# Recursive partitioning methods and Ensemble methods: an introduction

# Chapter 1

# Recursive partitioning methods

## 1.1 Classification and Regression Trees

Binary segmentation procedure consists of a recursive binary partition of a set of objects described by some explanatory variables (either numerical or and categorical) and a response variable. In the following, CART procedure (Breiman et al., 1984) is followed.
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 (see figure 1.1). Each node has a number such that generic node $t$ generates the left node $2t$ and the right node $(2t+1)$. This approach

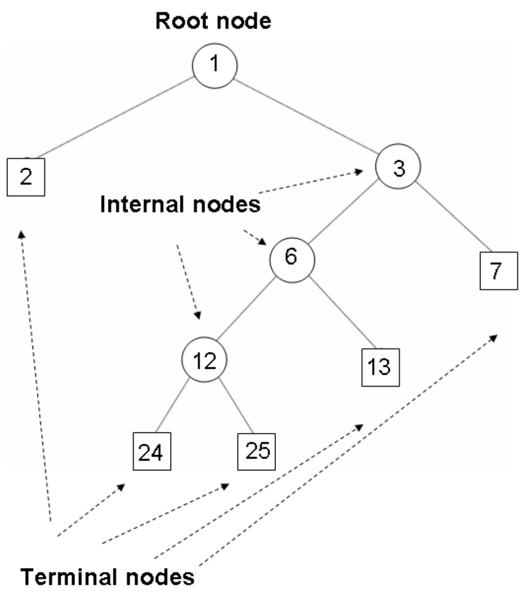

Figure 1.1: Tree-based structure

was proposed by authors of statistical software SPAD (Cisia Institute, France). In this way, it is always possible to recognize the position of each node given its number deriving the path from the node to the root node and vice versa. In example, in the above figure, the node 6 is the left node of its parent node 3 which is the right node of its parent node 1 (the root node).
Once the tree is built, a response value or a class label is assigned to each terminal node. According to the nature, categorical or numerical, of the response variable, in the framework of binary segmentation procedures a distinction is made between Classification Tree (for the

categorical response case) and Regression Tree (for the numerical response case). In classification tree case, when the response variable takes value in a set of previously defined classes, the node is assigned to the class which presents the highest proportion of observations (by voting); whereas in the regression tree case, the value assigned to cases in a given terminal node is the mean of the response variable values associated with the cases belonging to the given node. In both cases this assignment is probabilistic, in the sense that a measure of error is associated to it.

The main aim of the procedure is to define a classification/prediction rule on the basis of a learning set (also called training set), for which the values of a response variable $Y$, and of a set of $K$ explanatory variables $(X_1, \ldots, X_k, \ldots, X_K)$ (either numerical or/and categorical) have been recorded.

The recursive partitioning procedure follows a *divide and conquer* algorithm, in the sense that in principle the algorithm continues partitioning nodes until all leaves contain a single case or cases either belonging to the same class or presenting the same response value. This leads to overlarge trees with many rules which are hard to understand and overfit the data.

In practice, when performing binary segmentation one has to look for a compromise that allow for the trade-off between the *exploratory* and the *confirmatory* purposes of the tree structures methodology. A distinction is made between the two problems involved in investigating the data sets: that is, whether to explore dependency, or to predict and decide about future responses on the basis of the selected predictors.

*Explanation* can be obtained by performing a segmentation of the objects until a given stopping rule defines the final partition of the objects to interpret.

*Confirmation* is a completely different problem that requires the definition of decision rules, usually obtained by performing a *pruning*

procedure soon after a segmentation procedure. Therefore, a further step, tree simplification, is usually carried out to avoid *overfitting* and improve the understandability of the tree by retrospectively pruning some of the branches.
Summarising, 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

## 1.2 Splitting criteria

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. 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 $X_k$. 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:

$$X_1 \leq 3.5?$$
$$X_2 \leq 5?$$
$$X_3 \in \{a_1, a_3\}\,?$$

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. Table 1.1 reports the number of splitting variables that can be generated by any type of explanatory variable according to its scale of measurement. To each tree node the algorithm generates all the possible splitting variables and searches through them one by one, so the easiest case to deal with is binary variables that can generate just a single splitting variable, numeric variables are treated as ordered with $N$ categories. Finally, unordered variables are the most difficult to deal with because they can generate a very large number of splitting variables even for a small value of $M$. 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. As said before, the key idea is to split each node so that each descendant is more homogeneous than the data in the parent node. To

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

Table 1.1: Origin of the splitting variables

reach this aim, we need a measure of homogeneity to be evaluated by means of a *splitting criterion.* In the CART methodology 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.*
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 $\phi$ is a function of the set of all $J$-tuples of numbers $p(j|t)$ with the properties (Breiman et al., 1984, , p. 24):

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

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
$$i(t) = -sum_j (pj|t) log(j|t)$$

Another splitting criterion is the *twoing*, that can be used for multi-class classification problems. The idea is to find that grouping of all $J$

classes into two super-classes so that considered as a two-class problem, the greatest decrease in node impurity is realized. The following rules determine the optimal super-classes for a candidate split:

$$C_L = \{j : p_j(t_l) \geq p_j(t_r)\}$$
$$C_R = \{j : p_j(t_l) < p_j(t_r)\},$$

where $C_L$ and $C_R$ are two super-classes. The left super class has all of the target classes that tend to go left. The right super class has all of the target classes that tend to go right. Once the super-classes have been determined, the rest of the calculation is the same as the for the Gini criterion with the super classes as the binary target.

Talking 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.

$$i\,(t) = \frac{1}{N} \sum\nolimits_{x_n \in t} (y_n - \bar{y}_t)^2 \tag{1.1}$$

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

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:

$$\Delta i(t, s) = i(t) - [i(t_l)p_{t_l} + i(t_r)p_{t_r}] \tag{1.2}$$

$\Delta i$, called decrease in impurity, can be used as splitting criterion: a high value means that a proposed split is a good one. At a given node

$t$, a split $s^*$ maximising equation 1.2 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

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

To proceed with tree growing, CART procedure must compute the decrease in impurity associated to each possible split generated by each variable. For example, suppose to have a binary response variable and a set of six predictors as defined in table 1.2. In the root node the

| Variable | Nature | Categories | # of split |
|---|---|---|---|
| $X_1$ | *Nominal* | 6 | $2^5-1=31$ |
| $X_2$ | *Nominal* | 7 | $2^6-1=63$ |
| $X_3$ | *Ordinal* | 3 | $3-1=2$ |
| $X_4$ | *Binary* | 2 | 1 |
| $X_5$ | *Ordinal* | 5 | $5-1=4$ |
| $X_6$ | *Ordinal* | 4 | $4-1=3$ |

Table 1.2: Example of generation of splits according to the nature of the predictors

number of decreases in impurity to be computed is 31+63+2+1+4+3 = 104. Therefore, computational cost of CART is really high, because this procedure must be repeated until a stooping rule in tree-building occurs.

### 1.2.1 Two Stage splitting criterion

Mola and Siciliano (Mola and Siciliano, 1992, 1994) have proposed a Two-Stage splitting criterion to choose the best split. 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.

In the first stage, a variable selection criterion is applied to find one or more predictors that are the most predictive for the response variable. On the basis of the set of partitions generated by the selected predictor(s), a partitioning criterion is considered in the second stage in order to find the best partition of the objects at a given node. The criteria to be used in the two stages depends on the nature of the variables, the tool of interpretation and the desired description in the final output. The partitioning algorithm takes account of the computational cost induced by the recursive nature of the procedure and the number of possible partitions at each node of the tree. Further developments of the Two Stage procedure face the computational efficiency problem. In fact, from a computational point of view, the growing procedure is crucial when dealing with very large data sets or when dealing with ensemble methods. At any node t 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 $X$;

- **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 $X$

For classification trees the Global IPR is defined as $\tau$ index of Goodman and Kruskal

$$\tau_t(Y|X) = \frac{\sum_i \sum_j p_t^2(j|i)p_t(i) - \sum_j p_t^2(j)}{1 - \sum_j p_t^2(j)} \tag{1.3}$$

where $p_t(i)$, for $i = 1, \dots, I$, is the proportion of cases in node $t$ that have category $i$ of $X$, and $P_t(j|i)$, for $j = 1, \dots, J$, is the proportion of cases in the node $t$ belonging to class $j$ of $Y$ given the $i^{\text{th}}$ category of $X$. Note that the denominator in equation 1.3 is the Gini diversity index.
For regression trees, Global IPR can be defined as the Pearson's squared correlation $\eta^2$:

$$\eta^2_{Y|X}(t) = \frac{BSS_{Y|X}(t)}{TSS_Y(t)} \tag{1.4}$$

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 $X$.
In a similar way, the Local IPR for both classification and regression trees are defined as in equation 1.3 and 1.4, with the difference that in these cases indexes are computed between the response variable $Y$ and the set of split $s$ generated by the global IPR functions.
More precisely, 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

$$\tau_t(Y|s) = \frac{\sum_j p_l^2(j|tl)p_{tl} + \sum_j p_{tr}^2(j|r)p_{tr} - \sum_j p_t^2(j)}{1 - \sum_j p_t^2(j)} \tag{1.5}$$

whereas for regression trees it is

$$\eta^2_{Y|s}(t) = \frac{BSS_{Y|s}(t)}{TSS_Y(t)} \tag{1.6}$$

Two stage splitting criterion works as follow:

1. select the best predictor $X^*(t)$ at $t$ node by maximising equation 1.3 or 1.4 for classification or regression problems respectively:

2. select the best split $s^*(t)$ at node $t$ by maximizing equation 1.5 or 1.6 for all splits of $X^*(t)$ for classification or regression trees respectively

### 1.2.2 FAST splitting criterion

FAST algorithm (Fast Splitting for Splitting Tree) (Mola and Siciliano, 1997) provides a faster method to find the best split at each node when using CART methodology. As discussed in above section, when applying the two-stage criterion the best predictor could be found minimizing the Global Impurity Proportional Reduction factor due to any predictor $X$, then the Local Impurity Proportional Reduction factor determines the split with respect to all partitions derived from the best predictor.
Main issue of FAST is that the measure of Global IPR measure satisfies the following property:

$$\gamma(Y|X) \geq \gamma(Y|s) \tag{1.7}$$

in which $\gamma$ is the generic Global IPR measure, and $s$ is the set of split generated by $X$ variable.
FAST algorithm consists in two step:

- computing Global IPR measure as in equation 1.3 or 1.4 for all variables belonging to the predictor matrix $X$ and sort in decreasing order these measures;

- computing Local IPR measure as in equation 1.5 or 1.6 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.

The computational cost of FAST algorithm is really lower than the one of CART procedure, with the advantage that the final trees are exactly the same. In the example showed at the end of section 1.2.1 in the table 1.2, there is a set of six predictors, 2 nominal with 6 and 7 categories respectively, 3 ordinal with respectively 3, 5 and 4 categories and one binary variable. It was shown that CART procedure for each variable must examine each possible split to decide which one is the best. Considering the root node, CART technique has to compute (25-1)+(26-1)+2+1+4+3 = 104 splits. FAST algorithm computes at the beginning only six Global IPR measure (in this case there are only six predictors) and then only the local impurity reduction factor until inequality of the second step of the procedure is satisfied. In this small example, the number of computations made is 6+(25-1) = 30 (it is assumed that Global IPR measure relative to the second-best predictor is lower than the local impurity reduction factor obtained by the second one). The computational advantage of using FAST instead of CART is clear: one obtains the same tree-based structure with a great gain in terms of computational cost.

## 1.3 Stopping rules and assignment of the response classes/values to the terminal nodes

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;

- *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.

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. Specifically, node $t$ is assigned to class $j^*$ if the highest proportions of objects in node $t$ belong to class $j^*$ so that:

$$p(j^*|t) = \max_{j \in C} [p(j|t)]$$

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, so that $\bar{Y}_t$ is assigned to node $t$ where:

$$\bar{y}_t = \frac{1}{n(t)} \sum_{\mathbf{x}_n \in t} y_{i_t}$$

## 1.4 Pruning

Exploratory trees can be used to investigate the structure of data but they cannot be used in a straightforward way for induction purposes. For inductive purposes the question is: how large should be the tree? 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. To choose the "honest" tree in terms of its size, Breiman *et al.* (Breiman et al., 1984) defined the *minimal cost-complexity pruning.* Before proceeding with pruning description, the definition of an error measure of a tree structure is necessary.

- For classification trees, the error at the generic 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 $t^{\text{th}}$ 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 $t^{\text{th}}$ terminal node.

- For regression trees the error rate is defined exactly as in equation 1.1, that is as the sum of TSS in the $t^{\text{th}}$ 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.

Pruning procedure works as follow: 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

$$R_\alpha(T) = R(T) + \alpha \left|\tilde{T}\right| \tag{1.8}$$

where $\alpha$ is a non negative complexity parameter which "governs the tradeoff between tree size and its goodness of fit to the data" (Hastie et al., 2019; James et al., 2013).
The idea is, for each $\alpha$, 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 $\alpha$ increases the more the size of the tree decreases.
The pruning procedure is the same for both classification and regression cases, so the attention can be focused on the classification problem without loss of generality. 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 $\left|\tilde{T}\right|$. 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 $\alpha$ increases the two measures tends to became equal, this occurs for a critical value of $\alpha$ that can be found solving the above inequality:

$$\alpha = \frac{R(t) - R(T_t)}{\left|\tilde{T}_t\right| - 1} \tag{1.9}$$

so that $\alpha$ represents for any internal node $t$ the cost due to the removal of any terminal node of the branch.
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. It can be proved (Breiman et al., 1984) that the minimal cost-complexity pruning procedure produces the subtrees with the minimum error rate given the number of its terminal nodes. In other words, if $T_\alpha$ has five terminal nodes, there is no other subtree $T_s \subseteq T_{max}$ having five terminal nodes with smaller error (Breiman et al., 1984, , p.71).
To validate a tree-based structure one has to consider its accuracy: the misclassification ratio or the prediction error. In both classification and regression cases an estimation of the error rate is needed. There are three possible ways to estimate it:

- *Resubstitution estimate*
  Resubstitution estimate is computed by using the same dataset used to build the tree. It is an optimistic estimate, therefore it is not used.

- *Test set estimate*
  If the sample size is sufficiently large, data can be randomly splitted into two sub-samples (training sample and test sample). Then training sample is used to grow the tree-based structure and the test set is used to validate it.

- *Cross validation estimate*
  When sample size is not sufficiently large to be splitted into two sub-samples, one can use the cross-validation estimate. Data set is splitted into $V$ sub-samples approximately of the same size, then $V$ trees are built using the $V^{\mathrm{th}}$ sub-sample as test set and the other $V-1$ as training set. By averaging over the $V$ test set estimates, finally the cross-validation estimate of the error rate is achieved.

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 $(1-SErule)$. Denoting by $R^{ts}(T)$ the test set error estimate associated with a generic tree $T$ in the sequence $\Omega$, according to the $0-SE$ rule the tree $T^*$ will be selected if:

$$\mathrm{R^{ts}(T*)} = \min_{T \in S} \mathrm{R^{ts}(T)}$$

whereas, if $1-SE$ rule is employed tree $T^{**}$ will be selected if:

$$\mathrm{R^{ts}(T^{**})} \leq \left[\mathrm{R^{ts}(T^{*})} \pm \mathrm{SE}(\mathrm{R^{ts}(T^{*})})\right]$$

# Chapter 2

# Ensemble Methods

Ensemble methods are learning algorithm that construct a set of classifiers and then classify new data points by taking a vote of their predictions (Dietterich, 2000). 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).
There are several methods for constructing ensemble (Dietterich, 2000) (by enumerating the hypotheses or bayesian voting, by manipulating input features, by manipulating output targets), but the most popular ensemble methods work by manipulating the training examples.
These methods manipulate training examples through a re-sampling technique to generate multiple classifiers, then a learning algorithm is run several times with a different subset of training examples, as it can be seen in figure 2.1. The most famous ensembles belonging to this category are Bagging, Boosting and Random Forests.

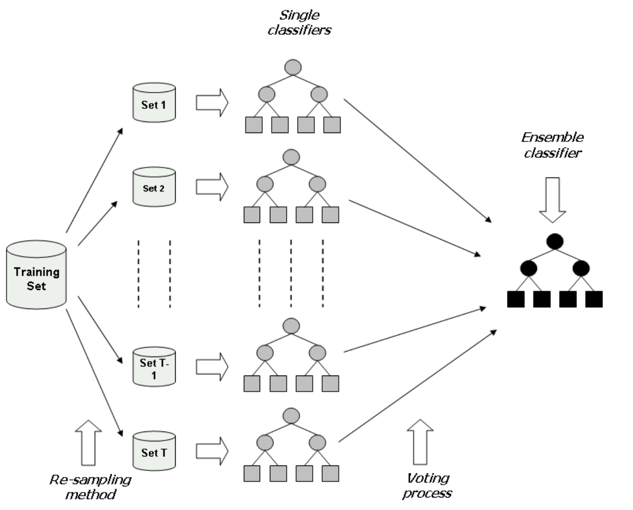

Figure 2.1: Ensemble methods working by manipulating training examples

## 2.1 Bagging

Bagging (Breiman, 1996) is an acronym for Bootstrap Aggregating: it forms a set of classifiers that are combined by voting by generating replicated bootstrap (Efron, 1979; Tibshirani and Efron, 1993) samples of the data. Table 2.1 shows the pseudo-code of Bagging algorithm.
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.

---

Let $\mathcal{L} = (x_i, y_i), \dots, (x_n, y_n)$ be a training sample, where $x_i \in X$ and $y_i \in R$ if numerical or $y_i \in \{1, \dots, J\}$ 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
  $\mathcal{H}(X) = \begin{cases} \text{aggregation by voting if } y_i \in \{1, \dots, J\} \\ {}_{av}h\left(x, \mathcal{L}^B\right) \text{ if } y_i \in R \end{cases}$
```

---

Table 2.1: Bagging algorithm

Bagging works well for unstable procedures. Both classification and regression methods are unstable in the sense that small perturbations in their training sets or in construction may result in large changes in the constructed predictor (Breiman, 1998). 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 (Tibshirani, 1996; Wolpert, 1997; Friedman, 1997). So, Bagging is a method of variance reduction (Friedman and Hally, 1999) and 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 (Hastie et al., 2019). As Breiman says (Breiman, 1996): *Bagging unstable classifiers usually* *improve them. Bagging stable classifiers is not a good idea.*

## 2.2 Boosting algorithms

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 recall the first boosting algorithms, whose 'philosophy' survives in all the more recent variants of the algorithm (gradient boosting, stochastic gradient boosting, logitboost, etc.) The main difference between Bagging and Boosting algorithms is that whereas Bagging uses the bootstrap as resampling method (that is, the probability of each individual to be included in the bootstrap training sample through the iterations is constant and equal to $1/n$, where $n$ indicates the sample size), 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. More precisely, starting from a uniform distribution of weights, these are increased for the $i^{\text{th}}$ 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.*
Therefore a weak learner is a supervised learning algorithm which returns a classification just slightly better than random choice, for example in the case of binary classification problems it must give back an error rate smaller than 50%.
Boosting has its roots in a theoretical framework for studying machine learning called the "PAC" learning model (Valiant, 1984). In brief, this theory states that a learning machine which is wrongly trained returns

an incorrect prediction even if it is trained a lot of time, but with high probability a well trained learning machine will solve the classification problem after a certain number of tests. In other words, the machine has to be *Probably Approximately Correct*.
Several boosting algorithms are developed in the last years (**?**), such as polynomial-time boosting algorithm (**?**) and boosting-by-majority algorithm (Freund, 1995), but doubtless the most famous boosting algorithm is AdaBoost developed by Freund and Schapire in 1995 (Freund and Schapire, 1997).

## 2.3 AdaBoost algorithms for classification and regression problems

Table 2.2 shows the pseudo-code of AdaBoost algorithm for binary classification problems. The algorithm takes as input a training set $\mathcal{L} = (x_i, y_i), \ldots, (x_n, y_n)$ in which $y_i = \{-1, +1\}$ and it calls a given weak learning algorithm repeatedly in a series of rounds $t, \ldots, T$. Main idea of the algorithm is to maintain a distribution of weights over $\mathcal{L}$. These weight are updated at each iteration $t$ according to the weighted error occurred by the weak learner in the last iteration. Weak learner has to define a *weak hypothesis* $h_t : X \to \{-1, +1\}$ by which it is possible to compute the error $\epsilon_t = Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right]$ (note that the error is computed over the distribution $D$ on which the weak learner is trained). Subsequently the algorithm chooses an $\alpha$ parameter which indicates the importance of the weak hypothesis $h_t$ to update the distribution $D$. The final boosted classifier, or *strong learner*, is the weighted majority vote of the $T$ weak hypotheses weighted by $\alpha_t$. AdaBoost is the acronym of Adaptive Boosting because it adapts to the error rates of the individual weak hypotheses. The most basic theoretical property of AdaBoost concerns its ability to reduce the training

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 $h_t : X \to \{-1, +1\}$
  - compute the error $\epsilon_t = Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right]$
  - 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:

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

Table 2.2: AdaBoost algorithm for binary response variable

error. It can be proved (Freund and Schapire, 1997; Meir and Rätsch,

2003) that the training error of the final hypothesis is at most

$$\frac{1}{n}\sum_{i=1}^{n}[H(x_i) \neq y_i] \leq \frac{1}{n}\sum_{i=1}^{n}\exp(-y_i\alpha_t h_t(x_i)) = \prod_{t=1}^{T} Z_t$$

By minimizing $Z_t$ this limit error can be minimized, and this can be obtained by choosing the suitable $\alpha$ parameter. The expression $Z_t = \sum_{i=1}^{n} D_t(i)\exp\left(-\alpha_t y_i h_t(x_i)\right)$ can be write as

$$Z_t = \sum_{i=1}^{n} D_t(i)\exp\left(-\alpha_t u_i\right) \tag{2.1}$$

where $u_i = y_i h_t(x_i) < 0$ if $y_i \neq h_t(x_i)$ and $u_i = y_i h_t(x_i) > 0$ if $y_i = h_t(x_i)$. If $Y \in \{-1, +1\}$, it follows that

$$\begin{aligned} Z_t &= \sum_{i=1}^{n} D_t(i)\exp\left(-\alpha_t u_i\right) \leq \\ &\leq \sum_{i=1}^{n} D_t(i)\left(\frac{1+u_i}{2}\exp(-\alpha_t u_i) + \frac{1-u_i}{2}\exp(\alpha_t u_i)\right) \end{aligned}$$

Recall that $\epsilon_t$ is the training error at $t^{th}$ iteration, it can be indicated as $\epsilon_t = \frac{1-u_i}{2}$ and $(1-\epsilon_t) = 1 - \left(\frac{1-u_i}{2}\right) = \frac{1+u_i}{2}$. Equation 2.1 becomes

$$\begin{aligned} Z_t = \sum_{i=1}^{n} D_t(i) \exp\left(-\alpha_t u_i\right) \leq \\ \leq \sum_{i=1}^{n} D_t(i) \left((1-\varepsilon_t)\exp(-\alpha_t u_i) + \varepsilon_t \exp(\alpha_t u_i)\right) \end{aligned} \tag{2.2}$$

The last part of equation 2.2 can be wrote as

$$(1-\varepsilon_t)\exp(-\alpha_t) + \varepsilon_t \exp(\alpha_t) \tag{2.3}$$

so, to minimize $Z_t$ one has to minimize expression 2.3 with respect to $\alpha$ and compute this parameter in that point, namely

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$$

By substituting this parameter in the expression 2.2 and by reducing, obtain

$$\prod_{t=1}^{T} Z_t = \prod_{t=1}^{T} \left[\sqrt{4\varepsilon_t(1-\varepsilon_t)}\right] = \prod_{t=1}^{T} \left[2\sqrt{\varepsilon_t(1-\varepsilon_t)}\right] \tag{2.4}$$

If weak learner works better than random choice, $\epsilon_t = 0.5 - \gamma_t$ where $\gamma_t$ is some positive parameter, and then $\gamma_t = 0.5 - \epsilon_t$. Equation 2.4 can be re-wrote as

$$\prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq \prod_{t=1}^{T} \exp\left(-2\gamma_t^2\right) = \exp\left(-2\sum_{t=1}^{T} \gamma_t^2\right) \tag{2.5}$$

in which the last term is the upper limit over training error of boosted classifier. Freund and Schapire (Freund and Schapire, 1997) showed how to bound the generalization error of the final hypothesis in terms of its training error, the sample size $n$ and the VC-dimension $d$ of the weak hypothesis space (which is a standard measure of the complexity of a space of hypotheses (Meir and Rätsch, 2003)). They proved that the upper bound of the generalization error, which can be interpreted as the expected value of the test error (Hastie et al., 2019), is at most (Bartlett et al., 1998)

$$P\left[\mathrm{margin}_f(x,y) \leq \theta\right] + \tilde{O}\left(\sqrt{\frac{d}{n\theta^2}}\right)$$

for any $\theta > 0$, in which margin is a number in $[0,1]$ which is an indicator of the "confidence" of the classification.
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. Table 2.3 shows the pseudo-code of AdaBoost algorithm for multiclass classification problems. In a multiclass problem the condition that the error rate of the base classifier is less than 0.5 can be too restrictive. For this reason Freund and Schapire (Freund and Schapire, 1997) introduced a pseudo-loss function of a confidence-rated classifier to be minimized in the boosting iterations instead of the error rate. The code in table 2.3 is the modified version of AdaBoost.M algorithm (Freund and Schapire, 1997) as made by Eibl and Pfeiffer (Eibl and Pfeiffer, 2002) and called AdaBoost.M1W.
Table 2.4 shows the AdaBoost code for regression problems. In this case, the error can be defined as a squared loss function, even if other loss functions, such as linear or exponential, could be used. In the regression case the final boosted classifier is obtained, in general, by averaging the single weak hypotheses through the iterations.
Code showed in the table 2.4 is the modification of AdaBoostR algorithm (Freund and Schapire, 1997) made by Drucker (Drucker, 1997;

```
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 hypothesis $h_t : X \to \{1, \ldots, J\}$
      – 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:
            $D_{t+1}(i) = \frac{D_t(i) e^{-\alpha_t I(h_t(x_i) = y_i)}}{Z_t}$
        where $Z_t$ is a normalization factor
  • Output: final boosted classifier:
        $\mathcal{H}(x) = arg\max_{y \in J}\left(\sum_{t=1}^{T} \alpha_t I\left(h_t(x) = y\right)\right)$
```

Table 2.3: AdaBoost algorithm for multiclass classifiers

Gey and Poggi, 2006). The aggregation process in this case is the weighted median.

Boosting is a method of bias reduction (Hastie et al., 2019), therefore it can be used with stable classifiers (e.g. with a stump), but it works really good also in reducing variance of a classifier. Sometimes it can produce overfitting phenomenon, but it can occur when weak learner has a too high error rate or when the boosted training error reaches to zero too fast (Freund and Schapire, 1997; Meir and Rätsch, 2003; Bartlett et al., 1998).

In general, ensemble methods allow to gain in prediction accuracy. When these are used in combination with tree-based models, if the goal is predicting as more as possible new observations, ensembles can help us to it achieve. If our goal is interpreting relationships among covariates, we could never use ensembles. As Breiman says (Breiman, 1996), *what one loses, with the trees, is a simple and interpretable structure. What one gains is increased accuracy.*

## 2.4 Random Forests

The Random Forests (RF, Breiman, 2001) constitute an improvement over Bagging. The algorithm is quite simple: it provides $T$ bootstrap replications (over the statistical units), and, supposing there are $M$ input variables, a random specification of $m < M$ predictors at each node, searching the best split on these $m$ predictors. The value of $m$ is held constant during the forest growing. The forest error rate depends on:

- the correlation between any two trees in the forest. Increasing the correlation increases the forest error rate;
- the strength of each individual tree in the forest. A tree with a low error rate is a strong classifier. Increasing the strength of the individual trees decreases the forest error rate.

Reducing $m$ reduces both the correlation and the strength. Increasing it increases both. Somewhere in between is an "optimal" range of

$m$. As a rule of thumb, one can set $m \approx \sqrt(M)$. This is the only adjustable parameter to which random forests is somewhat sensitive. In principle, RF do not need for a separate test set to get an unbiased estimate of the test set error. It can be internally estimated through the Out-Of-Bag (OOB) error estimate during the run:

- For each iteration $t$, $t = 1, \ldots, T$, each tree is built using a different bootstrap sample from the original data. About one-third of the cases are left out of the bootstrap sample and not used in the construction of the $t$th tree. Call these observations *out-of-bag* observations.

- Put each OOB case down the $t$th tree to get a classification. In this way, a test set classification is obtained for each case in about one-third of the trees.

- At the end of the run, take $j$ to be the class that got most of the votes every time the same case was OOB. The proportion of times $j$ is not equal to the true class is the OOB error estimate.

The OOB error during the iterations can be evaluated also as a stabilizer for Random Forests. When the OOB error is stable, then the training phase of Random Forests can be considered finished.
RF also provides the variable importance expressed as either prediction accuracy or Gini importance. In the first case, the OOB cases are used first to measure the prediction accuracy (i.e., misclassification error). Then, the values of the variable in the OOB sample are randomly shuffled, keeping all other variables the same. Finally, the decrease in prediction accuracy on the shuffled data is measured.
The Gini importance is instead computed by adding up the Gini decrease in impurity for each individual variable over all the $T$ trees in the forest.
Note that for numerical outcomes the measures are computed by using

the right quantities (i.e., mean squared error for prediction accuracy and reduction in sum of squared errors for the ‘Gini’ importance).

Let $\mathcal{L} = (x_i, y_i), \dots, (x_n, y_n)$ 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 $\epsilon_{D_t} = \displaystyle\sum_{i=1}^{n} D_t(i) L_t(i)$
  - set $\beta_t = \dfrac{\epsilon_{D_t}}{\max\limits_{1 \le i \le n} L_t(i) - \epsilon_{D_t}}$
  - set $w_k(i) = \dfrac{L_t(i)}{\max\limits_{1 \le i \le n} L_t(i)}$
  - 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:

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

Table 2.4: AdaBoost algorithm for regression problems

# Bibliography

Bartlett, P., Freund, Y., Lee, W. S., and Schapire, R. E. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686.

Breiman, L. (1996). Bagging predictors. *Machine learning*, 24:123–140.

Breiman, L. (1998). Arcing classifier (with discussion and a rejoinder by the author). *The annals of statistics*, 26(3):801–849.

Breiman, L. (2001). Random forests. *Machine learning*, 45:5–32.

Breiman, L., Friedman, J., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. CRC press.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Multiple Classifier Systems: First International Workshop, MCS 2000 Cagliari, Italy, June 21–23, 2000 Proceedings 1*, pages 1–15. Springer.

Drucker, H. (1997). Improving regressors using boosting techniques. In *Proceedings of the 14th International Conference on Machine Learning*, volume 97, pages 107–115.

Efron, B. (1979). Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26.

Eibl, G. and Pfeiffer, K. P. (2002). How to make adaboost. m1 work for weak base classifiers by changing only one line of the code. In *Machine Learning: ECML 2002: 13th European Conference on Machine Learning Helsinki, Finland, August 19–23, 2002 Proceedings 13*, pages 72–83. Springer.

Freund, Y. (1995). Boosting a weak learning algorithm by majority. *Information and computation*, 121(2):256–285.

Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139.

Friedman, J. H. (1997). On bias, variance, 01–loss, and the curse-of-dimensionality. *Data mining and knowledge discovery*, 1:55–77.

Friedman, J. H. and Hally, P. (1999). On bagging and nonlinear estimation. Technical report, Technical report: Stanford University.

Gey, S. and Poggi, J.-M. (2006). Boosting and instability for regression trees. *Computational statistics & data analysis*, 50(2):533–550.

Hastie, T., Tibshirani, R., Friedman, J. H., and Friedman, J. H. (2019). *The elements of statistical learning*. Springer.

James, G., Witten, D., Hastie, T., and Tibshirani, R. (2013). *An introduction to statistical learning*, volume 112. Springer.

Meir, R. and Rätsch, G. (2003). An introduction to boosting and leveraging. In *Advanced Lectures on Machine Learning: Machine Learning Summer School 2002 Canberra, Australia, February 11–22, 2002 Revised Lectures*, pages 118–183. Springer.

Mola, F. and Siciliano, R. (1992). A two-stage predictive splitting algorithm in binary segmentation. In Dodge, Y. and Whittakerr, J., editors, *Computational Statistics: COMPSTAT 92*, pages 179–184. Physica Verlag, Heidelberg.

Mola, F. and Siciliano, R. (1994). Alternative strategies and catanova testing in two-stage binary segmentation. In Diday, E., Lechevallier, Y., Schader, M., Bertrand, P., and Burtschy, B., editors, *New Approaches in Classification and Data Analysis: Proceedings of IFCS 93*, pages 316–323. Springer Verlag, Heidelberg.

Mola, F. and Siciliano, R. (1997). A fast splitting procedure for classification trees. *Statistics and Computing*, 7:209–216.

Tibshirani, R. (1996). Bias, variance and prediction error for classification rules. Technical report, Technical report: University of Toronto.

Tibshirani, R. J. and Efron, B. (1993). An introduction to the bootstrap. *Monographs on statistics and applied probability*, 57(1).

Valiant, L. G. (1984). A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142.

Wolpert, D. H. (1997). On bias plus variance. *Neural Computation*, 9(6):1211–1243.