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.

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

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.

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

To simplify, we restrict attention to binary partitions.

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:

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.

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

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

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

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

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:

Node impurity (Splitting):

Pruning:

2 Bagging

Decision trees suffer from *high variance*.

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

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.

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

While bagging can improve predictions for many regression methods, it's particularly useful for decision trees.

These trees are grown deep and not pruned.

How can bagging be extended to a classification problem?

2.1 Out-of-Bag Error

There is a very straightforward way to estimate the test error of a bagged model.

2.2 Interpretation

3 Random Forests

Random forests provide an improvement over bagged trees by a small tweak that decorrelates the trees.

As with bagged trees, we build a number of decision trees on bootstrapped training samples.

In other words, in building a random forest, at each split in the tree, the algorithm is not allowed to consider a majority of the predictors.

The main difference between bagging and random forests is the choice of predictor subset size m.

4 Boosting

The basic idea of *boosting* is to take a simple (and poorly performing form of) predictor and by sequentially modifying/perturbing it and re-weighting (or modifying) the training data set, to creep toward an effective predictor.

Consider a 2-class 0-1 loss classification problem. We'll suppose that output y takes values in $\mathcal{G} = \{-1, 1\}$. The AdaBoost.M1 algorithm is built on some base classifier form.

1. Initialize the weights on the training data.

2. Fit a $\mathcal{G}$-valued predictor/classifier $\hat{f}_1$ to the training data to optimize the 0-1 loss.

3. Set new weights on the training data.

4. For $m = 2, \ldots, M$,

5. Output an updated classifier based on "weighted voting".

4.1 Why might this work?

For g an arbitrary function of $\boldsymbol{x}$, consider a classifier built using g as a voting function, e.g. $f(\boldsymbol{x}) = \text{sign}(g(\boldsymbol{x}))$, ignoring the possibility that $g(\boldsymbol{x}) = 0$. Then

$$\mathbb{I}(y \neq \hat{y}) = \mathbb{I}(yg(\boldsymbol{x}) < 0).$$

Using the following fact,

$$\mathbb{I}(u < 0) \leq \exp(-u) \; \forall u,$$

provided $P(g(\boldsymbol{X}) = 0) = 0$, the 0-1 loss error rate for $f(\boldsymbol{x})$ is

$$\text{E}[\mathbb{I}(Y \neq \hat{Y})] = \text{E}[\mathbb{I}(Yg(\boldsymbol{X}) < 0)] \leq \text{E}[\exp(-Yg(\boldsymbol{X})].$$

In other words, the error rate is bounded above by expected exponential loss. AdaBoost works by **providing a voting function that produces a small value of this bound.**

To see this, we need to identify for each $\boldsymbol{u}$ a value a that optimizes $\text{E}\left[\exp(-aY)|\boldsymbol{X} = \boldsymbol{u}\right]$, where

$$\text{E}\left[\exp(-aY)|\boldsymbol{X} = \boldsymbol{u}\right] = \exp(-a)P\left[Y = 1|\boldsymbol{X} = \boldsymbol{u}\right] + \exp(a)P\left[Y = -1|\boldsymbol{X} = \boldsymbol{u}\right].$$

An optimal a is easily seen to be half the log odds ratio, i.e. the g optimizing the upper bound is

$$g\left(\boldsymbol{u}\right) = \frac{1}{2}\ln\left(\frac{P\left[yY = 1|\boldsymbol{X} = \boldsymbol{u}\right]}{P\left[Y = -1|\boldsymbol{X} = \boldsymbol{u}\right]}\right).$$

Now consider "base classifiers" $h_\ell\left(\boldsymbol{x}, \boldsymbol{\gamma}_\ell\right)$ taking values in $\mathcal{G} = \{-1, 1\}$ with parameters $\boldsymbol{\gamma}_l$ and functions built from them of the form

$$g_m\left(\boldsymbol{x}\right) = \sum_{l=1}^{m} \beta_\ell h_\ell\left(\boldsymbol{x}, \boldsymbol{\gamma}_\ell\right).$$

for training-data-dependent β_l and $\boldsymbol{\gamma}_l$.

Then, $g_m\left(\boldsymbol{x}\right) = g_{m-1}\left(\boldsymbol{x}\right) + \beta_m h_m\left(\boldsymbol{x}, \boldsymbol{\gamma}_m\right)$. Thus, successive g's are perturbations of the previous ones.

How can we define the perturbations to produce small values of the upper bound of our error ($\mathrm{E}[\exp(-Yg(\boldsymbol{X})]$)?

Well, we don't have a complete probability model for $(\boldsymbol{X}, Y)$ (if we did, we would be done). So, let's optmize an empirical version of this bound.

$$\begin{aligned} E_m &= \sum_{i=1}^{n} \exp(-y_i g_m(\boldsymbol{x}_i)) \qquad \text{(Now based on tr} \\ &= \sum_{i=1}^{n} \exp(-y_i g_{m-1}(\boldsymbol{x}_i) - y_i \beta_m h_m(\boldsymbol{x}_i, \boldsymbol{\gamma}_m)) \\ &= \sum_{i=1}^{n} \exp(-y_i g_{m-1}(\boldsymbol{x}_i)) \exp(-y_i \beta_m h_m(\boldsymbol{x}, \boldsymbol{\gamma}_m)), \end{aligned}$$

and let's call $v_{im} = \exp(-y_i g_{m-1}(\boldsymbol{x}_i))$.

We will consider optimal choice of $\boldsymbol{\gamma}_m$ and $\beta_m > 0$ for purposes of making g_m the best possible perturbation of g_{m-1} in terms of minimizing E_m.

1. Choice of $\boldsymbol{\gamma}_m$:

$$\begin{aligned} E_m &= \sum_{\substack{i \text{ with} \\ h_m(\boldsymbol{x}_i, \boldsymbol{\gamma}_m) = y_i}} v_{im} \exp(-\beta_m) + \sum_{\substack{i \text{ with} \\ h_m(\boldsymbol{x}_i, \boldsymbol{\gamma}_m) \neq y_i}} v_{im} \exp(\beta_m) \\ &= (\exp(\beta_m) - \exp(-\beta_m)) \sum_{i=1}^{n} v_{im} I\left[h_m(\boldsymbol{x}_i, \boldsymbol{\gamma}_m) \neq y_i\right] + \exp(-\beta_m) \sum_{i=1}^{n} v_{im} \end{aligned}$$

   Independentof β_m we need $\boldsymbol{\gamma}_m$ to minimize the v_{im}-weighted error rate of $h_m(\boldsymbol{x}, \boldsymbol{\gamma}_m)$. Call the optimized version $h_m(\boldsymbol{x})$. **This is the same as step 4a. in AdaBoost.m1.**

2. Choice of β_m:

$$\begin{aligned} E_m &= \exp(-\beta_m) \left(\sum_{\substack{i \text{ with} \\ h_m(\boldsymbol{x}_i, \boldsymbol{\gamma}_m) = y_i}} v_{im} + \sum_{\substack{i \text{ with} \\ h_m(\boldsymbol{x}_i, \boldsymbol{\gamma}_m) \neq y_i}} v_{im} \exp(2\beta_m) \right) \\ &= \exp(-\beta_m) \left(\sum_{i=1}^{n} v_{im} + \sum_{i=1}^{n} v_{im} \left(\exp(2\beta_m) - 1\right) I\left[h_m(\boldsymbol{x}_i) \neq y_i\right] \right) \end{aligned}$$

and minimization of E_m is equivalent to minimization of

$$\exp(-\beta_m)\left(1+(\exp(2\beta_m)-1)\frac{\sum_{i=1}^{N} v_{im} I\left[h_m\left(\boldsymbol{x}_i\right) \neq y_i\right]}{\sum_{i=1}^{N} v_{im}}\right).$$

Let

$$\overline{\operatorname{err}}_m^{h_m}=\frac{\sum_{i=1}^{n} v_{im} I\left[h_m\left(\boldsymbol{x}_i\right) \neq y_i\right]}{\sum_{i=1}^{n} v_{im}},$$

then a bit of calculus shows that the optimizing β_m is

$$\beta_m=\frac{1}{2}\ln\left(\frac{1-\overline{\operatorname{err}}_m^{h_m}}{\overline{\operatorname{err}}_m^{h_m}}\right).$$

Notice this coefficient is **exactly $\frac{\alpha_m}{2}$ from step 4b. and 4c. in AdaBoost.m1 (and the $\frac{1}{2}$ is irrelevant for the sign).

3. Updating weights v_{im}:

   Note that

$$\begin{aligned}
v_{i(m+1)} &= \exp(-y_i g_m\left(\boldsymbol{x}_i\right)) \\
&= \exp(-y_i\left(g_{m-1}\left(\boldsymbol{x}_i\right)+\beta_m h_m\left(\boldsymbol{x}_i\right)\right)) \\
&= v_{im}\exp(-y_i\beta_m h_m\left(\boldsymbol{x}_i\right)) \\
&= v_{im}\exp(\beta_m\left(2I\left[h_m\left(\boldsymbol{x}_i\right) \neq y_i\right]-1\right)) \\
&= v_{im}\exp(2\beta_m I\left[h_m\left(\boldsymbol{x}_i\right) \neq y_i\right])\exp(-\beta_m).
\end{aligned}$$

   Since $\exp(-\beta_m)$ is constant across i, it is irrelevant to weighting, and since the prescription for β_m produces half what AdaBoost prescribes in 4b. for α_m, the weights used in the choice of β_{m+1} and $h_{m+1}\left(\boldsymbol{x}, \boldsymbol{\gamma}_{m+1}\right)$ are exactly as in AdaBoost. Since g_1 corresponds to the first AdaBoost step, g_M is $1/2$ of the AdaBoost voting function and the g_m's generate the same classifier as the AdaBoost algorithm.

So, in conclusion, we have found g_M (a positive multiple of the AdaBoost voting function) which optimizes an empirical version of $\mathrm{E}\exp(-Yg(\boldsymbol{X}))$, the upper bound on our error rate!