Learning with Trees

Introduction

Decision trees have been around for a number of years. Their recent revival is due to the discovery that ensembles of slightly different trees tend to produce much higher accuracy on previously unseen data, a phenomenon known as generalization. Ensembles of trees will be discussed but let us focus first on individual trees.

Supervised Learning

Classification and Regression are supervised learning task. Learning set is of the form:

$$\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

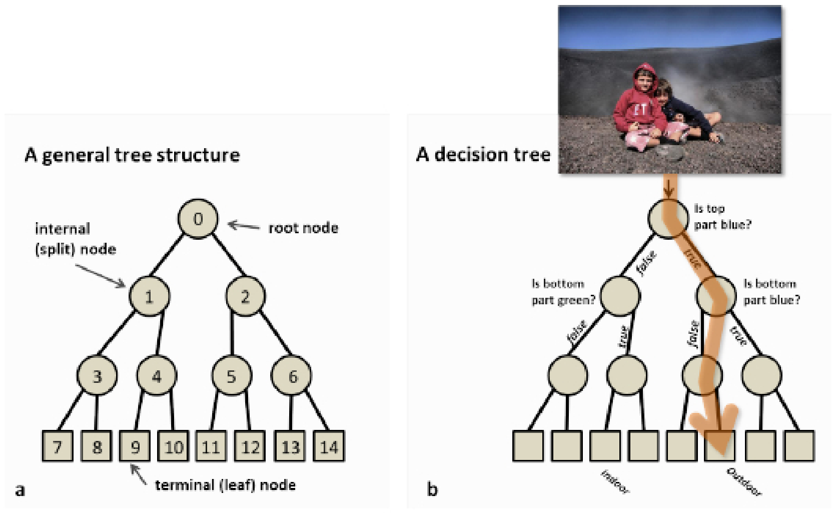
Classification (Two classes, binary)

$$\mathbf{x} \in \mathbb{R}^d, \qquad y \in \{-1, +1\}$$

Regression

$$\mathbf{x} \in \mathbb{R}^d, \qquad y \in \mathbb{R}$$

Decision tree basics

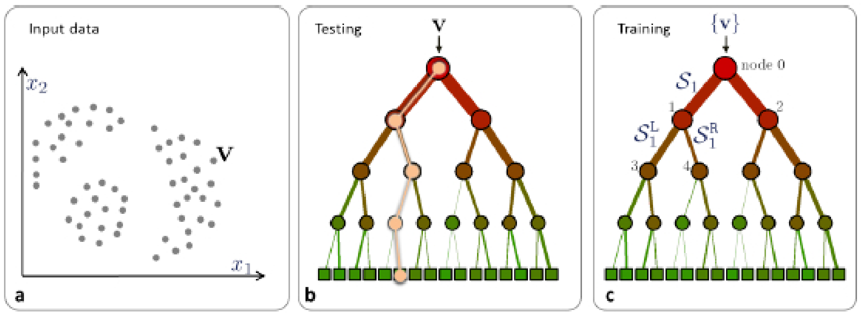


Decision tree

We can see Fig.1.

1. A tree is a set of nodes and edges organized in a hierarchical fashion. In contrast to a graph, in a tree there are no loops. Internal nodes are denoted with circles and terminal nodes with squares.
2. A decision tree is a tree where each split node stores a boolean test function to be applied to the incoming data. Each leaf stores the final answer (predictor)

Training and testing decision trees

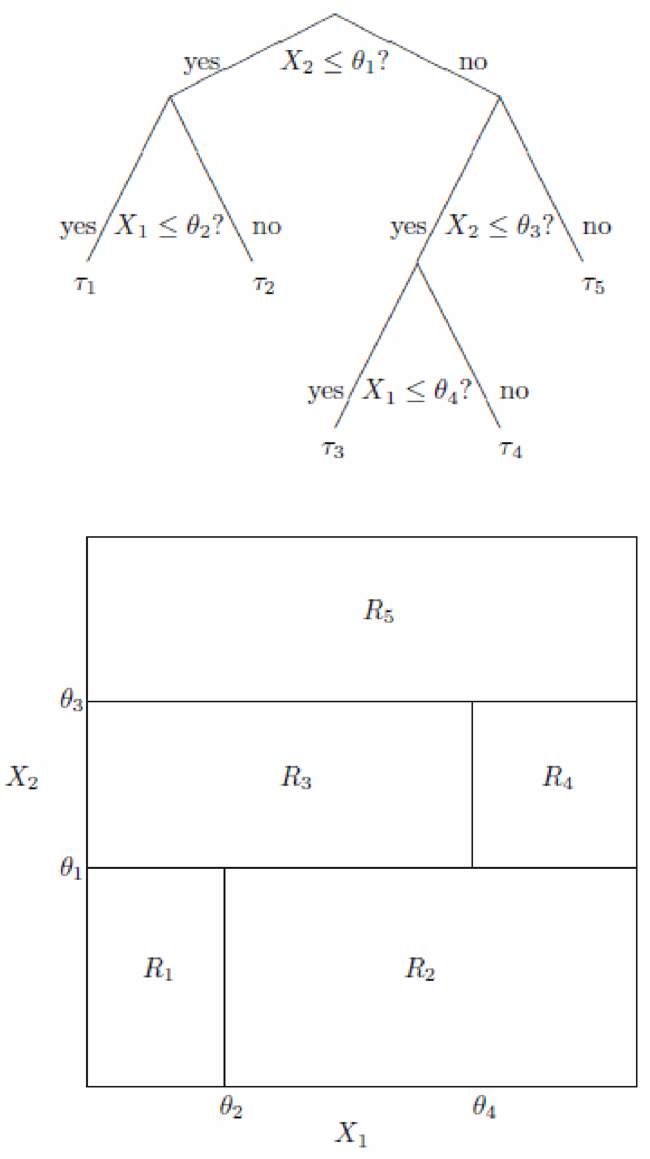


Basic notation

We can see Fig.2.

1. Input data is represented as a collection of points in the d-dimensional space which are labeled by their feature responses.
2. A decision tree is a hierarchical structure of connected nodes.

3. Training a decision tree involves sending all training data $\{v\}$ into the tree and optimizing the parameters of the split nodes so as to optimize a chosen cost function.
4. During testing, a split (internal) node applies a test to the input data v and sends it to the appropriate child. The process is repeated until a leaf (terminal) node is reached (beige path).



Tree-Growing Procedure

We can see Fig.3.

In order to grow a classification tree, we need to answer four basic questions:

1. How do we choose the Boolean conditions for splitting at each node?
2. Which criterion should we use to split a parent node into its two child nodes?
3. How do we decide when a node become a terminal node (i.e., stop splitting)?
4. How do we assign a class to a terminal node?

Choosing the best split for a Variable

Splitting Strategies

At each node, the tree-growing algorithm has to decide on which variable it is "best" to split. We need to consider every possible split over all variables present at that node, then enumerate all possible splits, evaluate each one, and decide which is best in some sense.

For a description of splitting rules, we need to make a distinction between ordinal (or continuous) and nominal (or categorical) variables. For a continuous or ordinal variable, the number of possible splits at a given node is one fewer than the number of its distinctly observed values. Suppose that a particular categorical variable is defined by m distinct categories, there are $2^{m-1} - 1$ distinct splits.

We first need to choose the best split for a given variable. Accordingly, we have to measure of goodness of a split. Let $C_1, \ldots, C_K$ be the $K \geq 2$ classes. For node τ, we denote by

$$P(X \in C_k | \tau)$$

the conditional probability that an observation x is in C_k given that it falls into the node τ.

Node information functions

We can see Fig.4

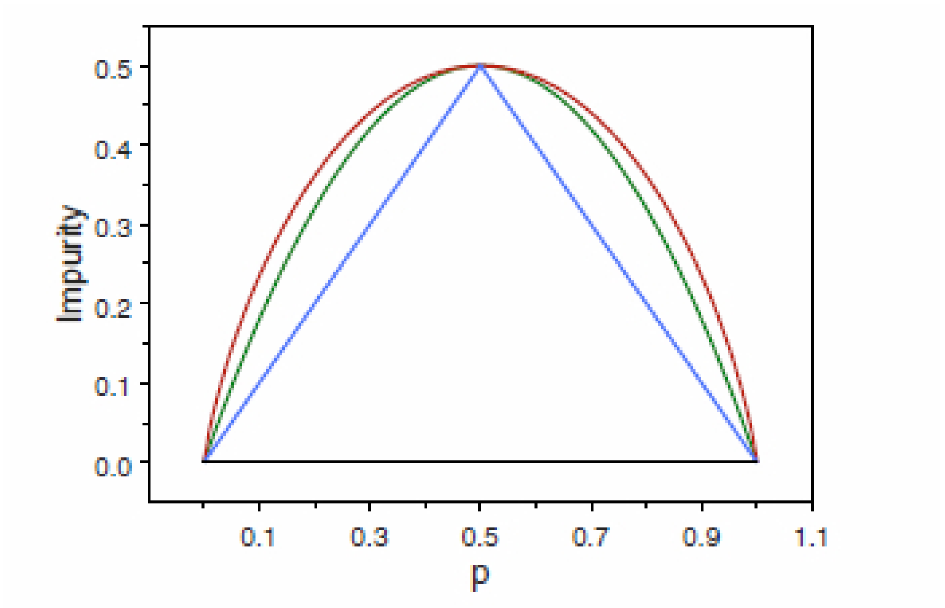


Figure 1: Node impurity functions for the two-class case. The entropy function (rescaled) is the red curve, the Gini index is the green curve, and the resubstitution estimate of the misclassification rate is the blue curve.

To choose the best split over all variables, we first need to choose the best split for a given variable. Accordingly, we define a measure of goodness of a split. For node τ, the node information function $i(\tau)$ is definded by

$$i(\tau) = \phi\Big(p(1|\tau), ..., p(K|\tau)\Big)$$

where $p(k|\tau)$ is an estimate of $P(X \in C_k|\tau)$.

We require ϕ to be a symmetric function, defined on the set of all K-tuples of probabilities $(p_1, \dots, p_k)$ with unit sum, minimized at the points $(1, 0, \cdots, 0)$, $(0, 1, \cdots, 0)$, $(0, 0, \cdots, 1)$, and maximized at the point $(\frac{1}{K}, \frac{1}{K}, \cdots, \frac{1}{K})$. One such function ϕ is the entropy function (also called cross-entropy),

$$i(\tau) = -\sum_{k=1}^{K} p(k|\tau) \log p(k|\tau)$$

An alternative to the the entropy is the Gini index, given by

$$i(\tau) = \sum_{k \neq k'}^{K} p(k|\tau)p(k'|\tau) = 1 - \sum_{k=1}^{K} p(k|\tau)^2$$

the misclassification rate

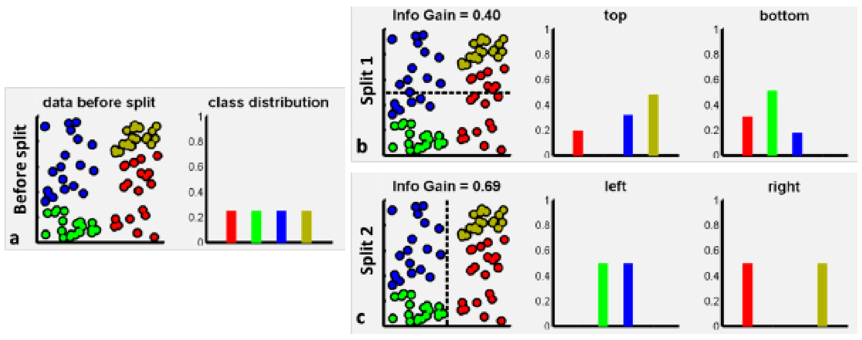
$$i(\tau) = \sum_{k=1}^{K} p(k|\tau)(1 - p(k|\tau))$$

and the deviance (it is equivalent to entropy)

$$i(\tau) = -2 \sum_{k=1}^{K} n_{k|\tau} \log(p(k|\tau))$$

where $n_{k|\tau}$ is the number of observations from class C_k at τ node.

Information gain



We can see Fig.5.

Suppose, at node τ , we apply split s so that a proportion p_L of the observations drops down to the left child-node τ_L and the remaining proportion p_R drops down to the right child-node τ_R

The goodness of split s at node τ is given by the reduction in impurity gained by splitting the parent node τ into its child nodes, τ_L and τ_R,

$$I(s, \tau) = i(\tau) - p_L i(\tau_L) - p_R i(\tau_R)$$

The best split for the single variable X_j is the one that has the largest value of $I(s, \tau)$ over all $s \in \mathcal{S}_j$, the set of possible distinct splits for X_j .

Example

	+1	-1	Total
$X_j \leq s$	n_{11}	n_{12}	n_{1+}
$X_j > s$	n_{21}	n_{22}	n_{2+}
Total	n_{+1}	n_{+2}	n_{++}

Consider first, the parent node τ, estimate $P(\frac{+1}{\tau})$ by n_{+1}/n_{++} and $P(\frac{-1}{\tau})$ by n_{+2}/n_{++}, and then the estimated impurity function is:

$$i(\tau) = -\Big(\frac{n_{+1}}{n_{++}}\Big) \log\Big(\frac{n_{+1}}{n_{++}}\Big) - \Big(\frac{n_{+2}}{n_{++}}\Big) \log\Big(\frac{n_{+2}}{n_{++}}\Big)$$

Note that $i(\tau)$ is completely independent of the type of proposed split.

Now, for the child nodes, τ_L y τ_R. We estimated $p_L = n_{1+}/n_{++}$ and $p_R = n_{2+}/n_{++}$

For $X_j \leq s$, we estimate $P(\frac{+1}{\tau_L}) = n_{11}/n_{1+}$ and $P(\frac{-1}{\tau_L}) = n_{12}/n_{1+}$ For the condition $X_j > s$, we estimate $P(\frac{+1}{\tau_R}) = n_{21}/n_{2+}$ and $P(\frac{-1}{\tau_R}) = n_{22}/n_{2+}$. We then compute:

$$i(\tau_L) = -\Big(\frac{n_{11}}{n_{1+}}\Big) \log\Big(\frac{n_{11}}{n_{1+}}\Big) - \Big(\frac{n_{12}}{n_{1+}}\Big) \log\Big(\frac{n_{12}}{n_{1+}}\Big)$$

$$i(\tau_R) = -\Big(\frac{n_{21}}{n_{2+}}\Big) \log\Big(\frac{n_{21}}{n_{2+}}\Big) - \Big(\frac{n_{22}}{n_{2+}}\Big) \log\Big(\frac{n_{22}}{n_{2+}}\Big)$$

and, then we can compute $I(s, \tau)$.

Recursive Partitioning

- In order to grow a tree, we start with the root node, which consists of the learning set $\mathcal{L}$. Using the "goodness-of-split" criterion for a single variable, the tree algorithm finds the best split at the root node for each of the variables, X_1 to X_r.
- The best split s at the root node is then defined as the one that has the largest value of over all r single-variable best splits at that node.

- We next split each of the child nodes of the root node in the same way. We repeat the above computations for the left child node, except that we consider only those observations that fall into each child node.
- When those splits are completed, we continue to split each of the subsequent nodes. This sequential splitting process of building a tree layer-by-layer is called recursive partitioning.

Stop spliting

- For example, we can declare a node to be terminal if it fails to be larger than a certain critical size (n_{min} is some previously declared minimum size of a node).
- Another early action was to stop a node from splitting if the largest goodness-of-split value at that node is smaller than a certain predetermined limit.
- These stopping rules, however, do not turn out to be such good ideas. A better approach (Breiman et al. 2008) is to let the tree grow to saturation and then ”prune” it back.

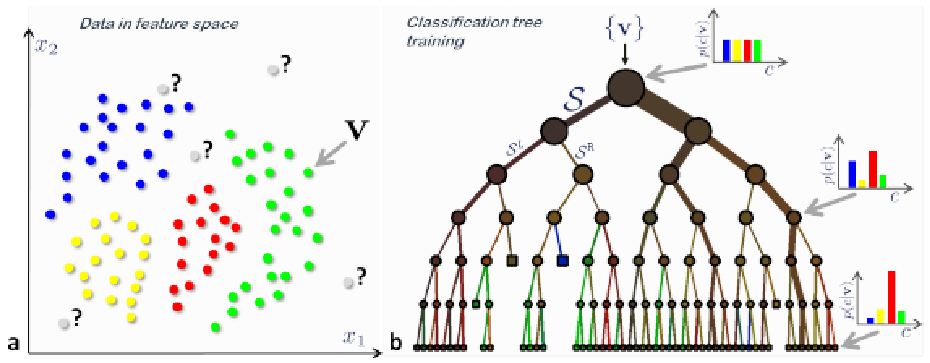
Plurality rule

We can see Fig.6.

How do we associate a class with a terminal node?

Suppose at terminal node τ there are $n(\tau)$ observations, of which $n_k(\tau)$ are from class C_k, $k = 1, ..., K$. Then, the class which corresponds to the largest of the $\{n_1(\tau), ..., n_k(\tau)\}$ is assigned to τ.

This is called the plurality rule. This rule can be derived from the Bayes's rule classifier, where we assign the node τ to class C_i if $p(i|\tau) = \max_k p(k|\tau)$.



Choosing the best split in Regression Trees

We now discuss the process of building a regression tree. Roughly speaking, there are two steps.

1. We divide the predictor space-that is, the set of possible values for $X_1, X_2, \ldots, X_p$ into J distinct and non-overlapping regions, $R_1, R_2, \ldots, R_J$.
2. For every observation that falls into the region R_j, we make the same prediction, which is simply the mean of the response values for the training observations in R_j.

For instance, suppose that in Step 1 we obtain two regions, R_1 and R_2, and that the response mean of the training observations in the first region is 10, while the response mean of the training observations in the second region is 20. Then for a given observation $X = x$, if $x \in R_1$ we will predict a value of 10, and if $x \in R_2$ we will predict a value of 20.

We now elaborate on Step 1 above. How do we construct the regions $R_1, ..., R_J$? In theory, the regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model. The goal is to find boxes $R_1, ..., R_J$? that minimize the RSS (Residual Sum of Squares), given by

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

where $\hat{y}_{R_j}$ is the mean response for the training observations within the j-th box. Unfortunately, it is computationally infeasible to consider every possible partition of the feature space into J boxes. For this reason, we take a top-down, greedy approach that is known as recursive binary splitting. The approach is top-down because it begins at the top of the tree (at which point all observations belong to a single region) and then successively splits the predictor space; each split is indicated via two new branches further down on the tree. It is greedy because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

In order to perform recursive binary splitting, we first select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j \leq s\}$ and $\{X|X_j > s\}$ leads to the greatest possible reduction in RSS. (The notation $\{X|X_j \leq s\}$ means the region of predictor space in which X_j takes on a value less than s). That is, we consider all predictors $X_1, ..., X_p$, and all possible values of the cutpoint s for each of the predictors, and then choose the predictor and cutpoint such that the resulting tree has the lowest RSS. In greater detail, for any j and s, we define the pair of half-planes

$$R_1(j, s) = \{X|X_j \leq s\}, \qquad R_2(j, s) = \{X|X_j > s\}$$

and we seek the value of j and s that minimize the equation

$$\sum_{i: x_i \in R_1} (y_i - \hat{y}_{R_1})^2 + \sum_{i: x_i \in R_2} (y_i - \hat{y}_{R_2})^2 \tag{1}$$

where $\hat{y}_{R_1}$ is the mean response for the training observations in $R_1(j, s)$, and $\hat{y}_{R_2}$ is the mean response for the training observations in $R_2(j, s)$. Finding the values of j and s that minimize (1) can be done quite quickly, especially when the number of features p is not too large.

Next, we repeat the process, looking for the best predictor and best cutpoint in order to split the data further so as to minimize the RSS within each of the resulting regions. However, this time, instead of splitting the entire predictor space, we split one of the two previously identified regions. We now have three regions. Again, we look to split one of these three regions further, so as to minimize the RSS. The process continues until a stopping criterion is reached; for instance, we may continue until no region contains more than five observations.

Once the regions $R_1, ..., R_J$ have been created, we predict the response for a given test observation using the mean of the training observations in the region to which that test observation belongs.

Estimating the misclassification rate

Let T be the tree classifier and let $\tilde{T} = \{\tau_1, \tau_2, ..., \tau_L\}$ denote the set of all terminal nodes of T. We can now estimate the true misclassification rate,

$$R(T) = \sum_{l=1}^{L} R(\tau_l)P(\tau_l), \tag{2}$$

for T, where $P(\tau)$ is the probability that an observation falls into node τ and $R(\tau)$ is the within-node misclassification rate of an observation in node τ.

If we estimate $R(\tau)$ by

$$r(\tau) = 1 - \max_k p(k|\tau)$$

and we estimate $P(\tau_l)$ by the proportion $p(\tau_l)$ of all observations that fall into node τ_l, then, the resubstitution estimate of $R(T)$ is

$$\hat{R}(T) = \sum_{l=1}^{L} r(\tau_l)p(\tau_l).$$

The resubstitution estimate $\hat{R}(T)$, however, leaves much to be desired as an estimate of $R(T)$.

First, bigger trees (i.e., more splitting) have smaller values of $\hat{R}(T)$; that is, $\hat{R}(T') \leq \hat{R}(T))$, where T' is formed by splitting a terminal node of T. For example, if a tree is allowed to grow until every terminal node contains only a single observation, then that node is classified by the class of that observation and $\hat{R}(T) = 0$.

Second, using only the resubstitution estimate tends to generate trees that are too big for the given data.

Third, the resubstitution estimate $\hat{R}(T)$ is a much-too-optimistic estimate of $R(T)$. More realistic estimates of $R(T)$ are given below.

Tree Pruning

Since decision trees have a very high tendency to over-fit the data, a smaller tree with fewer splits might lead to increase the generalization capability. Lower variance (estimation error) at the cost of a little bias (approximation error).

One possible alternative to the process described above is to build the tree only so long as the decrease in the node impurity measure, due to each split exceeds some (high) threshold. However, due to greedy nature of the splitting algorithm, it is too short-sighted since a seemingly worthless split early on in the tree might be followed by a very good split i.e., a split that leads to a large reduction in impurity later on.

Therefore, a better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a subtree.

Cost complexity pruning

A sequence of trees indexed by a nonnegative tuning parameter α is considered. For each value of α there corresponds a subtree $T \subset T_0$ such that the penalized misclassification rate

$$R_\alpha(T) = R(T) + \alpha|T|$$

is as small as possible. Here $|T|$ indicates the number of terminal nodes of the subtree T, Think of $\alpha|T|$ as a penalty term for tree size, so that $R_\alpha(T)$ penalizes $R(T)$ (2) for generating too large a tree. For each α, we then choose that subtree $T(\alpha)$ of T_0 that minimizes $R_\alpha(T)$.

The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data. When $\alpha = 0$, then the subtree T will simply equal T_0. As α increases, there is a price to pay for having a tree with many terminal nodes, and so the above equation will tend to be minimized for a smaller subtree.

Breiman et al. (1984) showed that for every α, there exists a unique smallest minimizing subtree.

Depending on the cost of each additional leaf (i.e. the α value) different sub-trees of T_0 minimise the error-complexity measure. Breiman and his colleagues proved that although α can run through a continuum of values there is a sequence of pruned trees such that each element is optimal for a range of α, and so there is only a finite number of interesting α values.

$$0 = \alpha_0 < \alpha_1 < \alpha_2 < \alpha_3 < \cdots < \alpha_M,$$

Furthermore, they developed an algorithm that generates a parametric family of pruned trees

$$T_0 \prec T_1 \prec T_2 \prec T_3 \prec \cdots \prec T_M,$$

such that each T_i in the sequence is characterised by adifferent value α_i. They proved that each tree T_i in this sequence is optimal from the error-complexity perspective within the interval $[\alpha_i, \alpha_{i+1})$.

So far, we have constructed a finite sequence of decreasing-size subtrees $T_1, T_2, T_3, \ldots, T_M$ by pruning more and more nodes from T_0. When do we stop pruning? Which subtree of the sequence do we choose as the "best" pruned subtree? Choice of the best subtree depends upon having a good estimate of the misclassification rate $R(T_k)$ corresponding to the subtree T_k. Breiman et al. (1984) offered two estimation methods: use an independent test sample or use cross-validation. When the data set is very large, use of an independent test set is straightforward and computationally efficient, and is, generally, the preferred estimation method. For smaller data sets, crossvalidation is preferred.

Advantages and disadvantages of trees

1. Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
2. Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches.
3. Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
4. Trees can easily handle qualitative predictors without the need to create dummy variables.
5. Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches.

However, by aggregating many decision trees, using methods like bagging, random forests, and boosting, the predictive performance of trees can be substantially improved. We introduce these concepts next.

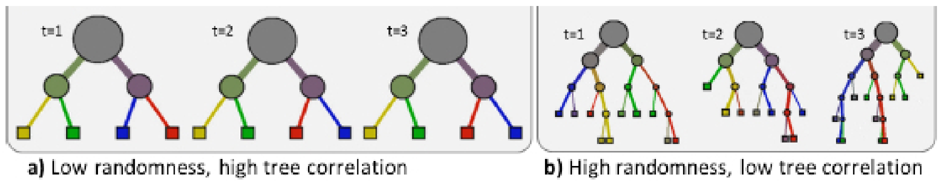
The randomness model

A key aspect of decision forests is the fact that its component trees are all randomly different from one another (Fig. 7). This leads to de-correlation between the individual tree predictions and, in turn, to improved generalization. Forest randomness also helps achieve high robustness with respect to noisy data.

Randomness is injected into the trees during the training phase. Two of the most popular ways of doing so are:

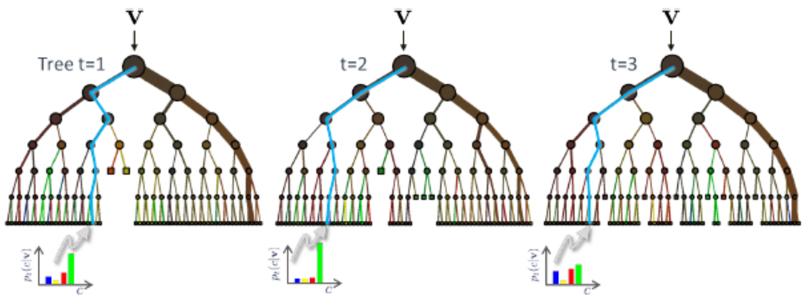
- random training data set sampling (bagging approach) and
- randomized node optimization.

These two techniques are not mutually exclusive and could be used together.


t=1
t=2
t=3
a) Low randomness, high tree correlation
t=1
t=2
t=3
b) High randomness, low tree correlation

The ensemble model

In a forest with T trees all trees are trained independently (and possibly in parallel) (Fig. 8). During testing, each test point v is simultaneously pushed through all trees (starting at the root) until it reaches the corresponding leaves. Tree testing can also often be done in parallel, thus achieving high computational efficiency on modern parallel CPU or GPU hardware. Combining all tree predictions into a single forest prediction may be done by a simple averaging operation.



Random Forest

Algorithm

1. Given training data set $\mathcal{L} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$. Fix $m \leq p$ ($m = \sqrt{p}$) and the number of trees B.
2. For $b = 1, ..., B$, do the following.
   (a) Create a bootstrap version of the training data $\mathcal{L}_b^*$, by randomly sampling the n rows with replacement n times.The sample can be represented by the bootstrap frequency vector w_b^*.
   (b) Grow a maximal-depth tree T_b using the data in $\mathcal{L}_b^*$, sampling m of the p features at random prior to making each split.
   (c) Save the tree, as well as the bootstrap sampling frequencies for each of the training observations.
3. Output the ensembles of the trees $\{T_b\}_1^B$

To make a prediction at a new point $\mathbf{x}$:

Regression: Compute the random-forest fit at any prediction point as the average

$$\hat{f}_{\text{rf}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} T_b(\mathbf{x})$$

Classification: Let $\hat{C}_b(\mathbf{x})$ be the class prediction of the b-th random forest tree. Then $\hat{C}_{rf}(\mathbf{x}) =$ majority vote $\{\hat{C}_b(\mathbf{x})\}_1^B$

Out-of-Bag error estimates

Random forests deliver cross-validated error estimates at virtually no extra cost. The idea is similar to the bootstrap error estimates.

In making the prediction for observation pair $(\mathbf{x}_i, y_i)$, we average all the random forest trees $T_b(\mathbf{x}_i)$ for which that pair is not in the corresponding bootstrap sample:

$$\hat{f}^{(i)}_{\text{rf}}(\mathbf{x}) = \frac{1}{B_i} \sum_{b: w^*_{bi}=0}^{B_i} T_b(\mathbf{x})$$

where B_i is the number of times observation i was not in the bootstrap sample.

We then compute the OOB error estimate

$$\text{err}_{OOB} = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}^{(i)}_{rf}(\mathbf{x}_i))$$

where L is the loss function of interest, such as misclassification or squarederror loss. If B is sufficiently large, we can see that the OOB error estimate is equivalent to leave-one-out cross-validation error.

Assessing variable importance

Computations are carried out one tree at a time.

As before, let T_b be the tree classifier constructed from the bootstrap sample $\mathcal{L}^*_b$. First, drop the *OOB* observations corresponding to $\mathcal{L}^*_b$ down the tree T_b, record the resulting classifications, and compute the *OOB* error rate, $PE_b(OOB)$. Next, randomly permute the *OOB* values on the j-th variable X_j while leaving the data on all other variables unchanged. If X_j is important, permuting its observed values will reduce our ability to classify successfully each of the *OOB* observations. Then, we drop the altered *OOB* observations down the tree T_b, record the resulting classifications, and compute the *OOB* error rate, $PE_b(OOB_j)$, which should be larger than the error rate of the unaltered data. A raw T_b-score for X_j can be computed by the difference between those two *OOB* error rates,

$$\text{raw}_b(j) = PE_b(OOB_j) - PE_b(OOB) \qquad b = 1, ..., B$$

Finally, average the raw scores over all the B trees in the forest,

$$\text{imp}(X_j) = \frac{1}{B} \sum_{b=1}^{B} \text{raw}_b(j)$$

to obtain an overall measure of the importance of X_j. Call this measure the raw permutation accuracy importance score for the j-th variable. We calculate the score for each variable, and at the end of the procedure we rank variables according to the score.

Bibliography

A. Criminisi, J. Shotton and E. Konukoglu. Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. Microsoft Research technical report TR-2011-114.

The Elements of Statistical Learning: Data Mining, Inference and Prediction. Second Edition. Springer. 2009.