Tree-based methods for classification and regression

Rebecca C. Steorts
Predictive Modeling

November 2015

Optional reading: ISL 4.1, 4.2, 4.4, ESL 4.1–4.3; ISL 8.1, ESL 9.2

Classification

Classification is a predictive task in which the response takes values across discrete categories (i.e., not continuous), and in the most fundamental case, two categories

Examples:

- Predicting whether a patient will develop breast cancer or remain healthy, given genetic information
- Predicting whether or not a user will like a new product, based on user covariates and a history of his/her previous ratings
- Predicting the region of Italy in which a brand of olive oil was made, based on its chemical composition
- Predicting the next elected president, based on various social, political, and historical measurements

Similar to our usual setup, we observe pairs (x_i, y_i), $i = 1, \ldots n$, where y_i gives the class of the ith observation, and $x_i \in \mathbb{R}^p$ are the measurements of p predictor variables

Though the class labels may actually be $y_i \in \{\text{healthy}, \text{sick}\}$ or $y_i \in \{\text{Sardinia}, \text{Sicily}, ...\}$, but we can always encode them as

$$y_i \in \{1, 2, \ldots K\}$$

where K is the total number of classes

- Note that there is a big difference between classification and clustering; in the latter,
- there is not a pre-defined notion of class membership (and sometimes, not even K),
- and we are not given labeled examples (x_i, y_i), $i = 1, \ldots n$, but only x_i, $i = 1, \ldots n$

- Assume training data (x_i, y_i), $i = 1, \ldots n$,
- Denote classification rule by $\hat{f}(x)$; given any $x \in \mathbb{R}^p$,
- This returns a class label $\hat{f}(x) \in \{1, \ldots K\}$

As before, we will see that there are two different ways of assessing the quality of $\hat{f}$: its predictive ability and interpretative ability

E.g., train on (x_i, y_i), $i = 1, \ldots n$, the data of elected presidents and related feature measurements $x_i \in \mathbb{R}^p$ for the past n elections, and predict, given the current feature measurements $x_0 \in \mathbb{R}^p$, the winner of the current election

In what situations would we care more about prediction error? And in what situations more about interpretation?

Binary classification and linear regression

Let's start off by supposing that $K = 2$, so that the response is $y_i \in \{1, 2\}$, for $i = 1, \ldots n$

You already know a tool that you could potentially use in this case for classification: linear regression. Simply treat the response as if it were continuous, and find the linear regression coefficients of the response vector $y \in \mathbb{R}^n$ onto the predictors, i.e.,

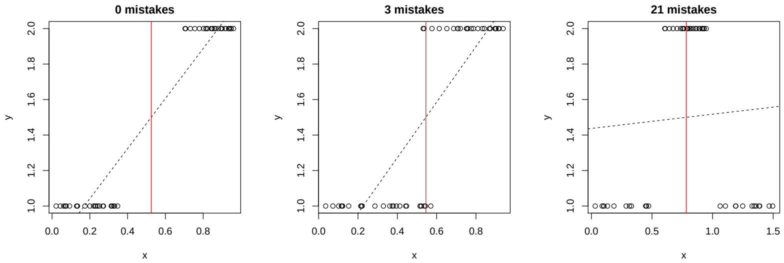
$$\hat{\beta}_0, \hat{\beta} = \underset{\beta_0 \in \mathbb{R},\, \beta \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - \beta_0 - x_i^T \beta)^2$$

Then, given a new input $x_0 \in \mathbb{R}^p$, we predict the class to be

$$\hat{f}^{\text{LS}}(x_0) = \begin{cases} 1 & \text{if } \hat{\beta}_0 + x_0^T \hat{\beta} \leq 1.5 \\ 2 & \text{if } \hat{\beta}_0 + x_0^T \hat{\beta} > 1.5 \end{cases}$$

(Note: since we included an intercept term in the regression, it doesn't matter whether we code the class labels as $\{1, 2\}$ or $\{0, 1\}$, etc.)

In many instances, this actually works reasonably well. Examples:



Overall, using linear regression in this way for binary classification is not a crazy idea. But how about if there are more than 2 classes?

Linear regression of indicators

This idea extends to the case of more than two classes. Given K classes, define the indicator matrix $Y \in \mathbb{R}^{n \times K}$ to be the matrix whose columns indicate class membership; that is, its jth column satisfies $Y_{ij} = 1$ if $y_i = j$ (observation i is in class j) and $Y_{ij} = 0$ otherwise

E.g., with $n = 6$ observations and $K = 3$ classes, the matrix

$$Y = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \in \mathbb{R}^{6 \times 3}$$

corresponds to having the first two observations in class 1, the next two in class 2, and the final 2 in class 3

To construct a prediction rule, we regress each column $Y_j \in \mathbb{R}^n$ (indicating the jth class versus all else) onto the predictors:

$$\hat{\beta}_{j,0}, \hat{\beta}_j = \underset{\beta_{j,0} \in \mathbb{R},\, \beta_j \in \mathbb{R}^p}{\operatorname{argmin}} \sum_{i=1}^n (Y_{ij} - \beta_{0,j} - \beta_j^T x_i)^2$$

Now, given a new input $x_0 \in \mathbb{R}^p$, we compute

$$\hat{\beta}_{0,j} + x_0^T \hat{\beta}_j, \quad j = 1, \ldots K$$

take predict the class j that corresponds to the highest score. I.e., we let each of the K linear models make its own prediction, and then we take the strongest. Formally,

$$\hat{f}^{\mathrm{LS}}(x_0) = \underset{j=1,\ldots K}{\operatorname{argmax}}\ \hat{\beta}_{0,j} + x_0^T \hat{\beta}_j$$

The decision boundary between any two classes j, k are the values of $x \in \mathbb{R}^p$ for which

$$\hat{\beta}_{0,j} + x^T \hat{\beta}_j = \hat{\beta}_{0,k} + x^T \hat{\beta}_k$$

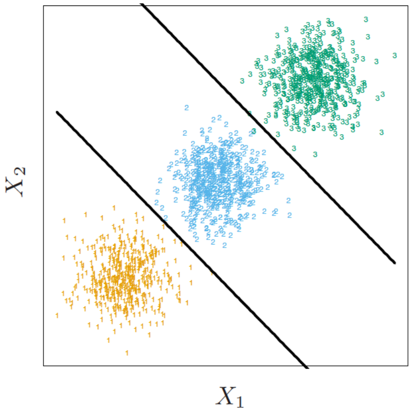
i.e., $\hat{\beta}_{0,j} - \hat{\beta}_{0,k} + (\hat{\beta}_j - \hat{\beta}_k)^T x = 0$

This defines a $(p-1)$-dimensional affine subspace in $\mathbb{R}^p$. To one side, we would always predict class j over k; to the other, we would always predict class k over j

For K classes total, there are $\binom{K}{2} = \frac{K(K-1)}{2}$ decision boundaries

Ideal result

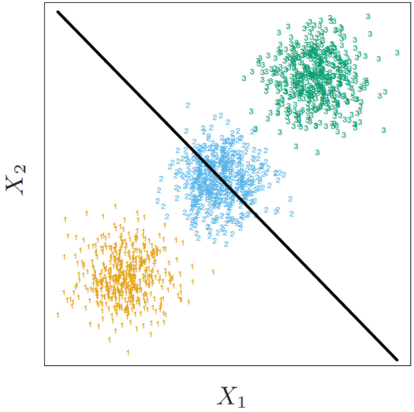
What we'd like to see when we use linear regression for a 3-way classification (from ESL page 105):



The plotted lines are the decision boundaries between classes 1 and 2, and 2 and 3 (the decision boundary between classes 1 and 3 never matters)

Actual result

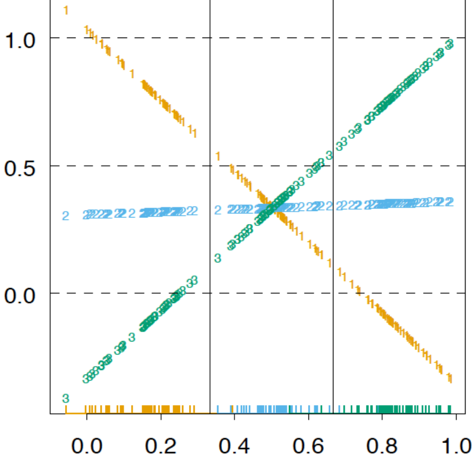
What actually happens when we use linear regression for this 3-way classification (from ESL page 105):



The decision boundaries between 1 and 2 and between 2 and 3 are the same, so we would never predict class 2. This problem is called masking (and it is not uncommon for moderate K and small p)

Why did this happen?

Projecting onto the line joining the three class centroids gives some insight into why this happened (from ESL page 106):



Methods I'm not covering

- LDA and QDA
- logistic versions
- See ISL to learn more

Tree-based methods

Tree-based based methods for predicting y from a feature vector $x \in \mathbb{R}^p$ 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

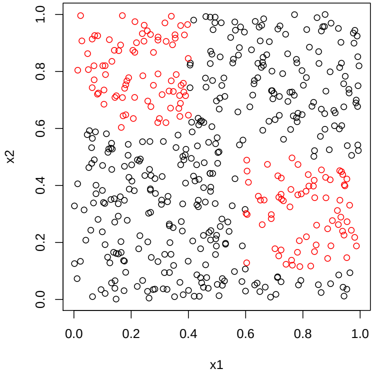
$$X_j \leq c \quad \text{and} \quad X_j > c$$

Then we proceed on each half

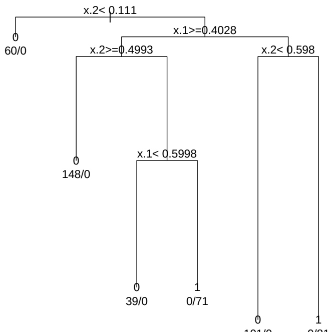
For simplicity, consider classification first (regression later). If a half is "pure", meaning that it mostly contains points from one class, then we don't need to continue splitting; otherwise, we continue splitting

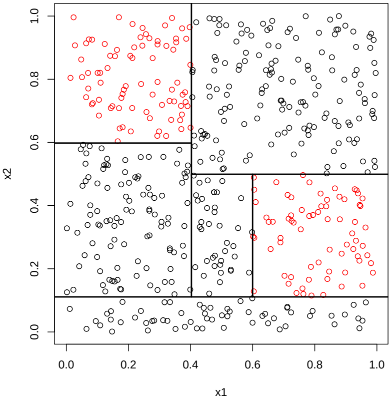
Example: simple classification tree

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


x2
x1
0.0
0.2
0.4
0.6
0.8
1.0

Classification trees

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

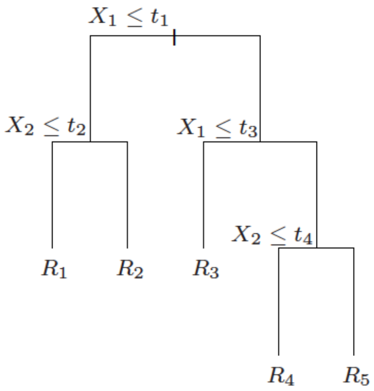
Let (x_i, y_i), $i = 1, \ldots n$ be the training data, where $y_i \in \{1, \ldots K\}$ are the class labels, and $x_i \in \mathbb{R}^p$ measure the p predictor variables. The classification tree can be thought of as defining m regions (rectangles) $R_1, \ldots R_m$, each corresponding to a leaf of the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

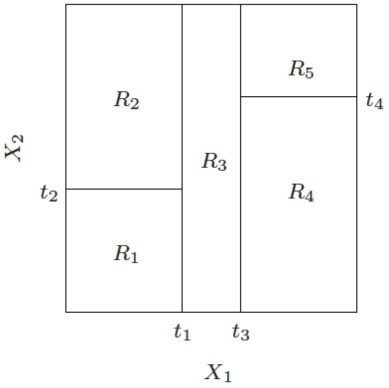
We assign each R_j a class label $c_j \in \{1, \ldots K\}$. We then classify a new point $x \in \mathbb{R}^p$ by

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

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)

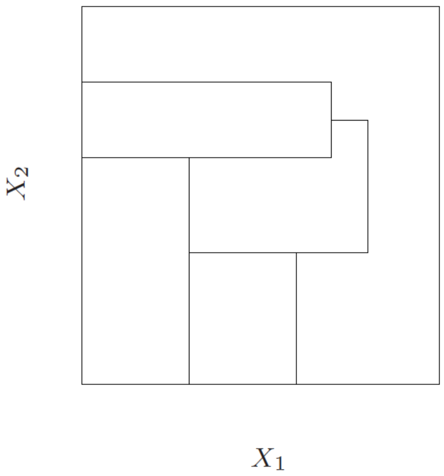
Example: regions defined by a tree





(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	Yes	Yes	No
LR	Yes	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

[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

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

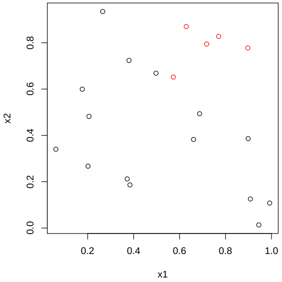
We then greedily chooses j, s by minimizing the misclassification error

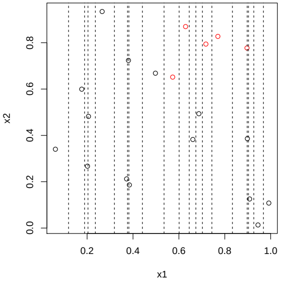
$$\underset{j,s}{\operatorname{argmin}} \; \Big(\big[1 - \hat{p}_{c_1}(R_1)\big] + \big[1 - \hat{p}_{c_2}(R_2)\big] \Big)$$

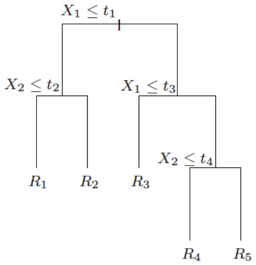
Here $c_1 = \operatorname{argmax}_{k=1,\ldots K} \hat{p}_k(R_1)$ is the most common class in R_1, and $c_2 = \operatorname{argmax}_{k=1,\ldots K} \hat{p}_k(R_2)$ is the most common class in R_2

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

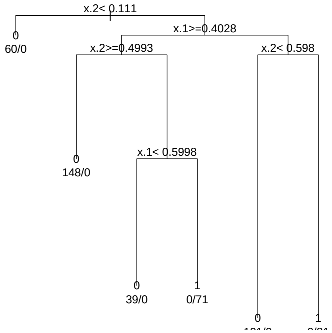
For any tree T, let $|T|$ denote its number of leaves. We define

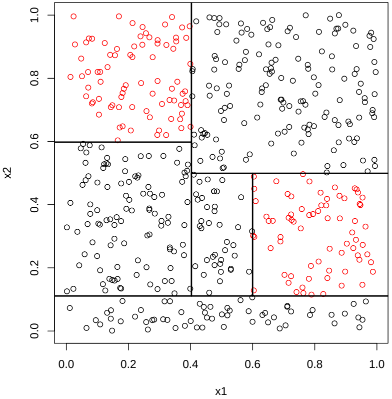
$$C_\alpha(T) = \sum_{j=1}^{|T|} \left[1 - \hat{p}_{c_j}(R_j)\right] + \alpha|T|$$

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

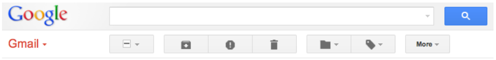
Example: spam data

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

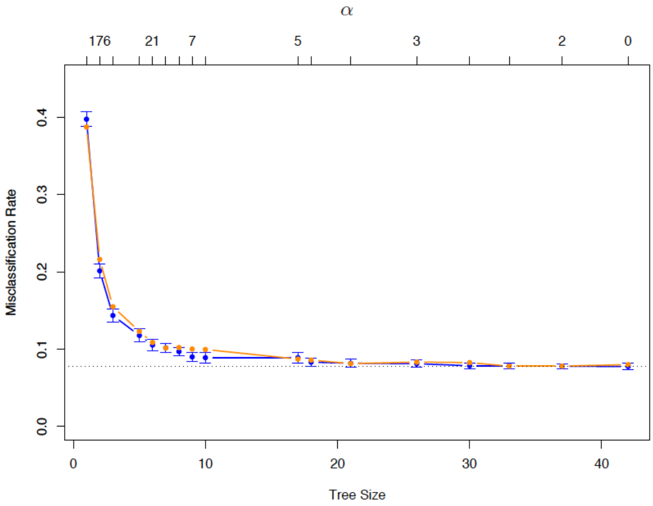
(Data from ESL section 9.2.5)

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

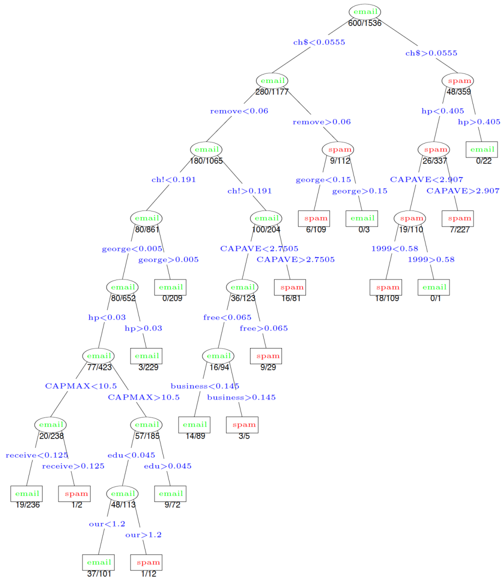


This is great! Every time you label an email as spam, gmail has more training datafg

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:

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

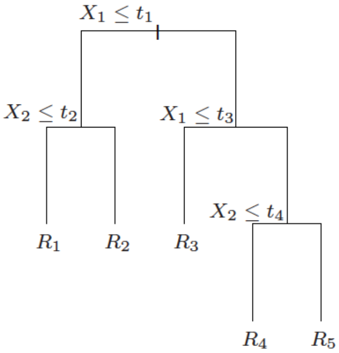
and the cross-entropy or deviance:

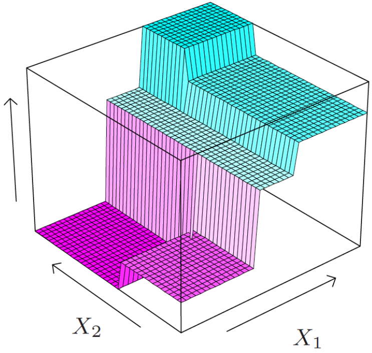
$$-\sum_{k=1}^{K} \hat{p}_k(R_j) \log\big\{\hat{p}_k(R_j)\big\}.$$

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 7)

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 smaller 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

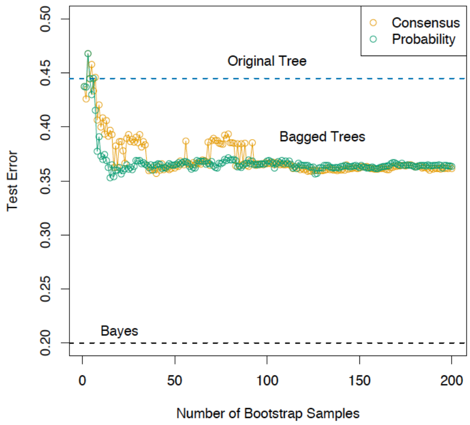
In this lecture, we learned about trees for classification and regression. Using trees, we divide the feature space up into rectangles by making successive splits on different variables, and then within each rectangle (leaf of the tree), the predictive task is greatly simplified. I.e., in classification, 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):


Consensus
Probability
Original Tree
Bagged Trees
Bayes
Test Error
Number of Bootstrap Samples