Linear and Non-linear Methods for Supervised Learning

Ordinary linear regression, logistic regression, and Naive Bayes all end up looking only at a linear combination of the inputs, x. But often y is related to x in a non-linear fashion. How can we handle that?

One simple (and traditional) approach: Just add extra input variables, like x_1^2, $\sin(x_2)$, $x_1 x_2$, etc. Then use a linear method with these extra input variables.

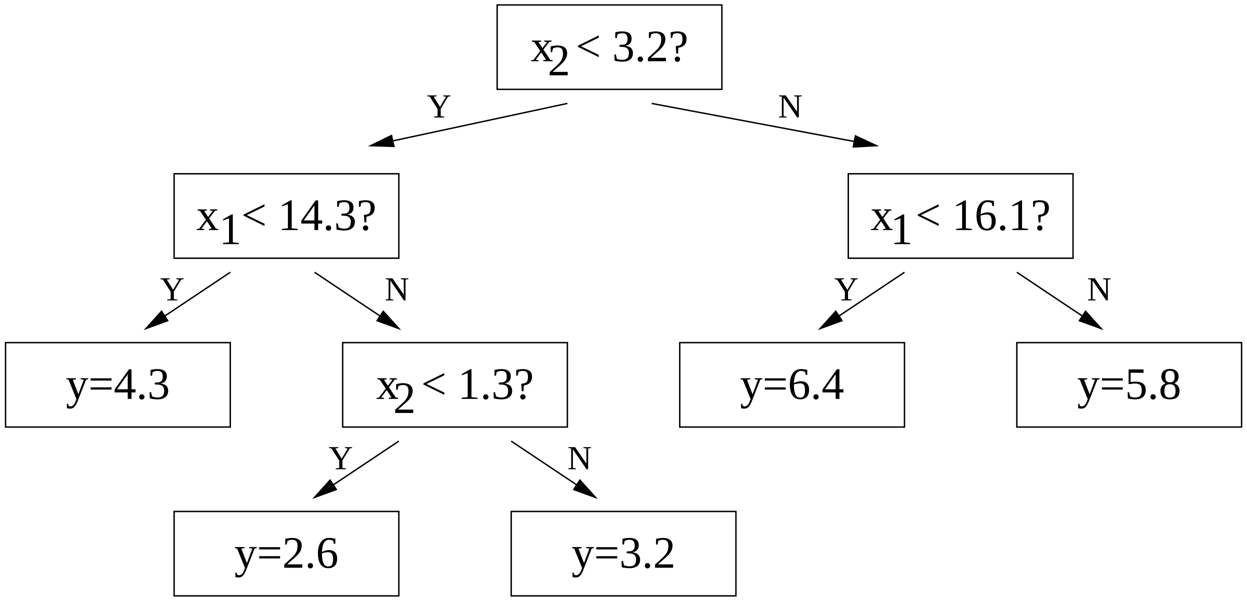
This works well if we know which functions of inputs are likely to be useful. But how do we choose functions of the inputs if we don't know much? Then we'd like to find these functions using the training data.

Many schemes along these lines have been proposed. We'll start with one in which the only functions of the inputs used are indicators of x being in some rectangular region...

Decision trees

Suppose the inputs, $x_1, \ldots, x_p$, are real valued. (Though this method can be adapted for discrete inputs too.) We could try to model y by successively "splitting" cases using criteria that depend on single input variables.

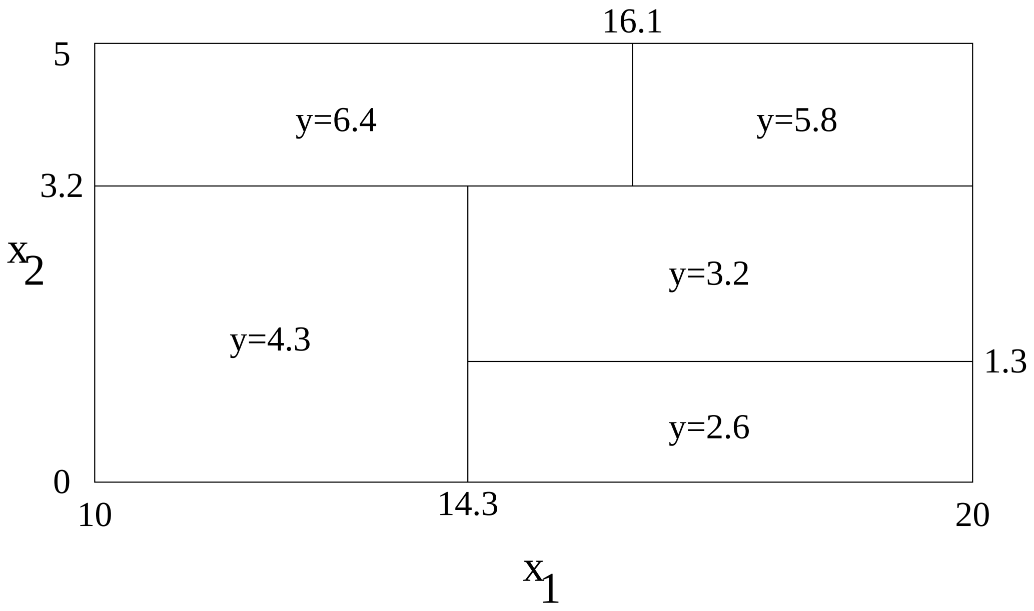
The resulting prediction process can be viewed as a tree of decisions. For example:



To make a prediction for a test case, we go down the tree to where it belongs, then predict the y value given at that leaf.

Rectangles Defined by Trees

One can also see the tree as defining rectangular regions of the input space. The tree from the preceding slide defines regions as follows:



How to Build Such Trees

Using the training data, we need to figure out the tree structure, the exact split points at internal nodes, and the y values at leaves.

It makes sense to use the average value of y in all training cases that get put in a leaf as the prediction for that leaf.

We start with a trivial tree where the root node is a leaf, whose y value is the mean of all the y values. We then split nodes until all nodes have less than some number of cases (or have the same y value for all cases).

At each iteration, we choose a node, k, an input variable, j, and a split value, v, for that variable. Various criteria have been used to choose among the possible splits.

One criterion is to choose the split that maximizes the likelihood (after finding y values for the new leaves). For regression with Gaussian residuals, this is the same as minimizing the squared error.

Note that this is a *greedy* procedure — it looks just one step ahead at each stage. A better tree with a given number of nodes might exist than is found this way.

Choosing Splits in More Detail

Here's is the split criterion in more detail.

We consider all possible nodes, k, and variables, j. For such a split, we need to consider only values v, that are mid-way between values for y in training cases that end up at node k.

We compute the decrease in squared error if we split node k on variable j at v:

$$\sum_{i \in \text{node } k} (y^{(i)} - \bar{y}_k)^2 \quad - \sum_{i \in \text{node } k} \left\{ \begin{array}{ll} (y^{(i)} - \bar{y}_{k_1})^2 & \text{if } x_j^{(i)} < v \\ (y^{(i)} - \bar{y}_{k_2})^2 & \text{if } x_j^{(i)} \geq v \end{array} \right\}$$

where $\bar{y}_k$ is the average value of y for training cases that are put in node k, and $\bar{y}_{k_1}$ and $\bar{y}_{k_2}$ are the averages for training cases in node k with $x_j < v$ and $x_j \geq v$, respectively.

Problems with Decision Trees

- As nodes are split, the amount of data used for subsequent splits gets smaller and smaller (exponentially quickly). So later splits may be very subject to chance.

- The tree has to discover over and over (on each branch) the effects of variables that really are additive in nature. In effect, the model assumes that *everything* interacts, when often some things don't.

- The tree is likely to become rather complex, and to *overfit* the data. This can be addressed by *pruning* the tree — merging some leaf nodes until a simpler tree is produced. Choosing how simple can be done using the log likelihood plus a penalty for the size of the tree.

- How big should the penalty be? This question, and similar questions for other models can be addressed using *cross validation*.

Pruning the Tree with a Penalty

After building a big tree, we can prune it back to minimize a penalized likelihood function. For regression problems with Gaussian residuals, this amounts to minimizing squared error plus a penalty term:

$$\sum_{k\in\mathcal{T}} \;\sum_{i\,\in\,\text{node }k} (y^{(i)} - \bar{y}_k)^2 \quad + \quad \lambda|\mathcal{T}|$$

where $\mathcal{T}$ is the set of nodes in the tree (with this pruning), $\bar{y}_k$ is the mean of target values for training cases in node k, and $|\mathcal{T}|$ is the number of nodes in $\mathcal{T}$.

This might seem hard, requiring a search of all possible prunings of the original tree. Fortunately, if you prune one leaf node at a time (the one that helps reduce squared error least) until you have only the root left, it's guaranteed that you'll come across the tree that minimizes the penalized squared error.

As λ increases, the pruned tree we choose will get smaller. We need to somehow choose λ.