FINAL DEGREE PROJECT

Random Forests: Properties and applications

Presented by:
Fátima del Pilar Villalba Pizarro

Supervised by:
DR. EMILIO CARRIZOSA PRIEGO

FACULTY OF MATHEMATICS
Statistics and Operational Research Department
Seville, September 2020

Índice general

Resumen

Los bosques aleatorios son una herramienta fundamental en el ámbito del aprendizaje supervisado, por lo que son empleados en multitud de disciplinas, demostrando grandes resultados y múltiples ventajas en problemas de clasificación y regresión.

En este trabajo se exponen los fundamentos y bases de los bosques aleatorios, destacando sus ventajosas características, para posteriormente hacer diferentes aplicaciones con ellos, con las que contrastar dichas ventajas, observar distintos aspectos que se harán patentes e incluso analizar su comportamiento cuando extendemos su uso al caso de los datos funcionales.

Abstract

Random forests are considered a fundamental tool in supervised learning. Consequently, random forests are used in a wide range of disciplines, yielding great results and demonstrating many advantages in classification and regression problems.

Along this work, the bases and foundations of random forests are exposed, emphasizing their advantageous properties, to subsequently make different applications with them, demonstrating their advantages, analyzing distinct aspects that become noticeable, and even studying the way they behave when extending their use to functional data.

Introducción

En un mundo en constante avance y desarrollo, es importante usar técnicas que modelen la realidad de manera eficiente. Muchos problemas de regresión y clasificación se pueden resolver mediante aprendizaje supervisado, es decir, mediante una técnica en la que, dada una muestra con distintos datos en forma de pares, uno de entrada (normalmente un vector) y otro de salida, se les atribuye un valor numérico o clase a cada par, con el fin de poder predecir la salida de futuros datos. Este método ha ido cobrando importancia en los últimos años tomando un papel relevante como puede verse en la Figura 1, que nos muestra el interés suscitado por este tema en relación con el número de búsquedas del término “aprendizaje supervisado" (en ingles “supervised learning") en *Google*.

Figura 1: Evolución del número de búsquedas del término “aprendizaje supervisado" (supervised learning) en Google desde 2004

Una herramienta particular de aprendizaje supervisado es la conocida como bosques aleatorios, o del inglés, random forests, desarrollados en el año 2001 por Leo Breiman [6]. Esta técnica se usa en distintas áreas como son la econometría [20], la quimioinformática [18], la bioinformática [10] o la medicina. En cuanto a esta última, se tiene que los bosques aleatorios son útiles en diferentes campos como son la selección de marcadores genéticos responsables de enfermedades, la microbiología o la epidemiología genética, e incluso para predecir la replicación de un virus como el HIV-1 [17].

La relevancia de los bosques aleatorios se debe a que, a diferencia de otras técnicas, los bosques aleatorios son eficaces incluso cuando se trabaja con variables con bastante

“ruido" o cuando el número de variables es muy grande, aún cuando éste supera al número de observaciones. Además, los bosques aleatorios nos dan una medida de la importancia de las distintas variables. En virtud de todas las ventajas que presentan los bosques aleatorios, éstos han sido descritos como “el algorítmo más éxitoso de los tiempos modernos" [3]. En la Figura 2 se puede ver el interés suscitado por los bosques aleatorios en los últimos dieciséis años en relación con las búsquedas en *Google* del término en inglés “random forests".

Figura 2: Evolución del número de búsquedas del término “bosques aleatorios" (random forests) desde 2004

Centrando el foco de atención de forma individualizada sobre los diststintos países, se observa en la Figura 3(a) que en China, una de las mayores potencias mundiales, el interés generado por los bosques aleatorios en el último año es máximo. En la Figura 3(b) se observa que también en Madrid, epicentro de España, este interés se ha maximizado en este último año.

(a) China (b) España

Figura 3: Interés en los diferentes países

Debido a todas las razones arriba expuestas, el tema de los “bosques aleatorios" ha sido elegido como campo de análisis de este trabajo de fin de grado.

Este trabajo está estructurado como sigue: En el Capítulo 1, se explica la forma de operar de los bosques aleatorios desde un punto de vista teórico, a partir de los

árboles de decisión, para posteriormente, en el Capítulo 2 profundizar en una de las mayores características de los bosques aleatorios, la importancia de las variables. Por último, pero no por ello menos importante, en el Capitulo 3, se implementarán los bosques aleatorios en `R`, realizando distintos experimentos, engrosando el rango de aplicaciones de los bosques aleatorios a los datos funcionales.

Introduction

In a world of continuous advancement, it is important to use techniques that model real problems efficiently. Many regression and classification problems can be solved by supervised learning, i.e. a strategy that, given a sample with input and output data, labels this information in order to predict the output of forthcoming data. This technique has taken in the last years an increasingly important role as it is shown in Figure 4, which shows the interest in relation with the number of searches of the term "supervised learning" in *Google*.

Figure 4: Supervised Learning interest in Google since 2004

A particular tool of supervised learning is the one developed by Leo Breiman (2001) [6], called **random forests**. They are used in many areas like econometric [20], chemoinformatic [18], bioinformatic [10] and medicine. For instance, in the last one it can help in important work-streams, such as selecting genetic markers responsible for a disease, in genetic epidemiology and microbiology or to predict the replication of viruses like HIV-1 [17].

Unlike other techniques, random forests are that relevant because they work successfully even when predictive variables are noisy or when the number of variables is bigger than the number of observations. Moreover, another positive aspect of random forest is the fact that it can measure the importance of variables [10]. Because of all their advantages, random forests have been described as “the most successful general-purpose algorithm in modern times” [3]. As it was already done for supervised learning, the interest aroused by the term "random forests" according to the searches of the item in *Google* can be seen in Figure 5.

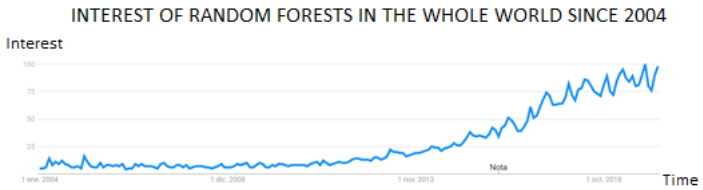


Figure 5: Evolution of the number of searches in the term "Random Forests" in Google since 2004

Focusing now on specific countries in Figure 6, it is concluded what follows. On a) it stands out the interest on random forests (in the last year) in China, one of world's most powerful countries, is maximum. On b) it is shown that in the epicenter of Spain, Madrid, has also maximized, over the last 12 months.

(a) China (b) Spain

Figure 6: Interest in different countries

Because of every reason portrayed above the topic "random forests" has been selected as the field of analysis for this final degree project.

This work is structured as follows. First, in Chapter 1, the main ingredients of the random forests are outlined, starting with the simplest case of a single decision tree, in order to focus in Chapter 2 on one of the main characteristics of random forests, namely, the measure of the importance of variables. Last but not least, in Chapter 3, random forests are implemented in `R`, expanding their use also to functional data.

Chapter 1

From CART to random forests

"Divide and conquer" [3]. This is the main idea that stands behind the term random forests. We just need to focus on the suitable elected name of the method, to understand the relation with this principle. Random forests yield a compound of two fundamental elements, consisting of an ensemble of decision trees sampled independently (but working together -forest-) that could also remember to a swarm, where randomness is applied at different points of the algorithm.

That is why in order to understand random forests, the way single decision trees work, must be first explained.

1.1. CART (Classification and Regression Trees)

Sometimes, in order to understand how things work it is important to ask the typical W-questions: When? Who? Why? This is the reason why we are going to focus first at the birth of the CART.

In what follows, an introduction to CART, based on [8], is presented.

In 1984 Breiman and Friedman published, in collaboration with Stone and Olshen among others, a book homonymous to this method: "Classification and Regression Trees". Until that date the biggest advance in classification programs, THAID, had been made in 1973 by Morgan and Messenger[14]. Motivated by the need of dealing with actual problems, Breiman and Friedman tried to improve this area with the participation of the other mentioned scientist. Particularly, Olshen contributed with a vision of this need in his own area, medicine. They exposed a clear example where decision trees could be helpful. "When a heart attack patient is admitted, 19 variables are measured during the first 24 hours. This includes blood pressure, age, and 17 other ordered and binary variables summarizing the medical symptoms considered as important indicators of the patient's condition". Here is where CART can be implemented and surpass its ancestor. [8]

Decision trees are considered leaders in their area because of their easy interpretability. The reason therefore resides in the scheme that follows such a tree: an if-then rule. [9]

1.1.1. Structure of decision trees

Let us explain now how decision trees work. We will begin with classification the trees and then go on with an in depth review of the regression trees.

Decision trees can handle any type of variables, categorical (i.e., they take values in a finite set not having any natural order) or numerical/continuous (i.e. it is a real number).

Let us see first which is the structure of such a tree.
A classification tree is a collection of nodes and branches that display a partition of the original set. Let t_0 be such original set. This set can be split into a compound of sets that form a partition of t_0. For simplicity, let us think we are working with the simplest type of tree, the binary tree, which means at each split two new sets are obtained. If we name the new sets obtained from t_0, t_1 and t_2 it is fulfilled $t_0 = t_1 \sqcup t_2$. This process is repeated with such new sets.

All these sets are called nodes. Three types of nodes are found:

- The original set t_0, called **root node**.
- The subsets that are not split, called **terminal nodes**. These form a partition of t_0. Each terminal node is assigned a class label, and it is possible to find more than just one terminal node with the same class label.
- The remaining nodes, called **nonterminal nodes**.

Such structure of a decision tree can be seen in Figure 1.1.

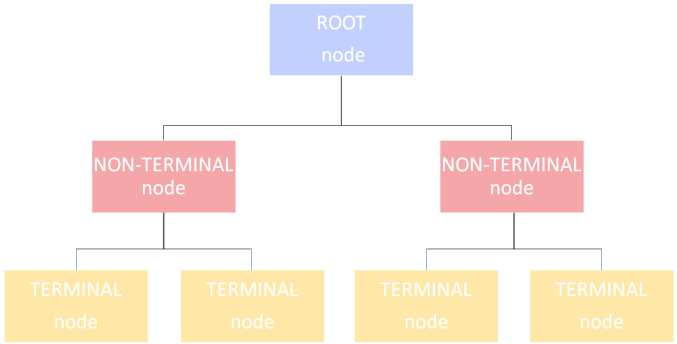


Figure 1.1: Strcture of a decision tree

Marital status	Age	Working
Married	28	Yes
Single	37	No
Divorced	42	No
Married	41	Yes

Table 1.1: Example of learning sample $\mathcal{L}$

1.1.2. Construction of decision trees

For the construction of the classification tree, a learning sample must be given. Let $\mathcal{L} = \{(\mathbf{X}_i, Y_i)_{i=1,..,N}\}$ denote the learning sample, where N is the number of observations, $\mathbf{X}_i \in \mathcal{X}$ a vector of M variables, named measurement vector, in the measurement space $\mathcal{X}$, and Y_i the corresponding observation from among K possible classes $C = \Big\{C_1, ..., C_K\Big\}$.

Example 1.1.1. *An example of what is explained above can be seen in Table 1.1, where it can be identified:*

- $N = 4$, $M = 2$, $K = 2$
- X^1 *is the variable* $Maritalstatus$ *that takes values in* $\Big\{Married, Single, Divorced\Big\}$
- X^2 *is the variable* Age *that takes values in* $\Big\{18, 19, 20, \dots\Big\}$
- $\mathcal{L} = \Big\{(X_1^1 = Married, X_1^2 = 28, Y_1 = Yes), (X_2^1 = Single, X_2^2 = 37, Y_2 = No), (X_3^1 = Divorced, X_3^2 = 42, Y_3 = No), (X_4^1 = Married, X_4^2 = 41, Y_4 = Yes)\Big\}$

According to the given learning sample the classification tree can be built. The construction of a tree is based on four points:

1. Topology and type of splitting
2. Selection of splits
3. Declaration of terminal nodes/Stop-splitting rule

4. Designation of class label at terminal nodes

The main goal is constructing the classification tree from the training set $\mathcal{L}$.
We now outline some details of the above mentioned elements.

1.1.2.1. Topology and type of splitting

According to the number of new nodes resulting from the splitting, two types of splitting can be distinguished: binary splitting, i.e., there are obtained two new nodes, or multi-splitting, that induces more than two new nodes. For continuous variables the second one is not really useful. It can also be set a difference between two types of splits according to the number of variables involved in the splitting. If only one variable takes part in the split, it is called univariate split, otherwise, it is named multivariate split. In what follows, just binary trees, with univariate split will be considered.

1.1.2.2. Spliting rules

In this section we will assume we are working with a standard structure, i.e., all measurement vectors $\mathbf{X}_i$ are of fixed dimensionality.

Many different criteria of splitting have been adopted. Out of then, Leo Breimann highlights two, namely:

- Gini criterion
- Twoing criterion

The Gini criterion is the one that is going to be explained, because it is also the one that uses the program `R` that is going to be used in the last chapter. Before going down to the explanation of how this criterion operates, we must first introduce a few fundamental concepts in two initial steps, that will converge in the third and last step.

1. Explanation of concepts related with probability
2. Introduction of sets of all possible splitting
3. Explanation of selecting the best split: Gini criterion

Probability concepts

Beginning with the first step: Remember we said we had the training sample $\mathcal{L} = \left\{(\mathbf{X}_i, Y_i)\right\}_{i=1,..,N}$, with $\mathbf{X}_i \in \mathcal{X}$ and dimension M, and $Y_i \in C = \left\{C_1, ..., C_K\right\}$.

The prior class probabilities $\pi(k)$, $1 \leq k \leq K$ is defined as:

$$\pi(k) = P(Y_i = C_k).$$

Let N_k be the number of cases of $\mathcal{L}$ in class C_k. Often the prior probabilities can be assessed like:

$$\pi(k) = \frac{N_k}{N}, 1 \leq k \leq K.$$

Suppose now we are at node t. Maintaining the same idea as above, let $N(t)$ be the total number of observations in $\mathcal{L}$ that have reached the node t, and let $N_k(t)$ denote the number of observations of class C_k in t. So, the quotient

$$\frac{N_k(t)}{N_k}$$

is interpreted as the proportion of observations of class C_k falling into t.

Using both definitions, we obtain the probability of an observation in $\mathcal{L}$ fulfilling two aspects simultaneously: falling into t and being of class C_k, given by

$$p(k,t) = \pi(k)\frac{N_k(t)}{N_k}.$$

If we sum in k, we obtain the probability of an observation of any class C_k reaching the node t

$$p(t) = \sum_k p(k,t),$$

and we can define now the probability the observation is of class C_k subject to having reached node t

$$p(k|t) = \frac{p(k,t)}{p(t)}$$

satisfying:

$$\sum_k p(k \mid t) = 1$$

According to the approach done for $\pi(k)$ we can also assume that

$$p(k|t) = \frac{N_k(t)}{N(t)}. \tag{1.1}$$

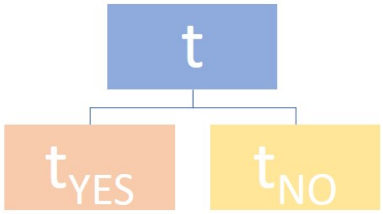


Figure 1.2: Parent node and descendant nodes

Splitting sets

Let us now continue with the second step and talk about two necessary sets Q and S. Supposed we want to do univariate binary splitting, just one single variable X^m, that matches with a coordinate of the measurement vector $\mathbf{X}$, is responsible for the split. In case X^m is categorical taking values in the set $B = \{b_1, \ldots, b_H\}$, we can define a set Q like

$$Q = \left\{ \text{Questions} \mid X_i^m \in A? \right\}, A \subset B$$

and in case X^m is a continuous variable:

$$Q = \left\{ \text{Questions} \mid X_i^m < c? \right\}.$$

This way, every question of Q suggests a possible split, as depicted in Figure 1.2.

With regard to the observations that have reached the node t, the question of Q has to be formulated, and there are just two possible answers: YES or NO. Depending on the answer, the observation reaches now the node to the left, let us call it $t_{left} = t_{YES}$, or the one to the right, namely $t_{right} = t_{NO}$.

Each question generates one possible split. The set of splits is the one called S. If the variable X_i^m is categorical, then there are $2^{H-1} - 1$ possible splits, because the number of possible splits is given by the number of possible questions, i.e., by $\mid Q \mid$, taking into account that $t_{left} = t_{YES}$ and $t_{right} = t_{NO}$ is symmetrical to $t_{left} = t_{NO}$ and $t_{right} = t_{YES}$, in other words, $X_i^m \in A$ is symmetrical to $X_i^m \notin A$. It is known that X_i^m takes values in $B = \{b_1, \ldots, b_H\}$ so $\mid B \mid = H$, and thus the number of possible subsets of B is 2^H, but it must be considered the symmetry above explained, and it must also be considered that $\emptyset$ (and by symmetry also the total) have to be factored out. So it follows that the number of questions in Q is $\mid Q \mid = 2^{H-1} - 1$, and therefore the number of possible splits.
In case the variable X_i^m is continuous, contrary to what may seem, the number of

distinct splits is also finite. There are at most as many splits as number of observations N, because this type of splits:

1. Consider all the distinct values of X_i^m that appear in $\mathcal{L}$
2. Sort them
3. Take c_i halfway between two of the ordered values.

Gini criterion

Next action is already the third step, and it consists in selecting the best split, among all the possibilities. Hence, we need to introduce two new concepts: *impurity function* and *impurity measure*.

Let us begin by defining the *impurity function*.

Definition 1.1.1. *A function $\phi : P \to \mathbb{R}$, where P is defined as $P = \left\{(p_1, \ldots, p_K) \mid \sum_k p_k = 1, p_k \geq 0, k = 1, .., K\right\}$, is said to be an impurity function if it satisfies the following properties:*

1. *ϕ achieves its maximum only at the point $(1/K, \ldots, 1/K)$*
2. *ϕ achieves its minimum at the points of the form $(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0)$, $\ldots, (0, \ldots, 0, 1)$*
3. *ϕ is a symmetric function of $p_1, \ldots, p_K$.*

Let us continue now explaining the *impurity measure* of a node t, $i(t)$. It is defined as (remember we defined $p(k \mid t)$ in (1.1) as the probability of the observation being of class C_k subject to have reached node t):

$$i(t) = \phi(p(1 \mid t), \ldots, p(K \mid t))$$

From the above, the decrease in impurity of a split s in a node t is defined as:

$$\Delta i(s, t) = i(t) - p_{right} i(t_{right}) - p_{left} i(t_{left}) \tag{1.2}$$

where p_{right} and p_{left} are the proportion of items of the node t sent by the split s to the node t_{right} and to the node t_{left}, respectively.

Now, once the impurity concepts have been explained, according to the probability concepts exposed above, we can finally talk about the criterion we mentioned before: **Gini Criterion**.

This criterion uses the impurity measure of a node t described by the following formula, called **Gini Index**:

$$\sum_{k \neq l} p(k \mid t) p(l \mid t)$$

or what is the same:

$$i(t) = (\sum_{k} p(k \mid t))^2 - \sum_{k} p^2(k \mid t) = 1 - \sum_{k} p^2(k \mid t). \tag{1.3}$$

Remember: $p(k \mid t)$ refers to the probability of assigning an item selected at random from node t to class C_k, and $p(l \mid t)$ expresses the probability of the object belonging to the class C_l.

Using this impurity measure, it must be calculated now the decrease in impurity $\Delta i(s,t)$ using (1.2) and (1.3). So,

$$\Delta i(s,t) = 1 - \sum_{k} p^2(k \mid t) - p_{right} \left[1 - \sum_{k} p^2(k \mid t_{right}) \right] - p_{left} \left[1 - \sum_{k} p^2(k \mid t_{left}) \right]$$

$$\Delta i(s,t) = - \sum_{k} p^2(k \mid t) + p_{right} \sum_{k} p^2(k \mid t_{right}) + p_{left} \sum_{k} p^2(k \mid t_{left})$$

The aim is to maximize $\Delta i(s,t)$. The better split is the one that achieves it.
Any split s verifies

$$\Delta i(s,t) \geq 0.$$

But, it can be demonstrated that $\Delta i(s,t)$ is a concave function. Hence, $\Delta i(s,t) = 0$ if and only if

$$p(k \mid t_{left}) = p(k \mid t_{right}) = p(k \mid t), k = 1, \ldots, K.$$

1.1.2.3. Stop-splitting rule

Once the splitting rule has been selected, the next action is to decide the stop-splitting rule, i.e, a rule that indicates when to declare a node a terminal node.

One of the initial stopping rules was based on fixing a threshold, call it $\beta > 0$, for the maximum **decrease in tree impurity**.

Definition 1.1.2. *The tree impurity $I : \mathcal{T} \longrightarrow \mathbb{R}$, where $\mathcal{T}$ is a set of trees, is a function defined from the impurity measure i as:*

$$I(T) = \sum_{t \in T'} I(t)$$

where T' is a set of terminal nodes achieved after some splitting. Let $I(t) = i(t)p(t)$ then the tree impurity is:

$$I(T) = \sum_{t \in T'} I(t) = \sum_{t \in T'} i(t)p(t)$$

So, the decrease in tree impurity is given by:

$$\Delta I(s,t) = I(t) - I(t_{right}) - I(t_{left})$$

Hence, the early stop-splitting is:

$$\max_{s \in S} \Delta I(s,t) < \beta$$

Nevertheless, this rule was shown not to be fully satisfactory and instead, the tree should be pruned, once it has grown much too large.

Therefore is also important to explain what "grown much too large" means. Hence, let us explain, before going on, three new concepts: **misclassification cost** $\mathbf{C(k \mid l)}$, **resubstitution estimate of the expected missclassification cost of a node** $\mathbf{t}$, $\mathbf{r(t)}$, and the **resubstitution estimate of the missclassification cost of a tree** $\mathbf{T}$, $\mathbf{R(T)}$.

We are going to tackle these question from a general perspective and particularize for the specific case of classification trees.

Definition 1.1.3. *Let d be a function $d : \mathcal{X} \to C$, called classifier. The **true misclassification rate of the function** d, denoted $R^*(d)$, constructed from the learning sample $\mathcal{L}$, indicates how accurate a classifier is, i.e., the probability of d misclassifying a new sample drawn from the same distribution as $\mathcal{L}$, by testing the classifier on subsequent cases whose correct classification has been observed. This function is given by:*

$$R^*(d) = P(d(\mathbf{X}) \neq Y)$$

where $\mathbf{X} \in \mathcal{X}, Y \in C$.

This means that a good classificator has a low value of $R^*(d)$. It follows that the prunning criterion must minimize $R^*(d)$.

The question is how can $R^*(d)$ be estimated. One of the techniques is using the **resubstitution estimate**.

The **resubstitution estimate** is the proportion of cases misclassified, and is given by:

$$R(d) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{X}(d(\mathbf{X}_i) \neq Y_i)$$

where $\mathcal{X}$ is the indicator function, which has value equal to 1 if the argument is true and 0 if it is false.

It is used $\mathcal{L}$ to construct d and also in order to calculate $R(d)$. So, if we use the value of $R(d)$ to estimate $R^*(d)$, the result will be unrealistic good, even being possible that the value of $R(d) = 0$ and so $R^*(d) = 0$, too. One way of enhancing the estimation, in case the learning sample $\mathcal{L}$ is large enough, is to divide $\mathcal{L}$ into two subsets $\mathcal{L}_1$ and $\mathcal{L}_2$, so that with $\mathcal{L}_1$ we construct d and with $\mathcal{L}_2$ calculate $R(d)$. The second subset $\mathcal{L}_2$ is called *test sample*. $\mathcal{L}_2$ can be considered independent of $\mathcal{L}_1$ and from the same distribution. However for samples $\mathcal{L}$ of small size, the cross-validation method can be used [16].

Now we can use what is explained above in the case of classification trees.

Definition 1.1.4. *$C(k \mid l)$ is defined as the cost of misclassifying, as class C_k, an item that indeed corresponds to class C_l, satisfying:*

$$C(k \mid l) \begin{cases} \geq 0 & if \quad k \neq l \\ = 0 & if \quad k = l \end{cases}$$

Definition 1.1.5. *The resubstitution estimate of the expected misclassification cost of a node t, $r(t)$ is defined as:*

$$r(t) = \min_k \sum_l C(k \mid l) p(l \mid t),$$

where $\sum_l C(k \mid l) p(l \mid t)$ is the estimated expected misclassification cost of an unknown class item that reaches node t and is classified as class C_k.

Definition 1.1.6. *The resubstitution estimate of the misclassification cost of a tree T, $R(T)$ is given by:*

$$R(T) = \sum_{t \in T'} r(t) p(t) = \sum_{t \in T'} R(t)$$

where $R(t) = r(t)p(t)$

The objective is to minimize the value of $R(T)$. Therefore an equilibrium between two opposing properties must be found:

1. If the number of splits increases, then the value of $R(T)$ decreases. In other words, if the number of terminal nodes increases, the value of $R(T)$ decreases.

2. A tree that is grown much too large, so that just one item of the learning sample reaches that terminal node, is overfitted, i.e. it classifies perfectly the given learning sample, but it is likely it does not classify correctly a new sample.
   If the number of terminal nodes increases *too much*, $R(T)$ increases.

Once it has been understood how to measure "how good" a tree is, we can now explain the procedure that must be follow in order to prune the tree properly.

First of all, a tree must grow until all terminal nodes are pure: In this first step, the tree must grow until, for every terminal node, it is satisfied that only one object has reached the terminal node in question. Let us call this tree T_{max}. Next step is pruning upward, i.e., cut off nodes successively. The final step consists of choosing from the collection of subtrees formed in the previous step, the "optimum-sized" tree.

Observation 1.1.1. *In fact, a value N_{min} (in general it takes the values 1 or 5) can be set, and the tree grows just until $N(t) \leqslant N_{min}$ is fulfilled. (Remember: $N(t)$ is the total number of observations in $\mathcal{L}$ that have reached the node t.)*

1.1.2.4. Designation of class label at terminal nodes

Now it is time to assign a class label at all the terminal nodes of the tree. Therefore the **class assignment rule** $\mathbf{C_k^*(t)}$ is used, where $C_k^*(t)$ denotes that C_k is the class given to the terminal node t. The rule is:

$$C_k^*(t) = C_k \quad \text{for the k satisfying} \quad p(k \mid t) = \max_l p(l \mid t)$$

If the maximum is attained at more than one value of k, any of such k can be taken.

1.1.3. Confusion matrix and accuracy

Until this moment it has been explained how to construct a classifier type called classification tree. The goodness of fit of the method is defined through a confusion matrix and the accuracy. The accuracy measures the proportion of well-classified items, so a high accuracy indicates that the classifier is good. **The higher the accuracy, the better the classifier**. Assume there are two possible classes: positive and negative. When predicting the class of an observation, there are four possible endings:

1. Predict class positive, being the truth class positive
2. Predict class positive, being the truth class negative
3. Predict class negative, being the truth class negative
4. Predict class negative, being the truth class positive

The four cases receive following names: true positive (TP), false positive (FP), true negative (TN) and false negative (FN), respectively.

		Predicted class	
		Positive	Negative
Real class	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

Table 1.2: Confusion matrix

This information can be set in form of a matrix, called confusion matrix, like the one in Table 1.2.

From the information of the confusion matrix, can the accuracy be finally calculated as:

$$Accuracy = \frac{True_{Positive} + True_{Negative}}{True_{Positive} + True_{Negative} + False_{Positive} + False_{Negative}}$$

Example 1.1.2. *Say being sick with COVID-19 means being positive, being healthy means being negative. When a test is done to a person that is infected with coronavirus and the test results positive, we have a true positive, otherwise the test has given a false result, it has given a false positive. Likewise for the healthy (negative) people. In this case it is more important to have a high true positive rate than a low false positive rate, but this depends on the study that is being done.*

1.1.4. Regression Trees

The discussion above is easily adapted to the case in which a regression, instead of a classification, problem is addressed. The reader is referred to [8].

1.2. Random forests

Random forests were introduced by Leo Breiman in 2001 [6] as an improvement of decision trees. One of the limitations of single decision trees is their weakness against even small perturbations of the training sample, responsible for a complete change of the resulting predictions [2]. Nevertheless, random forests overcome this obstacle and shine for their simplicity of use, accuracy and ability to deal whit samples of small size and high-dimensional feature spaces [3]. There is just one disadvantage by contrast with the classification trees: the greatful interpretation of CART gets partially lost. This will stand out, as soon as the operational method of random forest is explained.

1.2.1. Construction of random forests

The main elements of random forests are:

1. CART (In order to see how it works, see Section 1.1); and
2. Randomness.

Now let us see how these elements are used. As already explained, random forests are an ensemble of trees where randomness is introduced. Although these trees are constructed similar to the ones explained in Section 1.1, they are not exactly the same.

1.2.1.1. Bagging(Bootstrap-aggregating)

One of the main differences between the trees of random forests and the ones of Section 1.1, appears even in the beginning of the construction method: instead of using the whole learning sample in order to let each tree grow, the **bootstrap** method is used. Bootstrap consists in fixing for each tree from the original learning sample a second data set, named *bootstrap* of the same size as the original one. This is done taking **randomly** N items of the learning sample $\mathcal{L}$, in general, with replacement.
Once the bootstrap sample is created, there are some elements of the original data set left, that do not appear in the bootstrap with approximately one third of probability.[3]

The reason why such probability is about one third is quite simple: Starting from the learning sample $\mathcal{L}$ of size N, there must be selected N items with replacement. The aim, in order to be able to answer the question, is to calculate the probability of not selecting in this random process the $j-th$ observation. The probability of not selecting the mentioned item the first time is

$$\frac{N-1}{N}.$$

So, it follows that the probability of not selecting this observation in the N selections with replacement is

$$\left(\frac{N-1}{N}\right)^N.$$

This matches with a binomial distribution:

- $Z_j =$:random variable that counts how many times the observation $j-th$ is chosen to build the bootstrap
- $p =$: is the probability of coming out in one choice
- $q =$: is the probability of not coming out in one choice $q = \frac{N-1}{N}$

- $k =$: number of times of coming out

So, $P(Z_j = k) = \binom{N}{k} p^k q^{N-k}$ and taking $k = 0$ and q like already said:

$$P(Z_j = 0) = \left(\frac{N-1}{N}\right)^N$$

In the limit *(it turns from the binomial to the Poisson distribution)*:

$$\lim_{N\to\infty} \left(\frac{N-1}{N}\right)^N = e^{-1} \approx 0.368$$

what is approximately one third.

The non selected objects form the so called **OOB (out-of-bag)**. This set is used to estimate the error of the classifier [5].

The bootstrap method is used for every individual tree and then it is made the average of the different results of all trees. This is called **aggregating**, and both things together conform the **bagging** (contraction of **bootstrap-aggregating**)[3].

Observation 1.2.1. *It is worth to mention that some authors propose replacing the bagging for subagging in order to improve random forests. Subagging is a term coined by Bülmann and Yu, that means subsample aggregating[17][3], where subagging must be understood as bagging, but with the difference that instead of constructing a bootstrap sample, the new sample, called subsample, is of a smaller size than the original one.*

1.2.1.2. Randomness in splitting

Moreover, continuing with the building of the trees used in random forests, we find another difference to the ones presented in Section 1.1. Randomness is not only used in the bootstrap, but also in the splitting, because instead of constructing the trees according to the splitting criterion of the CART, the trees grow considering randomness for selecting $mtry$ variables from the total number of variables and choose then the best split. [7].

Observation 1.2.2. *The use of randomness in both above mentioned steps 1.2.1.1 Bagging,1.2.1.2 Randomness in splitting, helps to de-correlate the trees from each others, and so to enhance the accuracy[1].*

1.2.1.3. Unprunning

Another of the main differences between CART and the trees used in random forests is that the second ones are not pruned [17], [2].

Once the trees have been constructed, the random forest is built and can be used to classify a new item $\mathbf{X}$. Therefore every tree must provide its own classification, and the random forest classifies the object accordingly to that class that has achieved more goals.

Let us now schematize the construction of random forests with the information obtained until this moment.

1. Construct a tree
   - *a*) Make a bootstrap: This acts like a learning sample for this tree
   - *b*) Select $mtry$ variables
   - *c*) Select best split
2. Repeat step 1 for F trees

Once the construction of random forests is clear, let us now focus on the different properties such a method offers.

1.2.2. Proximity matrix

Similar samples will take similar routes along the decision trees that conform the random forests. That is the reason why the **proximity matrix** is of relevance. This matrix gives a measure of "proximity", i.e. of similarity between the different observations and this can be used afterwards in order to replace missing values according to a weighted sum of the proximities of the non-missing values.
Moreover, such a proximity matrix is also useful in order to detect outliers, as we will see later.

The big question now is, how is the proximity matrix built? The process is quite simple. The proximity matrix $Prox$ is a NxN matrix where each cell represents the proximity between the $i-th$ and the $j-th$ observation. Let us now explain the construction of each value. The whole learning sample, including the bootstrap and the OOB, both, must go down, one by one, all the trees that make up the forest. Every time the $i-th$ and the $j-th$ items end up in the same terminal node of a tree, the value of their common entry $Prox(i,j)$ increases by 1 unit. Once the observations have been drop down all the trees, the proximity matrix must be normalized, i.e., each cell must

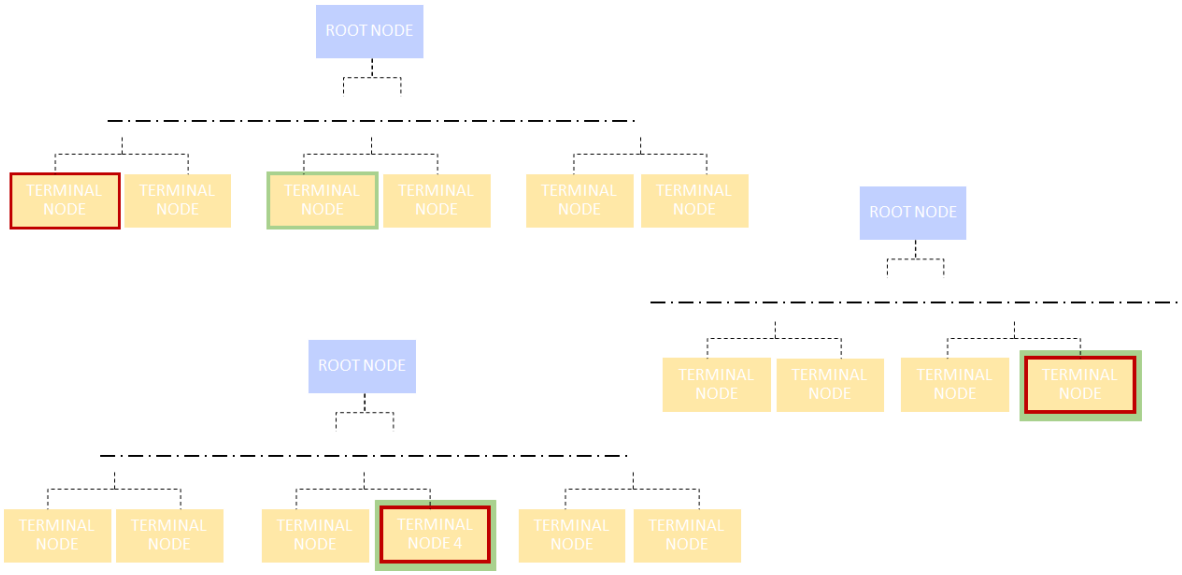


Figure 1.3: Random forest to construct proximity matrix

be divided by the total number of trees in the forest, F [7][1]. Thus, the proximity matrix is a symmetric matrix $Prox(i,j) = Prox(j,i)$, with all its entries between 0 and 1, where in particular all the cells of its diagonal are equal to one $Prox(i,i) = 1$. When $Prox(i,j) = 1$ we have that both observations are as similar as they could be, and on the contrary, if the value is zero, both observation are non similar, because they have never ended in any tree at the same terminal node. An illustrative way of representing the proximity matrix is using a heatmap.

Example 1.2.1. *Consider the forest consisting of the trees shown in Figure 1.3 Of course this would be in practice a too small forest, but let us take it just an explanatory scenario. If the red-box marks the terminal node where the $i-th$ observation ends, and the green one, where the $j-th$, then they end two times at the same terminal node. So $Prox(i,j) = Prox(j,i) = 2$, and dividing by the number of trees $F = 3$ we finally get $Prox(i,j) = Prox(j,i) = \frac{2}{3}$.*

1.2.2.1. Dissimilarity matrix [13][7]

In case we are not looking for the "clossness", i.e. how similar the observations are, but we want to know how "unalike" they are, the dissimilarity matrix has to be constructed. The entries of the dissimilarity matrix represent the "squared distances in a Euclidean space of dimension not greater than the number of cases"[7].

The dissimilarity matrix D is obtained from the proximity matrix as:

$$D(i,j) = 1 - Prox(i,j), i,j = 1,..,N.$$

D is symmetric, positive-definite and the cells of its diagonal are all zero.

Observation 1.2.3. *[13] Some authors also take the dissimilarity matrix as the proximity matrix, because in the end, both measure the "distance" between pairs of data. It is just important to declare clearly whether the proximity matrix is measuring similarity or dissimilarity.*

1.2.2.2. Outliers

As already mentioned, proximity matrices are used, among others, to locate outliers, i.e., observations that are **far** away from the rest. The word *far* has been emphasized. The reason is that, exactly that term, is the one that gives the solution to detect outliers. *Far* is an expression of distance, of proximity. Hence, outliers can be located using the proximity matrix. Breiman and Cutler proposed a measure of outlyingness of an observation in relation with the remanding data of the same class as the item in question [13]:

$$u_{i,k} = \frac{N}{\sum_{j \neq i} [Prox(i,j)]^2}, j = 1..., N, k = 1, .., K$$

where $Prox(i,j)$ represents the entry of row i and column j of the proximity matrix, i.e. the proximity between the $i-th$ and the $j-th$ observation, and k denotes the class C_k, to which both data correspond.
In order to normalize the measure of outlyingness, the following process must be applied:

$$\tilde{u}_{i,k} = \frac{u_{i,k} - m_k}{\sum_j \mid u_{j,k} - m_k \mid}$$

where m_k stays for the median over all items of class C_k

$$m_k = \frac{\sum_j u_{j,k}}{N_k}$$

where N_k represents the number of observation of class C_k.

Observation 1.2.4. *[7]*

1. *If the value is negative, then it is set zero.*
2. *If the value is above 10, it is marked as an outlier.*

Chapter 2

Importance of variables

As already pointed out, random forests lose interpretability compared to decision trees, because the last ones show a direct influence of the predictor variable at its position at the tree. While the decision trees show the influence that the predictor variables have, directly on their position at the tree [17], random forests are constructed in a more complex way, and they do not reveal the dependency directly.

In order to interpret random forest, one of the most important tasks in random forests are to be done: to measure the importance of the variables[3]. There are different ways of addressing this issue.

1. One of them, and probably the simplest one, consists in just counting how many times each variable is selected by the trees that conform the forest[17].

2. Another, more complex, measure of the importance of variables is the Mean Decrease Impurity(MDI). In case the selected splitting criterion is the Gini criterion, than the Mean Decrease Impurity is called Gini importance[17].
   This measure of variable importance is based on the following assumption: the Gini Index for a parent node has a higher value than the one of its descendants [13]. As we explained previously, when doing a split, the "best" variable for the split must be selected according to the decrease in impurity measure. So averaging the decrease in impurity over all the nodes of the trees in the forest, where the variable is the one selected for the splitting, the MDI is obtained. So, the higher the value of the MDI, the more important the variable [15].

3. A further way is the permutation of the values of the variable in consideration, called Mean Decrease Accuracy (MDA) [13]. It consists in permuting for $i = 1, ..., N$ the predictor variable X_i^m *($m-th$ coordinate of the measurement vector* $\mathbf{X}_i$*)*, dissociating this variable from its original observation Y_i.

Marital status	Working
Married	Yes
Single	No
Divorced	No
Married	Yes

Table 2.1: Zoom on variable X_i^1 *Marital status* of Table 1.1

Marital status	Working
Single	Yes
Married	No
Married	No
Divorced	Yes

Table 2.2: Permutation of the variable *Marital status* up to Table 2.1

For a better understanding of the permutation, let us go back to the example of Table 1.1. The variable X_i^1 was for the different $i's$ $X_1^1 = Married, X_2^1 = Single, X_3^1 = Divorced, X_4^1 = Married$, and the correspondent observations $Y_1 = Yes, Y_2 = Yes, Y_3 = No, Y_4 = Yes$.

Once the permutation of the variable is done, the appearance of Table 2.1 becomes, for instance, the one in Table 2.2.

If now the permuted variable, together with the non permuted variables, is used to predict the response, there are two possible scenarios. The first consists in anything changing with respect to the original scenario, where no permutation had be done. This would mean that the permutation of the variable has not influenced on the results, so this variable is not really important for the prediction. The second possible scenario is that after the permutation the number of observations misclassified increases, i.e. the accuracy decreases. The only justification for this outcome is that the variable has a significant importance for the prediction [17]. *(It is worth noting that the accuracy is obtained by classifying the OOB sample, so the permutation of the variable takes place just in the OOB)* [19].

If the decrease in accuracy, by permuting the variable, is averaged over all the trees in the forest, the MDA is obtained. So, the higher the MDA, the more important is the variable, because more decreases the accuracy if we permute the variable [15].

Chapter 3

Applications of random forests

Chapters 1 and 2 have been focused on the theory of random forests. Now, in this chapter, some applications of the above explained theory is shown, running different experiments with the software `R`. Therefore, in Section 3.1 the main ideas of computing random forests in `R` are introduced, in order to subsequently use this software for different applications with the data set `Wisconsin breast cancer diagnosis` in Section 3.2, and with two different data sets with functional data, namely, `tecator` and `growth`, in Section 3.3.

3.1. Random forests in R

Before presenting the different experiments, how random forests are implemented in `R` is outlined in this section.

First step is to install the package related with this item and then call such library:

```
install.packages("randomForest")
library(randomForest)
```

Once this has been done, it is possible to use the functions of such package with the data available. Hence, it is important to understand correctly how these functions work. Let us start focusing on the function `randomForest`. The help-environment of `R` provides 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, ...)
```

As most of the arguments of the function `randomForest` are explained by themselves, the most relevant ones are presented in Table 3.1.

Argument	Meaning and working
`formula`	data frame or matrix of predictors, or formula describing the model to be fitted
`data`	data frame containing the variables
`ntree`	Number of trees to grow
`mtry`	Number of variables randomly sampled as candidates at each split. By default for classification it takes $\sqrt{(p)}$, for regression $\frac{p}{3}$, with p the number of variables
`importance`	if it is set TRUE it calculates the importance of the predictors
`proximity`	if it is set TRUE it calculates the proximity matrix

Table 3.1: Meaning and working of the most important arguments of `randomForest`

Once the main points of the computing have been presented, two different experiments are going to be exposed. For the first one, a data set conformed by continuous predictor variables, and a categorical response variable is going to be analyzed, focusing on the variable importance. For the second experiment the use of random forests is going to be extended to functional data.

3.2. Experiment I

The data that is used along this application of random forests, has been obtained from the *UCI Machine Learning Repository* [11], and corresponds to the *Wisconsin breast cancer diagnosis*. This data is conformed by 31 predictor variables, and a categorical response variable. From among the predictors variables we distinguish the ID number (set in column one) and the ten principal ones (set from column 3 to 12), that represent the means:

```
1)ID number

3)  radius (mean of distances from center to points
on the perimeter)
4)  texture (standard deviation of gray-scale values)
5)  perimeter
6)  area
7)  smoothness (local variation in radius lengths)
8)  compactness (perimeter^2 / area - 1.0)
9)  concavity (severity of concave portions of
   the contour)
10) concave points (number of concave portions of the
    contour)
11) symmetry
12) fractal dimension ("coastline approximation" - 1)
```

The remaining variables (13 to 30) are the standard error and the worst case of the 10 already exposed predictor variables.

The response variable is:

```
2)Diagnosis (M = malignant, B = benign)
```

The first step consists in reading, and setting a headline to the data.

```
headline=c("Id","Classification","Radius","Texture",
          "Perimeter","Area","Smoothness","Compactness",
          "Concavity","Concave Points","Symmetry",
          "Fractal dimension","Radius_se","Texture_se",
          "Perimeter_se","Area_se","Smoothness_se",
          "Compactness_se","Concavity_se","Concave
           Points_se", "Symmetry_se","Fractal
           dimension_se","Radius_worst","Texture_worst",
          "Perimeter_worst","Area_worst",
```

```
            "Smoothness_worst","Compactness_worst",
            "Concavity_worst","Concave Points_worst",
            "Symmetry_worst","Fractal dimension_worst")


wisconsin=read.table("wdbc.data",sep = ",",
                     col.names = headline)
```

and in order to be able to call the variables by their names:

```
attach(wisconsin)
```

Now the data is correctly implemented and can be used for the experiment.

In order to allow the replication of this experiment, it is important to set a seed, and "fix" the randomness. Next, the `randomForest` is executed like it can be seen, using five variables ($\sqrt{31}$) at each split and growing 1000 trees, and the results are printed:

```
set.seed(1081)

w.rf=randomForest(Classification~.,data=wisconsin,
                  ntree=1000,mtry=5,importance=TRUE)

print(w.rf)
```

So, it is obtained:

```
Call:
 randomForest(formula = Classification ~ .,
 data = wisconsin,ntree = 1000,mtry = 5,
 importance = TRUE)
               Type of random forest: classification
                     Number of trees: 1000
No. of variables tried at each split: 5

        OOB estimate of  error rate: 3.34%
Confusion matrix:
    B   M class.error
B 350   7  0.01960784
M  12 200  0.05660377
```

that can be interpreted as follows. The OOB error is 3.34 %, i.e. this is the percentage of items of the OOB that have been wrongly classified, or, in other words, the random forest classifies 96.66% correctly. Also the confusion matrix is represented, so it can

be seen how many items have been correctly classified or not, i.e., the false positives (5.66%) and the false negatives (1.96%).

Above can be seen, that in order to obtain information about the importance of the variables, the entry `importance=TRUE` has also been included as an argument of the function `randomForest`. Just using now

```
importance(w.rf)
```

a list with the MDA and the MDI for each variable of the model is obtained:

```
                    MeanDecreaseAccuracy MeanDecreaseGini
Id                              4.918394        1.3460295
Radius                         13.210533       10.2696696
Texture                        17.081849        4.0069453
Perimeter                      14.565036       14.0129864
Area                           14.726296       10.8019264
Smoothness                     10.008769        1.7488165
Compactness                     8.583551        2.3981941
Concavity                      16.030364       10.3178601
Concave.Points                 21.449959       29.4106724
Symmetry                        5.431248        1.1330892
Fractal.dimension               5.452440        0.9137801
Radius_se                      14.810328        4.0634558
Texture_se                      5.709398        1.2645055
Perimeter_se                   13.998440        4.2729203
Area_se                        19.755545        8.5036794
Smoothness_se                   4.534937        1.1640510
Compactness_se                  8.130864        1.3049383
Concavity_se                    8.041339        1.6496968
Concave.Points_se               7.312380        1.1211280
Symmetry_se                     5.618074        1.0870049
Fractal.dimension_se            4.237092        1.3857821
Radius_worst                   24.541105       31.6980086
Texture_worst                  19.268786        5.0469406
Perimeter_worst                23.814279       32.1185216
Area_worst                     25.123753       29.3425631
Smoothness_worst               16.132225        3.2753597
Compactness_worst              11.639484        4.2820763
Concavity_worst                19.114623        8.8452845
Concave.Points_worst           25.047172       34.6431429
```

```
Symmetry_worst                9.118319          2.2282686
Fractal.dimension_worst       8.311042          1.8237781
```

To interpret this more easily, let us show it graphically in Figure 3.1 from:

```
varImpPlot(w.rf,n.var = 31)
```

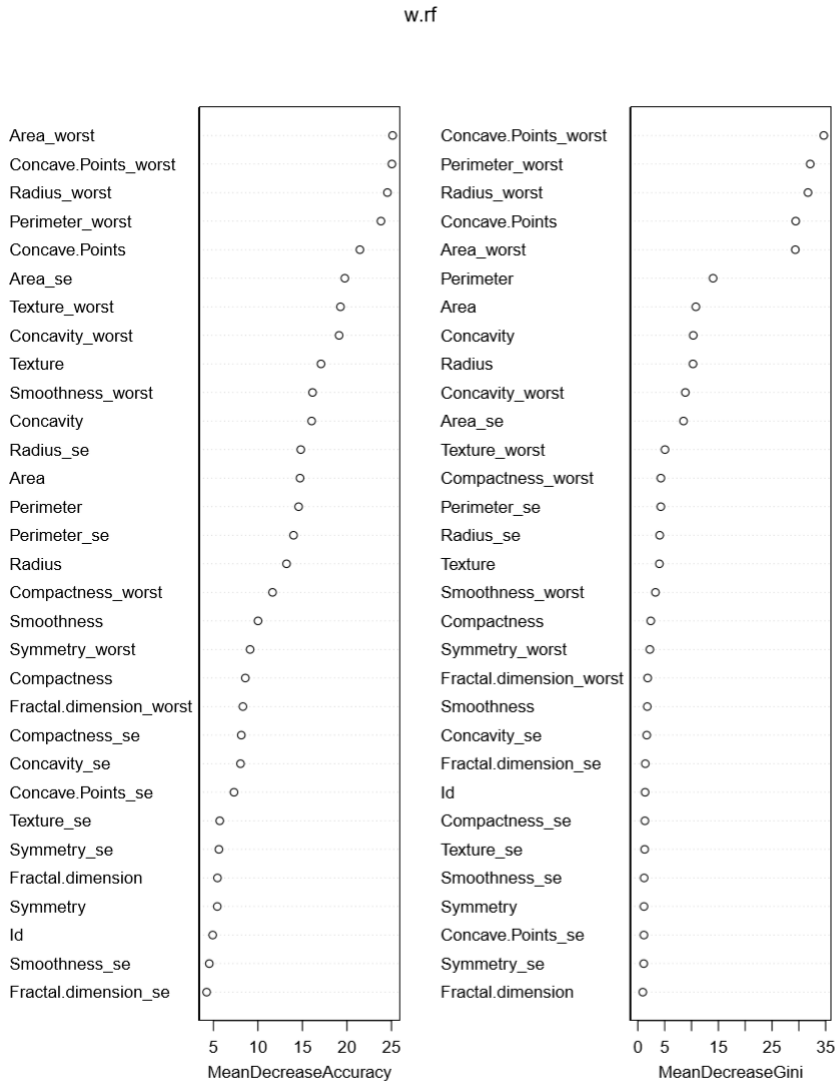


Figure 3.1: Importance of variables

As explained in Chapter 2, the higher the MDA (or the MDI), the more important the variable (according to the respective criteria). Although the results of the two meth-

ods are, in general, not exactly the same, both will identify very important or very superfluous variables. In this case, according to the MDA the most important variables, in descendant order of importance are: `Concave.Points_worst`, `Area_worst`, `Radius_worst` and `Perimeter _worst`. For MDI criterion these four are also the most important variables, permuting the order of `Concave.Points_worst`and `Perimeter_worst`. When considering variables that are not important for the random forest, it stands out that the MDA considers as completely superfluous variables less variables than MDI. For MDA we have, if we consider for example MDA under 10 as unimportant, in increasing order of importance: `Smoothness_se`, `Id`, `Symmetry`, `Fractal dimension`, `Symmetry_se`, `Texture_se` and a series of not that important variables as can be seen in the graphic. Nevertheless, according to MDI, there are a lot of superfluous variables, it suffices to take a look at the graphic and see how many variables have a value MDI=5 (not even under 10 as was done with MDA), outstanding `Fractal.dimension`, `Symmetry_se`, `Concave.Points_se` and `Symmetry` among others. As might be expected, in both cases, `Id` is considered an unimportant variable, because a number of identification assigned to a patient cannot be a reason to detect cancer.

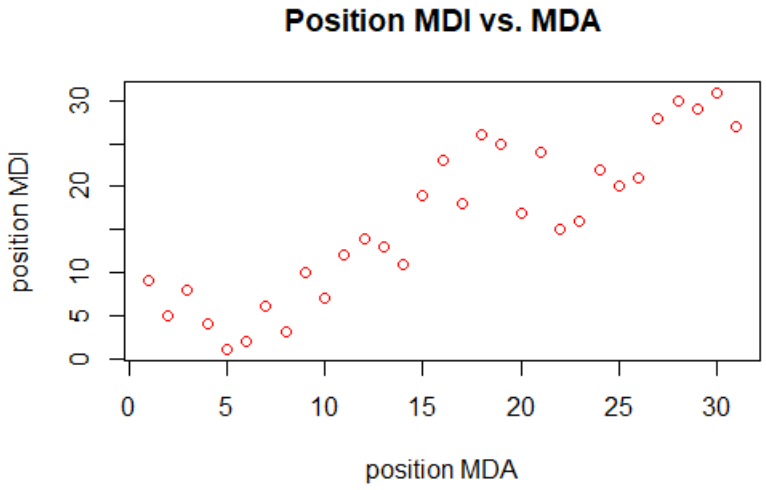


Figure 3.2: Ranking of the variables according to MDA and MDI

From the above information, it can be deduced that there exist some similarities between the ranking of the variables, when listing them according to the MDA or the MDI. So if we take a look at Figure 3.2 the relations between the position in variable importance that the different variables take, in concordance to both criteria, can be observed. Every variable is given the position they take in the ranking of variable

importance according to the MDA *x-label* and according to the MDI *y-label*. It emerges that the dependence between the variables is a point cloud with trend line position MDI=position MDA $(y = x)$. In Figure 3.3, not the positions of the variables, but their MDI and MDA exact values are shown.

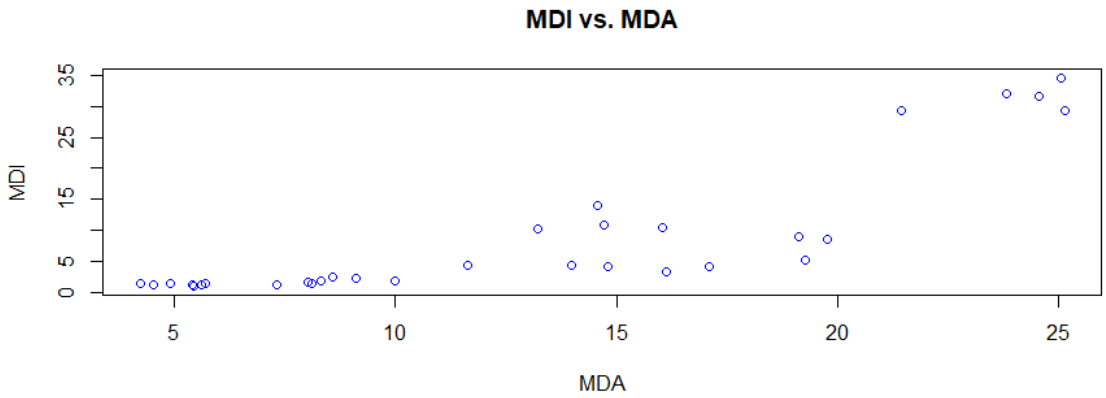


Figure 3.3: MDA and MDI values for the distinct variables

What happens if the variables considered as important are omitted from the model? And if the ones considered superfluous are omitted? When removing the important ones the OOB error should increase, and when deleting the unimportant ones, it should stay quite similar. Nevertheless, when computing the new random forest, owing to the fact that the internal process is subject to randomness, we cannot understand the new random as exactly the same as the previous one omitting some variables, but as a new one (even the OOB error could not verify what expected).

Another point is, the proximity matrix, that can also be obtained, as follows:

```
set.seed(1081)

w.rf=randomForest(Classification~.,data=wisconsin,
                  ntree=1000,mtry=5,proximity=TRUE,
                  proximity=TRUE)
w.rf$proximity

w.rf$proximity[1:4,1:4]
```

since the matrix's dimension is $569 * 569$, only a part of the top rows are given:

```
            1          2         3          4
1 1.00000000 0.59090909 0.7014925 0.02816901
2 0.59090909 1.00000000 0.8043478 0.04964539
3 0.70149254 0.80434783 1.0000000 0.11564626
4 0.02816901 0.04964539 0.1156463 1.00000000
```

It can be seen that the diagonal of the matrix are ones, because obviously a data is as "close" to itself as possible. Comparing different items it can be deduced, for example, that the entries 1 and 4 are very "far away", because the cell (2,4) (or (4,2)) is close to zero, or that the 2 and 3 are near to each other, i.e., they end up in many trees at the same terminal node. Nevertheless, there exist some data that do never end at the same node at any tree, their proximity is 0. Some examples are the entries 11 and 15 or 20 and 11.

```
w.rf$proximity[11,15]
w.rf$proximity[15,20]
```

Notation 3.2.1. *In order to obtain the same results as exposed, it is important, not only to set the same the seed, but also always compute the* `randomForest` *with the argument* `importance=TRUE`*, independent of what is really of interest is the OOB error, the importance of variables or the proximity matrix, because when setting this argument TRUE or FALSE the results are, owing to randomness, not equal.*

Conclusions of Wisconsin cancer

Last but not least, the conclusions inferred from the applications done with the *Wisconsin breast cancer diagnosis* data are outlined:

1. Random forests give, in an easy way, not only a simple result, but also a lot of information, because they give information, like the out of bag error (OOB), as also the number of items correctly or incorrectly classified (confusion matrix), or the proximity between the observations and even the importance of the variables.

2. Even when using a large number of variables, the random forests give results of small OOB errors, not overfitting.

3. No matter if using MDA or MDI for analyzing the importance of variables, most of the least and most important variables, according to both criteria, are the same ones. It has been even possible to analyze the position the different variables take when ranking them according to both criteria, and it stands out that there is a linear dependence (y=x) between them.

4. It was supposed that when removing important variables the OOB error should increase. Nevertheless, because of the internal randomness of the process, the forests cannot be compared.

3.3. Experiment II

Now it is time to see what happens when using functional data. For a functional data classification problem a sample of N observations is given and to each observation a pair (X_i, Y_i) is associated, but different as considered until this moment, X_i is not a numeric or a categorical value, it is a Riemann integrable function $X_i : [0, L] \longrightarrow \mathbb{R}$ [4].

How can a problem of functional data be addressed with random forests techniques? The function of each observation must be discretized, every one using the same sequence of arguments $(l^1, ..., l^M)$. In the end, there are M values X_i^m available for each item, or what is the same, a vector of M components, where each component corresponds to a value of $(l^1, ..., l^M)$. It follows that, after the discretization the scenario is an already known situation: for each observation there is a pair $(\mathbf{X}_i, Y_i)$.

3.3.1. Tecator

Let us start studying the dataset `tecator`, available with the `library(fda.usc)` of `R`. This database represents the results obtained on 215 meat pieces for an analysis with a food and feed analyzer, tecator. In fact, the data consists on 100 discretization points of the function *Absorbances for the different Wavelengths* in the interval $[850nm, 1050nm]$, for each observation. The absorbance at each wavelength stays for a variable X^m. The response Y_i is given by the percentages of fat of the meat piece (continuos values).

This data set can be analyzed as classification problem and also as a regression problem, depending on the treatment with the response variable Y_i. Both cases are going to be addressed in this text.

3.3.1.1. Classification of tecator

Starting, with the classification problem, the first step is to set different classes for the response Y_i. Therefore the following criterion is used:

- If $Y_i < 20$ set the class -1 = Low fat index
- If $Y_i \geq 20$ set the class $+1$ = High fat index

In order to compute this, following is done to get the `Data` that is going to be used with the `randomForest`.

```
data("tecator")

Explicativ=tecator$absorp.fdata$data
Response1=tecator$y$Fat

for (i in 1:length(Response1)){
  if (Response1[i]<20){
    Response1[i]=1
  }else{
    Response1[i]=-1
  }
}

Response=factor(Response1)

Names=c("Vari1","Vari2","Vari3","Vari4",...,"Res")
%Where it says ... the name of the 100 Variables must be
 set in.%

Data=data.frame(Explicativ,Response)
colnames(Data)<-Names
attach(Data)

plot.fdata(tecator$absorp.fdata,col=3+Response1)
```

The different curves, i.e. the values of $\mathbf{X}_i$ in the interval of $[850nm, 1050nm]$, with different color for each class, are plotted in Figure 3.4.

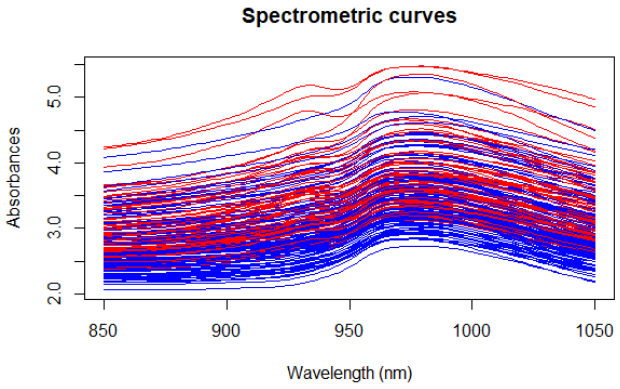


Figure 3.4: Spectometric curves for each observation i

Once the data has correctly been computed, next step is running the random forest and analyzing the variable importance.

```
set.seed(1081)
rf=randomForest(Res~.,data=Data,ntree=1000,
importance=TRUE,proximity=TRUE)
print(rf)
I=importance(rf)
varImpPlot(rf,n.var=100)
```

When analyzing the importance of variables, the number of variables is that big, that the matrix `I` does not give an easy overview on all values.When using `varImpPlot` we get an idea of the important variables, but it is difficult to see if just that variable is important or the variables in a particular range are important. Therefore let us better take a look at Figure 3.5.

It can be observed that for the MDA and MDI approximately the same variables are considered (un)important, grouped in different "valleys" and "mountains", so that the most important variables are identified approximately in the ranges $[Variable1, Variable20], [Variable34, Variable45]$ and $[Variable90, Variable100]$. Remember each variable corresponds to a wavelength of the range $[850nm, 1050nm]$, that had been discretized in a preprocessing stage. This entails that, when saying that the range $[Variable1, Variable20]$ involves important variables, what in fact is important are the wavelengths $850nm, 852nm, .., 888nm$.

By printing the results of the random forest an OOB error of 16,28% is obtained, so there is a considerable error. Is it possible to improve the OOB error any way? Let us see a way that may improve the OOB error. Although in order to work with random forests the data has been discretized, we start from functions (functional data). So the derivatives of the functions *(f)* can be calculated, and then we introduce the discretization of the derivatives as new variables. In our case we are going to use the first *(D1)* and the second derivatives *(D2)*. These correspond to the situation exposed in Figures3.6(a) and 3.6(b), where we can see the derivatives curves of the different observation vs. the wavelengths in two different colors, depending on the class the curves correspond to.

Afterwards different analysis can be done: just with the fist derivative *(D1)*, just with the second derivative *(D2)*, with the first derivative and the original function *(f+D1)*, with the second derivative and the original function *(f+D2)* and, at last, with the original function and both derivatives *(f+D1+D2)*.

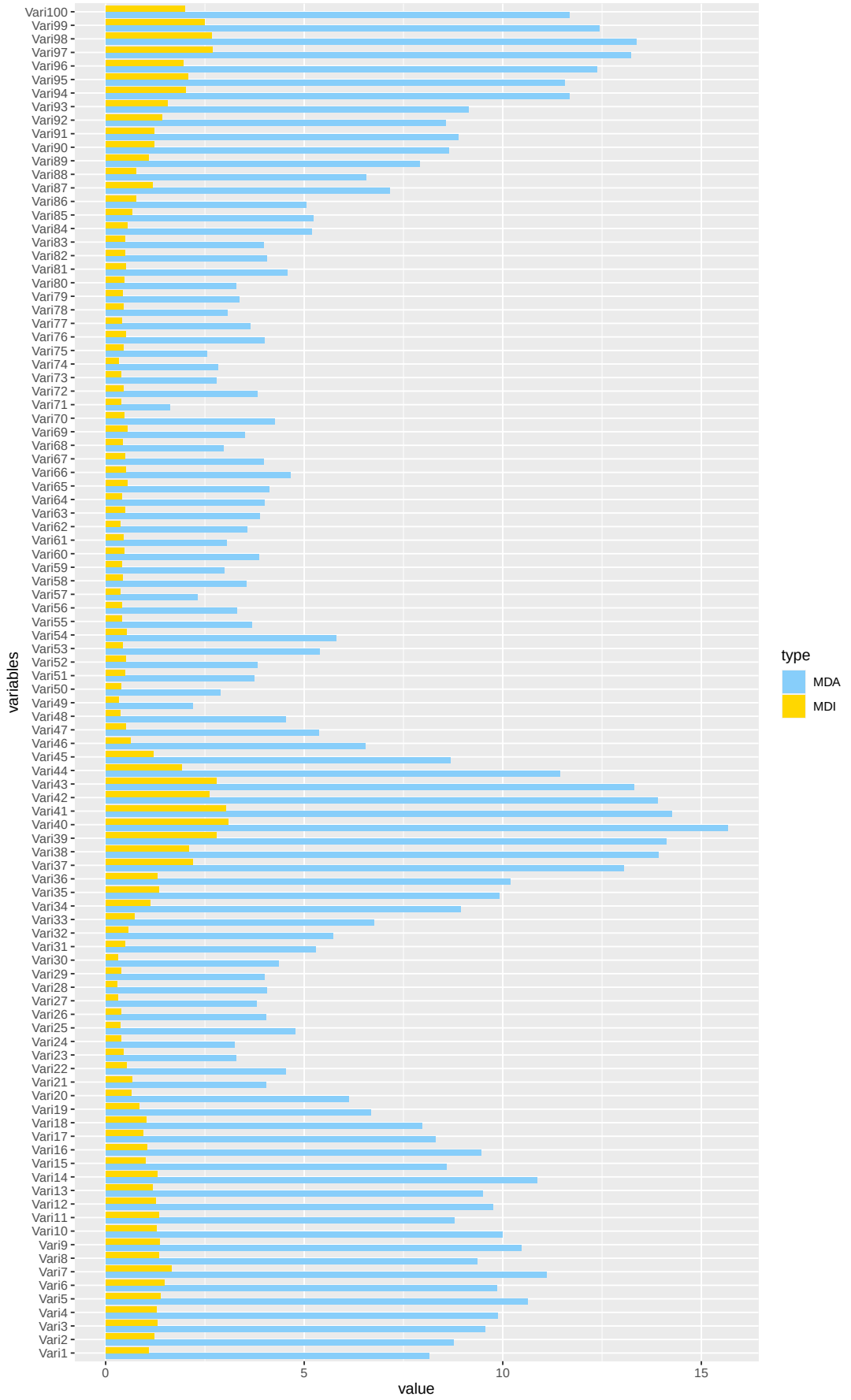
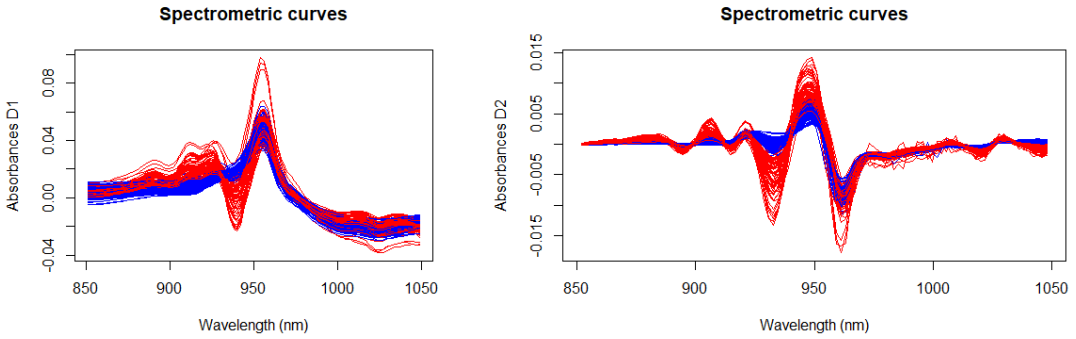


Figure 3.5: Variable Importance



(a) Curves of first derivative as functions of the wavelength

(b) Curves of second derivative as functions of the wavelength

Figure 3.6: First two derivatives of the absorbances with color as class-distinction

The first derivative is built as follows:

```
Deriv=matrix(,nrow=nrow(Explicativ),
             ncol=ncol(Explicativ)-1)
for (i in 1:nrow(Explicativ)){
  for (j in 1:ncol(Explicativ)-1){
    Deriv[i,j]=Explicativ[i,j+1]-Explicativ[i,j]
  }}
```

So now it can be computed the random forest with the first derivative, the same way as it was done with the original function just replacing `Explicativ` for `Deriv` and naming also the new variables.

```
DataDeriv1=data.frame(Deriv,Response)
set.seed(1081)
rfD1=randomForest(Res~.,data=DataDeriv1,ntree=1000,
importance=TRUE,proximity=TRUE)
print(rfD1)
```

In this case a OOB error of 3,26% is obtained, i.e. it has been reduced considerably.

When introducing the second derivative, i.e., using:

```
Deriv2=matrix(,nrow=nrow(Deriv),
ncol=ncol(Deriv)-1)
for (i in 1:nrow(Deriv)){
  for (j in 1:ncol(Deriv)-1){
    Deriv2[i,j]=Deriv[i,j+1]-Deriv[i,j]
  }}
```

```
NamesDeri2=c("Deri1",...,"Res")
DataDeriv2=data.frame(Deriv2,Response)
colnames(DataDeriv2)<-NamesDeri2
attach(DataDeriv2)

set.seed(1081)
rfD2=randomForest(Res~.,data=DataDeriv2,ntree=1000,
importance=TRUE,proximity=TRUE)
print(rfD2)
```

the OOB error is reduced until 0,93%.

If the results of the other combinations of function and derivatives before mentioned are to be calculated, the following must be computed:

```
Namestogether1=c("Vari1",...,"Deri1",...,"Res")
VariablesTogether1=cbind(Explicativ,Deriv)
DatosTogether1=data.frame(VariablesTogether1,Response)
colnames(DatosTogether1)<-Namestogether1
set.seed(1081)

rfT1=randomForest(Res~.,data=DatosTogether1,ntree=1000,
importance=TRUE,proximity=TRUE)
print(rfT1)
IT1=importance(rfT1)
varImpPlot(rfT1)
#################

Namestogether=c("Vari1",...,"2Deri1",...,
"Res")
VariablesTogether2=cbind(Explicativ,Deriv2)
DatosTogether2=data.frame(VariablesTogether2,Response)
colnames(DatosTogether2)<-Namestogether2
attach(DatosTogether2)

set.seed(1081)
rfT2=randomForest(Res~.,data=DatosTogether2,ntree=1000,
importance=TRUE,proximity=TRUE)
print(rfT2)
IT2=importance(rfT2)
varImpPlot(rfT2)
##################
```

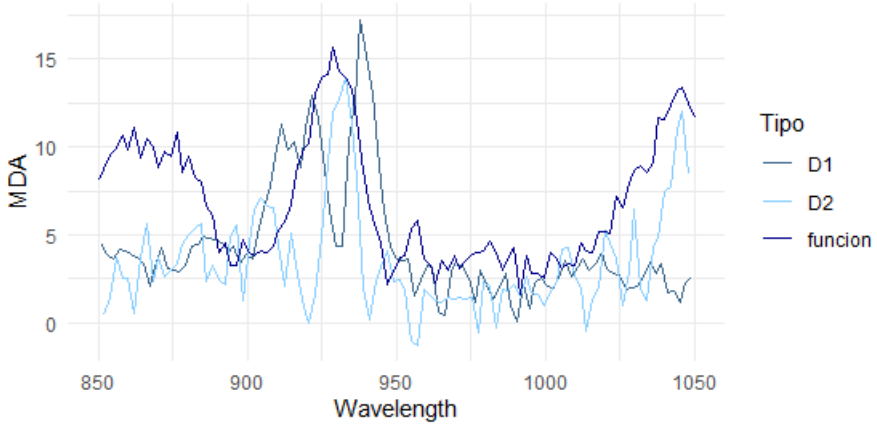
```
Namestogether=c("Vari1",...,"Deri1",...,"2Deri1",...,
                "Res")
VariablesTogether=cbind(Explicativ,Deriv,Deriv2)
DatosTogether=data.frame(VariablesTogether,Response)
colnames(DatosTogether)<-Namestogether
attach(DatosTogether)

set.seed(1081)
rfT=randomForest(Res~.,data=DatosTogether,ntree=1000,
importance=TRUE,proximity=TRUE)
print(rfT)
IT=importance(rfT)
varImpPlot(rfT)
##################

NamestogetherD=c("Deri1",...,"2Deri1",...,"Res")
VariablesTogetherD=cbind(Deriv,Deriv2)
DatosTogetherD=data.frame(VariablesTogetherD,Response)
colnames(DatosTogetherD)<-NamestogetherD
attach(DatosTogetherD)

set.seed(1081)
rfTD=randomForest(Res~.,data=DatosTogetherD,ntree=1000,
importance=TRUE,proximity=TRUE)
print(rfTD)
ITD=importance(rfTD)
varImpPlot(rfTD)
```

Let us now see the OOB errors for the different combinations in Table 3.2.

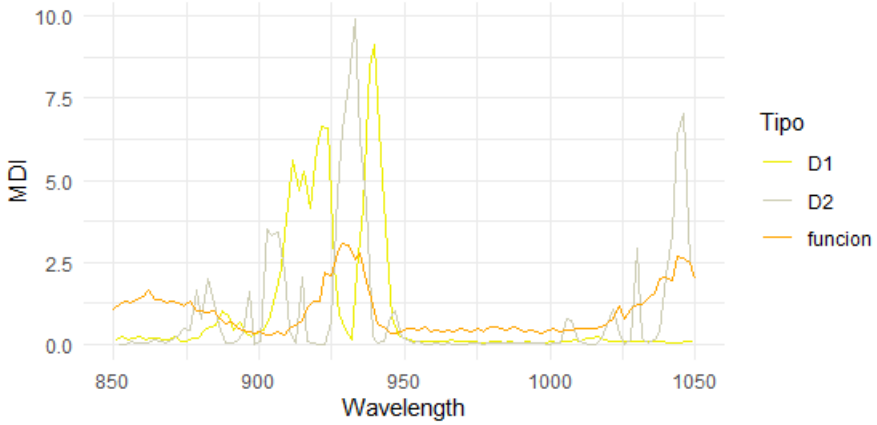
Functions used for the explicative variables	**OOB error**(%)
f	16,28
D1	3,26
D2	0,93
f+D1+D2	0,93
f+D1	2,79
f+D2	0,93

Table 3.2: OOB errors for the different combinations in the classification problem of tecator

The result obtained just with the second derivative *(D2)* is exactly the same result that is obtained when using the second derivative and the function with *(f+D1+D2)* or without *(f+D2)* the first derivative, i.e. the variables of the second derivative are the ones that better the results. When using the original function and the first derivative *(f+D1)* the results are much better than using only the original function *(f)* and a few better than using only the first derivative *(D1)*, but not as good as when using the second derivative *(D2)*, i.e. the first derivative betters the results obtained with the original function but not as much as the second derivative does. So, answering the question posed above, for this data the use of the derivatives is a clear way of improving the OOB.



(a) MDA for original function, first derivative and second derivative



(b) MDI for original function, first derivative and second derivative

Figure 3.7: Variable importance different derivatives

Another question that can be tried to be answered is if the variables that are impor-

tant for the original function, the first derivative and the second derivative correspond to the same wavelength. When representing the MDA and MDI of the random forests obtained up to the original function, the first derivative *(D1)* and the second derivative *(D2)*, approximately the variables corresponding to the same wavelengths are important, as can be seen in Figures3.7(a) and 3.7(b). Nevertheless, it is important to mention that, when speaking about a range, what has been just said is correctly, but when speaking about the values inside that range it follows that not exactly the same wavelengths are important for the distinct functions. Taking a look at the range between $900nm$ and $950nm$ approximately, it stands out that this range contains the most important wavelengths, independent of speaking of the original function, the first or the second derivative. Nevertheless, inside the range it must be emphasized that the original function and the second derivative have as most important wavelengths more or less the same values around $930nm$, but at this point, the first derivative achieves a minimum. It follows that the first derivative is complementary to the original function and the second derivative.

3.3.1.2. Regression of tecator

In the beginning it was said that the data set tecator originally consists of a compound of observations with the variable response fat index, a continuous variable, that we decided to transform into a qualitative variable grouping the fat indexes in two classes: high and low fat index, in order to address this problem as a classification problem. Nevertheless, it is also possible to use random forest techniques when using fat index as a continuous variable, considering this time the problem as a regression problem. Proceding this way, the only difference with respect to the classification problem (when computing it with `R`) is to keep the original response variable like we can see for the case with the original function *(For the rest of the programming just keep on (with Response coded like here) the same way as was done with the classification problem)*:

```
Explicativ=tecator$absorp.fdata$data
Response=tecator$y$Fat
Data=data.frame(Explicativ,Response)
```

Let us see now what happens when using the different combinations of original function and the first two derivatives as was done with the classification problem. Therefore, see Table 3.3 *(The results are obtained with seed 1081)*.

In relation with what can been observed in the Table 3.3 it can be said that when using the original function and the two derivatives *(f+D1+D2)* the best result is obtained. Nevertheless, the results are very similar to the ones of just using the second derivative or he second derivative and the original function *(D2 or f+D2)*. When just

Functions used for the explicative variables	**Variable explained**(%)	**Mean squared residuals**
f	70,41	47,80386
D1	98,68	2,135725
D2	99,05	1,526871
f+D1+D2	99,32	1,093536
f+D1	98,68	2,134657
f+D2	99,01	1,595362

Table 3.3: Different combinations in the regression problem of tecator

using the first derivative *(D1)* or the first derivative and the original function *(f+D1)* the same result is obtained for the percentage of the explained variability of the response variable, probably because the best results when splitting are obtained for the variables of the first derivative (in contrast to the variables of the original function), so using for the random forest just the first derivative *(D1)* or its combination with the original function *(f+D1)* does not present any difference. The worst result is obtained when using just the original function *(f)*.

3.3.2. Growth

The dataset called `growth`, available with the `library(fda)` of `R` has also been addressed as a classification problem. This data set represents the function of the height of 39 boys and 54 girls between the ages of 1 and 18, discretized into 31 points, not equally spaced *(1, 1.25, 1.5, 1.75, 2, 3, 4, 5, 6, 7, 8, 8.5, 9, 9.5, 10, 10.5, 11, 11.5, 12, 12.5, 13, 13.5, 14, 14.5, 15, 15.5, 16, 16.5, 17, 17.5, 18)*. So this way for each individual i there are 31 variables X_i^m, with $m = 1, ..., 31$ that can be used in order to classify with random forests the response variable, the sex of the individual. The different curves of height for the different ages are drawn in Figure 3.8, in different colors depending if it corresponds to a boy or a girl.

As done with the `tecator` data set, the two classes of the response variable of the `growth` data are going to be coded:

- "1" if it is a boy
- "-1" if it is a girl

So following must be computed:

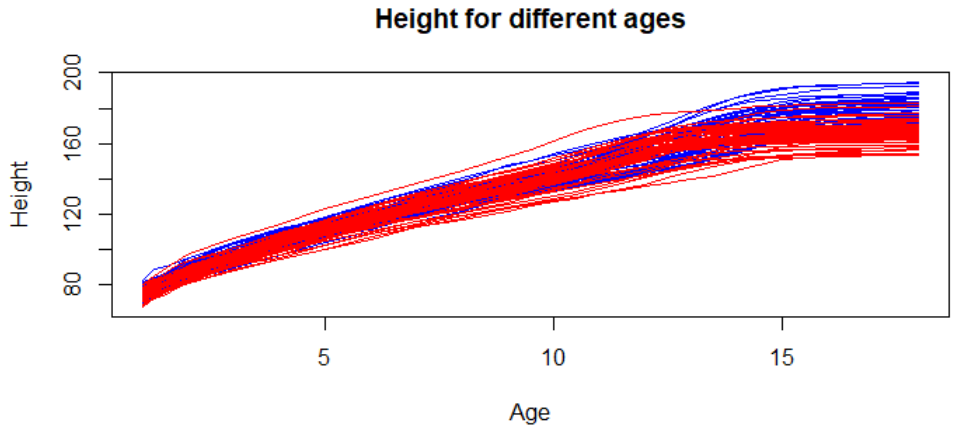


Figure 3.8: Height at the ages between 1 and 18

```
library(fda.usc)
library(randomForest)
#####################################################

data("growth")
names=c("Vari1","Vari2","Vari3","Vari4","Vari5","Vari6",
        "Vari7","Vari8","Vari9","Vari10","Vari11",
        "Vari12","Vari13","Vari14","Vari15","Vari16",
        "Vari17","Vari18","Vari19","Vari20","Vari21",
        "Vari22","Vari23","Vari24","Vari25","Vari26",
        "Vari27","Vari28","Vari29","Vari30","Vari31",
        "Res")

boys=rep(1,39)
girls=rep(-1,54)

Explicativ=rbind(t(growth$hgtm),t(growth$hgtf))
Response1=c(boys,girls)
Response=factor(Response1)

Data=data.frame(Explicativ,Response)
colnames(Data)<-names
attach(Data)
```

So, when computing the random forest with the original function

```
set.seed(33)
rf=randomForest(Res~.,data=Data,ntree=1000,
                importance=TRUE,proximity=TRUE)
print(rf)
```

that will have as output:

```
Type of random forest: classification
                      Number of trees: 1000
No. of variables tried at each split: 5

        OOB estimate of  error rate: 8.6%
Confusion matrix:
   -1  1 class.error
-1 48  6  0.11111111
1   2 37  0.05128205
```

So, the OOB error obtained is of 8,6% that we will try to improve as we did with the `tecator` data set. The confusion matrix is also shown in the output, with 48 "true positive", i.e. in this case, 48 girls classified as girls, and 37 "true negative", i.e. 37 boys classified by the random forest as boys.

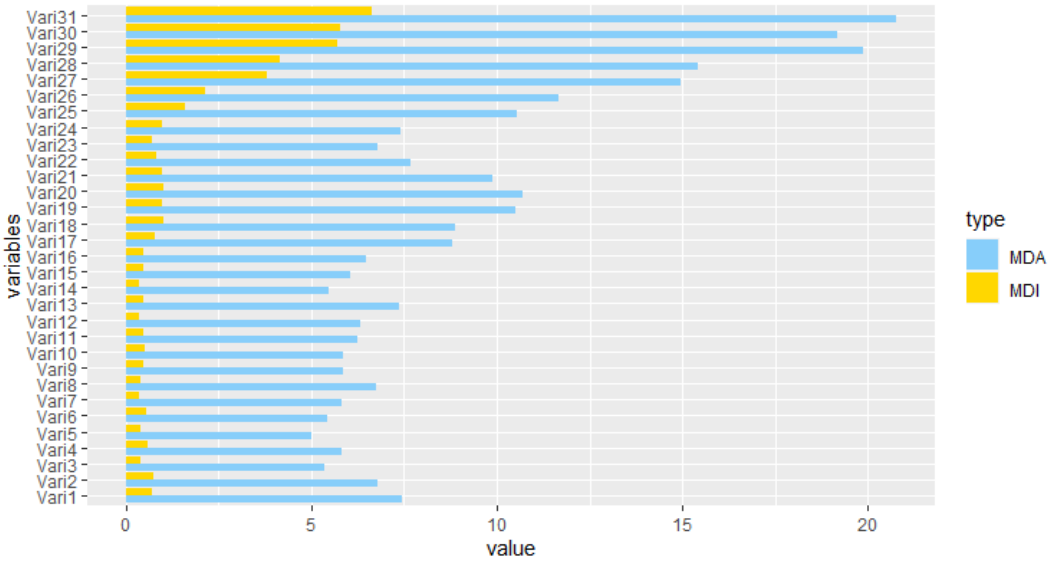
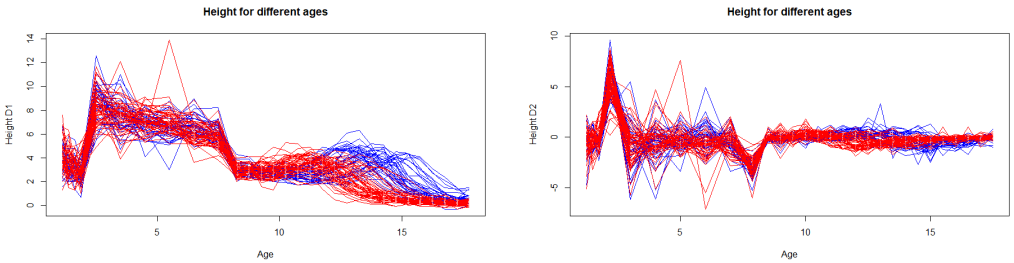


Figure 3.9: Variable Importance

Let us now see the importance of variables. According to Figure 3.8, it might be expected that the most important variables are those of ages round 15, because until

that moment, the curves of boys and girls are very similar, and only up to the mentioned age, the curves are different for most of the individuals of both classes. So, taking a look at Figure 3.9, it can been observed that, the most important variables, according to MDI and MDA are `Vari27` until `Vari31`, which correspond to the ages 16 to 18, i.e. the result obtained is coherent with what was expected.

Next step is to see what happens when introducing the first and the second derivatives. Therefore, let us see first the curves of these functions in Figures 3.10(a) and 3.10(b). From Figure 3.10(a) can it can be deduced that the most important ages in order to distinguish if the person is a boy or a girl, are approximately the ones over 13 years, because up to that moment, the curves of boys and girls are different. When focusing on the second derivative in Figure 3.10(b) it cannot be done such a deduction.



(a) Curves of first derivative of the height as functions of the age

(b) Curves of second derivative of the height as functions of the age

Figure 3.10: First two derivatives of the height with color as class-distinction

Taking a look at Figures3.11(a) and 3.11(b) it can be seen now which are the most important variables for the random forests when using the first and the second derivatives. Figure 3.11(a) shows that the variables `Deri19` until `Deri30` are the most important ones, i.e. the ages 12.25 until 17.75 as was predicted by Figure 3.10(a). Nevertheless, when looking at the important ages according to the second derivative different "valleys" and "mountains" can be identified, standing out the mountain between `Deri17` and `Deri20`, and the one between `Deri22` and `Deri27`, corresponding to the ages between 11.5 and 13, and between 14 and 16.5, respectively. It can also be seen that in the corresponding Figure for the first as also for the second derivative, some MDA values are negative. This means that when permuting the variable values over the different observations the accuracy of the random forests for new data is improved. Nevertheless, in this case we are working with a small database and additionally these values of MDA are very low in contrast to the variables that are considered as important, so they can be considered as zero.

Proceeding similar as with the previous database, the distinct results, obtained with the different combinations of function *(f)*, first derivative *(D1)* and second derivative *(D2)*, for the values of the OOB error are exposed ones in Table 3.4.

Functions used for the explicative variables	**OOB error**(%)
f	8,6
D1	9,68
D2	12,9
f+D1+D2	7,53
f+D1	8,6
f+D2	10,75

Table 3.4: OOB errors for the different combinations in the classification problem of growth

When using just the function *(f)*, just the first derivative *(D1)* or just the second derivative *(D2)*, becomes worse. If we combine the original function only with the first derivative or only with the second derivative, the result stays equal or worse, respectively. The reason is that the database growth is very small, so the random forests overfitts, i.e., it classifies well (too much) the data used to construct the forest but classifies badly new data. Nevertheless, combining all them, the derivatives improve the result of the OOB error of the original function. The best result is obtained for the function with the first and the second derivative *(f+D1+D2)*.

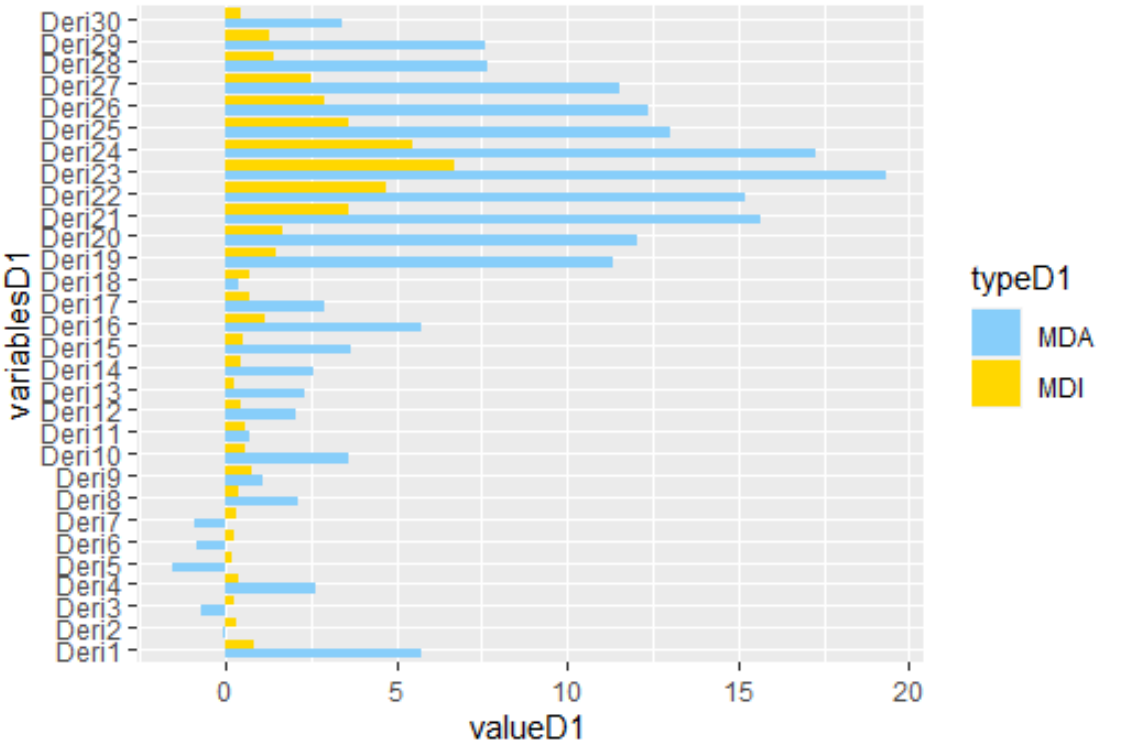
Conclusions of the analysis with the functional data sets tecator and growth

From all the above obtained information of the different classification and regression problems with functional data, using as databases *tecator* and *growth*, the following can be concluded:

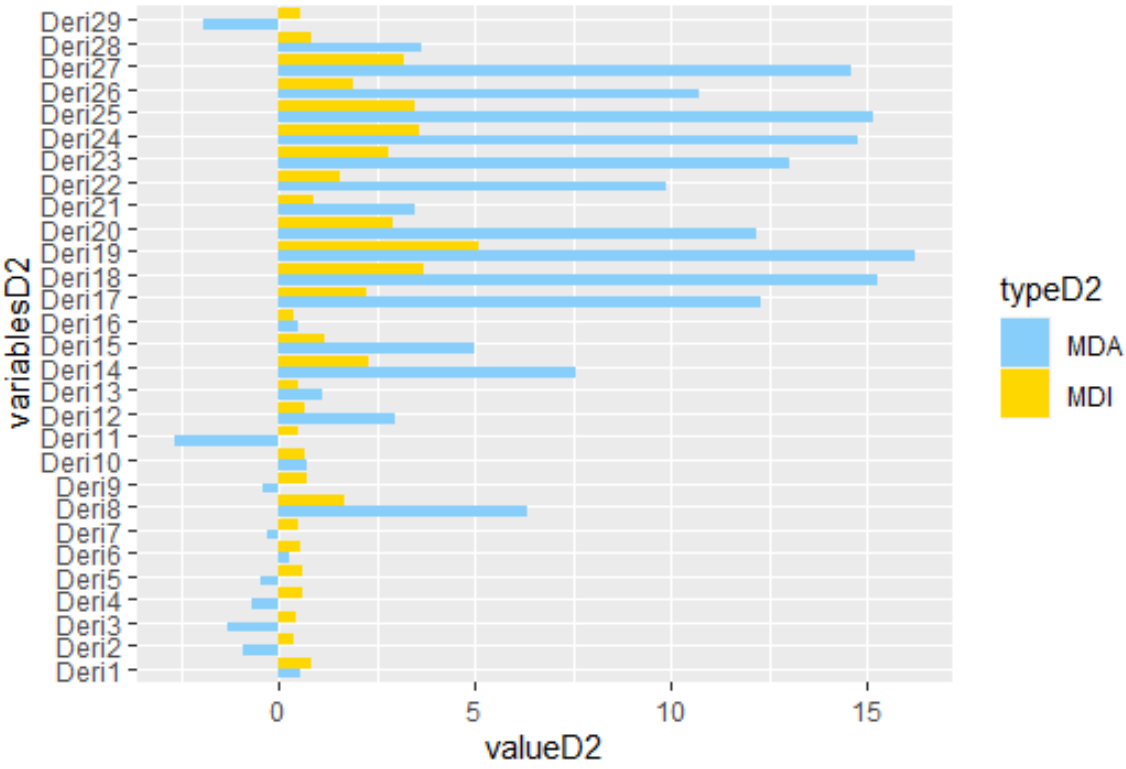
1. When using functional data, random forests can be used succesfully, because the results of the OOB error is considerably low, when speaking of classification problems, and the percetage of variability explained of the response variable is considerably high (in our case always over 90%), when speaking of regression problems

2. When using some derivatives, the results normally improve respect the ones of the original function. Nevertheless, introducing more than one derivative does not guarantee that the result will be better than when using just one. Moreover, it is also not possible to say, a priori, which derivative is better to introduce. Introducing many derivative may lead to overfitting.

Summing up, using random forests is an effective method when classifying or doing regression with functional data, but the best random forest will depend on the particular data, so some times it will be better to use just the first derivative, others a combination of the function and a derivative and so on.



(a) Variable Importance of the first derivative



(b) Variable Importance of the second derivative

Figure 3.11: Variable Importance derivatives

Chapter 4

Conclusions

"Divide and conquer" [3]. This way started the Chapter 1 of this final degree project and indeed, along the different pages it stands out that beyond words, random forests, a mechanism rooted on single decision trees adding techniques based on randomness like bagging and randomness in splitting, are a powerful strategy of solving classification and regression problems that affect today's society.

Random forests not only are able to solve such type of problems, but also do it in a successful way presenting a large number of advantages outlined throughout this text and that have been tested in Chapter 3. Random forests can be used for classification as also for regression problems, what, besides the wide range of problems that become manageable, opens in some cases the possibility of confronting questions in two different ways, giving different views that can improve the comprehension of the matter under consideration. An illustration of this is the experiment done with *tecator* (Subsection 3.3.1): when dealing the problem as a classification problem (Subsubsection 3.3.1.1) it can be said, setting a threshold, if the fat index is low or high, but when addressing this problem as a regression problem (Subsubsection 3.3.1.2) it can be even spoken about the concrete value of fat index, increasing the perspectives of the database *tecator*, not even changing the mechanism of operation, just using the right form of random forests. Both ways have illustrated that random forests work successfully with low OOB errors and a high variable explained percentage (for classification and regression, respectively).

Furthermore, another item that has been pointed out along this text is the importance of variables. Random forests not only uses the given variables to classify or do regression, but also identify the most and less important variables for solving the problem, and grade them in a twofold manner: according to the MDA and to the MDI. This has been shown in a detailed form when using the *Wisconsin breast cancer diagnosis* data (Section 3.2), concluding that, independent of the strategy used to determine the importance of the variable, the ranking established is more or less the same. So much so that, when representing the position of the variables in the grading for both criteria

(one in front of the other) the point cloud fits the line y=x.

Moreover, the efficiency of random forests has also been tested when using a large number of variables. This is shown in the analysis of the data set *Wisconsinn breast cancer diagnosis* (Section 3.2). Nevertheless, once dealt this question, we go a step further and introduce the topic of random forests with functional data. Just discretizing the function in different points, a problem of functional data becomes a problem of a high number of variables that can be dealt with random forests. Examples of that are the ones in Section 3.3, with *tecator* and *growth* databases. An important advantage that present functional data is that they give us a lot of information, because, although for working with random forests it is necessary to do that discretization, it cannot be forgotten, that these points originate from a function that is known. This implies that the derivatives can also be obtained, and introducing them (discretized) as new variables it can be tried to better the result given by the random forests. Not always the results will better. Not always the same combination of variables of the original function and derivatives is the best one. But it has been seen in both experiments above mentioned that many times using the derivatives improves the results. And it has been shown that this improvement can be very significant like in the experiment of *tecator*, where introducing the second derivative the OOB-error betters from $16,28\%$ (f) until $0,93\%$ (f+D2) or in the regression task the variable explained enhances from $70,41\%$ (f) until $99,05\%$ (D2) or $99,01\%$(f+D2).

When using functional data it can also been analyzed the importance of variables, that refer to the distinct points of the argument of the function that has been discretized. Here stands out that the MDA (and MDI) values behave similar in a range, forming "mountains and valleys". Additionally, also for functional data, the positions of the variables for MDA and MDI are quite similar.

Intuitive, comprehensible, a wide range of applications, successful results,... Speaking about random forests, not only the large number of variables they can deal with, but also the large number of advantages they present must be mentioned, a lot of them presented in this final degree project, taking that step further introducing its application with functional data.

Glossary

Symbols	Definition
t	node
t_0	root node
$\mathcal{L}$	learning sample
$\mathbf{X}_i$	measurement vector
Y_i	observation
M	length of vector $\mathbf{X}_i$
N	number of observations
C	set of classes
C_k	class k
K	number of classes
$\pi(k)$	prior class probability
N_k	number of observation of class C_k
$N(t)$	number of observation that have reached node t
$N_k(t)$	number of observations of class C_k that have reached node t
$p(k,t)$	probability of an observation falling into node t and being of class C_k
$p(t)$	probability of an observation reaching node t
$p(k \mid t)$	probability of an observation to be of class C_k subject to having reached node t
B	set of possible categories of X_i^m
Q	set of possible questions responsible for the split
S	set of possible splits
s	split
ϕ	impurity function
$i(t)$	impurity measure of a node t
$\Delta i(s,t)$	decrease in impurity of a split s in

	a node t
T	Tree
T'	Set of terminal nodes
$I(T)$	Tree Impurity
$R^*(d)$	True misclassification rate of the function d
$R(d)$	Resubstitution estimate
$C(k \mid l)$	Cost of misclassifying
$r(t)$	Resubstitution estimate of the expected misclassification cost of the node t
$R(T)$	Resubstitution estimate of the tree T
$C_k^*(t)$	Class assignment rule
F	Number of trees of the forest
$Prox$	Proximity matrix
$Prox(i,j)$	Cell (i,j) of the proximity matrix
$u_{i,k}$	Measure of outlyingness
$\tilde{u}_{i,k}$	Normalized measure of outlyingness
m_k	Mean of measure of outlyingness over all elements of class C_k
J_k	Number of observations of class C_k

Bibliography

[1] Nelson Lee Afanador, Agnieszka Smolinska, Thanh N. Tran, and Lionel Blanchet. Unsupervised random forest: a tutorial with case studies. *Chemometrics*, 30:232–241, 2016.

[2] Philippe Saint-Pierre Baptiste Gregorutti, Bertrand Michel. Correlation and variable importance in random forests. *Stat Comput*, 27:659–678, 2017.

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

[4] R. Blanquero, E. Carrizosa, A. Jiménez-Cordero, and B. Martín-Barragán. Functional-bandwidth kernel for support vector machine with functional data: An alternating optimization algorithm. *European Journal of Operational Research*, 275(1):195 – 207, 2019.

[5] Leo Breiman. Random forests. *Machine Learning*, 45.

[6] Leo Breiman. Statistical modeling: The two cultures. *Statistical Science*, 16(3):199–215, 2001.

[7] Leo Breiman. Manual on setting up, using, and understanding random forests v4.0. *Machine Learning. Journal*, 2003.

[8] Leo Breiman, Jerome H. Friedman, Richard A. Olshen, and Charles J. Stone. *Classification And Regression Trees*. CHAPMAN HALL/CRC, 1984.

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

[10] Ramón Díaz Uriarte and Sara Alvarez de Andrés. Gene selection and classification of microarray data using random forest. *BMC Bioinformatics*, 7:3, 2006.

[11] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[12] F. Hillier and G. Lieberman. *Introduccion a la investigacion de operaciones*. Mexico, D.F: McGraw-Hill/Interamericana, 2010.

[13] Alan Julian Izenman. *Modern Multivariate Statistical Techniques. Regression,Classification, and Manifold Learning*. Springer, 2013.

[14] Robert C. Messenger James N. Morgan. *THAID,a Sequential Analysis Program for the Analysis of Nominal Scale Dependent Variables*. Survey Research Center, Institute for Social Research, University of Michigan, 1973.

[15] M.R. Ortiz-Posadas. *Pattern Recognition Techniques Applied to Biomedical Problems*. Springer.

[16] Martha E. Stone. Cross-validation: A review. *Math. Operationforsch. Statist. Ser Statist*, 9:127–139, 1977.

[17] Carolin Strobl, Anne-Laure Boulesteix, Achim Zeileis, and Torsten Hothorn. Bias in random forest variable importance measures: Illustrations, sources and a solution.". *BMC bioinformatics*, 8(1):25, 2007.

[18] Vladimir Svetnik, Andy Liaw, Christopher Tong, J. Christopher Culberson, Robert P. Sheridan, and Bradley P. Feuston. Random forest: a classification and regression tool for compound classification and qsar modeling. *Journal of Chemical Information and Computer Sciences*, 43(6):1947–1958, 2003. PMID: 14632445.

[19] Jerome Friedman Trevor Hastie, Robert Tibshirani. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2009.

[20] Hal R. Varian. Big data: New tricks for econometrics. *Journal of Economic Perspectives*, 28(2):3–28, 2014.