12 Tree-based methods for classification and regression

Suppose that we are interested to predict a response or class Y from inputs $X_1, X_2, \ldots, X_p$ (see Tibshirani, 2013, and Steorts, 2014). We do this by growing a binary tree. At each internal node in the tree, we apply a test to one of the inputs, e.g. X_i. Depending on the outcome of the test, we go to either the **left** or the **right** sub-branch of the tree. Eventually we come to a leaf node, where we make a prediction. This prediction averages all the training data points which reach that leaf.

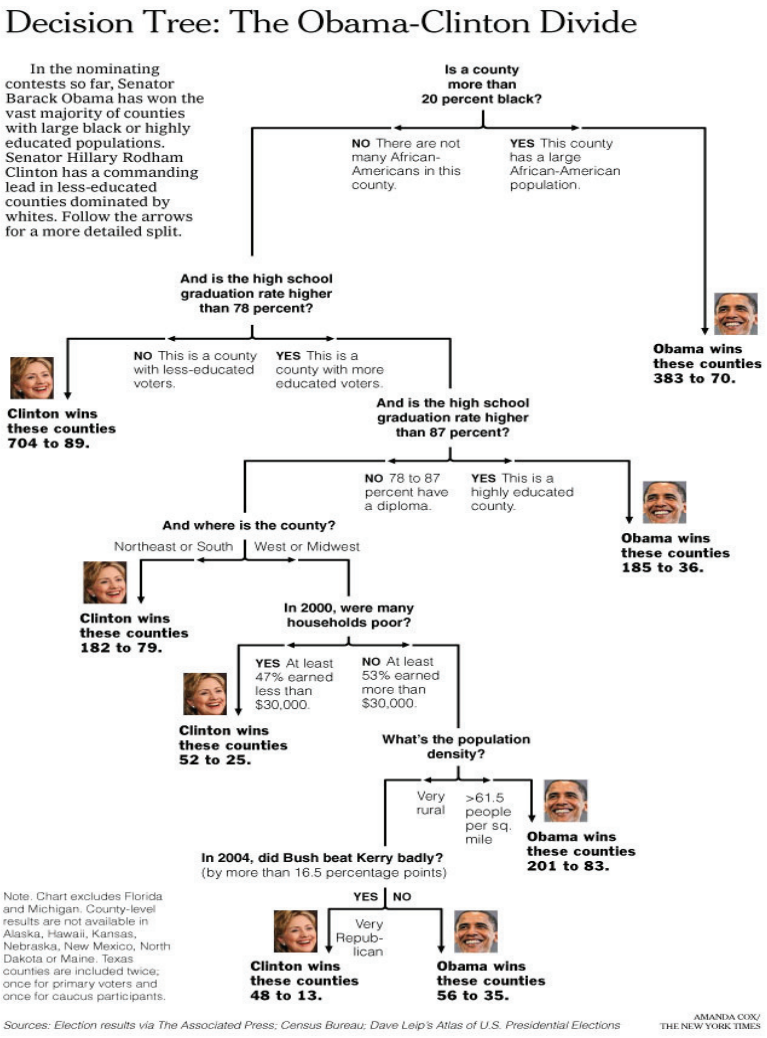

Figure 12.1: Classification tree for county-level outcomes in the 2008 Democratic Party primary (as of April 16, 2008), by Amanada Cox for *the New York Times*.

Why do this? Linear or polynomial regression predictors are global models, where a single predictive formula is supposed to hold over the entire data space. However, when the data exhibit

lots of features which interact in sophisticated nonlinear ways, assembling a single global model can be challenging and even infeasible. (Note that some of the non-parametric smoothers, e.g. GAM, try to fit models locally and then paste them together, but again they can be hard to interpret.)

An alternative approach to nonlinear regression is to **linearize** interactions: i.e. to sub-divide, or partition, the space into smaller regions, where the interactions are more manageable (almost linear). We then partition the sub-divisions again, until finally we get to chunks of the space which are so tame that we can fit simple models to them. The global model thus has two parts: one is just the recursive partition, the other is a simple model for each cell of the partition.

Prediction trees use the tree to represent the **recursive partition**. Each of the terminal nodes, or **leaves**, of the tree represents a cell of the partition, and has attached to it a simple model which applies in that cell only. A point x belongs to a leaf if x falls in the corresponding cell of the partition. To figure out which cell we are in, we start at the root node of the tree, and ask a sequence of questions about the features. The interior nodes are labeled with questions, and the edges or branches between them labeled by the answers. Which question we ask next depends on the answers to previous questions. In the classic version, each question refers to only a single attribute, and has a *yes* or *no* answer, e.g., "**What is the population density?**" or "**Where is the county?**" The variables can be of any combination of types (continuous, discrete but ordered, categorical, etc.). We can incorporate more than-binary questions, but that can always be accommodated as a larger binary tree. Asking questions about multiple variables at once is, again, equivalent to asking multiple questions about single variables. This constitutes the **recursive partition** part.

Now let us turn to the **simple local models**. For classic regression trees, the model in each cell is just a constant estimate of Y. I.e., suppose the points $(x_1, y_1), (x_2, y_2), \ldots (x_c, y_c)$ are all the samples belonging to the leaf-node l. Then our model for l is just

$$\hat{y} = \frac{1}{c} \sum_{i=1}^{c} y_i,$$

i.e. the sample mean of the response variable in that cell. This is a **piecewise-constant** model. There are several advantages to this:

- Making predictions is fast

- It is easy to understand what variables are important in making the prediction
- If some data is missing, we might not be able to go all the way down the tree to a leaf, but we can still make a prediction by averaging all the leaves in the sub-tree we do reach
- The model provides a jagged response, so it can work when the true regression surface is not smooth. If it is smooth, the piecewise-constant surface can approximate it arbitrarily closely (with enough leaves, i.e. approximation order)
- There are fast, reliable algorithms to learn these trees

Let us compare trees with the k-nearest-neighbors methods which is a popular nonparametric procedure for cluster detection, inference, smoothing etc. The idea of k-nearest-neighbors is to find the points which are most similar to the node of interest, and do what, on average, the neighbors do. There exist two main drawbacks to this approach:

- you define "**similarity**" entirely in terms of the inputs, not the response;
- k is constant everywhere; however, some points just might have much more similar neighbors than others.

Trees get around both problems: leaves correspond to regions of the input space (a neighborhood), but one where the responses are similar, as well as the inputs being nearby; and their size can vary arbitrarily. Prediction trees are **adaptive nearest-neighbor** methods.

12.1 Classification Trees

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 $x_i \in R^p$ represents 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 $x \in R^p$ by

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

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 as we would need to look at each region R_j.

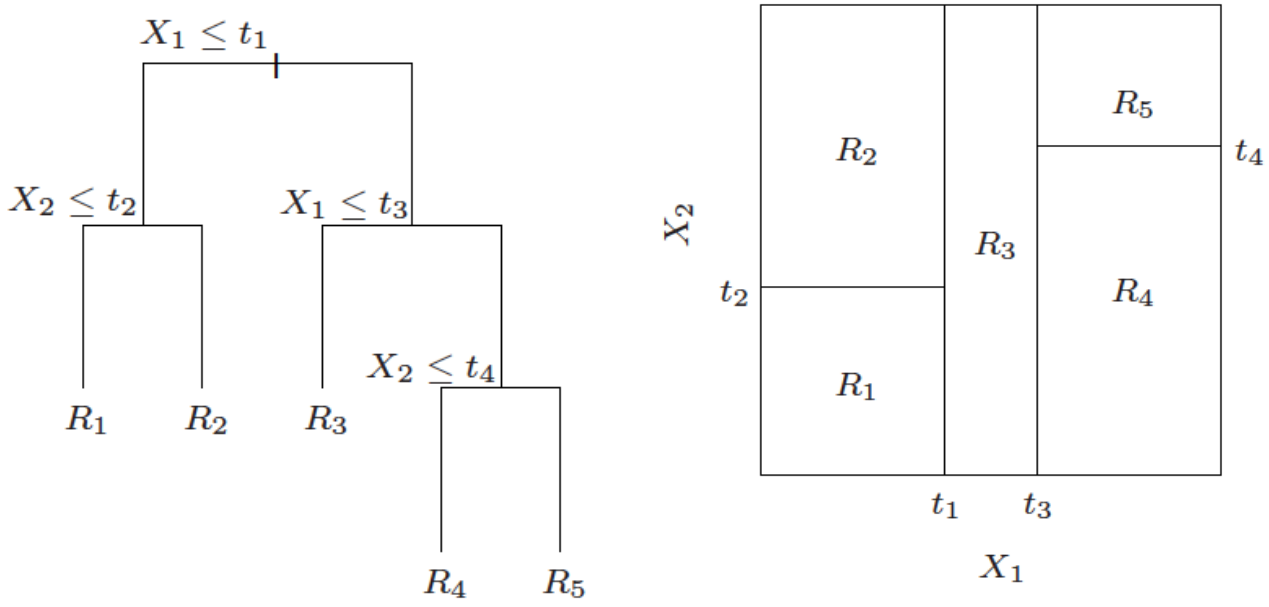

Figure 12.2: Regions defined by a tree (Hastie, Tibshirani, and Friedman, 2009).

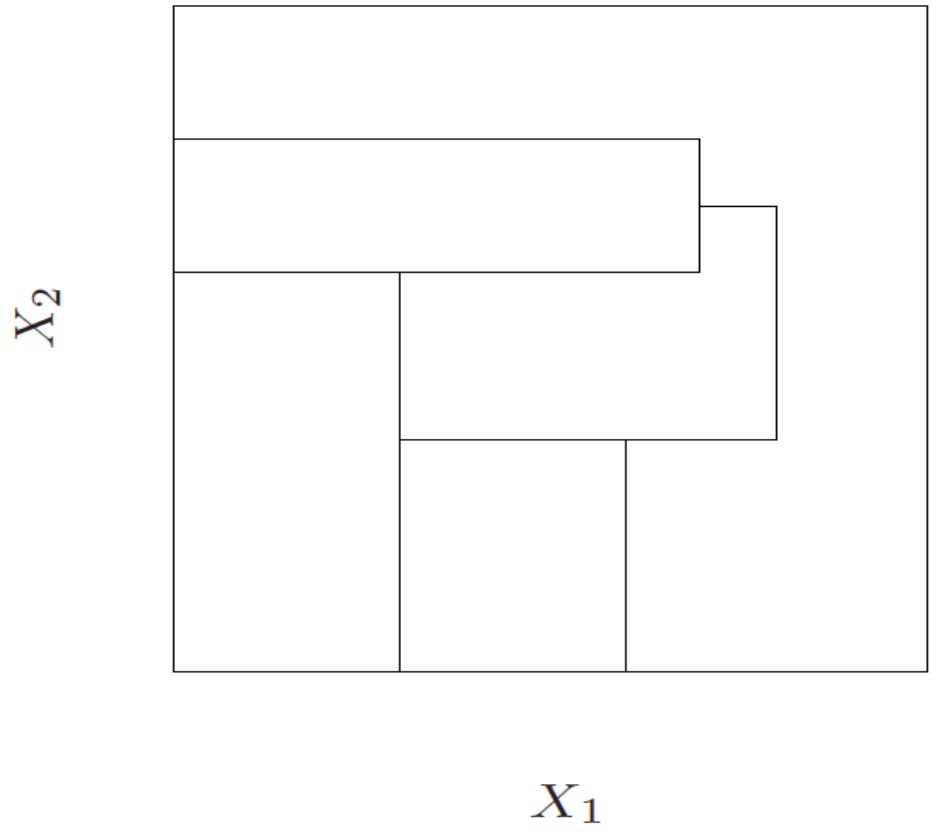

Figure 12.3: Regions not defined by a tree (Hastie, Tibshirani, and Friedman, 2009).

Table 12.1: Comparison of different methods

Method	Model assumptions?	Estimated probabilities?	Interpretable?	Flexible?
LDA	Yes	Yes	Yes	No
k-NN	No	No	No	Yes
Trees	No	Yes	Yes	Somewhat

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), $i = 1, \dots n$, e.g. n_j points. The predicted class c_j is just the most common occuring class among these points.

Moreover, for each class $k = 1, \dots K$, we can estimate the probability that the class label is k, given that the feature vector lies in region R_j, i.e. $P(C = k | X \in R_j)$, 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

$$c_j = \underset{k=1,\dots K}{arg\,\max} \hat{p}_k(R_j)$$

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

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

First, think about varying the depth of the tree: i.e.,

- 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 approximately 2^d nodes. (A balanced data is when data are appproximately evenly spread among classes. However, a frequent situation encountered in classification problems is

that of unbalanced data. A training dataset consisting of a disprortionately high number of examples from one class will result in a classifier that is biased towards this majority class. The most commonly used method is to **rebalance** the data by resampling.) At each node we could choose any of p the variables for the split – this means that the number of possibilities is

$$p \cdot 2^d,$$

which is huge even for moderate d.

12.2 The Classificaion and Regression Tree (CART) algorithm of Breiman et al., 1984

The CART algorithm of Breiman et al. (1984) selects the splits in a top down fashion: 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? This is 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.

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:

$$\begin{aligned} R_1 &= \big\{X \in R^p : X_j \leq s\big\} \\ R_2 &= \big\{X \in R^p : X_j > s\big\}. \end{aligned}$$

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

$$\underset{j,s}{arg\,min}\big([1 - \hat{p}_{c_1}(R_1)] + [1 - \hat{p}_{c_2}(R_2)]\big),$$

where $c_1 = \underset{k=1,\ldots K}{arg\,max}\hat{p}_k(R_1)$ is the most common class in R_1 and $c_2 = \underset{k=1,\ldots K}{arg\,max}\hat{p}_k(R_2)$ is the most common class in R_2.

We now repeat this within each of the newly defined regions R_1 and R_2. I.e., we again consider splitting all variables j and split points s within each of R_1 and R_2.

Note that to find the best split s, we do not need to consider an infinite dimensional set of possible choices s. In contrast, to split a region R_m on a variable j we can only consider $n_m - 1$ splits where n_m is the sample size of R_m.

By continuing on in this manner, we get a big tree T_0. Its leaves define regions $R_1, \dots R_m$. 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 the loss function

$$C_\alpha(T) = \sum_{j=1}^{|T|} \big(1 - \hat{p}_{c_j}(R_j)\big) + \alpha|T|.$$

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 V-fold crossvalidation.

Example. We model customer attrition rates in one of the oldest Credit Unions (CU) of Canada with $3.0 billion in assets under administration and over 40,000 clients. We use information on client's age, Average Days Between Transactions, Days Since Last Transaction, Number of Loans, Number of Past Transactions, Tenure, Total Borrowed and Total Savings.

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

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

However, there are other useful measures too, e.g. 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**:

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

Using these measures instead of misclassification error is sometimes preferable because they are more sensitive to changes in class probabilities.

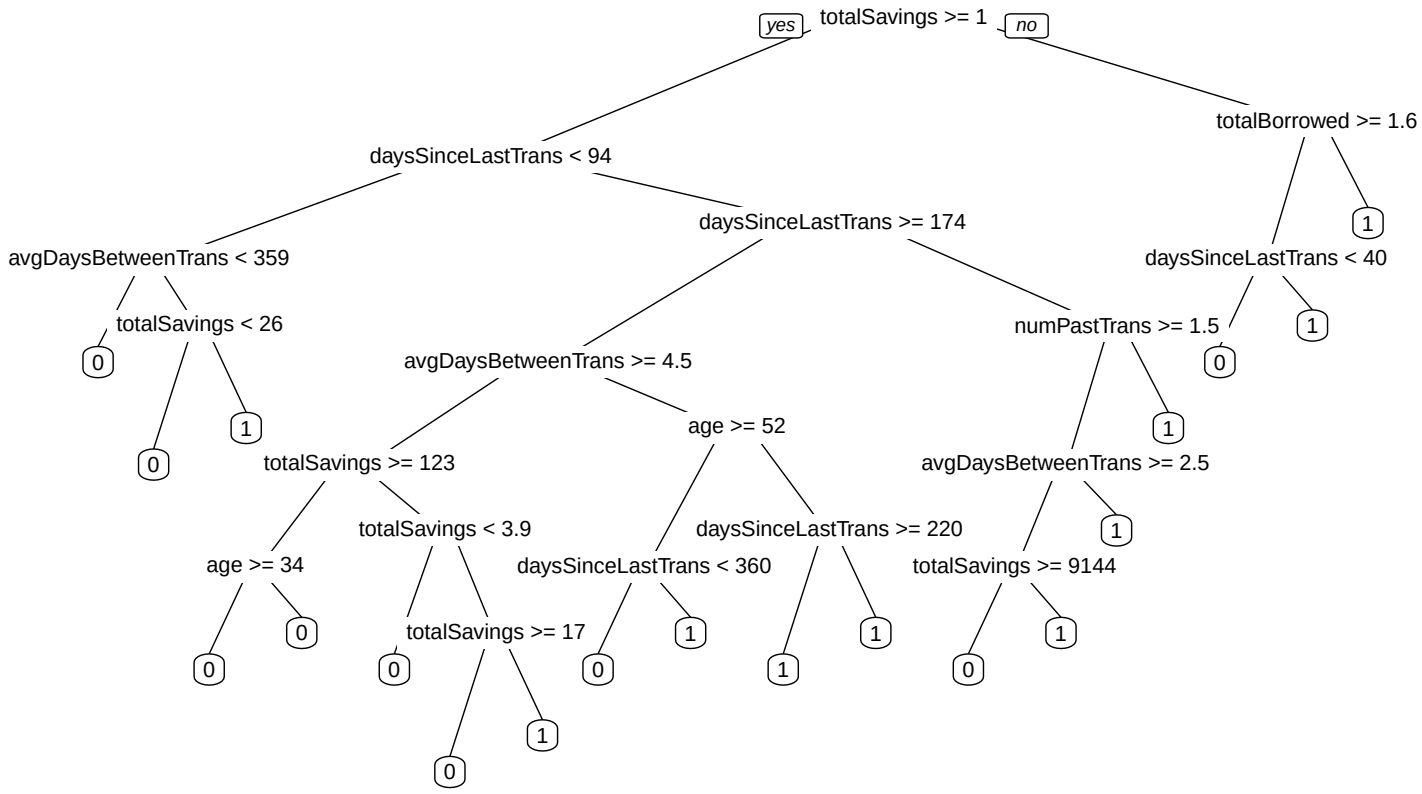

Figure 12.4: A Sample CART on Individual Variables

12.3 Regression trees

Suppose that now we would like to predict a continuous outcome instead of a class label. Essentially, everything stays the same except that now we just fit a constant inside each rectangle.

The estimated regression function takes the form

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

such that $x \in R_j$, i.e. just as in the classification problem. The quantities c_j are no longer predicted classes, but instead they are real numbers. How would we choose c_j? The idea is simple, i.e. 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 regression tree is that we use the **squared error loss** instead of **misclassification error** (or **the Gini index** or **deviance**) as in the classification tree to decide which region to split. Also, with the **squared error loss**, choosing c_j as above is optimal.

Unfortunately, the regression trees generally suffer from high variance because they are quite instable, i.e/ 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. I.e., once a split is made, it is permanent and can never be "unmade" further down in the tree. As a remedy, we use special modifications based the bootstrapping idea that have much better predictive performance than classical regression trees.