148 153 142 173 168 160 139 ...
##  $ ExAng    : int  1 0 1 0 0 1 0 0 0 0 ...
##  $ Oldpeak  : num  1.5 0.8 0.6 0.4 1.3 0.6 0 1 1.2 0.2 ...
##  $ Slope    : int  2 1 1 2 2 2 1 3 1 1 ...
##  $ Ca       : int  3 0 0 0 0 1 0 0 0 0 ...
##  $ Thal     : chr  "normal" "normal" "normal" "fixed" ...
##  $ AHD      : Factor w/ 2 levels "No","Yes": 2 1 1 1 1 2 1 2 1 1 ...
```

4.2 Build the Model

Using the tree library we will build out the decision tree

4.2.1 `tree` Function

```
tree(formula, data, weights, subset,
     na.action = na.pass, control = tree.control(nobs, ...),
     method = "recursive.partition",
     split = c("deviance", "gini"),
     model = FALSE, x = FALSE, y = TRUE, wts = TRUE, ...)
```

Arguments:

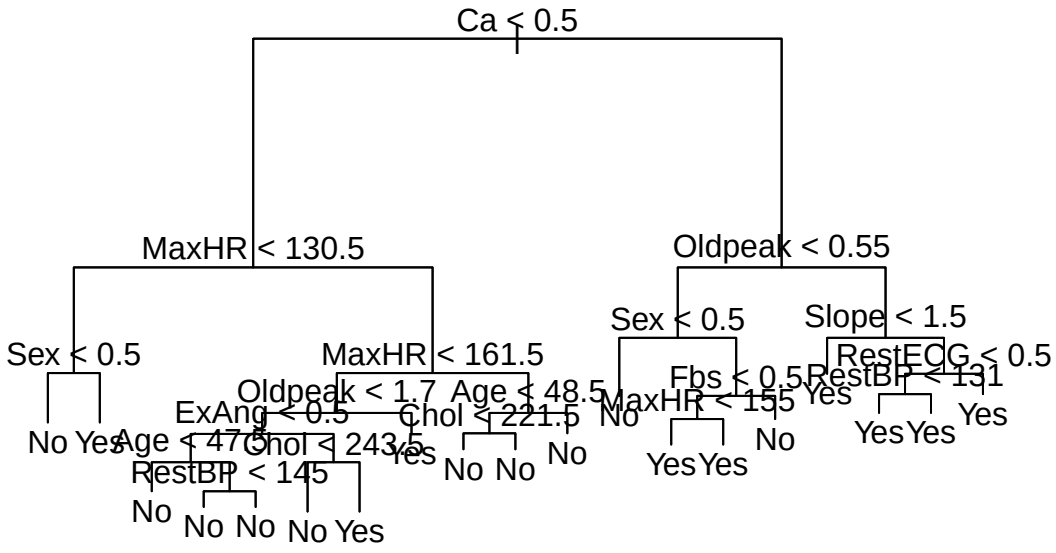
- formula: The left-hand-side (response) should be either a numerical vector when a regression tree will be fitted or a factor, when a classification tree is produced. The right-hand-side should be a series of numeric or factor variables separated by +; there should be no interaction terms. Both . and - are allowed: regression trees can have offset terms.
- data: A data frame in which to preferentially interpret formula, weights and subset.
- weights: Vector of non-negative observational weights; fractional weights are allowed.

4.2.2 Model

- We examine our factored AHD column with the rest of the columns as factor variables
- `plot` creates the tree but it has no labels until we add
- `text` which adds text labels to the tree

```
#install.packages("tree")
```

```
library(tree)
treemod <- tree(AHD~., data = train)
plot(treemod)
text(treemod)
```



4.3 Classification Tree

The classification tree will give us the misclassification or variance of various-sized trees to help us select the best size for pruning - Size in this case refers to number of terminal nodes

4.3.1 `cv.tree` Function

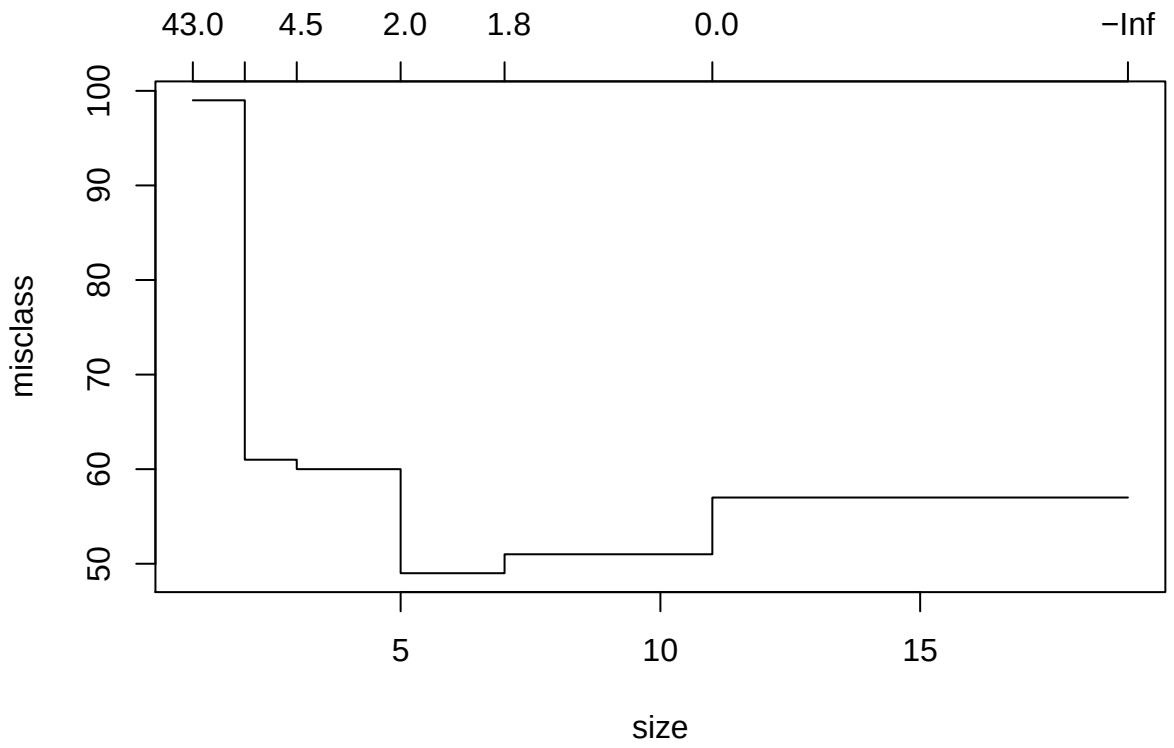
Runs a K-fold cross-validation experiment to find the deviance or number of misclassifications as a function of the cost-complexity parameter k.

```
cv.tree(object, rand, FUN = prune.tree, K = 10, ...)
```

- object: a tree object
- rand: Optional number of cases used to create object to assign cases to different groups (?)
- FUN: the pruning function
- k: number of folds for k-fold cross validation

4.3.2 The Model

```
cv.trees <- cv.tree(treemod, FUN = prune.misclass )
plot(cv.trees)
```



4.3.3 `prune.misclass` function

Determines a nested sequence of subtrees of the supplied tree by recursively "snipping" off the least important splits. This is an abbreviation for `prune.tree(method = "misclass")`

```
prune.misclass(tree, k = NULL, best = NULL, newdata,
               nwts, loss, eps = 1e-3)
```

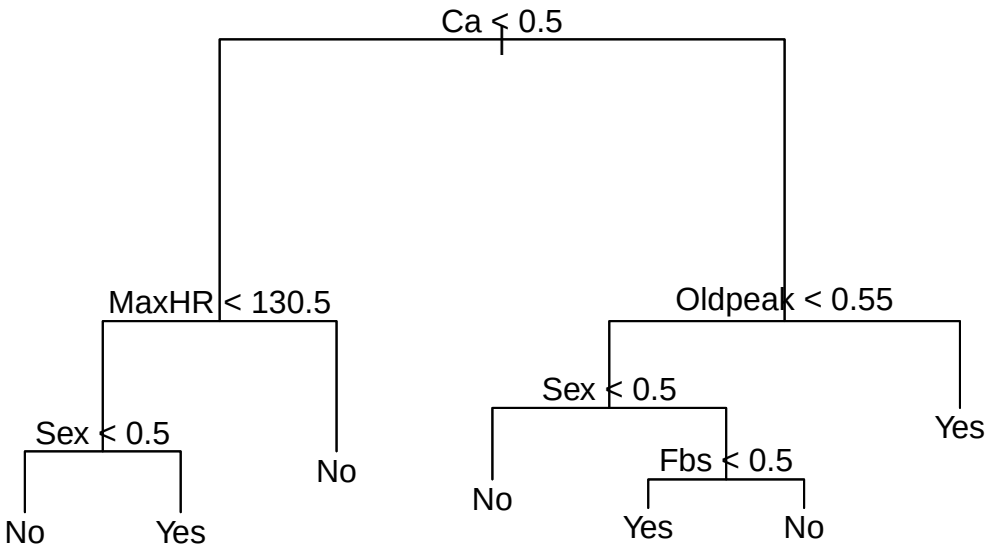
Arguments:

- tree: tree object
- k: cost-complexity parameter defining either a specific subtree of tree (k a scalar) or the (optional) sequence of subtrees minimizing the cost-complexity measure (k a vector). If missing, k is determined algorithmically.
- best: integer requesting the size (i.e. number of terminal nodes) of a specific subtree in the cost-complexity sequence to be returned. This is an alternative way to select a subtree than by supplying a scalar cost-complexity parameter k. If there is no tree in the sequence of the requested size, the next largest is returned.

4.4 Pruned Tree

- Based on the classification tree above, we see the lowest misclassifcations around 5
- For this example we went with 6 as `best`

```
prune.trees <- prune.misclass(treemod, best = 6)
plot(prune.trees)
text(prune.trees, pretty = 0)
```



4.5 Predict and Evaluate

4.5.1 `predict()` Function

Creates a vector of predicted class values which can be compared to the actual class values (`test`)

We then create a confusion matrix of our predictions and test

Our model isn't very good.

```
library(e1071)
treepred <- predict(prune.trees, test, type = "class")
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```
confusionMatrix(treepred, test$AHD)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  42  15
##        Yes  7  26
##
##               Accuracy : 0.7556
```

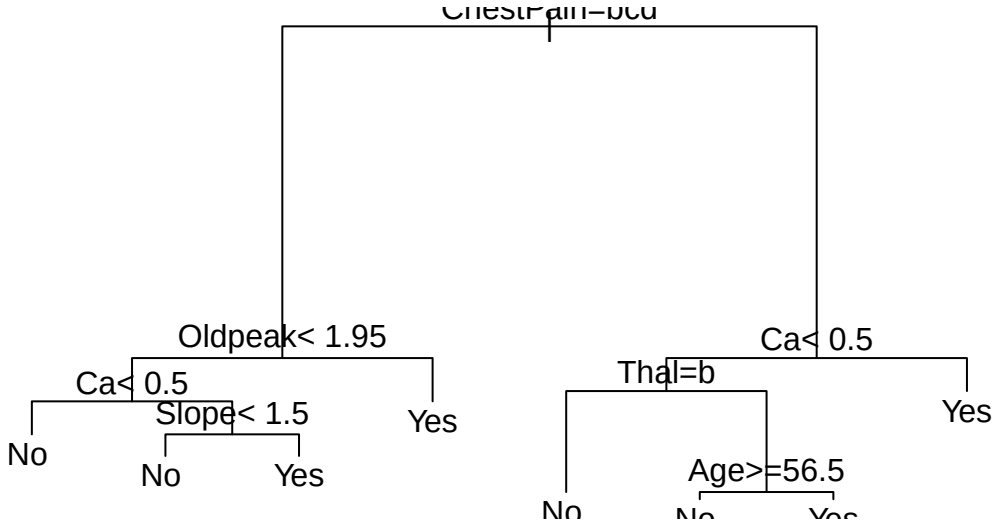
```
##                  95% CI : (0.6536, 0.84)
##     No Information Rate : 0.5444
##     P-Value [Acc > NIR] : 2.879e-05
##
##                   Kappa : 0.4992
##
##  Mcnemar's Test P-Value : 0.1356
##
##             Sensitivity : 0.8571
##             Specificity : 0.6341
##          Pos Pred Value : 0.7368
##          Neg Pred Value : 0.7879
##              Prevalence : 0.5444
##          Detection Rate : 0.4667
##    Detection Prevalence : 0.6333
##       Balanced Accuracy : 0.7456
##
##        'Positive' Class : No
##
```

5 Rpart Algorithm

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 4.0.2
```

```
rpartmod <- rpart(AHD~., data = train, method = "class")
plot(rpartmod)
text(rpartmod)
```



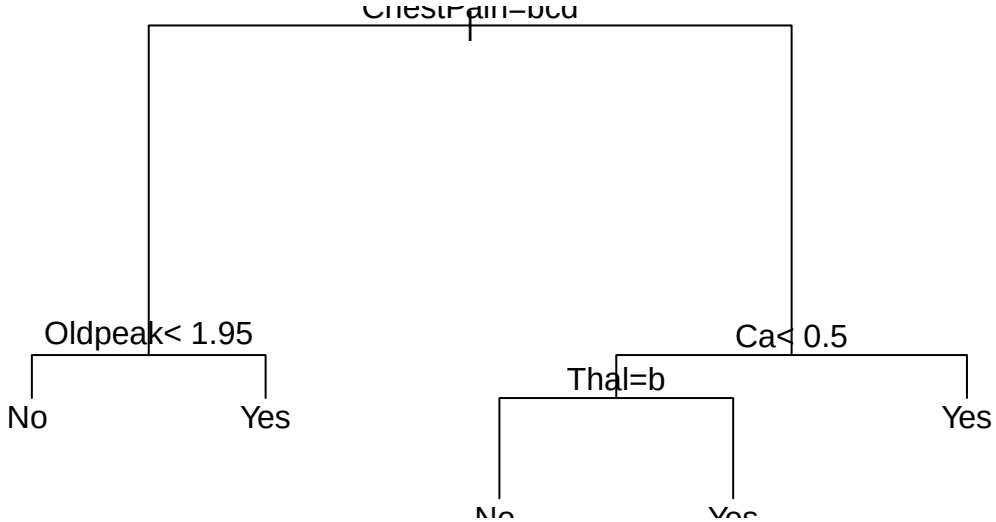
5.1 Pruning Rpart

I'm not sure what went wrong here but know that I'm very sorry

```
printcp(rpartmod)
```

```
##
## Classification tree:
## rpart(formula = AHD ~ ., data = train, method = "class")
##
## Variables actually used in tree construction:
## [1] Age       Ca        ChestPain Oldpeak   Slope     Thal
##
## Root node error: 98/213 = 0.46009
##
## n= 213
##
##         CP nsplit rel error  xerror     xstd
## 1 0.469388      0   1.00000 1.00000 0.074224
## 2 0.071429      1   0.53061 0.59184 0.066292
## 3 0.061224      3   0.38776 0.61224 0.066989
## 4 0.015306      4   0.32653 0.47959 0.061757
## 5 0.010204      6   0.29592 0.53061 0.063973
## 6 0.010000      7   0.28571 0.53061 0.063973
```

```
ptree <- prune(rpartmod, cp =  rpartmod$cptable[which.min(rpartmod$cptable[,"xerror"]),"CP"])
plot(ptree)
text(ptree)
```



5.2 Evaluating RPart

This would make more sense if my Rpart wasn't broken. Nevertheless, we persisted.

```
library(e1071)
rpartpred <- predict(ptree, test, type = "class")
confusionMatrix(rpartpred, test$AHD)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction No Yes
##        No  40   9
##        Yes  9  32
##
##                Accuracy : 0.8
##                  95% CI : (0.7025, 0.8769)
##     No Information Rate : 0.5444
##     P-Value [Acc > NIR] : 3.697e-07
##
```

```
##                   Kappa : 0.5968
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.8163
##             Specificity : 0.7805
##          Pos Pred Value : 0.8163
##          Neg Pred Value : 0.7805
##              Prevalence : 0.5444
##          Detection Rate : 0.4444
##    Detection Prevalence : 0.5444
##       Balanced Accuracy : 0.7984
##
##        'Positive' Class : No
##
```

6 Party Algorithm!!

Please accompany this section with "Let's Groove Tonight" by Earth, Wind, and Fire.

JK no party because this stupid package is buggy. Removed {r} code tag for now.

```
library(party)
partymod <- ctree(AHD~., data = train)
plot(partymod)
```

6.1 Evaluate your Party

```
partypred <- predict(partymod, test)
confusionMatrix(partypred, test$AHD)
```