Tree-based methods for classification and regression

Ryan Tibshirani
Data Mining: 36-462/36-662

April 11 2013

Optional reading: ISL 8.1, ESL 9.2

midterm is DH A302
6:30 - 8:30 pm

Tree-based methods

Tree-based based methods for predicting y from a feature vector $\boldsymbol{x} \in \mathbb{R}^p$ divide up the feature space into rectangles, and then fit a very simple model in each rectangle. This works both when y is discrete and continuous, i.e., both for classification and regression

Rectangles can be achieved by making successive binary splits on the predictors variables $X_1, \ldots X_p$. I.e., we choose a variable X_j, $j = 1, \ldots p$, divide up the feature space according to

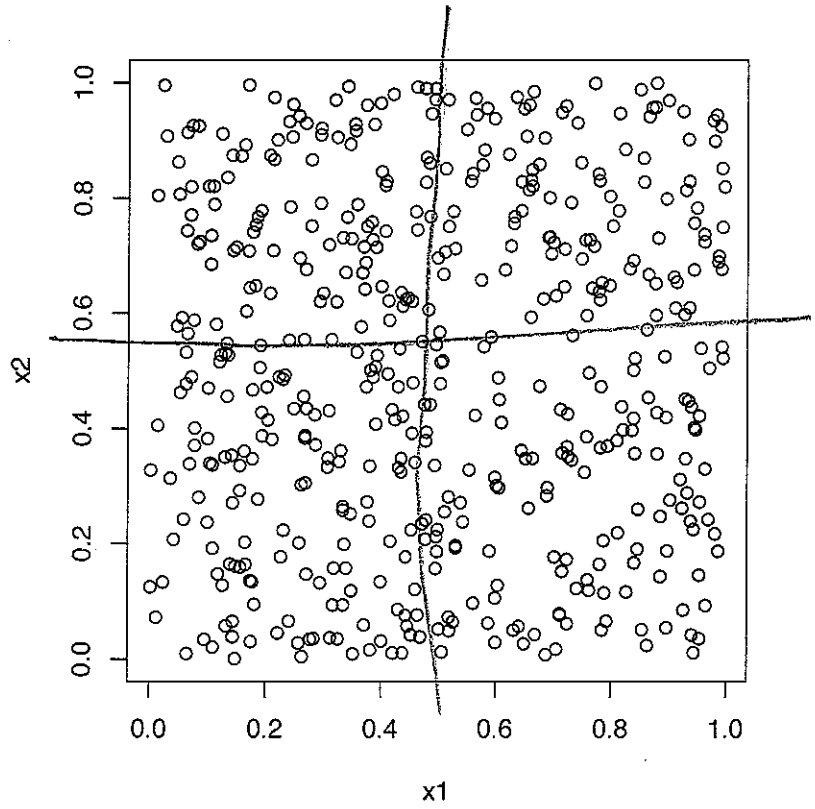
$$X_j \leq c \quad \text{and} \quad X_j > c$$

Then we proceed on each half

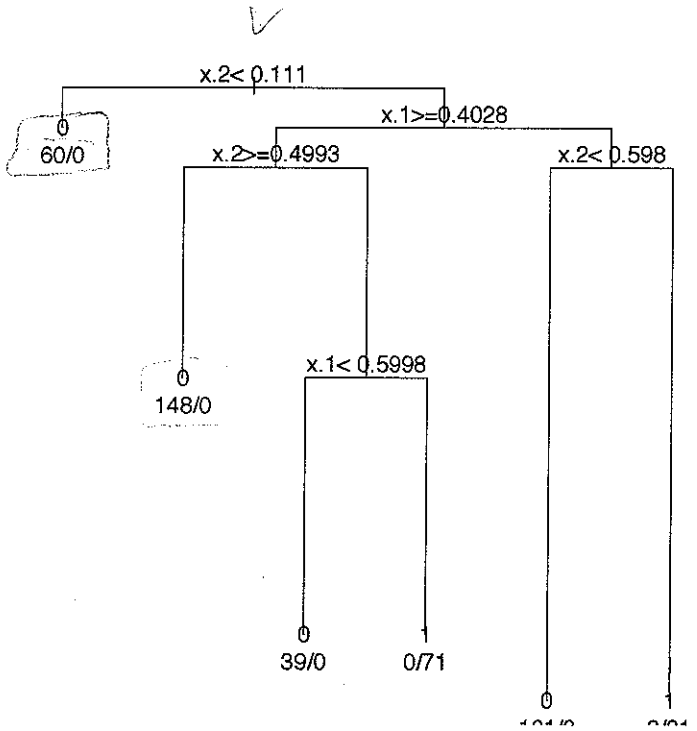
For simplicity, consider classification first (regression later). If a half is "pure", meaning that it mostly contains points from one class, then we don't need to continue splitting; otherwise, we continue splitting

Example: simple classification tree

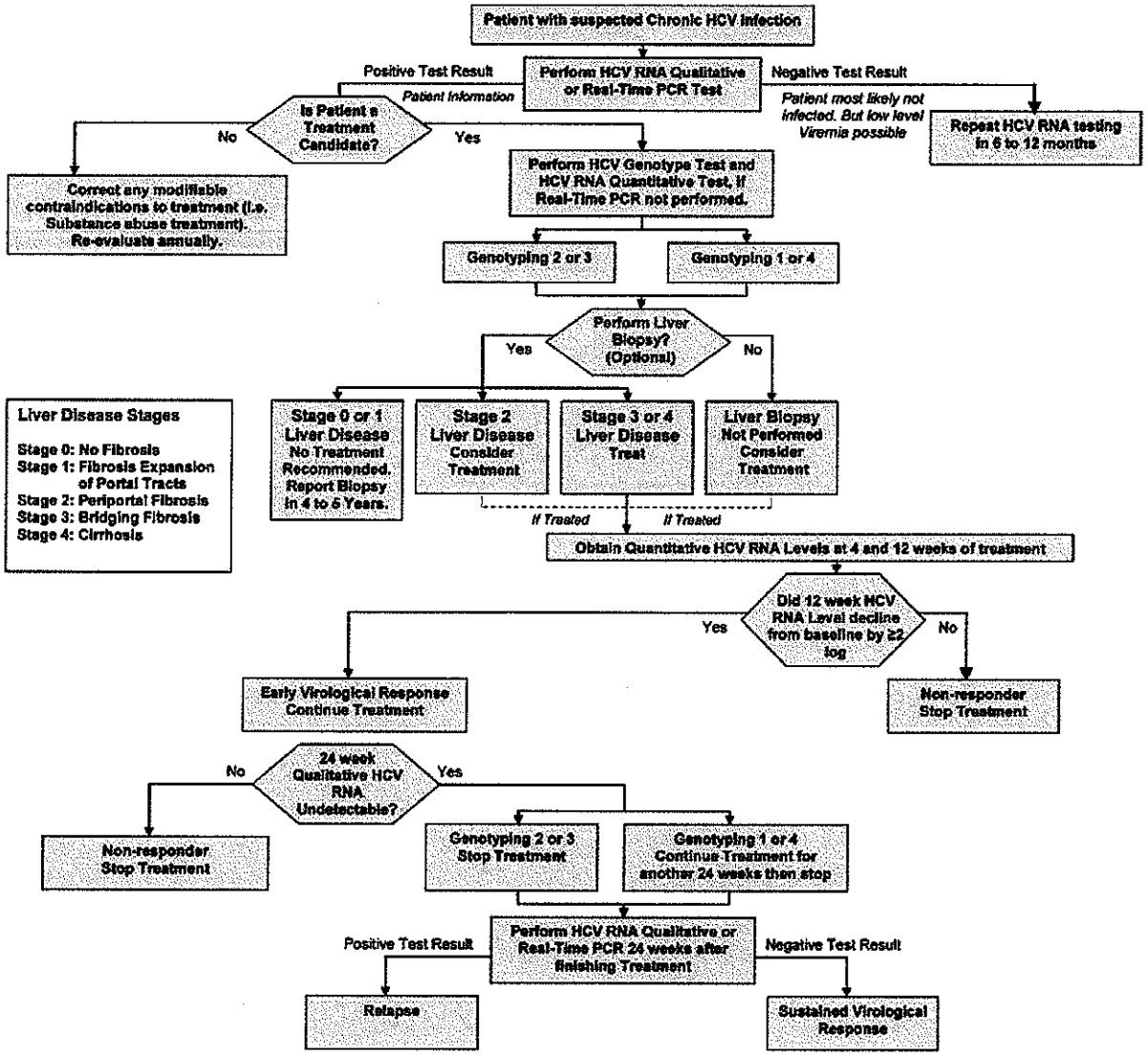
Example: $n = 500$ points in $p = 2$ dimensions, falling into classes 0 and 1, as marked by colors



Does dividing up the feature space into rectangles look like it would work here?


x.2< 0.111
0
60/0
x.1>=0.4028
x.2>=0.4993
x.2< 0.598
0
148/0
x.1< 0.5998
0
39/0
1
0/71
0
1
x1
x2
0.0
0.2
0.4
0.6
0.8
1.0

Example: HCV treatment flow chart



(From `http://hcv.org.nz/wordpress/?tag=treatment-flow-chart`)

Classification trees

Classification trees are popular because they are interpretable, and maybe also because they mimic the way (some) decisions are made

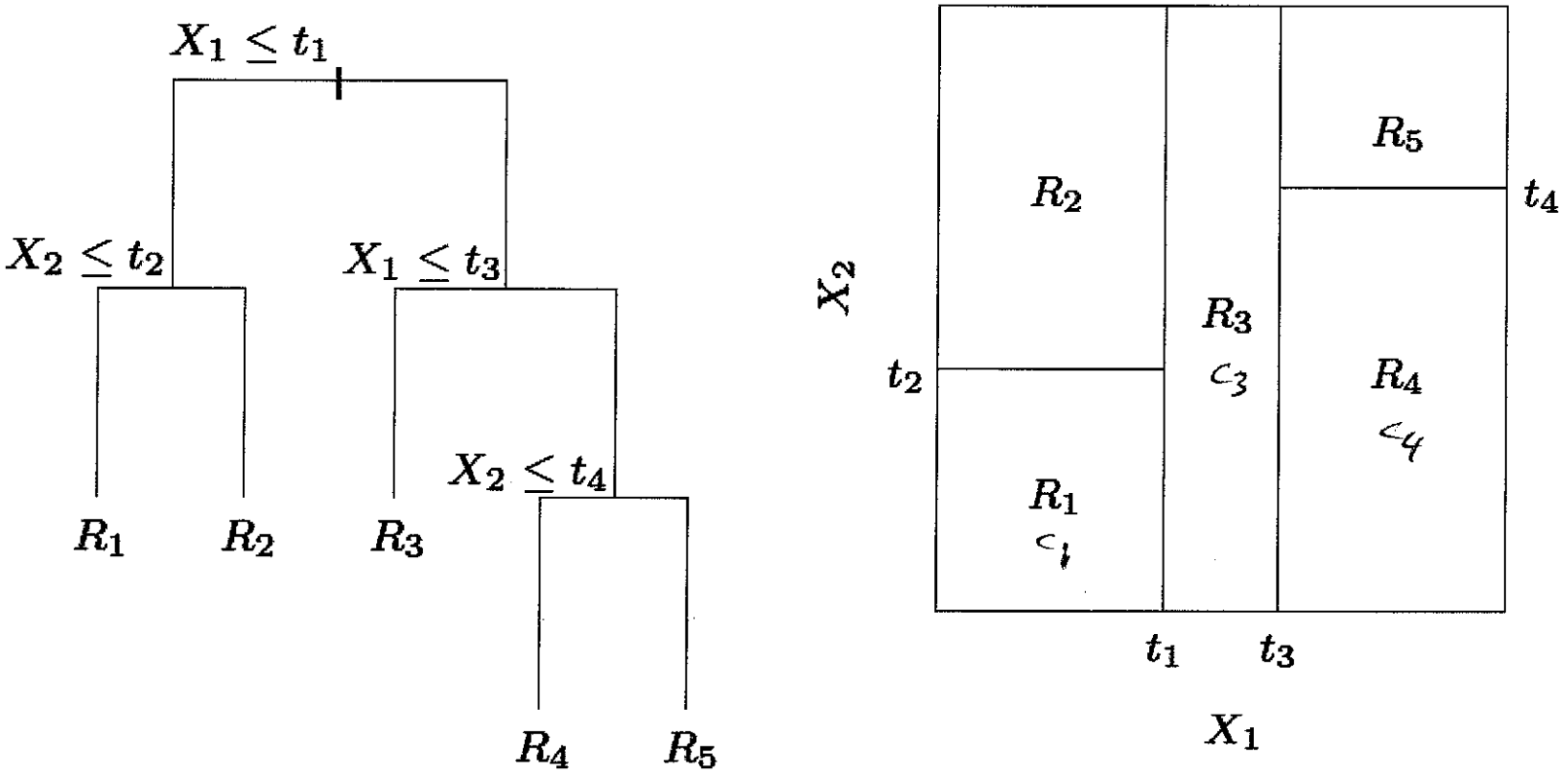
Let (x_i, y_i), $i = 1, \ldots n$ be the training data, where $y_i \in \{1, \ldots K\}$ are the class labels, and $x_i \in \mathbb{R}^p$ measure the p predictor variables. The classification tree can be thought of as defining m regions (rectangles) $R_1, \ldots R_m$, each corresponding to a leaf of the tree

We assign each R_j a class label $c_j \in \{1, \ldots K\}$. We then classify a new point $x \in \mathbb{R}^p$ by

$$\hat{f}^{\text{tree}}(x) \;=\; \sum_{j=1}^{m} c_j \cdot 1\{x \in R_j\} \;=\; c_j \text{ such that } x \in R_j$$

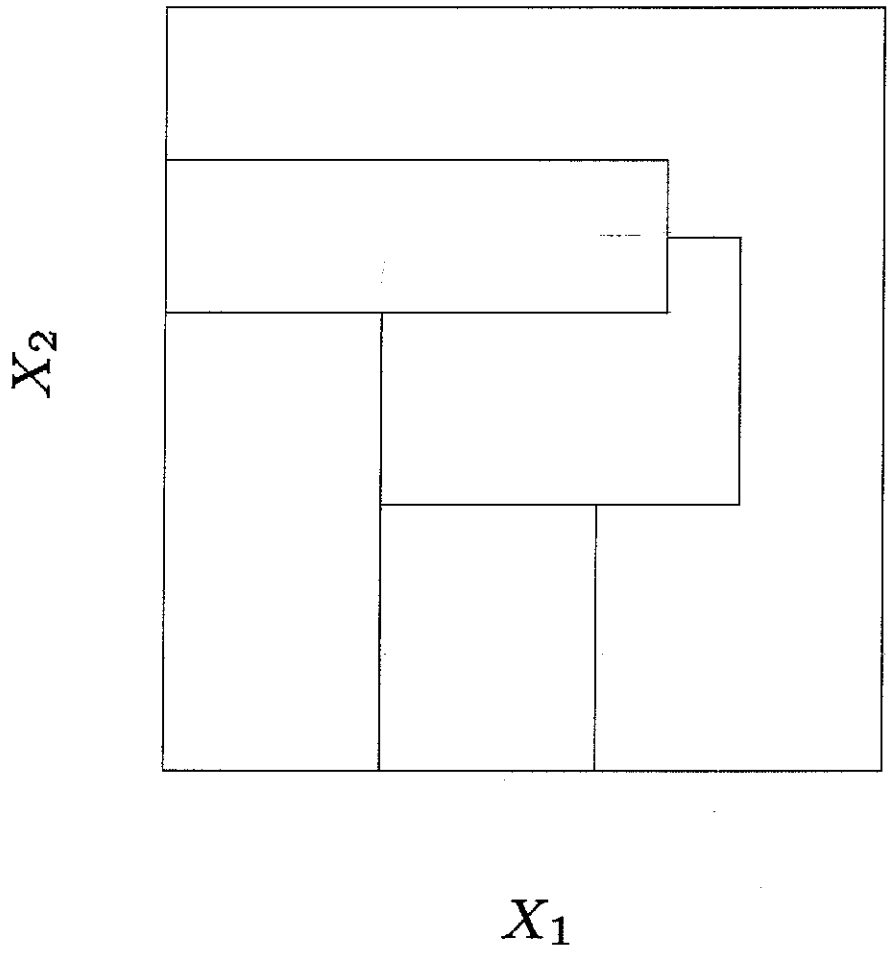
Finding out which region a given point x belongs to is easy since the regions R_j are defined by a tree—we just scan down the tree. Otherwise, it would be a lot harder (need to look at each region)

Example: regions defined by a tree



(From ESL page 306)

Example: regions not defined a tree



(From ESL page 306)

Predicted class probabilities

With classification trees, we can also get not only the predicted classes for new points but also the predicted class probabilties

Note that each region R_j contains some subset of the training data (x_i, y_i), $i = 1, \ldots n$, say, n_j points. The predicted class c_j is just most common occuring class among these points. Further, for each class $k = 1, \ldots K$, we can estimate the probability that the class label is k given that the feature vector lies in region R_j, $\mathrm{P}(C = k | X \in R_j)$, by

$$\hat{p}_k(R_j) = \frac{1}{n_j} \sum_{x_i \in R_j} 1\{y_i = k\}$$

the proportion of points in the region that are of class k. We can now express the predicted class as

$$c_j = \underset{k=1,\ldots K}{\operatorname{argmax}} \ \hat{p}_k(R_j)$$

Trees provide a good balance

	Model assumptions?	Estimated probabilities?	Interpretable?	Flexible?
LDA	Yes strongest	Yes	Yes	No
LR	Yes weaker	Yes	Yes	No
k-NN	No	No	No	Yes
Trees	No	Yes	Yes	Somewhat

	Predicts well?
LDA	Depends on X
LR	Depends on X
k-NN	If properly tuned
Trees	?

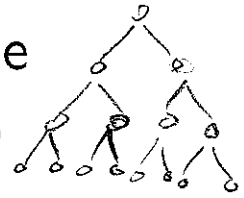
How to build trees?

There are two main issues to consider in building a tree:

1. How to choose the splits?
2. How big to grow the tree?

Think first about varying the depth of the tree ... which is more complex, a big tree or a small tree? What tradeoff is at play here? How might we eventually consider choosing the depth?

Now for a fixed depth, consider choosing the splits. If the tree has depth d (and is balanced), then it has $\approx 2^d$ nodes. At each node we could choose any of p the variables for the split—this means that the number of possibilities is



$$p \cdot 2^d$$

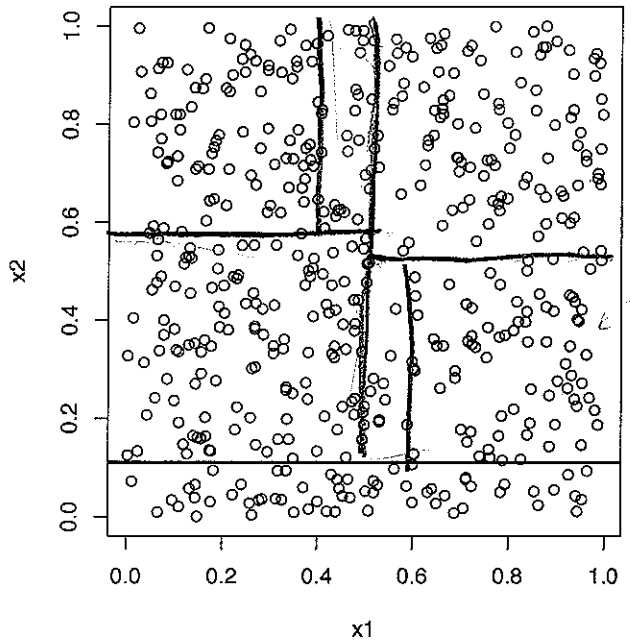
This is huge even for moderate d! And we haven't even counted the actual split points themselves

The CART algorithm

The CART algorithm[1] chooses the splits in a top down fashion: then chooses the first variable to at the root, then the variables at the second level, etc.

At each stage we choose the split to achieve the biggest drop in misclassification error—this is called a greedy strategy. In terms of tree depth, the strategy is to grow a large tree and then prune at the end

Why grow a large tree and prune, instead of just stopping at some point? Because any stopping rule may be short-sighted, in that a split may look bad but it may lead to a good split below it



[1] Breiman et al. (1984), "Classification and Regression Trees"

Recall that in a region R_m, the proportion of points in class k is

$$\hat{p}_k(R_m) = \frac{1}{n_m} \sum_{x_i \in R_m} 1\{y_i = k\}.$$

The CART algorithm begins by considering splitting on variable j and split point s, and defines the regions

$$R_1 = \{X \in \mathbb{R}^p : X_j \leq s\}, \quad R_2 = \{X \in \mathbb{R}^p : X_j > s\}$$

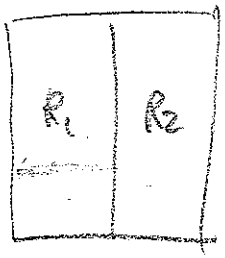
We then greedily chooses j, s by minimizing the misclassification error

$$\underset{j,s}{\text{argmin}} \left(\left[1 - \hat{p}_{c_1}(R_1)\right] + \left[1 - \hat{p}_{c_2}(R_2)\right] \right)$$

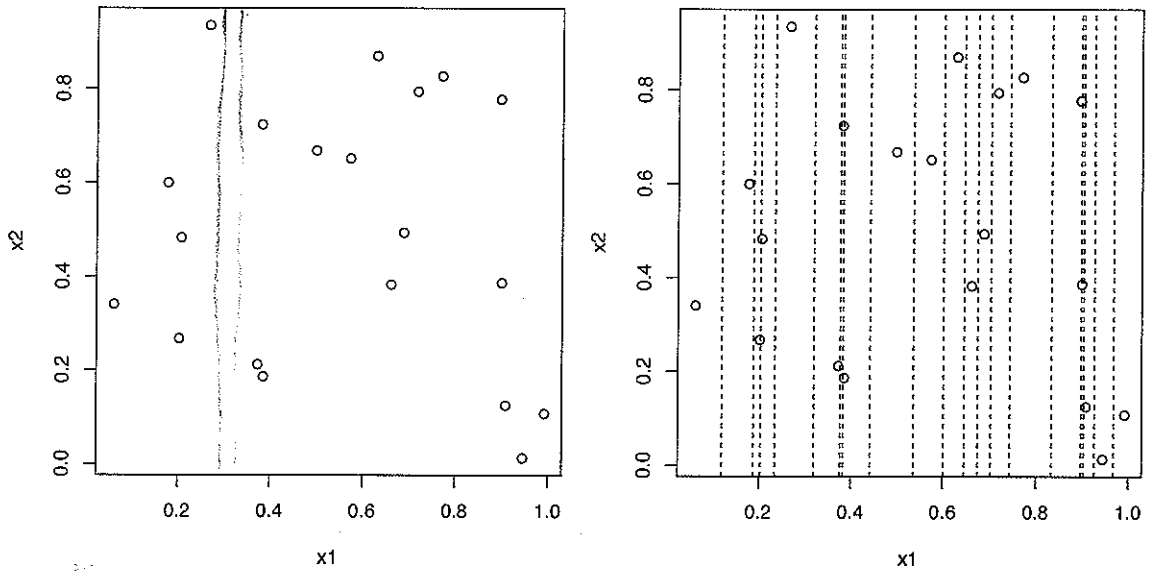
proportion of points in R_1 that aren't of predominant class

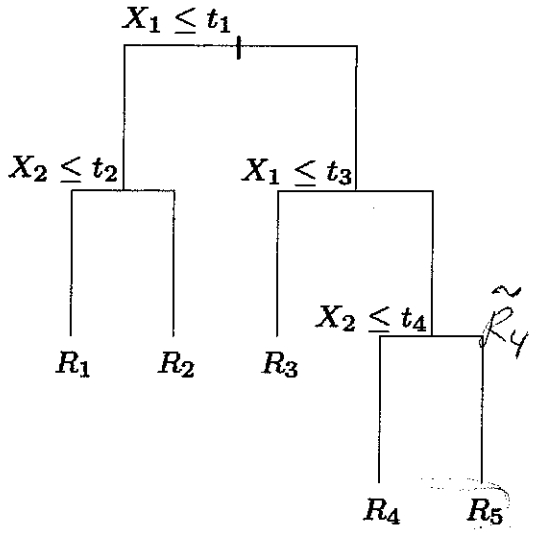
Here $c_1 = \text{argmax}_{k=1,\ldots K}\ \hat{p}_k(R_1)$ is the most common class in R_1, and $c_2 = \text{argmax}_{k=1,\ldots K}\ \hat{p}_k(R_2)$ is the most common class in R_2

Having done this, we now repeat this within each of the newly defined regions R_1, R_2. That is, it again considers splitting all variables j and split points s, within each of R_1, R_2, this time greedily choosing the pair that provides us with the biggest improvement in misclassification error



How do we find the best split s? Aren't there infinitely many to consider? No, to split a region R_m on a variable j, we really only need to consider n_m splits (or $n_m - 1$ splits)





Continuing on in this manner, we will get a big tree T_0. Its leaves define regions $R_1, \ldots R_m$. We then prune this tree, meaning that we collapse some of its leaves into the parent nodes

For any tree T, let $|T|$ denote its number of leaves. We define

$$C_\alpha(T) = \sum_{j=1}^{|T|} \left[1 - \hat{p}_{c_j}(R_j)\right] + \alpha|T|$$

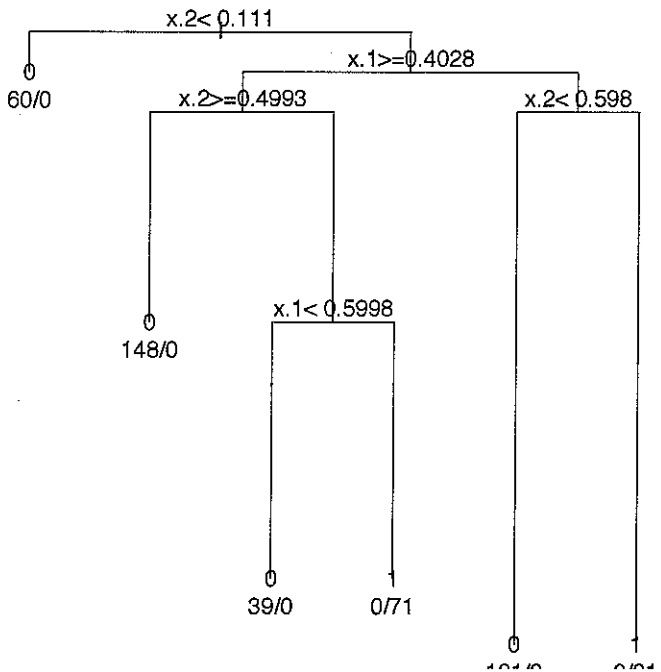
leaves

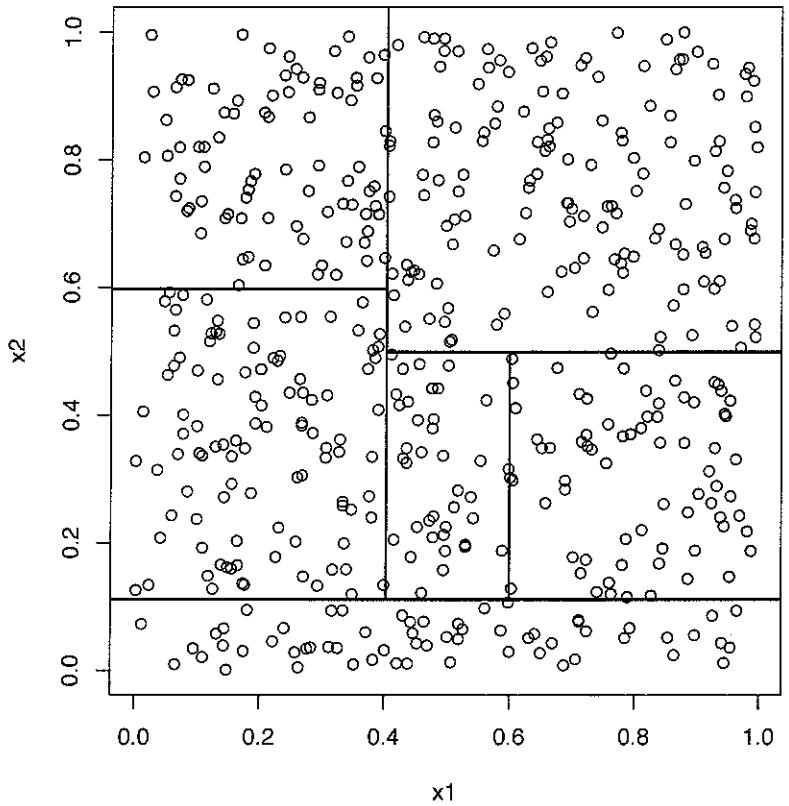
depth of tree

We seek the tree $T \subseteq T_0$ that minimizes $C_\alpha(T)$. It turns out that this can be done by pruning the weakest leaf one at a time. Note that α is a tuning parameter, and a larger α yields a smaller tree. CART picks α by 5- or 10-fold cross-validation

Example: simple classification tree

Example: $n = 500$, $p = 2$, and $K = 2$. We ran CART:





To use CART in R, you can use either of the functions `rpart` or `tree`, in the packages of those same names. When you call `rpart`, cross-validation is performed automatically; when you call `tree`, you must then call `cv.tree` for cross-validation

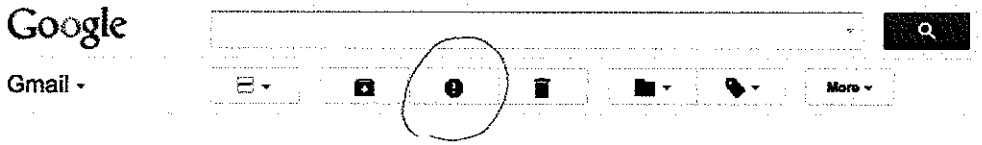
Example: spam data

Example: $n = 4601$ emails, of which 1813 are considered spam. For each email we have $p = 58$ attributes. The first 54 measure the frequencies of 54 key words or characters (e.g., "free", "need", "$"). The last 3 measure

- the average length of uninterrupted sequences of capitals;
- the length of the longest uninterrupted sequence of capitals;
- the sum of lengths of uninterrupted sequences of capitals

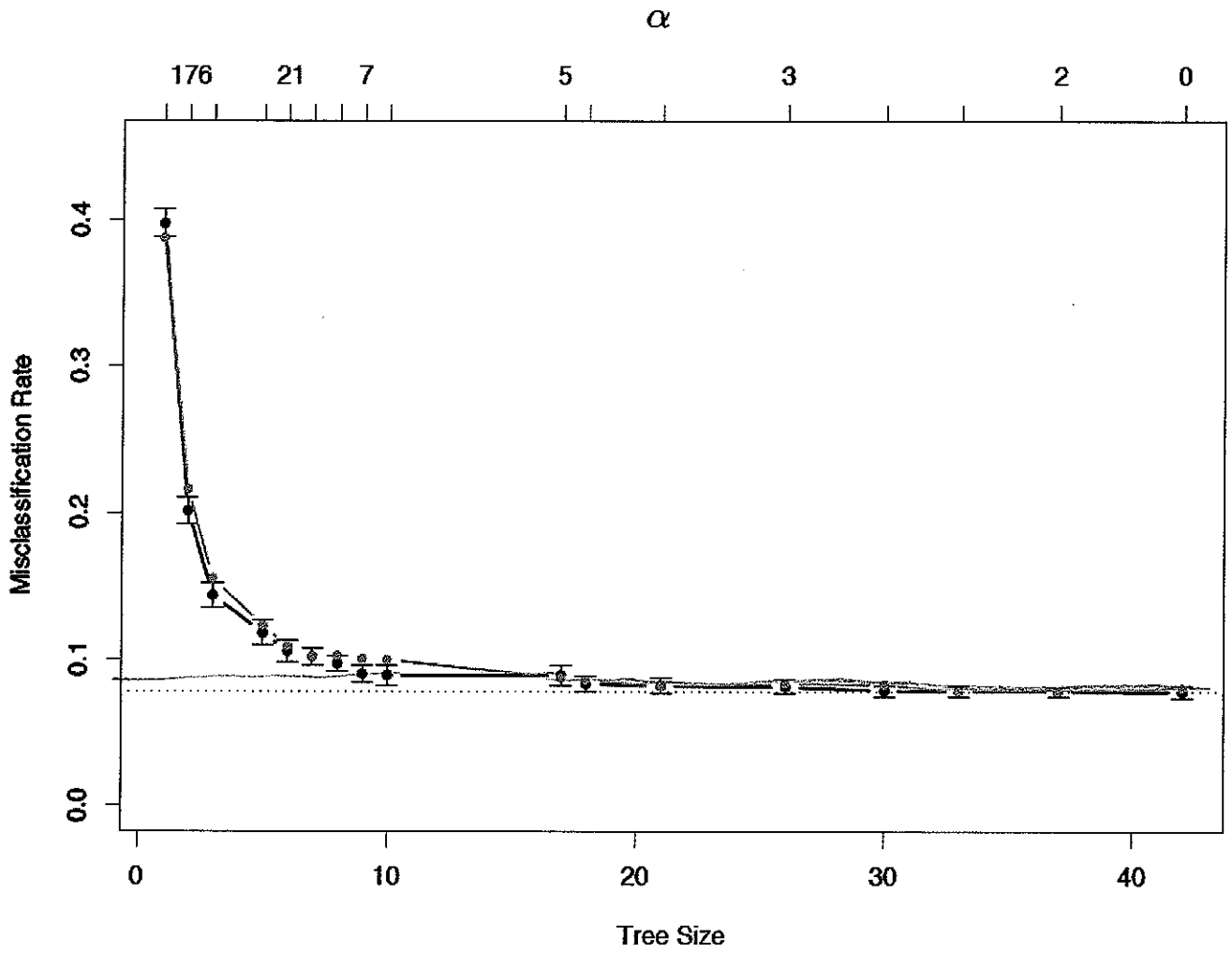
(Data from ESL section 9.2.5)

An aside: how would we possibly get thousands of emails labeled as spam or not?

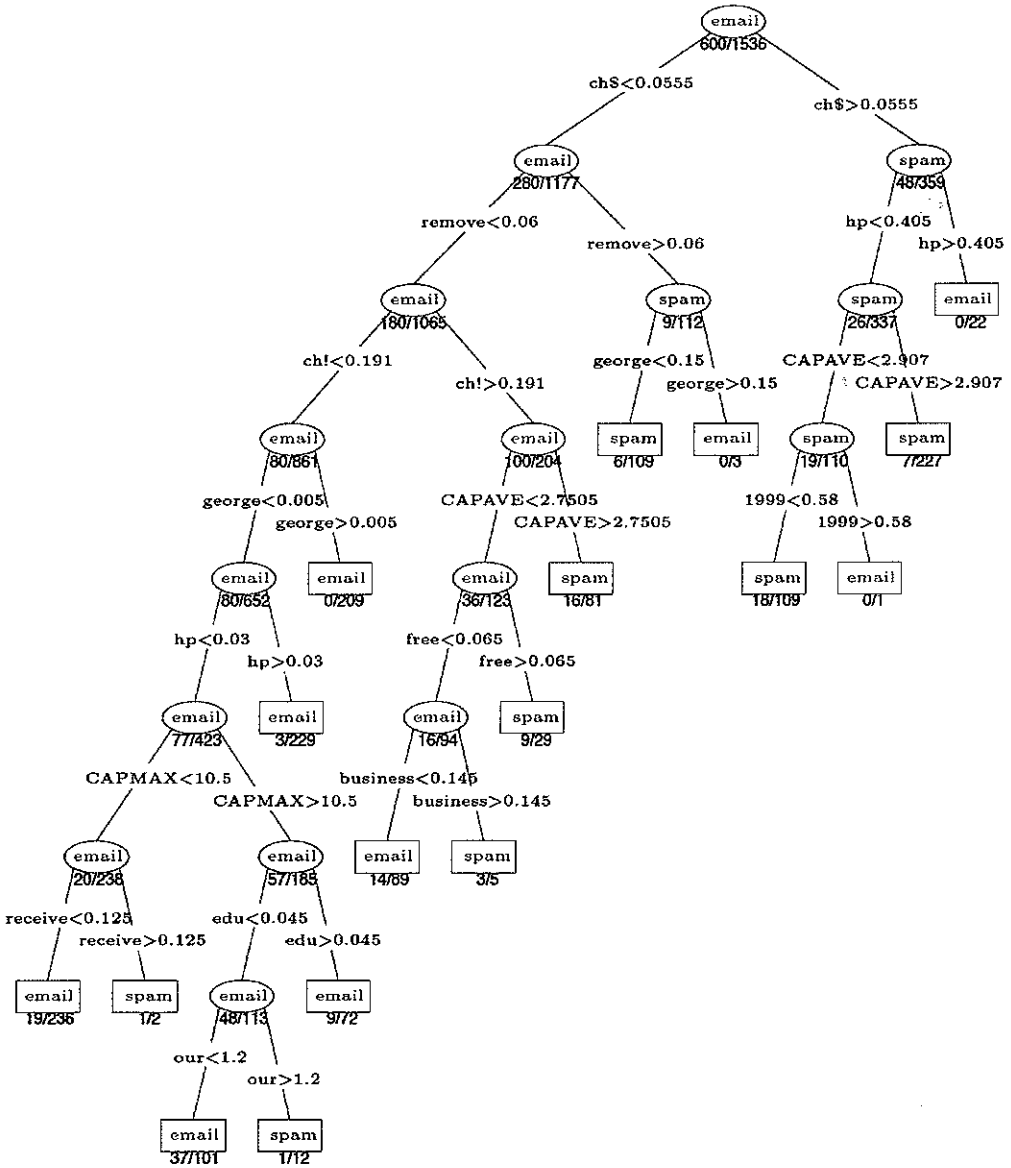


This is great! Every time you label an email as spam, gmail has more training data!

Cross-validation error curve for the spam data (from ESL page 314):



Tree of size 17, chosen by cross-validation (from ESL page 315):



Other impurity measures

We used misclassification error as a measure of the impurity of region R_j,

$$1 - \hat{p}_{c_j}(R_j)$$

But there are other useful measures too: the Gini index:

$$\sum_{k=1}^{K} \hat{p}_k(R_j)\big[1 - \hat{p}_k(R_j)\big],$$

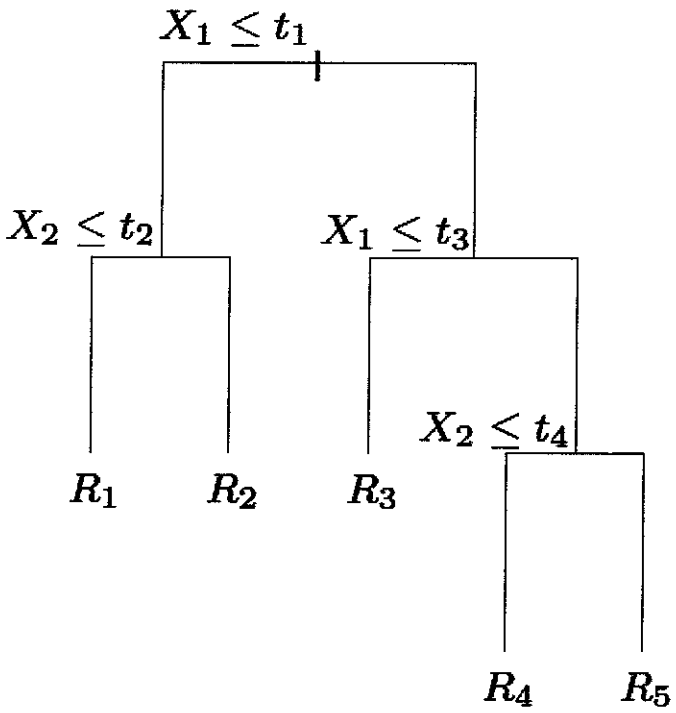
and the cross-entropy or deviance:

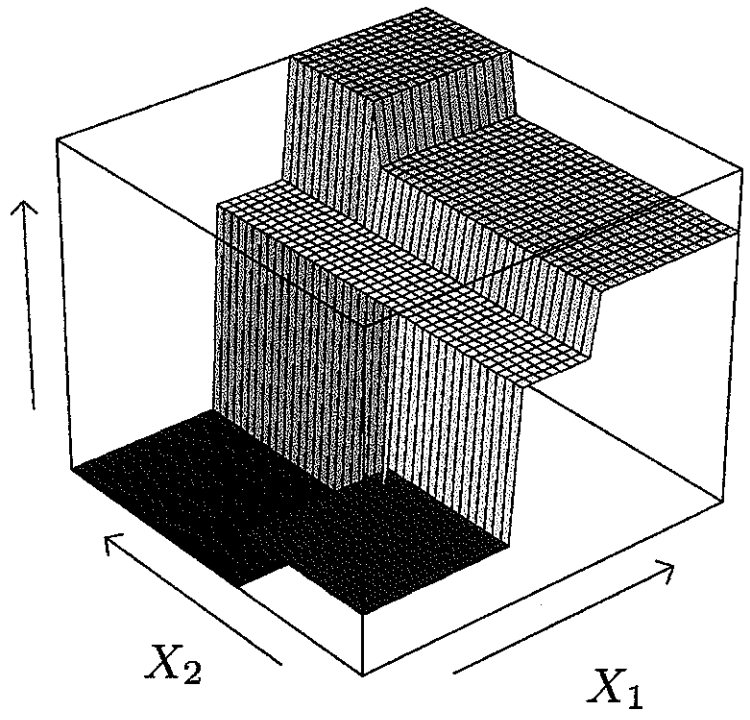
$$-\sum_{k=1}^{K} \hat{p}_k(R_j) \log\big\{\hat{p}_k(R_j)\big\}.$$

Using these measures instead of misclassification error is sometimes preferable because they are more sensitive to changes in class probabilities. Overall, they are all pretty similar (Homework 6)

Regression trees

Suppose that now we want to predict a continuous outcome instead of a class label. Essentially, everything follows as before, but now we just fit a constant inside each rectangle





The estimated regression function has the form

$$\hat{f}^{\text{tree}}(x) = \sum_{j=1}^{m} c_j \cdot 1\{x \in R_j\} = c_j \text{ such that } x \in R_j$$

just as it did with classification. The quantities c_j are no longer predicted classes, but instead they are real numbers. How would we choose these? Simple: just take the average response of all of the points in the region,

$$c_j = \frac{1}{n_j} \sum_{x_i \in R_j} y_i$$

The main difference in building the tree is that we use squared error loss instead of misclassification error (or Gini index or deviance) to decide which region to split. Also, with squared error loss, choosing c_j as above is optimal

How well do trees predict?

Trees seem to have a lot of things going in the favor. So how is their predictive ability?

Unfortunately, the answer is not great. Of course, at a high level, the prediction error is governed by bias and variance, which in turn have some relationship with the size of the tree (number of nodes). A larger size means smaller bias and higher variance, and a smaller tree means larger bias and smaller variance

But trees generally suffer from high variance because they are quite instable: a small change in the observed data can lead to a dramatically different sequence of splits, and hence a different prediction. This instability comes from their hierarchical nature; once a split is made, it is permanent and can never be "unmade" further down in the tree

We'll learn some variations of trees have much better predictive abilities. However, their predictions rules aren't as transparent

Recap: trees for classification and regression

In this lecture, we learned about trees for classification and regression. Using trees, we divide the feature space up into rectangles by making successive splits on different variables, and then within each rectangle (leaf of the tree), the predictive task is greatly simplified. I.e., in classification, we just predict the most commonly occuring class, and in regression, we just take the average response value of points in the region

The space of possible trees is huge, but we can fit a good tree using a greedy strategy, as is done by the CART algorithm. It also grows a large tree, and then prunes back at the end, choosing how much to prune by cross-validation

Trees are model-free and are easy to interpret, but generally speaking, aren't very powerful in terms of predictive ability. Next time we'll learn some procedures that use trees to make excellent prediction engines (but in a way we lose interpretability)

Next time: bagging

Fitting small trees on bootstrapped data sets, and averaging predictions at the end, can greatly reduce the prediction error (from ESL page 285):