Chapter 8 in ISL: Tree-Based Methods

28 March 2019

Introduction

Objective throughout this whole class: Predict a response variable Y using predictors $X_1, X_2, \ldots, X_p$.

Previously we have looked at **global models** much of the time.

We assumed there was some formula of the predictors that would model the values of the response:

$$\mu_{y \mid x} = f(\underline{X})$$

We assume this model holds over the *whole* space of our predictors.

- What if there are a lot of complicated predictors?
- What if f is a non-linear function?
- What if the relationship between Y and the predictors is very different for different regions of the predictors?
- Complicated models can be very difficult to find formulas that approximate them well.

We have gone over a couple non-parametric methods: kernel smoothing and KNN.

Next we will look at Classification and Regression trees.

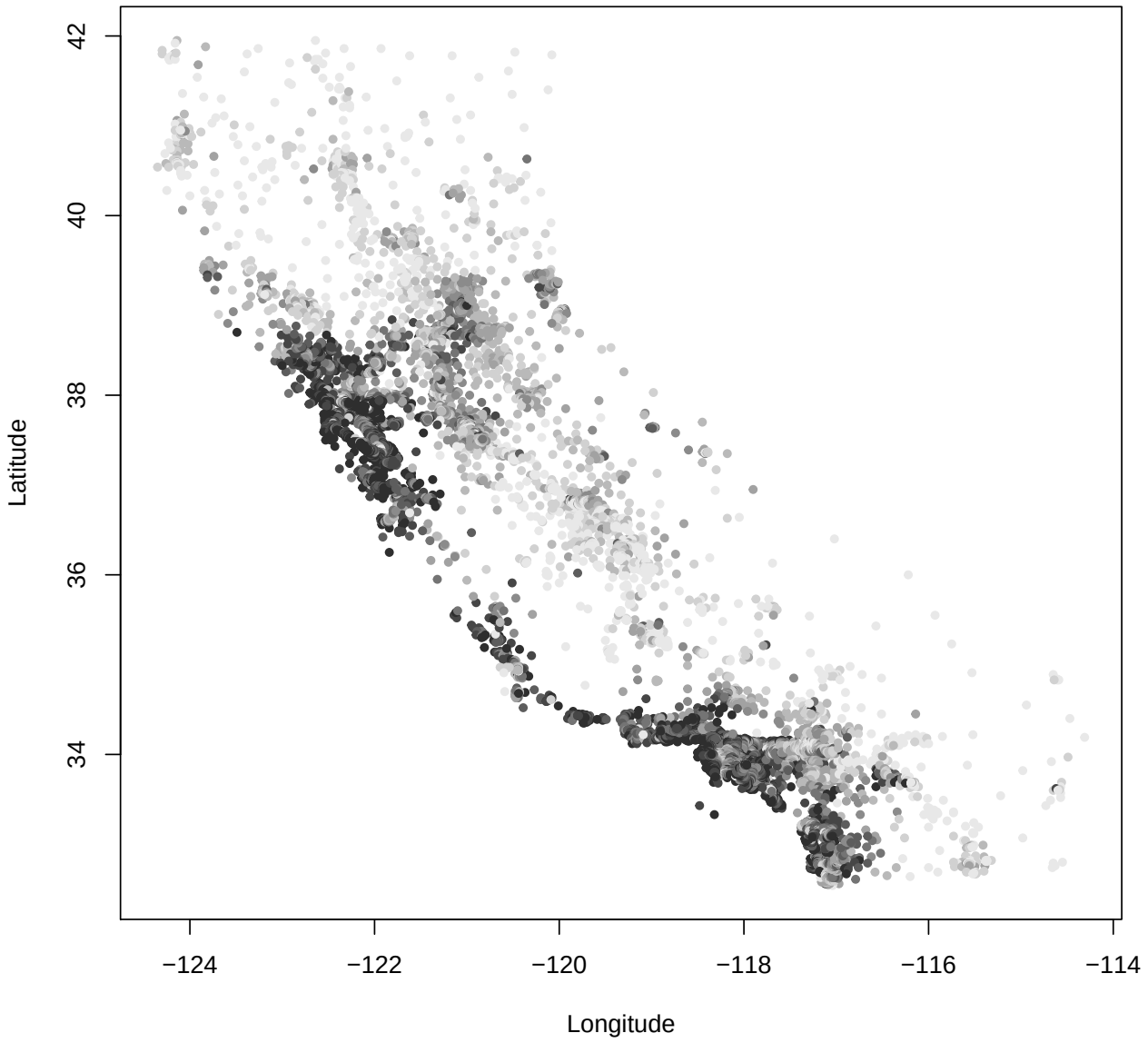
Example Data, California Median House Prices

What patterns do you see in median housing prices?

```
calif <- read.table("http://www.stat.cmu.edu/~cshalizi/350/hw/06/cadata.dat",
header = TRUE)

head(calif)
```

```
##   MedianHouseValue MedianIncome MedianHouseAge TotalRooms TotalBedrooms
## 1           452600       8.3252             41        880           129
## 2           358500       8.3014             21       7099          1106
## 3           352100       7.2574             52       1467           190
## 4           341300       5.6431             52       1274           235
## 5           342200       3.8462             52       1627           280
## 6           269700       4.0368             52        919           213
##   Population Households Latitude Longitude
## 1        322        126    37.88   -122.23
## 2       2401       1138    37.86   -122.22
## 3        496        177    37.85   -122.24
## 4        558        219    37.85   -122.25
## 5        565        259    37.85   -122.25
## 6        413        193    37.85   -122.25
```

```
price.deciles <- quantile(calif$MedianHouseValue, 0:10/10)
cut.prices <- cut(calif$MedianHouseValue, price.deciles, include.lowest = TRUE)
plot(calif$Longitude, calif$Latitude, col = grey(10:2/11)[cut.prices], pch = 20,
xlab = "Longitude", ylab = "Latitude")
```



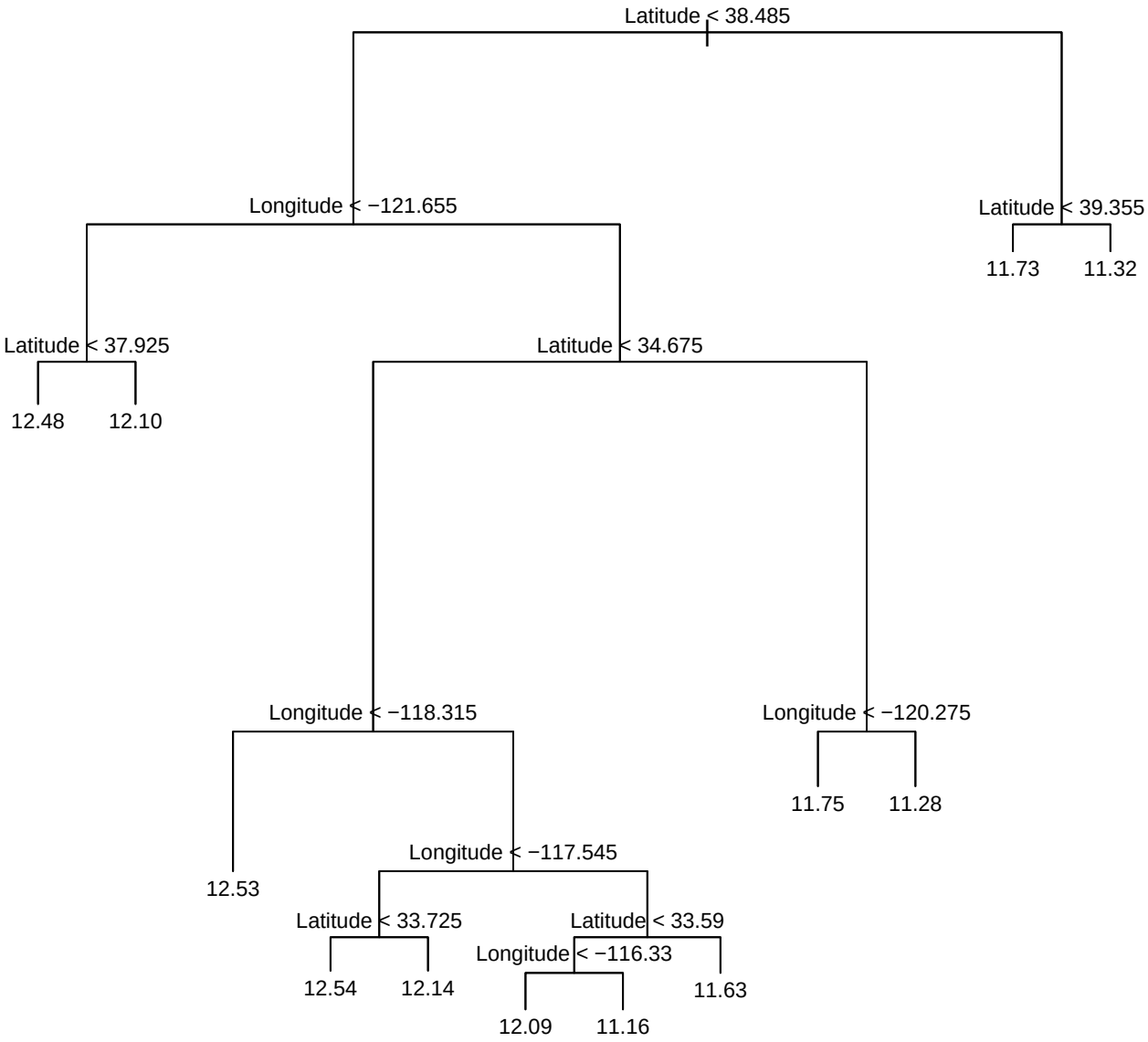
Trees in General

We create a **binary tree**:

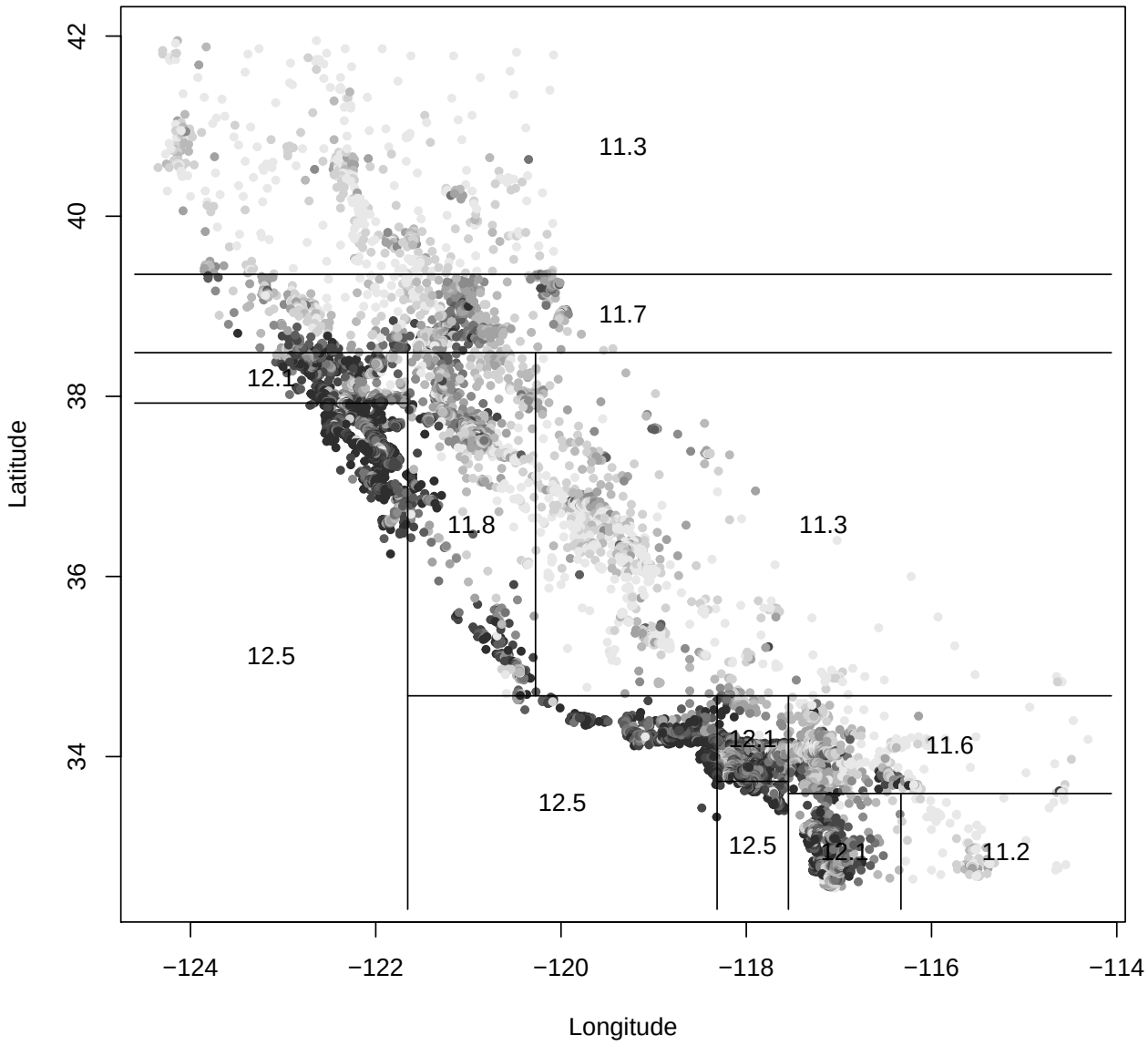
1. Create a **node**. Choose a predictor X_i and split it into two regions, $X_k < s$ and $X_s \geq s$.
2. Within each of those regions, choose a another predictor (or the same one), and split it into two regions.
3. Continue choosing predictor variables and splitting their regions until you reach an end point (you stop creating regions). This is known as a **leaf**.
4. Average the set of Y observations within the set of regions that define a leaf and average them together, (or look at the majority in classification). The average (or majority) is the predicted value of the response.

```
### Libraries !!!
library(tree)

treefit <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data = calif)
```

```
plot(treefit)
text(treefit, cex = 0.75)
```



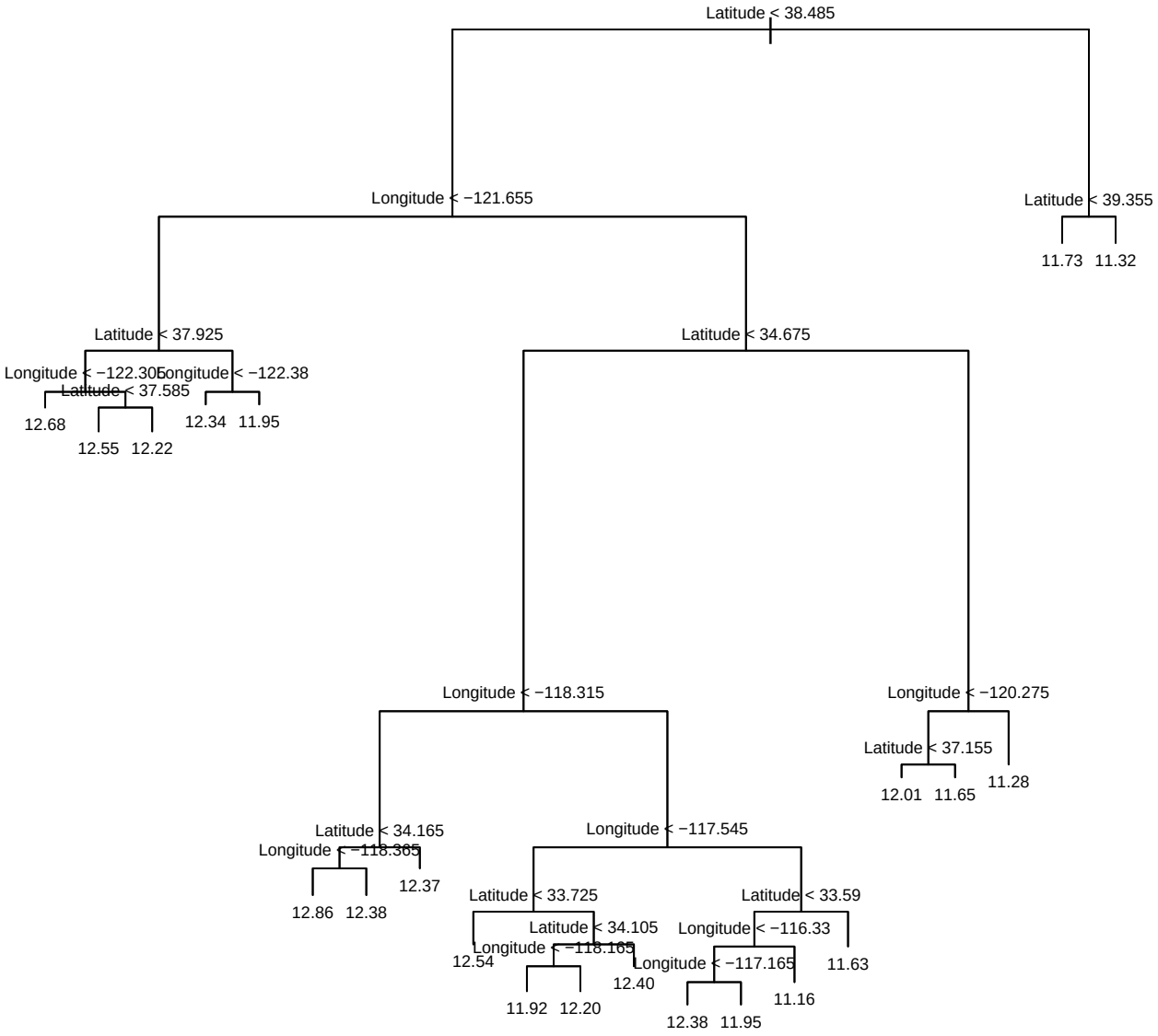

```
price.deciles <- quantile(calif$MedianHouseValue, 0:10/10)
cut.prices <- cut(calif$MedianHouseValue, price.deciles, include.lowest = TRUE)
plot(calif$Longitude, calif$Latitude, col = grey(10:2/11)[cut.prices], pch = 20,
xlab = "Longitude", ylab = "Latitude")
partition.tree(treefit, ordvars = c("Longitude", "Latitude"), add = TRUE)
```



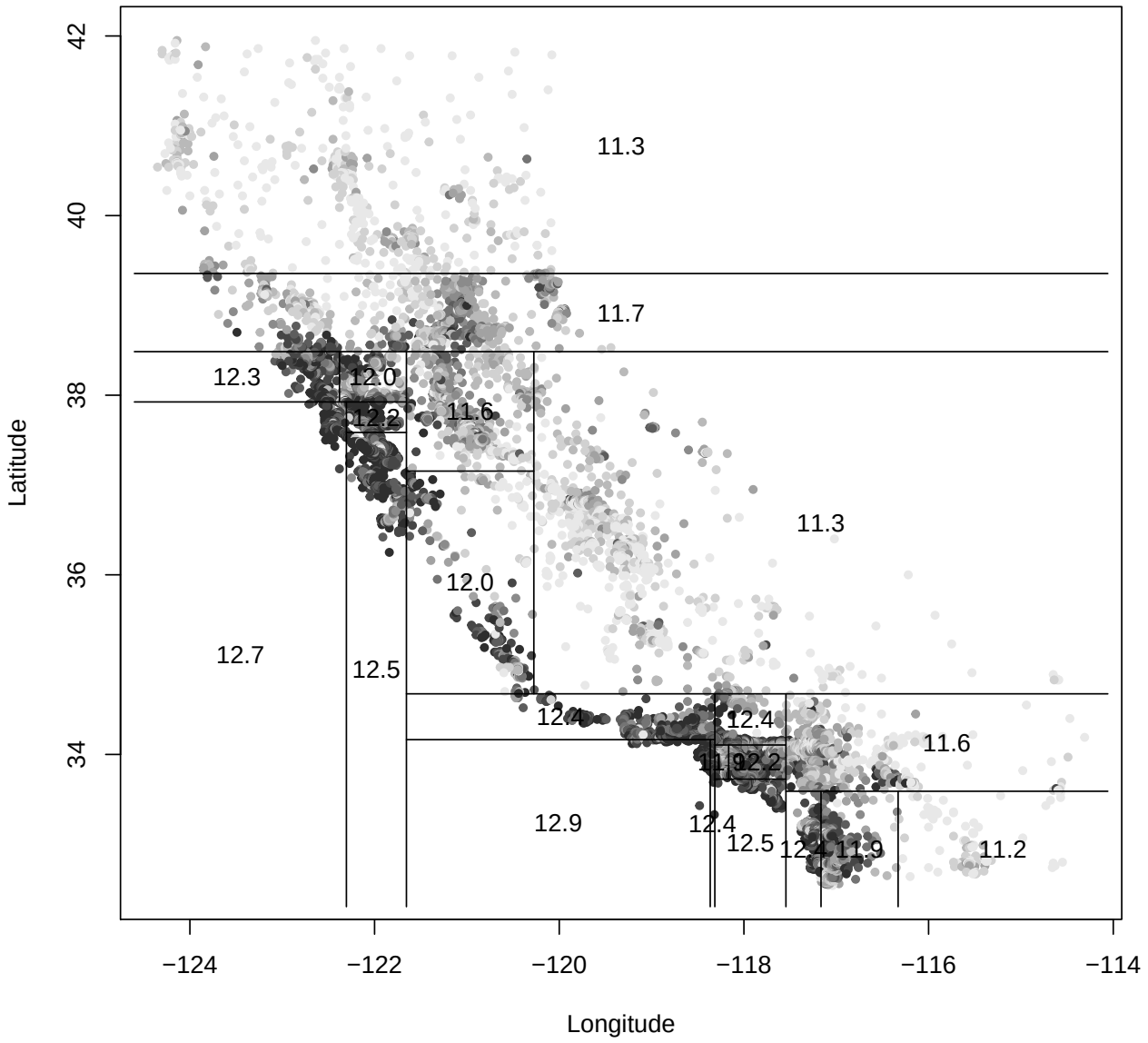
The flexibility of trees is going to be based on the number of nodes. More nodes means we are dividing the predictor space into small and smaller pieces.

More Nodes

```
treefit2 <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data = calif,
mindev = 0.005)

plot(treefit2)
text(treefit2, cex = 0.6)
```



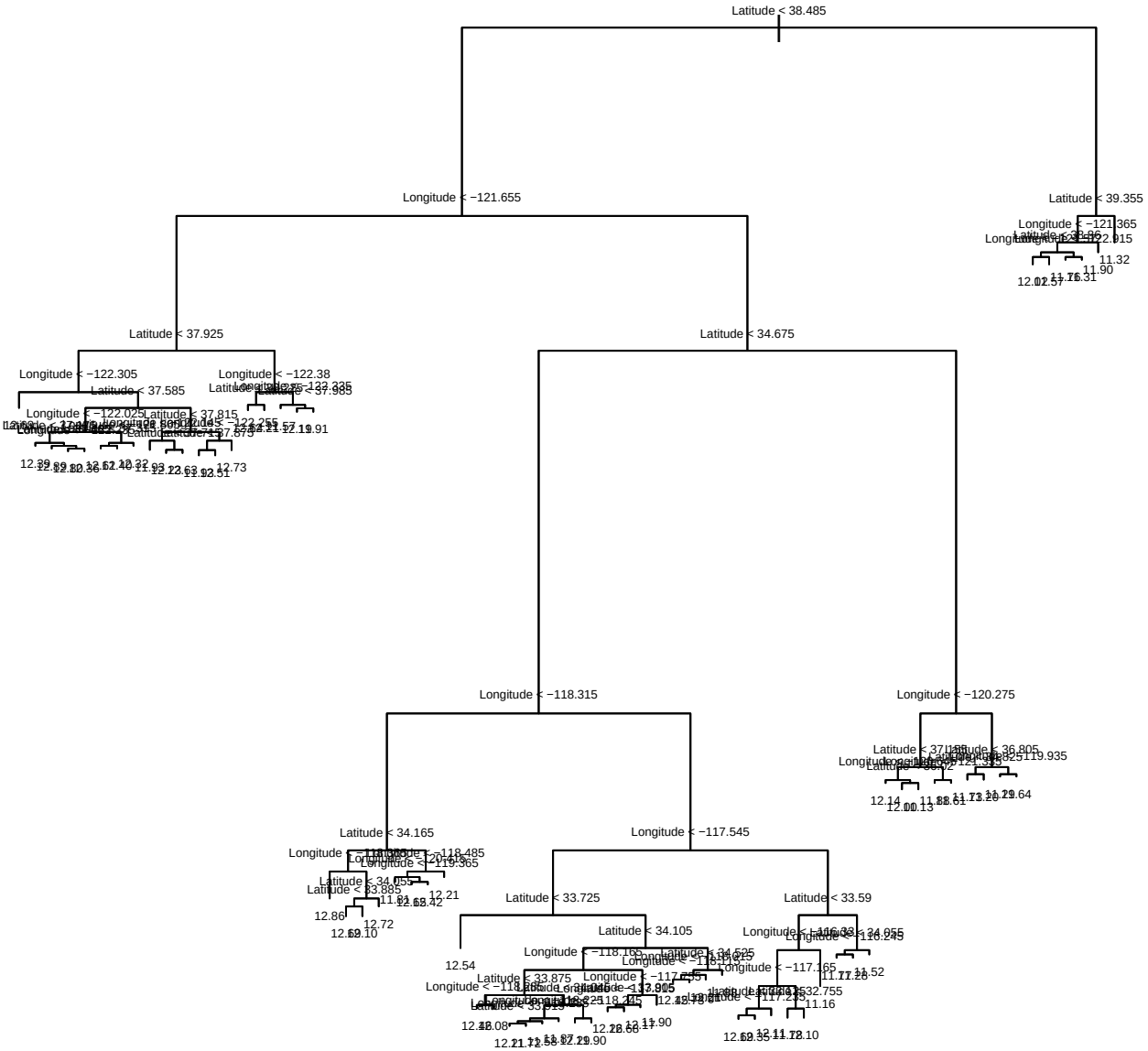

```
plot(calif$Longitude, calif$Latitude, col = grey(10:2/11)[cut.prices], pch = 20,
xlab = "Longitude", ylab = "Latitude")
partition.tree(treefit2, ordvars = c("Longitude", "Latitude"), add = TRUE)
```



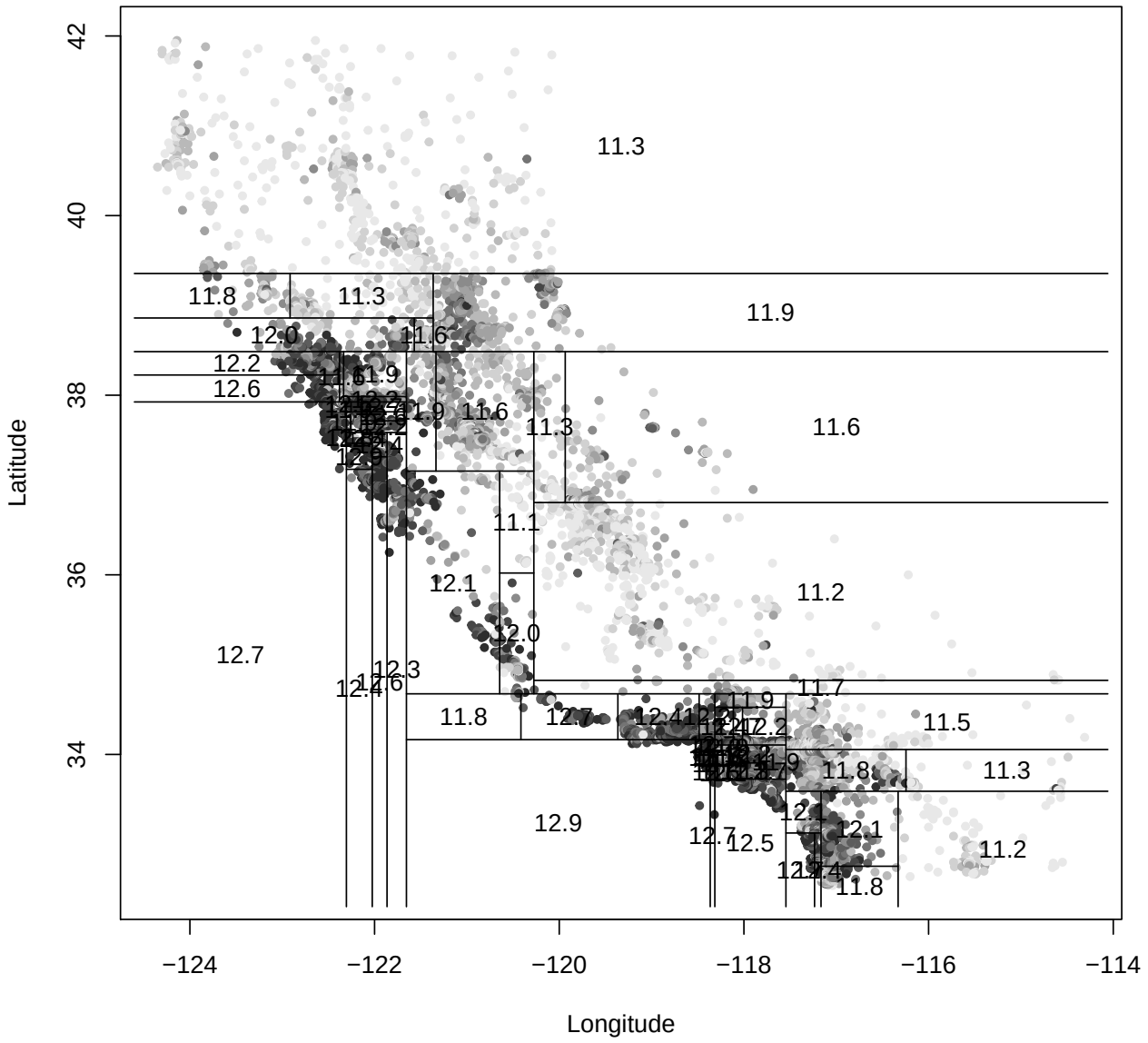

```
# treefit2
```

And Even More!

```
treefit3 <- tree(log(MedianHouseValue) ~ Longitude + Latitude, data = calif,
mindev = 0.001)

plot(treefit3)
text(treefit3, cex = 0.4)
```




```
plot(calif$Longitude, calif$Latitude, col = grey(10:2/11)[cut.prices], pch = 20,
xlab = "Longitude", ylab = "Latitude")
partition.tree(treefit3, ordvars = c("Longitude", "Latitude"), add = TRUE)
```

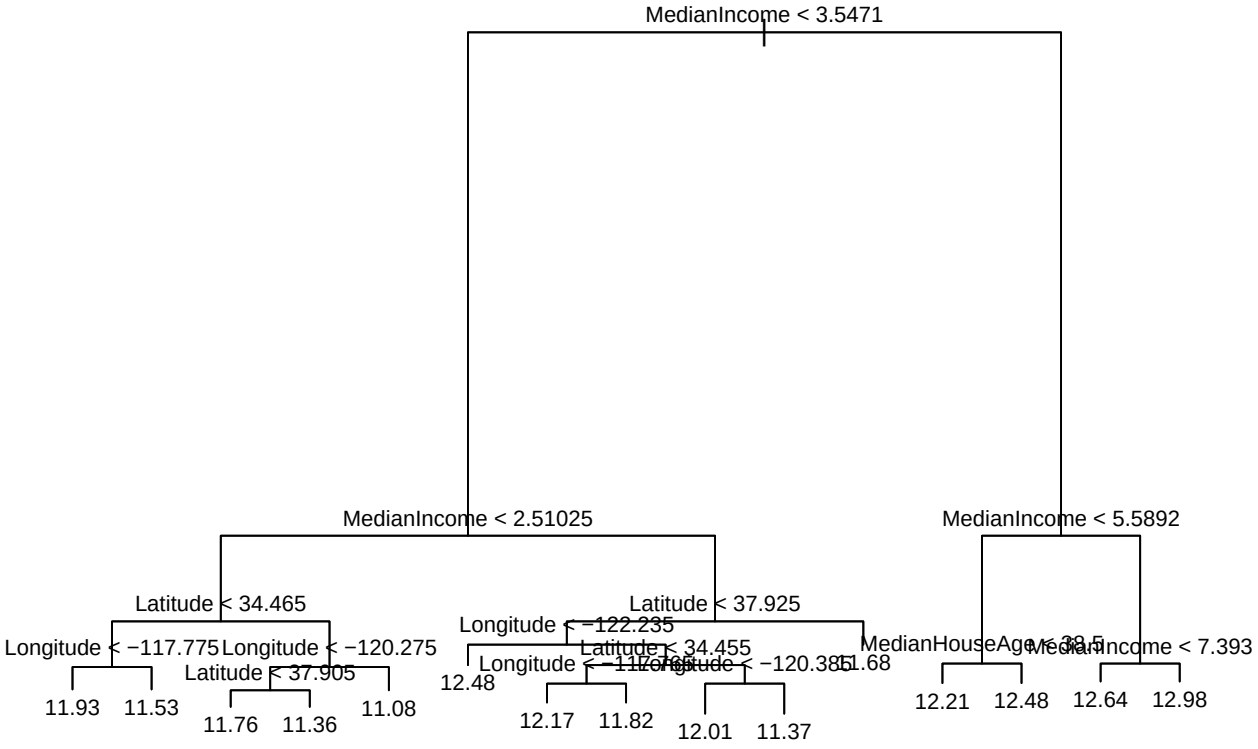



```
# treefit3
```

More Predictors

We are not restricted to variables that literally divide based on geometric regions.

```
treefit4 <- tree(log(MedianHouseValue) ~ ., data = calif)

plot(treefit4)
text(treefit4, cex = 0.75)
```



Getting predictions

1. We divide the predictor space (the set of possible values for $X_1, \ldots, X_p$) into M distinct and *non-overlapping* regions: $R_1, R_2, \ldots, R_M$
2. For every observation that falls into region R_m, we make the same prediction, which is simply the mean of the response values for the training observations in R_m

But, how do we choose the regions?

- Regions could be any shape, in theory. They just have to not overlap for the algorithm to work.
- We instead choose to divide the predictor space into *high-dimensional rectangles*. Just call them boxes.
- The goal would be to choose the boxes $R_1, \ldots, R_M$ to minimize:

$$RSS = \sum_{m=1}^{M} \sum_{x_i \in R_m} (y_i - \widehat{y}_{R_m})^2$$

Trees Versus Linear Models

Linear regression assumes the form:

$$f(\underline{X}) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j$$

Whereas regression trees assume the form:

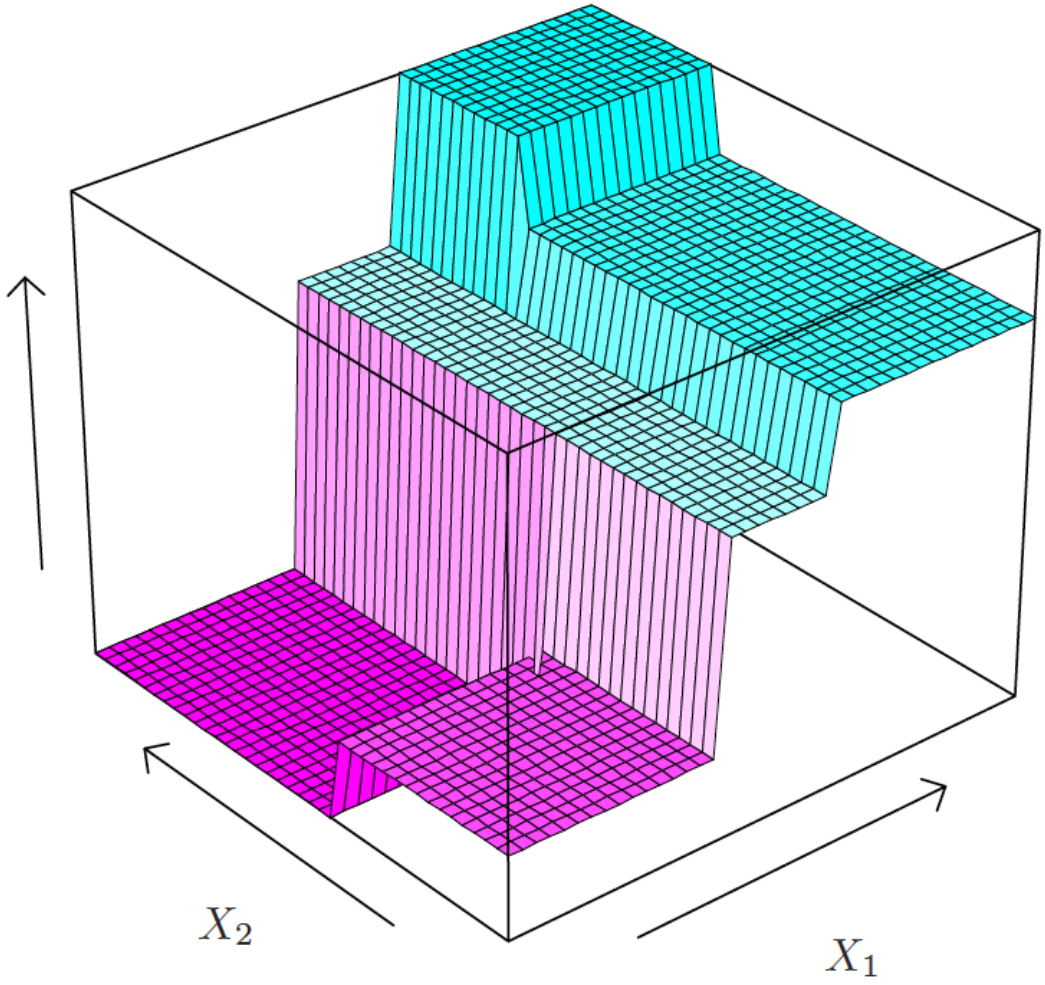


Figure 1:

$$f(\underline{X}) = \sum_{m=1}^{M} c_m \cdot I(\underline{X} \in R_m)$$

Recursive Binary Splitting

It is computationally infeasible to consider *every* possible choice of regions.

We instead take the *top down* approach described earlier. We call this process **Recursive Binary Splitting**

1. For each predictor X_j choose a cutpoint s_j such that splitting based off the regions $X_j < s_j$ and $X_j \geq s_j$ that results in the greatest possible reduction in the RSS for that predictor.
2. Use the predictor X_j and cutpoint s_j that resulted in the greatest reduction in RSS among the predictors and other cutpoints.

3. Repeat within each region, i.e. within the region $X_j < s$ and $X_j \geq s$. For each predictor, find the cutpoint. Choose predictor with cutpoint that reduces RSS by most.
4. Stop until all Y observations in a region are the same value, or there is a small number of response variables within each region, or the change in RSS is below some stopping criterion.

We choose a single variable to make a cut without looking at the any future cuts. This is aspect of the process is referred to as **greedy**.

Greedy means we try to grab as much reduction in the RSS as possible at a single cut within the tree instead of trying to reduce the RSS across all possible trees.

This greedy approach will lead to trees with too many leaves that overfits to the training data.

Tree Pruning

One approach to this issue of *greediness* is to take an initial large (many branches) tree T_0 and remove or **prune** the branches of T_0 until we end up with a smaller subtree T that minimizes a penalized RSS.

$$RSS(\alpha) = \sum_{m=1}^{M} \sum_{x_i \in R_m} (y_i - \widehat{y}_{R_m})^2 + \alpha |T|$$

- $|T|$ is the number of leaves in the subtree T.

What happens if $\alpha = 0$?

As we increase α we penalize a tree more and more for the number of leaves that it has. This reduces the number of branches or *prunes* the tree in a nested and predictable manner.

This process is referred to as **cost complexity pruning**.

1. Use recursive binary splitting to grow a large tree on the training data. Stop when each termminal node has fewer than some minimum number of observations.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.
3. Use K-fold of cross-validation to choose α. Divide the training observations into K folds. For each $k = 1, \ldots, K$:

- Repeat steps 1 and 2 on all but the kth fold of the training data.
- Evalute the mean squared prediction error on the data in the left-out k fold, as a function of α.
- Average the results for each value of α and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

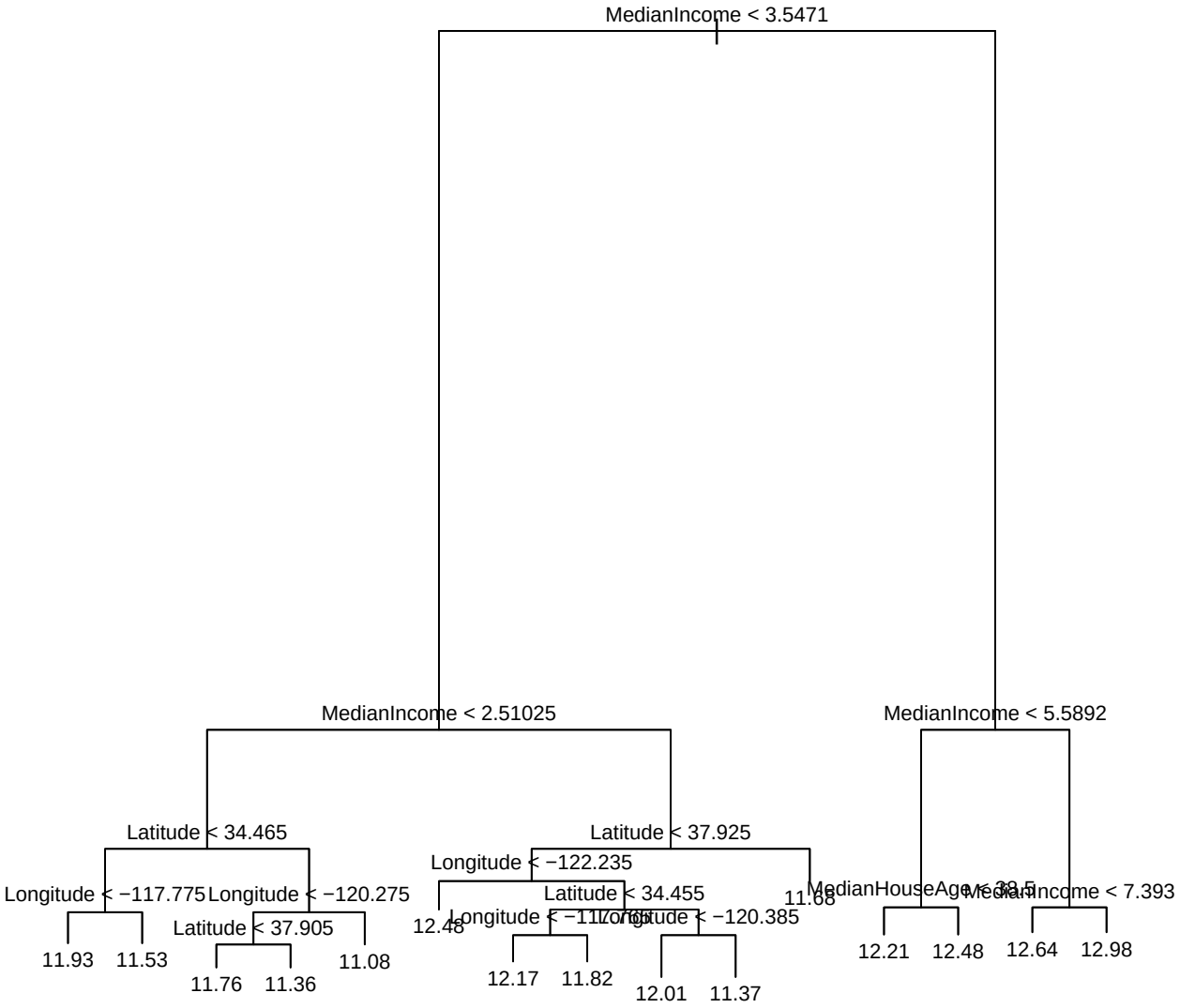
In practice, this is automatically done by the function `cv.tree` from `library(tree)`.

```
calitree <- tree(log(MedianHouseValue) ~ ., data = calif)
summary(calitree)
```

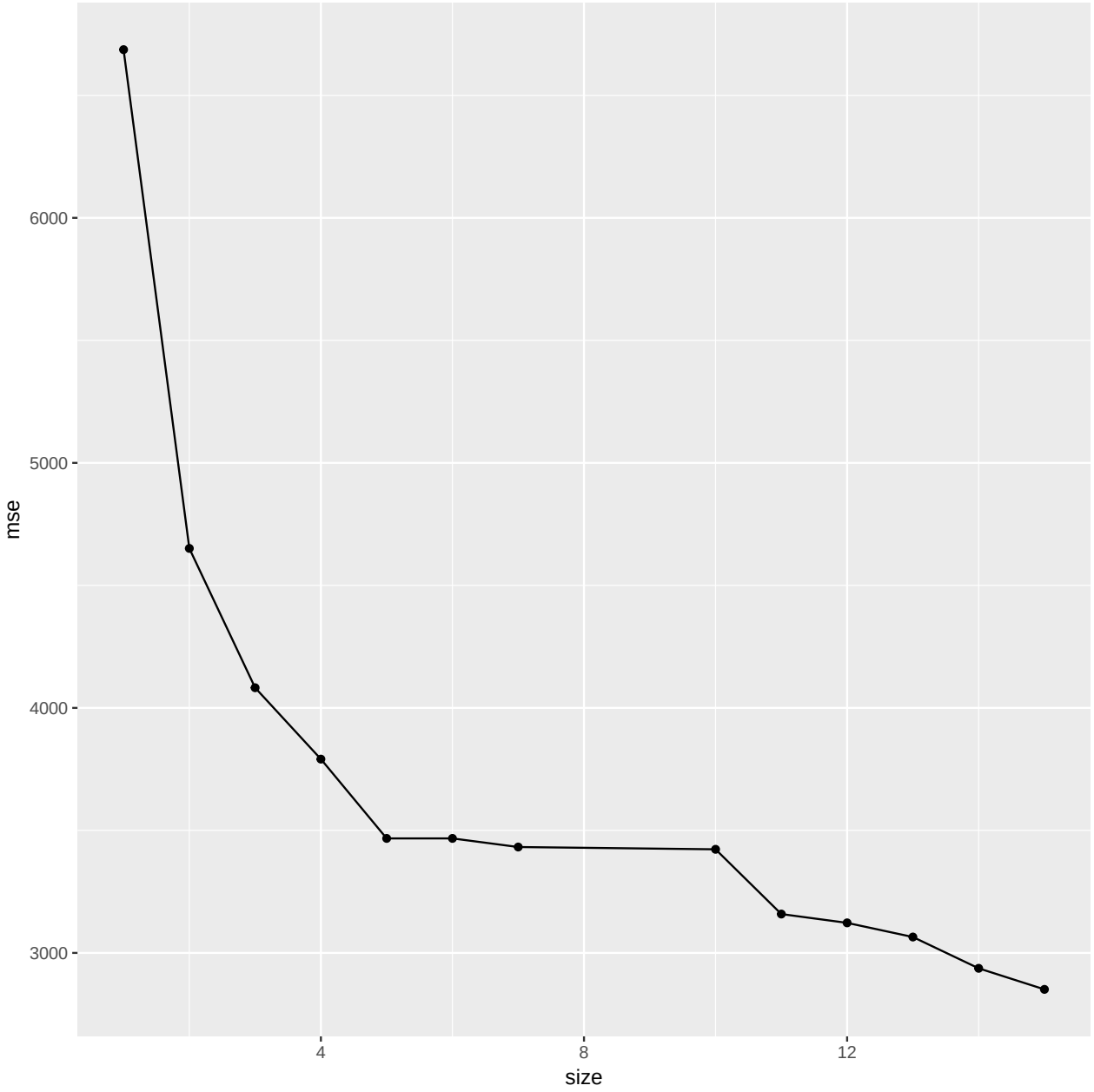
```
##
## Regression tree:
## tree(formula = log(MedianHouseValue) ~ ., data = calif)
## Variables actually used in tree construction:
## [1] "MedianIncome"   "Latitude"       "Longitude"      "MedianHouseAge"
## Number of terminal nodes:  15
## Residual mean deviance:  0.1321 = 2724 / 20620
## Distribution of residuals:
```

```
##      Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.86000 -0.22650 -0.01475  0.00000  0.20740  2.03900
```

```
plot(calitree)
text(calitree, cex = 0.75)
```



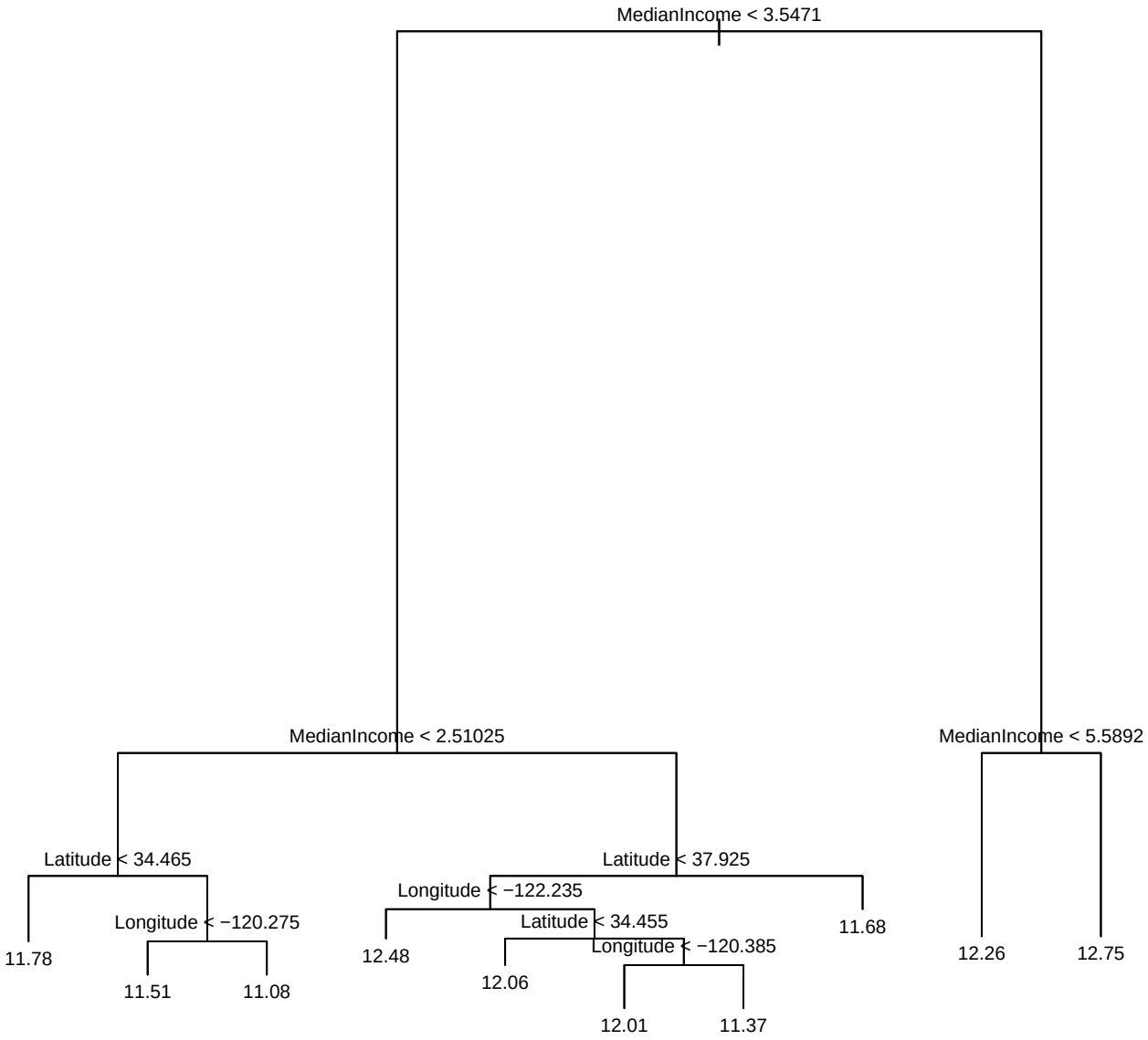

```
cv.cali = cv.tree(calitree)

cv.df <- data.frame(size = cv.cali$size, mse = cv.cali$dev)
ggplot(cv.df, aes(x = size, y = mse)) + geom_line() + geom_point()
```



In the case of this data, the most complex tree was chosen.

We can force pruning.

```
prune.cali = prune.tree(calitree, best = 10)
plot(prune.cali)
text(prune.cali, cex = 0.7)
```




```
summary(prune.cali)
```

```
##
## Regression tree:
## snip.tree(tree = calitree, nodes = c(18L, 7L, 42L, 6L, 8L))
## Variables actually used in tree construction:
## [1] "MedianIncome" "Latitude"     "Longitude"
## Number of terminal nodes:  10
## Residual mean deviance:  0.1509 = 3112 / 20630
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -2.64100 -0.25140 -0.01448  0.00000  0.25220  2.03900
```

```
prune.cali
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 20640 6685.00 12.08
```

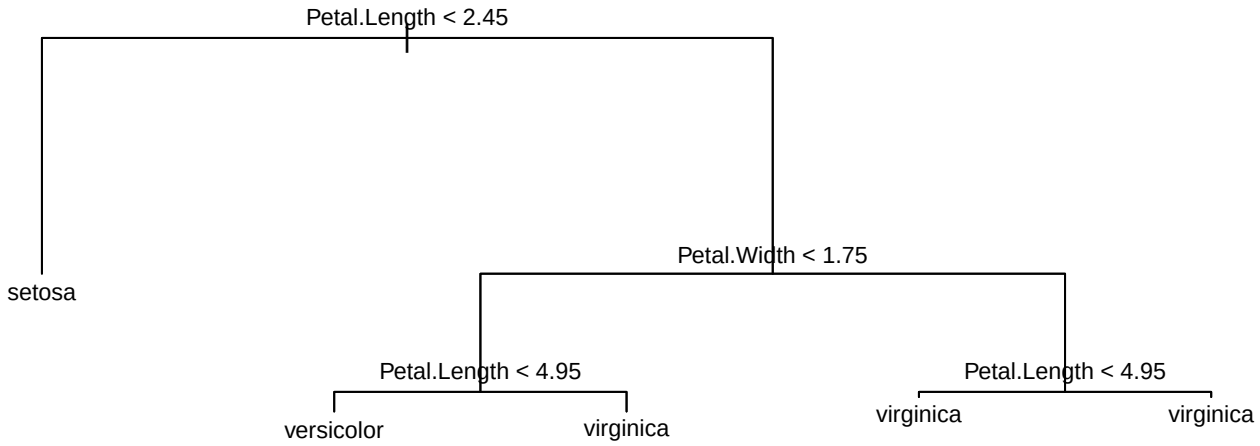
```
##    2) MedianIncome < 3.5471 10381 2662.00 11.77
##      4) MedianIncome < 2.51025 4842 1194.00 11.58
##        8) Latitude < 34.465 2322  450.20 11.78 *
##        9) Latitude > 34.465 2520  557.80 11.39
##         18) Longitude < -120.275 1792  386.00 11.51 *
##         19) Longitude > -120.275 728   77.14 11.08 *
##      5) MedianIncome > 2.51025 5539 1120.00 11.94
##       10) Latitude < 37.925 4435  901.70 12.01
##         20) Longitude < -122.235 351   47.73 12.48 *
##         21) Longitude > -122.235 4084  770.70 11.97
##           42) Latitude < 34.455 2814  410.80 12.06 *
##           43) Latitude > 34.455 1270  284.70 11.77
##             86) Longitude < -120.385 783  112.10 12.01 *
##             87) Longitude > -120.385 487   50.04 11.37 *
##       11) Latitude > 37.925 1104  123.70 11.68 *
##    3) MedianIncome > 3.5471 10259 1975.00 12.40
##      6) MedianIncome < 5.5892 7265 1156.00 12.26 *
##      7) MedianIncome > 5.5892 2994  298.70 12.75 *
```

Classification Trees

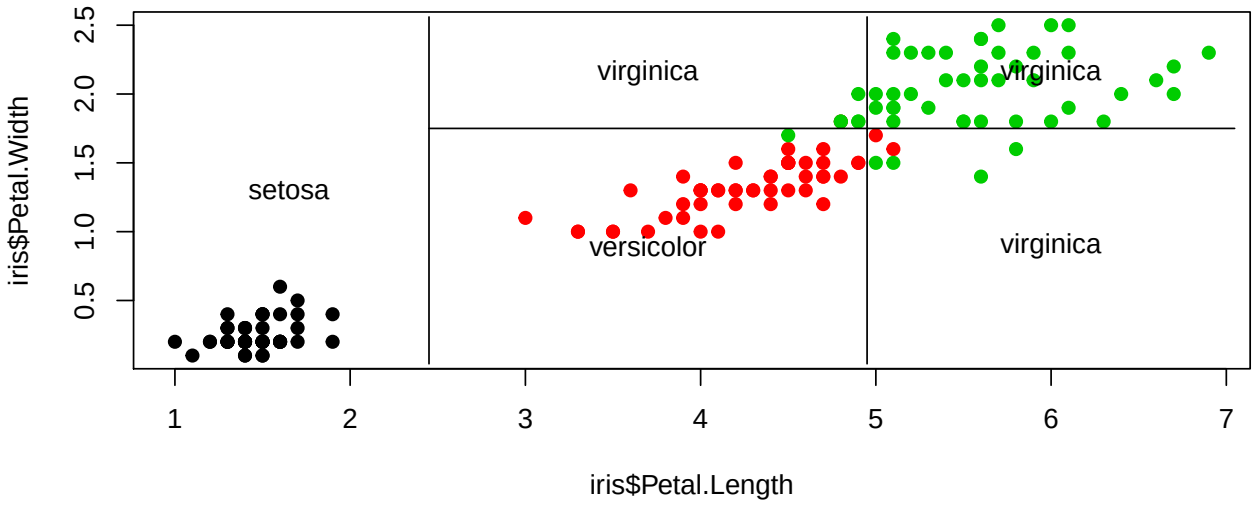
Trees "easily" adapt to classification problems. The only real difference in the outcome is going to be now the response is categorical instead of continuous.

- The predicted value of the response of observations is is the most commonly occuring class in a leaf.
- Branches are created in the same manner, we just have to adjust what we use for the RSS.

```
data("iris")

iris.tree <- tree(Species ~ Petal.Length + Petal.Width, data = iris)

plot(iris.tree)
text(iris.tree, cex = 0.75)
```




```
plot(iris$Petal.Length,iris$Petal.Width,pch=19,col=as.numeric(iris$Species))
partition.tree(iris.tree,label="Species",add=TRUE)
legend(1.75,4.5,legend=unique(iris$Species),col=unique(as.numeric(iris$Species)))
```



Error Criterion

There are many choices for a metric. Let p_{mk} be the proportion of training observations in the m^{th} region that are from the k^{th} class.

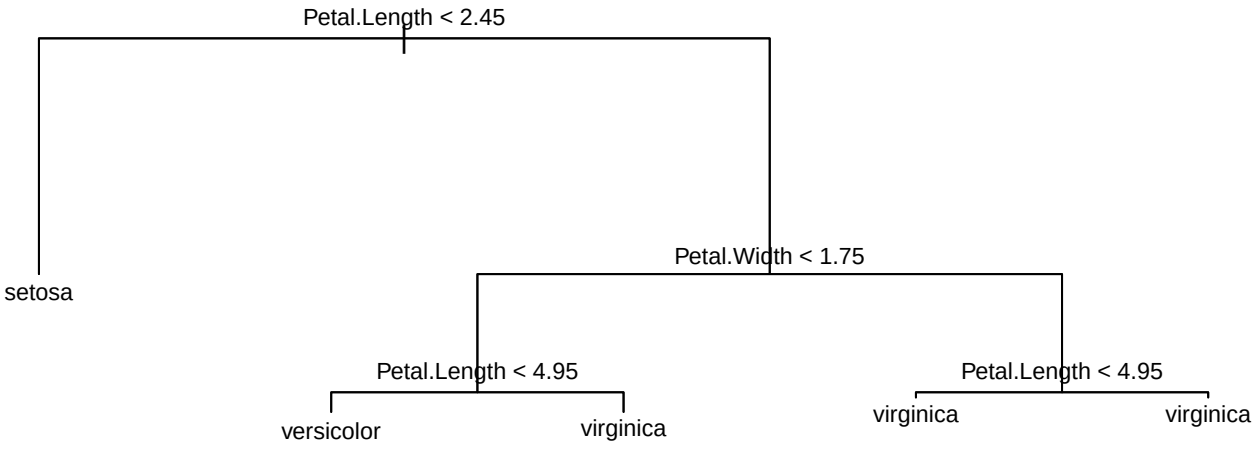
classification error rate:	$E = 1 - \max_k(\widehat{p}_{mk})$
Gini index:	$G = \sum_k \widehat{p}_{mk}(1 - \widehat{p}_{mk})$
cross-entropy:	$D = -\sum_k \widehat{p}_{mk} \log(\widehat{p}_{mk})$

Both Gini and cross-entropy can be thought of as measuring the purity of the classifier (small if all p_{mk} are near zero or 1). These are preferred over the classification error rate.

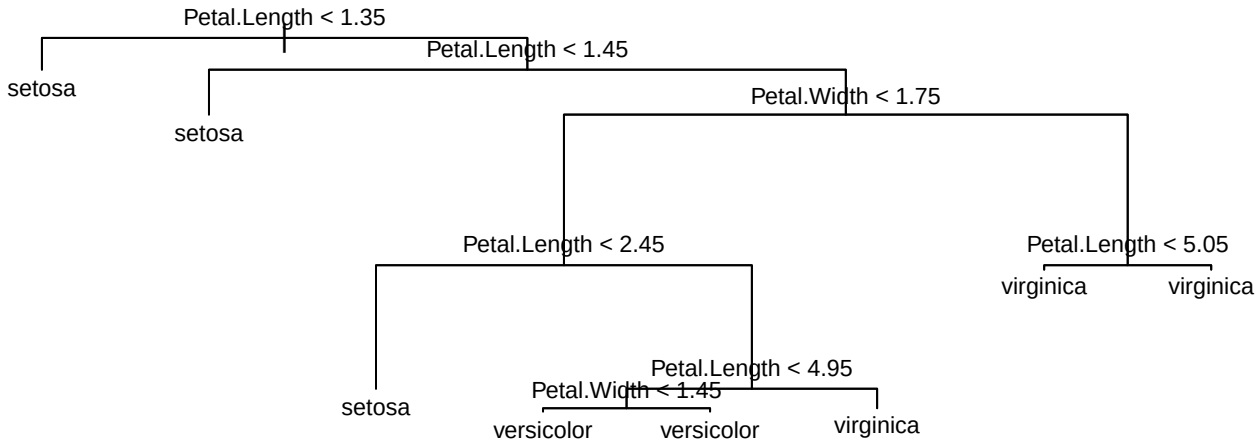
We build a classifier by growing a tree that minimizes E, G or D.

`R` will default to a deviance that is similar to the entropy criterion (`deviance`) but can also use the Gini index

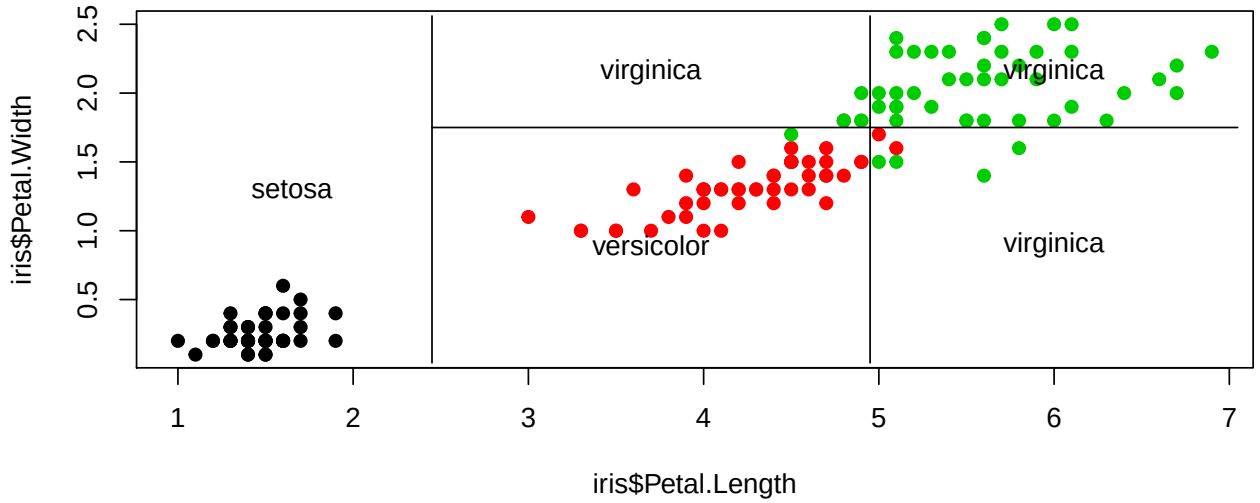
```
iris.tree.ent <- tree(Species ~ Petal.Length + Petal.Width, data = iris, split = "deviance")
iris.tree.gini <- tree(Species ~ Petal.Length + Petal.Width, data = iris, split = "gini")

plot(iris.tree.ent)
text(iris.tree.ent, cex = 0.75)
```



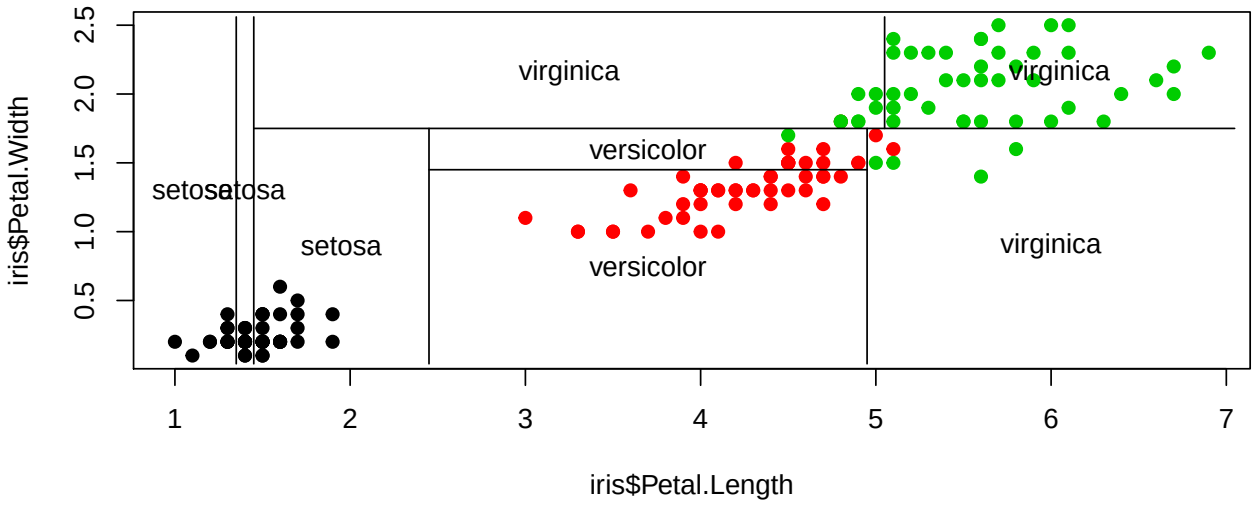
```
plot(iris.tree.gini)
text(iris.tree.gini, cex = 0.75)
```




```
plot(iris$Petal.Length,iris$Petal.Width,pch=19,col=as.numeric(iris$Species))
partition.tree(iris.tree.ent,label="Species",add=TRUE)
legend(1.75,4.5,legend=unique(iris$Species),col=unique(as.numeric(iris$Species)))
```




```
plot(iris$Petal.Length,iris$Petal.Width,pch=19,col=as.numeric(iris$Species))
partition.tree(iris.tree.gini,label="Species",add=TRUE)
legend(1.75,4.5,legend=unique(iris$Species),col=unique(as.numeric(iris$Species)))
```



Same Problem As Before, Same Solutions

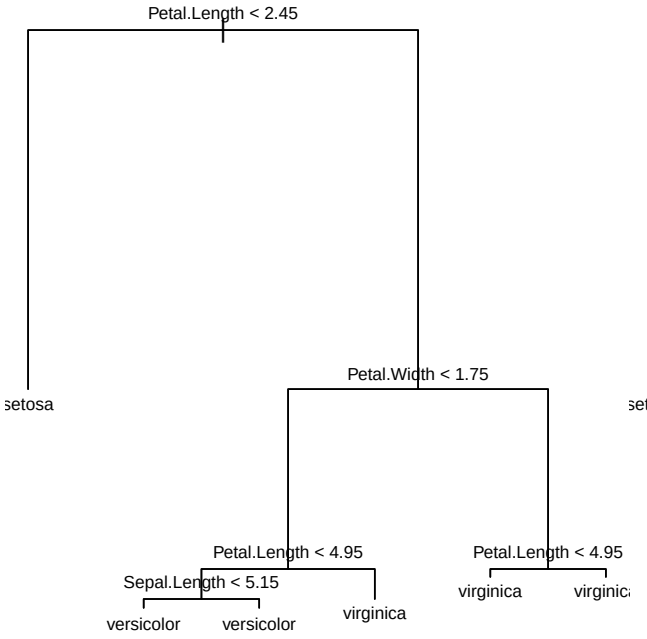
This method, like with regression trees, will tend to overfit the data.

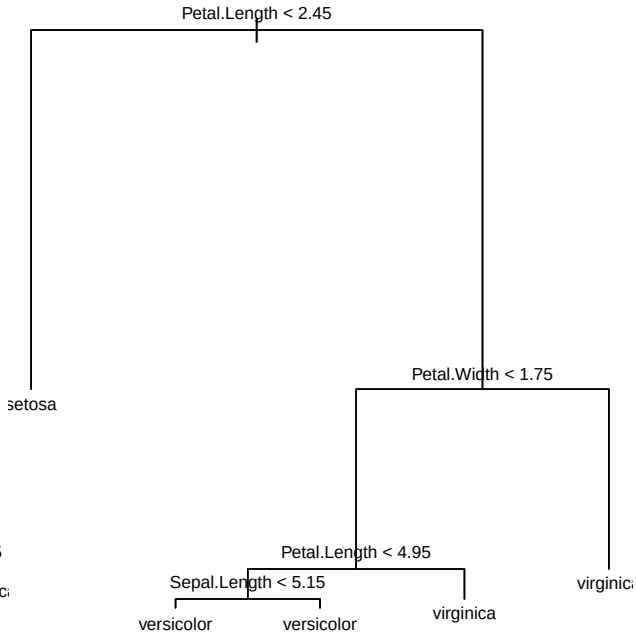
Our solutions are the same as before. **Weakest Link Pruning**

$$\sum_{m=1}^{M} \sum_{x_i \in R_m} (y_i \neq \widehat{y}_{R_m})^2 + \alpha |T|$$

Cross validation is used to choose the best α.

```
set.seed(121)
big.tree = tree(Species ~ ., data = iris)
tree.cv = cv.tree(big.tree, K=5)
best.k = tree.cv$k[which.min(tree.cv$dev)]
pruned = prune.tree(big.tree, k=best.k)

par(mfrow=c(1,2),mar=c(0,0,0,0))
plot(big.tree)
text(big.tree, cex=.7)
plot(pruned)
text(pruned, cex=.7)
```





When Do Trees Do Well

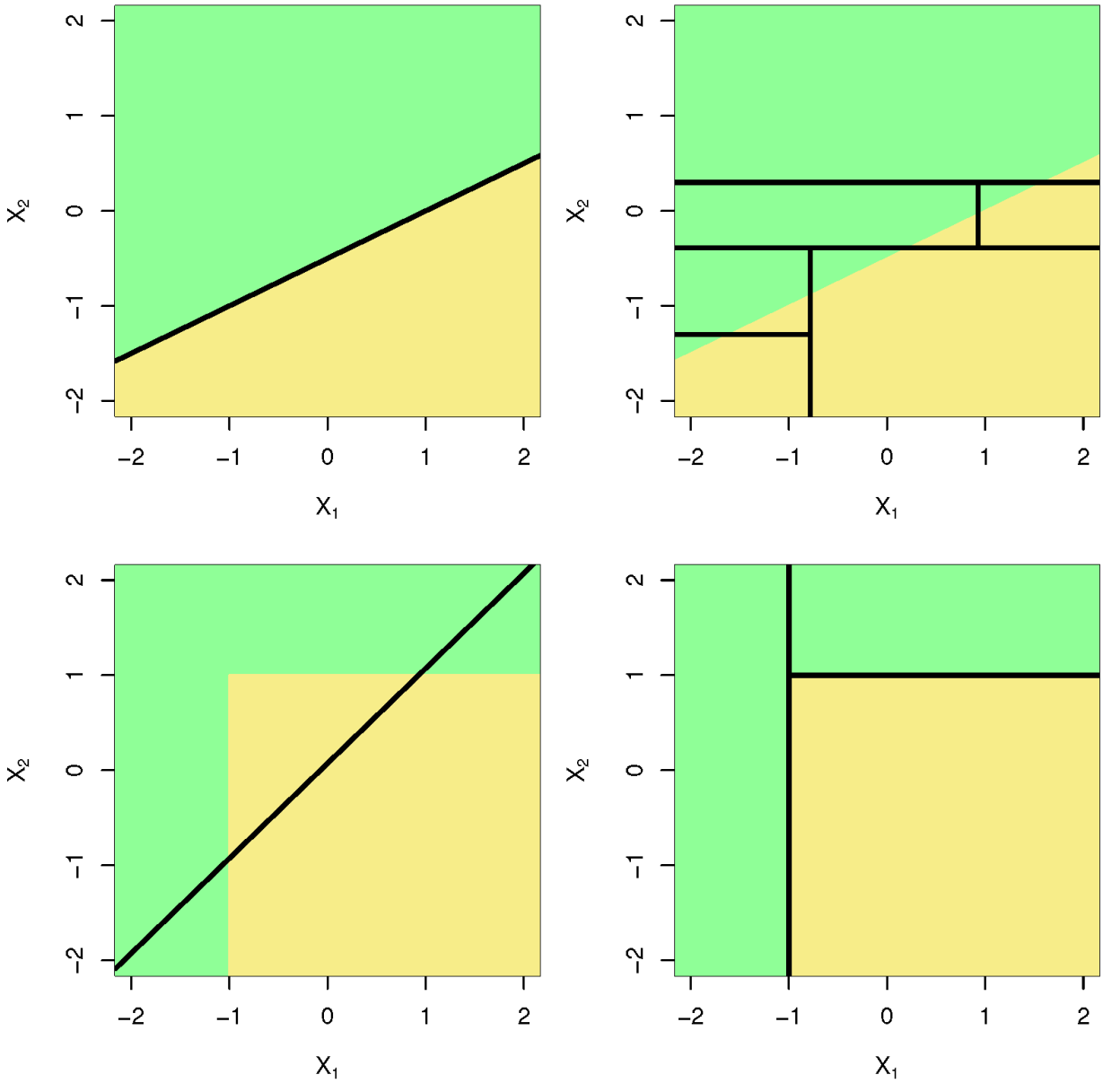


Figure 2: