Tree-based and Ensemble methods

Antonio D'Ambrosio

Outline

Introduction I

Tree-based methods involve stratifying or segmenting the predictor space into a number of simple regions. Since the set of splitting rules used to segment the predictor space can be summarized in a tree, these types of approaches are known as tree-based (or decision tree) methods.

Tree-based methods are supervised and non-parametric tools dealing with each kind of variables. When the response variable is numerical, we deal with *regression trees*. When the response variable is categorical we deal with *classification trees*.

Introduction II

Tree-based methods are mainly used for two purposes: exploratory and / or predictive. Their outputs are very easy to interpret, showing interaction among the predictors and automatically selecting the predictors.

The principle is simple: for each predictor generate a binary question involving all their categories in this way:

$$\begin{array}{ll} \text{is } X_j < 12.5? & \text{if } X_j \text{ is numerical;} \\ \text{is } X_j < \text{medium?} & \text{if } X_j \text{ is ordinal;} \\ \text{is } X_j \in \{\text{north, east}\}\,? & \text{if } X_j \text{ is categorical.} \end{array}$$

Then evaluate for each question the *decrease in impurity*, and *split* the sample into two sub-samples characterized by the splitting rule.

Introduction III

Tree-based methods are known as unstable tools: often, for decisional purposes, they are not so accurate.

For this reason, resampling methods are intensively used in order to improve their accuracy. In the framework of tree-based methods, these methods are called *ensemble methdos*. We will talk about

- Regression trees;
- Classification trees;
- Speeding up tree-growing;
- Ensemble methods.

Tree-growing I

The data are partitioned by choosing at each step a variable and a cut point along it according to a goodness of split measure which allows to select that variable and cut point that generates the most homogeneous subgroups respect to the response variable.
The procedure results in a nice and powerful graphical representation known as decision tree which express the sequential grouping process. Because of the evident analogy with the graph theory, a subset of observations is called *node* and nodes that are not split are called *terminal nodes* (or leaves).

Tree-growing II

Tree based methods involve the following steps:

- the definition of a splitting criterion;
- the definition of a stopping rule;
- the definition of the response classes/values to the terminal nodes;
- tree pruning, aimed at simplifying the tree structure, and tree selection, aimed at selecting the final decision tree for decisional purposes

Splitting criteria I

Let (Y, X) be a multivariate random variable where X is a set of K categorical or numerical predictors $(X_1, \ldots, X_k, \ldots, X_K)$ and Y is the response variable. The first problem in tree building is how to determine the binary splits of the data into smaller and smaller subgroups. Since the partitioning is just two branches, splitting variables need to be created from the original explanatory variables.

Splitting criteria II

Accordingly, data partitioning is based on a set of Q binary questions of the form:

$$\text{is } X_k \in A?,$$

so that, if X_k is categorical, A includes subsets of levels, while if X_j is numeric, Q includes all questions of the form:

$$\text{is } X_k \leq c?,$$

for all c ranging over the domain of X_k.

Splitting criteria III

For example, if $K = 3$, X_1, X_2 are numerical and $X_3 \in \{a_1, a_2, a_3\}$, Q includes all questions of the form:

$$\begin{aligned} X_1 &\leq 3.5? \\ X_2 &\leq 5? \\ X_3 &\in \{a_1, a_3\}\,? \end{aligned}$$

The set of possible splitting variables is finite and the number of splitting variables that can be created from a given explanatory variable depends on the type of variable, i.e., according to its measurement.

Splitting criteria IV

Explanatory variable	**Categories**	**# of splitting variables**
Numeric	N	$N-1$
Binary	2	1
Ordered	M	$M-1$
Unordered	M	$2^{M-1}-1$

Table: Origin of the splitting variables

Impurity measures I

Once that, at a given node, the set of binary questions has been created, some criterion which guides the search in order to choose the best one to split the node is needed.
The idea of finding splits of nodes which generate more homogeneous descendant nodes has been implemented for *classification trees* by introducing the so called *impurity function*.

Impurity measures II

Let $p(j|t) \geq 0$ be the proportions of cases in node t belonging to class j with $\sum_{j=1}^{J} p(j|t) = 1$.
An impurity function ϕ is a function of the set of all J-tuples of numbers $p(j|t)$ with the properties:

1. ϕ is maximum only at the point $\{1/J, 1/J, \ldots, 1/J\}$;
2. ϕ achieves its minimum only at the points $(1, 0, \ldots, 0), (0, 1, \ldots, 0), (0, 0, \ldots, 1)$;
3. ϕ is a symmetric function of $p(j|t)$.

Impurity measures III

There are several impurity functions satisfying these three properties. The most common are:

1. the error rate, or the misclassification ratio:

$$i(t) = 1 - max_j p(j|t)$$

2. the Gini diversity index

$$i(t) = 1 - \sum_j p(j|t)^2$$

3. the entropy measure

$$i(t) = -\sum_j p(j|t) log(j|t)$$

Impurity measures IV

About regression trees, the splitting criterion is based on the search of that split that generates the most different descendant nodes in terms of mean value of the response variable.

$$i(t) = \frac{1}{N} \sum_{x_n \in t} (y_n - \bar{y}_t)^2$$

which can be meant as the total sum of squares (TSS), divided by N, where N is the sample size, $\bar{y}_t = \frac{1}{N_t} \sum_{x_n \in t} y_n$, N_t is the total number of cases in the node t where the sum is over all y_n such that $x_n \in t$.

Impurity measures V

If s is a proposed split of a generic node t into two offspring t_l and t_r , and p_l and p_r are the proportions of objects in node t which the split s puts into nodes t_l and t_r respectively, then a measure of the change in impurity which would be produced by split s of node t is given by:

$$\Delta i(t, s) = i(t) - [i(t_l)p_{t_l} + i(t_r)p_{t_r}.]$$

Δi, called *decrease in impurity*, can be used as splitting criterion: a high value means that a proposed split is a good one.

Impurity measures VI

At a given node t, a split s^* maximising the decrease in impurity is optimal and used for generate two descendants t_l and t_r .
Let $\tilde{T}$ be the set of all terminal nodes of the tree T: the total impurity of any tree T is defined as

$$I(T) = \sum_{t \in \tilde{T}} i(t)\, p(t)$$

Stopping rules I

Once the rules for growing the tree has been defined, another set of rules to stop the building of the structure are needed. There is no unique rule to define the stopping of the procedure, but there are several rules used according the discretion of the researcher. Tree growing can be arrested considering a suitable combination of the following conditions:

- *Bound on the decrease in impurity*.
  A node is terminal if the reduction in impurity due to the further partition of the node is lower than a fixed threshold; a node should be splitted if their contribution to the total impurity reduction is significant;

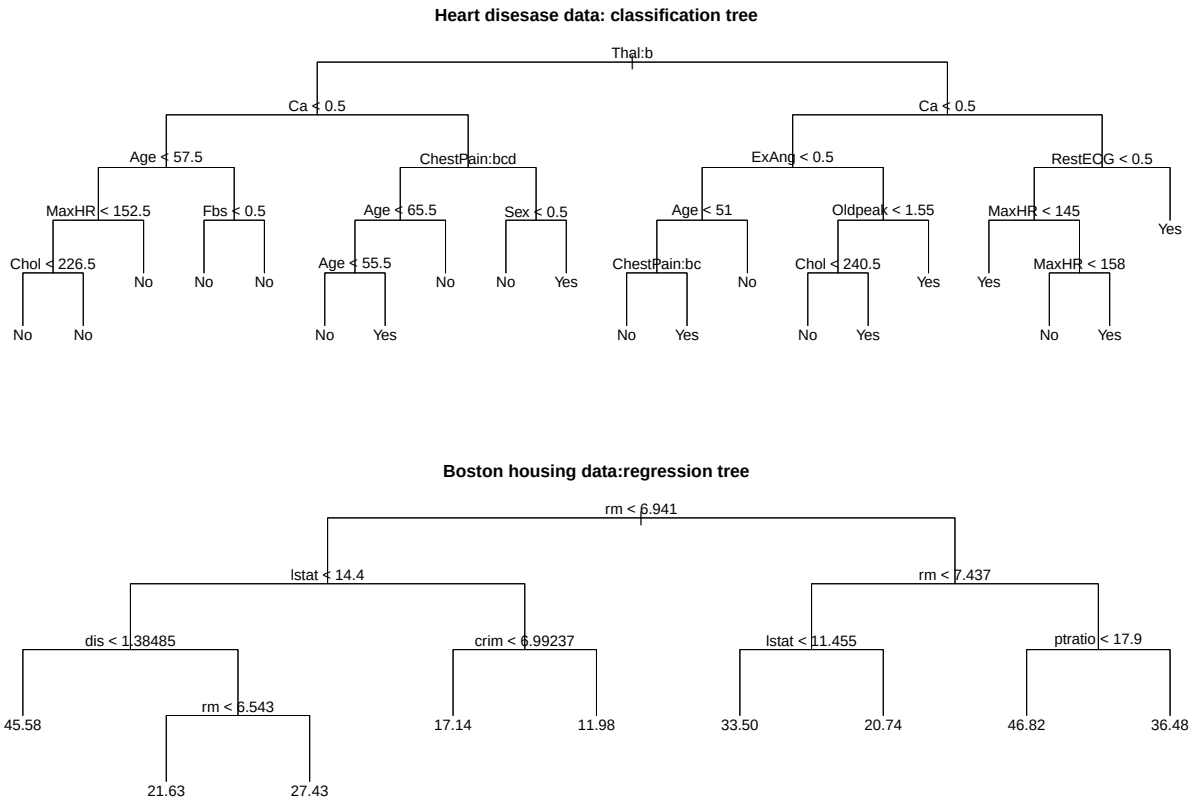
Stopping rules II

- *Bound on the number of observations*.
  In general, can be useless to continue splitting nodes with a few number of individuals: sample size within-node should be "rational";
- Tree size.
  A further condition could be based on either the total number of terminal nodes or the number of levels of the tree to limit its expansion.

Stopping rules III

Once the tree has been built, terminal nodes must be associated with a response.
In the case of classification trees the assignment of a response to each terminal node is based on a simple majority rule.
In the case the response variable is numeric the response values for the object falling into a given terminal node t can be summarised by means of a synthetic measure; in general this is simply given by the mean.

Classification and Regression trees: example



Tree pruning I

Exploratory trees can be used to investigate the structure of data but they cannot be used in a straightforward way for induction purposes.

A very large tree might overfit the data, while a small tree may not be able to capture the important structure.

Tree size is a tuning parameter governing the complexity of the model, and the optimal tree size should be adaptively chosen from the data.

Tree pruning II

In order to choose the "honest" tree in terms of its size, we refer to the *minimal cost-complexity pruning*. Before, the definition of an error measure of a tree structure is necessary.

Tree pruning III

- (Classification trees) The error at node t is defined as

$$r(t) = \frac{1}{n_t} \sum_{i=1}^{n_t} (\hat{Y}_t \neq Y_i)$$

where n_t is the size at t^{th} node, $\hat{Y}_t$ is the classification returned by the tree in the same node. The error rate of the overall tree is defined as

$$R(T) = \sum_{h \in H_T} r(t)p(t)$$

where H_T is the set of all terminal nodes of the tree T, and $p(t)$ is the proportion of cases falling into the t^{th} terminal node.

Tree pruning IV

- (Regression trees). The error rate is the sum of TSS in the t^{th} node divided by the total sample size, whereas the prediction error of overall tree is defined as

$$RR(T) = \frac{R(T)}{R(t_1)}$$

where $R(t_1)$ is the error in the root node.

Cost complexity pruning I

Let T_{max} be the maximum tree, let $\left|\tilde{T}\right|$ denote the set of all terminal nodes of T_{max}, that is its complexity. The cost-complexity measure is defined as

$$R_\alpha(T) = R(T) + \alpha \left|\tilde{T}\right|,$$

where α is a non negative complexity parameter which "governs the tradeoff between tree size and its goodness of fit to the data".

Cost complexity pruning II

The idea is, for each α, find the subtree $T^*_\alpha \supseteq T_{max}$ to minimize $R_\alpha(T)$. When $\alpha = 0$ the solution is the full tree T_{max}, and the more α increases the more the size of the tree decreases.
The cost complexity measure is defined for any internal node t and the branch T_t rooted at t as:

$$R_\alpha(t) = r(t)p(t) + \alpha$$
$$R_\alpha(T_t) = \sum_{h \in H_t} r(h)p(h) + \alpha \left| \tilde{T}_t \right|$$

where $R_{(t)}$ is the resubstitution error at node t, $p(t) = \frac{n(t)}{N}$ is the weight of node t given by the proportion of training cases falling in it and H_t is the set of terminal nodes of the branch having cardinality $\left| \tilde{T} \right|$.

Cost complexity pruning III

The branch T_t will be kept as long as:

$$R_\alpha(t) > R_\alpha(T_t)$$

the error complexity of node t being higher than the error complexity of its branch. As α increases the two measures tends to became equal, this occurs for a critical value of α that can be found solving the above inequality:

$$\alpha = \frac{R(t) - R(T_t)}{\left|\tilde{T}_t\right| - 1}$$

so that α represents for any internal node t the cost due to the removal of any terminal node of the branch.

Cost complexity pruning IV

The pruning process produces a finite sequences of subtrees $\Omega = T_1 \subset T_2 \subset \ldots \subset T_{max}$, where T_1 is a tree constituted only by the root node.

Note that the sequence of subtrees is always computed considering the training set.

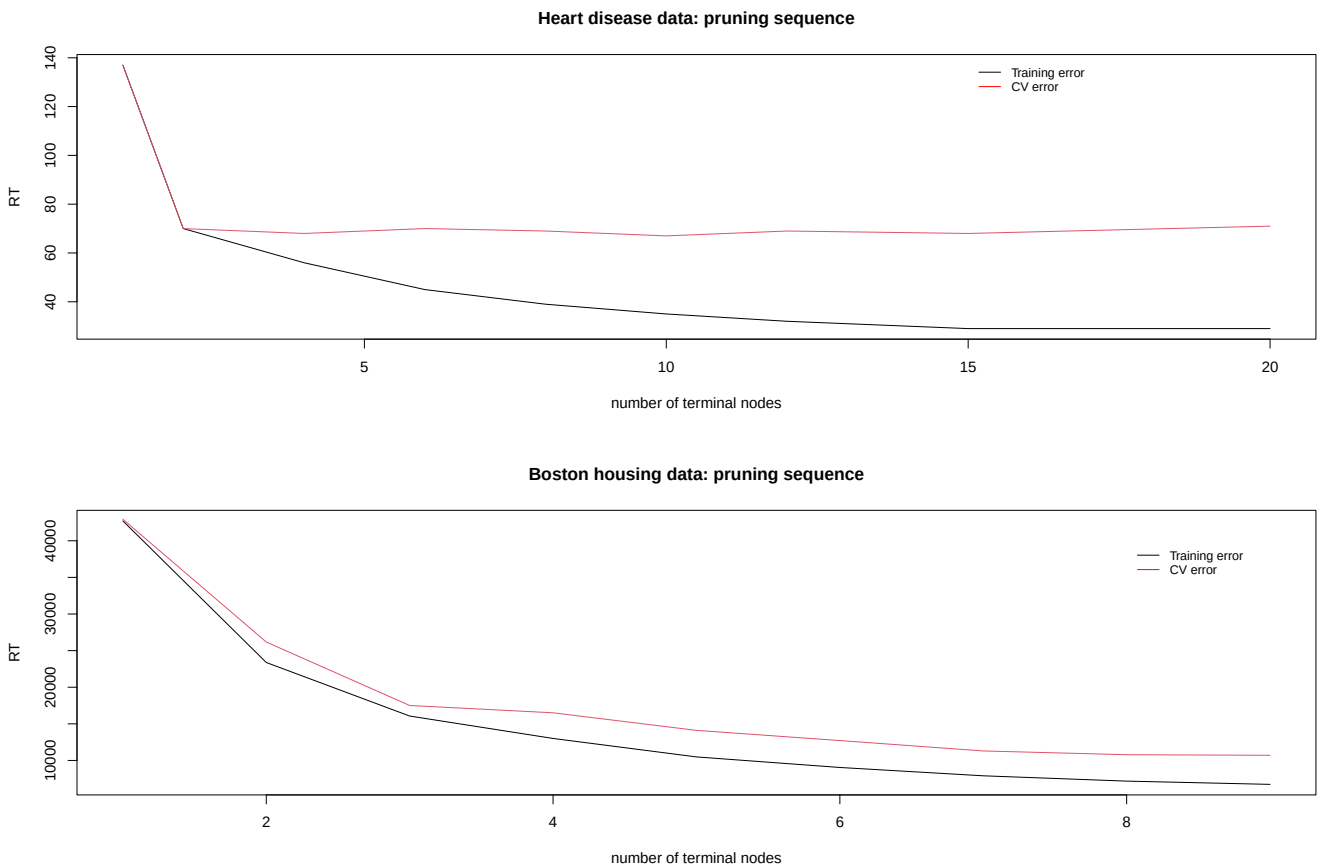
Cost complexity pruning V

There are three possible ways to select the "best" tree:

- *Resubstitution estimate*. It is an optimistic estimate, therefore it is not used.
- *Test set estimate*, if the sample size is sufficiently large.
- *Cross validation estimate*, ehen sample size is not sufficiently large to be splitted into two sub-samples.

A single final tree is then selected either as the one producing the smallest error estimate on an independent test set ($0 - SErule$) or the one which error estimate is within one standard error of the minimum ($1 - SErule$).

Pruning sequence: example



Two-stage I

Two-Stage splitting criterion to choose the best split was proposed by Mola and Siciliano (1992). This approach relies on the assumption that a predictor X_k is not merely used as a generator of partitions but it plays also a global role in the analysis.
At any node the two stages can be defined as:

- **global selection**; one or more predictors are chosen as the most predictive for the response variable according to a given criterion; the selected predictors are used to generate the set of partitions or splits. In this stage an index needs to be defined to evaluate the Global Impurity Proportional Reduction (Global IPR) of the response variable Y at node t, due to the predictor X;

Two-stage II

- **local selection**; the best partition is selected as the most predictive and discriminatory for the subgroups according to a given rule. In this stage one has to define an index as the Local Impurity Proportional Reduction (Local IPR) of the response Y due to the partition p generated by the predictor X.

Two-stage III

For classification trees the Global IPR is defined as τ index of Goodman and Kruskal

$$\tau_t(Y|X) = \frac{\sum_i \sum_j p_t^2(j|i) p_t(i) - \sum_j p_t^2(j)}{1 - \sum_j p_t^2(j)},$$

where $p_t(i)$, for $i = 1, \ldots, I$, is the proportion of cases in node t that have category i of X, and $P_t(j|i)$, for $j = 1, \ldots, J$, is the proportion of cases in the node t belonging to class j of Y given the i^{th} category of X. Note that the denominator in the equation is the Gini diversity index.

Two-stage IV

For classification trees, at each node t of the splitting procedure, a split s of the I categories of X into two sub-groups (e.g. $i \in l$ or $i \in r$), leads to the definition of a splitting variable X_s with two categories denoted by l and r. Local IPR is defined as

$$\tau_t(Y|s) = \frac{\sum_j p_l^2(j|tl)p_{tl} + \sum_j p_{tr}^2(j|r)p_{tr} - \sum_j p_t^2(j)}{1 - \sum_j p_t^2(j)}$$

Two-stage V

For regression trees, Global IPR can be defined as the Pearson's squared correlation η^2:

$$\eta^2_{Y|X}(t) = \frac{BSS_{Y|X}(t)}{TSS_Y(t)}$$

where SST is the total sum of squares of the numerical response variable Y and BSS is the between group sum of squares due to the predictor X.

Two-stage VI

for regression trees, local IPR is defined as

$$\eta^2_{Y|s}(t) = \frac{BSS_{Y|s}(t)}{TSS_Y(t)}$$

Two stage splitting criterion works as follow:

1. select the best predictor $X^*(t)$ at t node by maximising the Global IPR;
2. select the best split $s^*(t)$ at node t by maximizingthe Local IPR

Fast Algorithm for Splitting Tree I

FAST algorithm (Fast Algorithm for Splitting Tree) defined by Mola and Siciliano (1997) provides a faster method to find the best split at each node when using CART methodology.
Main issue of FAST is that the measure of Global IPR measure satisfies the following property:

$$\gamma(Y|X) \geq \gamma(Y|s)$$

in which γ is the generic Global IPR measure, and s is the set of split generated by X variable.

Fast Algorithm for Splitting Tree II

FAST algorithm consists in two step:

- computing Global IPR measure for all variables belonging to the predictor matrix X and sort in decreasing order these measures;
- computing Local IPR measure for the first previously ordered variable with maximum Global IPR. If Local IPR of this variable is higher than Global IPR of the second ordered X variable, stop the procedure, otherwise continue until inequality is satisfied.

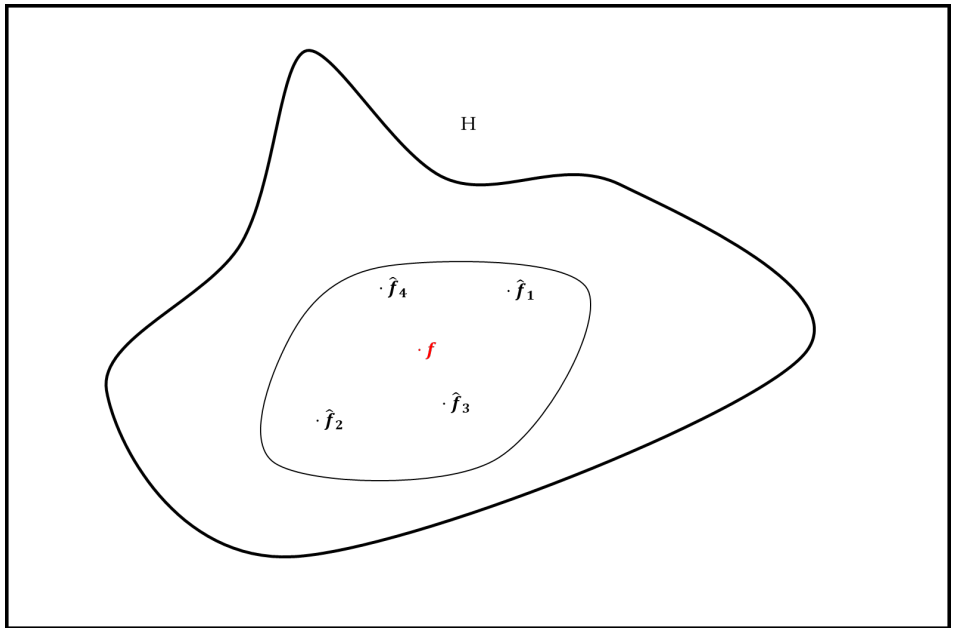
Ensemble methods I

Ensemble methods are learning algorithm that construct a set of classifiers and then classify new data points by taking a vote of their predictions.

A necessary and sufficient condition for an ensemble of classifiers to work better than any single classifier is that the classifiers to be aggregate must be accurate (e.g. they must have an error rate better than random choices) and diverse (e.g. the errors of the classifiers have to be unrelated).

Three reasons for constructing ensembles I

1. Statistical reason

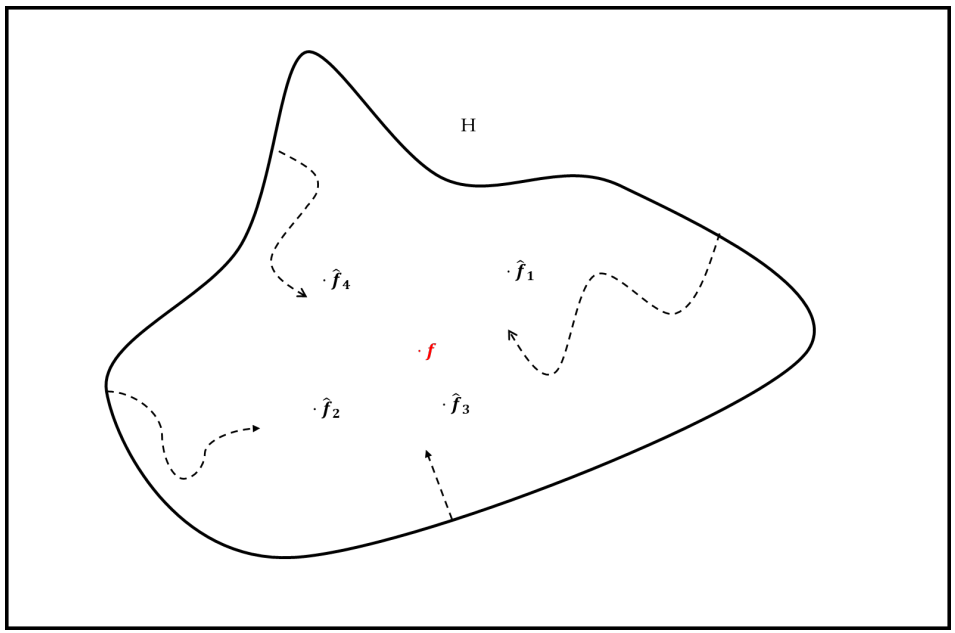


The outer curve denotes the hypothesis space H. The inner curve denotes the set of hypotheses that all give good accuracy on the training data. The point labeled f is the true hypothesis. Averaging the accurate hypotheses,we can find a good approximation to f.

"A learning algorithm can be viewed as searching a space H of hypotheses to identify the best hypothesis in the space. The statistical problem arises when the amount of training data available is too small compared to the size of the hypothesis space. Without sufficient data, the learning algorithm can find many different hypotheses in H that all give the same accuracy on the training data" (Dietterich, 2000)

Three reasons for constructing ensembles II

2. Computational reason

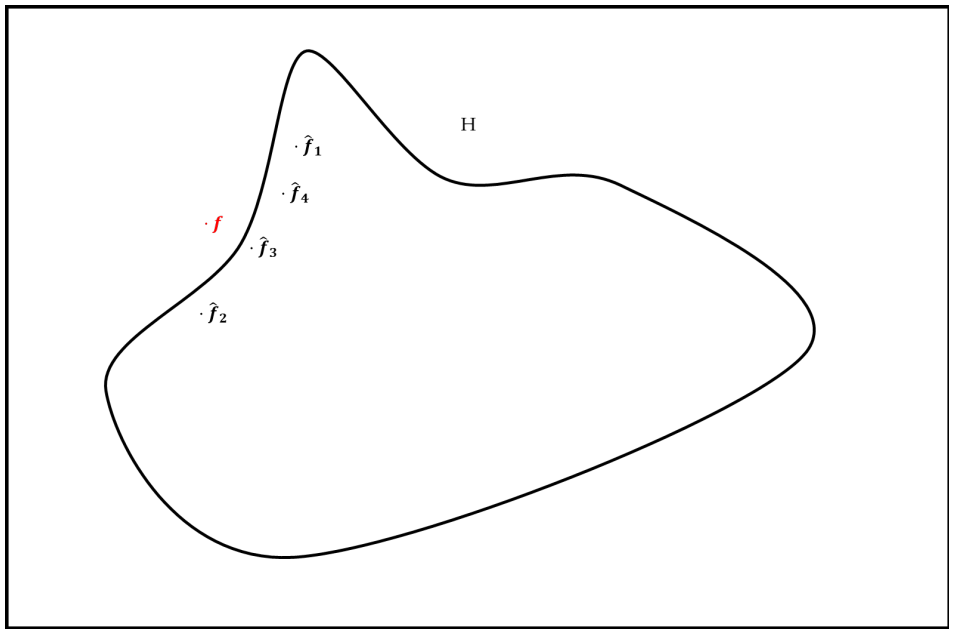


An ensemble constructed by running the local search from many different starting points may provide a better approximation to the true unknown function than any of the individual classifiers

"Many learning algorithms work by performing some form of local search that may get stuck in local optima... In cases where there is enough training data (so that the statistical problem is absent), it may still be very difficult computationally for the learning algorithm to find the best hypothesis...
An ensemble constructed by running the local search from many different starting points may provide a better approximation to the true unknown function than any of the individual classifiers" (Dietterich, 2000)

Three reasons for constructing ensembles III

3. Representational reason



The true function f cannot be represented by any of the hypotheses in H.

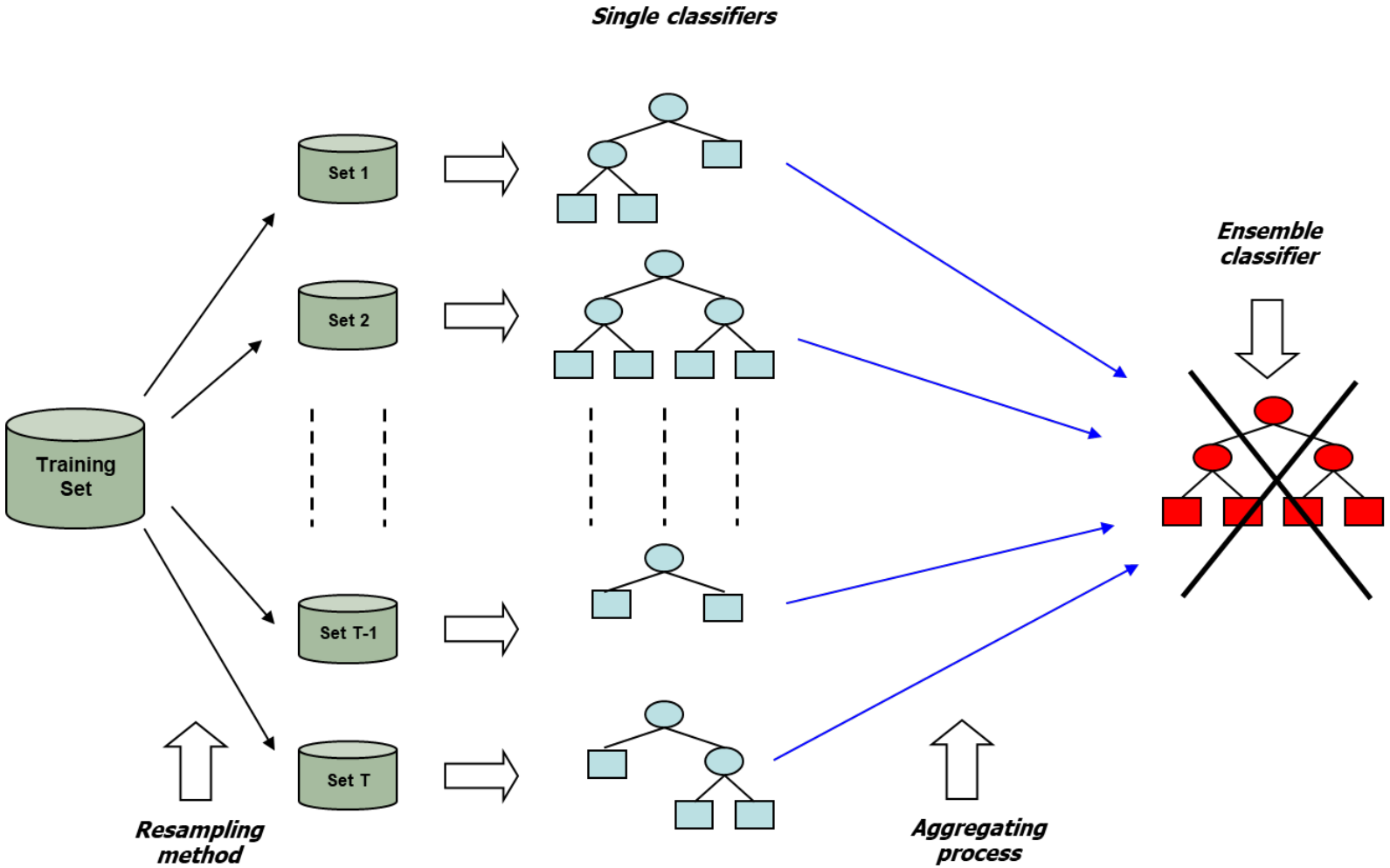
""In most applications of machine learning, the true function f cannot be represented by any of the hypotheses in H. By forming weighted sums of hypotheses drawn from H, it may be possible to expand the space of representable functions'"' (Dietterich, 2000)

How build up ensembles? I

There are two ways to building up ensemble methods:

- Using classifiers that are really different only theoretical, never used
- Manipulating training examples bagging, boosting, random forests

How build up ensembles? II



Manipulating training examples

Training examples are manipulated through re-sampling techniques to generate multiple classifiers, then a learning algorithm is run several times with a different subset of training examples.

The most famous ensembles belonging to this category are Bagging and Boosting and Random Forests.

Bagging I

Bagging is an acronym for Bootstrap Aggregating: it forms a set of classifiers that are combined by voting by generating replicated bootstrap samples of the data.

Idea: Let $X_1, \ldots, X_n$ be a set of i.i.d. observations with constant variance equal to σ^2: then the variance of $\bar{X}$ is equal to σ^2/n. "Averaging a set of observations reduces variance" (Breiman).

Bagging is an ensemble that reduces the variance (remember: prediction error is formed by irreducible error$+b^2+\sigma^2$)

Bagging II

Given a learning set $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$, the aim is to predict y using x as input by using a classifier $h(x, \mathcal{L})$. By using a sequence of t learning sets $\mathcal{L}_t$, with $t = 1, \ldots, T$, each consisting of n independent observations from the same distribution as in $\mathcal{L}$, goal is to get a better predictor than the single learning predictor set $h(x, \mathcal{L})$ coming from t bootstrap replications. The aggregating process is quite simple: if y is numerical the aggregated classifier is the average of each single classification over all the iterations of the procedure, if y is numerical the method of aggregating the classifiers is by voting.

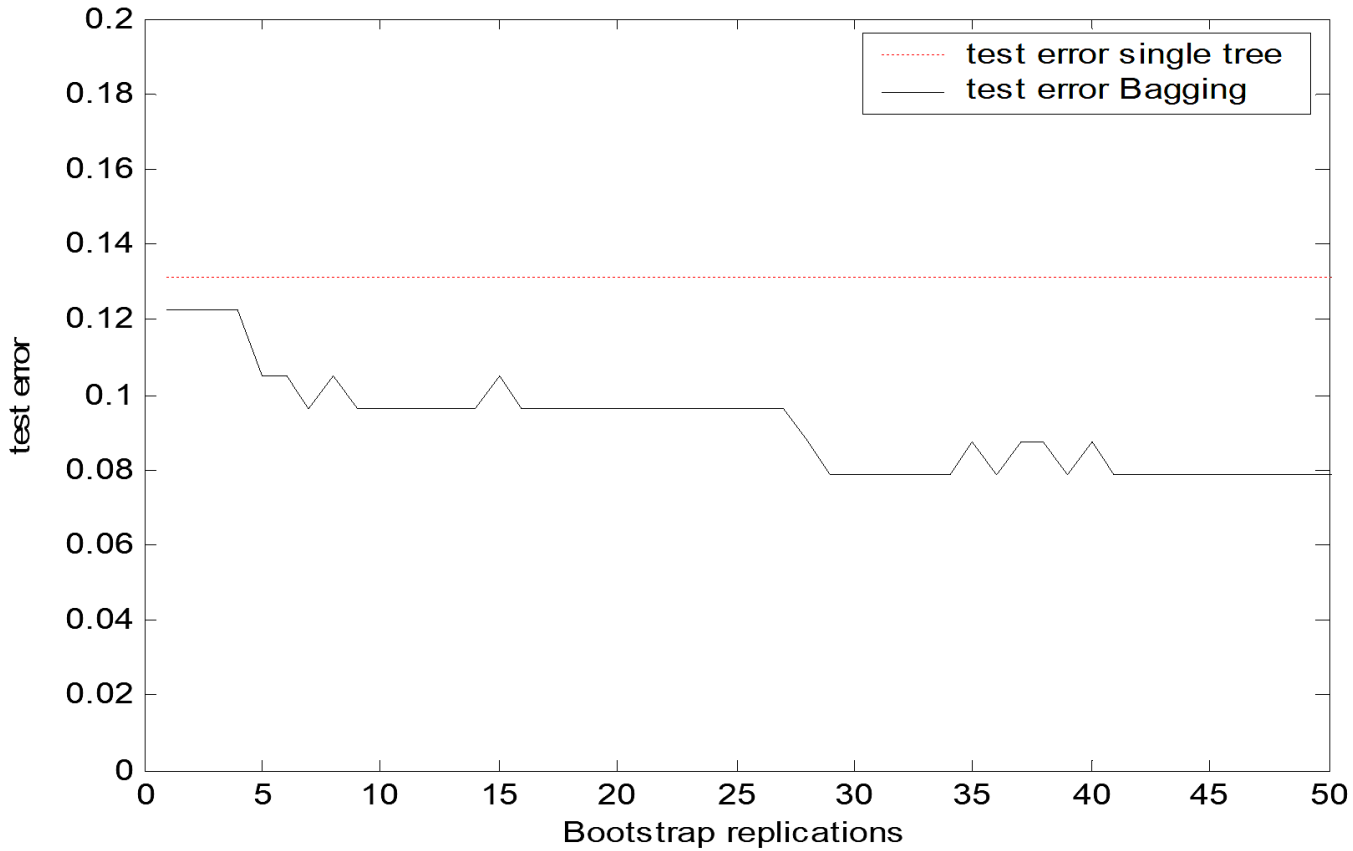
Bagging III

Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and $y_i \in R$ if numerical or $y_i \in \{1, \ldots, J\}$ if categorical.

- for $t = 1 : T$
  - generate a bootstrap replication $\mathcal{L}^B$ from $\mathcal{L}$
  - run a single classifier on $\mathcal{L}^B$
  - obtain the estimation $\hat{y}_i^t$ from the single classifier
- Output: final bagged classifier

$$\mathcal{H}(X) = \begin{cases} \text{aggregation by voting if } y_i \in \{1, \ldots, J\} \\ {}_{av}h\left(x, \mathcal{L}^B\right) \text{ if } y_i \in R \end{cases}$$

Bagging IV

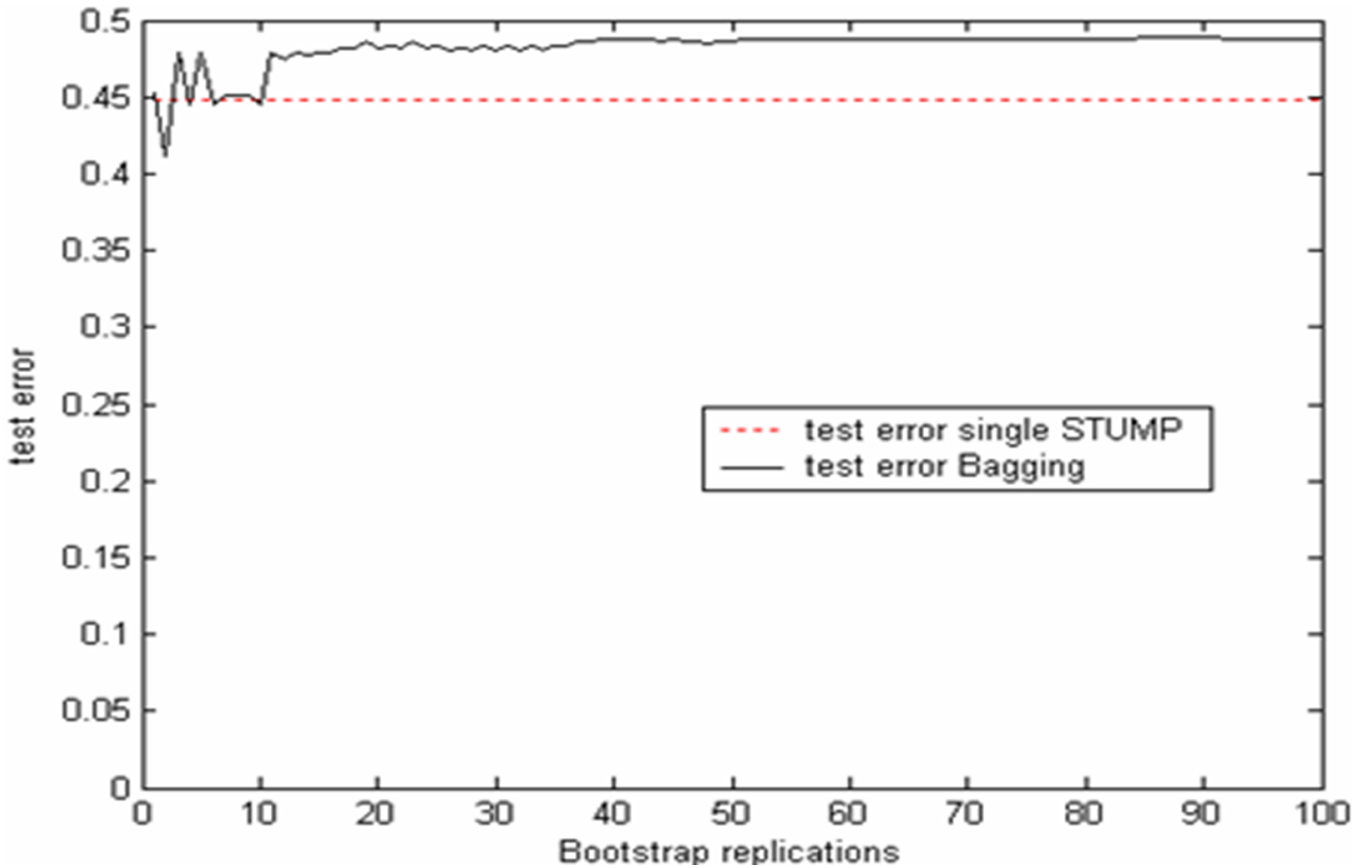


Wdbc dataset: (UCI Machine Learning Repository). Classifier: CART. Bootstrap replications: 50

Bagging, variance and bias I

Bagging works well with unstable classifiers. In general, a classifier is unstable when it is affected by high variance, whereas a classifier is stable when it is affected by high bias.
Bagging is a method of variance reduction.It works well with classification and regression trees because their are known as methods with high variance.
Bagging returns worse performance than single classifiers when it is used with stable classifiers, e.g. with a *stump*. *Bagging unstable classifiers usually improve them. Bagging stable classifiers is not a good idea* (Breiman, 1996).

Bagging, variance and bias II



Wdbc dataset: (UCI Machine Learning Repository). Classifier: STUMP. Bootstrap replications: 100. The performance of Bagging is worse than the one of the single classifier if it has low variance and high bias

Out-of-bag

On average, each bagged tree makes use of around two-thirds of the observations. The remaining one-third of the observations not used to fit a given bagged tree are referred to as the *out-of-bag* (OOB) observations. We can use the OOB observations to estimate the prediction error.
The OOB approach for estimating the test error is particularly convenient when performing bagging on large data sets for which cross-validation would be computationally onerous.

From Bagging to Random Forests

- Bagging today is no more used
- Other ensembles perform better (e.g., Boosting)
- Bagging principle still survives
- Random forests were born from Bagging

Random Forests I

Random forests provide an improvement over bagging.

Idea: decorrelate trees. (Breiman 1999, 2001)

Let $X_1, \ldots, X_n$ a set of **i.d.** observations with constant variance equal to σ^2. Assuming positive pairwise correlation ρ, we have that the variance of $\bar{X}$ is equal to $\rho\sigma^2 + (1-\rho)/n\sigma^2$.

The principle that inspired the Random Forests was "destroy" the correlation structure of the trees.

Random Forests II

As in bagging, a number of trees is built on bootstrapped training samples. On the other hand, when building these trees, a random sample of m predictors is chosen as split candidates from the full set of p predictors.
As a rule of thumb, a fraction $m \approx \sqrt{p}$ of predictors is used in each iteration.
Random forests can be used to rank the importance of variables in a regression or classification problem.

Random Forests III

Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and $y_i \in R$ if numerical or $y_i \in \{1, \ldots, J\}$ if categorical.

- for $t = 1 : T$
  - generate a bootstrap replication $\mathcal{L}^B$ from $\mathcal{L}$
  - grow a random forest to $\mathcal{L}^B$ by recursively repeating the following steps for each terminal node of the tree:
    1. select m variables at random from the p predictors
    2. pick the best variable/split-point among the m
  - obtain the estimation $\hat{y}_i^t$ from the single forest
- Output: final classifier

$$\mathcal{H}(X) = \begin{cases} \text{aggregation by voting if } y_i \in \{1, \ldots, J\} \\ {}_{av}h\left(x, \mathcal{L}^B\right) \text{ if } y_i \in R \end{cases}$$

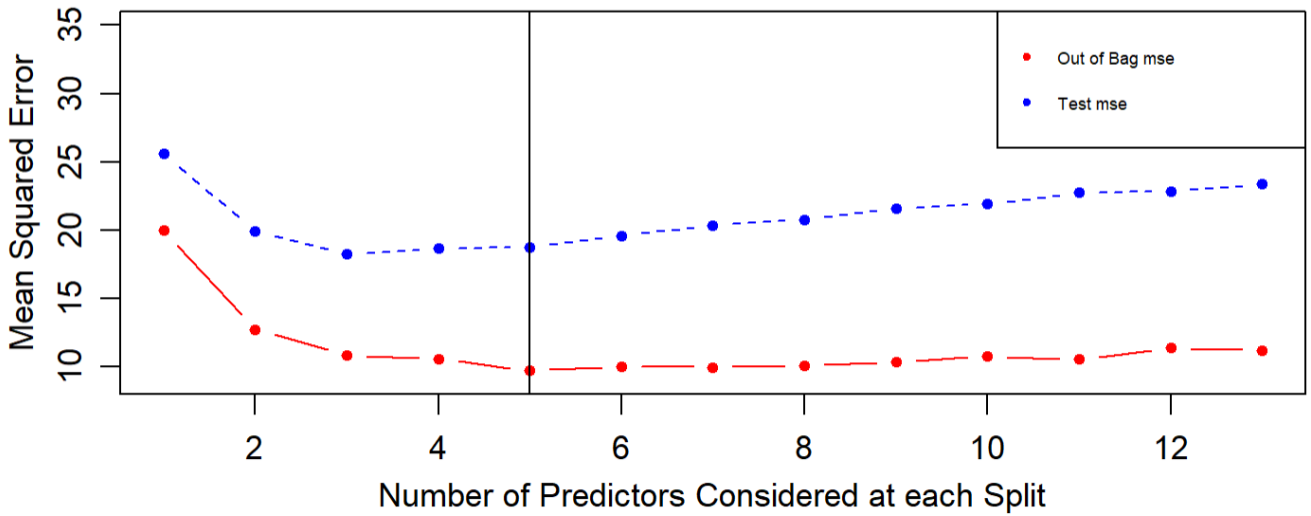
Random Forests IV

The out-of-bag observations are useful to observe the so-called out-of-bag error (OOB), which can be evaluated as a stabilizer for Random Forests.

When the OOB error is stable, then the training phase of Random Forests can be considered finished.

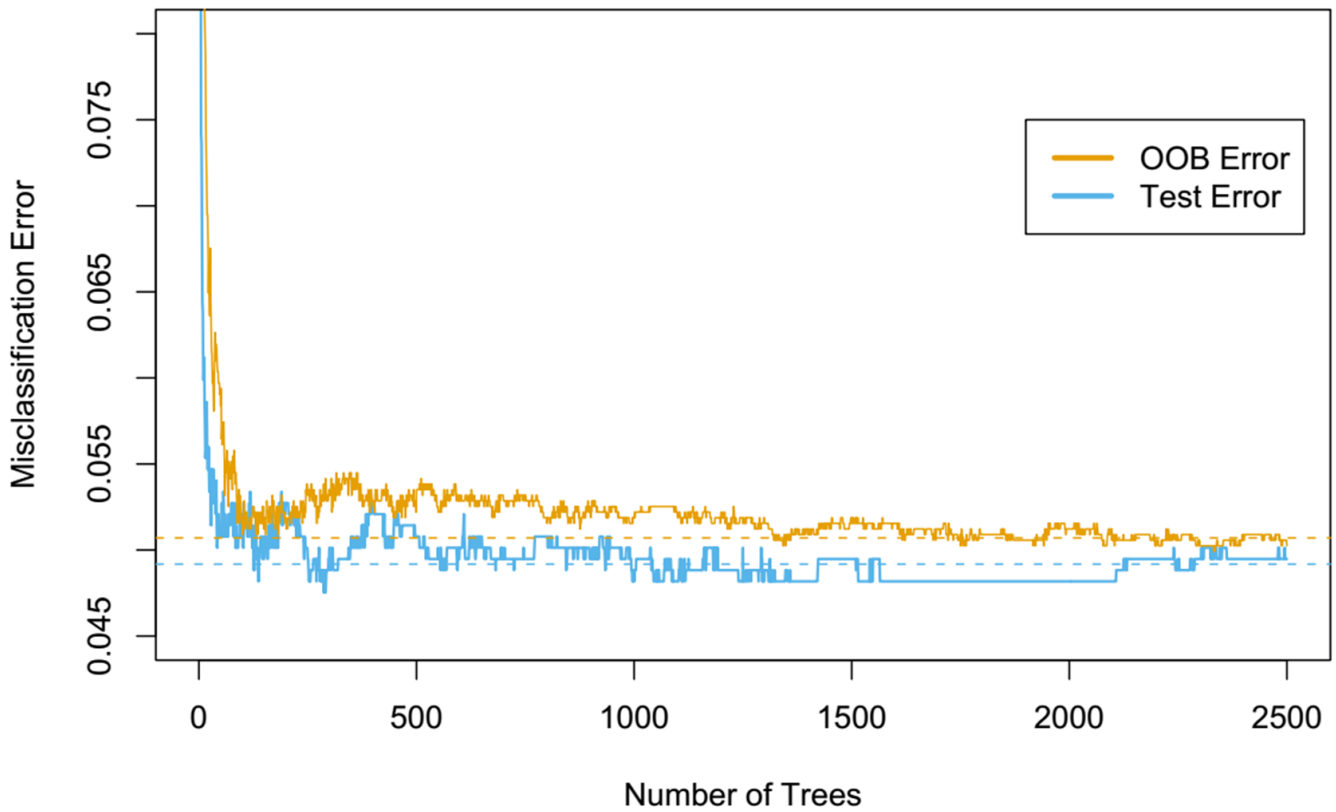
The OOB error is "almost identical to that obtained with n-fold cross validation" (Hastie, Tibshirani & Friedman (2008))

Random Forests V



Boston housing data set: (UCI Machine Learning Repository). Example of using out-of-bag error to chose the "right" number of predictors. In this case $m = 5$ predictors are suggested. Note that $m = 13$ means doing Bagging.

Random Forests VI



Spam data set: (UCI Machine Learning Repository). Classifier: CART. Bootstrap replications: 2600

Random Forests and Variables Importance I

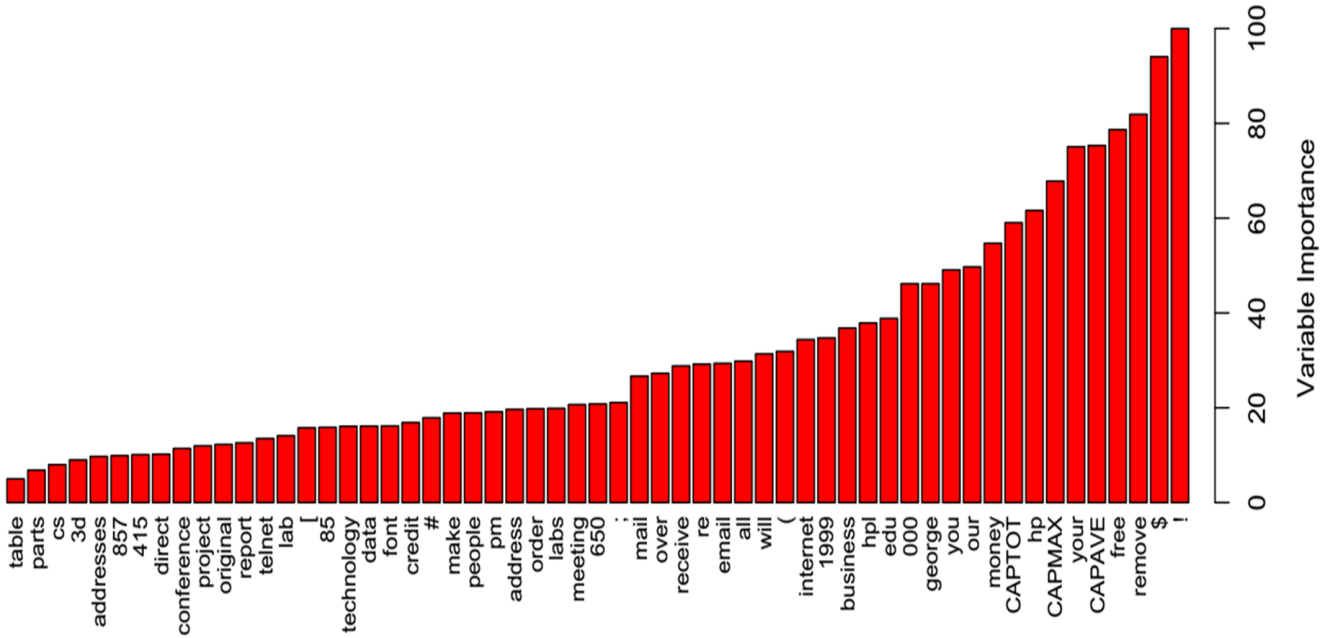
Ensemble methods don't allow interpretation of phenomena, they belong to the "black-box" classifiers' category.

Variable importance ranking is an aspect that, jointly with a prediction accuracy comparable with Boosting, has made Random Forests very popular

At each split generated by each single forest, the total amount of decrease in RSS (or Gini index) associated with the predictor that generated it is kept.

At the end of the procedure, the average of these quantities is calculated for each predictor, and the result is a graph that shows the importance of each variable in the 'explanation' of the phenomenon.

Random Forests and Variables Importance II



Spam data set: (UCI Machine Learning Repository). Example of variable importance with Random Forests

Boosting I

Boosting is a general method for improving the accuracy of any given learning algorithm provided that single classifications are better than random choices.
Here we present the original boosting algorithms. The main difference between Bagging and Boosting algorithms is that whereas Bagging uses the bootstrap as resampling method, Boosting uses a *weighted* bootstrap, in the sense that the probability of each individual to be included in the boosted training sample is not constant, but it is weighted by the (good or bad) classification obtained by the learning sample.

Boosting II

Starting from a uniform distribution of weights, these are increased for the i-th individual if he has been misclassified by the learning algorithm (or *weak learner*), otherwise these are decreased. This way, within the next iteration the probability of a misclassified instance to be included in the boosted training sample is higher than observations correctly classified, so the learning algorithm is forced to learn by its errors becoming a *strong learner*.
Several boosting algorithms are developed in the last years (polynomial-time boosting algorithm, boosting-by-majority algorithm,...). Doubtless the most famous boosting algorithm is AdaBoost developed by Freund and Schapire (1995)

Boosting III

Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and $y_i = \{-1, +1\}$

- initialize $D_1 = \frac{1}{n}$ for $i = \{1, \ldots, n\}$
- for $t = 1 : T$
  - train weak learner h_t using distribution D_t
  - obtain a weak hypotesys $h_t : X \rightarrow \{-1, +1\}$
  - compute the error $\epsilon_t = Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$
  - choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$
  - update D distribution:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \epsilon^{-\alpha_t} & \text{if } h_t(i) = y_i \\ \epsilon^{\alpha_t} & \text{if } h_t(i) \neq y_i \end{cases}$$

$$= \frac{D_t(i) \exp\left(-\alpha_t y_i h_t\left(x_i\right)\right)}{Z_t}$$

where Z_t is a normalization factor

- Output: final boosted classifier:

$$\mathcal{H}(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t\left(x\right)\right)$$

Adaboost for binary case

Boosting IV

For both multiclass and regression cases, main difference with the above described AdaBoost algorithm is about a suitable definition of the error and the computation of a loss function.

Multiclass boosting

Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and $y_i = \{1, \ldots, J\}$

- initialize $D_1 = \frac{1}{n}$ for $i = \{1, \ldots, n\}$
- for $t = 1 : T$
  - train weak learner h_t using distribution D_t
  - obtain a weak hypotesys $h_t : X \rightarrow \{1, \ldots, J\}$
  - compute the error $\epsilon_t = \sum_i D_t(i) I\left(h_t(x_i) \neq y_i\right)$
  - choose $\alpha_t = \ln\left(\frac{|J-1|\,(1-\epsilon_t)}{\epsilon_t}\right)$
  - update D distribution:

$$D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t I\left(h_t(x_i) = y_i\right)}}{Z_t}$$

where Z_t is a normalization factor

- Output: final boosted classifier:

$$\mathcal{H}(x) = \mathit{arg}\max_{y \in J}\left(\sum_{t=1}^{T} \alpha_t I\left(h_t(x) = y\right)\right)$$

AdaBoost algorithm for multiclass classifiers

(A variant of) Regression Boosting

Let $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and $y_i \in R$

- initialize $D_1(i) = \frac{1}{n}$
- for $t = 1 : T$
  - train weak learner h_t using distribution D_t
  - obtain a weak hypotesys $h_t : x \to y$
  - compute the quadratic loss function $L_t(i) = (y_i - h_t(x_i))^2$
  - compute an average loss $\epsilon_{D_t} = \sum_{i=1}^{n} D_t(i) L_t(i)$
  - set $\beta_t = \frac{\epsilon_{D_t}}{\max_{1 \le i \le n} L_t(i) - \epsilon_{D_t}}$
  - set $w_k(i) = \frac{L_t(i)}{\max_{1 \le i \le n} L_t(i)}$
  - set $g_t(i) = \beta_t^{1-w_k(i)} D_t(i)$
  - update D distribution:

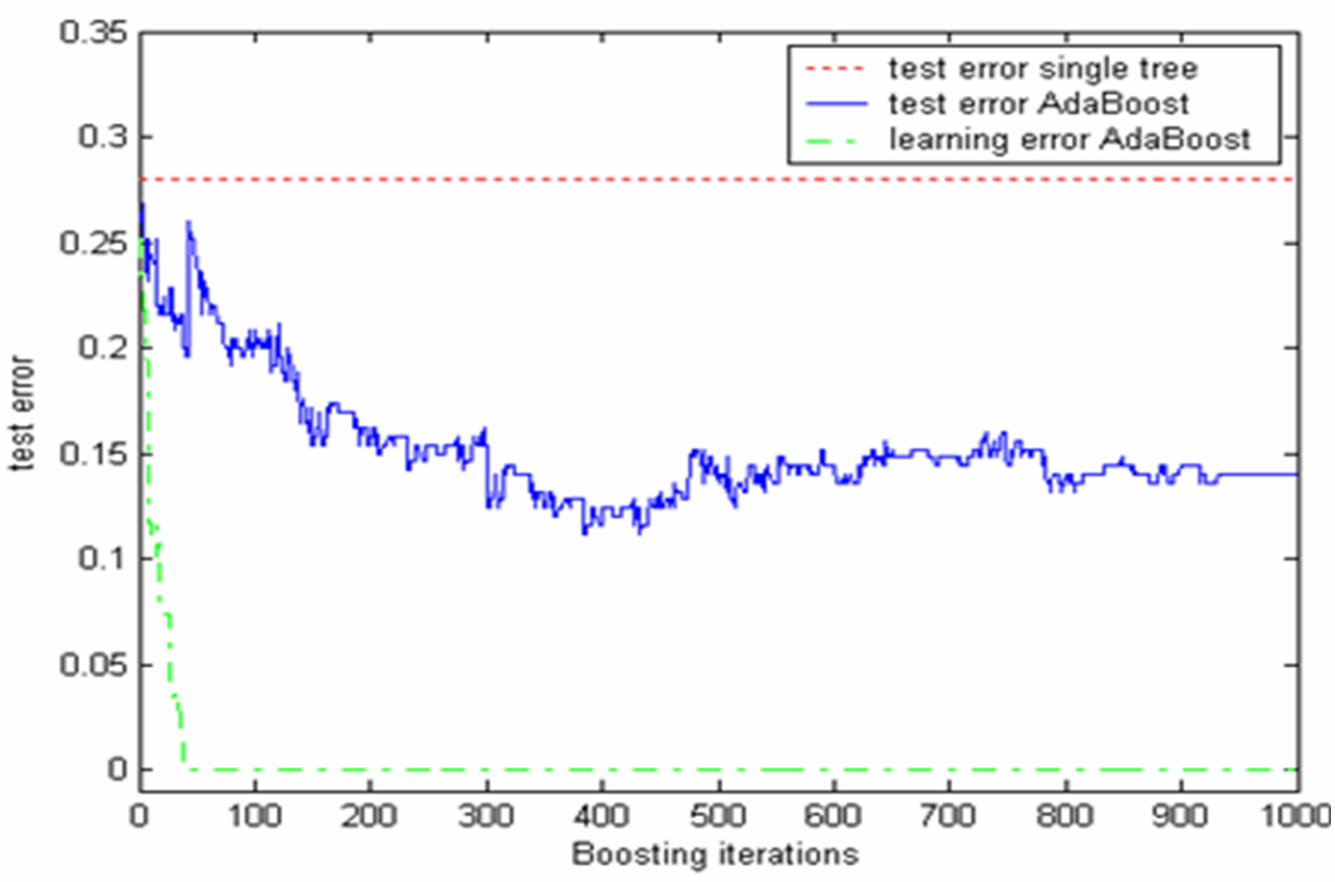
$$D_{t+1}(i) = \frac{g_t(i)}{\sum_i g_t(i)}$$

- Output: final boosted classifier:

$$\mathcal{H}(x) = inf \left\{ y \in Y : \sum_{t:h_t \le y} \log\left(\frac{1}{\beta_k}\right) \ge \frac{1}{2} \sum_t \log\left(\frac{1}{\beta_t}\right) \right\}$$

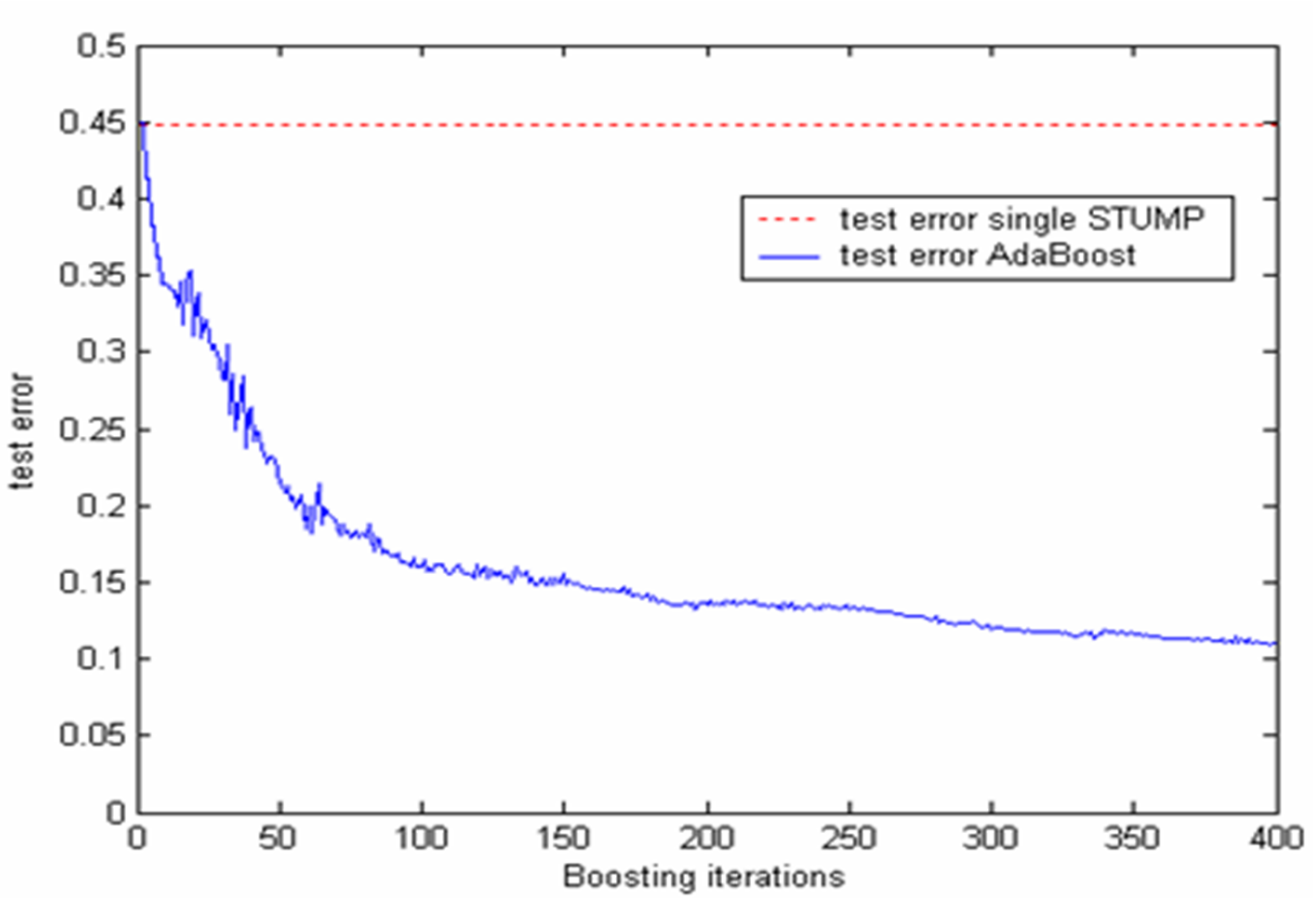
AdaBoost algorithm for regression problems

Adaboost: some examples I



German credit data set: (UCI Machine Learning Repository). The learning error goes to zero

Adaboost: some examples II



German credit data set: (UCI Machine Learning Repository). Adaboost works also reducing bias

Download and read the following papers:

Mola, F., and Siciliano, R. (1997). A Fast Splitting Procedure for Classification and Regression Trees, *Statistics and Computing*, 7, Chapman Hall, 208-216.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2), 123-140.

Freund, Y., Schapire, R., and Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780), 1612.

Breiman, L. (2001). Random Forests. *Machine learning*, 45, pp. 5–32