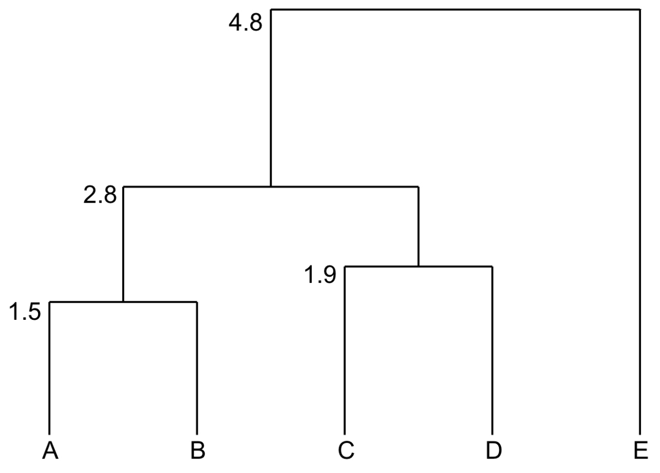
Statistical Machine Learning — 10 — Classification via Trees — October 1, 2015

Lecturer: Prof. Homrighausen — Scribe: Ryan Hicks

To start class, Veronica gave a presentation on Matrix Inverse Identities.

This document will pick up from slide 1 of Classification via Trees and continue though the end of the section.

§Trees



The tree or *dendrogram* pictured above has 5 terminal nodes (*leaves*) and 4 interior nodes. Each split or end point is called a *node*. Interior nodes lead to *branches*. In statistics, trees are formed by partitioning the predictor space into M regions, $R_1, R_2, \ldots, R_M$. Every observation that falls into a given region R_m is given the same prediction. We can use trees for regression, in which case the prediction is the average of the responses for a region, or classification, in which case the prediction is the plurality vote in that region. Because trees are quite simple, they are easy to interpret; however, that comes at a price. Trees are much less useful for prediction. This is mainly because trees can only make rectangular regions parallel to the coordinate axis, often missing much structure of the data. Furthermore, the fitting process does not reconsider splits in the tree once they have been made (a *greedy* algorithm). Regardless of whether the analyst is using trees for regression or classification, they need to choose some stopping point; the tree needs to stop splitting when there are too few observations in a leaf, exactly how many is too few is up to the analyst.

§Regression Trees

Here is a more detailed method for regression using trees:

For a given partition $R_1, \ldots, R_M$, the model for the response is:

$$f(X) = \sum_{m=1}^{M} c_m \mathbf{1}_{R_m}(X)$$

For this model, we need to estimate both R_m and c_m. However, searching over all possible regions not computationally feasible, so we use a greedy approach, as mentioned above.

First, we define the two half-planes:

$$r_1(j,s) = \{X | X^j \leq s\} \qquad \text{and} \qquad r_2(j,s) = \{X | X^j > s\}$$

Using the squared-error loss function, we solve:

$$\min_{j,s} \quad [\min_{c_1} \sum_{X_i \in r_1(j,s)} (Y_i - c_1)^2 + \min_{c_2} \sum_{X_i \in r_2(j,s)} (Y_i - c_2)^2]$$

This generates, for $n_k = \sum_{i=1}^{n} \mathbf{1}_{r_k}(X_i)$,

$$\hat{c}_k = n_k^{-1} \sum_{i: X_i \in r_k} Y_i$$

which is the sample mean. This process continues, with the next splits conditional on the minimizing $\hat{s}$.

§Classification Trees

Here is a detailed method for classification using trees:

For a given partition R_m and class g, define training proportions:

$$\hat{p}_{mg}(X) = \mathbf{1}_{R_m}(X) n_m^{-1} \sum_{i: X_i \in R_m} \mathbf{1}(Y_i = g)$$

This is really just mimicking the empirical conditional probability. Ideally, we would want $\mathbb{P}(Y = g|X)$, but there are most likely no observations at that point (a lack of replication), so we will have to settle for $\mathbb{P}(Y = g|X \in R_m)$.

Then, the classification is:

$$\hat{g}(X) = \arg\max_{g} \hat{p}_{mg}(X)$$

This classification presupposes that a partition R_m exists, so we ned to estimate it. For this we need to decide on an appropriate loss function.

§Node Impurity

In the literature, there are many ways to measure *node impurity*, but three common measures are:

Classification Error Rate:	$E = 1 - \max_g(\hat{p}_{mg})$
Gini Index:	$G = \sum_g \hat{p}_{mg}(1 - \hat{p}_{mg})$
Cross-Entropy:	$D = -\sum_g \hat{p}_{mg} \log(\hat{p}_{mg})$

For some theoretical reasons, using classification error rate tends to produce solutions sub-par to those of the other two. But whichever one is used, the tree is built by greedily minimizing the chosen criterion. There are many good graphs on slides 14 - 17 of Classification via Trees that go through an example of the differences of E, G, and D. However, all measures will tend to overfit, leading to trees with too many leaves. Although the analyst could increase the minimum number of observations required to split a node, this might easily correct too far and lead to underfitting. We need to burn off some errant leaves. Cross-validation is an obvious choice, but it turns out that it is computationally infeasible because we do not have a predictable path. We need a less complex approach, and that approach is *weakest-link pruning*:

$$\sum_{m=1}^{|T|} \sum_{i \in R_m} \mathbf{1}(Y_i \neq \hat{Y}_{R_m}) + \lambda |T|$$

where $|T|$ is the number of terminal nodes.

Now, we have introduced another tuning parameter into the mix, λ, but this can be chosen with cross-validation because the complexity is sufficiently reduced and now there is a predictable path. Pruned trees are simpler than unpruned trees and thus have higher interpretability and they also help with model selection. Note that a pruned tree will always be a subset (*nested*) of the unpruned tree.

§Trees in R

Here a short example:

```
require(rpart)
require(tree)
out.tree = tree(Y‘., data = X, split= ’gini’)
plot(out.tree)
text(out.tree)
```

Now prune the tree:

```
out.tree.orig = tree(Y ., data = X)
out.tree.cv = cv.tree(out.tree.orig, FUN = prune.misclass)
> out.tree.cv
$size
[1] 14 13 11 9 3 2 1

$dev
[1] 45 45 44 44 44 64 67

$k
[1] -Inf 0.0 2.0 2.5 3.0 15.0 20.0

$method
[1] ”misclass”
```

Here, k corresponds to the λ using weakest-link pruning and dev means the number of misclassifications, not deviance. Now choose the best tree:

```
best.size = out.tree.cv$size[which.min(out.tree.cv$dev)]
> best.size
[1] 11
out.tree = prune.misclass(out.tree.orig, best = best.size)
class.tree = predict(out.tree, X_0, type = 'class')
```