Tree-based Methods

Tree-based methods partition the feature space into a set of rectangles and then fit a simple model (like a constant) in each one.

simple regions

This results in a simple model, useful for interpretation

These simple tree models do not provide much predictive accuracy.

Combining a large number of trees can often result in dramatic improvements in prediction accuracy at the expense of interpretation.

bagging, random forests, boosting, etc.

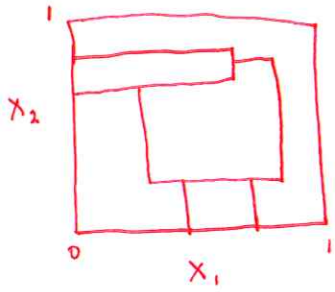
quantitative y response

categorical y response.

Decision trees can be applied to both regression and classification problems. We will start with regression.

1 Decision Trees

Let's consider a regression problem with continuous response Y and inputs X_1 and X_2, each taking values in the unit interval.



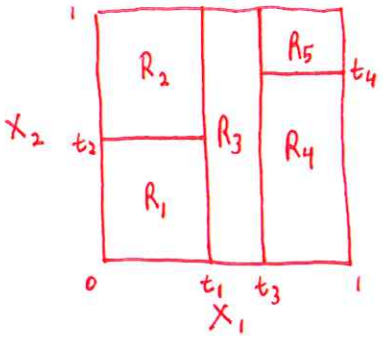
can partition feature space w/ lines parallel to coordinate axes.

model Y in each partition element as a constant

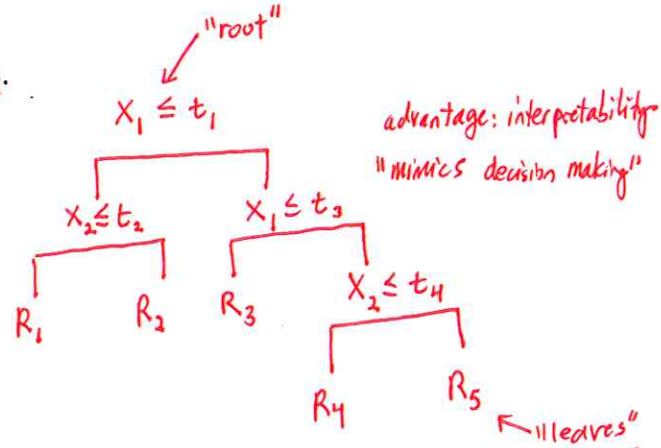
In each partition, we can model Y with a different constant. However, there is a problem:

Although each partition line has a simple description like $X_1 = C$, resulting regions are hard to describe.

To simplify, we restrict attention to binary partitions.



$\Longleftrightarrow$



advantage: interpretability
"mimics decision making"

The result is a partition into five regions $R_1, \ldots, R_5$. The corresponding regression model predicts Y with a constant c_m in region R_m:

$$\hat{f}(\underline{X}) = \sum_{m=1}^{5} c_m \mathbb{I}((X_1, X_2) \in R_m)$$

1.1 Regression Trees

How should we grow a regression tree? Our data consists of p inputs for $i = 1, \ldots, n$. We need an automatic way to decide which variables to split on and where to split them.

Suppose we have a partition into M regions and we model the response as a constant in each region. Want a final model that "closely" fits our actual data.

If we use sum of squares to evaluate "closeness" and thus minimize as a criteria to choose our model,

$$\sum_{i=1}^{N} (y_i - \hat{f}(x_i))^2$$

the best $\hat{c}_m$ is the mean $\hat{c}_m = \frac{\sum y_i}{i : x_i \in R_m}$

Finding the best binary partition in terms of minimum sums of squares is generally computationally infeasible.

So we use a top-down, greedy approach called recursive binary splitting.

(1) Select the predictor and cutpoint s s.t. splitting the predictor space into $\{\underline{X} \mid x_j \leq s\}$ and $\{\underline{X} \mid x_j > s\}$ leads to greatest reduction in RSS.

Consider all possible half-planes $R_1(j, s) = \{\underline{X}; x_j \leq s\}$ and $R_2(j, s) = \{\underline{X}; x_j > s\}$.

We seek j, s to minimize

$$\sum_{i : x_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{i : x_i \in R_2(j,s)} (y_i - \hat{c}_2)^2$$

(2) Repeat process, looking for next best combo of j, s but instead of splitting the whole space, split $R_1(j, s)$ and $R_2(j, s)$ to minimize RSS.

(3) Continue until stopping criteria is met (i.e. no region contains more than 5 observations).

The process described above may produce good predictions on the training set, but is likely to overfit the data.

tree may be too complex, fitting noise rather than signal.

A smaller tree, with less splits might lead to lower variance and better interpretation at the cost of a little bias.

(Bad) idea: only make a split if "large enough" drop in RSS.

↓

seemingly worthless split early might be followed by a good split.

A strategy is to grow a very large tree T_0 and then *prune* it back to obtain a *subtree*.

better idea!

"cost complexity pruning"

Consider a sequence of trees indexed by a nonnegative tuning parameter α.

For each α, $\exists$ a corresponding subtree $T \subset T_0$ s.t.

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{C}_m)^2 + \alpha |T| \text{ is minimized.}$$

$|T|$: # of terminal nodes in T.

α controls the trade-off between complexity & closeness of fit

When $\alpha = 0$, $T = T_0$

$\alpha \uparrow \Rightarrow$ price to pay for having many terminal nodes $\uparrow \Rightarrow$ smaller tree.

Choose α via CV.

1.2 Classification Trees

If the target is a classification outcome taking values $1, 2, \ldots, K$, the only changes needed in the tree algorithm are the criteria for splitting, pruning, and c_m.

c_m:

let $\hat{p}_{mk} = \frac{1}{n_m} \sum_{i: x_i \in R_m} \mathbb{I}(y_i = k)$ = prop. of class k in region m.

Then we classify obs. in Region m to class $k(m) = \underset{k}{\operatorname{argmax}}\, \hat{p}_{mk}$. (majority class).

Node impurity (Splitting):

what to minimize to choose j, s:

Misclassification error: $\frac{1}{n_m} \sum_{i: x_i \in R_m} \mathbb{I}(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}$.

result in pure terminal nodes

Gini Index: $\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$.

Deviance: $-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$.

$\Rightarrow$ use for splitting.

Pruning:

Can use any of above 3.

If prediction is the goal, typically use misclassification error.

2 Bagging

Decision trees suffer from *high variance*.

results depends greatly on the specific sample of data we have.

vs. low variance: will yield similar results w/ different data sets from same population

e.g. linear regression w/ $n >> p$.

Bootstrap aggregation or *bagging* is a general-purpose procedure for reducing the variance of a statistical learning method, particularly useful for trees.

For independent $Z_1, \ldots, Z_n$ each w/ variance σ^2

$$\text{Var}(\bar{Z}) = \frac{\sigma^2}{n}$$

i.e. averaging indep. obs. reduces variance.

So a natural way to reduce the variance is to take many training sets from the population, build a separate prediction model using each training set, and average the resulting predictions.

i.e. take B training data sets

calculate $\hat{f}^{(1)}(x), \ldots, \hat{f}^{(B)}(x)$

obtain low variance model:

$$\hat{f}_{avg}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{(b)}(x).$$

Of course, this is not practical because we generally do not have access to multiple training sets. Collecting training data is expensive.

$\Rightarrow$ use the bootstrap!

Fit our model on b^{th} bootstrapped data set to get $\hat{f}^{*(b)}(x)$ and average:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*(b)}(x).$$

"bagging" = bootstrap aggregating