TRABAJO FIN DE MÁSTER

# Aprendizaje Supervisado mediante Random Forests

*Presented by:*
**María Cristina Molero del Río**

*Supervisors:*
DR. RAFAEL BLANQUERO BRAVO
DR. EMILIO CARRIZOSA PRIEGO

FACULTAD DE MATEMÁTICAS
Departamento de Estadística e Investigación Operativa
Sevilla, Junio 2017

# Contents

**Resumen** **5**

**Introduction** **7**

**1 Supervised Classification** **9**
1.1 Scoring functions . . . 9
1.2 Validation techniques and performance criteria . . . 10
1.3 Supervised regression . . . 13

**2 Classification trees** **15**
2.1 Getting familiar with classification trees . . . 16
2.2 Topology and type of splitting . . . 18
2.3 Splitting criteria . . . 19
2.3.1 Impurity functions . . . 22
2.3.1.1 Classification error rate . . . 22
2.3.1.2 Gini index . . . 22
2.3.1.3 Cross-entropy . . . 22
2.3.2 Gain ratio . . . 23
2.4 Stopping criteria . . . 24
2.5 Labeling terminal nodes . . . 25
2.6 Classification tree step by step . . . 26
2.7 Tree Pruning . . . 31
2.8 Regression trees . . . 32

**3 Ensembling trees: Random Forests** **33**
3.1 Random forests. Influences and definition. . . . 33
3.1.1 Background . . . 33
3.1.2 Definition . . . 34
3.1.2.1 Parameters tuning . . . 35
3.2 Properties of Random Forests . . . 37
3.2.1 Variable importance . . . 37
3.2.1.1 Mean Decrease Accuracy . . . 37

3.2.1.2 Mean Decrease Impurity . . . . . . . . . . . . . . . 39
3.2.2 Proximity measure . . . . . . . . . . . . . . . . . . . . . . . . 40
3.2.2.1 Data visualization . . . . . . . . . . . . . . . . . . . 41
3.2.2.2 Outliers detection . . . . . . . . . . . . . . . . . . . 42
3.2.2.3 Missing values imputation . . . . . . . . . . . . . . . 42
3.2.2.4 Prototypes search . . . . . . . . . . . . . . . . . . . 43
3.3 Regression Random Forests . . . . . . . . . . . . . . . . . . . . . . . . 44

**4 Random Forests in R** **45**
4.1 Random Forests and properties . . . . . . . . . . . . . . . . . . . . . . 46
4.2 Feature Selection based on Random Forests . . . . . . . . . . . . . . . 52

**Bibliography** **60**

# Resumen

Muchos problemas de la vida real pueden modelarse como problemas de clasificación, tales como la detección temprana de enfermedades o la concesión de crédito a un cierto individuo. La Clasificación Supervisada [9] se encarga de este tipo de problemas: aprende de una muestra con el objetivo final de inferir observaciones futuras. Hoy en día, existe una amplia gama de técnicas de Clasificación Supervisada. En este trabajo nos centramos en los bosques aleatorios (Random Forests, [4]).

El Random Forests es una técnica de clasificación que consiste en construir una colección de árboles de decisión individuales [8] sobre los cuales se aplica aleatoriedad de cierta manera. Es conocido que esta técnica proporciona un buen rendimiento, incluso cuando trata con problemas de gran escala como los que se tienen en la actualidad. Sin embargo, existe una pequeña brecha entre la teoría relacionada con esta técnica y la experiencia empírica de la misma. El Random Forests también es útil en otros campos del Aprendizaje Automático: da medidas de importancia de las variables, que podrían utilizarse en la Selección de Atributos, y una matriz de proximidades entre las observaciones, lo que permite al analista detectar valores atípicos, reemplazar valores perdidos, buscar prototipos y obtener una visualización comprensible de los datos. Estas últimas propiedades hacen que el Random Forests sea una técnica aún más atractiva.

En este trabajo se hace, en primer lugar, una breve descripción de la Clasificación Supervisada, incluyendo las principales técnicas de validación y los criterios de rendimiento más relevantes. En segundo lugar, se explica en detalle la construcción de un árbol de clasificación. Seguidamente, se presenta el Random Forests y se revisan las propiedades principales del mismo. Por último, se muestran resultados experimentales en `R`.

# Introduction

Many problems in the real life can be modelled as classification problems: the early detection of diseases or the granting of credit to a certain individual, among others. Supervised Classification [9] handles this issue by learning from a sample in order to infer forthcoming observations. Nowadays, there exist a wide range of Supervised Classification techniques. Along this work, we will focus on Random Forests classification method [4].

Random forests is a collection of individual decision trees [8] on which randomness is applied somehow. This classification technique is well-known for providing great performance, even with large-scale problems. Nevertheless, there is a little gap between theory and empirical experience in this scheme. Random Forests are also useful for other fields in Machine Learning: they give measures of variables importance, which could be used in Feature Selection, and proximities between observations, which allows the analyst to detect outliers, replace missing values, search prototypes and obtain a comprehensive visualization of the data. These latter properties make Random Forests even more attractive.

This work is organised as follows. Chapter 1 introduces to the Supervised Classification, including the most relevant validation techniques and performace criteria. Next, in Chapter 2, the construction of a classification decision tree is addressed in detail. Then, Chapter 3 is enterely devoted to Random Forests. The main properties of Random Forests are reviewed. Last, in Chapter 4, computational experience with Random Forests is reported. The code in `R` software is also displayed.

# Chapter 1

# Supervised Classification

Supervised Learning is one of the most relevant tasks in Machine Learning and Data Mining. The general idea in Supervised Learning is to infer a function which is known only for some examples, with the final goal of mapping new forthcoming examples. Depending on the range of the inferred function, one can distinguish between Supervised Classification and Supervised Regression. Along this text we will mainly focus on Supervised Classification and briefly survey the regression case.

The aim of Supervised Classification is to seek procedures for classifying objects in a set $\Omega$ into a finite set $C$ of nominal values or classes. Each object $u$ in $\Omega$ has associated a pair $(\boldsymbol{x}^u, y^u)$, where $\boldsymbol{x}^u$, the predictor vector, takes values on a set $X$, usually assumed to be a subset of $\mathbb{R}^p$, and $y^u \in C$ is the class membership of the object. Henceforce, each component of the predictor vector and $y$ will be named predictor and response variables, respectively.

The whole information about all the objects in $\Omega$ is not available as a rule. Instead, we assume we are given a sample $D_n = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ of independent random variables distributed as the independent prototype pair $(X, C)$. The goal is to use the data set $D_n$ to construct a classifier, i.e., an estimate $m_n : X \longrightarrow C$ of the function $m(\boldsymbol{x})$, which gives to each $\boldsymbol{x}$ the class minimizing the misclassification cost.

## 1.1 Scoring functions

Actually, classifiers are based on scoring functions $f_c : X \rightarrow \mathbb{R}$ built for each class $c \in C$. These functions are in charge of ranking a forthcoming object: they indicate, in a certain sense, the likelihood that an object represented by $\boldsymbol{x}$ belongs to each class. In this way, the classifier will be given by the function

$$m_n(\boldsymbol{x}) \in \arg\max_{c \in C} f_c(\boldsymbol{x}) \ \forall \boldsymbol{x} \in X. \tag{1.1}$$

These scoring functions have a particular property: if every $f_c$ is replaced by $f_c + h$ for a common $h$, the same classifier $m_n$ is obtained. This property is interesting in

binary classification problems since one of the scoring functions could be set equal to zero and the classifier will only depend on the sign of one class, i.e., suppose the variable response $Y$ takes values in $C = \{\text{Positive}, \text{Negative}\}$, setting $f_{\text{Negative}} = 0$, the classifier (1.1) takes the form

$$m_n(\boldsymbol{x}) = \begin{cases} \text{Positive} & \text{if } f_{\text{Positive}}(\boldsymbol{x}) \geq 0 \\ \text{Negative} & \text{otherwise} \end{cases} . \tag{1.2}$$

## 1.2 Validation techniques and performance criteria

A large body of the literature has been devoted to develop different classification methods. Although in this text we are interested in Random Forests [4], which are widely used, many other classification methods have also been proposed: Support Vector Machines [11], Naïve Bayes [32], K-Nearest Neighbour [12], Neural Networks [14] and Logistic Regression [22], among others. They all essentially differ in the statistical assumptions made of the data and the type of algorithms needed to construct the classifier.

There is no classification technique that always outperforms the rest. In consequence, it is necessary to compare some of them and choose the best possible one. This is not an easy task, since several performance criteria, which will be discussed below, could be considered.

### Validation techniques

There exist different techniques for evaluating the power of prediction of each classifier, named as *validation techniques*.

The simplest technique is to split the sample $D_n$ into two disjoint sets: the *training set*, the subset used for constructing the classifier, and the *test set*, the subset used for estimating performance. These subsets are usually taken to contain two-thirds and one-third of the entire sample, respectively. In this technique, known as *hold-out* method, the classifier's evaluation depends largely on how the division of the data is made.

An improvement of the hold-out method is the well-known *$k$-folds cross-validation*. In $k$-folds cross-validation, the sample is divided into $k$ subsets of similar size. Each subset is used to test how the classifier constructed from the rest of the sample behaves. Hence there are $k$ classifiers and $k$ estimates of the performance measures. Averaging the $k$ estimates, one obtains the $k$-folds cross-validation estimate. The choice of the value of $k$ varies according to the sample size though experts generally take $k = 10$. When the test subsets are formed by one single instance, the technique is called *leave-one-out validation*.

In some cases, the sample size is not very large and validation techniques based on the idea of resampling are frequently used. A *bootstrap* [17] is a sample of the

sample, that is, a set of observations extracted from the original sample, with replacement and of the same size. The observations not appearing in the bootstrap sample are called *out-of-bag (OOB)* observations. Similarly to the procedure described for cross-validation, bootstrap aggregating [3], or *bagging*, is a methodology that generates multiple bootstrap samples and the classifier is trained with each one and then every classifier is tested using the same test set: the original sample. Averaging again, an estimate of the performance measure is obtained, $e_{boot}$. The issue is that there are observations in common between the training sets and the test set; thus, $e_{boot}$ will have a considerable optimistic bias. It is known that the probability that a given observation is part of a bootstrap sample is $1 - \left(1 - \frac{1}{n}\right)^n \underset{n}{\longrightarrow} 1 - e^{-1} \approx 0.632$; in other words, approximately one-third of the sample is the set of OOB observations. In this way, an estimate for each observation in the sample can be made by employing the third part of classifiers in whose construction the particular observation has not taken part. These results are averaged in order to obtain a unique estimate, $e_{OOB}$. A compromise solution is proposed between both estimates, $e_{0.632} = 0.368 e_{OOB} + 0.632 e_{boot}$.

Once the validation techniques have been defined, the most celebrated performance measures, made over the test set, are reviewed in the next subsection.

## Performance criteria

The most popular performance measure is the *accuracy*, defined as the proportion of correctly classified objects. The accuracy gives us an idea of the general behavior of the classifier; however, the classifier usually finds it easier to classify certain classes and therefore it is also of interest to know the proportion of correctly classified objects per class. In the particular case of binary classification problems, $C = \{\text{Positive}, \text{Negative}\}$, these correct classification rates are called *sensitivity* and *specificity*, respectively, and they can be deduced from the contingency table, a double entry table that faces the predictions against the real classes, known as *confusion matrix*, is commonly displayed, see Figure 1.1.

According to the above table,

$$\begin{aligned}
\text{Accuracy} &= \frac{TP + TN}{TP + FN + FP + TN} \\
\text{Sensitivity} &= \frac{TP}{TP + FN} \\
\text{Specificity} &= \frac{TN}{FP + TN}.
\end{aligned}$$

The *F-measure*, the *kappa coefficient* and the so-called *area under the Receiver Operating Characteristic (ROC) curve, AUC*, are prominent for binary classification problems too.

| | | PREDICTED CLASS | |
|---|---|---|---|
| | | POSITIVE CLASS | NEGATIVE CLASS |
| TRUE CLASS | POSITIVE CLASS | TRUE POSITIVE (TP) | FALSE NEGATIVE (FN) |
| | NEGATIVE CLASS | FALSE POSITIVE (FP) | TRUE NEGATIVE (TN) |

Figure 1.1: Confusion matrix.

The F-measure is defined as the weighted harmonic mean of the accuracy and sensitivity, that is,

$$F_\alpha = \frac{1}{\alpha \frac{1}{Accuracy} + (1-\alpha)\frac{1}{Sensitivity}} = \frac{TP}{\alpha\,(FP - FN) + TP + FN},$$

where the weight $\alpha \in [0, 1]$.

The kappa coefficient, $\kappa$, is an index of concordance between the predicted classes and the real observations, but contrary to the accuracy, $\kappa$ takes into consideration the possible concordances due to chance.

$$\kappa = \frac{Accuracy - P_{chance}}{1 - P_{chance}},$$

where

$$P_{chance} = \frac{(TP + FN)\,(TP + FP) + (FP + TN)\,(TN + FN)}{(TP + FN + FP + TN)^2}$$

is the hypothetical probability of concordance by chance. For $\kappa = 1$, the level of concordance is perfect while for $\kappa = 0$, the concordande is due to chance. When $\kappa$ reaches negative values the concordance is less than one would expect by chance.

Last but not least, another performance measure is the AUC, or area under the ROC curve. The ROC curve is generated by replacing (1.2) by the parametric class of classifiers

$$m_{n,\theta}(\boldsymbol{x}) = \begin{cases} \text{Positive} & \text{if } f_{\text{Positive}}(\boldsymbol{x}) \geq \theta \\ \text{Negative} & \text{otherwise} \end{cases},$$

where $\theta$ is the discrimination threshold between classes. For each value of $\theta$, the classifier's sensitivity and specificity are computed and the ROC curve shows the sensitivity

against $(1 - \text{Specificity})$, for different values of the parameter $\theta$. As it can be seen in Figure 1.2, for a fully random classification, the ROC curve corresponds to the segment joining the points $(0, 0)$ and $(1, 1)$. Hence, classification methods better than fully random classification lead to ROC curves above such segment, with a larger area under their ROC curve. In fact, the point $(0, 1)$ represents the ideal classifier, as every instance has been correctly classified. Therefore, the larger the AUC, the better the classifier behaves.

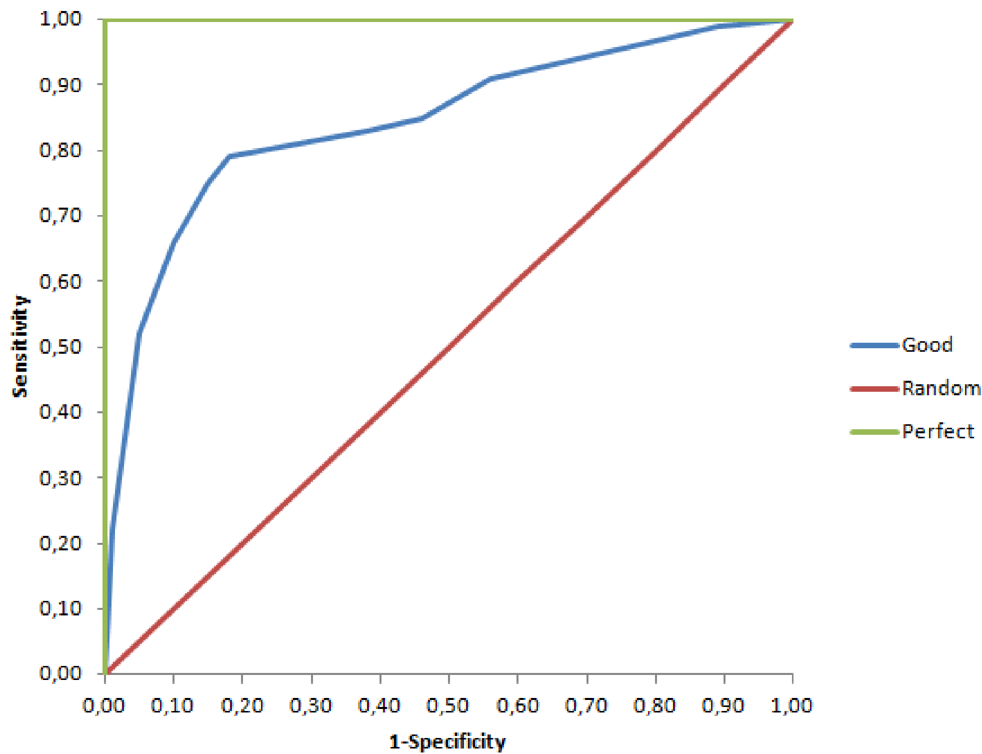



Figure 1.2: ROC curves.

## 1.3 Supervised regression

In supervised regression, the scenario is similar to what was explained for supervised classification, except that it is used to predict a quantitative response rather than a qualitative one. The classifier is now defined as an estimate $m_n : X \longrightarrow \mathbb{R}$ of the function $m(\boldsymbol{x}) = \mathbb{E}\left[C|X = \boldsymbol{x}\right]$, in the sense of least squares.

For regression tasks, the above-described validation techniques do not change and prediction quality is mainly measured by the *Mean Squared Error* (MSE) which consists of the average of the squared errors made in each observation in a test sample

$D^{test}$ of size $|D^{test}|$,

$$MSE(D^{test}) = \frac{1}{|D^{test}|} \sum_{(\boldsymbol{x}_i, y_i) \in D^{test}} \left(y_i - m_n\left(\boldsymbol{x}_i\right)\right)^2.$$

The ideal is to get it as small as possible.

# Chapter 2

# Classification trees

Decision trees constitute a flexible nonparametric tool for predicting. Originally, decision trees were employed for regression, aiming to combat the limitations of multiple regression techniques. The first system used was the AID (Automatic Interaction Detection) [31], developed by Morgan and Sonquist in the sixties, so that trees could be considered a relatively new procedure. It took a decade for a decision tree to be used in classification, leading to the program THAID [30]. Over the years, improvements have been made to the pioneers, with other algorithms such as CLS (Concept Learning System) [23], CHAID (Chisquare-Automatic-Interaction-Detection) [25], CART (Classification and Regression Trees) ([20] [8]), ID3 ([36] [34]), ACLS [33], ASSISTANT [26] [10] and C4.5 [35], but, probably, the outstanding implementations by the time were CART and C4.5.

Decision trees are known to be leaders in terms of interpretability: as we will see in subsequent sections, the tree diagram allows, even non-experts, to interpret the prediction easily. Their popularity is also due to their ability for measuring the importance of predictor variables over the response variable and, therefore, to find out irrelevant features in the model. Moreover, decision trees deal with any type of variables: categorical and continuous. This characteristic differentiates decision trees from other prediction techniques like Neural Networks, among others, in which dummy variables are to be created before building the classifier. In fact, decision trees do not require any data preprocessing like normalizing data or deleting from the study those objects containing some missing value. And as if that were not enough, the speed of construction of a decision tree is considerably quick in comparison with Support Vector Machines, for instance, that comprise solving a huge optimization problem.

Although decision trees are endowed with good properties, summarized in the previous paragraph, empirical evidence shows that they may not be as competitive as other classifiers. For this reason, decision trees are not usually applied on their own. A combination of a large number of them is used in their stead: bagging [3], random forests [4] or boosting [19], to name the most relevant. Although these variants entail a loss of interpretability, a great gain in prediction accuracy is obtained, which makes

tree-based decision making at the same level as other powerful techniques.

In the remainder of this chapter, the goal is to provide an understandable description of the construction of a classification tree, with main focus on the CART model. Finally, we will slightly move on to regression trees.

## 2.1 Getting familiar with classification trees

Prior to formally tackling the construction of a classification tree, some general notions will be provided to readers. The main elements of a tree are presented too. Figure 2.1 depicts an example of a diagram tree for a hypothetical two-class classification problem, $C = \{\text{Positive}, \text{Negative}\}$.

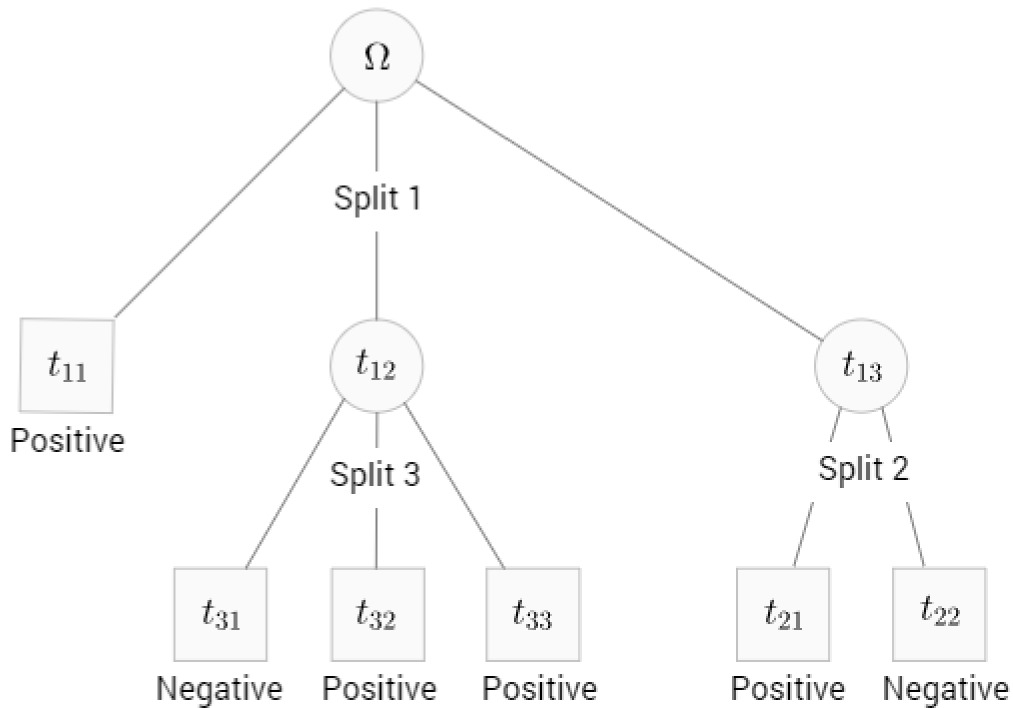



Figure 2.1: A diagram tree.

As it can be seen in Figure 2.1, a classification tree is a classifier that involves the partitioning of $\Omega$, carried out by consecutive and descendant divisions of disjoint subsets of $\Omega$. For instance, $t_{21}$ and $t_{22}$ are disjoint and add up the total previous subset,

$$t_{13} = t_{21} \sqcup t_{22}.$$

Subsets are known as *nodes*. The *root node* is the one that appears the highest, representing $\Omega$ itself. Non-split subsets, indicated by rectangular boxes, are called

*terminal nodes* and they form a partition of $\Omega$,

$$\Omega = t_{11} \sqcup t_{31} \sqcup t_{32} \sqcup t_{33} \sqcup t_{21} \sqcup t_{22}.$$

The rest of nodes are *non-terminal nodes*.

A class label of $C$ is assigned to each terminal node in such a way that there may be more than one terminal node with the same class label. The way to do this assignment is yet to be defined.

Besides nodes, *branches* are another element of a tree. They indicate the different decision options that can be taken in each split. Splits are due to conditions on the variables in $\boldsymbol{x} = (x_1, \ldots, x_p)$. For example, Split 2 into $t_{21}$ and $t_{22}$ could be of the form

$$\begin{aligned} t_{21} &= \{\boldsymbol{x} \in t_{13} : x_2 + x_4 \leq 3\} \\ t_{22} &= \{\boldsymbol{x} \in t_{13} : x_2 + x_4 > 3\} \end{aligned} \tag{2.1}$$

or Split 3 into $t_{31}$, $t_{32}$ and $t_{33}$:

$$\begin{aligned} t_{31} &= \{\boldsymbol{x} \in t_{12} : x_5 = \text{Blue}\} \\ t_{32} &= \{\boldsymbol{x} \in t_{12} : x_5 = \text{Pink}\} \\ t_{33} &= \{\boldsymbol{x} \in t_{12} : x_5 = \text{Grey}\}\,. \end{aligned}$$

In order to predict the class of a given object making use of the classification tree in the Figure 2.1, the procedure is to start by the first split and take the branch containing the condition satisfied by the object. In this way, the object gets to another node from with the same procedure is to be done and repeated until a terminal node is reached. The predicted class for the object will be given by the class label attached to that terminal node.

According to this brief introduction to classification trees, it follows that the growth of a tree depends on four basic ingredients:

- **Topology of the tree and type of splittings**. First of all, the topology of the tree is to be chosen, that is, the number of branches allowed in splits. In most cases, trees are assumed to be binary. The kind of conditions on splitting have to be decided too.

- **Splitting criterion**. At each non-terminal node, not any split works. It is necessary to define a splitting criterion from which the selected split makes prediction accuracy improves among the rest.

- **Stopping criterion**. Deciding when to continue splitting or declare a node terminal is another task that should be taken into account.

- **Classes assignment**. Finally, once the terminal nodes have been located, the last step is to assign a class label to each one of these nodes.

The essence of the problem is, therefore, how to address these issues to obtain an accurate classifier. This will be discussed in the next sections.

## 2.2 Topology and type of splitting

The topology or shape of a decision tree comprises, as advanced before, the determination of how many different branches or options there are when a split is carried out to create new nodes. There exist *binary splitting* and *multi-splitting*.

Let us address separately the cases in agreement with the type of variable under consideration. Recall that working with both, categorical and continuous variables, is feasible in this framework. Let us start with categorical variables.

Suppose we depart from a node $t$. Let $x_m$, $1 \leq m \leq p$, be a categorical variable, taking values, say, in $\{b_1, \ldots, b_Q\}$. For binary splitting, two alternatives are frequently considered. For $i = 1, \ldots, Q$ the first alternative is to consider only as possible splits the following subsets:

$$t_L = \{u \in t : x_m^u = b_i\}$$
$$t_R = \{u \in t : x_m^u \neq b_i\} \, ,$$

denoting $t_L$ and $t_R$ the left and right new subsets. $Q$ possible splits are done in this way. The second alternative is to consider every possible subset of $\{b_1, \ldots, b_Q\}$, that is,

$$t_L = \{u \in t : x_m^u \in S\}$$
$$t_R = \{u \in t : x_m^u \notin S\} \, ,$$

where $S$ ranges over all subsets of $\{b_1, \ldots, b_Q\}$. In this case, since $t_L$ and $t_R$ generate the same subsets with $L$ and $R$ reversed, $2^{Q-1} - 1$ splits are necessary.

For multi-splitting, a new branch per each $i$, $1 \leq i \leq Q$, is created, subdividing the current node into

$$t_i = \{u \in t : x_m^u = b_i\} \, . \tag{2.2}$$

Let $x_m$, $1 \leq m \leq p$, be continuous now, so we have from $D_n$ $n$ real values of $x_m$. Assuming that these values are in order from lowest to highest, at most $n - 1$ different divisions can be done for binary splitting at most, by separating the $l$ first objects, already ordered up to $x_m$, and the remaining $n - l$, with $l = 1, \ldots, n - 1$, i.e.:

$$\begin{array}{rcl} t_L & = & \{u \in t : x_m^u \leq c_l\} \\ t_R & = & \{u \in t : x_m^u > c_l\} \end{array} \tag{2.3}$$

where the cutpoint $c_l$ is the halfway between consecutive data values of $x_m$. However, not every divisions must be considered; those splits in which there is not a change of class are dominated by the others. Anyway, the bigger is $n$, the bigger is the number of potential cutoffs to be taken into consideration, which is time-consuming. In order to reduce the computational burden, continuous variables can be discretized in a previous step. Multi-splitting in continuous variables is not of interest because it has been proved that there is not any advantage in prediction accuracy over binary splitting, [27].

The process explained above is actually applied to *univariate splits*, that is, splits defined by one single predictor variable. Nevertheless, this consideration can be extended to *multivariate splits*, splits in which several predictor variables take part, see (2.1). This extension seems to be plausible in terms of accuracy. Linear splits are the most popular multivariate splits. Aside from heuristic algorithms, they can be performed by using Linear Discriminant Analisis or the linear SVM, to name a couple of them.

The simplest topology of a decision tree is when treating with univariate splits and binary splittings. This method is known as *recursive binary splitting*.

## 2.3 Splitting criteria

The problem of building an optimal binary tree is NP-complete. Due to the complexity of this elementary tree, all trees are constructed by a greedy procedure: at each non-terminal node, all possible splits are generated and the best of them at the current step, according to a splitting criterion, is selected. However, this election could not be optimal for future steps. In this section, the splitting criterion par excellence is described.

Assuming that the set $S$ of possible splits is already computed for a particular non-terminal node, the issue is to decide which of them is the best option in order to improve the classifier accuracy.

First of all, some definitions and notations are needed. Remind the general framework: we are giving a sample $D_n = \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)\}$ with $\boldsymbol{x}_i \in X$ and $y_i \in C = \{C_1, \ldots, C_k\}$, $1 \leq i \leq n$. Denote $n_j$ the number of observations in $D_n$ that belong to the same class $C_j$, for $1 \leq j \leq k$. Prior class probabilities $\pi_j$ can be then estimated from the sample as follows

$$\pi_j = \frac{n_j}{n}.$$

Given a node $t$, let $n_j(t)$ be the number of observations in $t$ that belong to the same class $C_j$, $1 \leq j \leq k$, and $n(t)$ the total numbers of observations that have fallen into node $t$; the proportion of examples belonging to $C_j$ fixed an arbitrary node $t$ is given by

$$\pi_j(t) = \frac{n_j(t)}{n(t)}. \tag{2.4}$$

Then, an estimation of the probability that an example reaches the node $t$ and belongs to class $j$ can be deduced:

$$P(C_j, t) = \pi_j \pi_j(t).$$

On another hand, the marginal probability that an observation reaches the node $t$ is

given by

$$P(t) = \sum_{j=1}^{k} P(C_j, t).$$

The probability that an observation belongs to class $j$ given that it falls into node $t$ is defined by

$$P(C_j|t) = \frac{P(C_j, t)}{P(t)}.$$

When $\pi_j$ are estimated using (2.4), one has

$$P(C_j|t) = \frac{n_j(t)}{n(t)},$$

so these probabilities are the relative proportions of class $j$ in node $t$.

The above definitions will help us to understand the splitting criteria. The way par excellence to select the best split, according to the types introduced in Section 2.2, is to choose the split that best separate the objects in the training sample up to their classes, that is, that produces the maximum reduction in the diversity or *impurity* of objects associated to resultant nodes. To better see this idea, imagine a four-classes binary tree. The proportions of the classes in the initial node are equal, i.e., $P(j|t_0 = \Omega) = 1/4 \ \forall j = 1, \ldots, 4$. A good splitting criterion would be to take the split that leads to two new descendant nodes in which there are only the half of the classes, as shown in Figure 2.2. A quantitative measure of the effectiveness of a split in this sense

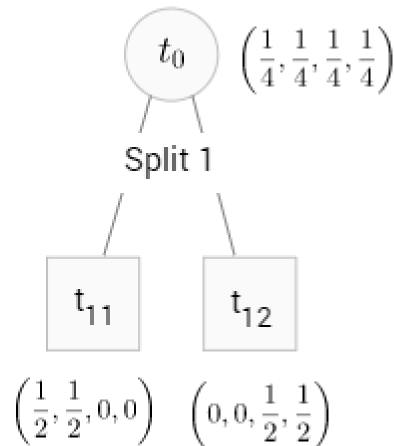



Figure 2.2: First split of a four-classes binary tree. The frequencies of classes that fall into each node appear next.

is obtained by the concept of impurity.

**Definition 2.3.1** (Impurity function). *Let $\Phi : P \longrightarrow \mathbb{R}$ where*

$$P = \left\{ (p_1, \ldots, p_m) : \sum_{j=1}^{m} p_j = 1,\ p_j \geq 0,\ j = 1, \ldots, m \right\}.$$

*$\Phi$ is an impurity function if it verifies*

*(1) $\Phi$ reaches its unique maximum at the point $(\frac{1}{m}, \ldots, \frac{1}{m})$.*

*(2) $\Phi$ achieves its minima exclusively at the points $(1, 0, \ldots, 0)$, $(0, 1, 0, \ldots, 0)$, $\ldots$, $(0, \ldots, 0, 1)$.*

*(3) $\Phi$ is a symmetric function of $p_1, \ldots, p_m$, i.e., if there is a permutation of the variables $p_j$, $\Phi$ will remain constant.*

**Definition 2.3.2** (Impurity measure). *Let $\Phi$ be an impurity function. An impurity measure $i(t)$ of any node $t$ is defined as*

$$i(t) = \Phi\left(P(C_1|t), \ldots, P(C_k|t)\right),$$

*where $P(C_j|t)$ is the estimated probability of class $j$ within node $t$.*

In this way, the impurity measure will achieve its maximum when every classes in node $t$ are in the same proportion, and its minimum when there is only one class in node $t$.

Since the fundamental idea is to produce purer nodes, the selection of the split of a parent node will be done in terms of a new concept: the *information gain*, which measures somehow the purity gained when a node is split into descendant nodes according to a type of splitting. In what follows, this concept is formally defined.

**Definition 2.3.3** (Information gain). *Let $s \in S$ be a possible split, let $r$ be the number of branches considered in the growth of the tree and let $i$ be an impurity function. The information gain of $s$ relative to node $t$ is defined as the average reduction of impurity obtained by splitting the observations within node $t$ up to $s$*

$$G(t, s) = i(t) - \sum_{j=1}^{r} q_j i(t_j), \tag{2.5}$$

*where $t_j$ is each descendant node originated in the splitting and $q_j$ the proportion of observations (within node $t$) that become elements from new node $t_j$.*

By virtue of the above definition, the selected split over the current set of nodes $\Omega_c$ will be the one that maximizes the corresponding information gain:

$$G(t^*, s^*) = \max_{t \in \Omega_c, s \in S} \{G(t, s)\}. \tag{2.6}$$

It is not hard to see from (2.5) that maximizing the information gain when there is one single node $t$ is equivalent to minimizing the term that is substracted in the formula, i.e., minimizing the weighted average impurity of the possible new descendant nodes of $t$.

Therefore, once an impurity function is chosen, the selection criterion is perfectly defined.

### 2.3.1 Impurity functions

Common impurity functions are introduced now, that is, functions that present the desirable properties listed in Definition 2.3.1. They are: the *classification error rate* or misclassification rate, the *Gini index* and the *cross-entropy*.

#### 2.3.1.1 Classification error rate

Given the vector $(p_1, \dots, p_m) \in P$, the classification error rate (CER) is defined as

$$\Phi(p_1, \dots, p_m) = 1 - \max_{1 \le j \le m} \{p_j\} .$$

In the field of decision trees, the elements $p_j$ are assumed to be $P(C_j|t)$ according to Definition 2.3.2, thus, CER would be the fraction of observations in node $t$ that do not belong to the most common class. It seems that CER has the deficiency of not being sensitive enough for the overall tree-growing procedure. For further details, the reader is refered to [8]. The following impurity functions are preferable instead.

#### 2.3.1.2 Gini index

Given the vector $(p_1, \dots, p_m) \in P$, the Gini index is defined as

$$\Phi(p_1, \dots, p_m) = \sum_{j=1}^{m} p_j(1 - p_j) = 1 - \sum_{j=1}^{m} p_j^2.$$

When $p_j = P(C_j|t)$, the Gini index measures the total variance across all the $k$ classes. By the way this index is defined, it takes smaller values if every $p_j$ in node $t$ is close to zero or one. In this sense, node impurity is perfectly measured: a small value will indicate that the node is dominated by a single class.

#### 2.3.1.3 Cross-entropy

Given the vector $(p_1, \dots, p_m) \in P$, the cross-entropy is defined as

$$\Phi(p_1, \dots, p_m) = - \sum_{j=1}^{m} p_j \log_2(p_j)$$

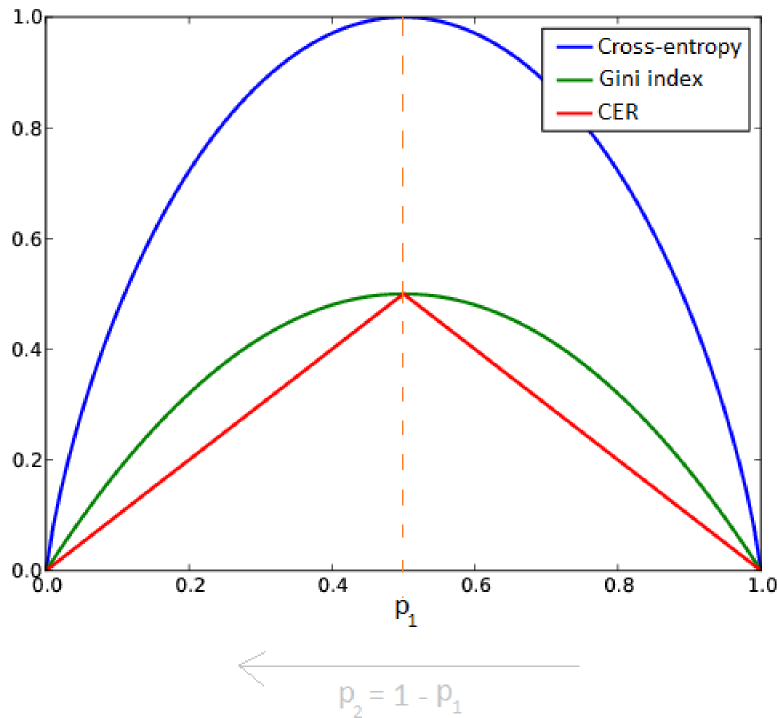



Figure 2.3: Impurity functions for two classes.

with the agreement $0 \log_2 0 = 0$. The negative sign is due to the $p_j's$: in this field, they represent probabilities and it follows that $\log p_j < 0$ when $0 \leq p_j \leq 1$.

The interpretation with $p_j = P(C_j|t)$ is quite similar to the Gini index: if a node is pure, the cross-entropy will take a small value. In fact, the Gini index and the cross-entropy are usually alike numerically.

Actually, the concept of entropy was first introduced in Information Theory. In this way, it can be seen as the minimum number of information bits needed to encode the classification of any object in a particular node. For instance, $\Phi(1, 0, \ldots, 0) = 0$ since the given object belongs to $C_1$ undoubtedly, with no need for any message or any bit to transmit its information.

In Figure 2.3, these three impurity functions are shown for a two-class problem.

### 2.3.2 Gain ratio

In the case of non-binary trees, the information gain is a measure that gives advantage to those variables that have a lot of categories when deciding the splitting. To deal with this problem, alternative measures for the selection of the splitting variable have been proposed. One of them is the *gain ratio*, which penalizes variables with many categories by means of the *information value*.

**Definition 2.3.4** (Information value)**.** *Let $A$ be a variable with $r$ possible categories. Let $m_j(t)$ be the number of examples of category $j$ that fall into node $t$. Then, the information value $Iv(A,t)$ of $A$ into t is defined as*

$$IV\,(A,t) = -\sum_{j=1}^{r} \frac{m_j(t)}{n(t)} \log\left(\frac{m_j(t)}{n(t)}\right). \tag{2.7}$$

The information gain is nothing more than the cross-entropy defined above but, instead of considering the outputs of the classification, the categories of variable $A$ are now adressed. From the information gain, the gain rate is defined.

**Definition 2.3.5** (Gain rate)**.** *Let $t$ be a node and $A$ a particular variable. The gain rate of $A$ in node $t$ is:*

$$GR\,(A,t) = \frac{G(A,t)}{IV(A,t)} \tag{2.8}$$

*where $G$ and $IV$ refer to the information gain and variable, respectively.*

The gain rate presents an issue: its denominator can be near zero if $m_j(t)$ and $n(t)$ are closer to each other for some category. A heuristic rule can be adopted: first, the information gain is computed for each variable candidate to the splitting and then, making use of the same selection rule based on the gain rate (2.6), the splitting is chosen. The only difference is that only those variables whose gain is above average will be considered.

## 2.4 Stopping criteria

In the previous section, the methodology for evaluating the quality of a particular split has been analyzed. The following step is to decide when to declare a node terminal.

The process of growing a tree could continue until every node contains one single observation. In this way, there will be one terminal node per observation in the given sample, so each one will be labelled with the class of its corresponding observation. This proposal can present serious deficiencies in practice, ending up in the great problem of Machine Learning: the *overfitting*. The overfitting is the effect of getting any learning algorithm that has learned too much from the training sample and is not able to generalize and predict new examples, see Figure 2.4. Several criteria that help to avoid overfitting are usually taken.

A first criterion is not to split a node $t$ if the maximum information gain that one would obtain by making the split is lower than some pre-set threshold $\beta$:

$$\max_{s\in S} G\,(s,t) \leq \beta.$$

Although this rule seems to be quite reasonable, it may not provide satisfactory outcomes: if $\beta$ is too small, the resulting tree may be too complex and overfit the training

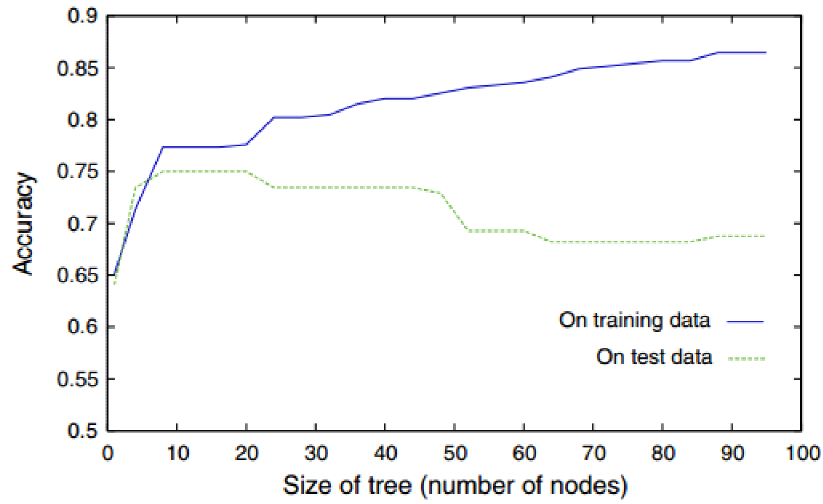



Figure 2.4: Overfitting.

data; if $\beta$ increases, it takes chances of stopping splitting nodes with low maximum information gain, but whose descendant nodes would do suffer splits with a high maximum information gain.

Another rule that is also used is the one based on stopping the subdivision of a node if it does not present a minimum number of training examples. 1, 5 or 10 are typical values.

Finally, a criterion based on statistic tests exists and it is due to [25]. The idea is to continue splitting until the class distribution of the available observations is independent to the variables in the predictor vector. For categorical variables, one can use the $\chi^2$ test; for continuous ones, the $F-$ Student test.

Actually, an alternative of these rules is done in practice. Instead of stopping the growth of the tree, a very large tree is built and properly pruned later so that the branches that explain worse are eliminated. This process of pruning a tree is developed in Section 2.7.

## 2.5 Labeling terminal nodes

Once terminal nodes are declared by means of a stopping criterion, the class labels assignment is still left. Recall from Section 2.1 that the set of terminal nodes $T$ induces a partition of $\Omega$. Let $t \in T$ be a terminal node. The assignment rule is

$$\begin{aligned} A\colon T &\to C \\ t &\mapsto \arg\max_{C_j} n_j(t), \end{aligned} \tag{2.9}$$

that is, depending on the classes of the observations fallen in the given node, the assigned label will be the one corresponding to the most frequent class.

The *size* of the classification tree is defined as the total number of terminal nodes, $|T|$.

**Note 2.5.1.** *The outcomes of a decision tree are invariant to monotone transformations of the input variables.*

## 2.6 Classification tree step by step

At this juncture, after describing theoretically the top-down construction of a classification tree, a simple example is now shown. Firstable, see Algorithm 1 to have a pseudocode with the main steps of the construction of a classification tree.

**Algorithm 1:** Top-down construction of a classification tree.

**Input**: the training set $D_n$, the type of splitting, the number of branches allowed, the splitting criterion and the stopping criterion.

Set $P := \{D_n\}$ the initial list of non-terminal nodes;
Set $P_{\text{final}} := \emptyset$;
**while** $P \neq \emptyset$ **do**
    **for** $t \in P$ **do**
        **if** *in $t$ the stopping criterion is satisfied* **then**
            Delete $t$ from $P$;
            $P_{\text{final}} \leftarrow$ Concatenate$(P_{\text{final}}, t)$;
        **end**
        Compute the information gain for $t$ along every predictor variable, according to the topology of the tree;
    **end**
    Choose the best split, the one whose information gain is maximum;
    Cut the node $t$ according to the best split and call $t_1, \dots, t_r$ the resulting nodes;
    Remove $t$ from $P$;
    $P \leftarrow$ Concatenate$(P, t_1, \dots, t_r)$;
**end**
Label nodes in $P_{\text{final}}$ using definition (2.9);

**Output**: A classification tree by terms of the set of terminal nodes $P_{\text{final}}$.

Now, imagine it is interesting to know the policy employed by a certain bank upon granting a loan, depending on two characteristics of the applicants: their *age* and their *income level*. In this way, suppose this information is got for eleven old applicants, as

| Applicant | Age | Income level | Loan granted |
|---|---|---|---|
| 1 | 22 | Medium | No |
| 2 | 26 | Medium | No |
| 3 | 30 | Medium | Yes |
| 4 | 32 | Low | No |
| 5 | 40 | High | Yes |
| 6 | 45 | Medium | Yes |
| 7 | 60 | High | No |
| 8 | 54 | Medium | Yes |
| 9 | 50 | Low | No |
| 10 | 48 | High | Yes |
| 11 | 20 | High | No |

Table 2.1: Sample.

well as the final decision of the bank, see Table 2.1. Then, there is a continuous variable (age) and a categorical one (income level); and the set of classes is $C = \{\text{Yes}, \text{No}\}$.

First, every element of the classification tree is to be chosen: univariate splits will be considered; for the age, splits will be like in (2.3), and for the income level, multi-splitting is used, (2.2); the impurity measure selected is the cross-entropy; and the stopping criterion will be not to continue splitting a node that has less than 5 observations. Let us start!

The initial node contains the eleven observations. Since there are 6 "No" and 5 "Yes", the cross-entropy for $t_0$ is:

$$i(t_0) = -\left[\frac{6}{11}\log_2\frac{6}{11} + \frac{5}{11}\log_2\frac{5}{11}\right] = 0.99.$$

Now, the information gain is computed for every possible split. Since univariate splits are used, age and income level can be studied separately.

For the age, the first step is to order the different ages and take into account as many splits as changes in classes are. The arrows in Table 2.2 indicate the cutpoints to be taken.

| Applicant | 11 | 1 | 2 | 3 | 4 | 5 | 6 | 10 | 9 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Age | 20 | 22 | 26 | 30 | 32 | 40 | 45 | 48 | 50 | 54 | 60 |
| Loan granted | No | No | No | Yes | No | Yes | Yes | Yes | No | Yes | No |

$\uparrow$ (26|30) $\uparrow$ (30|32) $\uparrow$ (32|40) $\uparrow$ (48|50) $\uparrow$ (50|54) $\uparrow$ (54|60)

Table 2.2: Age. Split 1.

Then, it can be seen in Table 2.3 the information gain.

| Age | | | | | | | |
|---|---|---|---|---|---|---|---|
| Cutpoint | Splits | $q_i$ | $P(\text{No}\|t_i)$ | $P(\text{Yes}\|t_i)$ | $i(t_i)$ | $\sum_j q_j i(t_j)$ | $G(t_0, s)$ |
| 28 | Age $\leq$ 28 | $3/11$ | $3/3$ | $0/3$ | 0 | 0.76 | 0.23 |
| | Age $>$ 28 | $8/11$ | $5/8$ | $3/8$ | 0.95 | | |
| 31 | Age $\leq$ 31 | $4/11$ | $3/4$ | $1/4$ | 0.81 | 0.92 | 0.07 |
| | Age $>$ 31 | $8/11$ | $5/8$ | $3/8$ | 0.98 | | |
| 36 | Age $\leq$ 36 | $5/11$ | $4/5$ | $1/5$ | 0.72 | 0.83 | 0.16 |
| | Age $>$ 36 | $6/11$ | $2/6$ | $4/6$ | 0.92 | | |
| 49 | Age $\leq$ 49 | $8/11$ | $4/8$ | $4/8$ | 1 | 0.98 | 0.01 |
| | Age $>$ 49 | $3/11$ | $2/3$ | $1/3$ | 0.92 | | |
| 52 | Age $\leq$ 52 | $9/11$ | $5/9$ | $4/9$ | 0.99 | 0.99 | 0 |
| | Age $>$ 52 | $2/11$ | $1/2$ | $1/2$ | 1 | | |
| 56 | Age $\leq$ 56 | $10/11$ | $5/10$ | $5/10$ | 1 | 0.91 | 0.08 |
| | Age $>$ 56 | $1/11$ | $1/1$ | $0/1$ | 0 | | |

Table 2.3: Information gain. Age. Split 1.

For the income level, see Tables 2.4 and 2.5.

| Income level | Low | | Medium | | | | | High | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Applicant | 4 | 9 | 1 | 2 | 3 | 6 | 8 | 5 | 7 | 10 | 11 |
| Loan granted | No | No | No | No | Yes | Yes | Yes | Yes | No | Yes | No |

Table 2.4: Income level. Split 1.

| Income level | | | | | | |
|---|---|---|---|---|---|---|
| Splits | $q_i$ | $P(\text{No}\|t_i)$ | $P(\text{Yes}\|t_i)$ | $i(t_i)$ | $\sum_j q_j i(t_j)$ | $G(t_0, s)$ |
| Income level = Low | $2/11$ | $2/2$ | $0/2$ | 0 | 0.80 | 0.17 |
| Income level = Medium | $5/11$ | $2/5$ | $3/5$ | 0.97 | | |
| Income level = High | $4/11$ | $2/4$ | $2/4$ | 1 | | |

Table 2.5: Information gain. Income level.

Regarding Tables 2.3 and 2.5, one can see that the first split to be done is "Age $\leq$ 28" and "Age $<$ 28". In this sense, the variable age is the one that best explains the response variable, i.e., the one that best classifies loans applicants. See Figure 2.5. The resulting nodes are $t_{11}$ and $t_{12}$. The node $t_{11}$ is already represented with a squared box, indicating that the stopping criteria has been satisfied and, also, the class label has been assigned regarding the most ocurring class in that node. The next step will be to split

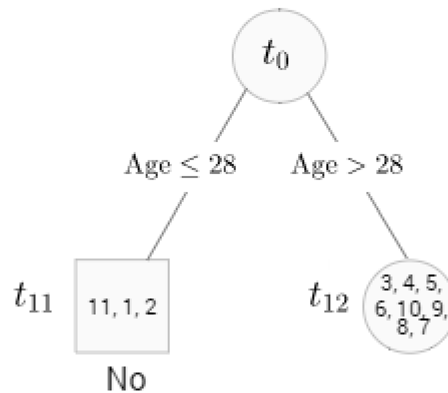



Figure 2.5: First split. Numbers in nodes $t_{11}$ and $t_{12}$ correspond to applicants fallen into each node.

| Applicant | Age | Income level | Loan granted |
|---|---|---|---|
| 3 | 30 | Medium | Yes |
| 4 | 32 | Low | No |
| 5 | 40 | High | Yes |
| 6 | 45 | Medium | Yes |
| 7 | 60 | High | No |
| 8 | 54 | Medium | Yes |
| 9 | 50 | Low | No |
| 10 | 48 | High | Yes |

Table 2.6: Subsample in node $t_{12}$.

the node $t_{12}$, see Table 2.6. Again, the same procedure as before is repeated only with this subsample.

$$i(t_{12}) = -\left[\frac{3}{8}\log_2\frac{3}{8} + \frac{5}{8}\log_2\frac{5}{8}\right] = 0.95.$$

| Applicant | 3 | 4 | 5 | 6 | 10 | 9 | 8 | 7 |
|---|---|---|---|---|---|---|---|---|
| Age | 30 | 32 | 40 | 45 | 48 | 50 | 54 | 60 |
| Loan granted | Yes | No | Yes | Yes | Yes | No | Yes | No |

↑ ↑ ↑ ↑ ↑

Table 2.7: Second split. Age.

| Age | | | | | | | |
|---|---|---|---|---|---|---|---|
| Cutpoint | Splits | $q_i$ | $P(\text{No}\vert t_i)$ | $P(\text{Yes}\vert t_i)$ | $i(t_i)$ | $\sum_j q_j i(t_j)$ | $G(t_{12}, s)$ |
| 31 | Age $\leq$ 31 | $1/8$ | $0/1$ | $1/1$ | 0 | 0.86 | 0.09 |
| | Age $>$ 31 | $7/8$ | $3/7$ | $4/7$ | 0.98 | | |
| 36 | Age $\leq$ 36 | $2/8$ | $1/2$ | $1/2$ | 1 | 0.94 | 0.01 |
| | Age $>$ 36 | $6/8$ | $2/6$ | $4/6$ | 0.92 | | |
| 49 | Age $\leq$ 49 | $5/8$ | $1/5$ | $4/5$ | 0.72 | 0.79 | 0.16 |
| | Age $>$ 49 | $3/8$ | $2/3$ | $1/3$ | 0.92 | | |
| 52 | Age $\leq$ 52 | $6/8$ | $2/6$ | $4/6$ | 0.92 | 0.94 | 0.01 |
| | Age $>$ 52 | $2/8$ | $1/2$ | $1/2$ | 1 | | |
| 56 | Age $\leq$ 56 | $7/8$ | $2/7$ | $5/7$ | 0.86 | 0.75 | 0.3 |
| | Age $>$ 56 | $1/8$ | $1/1$ | $0/1$ | 0 | | |

Table 2.8: Information gain. Age. Split 2.

| Income level | Low | | Medium | | | High | | |
|---|---|---|---|---|---|---|---|---|
| Applicant | 4 | 9 | 3 | 6 | 8 | 5 | 7 | 10 |
| Loan granted | No | No | Yes | Yes | Yes | Yes | No | Yes |

Table 2.9: Income level. Split 2.

| Income level | | | | | | |
|---|---|---|---|---|---|---|
| Splits | $q_i$ | $P(\text{No}\vert t_i)$ | $P(\text{Yes}\vert t_i)$ | $i(t_i)$ | $\sum_j q_j i(t_j)$ | $G(t_{12}, s)$ |
| Income level = Low | $2/8$ | $2/2$ | $0/2$ | 0 | 0.35 | 0.60 |
| Income level = Medium | $3/8$ | $0/3$ | $3/3$ | 0 | | |
| Income level = High | $3/8$ | $1/3$ | $2/3$ | 0.92 | | |

Table 2.10: Information gain. Income level. Split 2.

See Tables 2.7, 2.8, 2.9 and 2.10. It is deduced that the second and last split, as it can be seen in Figure 2.6, is the multi-splitting by the income level. Last, the new terminal nodes have been labeled. The conclusion drawn from this classifier is the following: it is likelt that, if an applicant is early age or is older but has low income, the bank does not look like to give that loan. In other cases, the response is “Yes”.

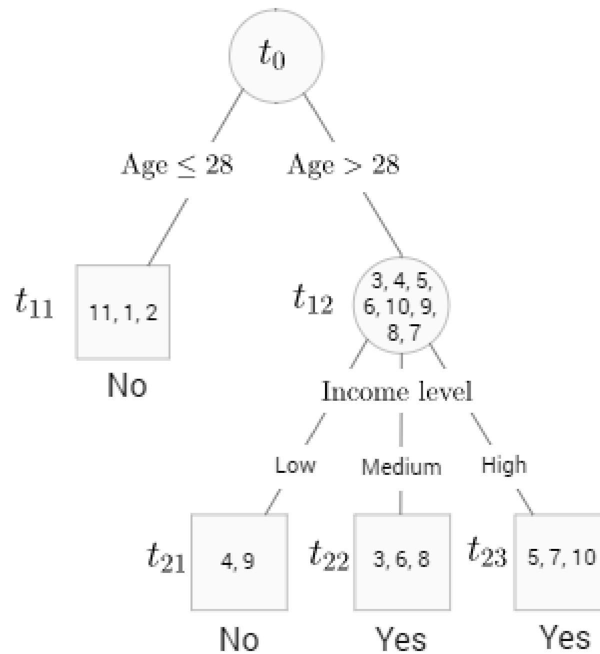



Figure 2.6: Second split. Final tree.

## 2.7 Tree Pruning

Building a too big tree may cause overfitting and building a too small tree, perhaps, all the information available in the sample $D_n$ for classification may not be used. So it is sometimes preferable to search the right-sized tree. The usual way to do this is to build a tree large enough $T_{\max}$ (for instance, until every node contains one single observation) and then, having the general overview of the tree, to prune it up in a proper way to obtain a subtree.

Given a node $t$, to prune a tree $T$ from node $t$ consists of deleting every descesdant node of $t$; that is, if $T_t$ is the subtree with root node $t$, the pruning of $T_t$ is to delete every node in $T_t$ except $t$, which will turn into a terminal node. As usual, the most frequent class will label this new node. The resulting tree $T'$ is a subtree of $T$, $T' \preceq T$. Such reduction should be made only if $T'$ does not performs worse than $T$ over the set of objects used for its validation.

The best known procedure for pruning a tree is *cost-complexity pruning*, proposed in [8]. Given a classification tree $T$, its performance is defined as a sum of some measure over every terminal node, i.e.,

$$R(T) = \sum_{t=1}^{|T|} i(t) = \sum_{t=1}^{|T|} \Phi\left(P(C_1|t), \ldots, P(C_k|t)\right),$$

where $\Phi$ usually denotes the fraction of cases in the training sample that are misclassified, but any impurity function can be used. $R(T)$ is called the *resubstitution error* of $T$. So, given a real number $\alpha$, the total cost $R_\alpha(T)$ of tree T is defined as

$$R_\alpha(T) = R(T) + \alpha|T|. \tag{2.10}$$

The parameter $\alpha$ is the penalty imposed over the complexity (size) of the tree; while small values of $\alpha$ will originate trees with a huge number of terminal nodes, big values of $\alpha$ will originate trees with few terminal nodes. The basic idea given in [8] is to select a subtree $T(\alpha)$, with the same root node that $T_{\max}$, that minimizes (2.10).

## 2.8 Regression trees

For regression trees, the construction is quite similar. In fact, Sections 2.2 and 2.4 can be applied to the regression case without change. Regarding splitting criteria, the split is selected by minimizing the Residual Sum of Squares (RSS). The residual of each observation is computed as the difference of its value in the response variable and the mean value of all observations in the same node. By last, as expected, the prediction or value associated to each terminal node is usually the mean of the response variable in every training object that has fallen into the same terminal node.

# Chapter 3

# Ensembling trees: Random Forests

*Random forests* (RFs) are a state of the art prediction method. RFs are known for usually providing great predictions and being flexible enough to deal with large-scale problems, even in settings where the number of variables is much larger than the number of observations. Despite being widely used, RFs' performance is not supported by many theoretical results. There is, therefore, a little gap between theory and empirical experience in this scheme. Even so, along this chapter, the most celebrated theoretical results of RFs will be sketched out.

A random forest is a collection of individual decision trees, reviewed in Chapter 2, each of which is constructed by applying randomness twice: first, a training sample is selected randomly for each tree and, secondly, randomness is injected somehow in the split selection process. The term random forest is attributed to Breiman, who first introduced it in [4].

RFs inherit the properties from decision trees, which were named in the introduction of Chapter 2: they give measures of variables importance and proximities between observations, mainly. The extension of these properties to RFs usually makes the conclusions more reliable since a collection of different predictors are now taken into account. In addition to others, these properties are fully discussed afterwards.

## 3.1 Random forests. Influences and definition.

### 3.1.1 Background

First of all, the road to RFs is introduced to the reader in this first section.

Decision trees, discussed in Chapter 2, are known to suffer from high variance. This means that if two decision trees are grown over two disjoint subsamples from the training data, they may lead to quite different results. A general procedure for reducing variance of any learning method is bagging [3], which has already been presented in Chapter 1. The idea of bagging given in Chapter 1 was to give a technique for handling

small datasets. In this context, using bagged trees is based on the statistical result outlined in Note 3.1.1.

**Note 3.1.1.** *Given $B$ independent observations, each with variance $\sigma^2$, the variance of the mean of these observations is reduced to $\sigma^2/B$.*

So, considering a collection of decision trees is translated into a reduction of variance, which directly leads to an increase of prediction accuracy. The *random subspace* method was also proposed for constructing decision forests [21]. This method aims to reduce correlation between trees by randomizing the predictor variables to use in each individual tree. The next stop to RFs is *random split selection* [16], which uses bagging with the only difference that at each node, among the $k$ best potential splits, the final split is chosen in a random way. Breiman emphasizes that the key paper, the one that was really decisive to develop random forests, was [1], in which a random selection of features at each split is done for a particular problem. And this is how the Random Forests grew up [4].

### 3.1.2 Definition

Random forests share common characteristics with bagging: $B$ decision trees are grown over $B$ bootstrapped training samples and the prediction is the class with majority vote too; but, at each time a split is considered, a random sample of $m$ predictor variables is chosen randomly among the $p$ initial predictor variables. Remember from Chapter 1 that the observations not appearing in the bootstrap sample are called OOB observations. At first sight, this randomization of bagged trees seems not to make sense but it helps to "decorrelate" trees. In order to understand how it decorrelates trees we will use the explanation given in [24], which is quite simple. Consider a dataset that contains a very strong predictor variable together with other moderately strong predictor variables. Then, although the number of decision trees is large, most of them will locate the strong predictor variable in the top split. In consequence, every decision tree will look quite similar to each other, and predictions in all the trees will be highly correlated. This fact would imply that the reduction in variance will not be as important as expected.

**Definition 3.1.1** (Random forests' prediction)**.** *Let $\{m_n(\boldsymbol{X}, \theta_b)\}_{1 \leq b \leq B}$ be a family of $B$ individual decision trees, built as in Algorithm 2. Given an unlabeled observation $\boldsymbol{x}$, its predicted class using RFs is*

$$m_n(\boldsymbol{x}, \theta_1, \dots, \theta_B) = \arg\max_{C_j} \sum_{b=1}^{B} I_{\{m_n(\boldsymbol{x}, \theta_b) = C_j\}},$$

*where $I(\cdot)$ is the indicator function.*

**Algorithm 2:** RF's construction.

**Input**: inputs in Algorithm 1, the number of trees $B$ and the number of predictor variables to select randomly at each split $p$.

**for** $b \in \{1, \ldots, B\}$ **do**

Generate a bootstrap sample of $D_n$, $D_n(\theta_b)$.
Store the OOB observations in $\overline{D_n(\theta_b)}$.
Construct an individual decision tree according to Algorithm 1 over $D_n(\theta_b)$.
Compute the bootstrap error for the tree, i.e.,

$$e_{boot}(b) = \frac{\sum_{(\boldsymbol{x},y)\in\overline{D_n(\theta_b)}} I_{\{m_n(\boldsymbol{x};\theta_b)\neq y\}}}{|\overline{D_n(\theta_b)}|}.$$

**end**

Obtain the bootstrap error by averaging over all the trees, i.e.,

$$e_{boot} = \frac{1}{B}\sum_{b=1}^{B} e_{boot}(b).$$

**Output**: The collection of individual decision trees for RFs, $\{m_n(\boldsymbol{X}, \theta_b)\}_{1\leq b\leq B}$, and the bootstrap error, $e_{boot}$.

#### 3.1.2.1 Parameters tuning

Research in parameters tuning is scarce in RFs. Article [15] handles these issues.

The first parameter is the size of the forest, that is, the number $B$ of individual decision trees to be grown. Breiman [4] introduces an interesting result for $B$: RFs do not overfit as larger $B$ is, but yield a limiting value of the generalization error. Theorem 3.1.1 picks up this result. Two previous definitions are needed.

**Definition 3.1.2** (Margin function)**.** *Let* $\{m_n(\boldsymbol{X}, \theta_b)\}_{1\leq b\leq B}$ *be a family of* $B$ *individual decision trees. The margin function* $mg(\cdot)$ *is defined as*

$$mg(\boldsymbol{X}, Y) = \left(\frac{1}{B}\sum_{b=1}^{B} I_{\{m_n(\boldsymbol{X},\theta_b)=Y\}}\right) - \max_{j\neq Y}\left(\frac{1}{B}\sum_{b=1}^{B} I_{\{m_n(\boldsymbol{X},\theta_b)\neq j\}}\right)$$

*where* $I(\cdot)$ *is the indicator function.*

According to Definition 3.1.2, the margin function measures how much (in frequency) objects correctly classified exceed objects misclassified in any other class. Thus, the larger this margin function is, the more confidence in the prediction.

**Definition 3.1.3** (Generalization error)**.** *The generalization error of a random forest is defined as the probability that the margin function is negative, i.e.:*

$$PE^* = \mathbb{P}_{(\boldsymbol{X},Y)} \left[ mg(\boldsymbol{X}, Y) < 0 \right].$$

**Theorem 3.1.1.** *The generalization error $PE^*$ converges almost surely to*

$$\mathbb{P}_{(\boldsymbol{X},Y)} \left[ \mathbb{P}_\theta \left( m_n(\boldsymbol{X}, \theta) = Y \right) - \max_{j \neq Y} \left( m_n(\boldsymbol{X}, \theta) = j \right) < 0 \right].$$

The proof of Theorem 3.1.1 follows directly from the Strong Law of Large Numbers and it can be found in [4]. Actually, this result can be extended to any ensemble of classifiers. One consequence of Theorem 3.1.1 is that one does not require to compute a very large number of decision trees, since, from a value $B^*$ on, the predictive power of the model will stay the same. In this context, Latinne el al. propose in [28] an approach for determining a priori the number $B$ in order to obtain an accuracy similar to the one obtained with a larger $B$. This approach is based on a non-parametric test, the McNemar test [29]. Given two random forests $RF_m$ and $RF_n$ of size $m$ and $n$, respectively, the McNemar test compares the number of examples misclassified by $RF_m$ but not by $RF_n$ (labeled $M_{mn}$) and the number of examples misclassified by $RF_n$ but not by $RF_m$ (labeled $M_{nm}$), i.e., informally, the test is

$$H_0 : \text{There is no difference between } RF_n' \text{ and } RF_m' \text{ predictions.}$$

There are three possible answers when computing the McNemar test: to reject $H_0$ with $M_{mn} > M_{nm}$, in this case the conclusion is that combining $n$ decision trees yields a significant improvement in performance than combining $m$ decision trees, and the procedure should carry on with a $B$ larger than $m$; to reject $H_0$ with $M_{nm} > M_{mn}$, here the procedure must stop and use $B = m$; or not to have significant evidence for rejecting $H_0$, which means that there is not significant difference between growing $n$ or $m$ decicion trees, so the final decision must be to construct the minimum classifiers as possible: $B = m$. Experimental results in [28] show that $B$ can be limited significantly. Nevertheless, authors in general, included [15], believe that the parameter $B$ is irrelevant and opt not to tune it, as long as they take it large enough for getting the stability (Breiman [6] proposes $B = 1000$ or $B = 5000$) but for computations to be completed within a reasonable time.

The second parameter is $m$, the number of predictor variables taking part in each split. Breiman in [5] found $m = \lceil \sqrt{p} \rceil$ to be a good choice since he obtained generally near optimum results. His advice is to grow three random forests with $m = \lceil \sqrt{p} \rceil$, $m = 2\lceil \sqrt{p} \rceil$ and $m = \frac{1}{2}\lceil \sqrt{p} \rceil$, respectively, and observe the one that performs the best and move around that value. Breiman also points out that a higher $m$ works better when noise predictor variables are present. In [15], they conclude, again, that tuning this parameter is not an interesting task.

Aside from choosing $B$ and $m$, the elements of each tree have to be decided, that is, the topology of the trees, the types of splitting, the splitting criterion, the stopping criterion and if they are pruned trees or not. So far, the proposed RFs' are collections of unpruned trees. The elements named are the ones that differentiate between trees. For instance, Breiman's original forest uses CARTs: unpruned trees with univariate splits, binary splitting, the information gain as a splitting criterion and the nodesize criterion. In the `R` package `randomForest`, 1 is set as a default value of `nodesize`, and 5 for regression. These values are reported to be a good choice in [15].

## 3.2 Properties of Random Forests

During the construction of the individual trees the OOB observations can be used to measure the performance of the forest without requering an independent validation set (as we could appreciate in Algorithm 2), as well as obtaining some information about the data. Also, the output of the individual trees give us some interesting information. Hereafter, the main properties of RFs are being reviewed.

### 3.2.1 Variable importance

RFs provide variable importance measures, making them very interesting since a hierarchy of the predictor variables can be obtained from the model, that is, it can be measured how related each predictor variable is to the response variable. Moreover, these measures help with another appealing task in Machine Learning: *Feature Selection*. Feature Selection is the process of discarding irrelevant or redundant predictor variables, without losing power in prediction. In this way, the robustness of the classifier may be improved and computing time will be reduced.

Two embedded methods for measuring variable importance are described: the Mean Decrease Accuracy (MDA, [4]) and the Mean Decrease Impurity (MDI, [5]). They are called embedded because they are specific for RFs and are computed during the training process.

**Note 3.2.1.** *Breiman [5] proposed to grow more than the usual number of trees if one searches for some auxiliary information like variable importance measures or proximities (which will be seen later), to make these measures stable. In [15], Breiman's thesis is supported experimentally: as $B$ grows, the variable importance measures start to be stable.*

#### 3.2.1.1 Mean Decrease Accuracy

The Mean Decrease Accuracy (MDA, [4]), also known as the permutation importance measure, is one of the most common variable importance measures. MDA is based on the following principle: if a variable is not influential in the model, rearranging the

values it takes should not degrade prediction accuracy. If the predictor variable brings nothing but random noise, the prediction accuracy will like not to be affected after the permutation. OOB observations will be the main characters in MDA. Every time an individual tree is grown over a bootstrap sample, accuracy in OOB observations is going to be computed. Also, accuracy in OOB observations after permuting the values of some variable will be computed. The MDA of that variable is obtained by averaging over all trees the differences of both accuracies. See Algorithm 3.

**Algorithm 3:** Computing MDA.

**Input**: inputs in Algorithm 2.

**for** $b \in \{1, \ldots, B\}$ **do**

Generate a bootstrap sample of $D_n$, $D_n(\theta_b)$.
Store the OOB observations in $\overline{D_n(\theta_b)}$.
Construct an individual decision tree according to Algorithm 2 over $D_n(\theta_b)$.
Compute the number of correctly classified samples in $\overline{D_n(\theta_b)}$,i.e.,

$$Acc_b = \sum_{(\boldsymbol{x},y)\in\overline{D_n(\theta_b)}} I_{\{m_n(\boldsymbol{x};\theta_b)=y\}}.$$

**for** $j \in \{1, \ldots, p\}$ **do**

Permute randomly the values that predictor variable $j$ takes on the set $\overline{D_n(\theta_b)}$, yielding a sample $\overline{D_n(\theta_b)}^j$.
Compute the number of correctly classified samples in $\overline{D_n(\theta_b)}^j$,i.e.,

$$Acc_{jb} = \sum_{(\boldsymbol{x},y)\in\overline{D_n(\theta_b)}^j} I_{\{m_n(\boldsymbol{x};\theta_b)=y\}}.$$

Compute $dif_{jb} = Acc_b - Acc_{jb}$.

**end**

**end**

**for** $j \in \{1, \ldots, p\}$ **do**

$$MDA(X^{(j)}) = \frac{1}{B}\sum_{b=1}^{B} dif_{jb}.$$

**end**

**Output**: Mean Decrease Accuracy (MDA) for each predictor variable.

Therefore, the larger the MDA, the better the associated predictor variable.

**Note 3.2.2.** *If permutations are done over OOB observations in the same class, a measure of variable importance over each class is obtained.*

#### 3.2.1.2 Mean Decrease Impurity

Another way for ranking predictor variables is Mean Decrease Impurity (MDI, [5]). Recall that the splitting criterion used for growing a decision tree, extensively explained in Section 2.3, was to select among every non-terminal node that split that maximizes the information gain. This means that if a variable appears in a given node is because, among the other $m$ preselected variables, it is the one that best separates between classes. As a result, the MDI is based on that idea: given a predictor variable $X^j$, its corresponging MDI is obtained by averaging over all the trees in the forest the decrease of impurity (or, equivalently, information gain in our language) corresponding to splits along that variable, which is weighted with the fraction of examples falling in that node. As we can see, here the OOB observations do not take part, only the bootstrapped training samples are used. See Algorithm 4.

When the impurity function is the Gini index, this measure is commonly called *Gini importance*.

---

**Algorithm 4:** Computing MDI.

---

**Input**: inputs in Algorithm 2.

**for** $b \in \{1, \ldots, B\}$ **do**
  Generate a bootstrap sample of $D_n$, $D_n(\theta_b)$.
  Construct an individual decision tree according to Algorithm 2 over $D_n(\theta_b)$.
  Initialize $WIG$ as the null vector of dimension p.
  **for** $j \in \{1, \ldots, p\}$ **do**
    **for** $t \in \{1, \ldots, number.of.non.terminal.nodes\}$ **do**
      **if** *$j$ partitions node $t$* **then**

$$WIG(X^j) = WIG(X^j) + \frac{N_t}{N} IG(t, s)$$

      **end**
    **end**
  **end**
**end**
**for** $j \in \{1, \ldots, p\}$ **do**

$$MDI(X^j) = \frac{1}{B} \sum_{b=1}^{B} WIG(X^j).$$

**end**

**Output**: Mean Decrease Impurity (MDI) for each predictor variable.

---

**Note 3.2.3.** *The importance of a predictor variable is usually given by its relative influence, which is simply the fraction of its importance measure over the sum of the*

*importance measures of all variables:*

$$RIM(X^j) = \frac{IM(X^j)}{\sum_{i=1}^{p} IM(X^i)},$$

*where $RIM$ is the abbreviation of Relative Importance Measure, $IM$ is the Importance Measure that can be any of $MDA$ and $MDI$ and $X^j$ refers to the j-th predictor variable.*

### 3.2.2 Proximity measure

A proximity measure quantifies the similarity or dissimilarity of pairs of objects. RFs give a novel and embedded way to obtain a similarity measure between objects.

The standard proximity measure was proposed by Breiman [5] and is computed as follows. Let $i$ and $j$ be two objects, the similarity measure between both, $\delta_{ij}$, is the proportion of trees that the RF places both in the same terminal node. Starting with $\delta_{ij} = 0$, object $i$ and object $j$ are applied down each tree and, each time they end up in the same terminal node, $\delta_{ij}$ is increased by one. Finally, this measure is normalized by the number of trees $B$. Usually, this proximity measure is calculated while the construction of RF taking use of the OOB observations. Now, it is not normalized by $B$ but by the number of trees where each pair of OOB observations concur.

Proximities are represented by an object-by-object matrix, $\Delta = (\delta_{ij})$, which is symmetric. Every $\delta_{ij}$ takes values on the closed interval $[0, 1]$. $\delta_{ij}$ near to one means that objects $i$ and $j$ are alike; so, the main diagonal only contains ones since any object is trivially similar (equal) to itself. As consequence, the closer $\delta_{ij}$ is to zero, the more dissimilar objects $i$ and $j$ are.

Recall that Breiman in [4] and authors in [15] find it necessary to take a large number of trees to get stable estimates of data proximity. In [18], a new proximity measure when few trees are grown is proposed:

$$\delta_{ij} = \frac{1}{B} \sum_{b \in B} \frac{1}{e^{w \cdot g_{ijb}}},$$

where $g_{ijb}$ is the number of branches between the two terminal nodes where $i$ and $j$ have fallen in tree $b$, and $w$ is an arbitrary parameter that controls the influence of the distance between both terminal nodes. To know how $g$ works, go to Figure 2.1 and check that $g_{t_{11},t_{21},1} = 3$. In this way, given a tree $b$, if $i$ and $j$ end up in the same terminal node, $g_{i,j,b} = 0$ and $\delta_{ij}$ will be increased by one as in the original proximity measure.

In addition to give a different proximity measure using RFs, in [18] an approach for assessing the quality of data proximity matrices is proposed: to use them as kernel matrices in a Support Vector Machine (SVM) classifier. The best data proximity matrix will be the one that gives the highest classification accuracy. Both proximity measures,

the standard and the new ones, are compared thereby. An SVM based on the radial basis function kernel is also used. Experimental results across four data sets show that the proposed measure improves the data proximity estimate, especially when RFs are made of a small number of trees. Furthermore, an SVM exploiting the suggested proximity matrix kernel has been able to outperform an SVM based on the standard radial basis function kernel in many cases.

A data proximity matrix is an important information source from which RFs can take sides for many relevant tasks in data mining: data visualization via scaling, outlier detection, missing values imputation and prototypes search; some of them specific for RFs' similarities and others, more general. A review of how the proximity matrix obtained with RF can be applied to these fields is done next, [7].

#### 3.2.2.1 Data visualization

Data visualization includes every technique that helps to uncover hidden patterns in data by means of pictures. In order to provide a visual representation of the pattern of proximities, RF's similarity measure in our case, the classical approach used for this is *MultiDimensional Scaling* (MDS, [13]). MDS is similar to *Principal Component Analysis* (PCA) with the main difference that PCA uses as input the correlation matrix and MDS, any proximity matrix (dissimilarity matrices, generally). The RF's dissimilarity matrix is defined as $\bar{\Delta} = (\bar{\delta}_{ij})$, where

$$\bar{\delta}_{ij} = \sqrt{1 - \delta_{ij}}.$$

Without going into much detail, the main idea of the MDS is, given $\bar{\Delta}$, to construct $n$ vectors, $v_1, \dots, v_n \in \mathbb{R}^m$, as many as observations in the sample, such that $\|v_i - v_j\| \simeq \bar{\delta}_{ij}\ \forall i, j = 1, \dots, n$, where $\| \cdot \|$ is an arbitrary norm. When this norm is the Euclidean one, the MDS is known as the Classical MDS (CMDS). In other words, if we are dealing with CMDS, for instance, the method returns a set of points in a low dimensional Euclidean space such that the Euclidean distances between the points are preserved.

The $m$ new coordenates are ordered in the sense that for $i < j$, the $i$-th coordenate explains more about the proximity of data than the $j$-th coordenate, $\forall i, j = 1, \dots, m$. For this reason, the first two coordinates of all new points $v_1, \dots, v_n$ are usually projected down into the two dimensional plane and if points of different classes are also coloured differently, an overview of how separated the classes are can be observed. In this way, this picture would allow analysts to decide if the model explains the response variable properly or, by contrast, the information obtained is confusing. The way of interpreting this graph is: the closer the distance between points, the closer the similarity between their corresponding objects.

In Chapter 2 of [13], a practical algorithm for the CMDS and its theoretical development can be found.

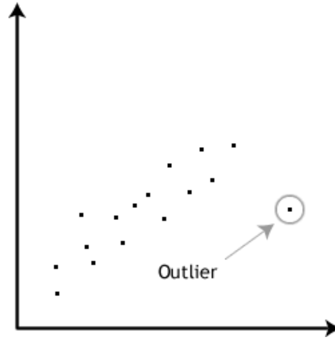



Figure 3.1: Representation of an outlier.

#### 3.2.2.2 Outliers detection

In Statistics, an outlier is an observation that is numerically distant from the rest of the data. See Figure 3.1. It is quite interesting to explore data in order to locate outliers, since the conclusions obtained from a particular model can be disturbed if they include such observations. Sometimes they should even be excluded.

Proximities allow to obtain a measure of outlyingness for each observation in the data set. In this setting, Supervised Classification and Proximity Measures, outliers are defined as observations having weak proximities to the remaining observations in the same class. Given an observation $(\boldsymbol{x}_i, y_i)$, let $I_i$ the set of observations in $D_n$ that belongs to the same class, that is,

$$I_i = \{(\boldsymbol{x}_j, y_j) \in D_n : y_j = y_i, i \neq j\}\,.$$

Take the inverse of the average of the squared similarity measures in $I_i$, yielding to

$$O_i = \frac{|I_i|}{\sum_{j \in I_i} \delta_{ij}^2}.$$

Now, $O_i$'s are normalized by substracting the median in $I_i$, $M_i$, and dividing by the mean absolute deviation of $O_i$ from the median, $MAD_i = |O_i - M_i|$. The final outlying measure for observation $i$ is

$$O_i^* = \frac{O_i - M_i}{MAD_i}.$$

This measure of outlyingness takes large values for instances away from others in its same class. Generally, a value over 10 for a given observation is a sufficient reason to suspect that such observation is a potential outlier, [5]. Negative values of this measure are set to zero.

#### 3.2.2.3 Missing values imputation

Observations with missing values are those in which the value of any of the predictor variables is unknown. A common practice is to get rid of those records that contain

missing values or of the predictor variable that contains them. The former is not effective if the size of $D_n$ is small; the latter is even worse, since that predictor variable could be decisive for the classification. A first and cheap solution for not losing any information is to replace every missing value in the $m$-th predictor variable by the median or the most frequent value of non-missing values in that predictor variable, depending on whether it is a continuous or categorical variable, respectively. Breiman [5] also proposes an iterative process using RFs for improving this missing values imputation by means of RFs and their proximity measure. See Algorithm 5.

**Algorithm 5:** Missing values imputation using RF.

**Step 1** Make an initial estimate of the missing values using the median or mode as appropriate, depending on if the variable is continuous or categorical, respectively.

**Step 2** Compute Algorithm 2 and obtain the data proximity matrix.

**Step 3** Make a new imputation of initial missing values:

*Continuous* Weighted average of non-missing values in that variable, using proximities as weights.

*Categorical* The most frequent modality of non-missing values in that variable, weighting the frequencies by the proximities.

**Step 4** Repeat **Step 2** and **Step 3** to ensure proximities convergence.

The missing values imputation method described above is more expensive; nevertheless, it has turned out to be remarkably effective. For Step 4 in Algorithm 5, Breiman pointed to make from 4 to 6 iterations.

#### 3.2.2.4 Prototypes search

Through the proximity matrix obtained in RF, prototypes can be estimated. Given a class $C_i$, a prototype is defined as a representative instance of the whole class $C_i$. The

method for selecting the prototypes for each class is explained in Algorithm 6.

**Algorithm 6:** Protoypes selection.

**Input**: the sample $D_n$, the similarities $\delta_{ij}$ and a scalar $s$.

For each observation in $D_n$, the $s$ nearest neighbors using $\delta_{ij}$.
For each class, the observation that has more neighbors of that class is identified. The prototype for the class is the neighbors' medioid.

**Output**: A prototype for each class.

The medioid in Algorithm 6 is obtained by calculating the median for the continuous predictor variables and the mode for the categorical ones.

## 3.3 Regression Random Forests

Regression RFs are a collection of regression decision trees. Randomness is applied twice in the same way as for Classification RFs. For regression, $m$ is suggested to be $\lceil \frac{p}{3} \rceil$ and, for Breimans' forests, `nodesize` is usually $5$. As expected, the prediction of a collection of regression decision trees is the mean of the predictions over each individual regression tree.

# Chapter 4

# Random Forests in R

This last chapter is devoted to put into practice everything learned about RFs in the previous chapters using the software R. First, the necessary code for exploiting RFs' properties is shown; second, an experiment to measure the ability of RFs for Feature Selecion tasks.

The package randomForest is going to be used along this chapter, whose main function is also called *randomForest* that has as input arguments the following.

```
## S3 method for class 'formula'
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
## Default S3 method:
randomForest(x, y=NULL, xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
             max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
             replace=TRUE, classwt=NULL, cutoff, strata,
             sampsize = if (replace) nrow(x) else
              ceiling(.632*nrow(x)),
             nodesize = if (!is.null(y) && !is.factor(y))
              5 else 1,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest),
             corr.bias=FALSE,
             keep.inbag=FALSE, ...)
```

Most arguments are selfexplanatory. Note that the default value of importance is FALSE but, in the case the analyst writes TRUE the ranking of the predictor variables explained in the Subsection 3.2.1 is done. Furthermore, proximity works similar:

it will take the value TRUE if the analyst wants to store the standard proximity matrix and FALSE, otherwise.

## 4.1 Random Forests and properties

The purpose of this section is to show the properties of RFs using R. The study will be carried out over the well-known Iris data set, which contains the measures in centimeters of the predictor variables *Sepal Length*, *Sepal Width*, *Petal Length* and *Petal Width* for 150 flowers of three different species: *Setosa*, *Versicolor* and *Virginica*. This data set is balanced: there are exactly 50 individuals from each of the classes. For classification, the goal would be to correctly classify the three species according to the four characteristics of the flowers.

First, we load the data and get a brief statistical summary of the data.

```
data(iris)
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",
##                "virginica"
```

```
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length
##  Min.   :4.300   Min.   :2.000   Min.   :1.000
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600
##  Median :5.800   Median :3.000   Median :4.350
##  Mean   :5.843   Mean   :3.057   Mean   :3.758
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100
##  Max.   :7.900   Max.   :4.400   Max.   :6.900
## Petal.Width          Species
##  Min.   :0.100   setosa    :50
##  1st Qu.:0.300   versicolor:50
##  Median :1.300   virginica :50
##  Mean   :1.199
##  3rd Qu.:1.800
##  Max.   :2.500
```

```
library(randomForest)
```

In this context, we do not worry about tuning any parameter and we build the random forest taking $B = 1000$ and $m = 2$. The training sample will be the entire data set. Because of randomness in this technique, it is advisable to previously set a seed in order to make the results reproducible.

```
set.seed(1349187)
iris.rf = randomForest(Species~., data=iris, ntree=1000,
                       mtry=2)
print(iris.rf)
```

```
##
## Call:
##  randomForest(formula = Species ~ ., data = iris,
##  ntree = 1000, mtry = 2)
##                Type of random forest: classification
##                      Number of trees: 1000
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 4.67%
## Confusion matrix:
##            setosa versicolor virginica class.error
## setosa         50          0         0        0.00
## versicolor      0         47         3        0.06
## virginica       0          4        46        0.08
```

As it can be seen above, we obtain a prediction error rate of 4.67% and, in view of the confusion matrix, we conclude that the specie *Setosa* is the best classified over the OOB observations. Moreover, the model is presumed to confuse the species *Versicolor* and *Virginica*.

The first property we are going to obtain from the built random forest is the importance of the four predictor variables in the model. In order to rank them according to their importance, the same random forest will be re-build imposing `importance = TRUE`.

```
set.seed(1349187)
iris.rf = randomForest(Species~., data=iris, ntree=1000,
                       mtry=2, importance=TRUE)
importance(iris.rf)
```

```
##                  setosa versicolor virginica
```

```
## Sepal.Length  8.916362  10.098419 12.051028
## Sepal.Width   6.624870   1.037431  8.454989
## Petal.Length 30.762171  46.828426 39.621372
## Petal.Width  32.085780  46.794090 44.675201
##              MeanDecreaseAccuracy MeanDecreaseGini
## Sepal.Length    15.506218          9.976061
## Sepal.Width     8.264094           2.454234
## Petal.Length    46.363973          42.803821
## Petal.Width     47.402607          44.058583
```

The previous table shows the measures of importance of each variable. Let us see it graphically.

```
varImpPlot(iris.rf)
```

Both measures of variable importance reviewed are represented in the picture above: on the left side, the MDA; on the right side, the MDI using the Gini index as the impurity function. On this occasion, there is no discrepancy between both measures since they point out that the predictor variables *Petal.Width* and *Petal.Length* are the most important.

Next, following the same order than in Section 3.2, let us obtain the standard proximity matrix. Again, the same forest is built, now adding `proximity = TRUE`.

```
set.seed(1349187)
iris.rf = randomForest(Species~., data=iris, ntree=1000,
                       mtry=2, proximity=TRUE)
```

A matrix of dimension 150 by 150 is obtained. Let us see, for example, the measure of proximity for the first five observations, which correspond to flowers of the species *Setosa*.

```
iris.rf$proximity[1:5,1:5]
```

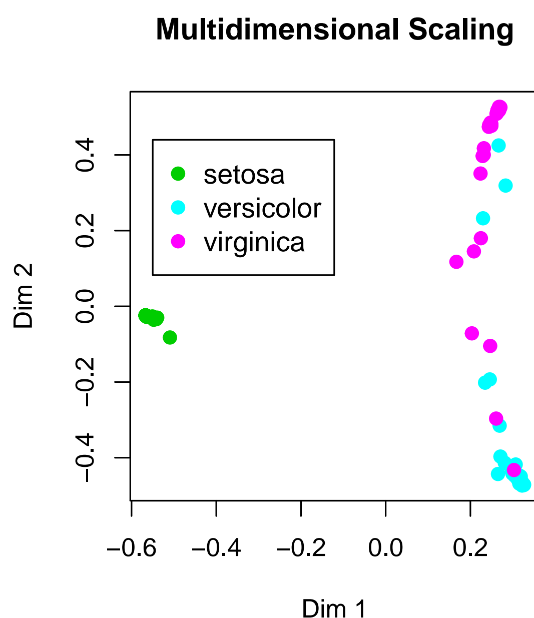
```
##           1         2         3         4         5
## 1 1.0000000 0.9683544 0.9931973 0.9930070 1.0000000
## 2 0.9683544 1.0000000 0.9877301 0.9790210 0.9791667
## 3 0.9931973 0.9877301 1.0000000 1.0000000 0.9925373
## 4 0.9930070 0.9790210 1.0000000 1.0000000 0.9913043
## 5 1.0000000 0.9791667 0.9925373 0.9913043 1.0000000
```

It is observed that the first five cases are closely related. Also, remember that they are part of the species that our model classifies best.

From now on, we exploit the ability of the proximity matrix.

### Data visualization

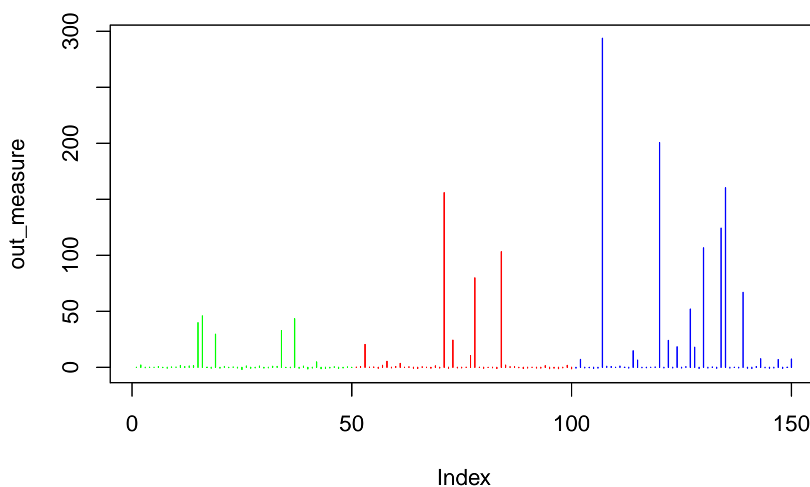
```
MDSplot(iris.rf,iris$Species,pch=19,cex=1.1,
        palette=c(3,5,6),main="Multidimensional Scaling")
legend(-0.55,0.44,col=c(3,5,6),pch=19,
        legend=levels(iris$Species),cex=1.1)
```




This graph shows that the *Setosa* species is clearly distinguished from the other two species, as the confusion matrix advanced.

**Outliers detection**

```
out_measure = outlier(iris.rf)
plot(out_measure,type="h",col=c("green","red","blue"),
[as.numeric(iris$Species)])
```




The species *Setosa* does not present huge levels of outlyingness, compared to species *Versicolor* and the *Virginica*, being the latter the one with more outliers.

**Missing values imputation**

Iris database does not present missing values; however, we will do an experiment in order to see how good the imputation method provided by random forests is when replacing missing values. Suppose we do not know the values of the variable *Sepal.Length* of flowers 1 (*Setosa*), 51 (*Versicolor*) and 101 (*Virginica*). The algorithm of imputation of lost values will be applied and, later, the difference between the predicted and the real values of the predictor variable *Sepal.Length* for the three flowers is going to be computed.

```
iris.na = iris
iris.na[1,"Sepal.Length"]=NA
iris.na[51,"Sepal.Length"]=NA
iris.na[101,"Sepal.Length"]=NA
set.seed(1349187)
iris.imputed = rfImpute(Species~., iris.na)
```

```
(dif1 =iris.imputed[1,"Sepal.Length"]
        -iris[1,"Sepal.Length"])
```

```
## [1] -0.1046023
```

```
(dif51 =iris.imputed[51,"Sepal.Length"]
        -iris[51,"Sepal.Length"])
```

```
## [1] -1.125318
```

```
(dif101 =iris.imputed[101,"Sepal.Length"]
        -iris[101,"Sepal.Length"])
```

```
## [1] 0.4308188
```

Except for the flower 51, the method has a proper performance.

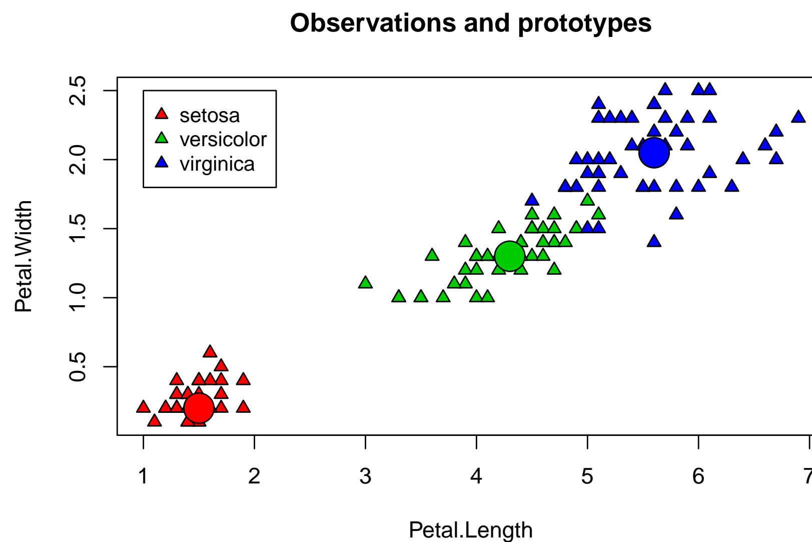
**Prototypes search**

Last, a representative instance for each species is being computed.

```
x =iris[,names(iris)!="Species"]
label =iris$Species
(iris.prot=classCenter(x,label,iris.rf$proximity))
```

```
##            Sepal.Length Sepal.Width Petal.Length
## setosa              5.0         3.4          1.5
## versicolor          5.8         2.8          4.3
## virginica           6.5         3.0          5.6
##             Petal.Width
##                    0.20
##                    1.30
##                    2.05
```

For every species, the previous table shows the medioid, i.e., the instance that best represents its class. We will represent the prototypes together with the observations over the predictor variables related to the petals.




## 4.2 Feature Selection based on Random Forests

RFs provide two measures of variable importance: the MDA and the MDI, previously reviewed in Subsection 3.2.1. In this section, the purpose is to use these measures as Feature Selection techniques and investigate their performance. Given a data set, the idea is to train different classifiers with different techniques of Feature Selection in the current literature (including those for RFs' variable importance measures) and check which technique performs best in terms of the accuracy over a test set, totally independent from the training set. The technique that gives the highest accuracy over the whole set of classifiers will be considered as the best. Remember from Chapter 1 that the accuracy is defined as the proportion of observations correctly classified by a given classifier.

The package `randomForest` will be used for computing MDA and MDI. The other techniques will be taken from the package `FSelector`, which contains some functions for selecting attributes from a given dataset. Among them, the functions *cfs*, *chi.squared*, *oneR*, *relief* and *consistency* are used. While the techniques *cfs* and *consistency* give a subset of predictor variables to consider, the techniques *chi.squared*, *oneR* and *relief* perform similar to MDA and MDI: their outcoming is a ranking of the predictor variables, so the size of the subset of predictor variables is to be chosen. In this study, around the $25\%$ of the whole set of predictor variables is taken.

The classifiers to be used in this study are: Naive Bayes (NB), Radial Basis Function (RBF) SVM and of course Random Forests (RF).

An outline of the experiment can be seen in Algorithm 7.

**Algorithm 7:** Comparison of Feature Selection techniques.

Given a data set:
**Step 1**. Split the sample into a training sample ($70\%$) and a test sample ($30\%$).
**Step 2**. Obtain the subset of predictor variables considered for every Feature Selection technique using the training sample.
**Step 3.**

- Train the different classifiers making their corresponding parameters tuning for every subset of predictor variables obtained in **Step 2**. The training sample is used.

- Measure the performance (accuracy) over the test sample of every pair Feature Selection technique - Classifier.

**Step 4**. Get conclusions.

Three different data sets are used for this task: Breast Cancer Wisconsin, Ionosphere and Spam data sets:

- Breast Cancer Wisconsin, which consists of a sample of size $569$ on which $30$ continuous features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. Moreover, there is a variable that identifies every observation, which is out of the study, and the response variable *malignant* that takes the value $1$ if the cell is malignant and $0$, otherwise.

- Ionosphere, which comprises $351$ observations (electrons in the ionosphere) on which $35$ variables were measured. The first $34$ variables are predictor and continuous, except two of them that have been removed from the study; the last one is the response variable and takes *good* if returns are those showing evidence of some type of structure in the ionosphere and *bad*, if returns are those that do not; their signals pass through the ionosphere.

- Spam, which contains a sample of $4601$ e-mails spam and non-spam so the response variable takes one of these values. In addition to this class label there are $57$ predictor variables indicating the frequency of certain words and characters in the e-mail.

The code used in R for the Spam dataset will be displayed next. For Breast Cancer Wisconsin and Ionosphere data sets, the procedure is the same.

First, the data set Spam is loaded from the library *kernlab*. During the experiment, when dealing with randomness, seeds are going to be set in order to make the results reproducible at any code line.

```
library(kernlab)
data(spam)
```

**Step 1**

```
set.seed(123)
n <- nrow(spam)
trainindex <- sample(1:n, size = floor(0.7*n))
spam.train <- spam[trainindex,]
spam.test <- spam[-trainindex,]
```

**Step 2**

At the same time each subset of predictor variables is obtained for a given Feature Selection technique, the training and the test samples for that restricted set of predictor variables are going to be stored.

MDA and MDI

The way of obtaining both variable importance measures provided by RF is based on Breiman's advice: $B$ is taken large enough in order to get stable measures, $B = 5000$, and $m$ is tuned between $m_1 = \lceil\sqrt{p}\rceil$, $m_2 = \frac{1}{2}\lceil\sqrt{p}\rceil$ and $m_3 = 2\lceil\sqrt{p}\rceil$. The chosen value of $m$ will be the one that give the low error rate in the OOB observations. For the MDI, the Gini index is taken as the impurity function.

```
library(randomForest)
```

```
p <- ncol(spam)-1
m1 <- round(sqrt(p))
m2 <- 0.5*m1
m3 <- 2*m1
```

```
set.seed(123)
RFm1 <- randomForest(type~., data=spam.train, ntree=5000,
                    mtry=m1)
print(RFm1)
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = spam.train,
##              ntree = 5000, mtry = m1)
##               Type of random forest: classification
```

```
##                      Number of trees: 5000
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 4.63%
## Confusion matrix:
##         nonspam spam class.error
## nonspam    1897   60  0.03065917
## spam         89 1174  0.07046714
```

```
set.seed(123)
RFm2 <- randomForest(type~., data=spam.train, ntree=5000,
                     mtry=m2)
print(RFm2)
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = spam.train,
##             ntree = 5000, mtry = m2)
##                Type of random forest: classification
##                      Number of trees: 5000
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 5%
## Confusion matrix:
##         nonspam spam class.error
## nonspam    1899   58  0.02963720
## spam        103 1160  0.08155186
```

```
set.seed(123)
RFm3 <- randomForest(type~., data=spam.train, ntree=5000,
                     mtry=m3)
print(RFm3)
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = spam.train,
##             ntree = 5000, mtry = m3)
##                Type of random forest: classification
##                      Number of trees: 5000
## No. of variables tried at each split: 16
##
##         OOB estimate of  error rate: 4.88%
```

```
## Confusion matrix:
##         nonspam spam class.error
## nonspam    1892   65  0.03321410
## spam         92 1171  0.07284244
```

According to the results, $m = m_1$:

```
set.seed(123)
RF <- randomForest(type~., data=spam.train, ntree=5000,
                   mtry=m1, importance=TRUE)
names <- colnames(spam)
size <- ceiling(0.25*p)
```

```
subsetMDA <-names[order(RF$importance[,"MeanDecreaseAccuracy"],
                  decreasing =TRUE)][1:size]
trainMDA <- spam.train[,c(subsetMDA,"type")]
testMDA <- spam.test[,c(subsetMDA,"type")]
```

```
subsetMDGini <-names[order(RF$importance[,"MeanDecreaseGini"],
                  decreasing =TRUE)][1:size]
trainMDGini <- spam.train[,c(subsetMDGini,"type")]
testMDGini <- spam.test[,c(subsetMDGini,"type")]
```

Now, from `FSelector` package.

```
library(FSelector)
library(RWeka)
```

CFS

```
subsetCFS <-cfs(type~., spam.train)
trainCFS <- spam.train[,c(subsetCFS,"type")]
testCFS <- spam.test[,c(subsetCFS,"type")]
```

chi.squared

```
weightsChi <-chi.squared(type~.,spam.train)
subsetChi <-cutoff.k(weightsChi, size)
trainChi <- spam.train[,c(subsetChi,"type")]
testChi <- spam.test[,c(subsetChi,"type")]
```

oneR

```
weightsOneR <- oneR(type~.,spam.train)
subsetOneR <- cutoff.k(weightsOneR,size)
trainoneR <- spam.train[,c(subsetoneR,"type")]
testoneR <- spam.test[,c(subsetoneR,"type")]
```

relief

```
weightsRelief <- relief(type ., spam.train, neighbours.count
                        = 5, sample.size = 20)
subsetRelief <- cutoff.k(weightsRelief, size)
trainrelief <- spam.train[,c(subsetrelief,"type")]
testrelief <- spam.test[,c(subsetrelief,"type")]
```

Consistency

```
subsetConsistency <- consistency(type~., spam.train)
trainConsistency <- spam.train[,c(subsetConsistency,"type")]
testConsistency <- spam.test[,c(subsetConsistency,"type")]
```

**Step 3** In this step every classifier has to be trained with the training sample of each Feature Selection technique and, then, be evaluated over their corresponding test sample. Here, the pairs NB-MDA, RBF SVM-MDA and RF-MDA are displayed. Nonetheless, all the pairs have been computed and are shown in Table 4.1.

NB

NB does not require any tuning procedure, which speeds up the experiment.

```
library(e1071)
```

```
NBMDA <- naiveBayes(type~.,data = trainMDA)
preditestNBMDA <- predict(NBMDA,testMDA[,-ncol(trainMDA)])
confutestNBMDA <- table(testMDA[,ncol(trainMDA)],
                         preditestNBMDA)
(NBMDAaccuracy <- 100*(confutestNBMDA[1,1]+
                  confutestNBMDA[2,2])/sum(confutestNBMDA))
```

```
## [1] 82.26
```

RBF SVM

The RBF SVM has two parameters to tune. The same parameters grid has been made for all cases and the pair of parameters that give the best performance is chosen.

```
set.seed(123)
tunedMDA <- tune.svm(type~., data = trainMDA,
                      gamma = 10^(-6:-1), cost = 10^(-1:1))
SVMMDA <- tunedMDA$best.model
preditestSVMMDA <- predict(SVMMDA,testMDA[,-ncol(trainMDA)])
confutestSVMMDA <- table(testMDA[,ncol(trainMDA)],
                         preditestSVMMDA)
(SVMMDAaccuracy <- 100*(confutestSVMMDA[1,1]+
                   confutestSVMMDA[2,2])/sum(confutestSVMMDA))
```

```
## [1] 92.54
```

RF

For RF, the phase of the parameters tuning has been identical to how variable importance measures provided by RF were obtained in **Step 2**: $B = 5000$ and $m$ is chosen among the values $m_1 = \lceil\sqrt{p}\rceil$, $m_2 = \frac{1}{2}\lceil\sqrt{p}\rceil$ and $m_3 = 2\lceil\sqrt{p}\rceil$, now being $p$ the size of the subset of predictor variables for a given Feature Selection technique.

```
pMDA <- length(trainMDA)-1
mMDA1 <- round(sqrt(pMDA))
mMDA2 <- 0.5*mMDA1
mMDA3 <- 2*mMDA1
```

```
set.seed(123)
RFmMDA1 <-randomForest(type~., data=trainMDA, ntree=5000,
                       mtry=mMDA1)
print(RFmMDA1)
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = trainMDA,
##               ntree = 5000, mtry = mMDA1)
##                Type of random forest: classification
##                      Number of trees: 5000
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 5.65%
## Confusion matrix:
##          nonspam spam class.error
## nonspam     1887   70  0.03576903
## spam         112 1151  0.08867775
```

```
set.seed(123)
RFmMDA2 <-randomForest(type~., data=trainMDA, ntree=5000,
                       mtry=mMDA2)
print(RFmMDA2)
```

```
##
## Call:
##  randomForest(formula = type ~ ., data = trainMDA,
##              ntree = 5000, mtry = mMDA2)
##             Type of random forest: classification
##                   Number of trees: 5000
## No. of variables tried at each split: 2
##
##       OOB estimate of  error rate: 5.99%
## Confusion matrix:
##          nonspam spam class.error
## nonspam     1894   63  0.03219213
## spam         130 1133  0.10292953
```

```
set.seed(123)
RFmMDA3 <-randomForest(type~., data=trainMDA, ntree=5000,
                       mtry=mMDA3)
print(RFmMDA3)
```

```
##
## Call:
## randomForest(formula = type ~ ., data = trainMDA,
##             ntree = 5000, mtry = mMDA3)
##             Type of random forest: classification
##                   Number of trees: 5000
## No. of variables tried at each split: 8
##
##         OOB estimate of  error rate: 5.65%
## Confusion matrix:
##         nonspam spam class.error
## nonspam     1879   78  0.03985692
## spam         104 1159  0.08234363
```

According to the results, $m = m_1$:

```
RFMDA <- RFmMDA1
preditestRFMDA <- predict(RFMDA,testMDA[,-ncol(trainMDA)])
confutestRFMDA <- table(testMDA[,ncol(trainMDA)],
                        preditestRFMDA)
(RFMDAaccuracy <- 100*(confutestRFMDA[1,1]+
                 confutestRFMDA[2,2])/sum(confutestRFMDA))
```

```
## [1] 93.92
```

**Step 4** Finally, results are presented for each data set in the study and they are discussed. The performance of NB, RBF SVM and RF without doing any Feature Selection technique has also been computed.

| Classifiers | noFS | MDA | MDGini | CFS | Chi | oneR | Relief | Consistency |
|---|---|---|---|---|---|---|---|---|
| NB | 72.48 | 82.26 | **86.75** | 86.17 | 83.13 | 53.66 | 68.79 | 74.51 |
| SVM | 93.55 | 92.54 | **93.12** | 91.74 | 92.90 | 86.68 | 86.89 | 92.54 |
| RF | 94.42 | 93.92 | **94.06** | 92.40 | 93.99 | 87.83 | 88.05 | 93.99 |

Table 4.1: Feature Selection experiment. Accuracy in Spam data set.

| Classifiers | noFS | MDA | MDGini | CFS | Chi | oneR | Relief | Consistency |
|---|---|---|---|---|---|---|---|---|
| NB | 79.24 | 89.62 | **92.45** | 91.51 | 56.80 | 87.74 | 88.68 | 83.96 |
| SVM | 96.23 | 94.34 | 94.34 | **95.28** | 91.51 | 90.57 | 94.34 | 88.68 |
| RF | 95.28 | 93.40 | 93.40 | **96.23** | 93.40 | 88.68 | 89.62 | 92.45 |

Table 4.2: Feature Selection experiment. Accuracy in Ionosphere data set.

| Classifiers | noFS | MDA | MDGini | CFS | Chi | oneR | Relief | Consistency |
|---|---|---|---|---|---|---|---|---|
| NB | 97.66 | **98.25** | 95.32 | 97.66 | 92.98 | 91.81 | 97.08 | **98.25** |
| SVM | 97.08 | 95.91 | 98.25 | 95.91 | 96.49 | 92.98 | **98.83** | 95.91 |
| RF | 96.49 | 94.74 | 95.91 | 93.57 | 94.15 | 91.23 | 95.91 | **96.50** |

Table 4.3: Feature Selection experiment. Accuracy in Breast Cancer Wisconsin data set.

Tables 4.1, 4.2 and 4.3 illustrate the power of RFs variable importance measures. The MDA and the MDI using the Gini index usually head the set of best Features Selection techniques, even being leader in some cases. It seems that the MDI usually performs a little better than the MDA for these data sets.

# Bibliography


[1] Yali Amit and Donald Geman. Shape quantization and recognition with randomized trees. *Neural computation*, 9(7):1545–1588, 1997.

[2] Gérard Biau and Erwan Scornet. A random forest guided tour. *Test*, 25(2):197–227, 2016.

[3] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[4] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] Leo Breiman. Manual on setting up, using, and understanding random forests v3.1. https://www.stat.berkeley.edu/b̃reiman/using_random_forests_v3.1.pdf. 2003a.

[6] Leo Breiman. Manual on setting up, using, and understanding random forests v4.0. https://www.stat.berkeley.edu/b̃reiman/using_random_forests_v4.0.pdf. 2003a.

[7] Leo Breiman and Adele Cutler. Random forests-classification description. https://www.stat.berkeley.edu/ breiman/randomforests/interface04.pdf. *Department of Statistics, Berkeley*, 2, 2007.

[8] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[9] Emilio Carrizosa and Dolores Romero Morales. Supervised classification and mathematical optimization. *Computers & Operations Research*, 40(1):150–165, 2013.

[10] Bojan Cestnik, Igor Kononenko, Ivan Bratko, et al. Assistant 86: A knowledge-elicitation tool for sophisticated users. In *EWSL*, pages 31–45, 1987.

[11] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[12] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[13] Trevor F Cox and Michael AA Cox. *Multidimensional scaling*. CRC press, 2000.

[14] Howard B Demuth, Mark H Beale, Orlando De Jess, and Martin T Hagan. *Neural network design*. Martin Hagan, 2014.

[15] Ramón Díaz-Uriarte and Sara Alvarez De Andres. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):3, 2006.

[16] Thomas G Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine learning*, 40(2):139–157, 2000.

[17] Bradley Efron and Robert J Tibshirani. *An introduction to the bootstrap*. CRC press, 1994.

[18] Cristofer Englund and Antanas Verikas. A novel approach to estimate proximity in a random forest: An exploratory study. *Expert Systems with Applications*, 39(17):13046–13050, 2012.

[19] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[20] Jerome H Friedman. A recursive partitioning decision rule for nonparametric classification. *IEEE Trans. Comput.*, 26(SLAC-PUB-1573-REV):404, 1976.

[21] Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence*, 20(8):832–844, 1998.

[22] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[23] Earl B Hunt, Janet Marin, and Philip J Stone. Experiments in induction. 1966.

[24] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 6. Springer, 2013.

[25] Gordon V Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, pages 119–127, 1980.

[26] Igor Kononenko, Ivan Bratko, and Esidija Roskar. Experiments in automatic learning of medical diagnostic rules. Technical report, Technical Report, Jozef Stefan Institute, Ljubljana, Yugoslavia, 1984.

[27] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007.

[28] Patrice Latinne, Olivier Debeir, and Christine Decaestecker. Limiting the number of trees in random forests. In *International Workshop on Multiple Classifier Systems*, pages 178–187. Springer, 2001.

[29] Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

[30] James N Morgan and Robert C Messenger. Thaid, a sequential analysis program for the analysis of nominal scale dependent variables. 1973.

[31] James N Morgan and John A Sonquist. Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58(302):415–434, 1963.

[32] Kevin P Murphy. Naive bayes classifiers. *University of British Columbia*, 2006.

[33] A Patterson and T Niblett. Acls user manual, 1983.

[34] JR Quilan. Learning efficient classification procedures and their application to chess end games. *Machine Learning: An Artificial Intelligence Approach*, 1, 1983.

[35] J Ross Quinlan. *C4.5: programs for machine learning*. Elsevier, 2014.

[36] J Ross Quinlan et al. *Discovering rules by induction from large collections of examples*. Expert systems in the micro electronic age. Edinburgh University Press, 1979.