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

These trees are grown deep and not pruned.

$\Rightarrow$ each tree has low bias and high variance.

averaging trees reduces variance by combining hundreds or thousands of trees.
↳ won't lead to overfitting, but can be slow.

How can bagging be extended to a classification problem? (averaging no longer an option)

(most common) majority vote: For a test obs., record class that is predicted by each tree, prediction is class predicted most often.

(usually better) probabilities: average class probabilities, then classify.

2.1 Out-of-Bag Error

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

Key: trees are repeatedly fit to bootstrapped subsets of observations.

$\Rightarrow$ on average each tree use $\approx 2/3$ of the data to fit the tree.

i.e. $\approx 1/3$ of observations are NOT used to fit the tree (out-of-bag).

idea: We can predict the i^{th} response using all trees in which observation was OOB.

This leads to $\approx \frac{B}{3}$ predictions for i^{th} observation.

Average of predictions to get a single OOB prediction for i^{th} observation

$\Rightarrow$ we can get OOB predictions for each training obs to get OOB MSE (or OOB classification error)

which estimates TEST MSE

valid b/c only ever use predictions from trees that didn't use that point in fitting.

2.2 Interpretation

Bagging typically results in improved predictive performance (over a single tree) at the expense of interpretability

↳ one of the biggest advantages of trees !!

↳ no longer possible to represent model as a single tree

⟹ no longer know which variables are the most important to predict response.

What to do?

↳ obtain overall summary of importance using RSS (or Gini)

- record total amount RSS (or Gini) is decreased due to splits over a given predictor averaged over B trees
- large value indicates important predictor.

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.

But when building the trees, a random sample of m predictors is chosen as split candidates

↳ split only allowed to consider one of the candidates

↳ fresh sample of candidates every split

↳ typically $m \approx \sqrt{p}$.

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. Why?

Suppose there is one strong predictor and a number of moderate predictors in the data set.

In the collection of trees, most (or all) will have the top predictor as the top split!

$\Rightarrow$ all of the bagged trees will look very similar

$\Rightarrow$ predictions will be highly correlated.

i.e, bagging → averaging highly (positively) correlated values does not lead to much variance reduction!

RFs overcome this by forcing each split to consider only a subset of predictors

$\Rightarrow$ on average $\frac{p-m}{p}$ of splits will not even consider the strong predictor $\Rightarrow$ other predictors will have a chance.

The main difference between bagging and random forests is the choice of predictor subset size m. If $m = p \Rightarrow$ random forest = bagging.

Using small m helps w/ correlated predictors.

As with bagging, we're not concerned about overfitting w/ large B.

Can estimate OOB error and examine importance in same way.

4 Boosting

high bias, low variance

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 $\underbrace{\text{creep}}_{\text{"slow" learning}}$ toward an effective predictor.

$L_{01}(\hat{y}, y) = \sum_{i=1}^{n} (y_i \neq \hat{y}_i)$.

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.

f. This can be almost any classifier. Works best with low variance, high bias classifiers. Most people use f to be a tree w/ 2 terminal nodes ("stubs").

Algorithm (AdaBoost.M1)

1. Initialize the weights on the training data. (x_i, y_i) $i = 1, \dots, n$

$$w_{i1} = \frac{1}{n}, \quad i = 1, \dots, n$$

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

$\{-1,1\}$ — like a stub — (minimize) $\sum_{i=1}^{n} \mathbb{I}(y_i \neq \hat{f}(x_i))$

$$\text{let} \quad \overline{err}_1 = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}(y_i \neq \hat{f}_1(x_i))$$

$$\alpha_1 = \ln\left(\frac{1 - \overline{err}_1}{\overline{err}_1}\right)$$

3. Set new weights on the training data.

$$w_{i2} = \frac{1}{n} \exp(\alpha_1 \mathbb{I}(y_i \neq \hat{f}_1(x_i))) \quad i = 1, \dots, n$$

this upweights mis-classified observations by a factor of $\frac{1 - \overline{err}_1}{\overline{err}_1}$

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

a. Fit a $\mathcal{G}$-valued classifier $\hat{f}_m$ to training data to optimize $\sum_{i=1}^{n} w_{im} \mathbb{I}(y_i \neq \hat{f}(x_i))$.

b. Let $\overline{err}_m = \frac{1}{\sum_{i=1}^{n} w_{im}} \sum_{i=1}^{n} w_{im} \mathbb{I}(y_i \neq \hat{f}_m(x_i))$

c. set $\alpha_m = \ln\left(\frac{1 - \overline{err}_m}{\overline{err}_m}\right)$

d. update weights as

$$w_{i(m+1)} = w_{im} \exp(\alpha_m \mathbb{I}(y_i \neq \hat{f}_m(x_i))) \quad i = 1, \dots, n$$

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

$$\hat{f}(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m \hat{f}_m(x_i)\right)$$

classifiers with small $\overline{err}_m$ get big positive weights in the voting.

AdaBoost can be adapted for regression problems with a different loss function (which leads to different error, weights, etc.)

This works well!

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

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

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 $\mathrm{E}\left[\exp(-aY)|\boldsymbol{X} = \boldsymbol{u}\right]$, where

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

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 ($\mathbf{E}[\exp(-Yg(\boldsymbol{X})])$?

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:

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

and minimization of E_m is equivalent to minimization of

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

Let

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

then a bit of calculus shows that the optimizing β_m is

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

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

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

   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!