ECON 950 — Winter 2019
Prof. James MacKinnon

5. Trees and Forests

Tree-based methods partition the feature space into a set of rectangles.

One popular method is **CART**, for **classification and regression trees**.

A single tree typically does not perform all that well. But multiple trees can be combined into **random forests** that often perform very well.

5.1. Regression Trees

To grow a **regression tree**, we use **recursive binary splitting**.

1. Start by splitting the space into two regions. Find the predictor and split point that gives the best fit, where the predictor is the mean of Y in each region.

2. Next, split one of the regions into two regions, again using the predictor and split point that gives the best fit.

3. Next, do it again. The region we split could be the one we did not split in step 2, or it could be one of the ones we did split.
4. Continue until a stopping rule tells us to stop. For example, we might stop if all regions contain less than 5 observations.

Formally, if the space is divided into M regions, the response is

$$f(\boldsymbol{x}) = \sum_{m=1}^{M} c_m \mathbb{I}(\boldsymbol{x} \in R_m), \tag{1}$$

where we will estimate the c_m coefficients. If the objective is to minimize

$$\sum_{i=1}^{N} \big(y_i - f(\boldsymbol{x}_i)\big)^2, \tag{2}$$

the best choice for $\hat{c}_m$ is

$$\bar{y} \,|\, \boldsymbol{x}_i \in R_m = \frac{\sum_{i=1}^{N} y_i \mathbb{I}(\boldsymbol{x}_i \in R_m)}{\sum_{i=1}^{N} \mathbb{I}(\boldsymbol{x}_i \in R_m)} = \frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_m} y_i, \tag{3}$$

where N_m is the number of points in R_m. This is just the average of the y_i in region m.

Initially, we make one split so as to obtain two regions. If we split according to variable j at the point s, the regions are

$$R_1(j,s) = \{\boldsymbol{x} \,|\, x_j \leq s\} \ \text{ and } \ R_2(j,s) = \{\boldsymbol{x} \,|\, x_j > s\}. \tag{4}$$

Since the estimates $\hat{c}_1$ and $\hat{c}_2$ for regions 1 and 2, respectively, will be

$$\hat{c}_1 = \frac{1}{N_1} \sum_{\boldsymbol{x}_i \in R_1} y_i \quad \text{and} \quad \hat{c}_2 = \frac{1}{N_2} \sum_{\boldsymbol{x}_i \in R_2} y_i, \tag{5}$$

we want to choose j and s to minimize

$$\sum_{x_i \in R_1(j,s)} (y_i - \hat{c}_1)^2 + \sum_{x_i \in R_2(j,s)} (y_i - \hat{c}_2)^2. \tag{6}$$

This is not hard to do efficiently, although the programming may be tricky.

For each variable j, we just search over s to minimize (6). Then we choose the variable that, for the optimal choice of s, yields the lowest value.

Next, we split each of the two regions, and so on. See ESL-fig9.02.pdf.

The preferred strategy is to grow a very large tree, say T_0, stopping when every region is very small.

An alternative would be to stop as soon as the best proposed split has a sufficiently small effect on the fit. But this is too short-sighted.

The large tree, T_0, almost certainly suffers from over-fitting. We therefore **prune** the tree using **cost-complexity pruning**. Let

$$Q_m(T) = \frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_m} (y_i - \hat{c}_m)^2, \tag{7}$$

where T denotes some subtree of T_0, with terminal nodes (leaves) indexed by m. The quantity $Q_m(T)$ is a measure of **node impurity**.

Then define the **cost-complexity criterion**

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|. \tag{8}$$

Here $|T|$ is the number of nodes in the tree T, and α is a tuning parameter.

$C_\alpha(T)$ is simply the sum over all terminal nodes of the squared error losses, plus a penalty term. Note that the N_m factors in (7) and (8) cancel out.

For each α, we find the subtree that minimizes (8) by undoing some of the splits that we made previously.

This is done by **weakest-link pruning**, which collapses the split that causes the smallest increase in $C_\alpha(T)$.

The value of α is chosen by K-fold cross-validation, with K normally 5 or 10.

This yields the tuning parameter $\hat{\alpha}$ and the associated tree $T_{\hat{\alpha}}$.

It turns out that, as α increases, branches get pruned from the tree in a nested and predictable fashion. This make it easy to obtain the whole sequence of subtrees as a function of α.

5.2. Classification Trees

Since we were minimizing (2), the procedure just described was constructing a **regression tree**.

To construct a **classification tree**, we need to minimize something else.

Let

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_m} \mathbb{I}(y_i = k) \tag{9}$$

denote the proportion of class k observations in node m.

We can assign node m to class k if $\hat{p}_{mk}$ is higher for k than for any other class. The class with the highest proportion of the observations in node m is denoted $k(m)$.

Three measures of node impurity are **misclassification error**

$$\frac{1}{N_m} \sum_{1 \in R_m} \mathbb{I}\big((y_i \neq k(m)\big), \tag{10}$$

the **Gini index**

$$\sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}), \tag{11}$$

and the **cross-entropy** or **deviance**

$$-\sum_{k=1}^{K} \hat{p}_{mk}\log(\hat{p}_{mk}). \tag{12}$$

The misclassification error is not differentiable, and it is not sensitive to the values of $\hat{p}_{mk}$ except at points where $k(m)$ changes.

Instead of classifying each node, we could simply assign probabilities. Then the training error would be

$$\sum_{k=1}^{K}\sum_{m=1}^{M}\big(\mathbb{I}(y_i \in k)(1-\hat{p}_{mk}) + \mathbb{I}(y_i \notin k)\hat{p}_{mk}\big). \tag{13}$$

The variance of a 0-1 response with probability $\hat{p}_{mk}$ is $\hat{p}_{mk}(1-\hat{p}_{mk})$. Summing this over all classes gives the Gini index (11).

5.3. MARS: Multivariate Adaptive Regression Splines

MARS uses piecewise linear basis functions of the form $(x - t)_+$. The definition is

$$(x - t)_+ = \max(x - t, 0). \tag{14}$$

This is a linear spline with a knot at t.

The functions $(x - t)_+$ and $(t - x)_+$ are called a **reflected pair**.

If we allow for knots at every $\boldsymbol{x}_i$, we obtain the collection of basis functions

$$\mathcal{C} = \big\{(x_j - t)_+, (t - x_j)_+\big\},\ t \in \{x_{1j}, \ldots, x_{Nj}\},\ j = 1, \ldots, p. \tag{15}$$

The model is constructed like a forward stepwise regression, but using basis functions in $\mathcal{C}$ or products of them as inputs. It can be written as

$$f(\boldsymbol{x}) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(\boldsymbol{x}), \tag{16}$$

where M could be a big number. We simply estimate (16) by OLS.

The trick to MARS is how we add basis functions. We start with just the constant.

Then we try adding all products of a function h_m that is already in $\mathcal{M}$ with one of the reflected pairs in $\mathcal{C}$.

Each new term has the form

$$\hat{\beta}_{M+1} h_\ell(\boldsymbol{x})(x_j - t)_+ + \hat{\beta}_{M+2} h_\ell(\boldsymbol{x})(t - x_j)_+, \tag{17}$$

where $h_\ell(\boldsymbol{x})$ is already in the model, and $\hat{\beta}_{M+1}$ and $\hat{\beta}_{M+2}$ are least squares coefficients estimated along with the other M coefficients.

This seems insanely expensive, but it is not, because the values of $(x_j - t)_+$ and $(t - x_j)_+$ change in very simple ways as we vary the location of the knot t.

Let the x_{kj} be sorted from largest to smallest. Then, as t moves from x_{kj} to $x_{k+1,j}$, one element of each half of a reflected pair changes from 0 to $x_{k+1,j} - x_{kj}$, and the corresponding element of the other half changes from $x_{k+1,j} - x_{kj}$ to 0.

Except for the elements at the new and old knots, every element that was 0 before is still 0, and every element that was positive before has increased (or decreased) by $|x_{k+1,j} - x_{kj}|$.

By somewhat tricky programming, we can update the least squares fit in $O(1)$ operations as we move from one knot to the next.

This means that we can try every possible knot for each candidate set of reflected pairs in $O(N)$ operations.

See ESL-fig9.10.pdf.

5.4. MARS and CART

Although MARS was originally designed for regression, it can also be used for classification.

When there are just two classes, simply code them as 0 and 1.

When there are K classes, code them as 0/1 variables and use a multi-response MARS regression, with the same basis functions for every response. There will be $K-1$ such variables.

Suppose we make two changes to MARS:

- Replace the piecewise linear basis functions by step functions, $\mathbb{I}(x-t>0)$ and $\mathbb{I}(x-t\le 0)$.

- When a model term is multiplied by a candidate term, it gets replaced by the interaction and is not available for further interactions.

After these modifications, MARS is CART! At least, the forward part of modified MARS and the tree-growing part of CART are the same.

Multiplying a step function by a pair of reflected step functions is equivalent to splitting a node at the step.

The second modification implies that a node may not be split more than once.

MARS does actually have such a restriction. This makes it better at handling additive effects.

5.5. Hierarchical Mixtures of Experts

HME is a bit like CART, but the splits are probabilistic, and regression or logistic models are fit at the terminal nodes.

Probabilistic splits are sometimes called **soft splits**. See ESL-fig9.13.pdf.

The top **gating network** has output

$$g_j(\boldsymbol{x}, \boldsymbol{\gamma}_j) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{\gamma}_j)}{\sum_{k=1}^{K} \exp(\boldsymbol{x}^\top \boldsymbol{\gamma}_k)}, \quad j = 1, 2, \ldots, K. \tag{18}$$

where each $\boldsymbol{\gamma}_j$ has to be estimated.

The output in (18) is the probability of assigning an observation with feature vector $\boldsymbol{x}$ to branch k.

At the second level, the gating networks have the form.

$$g_{\ell \mid j}(\boldsymbol{x}, \boldsymbol{\gamma}_{j\ell}) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{\gamma}_{j\ell})}{\sum_{k=1}^{K} \exp(\boldsymbol{x}^\top \boldsymbol{\gamma}_{jk})}, \quad \ell = 1, 2, \ldots, K. \tag{19}$$

The output here is the probability of assigning $\boldsymbol{x}$ to branch ℓ conditional on assignment to branch j at the higher level.

At the lowest level, where there are **terminal nodes**, we have either a regression model or a logit model.

These models have parameter vectors $\boldsymbol{\theta}_{j\ell}$, which is confusing because j now indexes terminal nodes.

For the regression case, $\boldsymbol{\theta}_{j\ell} = [\boldsymbol{\beta}_{j\ell}, \sigma_{j\ell}]$ for the model

$$y = \boldsymbol{x}^\top \boldsymbol{\beta}_{j\ell} + \varepsilon, \quad \varepsilon \sim \mathrm{N}(0, \sigma^2_{j\ell}). \tag{20}$$

For the binary response case,

$$\Pr(y = 1 \,|\, \boldsymbol{x}, \boldsymbol{\theta}_{j\ell}) = \frac{1}{1 + \exp(-\boldsymbol{x}^\top \boldsymbol{\theta}_{j\ell})}\,. \tag{21}$$

Then the probability that $Y = y$ is

$$\Pr(y \,|\, \boldsymbol{x}, \boldsymbol{\Psi}) = \sum_{j=1}^{K} g_j(\boldsymbol{x}, \boldsymbol{\gamma}_j) \sum_{\ell=1}^{K} g_{\ell\,|\,j}(\boldsymbol{x}, \boldsymbol{\gamma}_{j\ell}) \Pr(\boldsymbol{y} \,|\, \boldsymbol{x}, \boldsymbol{\theta}_{j\ell}). \tag{22}$$

The loglikelihood is

$$\sum_{i=1}^{N} \log\bigl(\Pr(y \,|\, \boldsymbol{x}, \boldsymbol{\Psi})\bigr). \tag{23}$$

This can be estimated using the EM algorithm. We need latent variables Δ_j and $\Delta_{\ell\,|\,j}$. One of the Δ_j is 1, and the rest are 0. Similarly, for each j, one of the $\Delta_{\ell\,|\,j}$ is 1 and the rest are 0.

In the E-step, we take expectations of all these latent variables conditional on the parameters and the data.

In the M-step, we maximize the functions that appear in (23), using the expectations of the latent variables as weights.

Because the loglikelihood is a smooth function of the weights, conventional maximization algorithms work.

The soft splits allow HME models to capture cases where the transition from low to high response is gradual.

It is not clear how deep to make the tree.

Because the EM algorithm can take a long time to converge, HME models can be expensive, even though the individual logistic regressions are not too expensive.

Like **neural networks**, HME models use logistic regression extensively and have multiple layers.

How they are related is not entirely clear.

5.6. Bootstrap Methods

It seems natural to use bootstrap methods to measure a model's performance without employing a separate test datatset.

Many bootstrap methods, such as the residual bootstrap, wild bootstrap, and (most of all!) the parametric bootstrap assume that the model being estimated is true.

This makes them unsuitable for this purpose.

One method that does not is the **pairs bootstrap**, where each bootstrap sample is obtained by resampling from the $(\boldsymbol{x}_i, y_i)$ pairs, which we may denote $\boldsymbol{z}_i$.

The $N \times (p+1)$ matrix $\boldsymbol{Z}$ has typical row $\boldsymbol{z}_i$.

It is easy to draw B bootstrap datasets by resampling from the $\boldsymbol{z}_i$. We can call them $\boldsymbol{Z}_b^*$ for $b = 1, \ldots, B$.

We could then apply whatever methods we used with the actual training set to each of the $\boldsymbol{Z}_b^*$.

Let $\hat{f}_b^*(\boldsymbol{x}_i)$ be the predicted value from bootstrap sample b for the point $\boldsymbol{x}_i$.

One estimate of the **prediction error**, called **Err**, is

$$\widehat{\text{Err}}_{\text{boot}} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L\big(y_i, \hat{f}_b^*(\boldsymbol{x}_i)\big), \tag{24}$$

where $L(\cdot)$ is some loss function.

Just what $L(\cdot)$ is will depend on whether we are regressing or classifying and what we care about. For example, it might be the squared error $(y_i - \hat{f}_i)^2$.

Notice that we are comparing the actual outcome for observation i with the fitted value from each of the bootstrap samples.

The bootstrap samples are being used as training samples, and the actual training set is being used as the test sample. But they have many points in common.

This contrasts with cross-validation, where training is done on $K - 1$ folds, the omitted fold is used for testing, and we sum over the K folds.

Consider again the case of 1NN classification with two equal-sized classes and no information in the predictors.

The true error rate should be 0.5.

For the training sample, the error rate will be 0, because the nearest neighbor to $\boldsymbol{x}_i$ is itself.

For the bootstrap, the error rate will be the probability that any bootstrap sample contains $\boldsymbol{x}_i$. This is simply

$$1 - \left(1 - \frac{1}{N}\right)^N . \tag{25}$$

As $N \to \infty$, (25) tends to $1 - e^{-1} \approx 0.63212$.

Thus the probability that bootstrap sample b contains $\boldsymbol{x}_i$ is roughly 0.632.

For bootstrap samples that contain $\boldsymbol{x}_i$, their contribution to the term inside the double sum in (24) is 0. For bootstrap samples that do not contain $\boldsymbol{x}_i$, their contribution to that double sum is 0.5.

Therefore,

$$\widehat{\text{Err}}_{\text{boot}} \approx 0.632 \times 0 + 0.368 \times 0.5 = 0.184 << 0.50. \tag{26}$$

A better approach is to mimic cross-validation by using the **leave-one-out bootstrap**. For each observation i, we use only the bootstrap samples that do not contain that observation.

We obtain the leave-one-out bootstrap prediction error estimate

$$\widehat{\mathrm{Err}}^{(1)} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L\big(y_i, \hat{f}_b^*(\boldsymbol{x}_i)\big), \tag{27}$$

where C^{-i} denotes the set of indices of bootstrap samples that do not contain observation i, and $|C^{-i}|$ is the number of such samples.

Of course, we may need B to be fairly large to ensure that $|C^{-i}| > 0$ for all i.

$\widehat{\mathrm{Err}}^{(1)}$ solves the overfitting problem, but it has another problem.

Even though each bootstrap sample contains N observations, on average it only contains $0.632N$ *distinct* observations.

Thus it may be biased, in roughly the same way that 3-fold cross-validation, which uses $2N/3$ observations, would be biased.

One crude (but theoretically sophisticated) solution is the **.632 estimator**

$$\widehat{\mathrm{Err}}^{(.632)} = 0.368 \times \overline{\mathrm{err}} + 0.632 \times \widehat{\mathrm{Err}}^{(1)}. \tag{28}$$

It is a weighted average of the training error rate and the leave-one-out bootstrap prediction error estimate.

ESL claims that (28) works well in “light fitting” situations but not in overfit ones.

They give an example where $\widehat{\mathrm{Err}}^{(1)}$ works perfectly and $\widehat{\mathrm{Err}}^{(.632)}$ is too optimistic.

The **no-information error rate** γ is defined to be the error rate for our prediction rule if the y_i and $\boldsymbol{x}_i$ were actually independent.

An estimate is

$$\hat{\gamma} = \sum_{i=1}^{N} \sum_{i'=1}^{N} L\big(y_i, f(\boldsymbol{x}_{i'})\big). \tag{29}$$

Consider a dichotomous classification problem. Let $\hat{p}_1$ be the observed proportion of the y_i that equal 1 and $\hat{q}_1$ be the observed proportion of the $f(\boldsymbol{x}_{i'})$ that equal 1.

$$\hat{\gamma} = \hat{p}_1(1 - \hat{q}_1) + \hat{q}_1(1 - \hat{p}_1).$$

With 1NN, $\hat{p}_1 = \hat{q}_1$, so that $\hat{\gamma} = 2\hat{p}_1(1 - \hat{p}_1)$, which equals 0.5 when $\hat{p}_1 = 0.5$.

The **relative overfitting rate** is

$$\hat{R} = \frac{\widehat{\mathrm{Err}}^{(1)} - \overline{\mathrm{err}}}{\hat{\gamma} - \overline{\mathrm{err}}}. \tag{30}$$

Then we get the .632+ estimator

$$\widehat{\mathrm{Err}}^{(.632+)} = (1 - \hat{w}) \times \overline{\mathrm{err}} + \hat{w} \times \widehat{\mathrm{Err}}^{(1)}. \tag{31}$$

This looks like (28), but instead of using weights $1-0.632$ and 0.632, we use weights $1 - \hat{w}$ and $\hat{w}$, where

$$\hat{w} = \frac{.632}{1 - .368\hat{R}}. \tag{32}$$

In a case with extreme overfitting like 1NN, $\hat{R} = 1$, so that $\hat{w} = 1$, and

$$\widehat{\mathrm{Err}}^{(.632+)} = \widehat{\mathrm{Err}}^{(1)}. \tag{33}$$

In cases with less overfitting,

$$\overline{\text{err}} \leq \widehat{\text{Err}}^{(.632+)} \leq \widehat{\text{Err}}^{(1)}. \tag{34}$$

5.7. Bagging

The idea of **bootstrap aggregation**, or **bagging**, is to generate a prediction from each of B bootstrap samples and average them.

This can be used with many different prediction methods, but it only makes sense when the prediction is a nonlinear function of the data, as it is for CART.

Again, let $\hat{f}_b^*(\boldsymbol{x})$ be the prediction for the point $\boldsymbol{x}$ based on bootstrap sample b. Then the bagging estimate is

$$\hat{f}_{\text{bag}}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b^*(\boldsymbol{x}). \tag{35}$$

This is an approximation to $\mathrm{E}\big(\hat{f}_b^*(\boldsymbol{x})\big)$ based on the empirical distribution of the points $(\boldsymbol{x}_i, y_i)$.

We can think of $\mathrm{E}\big(\hat{f}_b^*(\boldsymbol{x})\big)$ as an estimate of the "ideal" aggregating estimate, $f_{\mathrm{ag}}(\boldsymbol{x})$, in which the expectation is taken over the actual distribution of $(\boldsymbol{x}, y)$.

Of course, $f_{\mathrm{ag}}(\boldsymbol{x})$ is infeasible, because we do not know the distribution of the $(\boldsymbol{x}, y)$. We have to hope that the empirical distribution of the $(\boldsymbol{x}_i, y_i)$ provides a good approximation.

The expectation of the bagging estimate will differ from $\hat{f}(\boldsymbol{x})$ only when the latter is a nonlinear or adaptive function of the data. Thus bagging regressions is not helpful.

When $\hat{f}(\boldsymbol{x})$ is linear in $\boldsymbol{y}$, averaging over the $\hat{f}_b^*(\boldsymbol{x})$ is equivalent to averaging over the $\boldsymbol{y}_b^*$ and then applying a linear operator to the average.

But when $\hat{f}(\boldsymbol{x})$ is nonlinear in $\boldsymbol{y}$, the bagging estimate may be substantially more efficient than $\hat{f}(\boldsymbol{x})$ itself. It is often helpful for trees.

Suppose we could draw bootstrap samples from the actual distribution of the $(\boldsymbol{x}_i, y_i)$. Applying our prediction procedure to such a sample would yield $\hat{f}^*(\boldsymbol{x})$.

Thus $\hat{f}^*(\boldsymbol{x})$ has mean $f_{\mathrm{ag}}(\boldsymbol{x})$.

Evidently,

$$\begin{aligned} \mathrm{E}\big(y - \hat{f}^*(\boldsymbol{x})\big)^2 &= \mathrm{E}\big(y - f_{\mathrm{ag}}(\boldsymbol{x}) + f_{\mathrm{ag}}(\boldsymbol{x}) - \hat{f}^*(\boldsymbol{x})\big)^2 \\ &= \mathrm{E}\big(y - f_{\mathrm{ag}}(\boldsymbol{x})\big)^2 + \big(\hat{f}^*(\boldsymbol{x}) - f_{\mathrm{ag}}(\boldsymbol{x})\big)^2 \\ &\geq \mathrm{E}\big(y - f_{\mathrm{ag}}(\boldsymbol{x})\big)^2. \end{aligned} \tag{36}$$

The inequality arises because of the second term in the second line, which is the variance of $\hat{f}^*(\boldsymbol{x})$ around its mean of $f_{\mathrm{ag}}(\boldsymbol{x})$.

So in this (admittedly infeasible) case, aggregating reduces variance.

Since bagging is the feasible analog of what we did in (36), it seems plausible that it too will reduce variance.

The argument in (36) relied upon the additivity of squared bias and variance, which is not true for classification with 0-1 loss.

Unfortunately, bagging a bad classifier can make it worse.

In the Bayesian context, we can think of $\hat{f}(\boldsymbol{x})$ as a posterior mode and $\hat{f}_{\mathrm{bag}}(\boldsymbol{x})$ as posterior mean.

For symmetric, unimodal distributions like the Gaussian, they are identical. But for skewed distributions, they could be quite different.

Because it is the posterior mean (not the posterior mode) that minimizes squared error loss, it is not surprising that bagging often helps.

5.8. Random Forests

"Random forests" is a relatively recent (Breiman, 2001) method that is easy to use and often works well.

Interestingly, Breiman was 72 or 73 when this paper appeared, and it has over 42,700 citations. Unfortunately, he died four years later.

As we just saw, bagging, that is, averaging over the predictions from a number of bootstrap samples, reduces variance but not bias.

This can work well if all the models are approximately unbiased and not very correlated with each other.

If we average B random variables y_b each with variance σ^2, the variance of the average is σ^2/B.

If the random variables are correlated, with variance σ^2 and covariance $\rho\sigma^2$, we instead find that

$$\begin{aligned}\mathrm{Var}(\bar{y}) &= \frac{1}{B^2}\sum_{b=1}^{B}\mathrm{Var}(y_b) + \frac{2}{B^2}\sum_{b=1}^{B}\sum_{b'=b+1}^{B}\mathrm{Cov}(y_b, y_{b'}) \\ &= \frac{1}{B}\sigma^2 + \frac{1}{B^2}B(B-1)\rho\sigma^2 \\ &= \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.\end{aligned} \tag{37}$$

As B increases, the second term tends to zero, like σ^2/B in the uncorrelated case.

But the first term does not tend to zero. No matter how many random variables we average (in this case, predictions from different models), we cannot reduce the variance below $\rho\sigma^2$.

Random forests is a form of bagging applied to classification trees, but modified so as to reduce correlation across the trees.

The trick is not to allow all possible splits. Instead, the algorithm randomly selects m out of p possible variables for splitting.

Typically, m is quite small, like $\sqrt{p}$. That is the default for classification. The default for regression is $p/3$.

If $m = p$, random forests is simply bagging applied to trees.

Here is the random forests algorithm:

For $b = 1$ to B,

1. Draw a bootstrap sample $\boldsymbol{Z}^{*b}$ from the training data.

2. Grow a random-forest tree T^b by repeating the following steps for each terminal node until the minimum node size is reached:

    i. Select m variables at random from the p variables;

    ii. Pick the best variable and split-point among the m variables, and split that terminal node into two daughter nodes.

3. Output the ensemble of trees $\{T_b\}_1^B$.

4. For regression, the prediction is

$$\hat{f}_{\mathrm{rf}}^{B}(\boldsymbol{x}) = \frac{1}{B}\sum_{b=1}^{B} T_b(\boldsymbol{x}). \tag{38}$$

For classification, each tree makes a prediction. For the random forest, choose the class that gets the most votes.

Like other classification methods, this may give unsatisfactory results if the conditional probabilities of some classes are never very large.

Would it be better to average the estimated probabilities?

Reducing m tends to reduce the correlation between any pair of trees and hence reduce the variance in (37).

5.9. Out-of-bag samples

It is possible to perform a sort of cross-validation while constructing a random forest, without explicitly constructing cross-validation subsamples.

For each observation, keep track of whether it appears in bootstrap sample b or not.

If it appears, omit that bootstrap sample in the **out-of-bag**, or **OOB**, predictor.

The usual predictor (38) is based on all bootstrap samples, while the OOB one omits approximately 36.8% of them.

Use the OOB errors in the same way you would normally use errors from cross-validation samples, in this case to decide how many trees to average.

For regression, simply average the OOB predictors and find the resulting residuals to obtain mean squared error.

For classification, we can take a majority vote or use the average of the predictions.

Figures 15.1 to 15.3 from ESL compare random forests with other methods; see ESL-fig15.01-03.pdf.

In these examples, random forests beat bagging but perform less well than boosting.