# Random Forests

**Chapter** *in* Machine Learning · January 2011

DOI: 10.1007/978-1-4419-9326-7_5 · Source: DBLP



**3 authors:**

Adele Cutler
Utah State University
**44** PUBLICATIONS **7,203** CITATIONS
SEE PROFILE

David Richard Cutler
Utah State University
**49** PUBLICATIONS **6,165** CITATIONS
SEE PROFILE

John R Stevens
Utah State University
**117** PUBLICATIONS **4,554** CITATIONS
SEE PROFILE

# Random Forests

Adele Cutler, D. Richard Cutler and John R. Stevens

## 1 Introduction

Random Forests were introduced by Leo Breiman [6] who was inspired by earlier work by Amit and Geman [2]. Although not obvious from the description in [6], Random Forests are an extension of Breiman's bagging idea [5] and were developed as a competitor to boosting. Random Forests can be used for either a categorical response variable, referred to in [6] as "classification", or a continuous response, referred to as "regression". Similarly, the predictor variables can be either categorical or continuous.

From a computational standpoint, Random Forests are appealing because they

- naturally handle both regression and (multiclass) classification;
- are relatively fast to train and to predict;
- depend only on one or two tuning parameters;
- have a built in estimate of generalization error;
- can be used directly for high-dimensional problems;
- can easily be implemented in parallel.

Statistically, Random Forests are appealing because of the additional features they provide, such as

- measures of variable importance;
- differential class weighting;
- missing value imputation;
- visualization;

Adele Cutler
Utah State University, Logan, UT 84322-3900, e-mail: adele.cutler@usu.edu,

D. Richard Cutler
Utah State University, Logan, UT 84322-3900, e-mail: richard.cutler@usu.edu,

John R. Stevens
Utah State University, Logan, UT 84322-3900, e-mail: John.R.Stevens@usu.edu.

- outlier detection;
- unsupervised learning.

This chapter gives an introduction to the Random Forest method for classification and regression, including a brief description of the types of classification and regression trees used in the Random Forests algorithm. The chapter describes how out of bag data are used not only to give a fast estimate of generalization error, but also to estimate variable importance. A discussion of some important practical issues such as tuning the algorithm and weighting classes to deal with unequal sample sizes is also included. Methods for finding Random Forest proximities and using them to give illuminating plots as well as imputing missing values are presented. Finally, references to extensions of the Random Forest method are given.

## 2 The Random Forest Algorithm

As the name suggests, a Random Forest is a tree-based ensemble with each tree depending on a collection of random variables. More formally, for a $p$-dimensional random vector $X = (X_1, \ldots, X_p)^T$ representing the real-valued input or predictor variables and a random variable $Y$ representing the real-valued response, we assume an unknown joint distribution $P_{XY}(X,Y)$. The goal is to find a prediction function $f(X)$ for predicting $Y$. The prediction function is determined by a loss function $L(Y, f(X))$ and defined to minimize the expected value of the loss

$$E_{XY}(L(Y, f(X))) \tag{1}$$

where the subscripts denote expectation with respect to the joint distribution of $X$ and $Y$.

Intuitively, $L(Y, f(X))$ is a measure of how close $f(X)$ is to $Y$; it penalizes values of $f(X)$ that are a long way from $Y$. Typical choices of $L$ are *squared error loss* $L(Y, f(X)) = (Y - f(X))^2$ for regression and *zero-one loss* for classification:

$$L(Y, f(X)) = I(Y \neq f(X)) = \begin{cases} 0 \text{ if } Y = f(X) \\ 1 \text{ otherwise.} \end{cases} \tag{2}$$

It turns out (see, for example, [10] Sect. 2.4) that minimizing $E_{XY}(L(Y, f(X)))$ for squared error loss gives the conditional expectation

$$f(x) = E(Y|X = x) \tag{3}$$

otherwise known as the *regression function*. In the classification situation, if the set of possible values of $Y$ is denoted by $\mathscr{Y}$, minimizing $E_{XY}(L(Y, f(X)))$ for zero-one loss gives

$$f(x) = arg\max_{y \in \mathscr{Y}} P(Y = y|X = x), \tag{4}$$

otherwise known as the *Bayes rule*.

Ensembles construct $f$ in terms of a collection of so-called "base learners" $h_1(x),\ldots,h_J(x)$ and these base learners are combined to give the "ensemble predictor" $f(x)$. In regression, the base learners are averaged

$$f(x) = \frac{1}{J}\sum_{j=1}^{J} h_j(x), \tag{5}$$

while in classification, $f(x)$ is the most frequently predicted class ("voting")

$$f(x) = arg\max_{y\in\mathscr{Y}} \sum_{j=1}^{J} I(y = h_j(x)). \tag{6}$$

In Random Forests the $j$th base learner is a tree denoted $h_j(X,\Theta_j)$, where $\Theta_j$ is a collection of random variables and the $\Theta_j$'s are independent for $j = 1,\ldots,J$. Although the definition of a Random Forest is very general, they are almost invariably implemented in the specific way described in Sect. 2.2. To understand the Random Forest algorithm, it is important to have a fundamental knowledge of the type of trees used as base learners.

## *2.1 Introduction to Classification and Regression Trees*

The trees used in Random Forests are based on the binary recursive partitioning trees in the monograph [4] and also described in [10], [14] and [26]. These trees partition the predictor space using a sequence of binary partitions ("splits") on individual variables. The "root" node of the tree comprises the entire predictor space. The nodes that are not split are called "terminal nodes" and form the final partition of the predictor space. Each nonterminal node splits into two descendant nodes, one on the left and one on the right, according to the value of one of the predictor variables. For a continuous predictor variable, a split is determined by a split-point; points for which the predictor is smaller than the split-point go to the left, the rest go to the right (See Fig. 1).

A categorical predictor variable $X_i$ takes values from a finite set of categories $S_i = \{s_{i,1},\ldots,s_{i,m}\}$. A split sends a subset of these categories $S \subset S_i$ to the left and the remaining categories to the right (See Fig. 2).

The particular split a tree uses to partition a node into its two descendants is chosen by considering every possible split on every predictor variable and choosing the "best" according to some criterion. In the regression context, if the response values at the node are $y_1,\ldots,y_n$, a typical splitting criterion is the mean squared residual at the node

$$Q = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2 \tag{7}$$

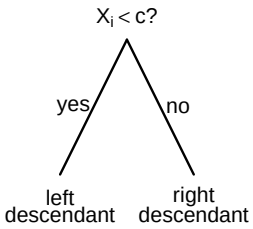


**Fig. 1** Splitting on a continuous predictor variable $X_i$, using split-point $c$.

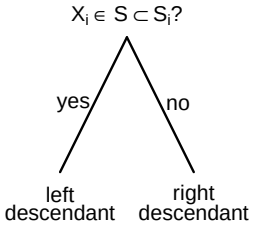


**Fig. 2** Splitting on a categorical predictor variable $X_i$, using subset $S \subset S_i$.

where $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the predicted value at the node (the average of the response values). In the classification context where there are $K$ classes denoted $1,\ldots,K$, a typical splitting criterion is the Gini index

$$Q = \sum_{k \neq k'}^{K} \hat{p}_k \hat{p}_{k'} \tag{8}$$

where $\hat{p}_k$ is the proportion of class $k$ observations in the node:

$$\hat{p}_k = \frac{1}{n}\sum_{i=1}^{n} I(y_i = k). \tag{9}$$

The splitting criterion gives a measure of "goodness of fit" (regression) or "purity" (classification) for a node, with large values representing poor fit (regression) or an impure node (classification). A candidate split creates two descendant nodes,

one on the left and one on the right. Denoting the splitting criteria for the two candidate descendants as $Q_L$ and $Q_R$ and their sample sizes by $n_L$ and $n_R$, the split is chosen to minimize $Q_{\text{split}} = n_L Q_L + n_R Q_R$.

For a continuous predictor variable, finding the best possible split entails sorting the values of the predictor and considering splits between every distinct pair of consecutive values. Typically the midpoint of the interval is used, although any value in the interval would suffice. The values of $Q_L$, $Q_R$ and hence $Q_{\text{split}}$ are computed for each of these possible split points, usually using a fast update algorithm. For a categorical predictor variable, $Q_L$, $Q_R$ and $Q_{\text{split}}$ are computed for all possible ways of choosing a subset of categories to go to each descendant node.

Once a split has been selected, the data are partitioned into the two descendant nodes and each of these nodes is treated in the same way as the original node. The procedure continues recursively until a stopping criterion is met. For example, the procedure may stop when all unsplit nodes contain fewer than some fixed number of cases. When the stopping criterion is met, unsplit nodes are called "terminal nodes". A predicted value is obtained for all observations in the terminal nodes by averaging the response for regression problems or computing the most frequent class for classification problems. To predict at a new point, its set of predictor values are used to pass the point down the tree until it falls into a terminal node and the prediction for the terminal node is used as the prediction for the new point.

**Algorithm 1** Binary Recursive Partitioning
Let $\mathscr{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$ denote the training data, with $x_i = (x_{i,1}, \ldots, x_{i,p})^T$.

1. Start with all observations $(x_1, y_1), \ldots, (x_N, y_N)$ in a single node.
2. Repeat the following steps recursively for each unsplit node until the stopping criterion is met:
   a. Find the best binary split among all binary splits on all $p$ predictors.
   b. Split the node into two descendant nodes using the best split (step 2a).
3. For prediction at $x$, pass $x$ down the tree until it lands in a terminal node. Let $k$ denote the terminal node and let $y_{k_1}, \ldots, y_{k_n}$ denote the response values of the training data in node $k$. Predicted values of the response variable are given by:
   - $\hat{h}(x) = \bar{y_k} = \frac{1}{n}\sum_{i=1}^{n} y_{k_i}$ for regression
   - $\hat{h}(x) = arg\max_y \sum_{i=1}^{n} I(y_{k_i} = y)$ for classification, where $I(y_{k_i} = y) = 1$ if $y_{k_i} = y$ and 0 otherwise.

Often, trees are deliberately grown larger than necessary and "pruned" back to prevent over fitting [4]. Although pruning is very important to prevent over-fitting for stand-alone trees, it is not used in Random Forests, so it will not be described here, but the interested reader is referred to [4] or [14].

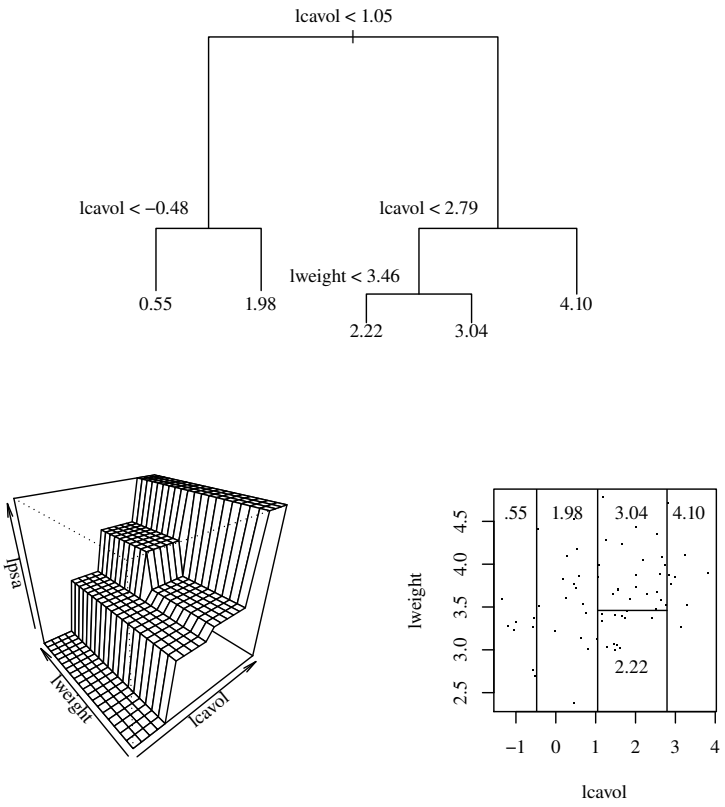


**Fig. 3** Regression tree for 2-dimensional prostate cancer data (Example 1). The top panel shows the tree diagram, the bottom left contains a perspective plot of the fitted regression surface, the bottom right shows the partitioning of the predictor space.

**Example 1** Prostate Cancer Data

To illustrate regression trees, data from the prostate cancer study of [23], also studied in [10] is used. The response variable is the level of prostate-specific antigen (lpsa). The predictor variables are log cancer volume (lcavol), log prostate weight (lweight), age, log of the amount of benign prostatic hyperplasia (lbph), seminal vesicle invasion (svi), log of capsular penetration (lcp), Gleason score (gleason), and percent of Gleason scores 4 or 5 (pgg45). A regression tree was fit to two of the predictor variables, namely, log cancer volume (lcavol) and log prostate weight (lweight). The top panel of Figure 3 shows the regression tree. At each node, cases

that satisfy the inequality go to the left, while ones that do not satisfy the inequality go to the right. Each terminal node results in a single predicted value, namely the average value of the response for the observations falling into the node. At the bottom left, Figure 3 shows a perspective plot of the piecewise linear regression surface corresponding to the regression tree in the top panel. On the bottom right, Figure 3 shows the partitioning of the predictor space. For continuous predictors such as these, the splits are parallel to the coordinate axes and the predictor space is divided into (hyper-) rectangles, each with a single predicted value. Each of the 5 rectangles corresponds to one of the terminal nodes in the tree.

Trees are popular for a wide range of problems, in part because trees can model complex interactions. The rank-based nature of the splits makes trees robust to outliers and insensitive to monotone transformations of the predictor variables. A summary of the characteristics that make trees popular, even for low-dimensional problems, is ([10] Sect. 10.7) that trees:

- can model interactions;
- naturally handle both regression and (multiclass) classification;
- naturally handle both continuous and categorical predictor variables;
- handle missing values in the predictor variables;
- are robust to outliers in the predictor variables;
- are insensitive to monotone transformations of the predictor variables;
- scale well for large sample sizes;
- deal well with irrelevant predictor variables.

Neither support vector machines nor neural networks rate highly on any of the above characteristics ([10] Sect. 10.7). On the downside, regression trees have sharp jumps in the predictions at the edges of the nodes. Also they

- are not good at capturing relationships involving linear combinations of predictor variables;
- are known to be unstable in the sense that if the data are perturbed slightly, the tree can change substantially;
- are not as accurate as some of the more recently developed methods.

Trees enjoy a mixed reception when it comes to interpretability. Tree diagrams are easily understood, but interpretation can be difficult because adjacent or nearby rectangles can appear in quite distant parts of the tree. A less obvious problem occurs when two or more predictor variables are highly correlated within a node. Such variables are called surrogates, and lead to similar splits of the node. However, they make interpretation more difficult because *different* surrogates may be selected for splits at this and descendant nodes. If there are only a few predictor variables, good software can help keep track of surrogates, but in very high-dimensional examples the task becomes much more difficult and it may be impossible to extract a coherent story from the tree diagram.

Perhaps the single largest drawback of trees is that they are not as accurate as more recently developed methods. However, they are the building blocks of many ensemble methods including Random Forests.

## 2.2 Random Forest Definition

As mentioned earlier in this Section, a Random Forest uses trees $h_j(X,\Theta_j)$ as base learners. For training data $\mathscr{D} = \{(x_1,y_1),\ldots,(x_N,y_N)\}$, where $x_i = (x_{i,1},\ldots,x_{i,p})^T$ denotes the $p$ predictors and $y_i$ denotes the response, and a particular realization $\theta_j$ of $\Theta_j$, the fitted tree is denoted $\hat{h}_j(x,\theta_j,\mathscr{D})$. While this is the original formulation from Breiman [6], in practice the random component $\theta_j$ is not considered explicitly but is implicitly used to inject randomness in two ways. First, as with bagging, each tree is fit to an independent bootstrap sample from the original data. The randomization involved in bootstrap sampling gives one part of $\theta_j$. Second, when splitting a node, the best split is found over a randomly selected subset of $m$ predictor variables instead of all $p$ predictors, independently at each node. The randomization used to sample the predictors gives the remaining part of $\theta_j$.

The trees are grown without pruning. Initially, Breiman [6] suggested growing them until the terminal nodes were pure (classification) or until there were fewer than a prespecified number of data points in each terminal node (regression). More recently [21] suggests controlling the maximum number of terminal nodes.

The resulting trees are combined by unweighted voting if the response is categorical (classification) or unweighted averaging if the response is continuous (regression).

**Algorithm 2** Random Forests
Let $\mathscr{D} = \{(x_1,y_1),\ldots,(x_N,y_N)\}$ denote the training data, with $x_i = (x_{i,1},\ldots,x_{i,p})^T$. For $j = 1$ to $J$:

1. Take a bootstrap sample $\mathscr{D}_j$ of size $N$ from $\mathscr{D}$.
2. Using the bootstrap sample $\mathscr{D}_j$ as the training data, fit a tree using binary recursive partitioning (Sect. 2.1):
   a. Start with all observations in a single node.
   b. Repeat the following steps recursively for each unsplit node until the stopping criterion is met:
      i. Select $m$ predictors at random from the $p$ available predictors.
      ii. Find the best binary split among all binary splits on the $m$ predictors from step i.
      iii. Split the node into two descendant nodes using the split from step ii.

To make a prediction at a new point $x$,

- $\hat{f}(x) = \frac{1}{J}\sum_{j=1}^{J}\hat{h}_j(x)$ for regression
- $\hat{f}(x) = arg\max_y \sum_{j=1}^{J} I(\hat{h}_j(x) = y)$ for classification

where $\hat{h}_j(x)$ is the prediction of the response variable at $x$ using the $j$th tree (Algorithm 1).

## 2.3 Using Out-Of-Bag Data

When a bootstrap sample is taken from the data, some observations do not make it into the bootstrap sample. These are called "out-of-bag data", and are extremely useful for estimating generalization error and variable importance (see Sect. 3).

To estimate generalization error, first note that if the trees are large, predictions naively obtained using all the trees will be overly optimistic if used to predict the response variable for observations that were in the training set $\mathscr{D}$. For this reason, prediction of the response variable for observations that were in the training set is only done using trees for which the observation is out-of-bag. These predictions are called out-of-bag predictions.

**Algorithm 3** Out-of-Bag Predictions
Let $\mathscr{D}_j$ denote the $j$th bootstrap sample and $\hat{h}_j(x)$ denote the prediction at $x$ from the $j$th tree, for $j = 1,\ldots,J$. For $i = 1$ to $N$:

1. Let $\mathscr{J}_i = \{j : (x_i,y_i) \notin \mathscr{D}_j\}$ and let $J_i$ be the cardinality of $\mathscr{J}_i$ (Algorithm 2).
2. Define the out-of-bag prediction at $x_i$ to be
   - $\hat{f}_{\text{oob}}(x_i) = \frac{1}{J_i}\sum_{j\in\mathscr{J}_i}\hat{h}_j(x_i)$ for regression
   - $\hat{f}_{\text{oob}}(x_i) = arg\max_y \sum_{j\in\mathscr{J}_i} I(\hat{h}_j(x_i) = y)$ for classification

   where $\hat{h}_j(x_i)$ is the prediction of the response variable at $x_i$ using the $j$th tree (Algorithm 1).

For regression with squared error loss, generalization error is typically estimated using the out-of-bag mean squared error (MSE):

$$MSE_{\text{oob}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}_{\text{oob}}(x_i))^2 \tag{10}$$

where $\hat{f}_{\text{oob}}(x_i)$ is the out-of-bag prediction for observation $i$.

For classification with zero one loss, generalization error rate is estimated using the out-of-bag error rate:

$$E_{\text{oob}} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq \hat{f}_{\text{oob}}(x_i)). \tag{11}$$

A common misconception is that the out-of-bag error rate is obtained by computing the out-of-bag error rate for each tree, and averaging these error rates to give the out-of-bag error rate for the forest. Instead, we use the error rate of the out-of-bag predictions. This allows us to obtain a classwise error rate for each class, and an out-of-bag "confusion matrix" by cross-tabulating $y_i$ and $\hat{f}_{\text{oob}}(x_i)$.

**Example 2** Mease and Wyner Data

To illustrate the use of the out-of-bag data, we consider a simulation model used by Mease and Wyner [17]. For input variables $X_1,\ldots,X_p$ independently taken from the standard uniform distribution U[0,1], the response variable $Y \in \{0,1\}$ is generated using

$$P(Y=1|X_1,\ldots,X_p) = \begin{cases} q & \text{if } \sum_{l=1}^{L} X_l \le L/2 \\ 1-q & \text{otherwise.} \end{cases}$$

For $q < 0.5$, the Bayes' rule classifies an observation $x_1,\ldots,x_p$ into class 0 if $\sum_{l=1}^{L} x_l \le L/2$ and into class 1 otherwise. For $q > 0.5$, the class labels are reversed, and in both cases the Bayes' error is $q$. In this way, the first $L$ predictors are important and the remaining $p-L$ predictors are noise. Using $L = p = 2$ and $q = 0.1$, Fig. 4 shows the out of bag error estimate and the test set error estimate for a training set of size $N = 1000$ and a test set of size 10,000 as the number of trees increases from $J = 1$ to $J = 500$. The out-of-bag error rate tracks the test set error rate quite closely in this Fig. 4. We chose a case for which the out-of-bag error rate and test set error rate were quite similar. Other runs showed the out-of-bag error rate to be somewhat higher or somewhat lower than the test set error rate. Table 1 shows the out-of-bag confusion matrix and test set confusion matrix for the run shown in Fig. 4. Note that the out-of-bag confusion matrix is obtained using the out-of-bag prediction for each observation in the training set, against the nominal class.

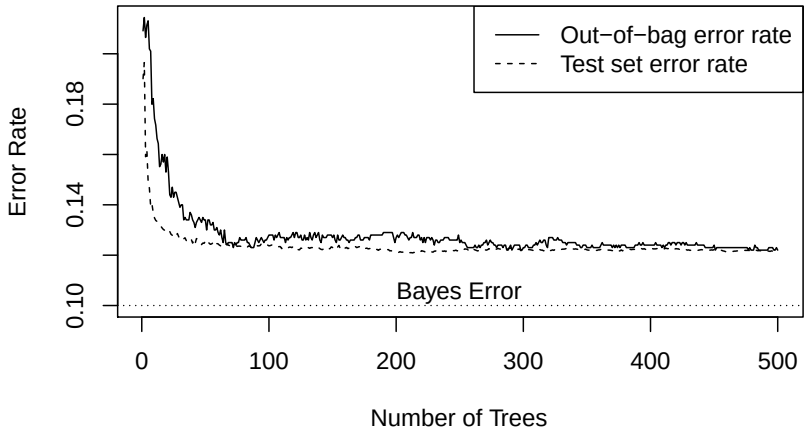


**Fig. 4** Out-of-bag and test set error rate for Mease and Wyner data (Example 2).

**Table 1** Out-of-bag and test set confusion matrices for Mease and Wyner data (Example 2).

Out-of-bag confusion matrix

| | Predicted | | |
|---|---|---|---|
| | Class 0 | Class 1 | Total |
| Nominal Class 0 | 417 | 58 | 475 |
| Nominal Class 1 | 64 | 461 | 525 |
| Total | 481 | 519 | 1000 |

Test set confusion matrix

| | Predicted | | |
|---|---|---|---|
| | Class 0 | Class 1 | Total |
| Nominal Class 0 | 4409 | 626 | 5035 |
| Nominal Class 1 | 590 | 4375 | 4965 |
| Total | 4999 | 5001 | 10000 |

## *2.4 Tuning*

Although Random Forests have the reputation of working quite well right out of the box, there are three parameters that may be tuned to give improved accuracy for particular situations:

- $m$, the number of randomly selected predictor variables chosen at each node
- $J$, the number of trees in the forest
- *tree size*, as measured by the smallest node size for splitting or the maximum number of terminal nodes.

The only one of these parameters to which Random Forests is somewhat sensitive appears to be $m$. In classification, the standard default is $m = \sqrt{M}$, where $M$ is the total number of predictors. In regression, the default is $m = N/3$, where $N$ is the sample size. If tuning is necessary, $m$ can be chosen using the out-of-bag error rate, but then this no longer gives an unbiased estimate of generalization error. However, typically Random Forests are not very sensitive to $m$, so fine-tuning is not required and overfitting effects due to choice of $m$ should be relatively small, as demonstrated by [9].

For many ensemble methods, generalization error initially decreases as $J$ increases, but at some point $J$ becomes too large and overfitting sets in, with an associated increase in generalization error. This is not the case with Random Forests. For small values of $J$, the out-of-bag estimate can be unstable and inaccurate. However, as $J$ increases Breiman showed [6] that the generalization error for Random Forests converges almost surely to a limit. In practice, this means $J$ can be chosen as large as desired, without fear of increasing the generalization error. The only real concern with $J$ is that it not be too small, and usually the out-of-bag error rate can be used to decide when $J$ is large enough that the estimated generalization error has stabilized. Often a plot such as that shown in Fig. 4 is used to decide whether or not $J$ is large enough.

Breiman's original work [6] recommends growing very large trees. In a recent paper by Segal and Xiao [21], the authors give a classification example for which a forest of large trees overfits and suggest this was not observed in Breiman's original work because the benchmark datasets all came from the University of California at Irvine (UCI) repository and happen to share properties that make large trees nearly optimal. In problems for which large trees overfit, users can tune using either the number of nodes or the smallest nodesize. Out-of-bag error rates can be used to

choose the tuning parameter, understanding that such use will lead to a bias in the estimated generalization error.

### *2.5 Weighting*

Unbalanced data sets, where some classes are much smaller than others, present a challenge to many classifiers. A naive classifier will work on getting the large classes right, while allowing a high error rate for the small classes. Random Forests has an effective method for weighting the classes to give balanced results in unbalanced data (`www.math.usu.edu/~adele/forests`). One reason to do this is that the important predictor variables may be different when the method is forced to pay greater attention to a small class. Even in the balanced case, the weights can be adjusted to give lower error rates to decisions that have a high misclassification cost. For example, it is often more serious to incorrectly conclude that someone is healthy than it would be to incorrectly conclude that someone is ill. Example 3 in Sect. 3.1 illustrates the effect of different weights on permutation variable importance.

## 3 Variable Importance

Measures of the importance of the predictor variables are useful for variable selection and for interpreting the fitted forest. While it is standard in many applications to run a principal components analysis (PCA) to reduce dimensionality before fitting a classifier or regression predictor, it is possible that the principal components do not capture the important information for the prediction problem. In this case, it may be preferable to obtain variable importance directly from the algorithm and then re-fit using only the most important predictors.

### *3.1 Permutation Importance*

Random Forests use an unusual but intuitive measure of variable importance. To measure the importance of variable $k$, the following procedure is performed for each tree. First, the out-of-bag observations are passed down the tree and the predicted values are computed. Next, the values of variable $k$ are randomly permuted in the out-of-bag data, keeping all the other predictor variables fixed. These modified out-of-bag data are passed down the tree and the predicted values are computed. This process gives two sets of out-of-bag predictions for each observation: one set obtained from real data, the other set from variable-$k$-permuted data. For classification, the difference between the error rate of the predictions obtained from permuted data and those obtained using permuted data gives a measure of variable importance

for the observation. The same procedure is used for regression, but using MSE instead of error rates. For classification, class-wise variable importance is computed by averaging over observations from the same class. Overall variable importance is computed by averaging over all the observations.

**Algorithm 4** Permutation Variable Importance
To find the importance of variable $k$, for $k = 1$ to $p$:

1. (Find $\hat{y}_{i,j}$) For $i = 1$ to $N$:
   a. Let $\mathscr{J}_i = \{j : (x_i, y_i) \notin \mathscr{D}_j\}$ and let $J_i$ be the cardinality of $\mathscr{J}_i$ (Algorithm 2).
   b. Let $\hat{y}_{i,j} = \hat{h}_j(x_i)$ for all $j \in \mathscr{J}_i$.
2. (Find $\hat{y}^{\star}_{i,j}$) For $j = 1$ to $J$:
   a. Let $\mathscr{D}_j$ be the $j$th bootstrap sample (Algorithm 2).
   b. Let $\mathscr{F}_j = \{i : (x_i, y_i) \notin \mathscr{D}_j\}$.
   c. Randomly permute the value of variable $k$ for the data points $\{x_i : i \in \mathscr{F}_j\}$ to give $\mathscr{P}_j = \{x_i{}^{\star} : i \in \mathscr{F}_j\}$.
   d. Let $\hat{y}^{\star}_{i,j} = \hat{h}_j(x_i{}^{\star})$ for all $i \in \mathscr{F}_j$.
3. For $i = 1$ to $N$:
   - For classification: $\text{Imp}_i = \frac{1}{J_i}\sum_{j\in\mathscr{J}_i} I(y_i \neq \hat{y}^{\star}_{i,j}) - \frac{1}{J_i}\sum_{j\in\mathscr{J}_i} I(y_i \neq \hat{y}_{i,j})$.
   - For regression: $\text{Imp}_i = \frac{1}{J_i}\sum_{j\in\mathscr{J}_i}(y_i - \hat{y}^{\star}_{i,j})^2 - \frac{1}{J_i}\sum_{j\in\mathscr{J}_i}(y_i - \hat{y}_{i,j})^2$.

Algorithm 4 gives the importance of a particular variable, denoted by $k$ in the algorithm description, on the predictions for a particular observation, denoted by $i$. The values can be used as measures of *local* variable importance, or they can be averaged over all observations to give measures of overall importance of the variable. The largest values are generally plotted (Fig. 5).

Intuitively, the permutation-based importance of variable $k$ is an estimate of the how much the prediction error or MSE on a test set would increase if the value of variable $k$ were randomly permuted in the test set. In this sense, it is similar to the coefficient-based measures of importance used in methods such as linear regression or logistic regression - they measure how much the prediction would change if the value of the predictor increased by one unit, keeping everything else the same. Quite a different measure is obtained, for both Random Forests and classical methods, if variable $k$ is removed and the model is refit, because in this case predictors that are correlated with the one of interest can give a similar fit and make the variable appear unimportant. In contrast, if an important predictor variable is correlated with other predictor variables, Random Forests sometimes splits on one and sometimes on another, due to the random choice of predictors at each node. Therefore, Random Forests permutation importance tends to identify all of the correlated predictors as important if any one of them is important.

One attractive feature of all tree-based methods is their ability to capture complex interactions between predictors. If Random Forests captures such an interaction, the

variables involved are likely to show up as "important" because randomly permuting one of them destroys the predictive power of the interaction.

To illustrate the behavior of Random Forest permutation importance, a regression forest was fit to the prostate data (Example 1). A permutation importance plot is given in Fig. 5, showing that the three most important variables are lcavol, lweight and svi. Interestingly, these are the same three variables chosen by lasso (see Fig 3.10 of [10]).

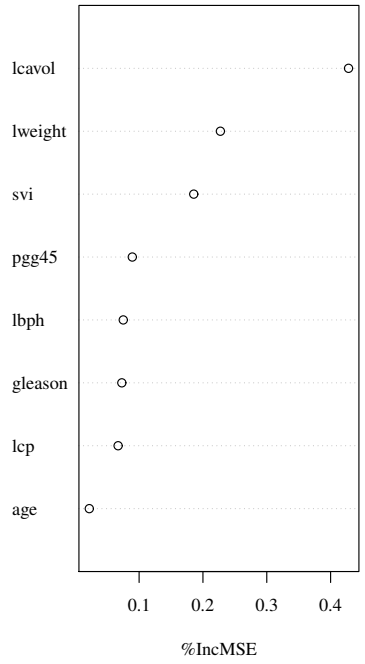


**Fig. 5** Permutation variable importance, prostate data (Example 1).

**Example 3** Normal Mixture

To illustrate the behavior of Random Forest variable importance when classes are weighted differently, consider a bivariate normal mixture of three classes

$$\pi_1 N(\mu_1, I) + \pi_2 N(\mu_2, I) + \pi_3 N(\mu_3, I)$$

where $N(\mu, I)$ denotes the bivariate normal density with mean $\mu$ and covariance matrix the identity. Generating $N = 300$ observations from such a mixture with $\mu_1 = (0,0)^T, \mu_2 = (0,3)^T, \mu_3 = (3,3)^T$ and $\pi_1 = 0.4, \pi_2 = 0.4, \pi_3 = 0.2$ gave the data shown in Figure 6. Fitting Random Forests using $J = 500$ trees and $m = 1$ for two different weighting schemes gave the results in Table 2. Equal weighting gives the lowest overall error rate. Increasing the weight on class 3, the smallest class, reduces the class 3 error rate from 8.9% to 5.0% and increases the error rates of the other two classes, giving an overall increase in error rate from 10.4% to 11.8%. More interestingly, equal weighting ranks variable 1 as more important than variable 2, while increasing the weight on class 3 reverses the ranking.

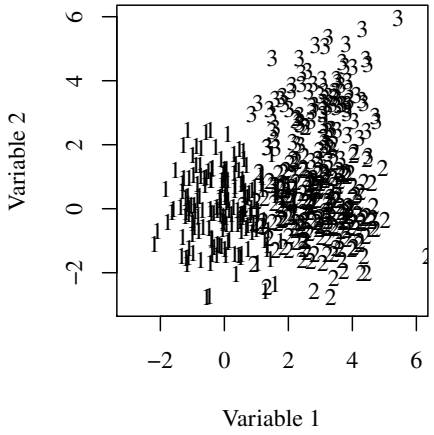


**Fig. 6** Bivariate normal mixture of three classes (Example 3).

**Table 2** Impact of class weights on error rates and permutation importance.

| | Class 1 | Class 2 | Class 3 | Class 1 | Class 2 | Class 3 |
|---|---|---|---|---|---|---|
| Weights | 1/3 | 1/3 | 1/3 | 1/7 | 1/7 | 5/7 |
| Classwise error rate (percent) | 8.2 | 13.0 | 8.9 | 8.7 | 17.7 | 5.0 |
| Overall error rate (percent) | | 10.4 | | | 11.8 | |
| Permutation importance (variable 1) | | 32.7 | | | 73.8 | |
| Permutation importance (variable 2) | | 22.3 | | | 49.1 | |

# 4 Proximities

Random Forests proximities are used for missing value imputation and visualization.

## *4.1 Definition*

The proximity between two observations is the proportion of the time that they end up in the same terminal node, where the proportion is taken over the trees in the forest. If two observations are always in the same terminal node, their proximity will be 1. If they are never in the same terminal node, their proximity will be 0. The proximity between two observations is a measure of how close together they are in predictor space, but it automatically gives more weight to predictors that are useful for predicting the response. Observations that are very far apart in Euclidean space may have quite a large proximity if they only differ on weak or irrelevant predictors, while observations that are relatively close together in Euclidean space may have relatively small proximities if they differ on predictors that are crucial for predicting the response.

## *4.2 Missing Value Imputation*

Random Forests imputes missing values using the proximities described above. The procedure is iterative: an initial forest is built using median imputation, proximities are calculated, and new imputations are obtained by a proximity-weighted average for a continuous predictor or a proximity-weighted vote for a categorical predictor. A new forest is built, giving new proximities and imputations. Usually 5 or 6 iterations are sufficient to give stable imputations. Although no formal analysis has been done, the fact that the method uses proximity-based nearest neighbors suggests that it will be valid if values of the predictors are missing at random.

## *4.3 Visualization*

From a statistical perspective, one of the difficult aspects of high-dimensional data analysis is that it is not obvious how to get a good “feel” for the data. Are there interesting patterns or structures, such as sub-groups within the known classes? Are there outliers? In a multi-class situation, are some of the groups separated while others overlap? Random Forests provide a way to look at the data to give some insight into these questions. This is done by computing proximities, deriving a distance matrix, and performing classical multidimensional scaling (MDS) to obtain two-

or three-dimensional plots. Each point on such a plot represents one of the observations and the distances between the points reproduce, as closely as possible, the proximity-based distances. Such a plot can be used to pick out subgroups of cases that almost always stay together in the trees, or outliers that are almost always alone in a terminal node.

**Example 4** Microarray Data

To illustrate the potential usefulness of visualization using the proximity matrix, we consider the prostate cancer microarray data [22]. These data have 6033 gene expression values for 102 arrays (50 normal samples and 52 tumor samples). We used the normalization described by Dettling [8]. Figure 7 (left) shows the first two dimensions of the MDS plot based on the Random Forest proximity matrix.

A natural question at this point is whether it would be just as good to use MDS on a conventional distance, such as Euclidean distance or one of the other distances commonly used in cluster analysis. This can certainly be done, but one of the difficulties is that a conventional distance can be dominated by noisy and uninformative predictors that may drown out the effects of the important predictors. This behavior can be seen in Figure 7, which presents the MDS plot derived from the proximity matrix and the MDS plot derived from Euclidean distance for the microarray data in Example 4. The proximity plot reveals much more structure than the plot based on Euclidean distances, including an outlier that could be of interest to the investigators.

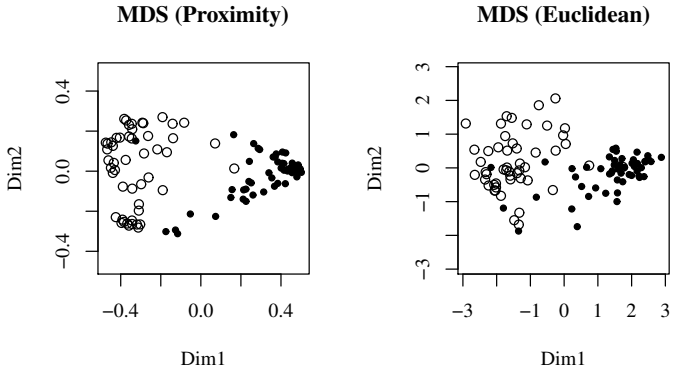


**Fig. 7** MDS plot from the Random Forests proximities (left) and from Euclidean distance (right) for Example 4. Solid circles represent cancer cases, open circles represent controls.

## 5 Software

Commercial software for Random Forests is available from www.salford-systems.com. The R package is `randomForest` [15] and this, along with R [19], is available from the CRAN website `www.cran.r-project.org`. Open source FORTRAN software for Random Forests is available from `www.math.usu.edu/~adele/forests`.

## 6 Summary

Random Forests are a multipurpose tool, applicable to both regression and classification problems, including multiclass classification. They give an internal estimate of generalization error so crossvalidation is unnecessary. They can be tuned, but often work quite well with default tuning parameters. Variable importance measures are available, which can be used for variable selection. Random Forests produce proximities, which can be used to impute missing values. Proximities can also provide a wealth of information by enabling novel visualizations of the data. Random Forests have been successfully used for a wide variety of applications and enjoy considerable popularity in several disciplines.

## 7 Bibliographical and Historical Remarks

The Random Forest algorithm was the last major work of Leo Breiman [6].

Theoretical developments have been difficult to achieve. In the original paper, Breiman [6] suggested that Random Forests work by reducing correlation, while keeping the variance relatively small. Lin and Jeon [16] show that RF behaves like a nearest neighbor classifier with an adaptive metric. More recently, Biau et al. address consistency [3].

Several extensions have been published, for example [9] developed a variable selection procedure, [18] introduced quantile regression forests, and [12], [13] considered forests for survival analysis. More recently, [21] extends Random Forests for multivariate responses. Amaratunga et al. [1] suggest an extension to very high dimensional data.

Applications of Random Forests are numerous and only a few can be mentioned here. Statnikov [24] compares random forests and support vector machines for microarray-based cancer classification. Schroff et al. [20] used Random Forests for image segmentation. Chen et al. [7] use Random Forests to identify genetic interactions, while Goldstein et al. [11] and [25] apply Random Forests to SNP-based genome-wide association data.

## References


1. Amaratunga, D., Cabrera, J., Lee, Y.-S.: Enriched random forests. Bioinformatics **24** (18) pp. 2010–2014 (2008).
2. Amit, Y., Geman, D.: Shape quantization and recognition with randomized trees. Neural Computation **9**(7) pp. 1545-1588 (1997).
3. Biau, G., Devroye, L., Lugosi, G.: Consistency of Random Forests and Other Averaging Classifiers. Journal of Machine Learning Research **9** pp. 2039–2057, (2008).
4. Breiman, L., Friedman, J., Olshen, R., Stone, C.: Classification and Regression Trees. Wadsworth, New York (1984).
5. Breiman, L.: Bagging Predictors. Machine Learning **24** (2) pp. 123–140 (2001).
6. Breiman, L.: Random Forests. Machine Learning **45** (1) pp. 5–32 (2001).
7. Chen, X., Liu, C.-T., Zhang, M., Zhang, H.: A forest-based approach to identifying gene and genegene interactions. Proc Natl Acad Sci U S A. **104** (49) pp. 19199-19203 (2007).
8. Dettling, M.: BagBoosting for Tumor Classification with Gene Expression Data. Bioinformatics **20** (18) pp. 3583–3593 (2004).
9. Diaz-Uriarte, R., Alvarez de Andres, S.: Gene Selection and Classification of Microarray Data Using Random Forest. BMC Bioinformatics **7** (1) 3. (2006).
10. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition. Springer Series in Statistics, Springer, New York (2009).
11. Goldstein, B., Hubbard, A., Cutler, A. Barcellos, L.: An application of Random Forests to a genome-wide association dataset: Methodological considerations & new findings. BMC Genetics **11** (1) 49 (2010).
12. Hothorn, T., Bühlmann, P., Dudoit, S., Molinaro, A., Van Der Laan, M.: Survival Ensembles. Biostatistics **7** (3) pp. 355–373 (2006).
13. Ishwaran, H., Kogalur, U.B., Blackstone, E.H., Lauer, M.S.: Random survival forests. Annals of Applied Statistics **2** (3) pp. 841–860, (2008).
14. Izenman, A.: Modern Multivariate Statistical Techniques. Springer Texts in Statistics, Springer, New York (2008).
15. Liaw, A., Wiener, M.: Classification and Regression by randomForest. R News **2** (3) pp. 18–22, (2002).
16. Lin, Y., Jeon, Y.: Random Forests and Adaptive Nearest Neighbors. Journal of the American Statistical Association **101** (474) pp. 578–590, (2006).
17. Mease, D., Wyner, A.: Evidence Contrary to the Statistical View of Boosting. Journal of Machine Learning Research **9** pp. 131–156 (2008).
18. Meinshausen, N.: Quantile Regression Forests. Journal of Machine Learning Research **7** pp. 983–999, (2006).
19. R Development Core Team: R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, (2011).
20. Schroff, F., Criminisi, A., Zisserman, A.: Object Class Segmentation using Random Forests. Proceedings of the British Machine Vision Conference 2008, British Machine Vision Association, **1** (2008).
21. Segal, M., Xiao, Y.: Multivariate Random Forests. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **1** (1) pp. 80-87, (2011).
22. Singh D., Febbo P.G., Ross K., Jackson D.G., Manola J., Ladd C., Tamayo P., Renshaw A.A., D'Amico A.V., Richie J.P., Lander E.S., Loda M., Kantoff P.W., Golub T.R., Sellers W.R.: Gene expression correlates of clinical prostate cancer behavior. Cancer Cell **1** (2) pp. 203-9 (2002).
23. Stamey, T., Kabalin, J., McNeal J., Johnstone I., Freiha F., Redwine E., Yang N.: Prostate specific antigen in the diagnosis and treatment of adenocarcinoma of the prostate. II. Radical prostatectomy treated patients. Journal of Urology **16** pp. 1076-1083, (1989).
24. Statnikov, A., Wang, L., Aliferis, C.: A comprehensive comparison of random forests and support vector machines for microarray-based cancer classification. BMC Bioinformatics **9** (1) 319 (2008).

25. Wang, M., Chen, X., Zhang, H.: Maximal conditional chi-square importance in random forests. **26** (6): pp. 831-837 (2010).
26. Zhang, H., Singer, B.H.: Recursive Partitioning and Applications, Second Edition. Springer Series in Statistics, Springer, New York (2010).