# Árboles de Regresión. Algunos algoritmos y extensiones a métodos de consenso.



**David Gonzalo Ejea Carbonell**

Trabajo de fin de grado en Matemáticas
Universidad de Zaragoza

Director del trabajo: José Tomás Alcalá Nalvaiz
27 de noviembre de 2017

# Summary

The main object of interest in this work will be the so called Regression Trees. This tool and its related techniques, emerged as an evolution of other related approaches like the AID by Morgan and Sonquist [14] (1963) for regression problems, its adaptation, THAID, to classification by Messenger and Madell [13] (1972) or the recursive partitioning for regression proposed by Kass [10] in 1980.

Nevertheless, the appearance in 1984 of the book Clasification and Regression Trees (CART) by Breiman et al. [3] supposed a turning point in this kind of techniques developing a well-established theory. As of today, this book and its corresponding approach based on a recursive partitioning and its posterior prunning, remains as the cornerstone of the recent tree theory and it will be the basis of what we will develop in this work.

Unlike the usual linear regression approach, tree-based methods turn out to be non-parametric methods, i.e., they try to fit the available data without a predetermined functional form depending on a set of parameters. In particular, fitting a regression tree can be seen as an additive model of the form

$$f(X) = c_1 \cdot 1_{\{X \in R_1\}} + \cdots + c_M \cdot 1_{\{X \in R_M\}},$$

where $R_1, \ldots, R_M$ are boxes in the space of the independent variables. Also, the number of them, $M$, must be estimated too. In this way, regression trees can be viewed like piecewise-constant-regression, but they count with some desirable properties over the least squares linear regression like an easier interpretability and representation, a better robustness and dummy variables treatment and the lack of previous assumptions concerning the data.

While formally, a mathematical tree is any acyclic connected graph, regression and classification trees, as understood in the CART context, are directed graphs and represent in each of their nodes a conditional statement over the space of variables taking the form $X \geq c$, being $X$ a variable and $c$ a real constant. Each of these nodes, also called splits by their partition of the variable space, conform a sequence of rules of recursive partitioning leading to a terminal node, or leaf by analogy, in which a prediction for data in this region is made. This prediction will be the mean over the training data in each terminal node region in regression, and the majority class in a classification problem.

Also, we present different typical options for the goodness of fit in classification trees, in contraposition with the predominant one in regression, the Sum of Squared Errors.

One advantage of the trees is that they offer an elegant solution for the treatment of missing data by making use of the *surrogate splits*. These are alternatives splitting rules whenever the variable of interest is missing on a data point.

On the other hand, a common problem in data modelling is the overfitting, which is the name given to the phenomena of fitting a model which woks very well over the training data but leads to a poor performance over a test data. In this work, we review different approaches to mitigate this problem.

The original solution comes with the pruning. This method consists in removing some branches (paths from a node to every connected leaf) through a minimisation of a goodness of fit measure plus a penalty term depending on the number of terminal nodes, or equivalently, the complexity of the model.

In recent years, aggregation methods have arised as another strategy to reduce overfitting and lead to better predictions. They are based in combining multiple models and we can distinguish two types:

- Averaging methods: Different trees models are fitted independently by resampling. Finally, individual predictions are gathered into one single outcome. In this work, we are introducing Bagging

and Random Forest as a part of this kind of techniques.

The idea behind bagging relies in adjusting trees and then average the outcome to reduce overfitting. Nevertheless, in the presence of variables with a relatively high predictive power compared to others, trees are often very correlated leading to a bad performance.

Random forest arises as a modification of the bagging technique in order to correct this problem by forcing the trees to use a large set of the available variables. Typically, each node is allowed to use a single variable among a reduced set of the order of the third part of the total variables chosen at random. Finally, the outcomes are averaged like in the bagging procedure.

We will also consider a methodology to evaluate the relative predictive power of variables in these methods via permutations of variables.

- Boosting methods: Unlike the previous methods, trees are grown sequentially using information from the preceding ones. These methods do not use bootstrap but a modified version of the data is used. Boosting, which will be described and used in this work, or AdaBoost are examples of this methodology.

  The boosting approach tries to build a model which learns slowly from data emphasizing on the worst predicted points to correct the bad performance of the method. At each stage of the procedure, residuals are fitted with trees and the boosting regressor is updated determined by different learning parameters to module the learning speed.

In the last chapter, an application of the tree theory over a soccer/football dataset is provided. In particular we have 38 variables out of 547 football players of the Spanish national competition during the 2016/2017 season.

First of all, we find that the performance of trees are poor compared to linear regression. While averaging methods stay very close to linear regression, boosting beats in goodness of fit every other considered method.

Checking the relative importance of variables, we find that the rating of player is mostly determined by the minutes that he plays, the number of his appearances in the starting team, his current team, the average number of good passes and shot, as well as the number of goals that he scores.

Finally, the considered methods are applied over the same dataset but just considering the most important variables. While results are qualitatively the same, we find that the averaging methods achieve the same or even a better performance than an equivalent linear regression model, yielding a random forest model almost with the same predictive power than a boosting model.

# Índice general

# Capítulo 1

# Introducción

## 1.1. Estructura de la memoria.

El concepto central de esta memoria van a ser los árboles de regresión, en concreto los propuestos por Leo Breiman y descritos en libro de CART [3] (Classification and regression trees) como se explicará posteriormente.

El capítulo 1 mostrará en primer lugar el contexto histórico necesario para comprender el nacimiento de estas conjunto de técnicas, así como los hitos más importantes en la evolución de las mismas así como del marco en el que vamos a trabajar. La segunda parte de este capítulo estará dedicado al problema general de la regresión, así como la comparación cualitativa entre los métodos tratados por esta memoria y los métodos lineales, señalando sus diferencias de formulación, filosofía y ventajas.

En el capítulo 2 se introducirán los objetos matemáticos pertinentes para alcanzar la definición formal de árbol, así como la metodología empleada para su construcción. Se expondrán los puntos fuertes y débiles generales del uso de árboles, diferentes metodologías que existen, así como técnicas de ajuste con el fin de conseguir mejores resultados (poda de árboles), el tratamiento de datos faltantes, y su extensión a problemas de clasificación.

El capítulo 3 comprenderá una revisión de diferentes métodos de agregación de modelos orientados al caso de árboles, i.e., métodos que construyen un clasificador o modelo de regresión, según sea el caso, combinando diferentes árboles construidos a partir de los mismos datos de entrenamiento mediante técnicas bootstrap, en el caso de los *averaging methods*, o modelos basados en la construcción secuencial con un aprendizaje progresivo como en los *boosting methods*. Una vez expuestas las nociones básicas en torno al bootstrap, exploraremos distintas opciones de la aplicación de los métodos de agregación, entre los cuales encontramos el Bagging, Random Forest o Boosting.

Por último, en el capítulo 4, se ilustrara el uso de las técnicas introducidas sobre un conjunto de datos reales. El conjunto de datos estará formado por 38 variables y 547 jugadores de la liga española de 2016/17, el objetivo principal de esta parte sera describir qué variables son las más importantes a la hora de formar el rating y entender cómo se podría calcular con un número reducido de variables y cuánto de buena sería esta aproximación.

## 1.2. Contexto histórico

Estamos rodeados de una gran cantidad de datos de todo tipo y es por ello que necesitamos diferentes técnicas para comprenderlos, manipularlos, y en última instancia, obtener información y encontrar pautas que los relacionen. Con esta meta Morgan y Sonquist [14] (1963) propusieron el programa AID (Automatic Interaction Detection), el cual representa el primer método de ajuste de los datos basados en modelos de árboles de regresión. Empezando de lo que se llamará un nodo raíz, el algoritmo AID separa los datos recursivamente en dos nodos descendientes del anterior. En los problemas de regresión, donde la variable de interés toma valores numéricos con una relación de orden, los cortes tomaban la forma "$X \leq c$" de manera que la suma de los errores cuadráticos dentro de cada nodo (a modo de medida de

impureza) se minimizara, procediendo así hasta que la mejora dividiendo nodos no fuera sustancial en base a un criterio inicial.

Dicha idea es posteriormente adaptada con éxito a los problemas de clasificación, ver por ejemplo los trabajos de Light y Margolin [11] (1971) y el de Messenger y Mandell [13] (1972) donde introducen el método THAID (THeta Automatic Interaction Detection), donde la variable dependiente no posee una relación de orden, adaptando la medida de impureza con otros criterios basados en la entropía o el índice Gini en los primeros años. En esta línea, es en 1980 cuando Kass [10] propone un algoritmo recursivo basado en la regresión paso a paso para la selección de nodos llamado CHAID (Chi Square Automatic Interaction Detection).

Sobre estas bases, se perfeccionan lentamente los algoritmos de partición de nodos, hasta la aparición de la publicación del trabajo de CART (Classification and Regression Trees) por Leo Breiman [3] en 1984. El enfoque que propone es similar al antes mencionado del AID, dejando en este caso crecer el árbol sin regla de parada, para posteriormente podarlo minimizando el error estimado mediante validación cruzada. Además, este procedimiento subsana carencias de los métodos anteriores como el sub-entrenamiento y el sobre-entrenamiento a expensas de un mayor coste computacional y puede lidiar con datos faltantes mediante el uso de los llamados nodos "subrogados". Además, permite extensiones en las que se pueden establecer cortes lineales, es decir, cortes que empleen una combinación lineal de variables en contraposición a lo que sucedía en el AID.

Después de la salida del libro de Breiman se han propuesto diversas variaciones del método original, pero la idea principal sigue subyaciendo en el particionamiento recursivo y en la posterior poda. Entre estos trabajos podemos destacar RETIS de Karalic (1992, [9]) o M5 de Quinlan(1992, [15]) que abogan por enfoques diferentes de como integrar modelos lineales en las hojas de los árboles de regresión.

De este modo, los métodos basados en la metodología CART han suscitado un gran interés por parte de la comunidad científica debido a su fácil uso y razonable precisión de predicción, poco coste computacional, una muy arraigada disponibilidad en los paquetes de software y una gran versatilidad que permite su aplicación en una amplía tipología de problemas, así como una clara interpretabilidad de los resultados.

## 1.3. Motivación de la Regresión lineal y Árboles

Los problemas de regresión tienen como variable dependiente, es decir, como la variable cuyo comportamiento queremos modelar a través de otras medidas numéricas, una variable numérica dotada de una relación de orden. En contraposición a este concepto encontramos las variables cualitativas, objeto de estudio de los problemas de clasificación, donde las variables toman valores en un número determinado de clases sin ordenar. Por ejemplo, variables binarias, o etiquetas de diferentes conceptos, que si bien pueden ser representadas por números, en realidad no son más que representantes que carecen de reglas aritméticas. Esta dicotomía se trasladará también al ajuste de árboles, cuando hay variables dependientes cualitativas se utilizaran árboles de clasificación y si son cuantitativas se usaran árboles de regresión, siendo la construcción de estos muy similares pero con ciertas diferencias.

Si representamos dicha variable cuantitativa de interés $Y$, y una serie de $p$ variables predictoras (bien cuantitativas o cualitativas) $X_1, X_2, \ldots, X_p$ mediante el vector $X$, podemos escribir el problema de regresión de la siguiente manera general

$$Y = f(X) + \varepsilon, \tag{1.1}$$

donde $f$ es una cierta función desconocida de $X_1, X_2, \ldots, X_p$ y $\varepsilon$ es un término de error estocástico independiente de $X$ de media cero. Es decir, $f$ representa la relación sistemática que $X$ da sobre $Y$.

Tanto para realizar predicciones como inferencia, es decir, extraer conocimiento del problema real a partir de los datos, nuestro primer objetivo será hallar una función $\hat{f}$, que en base a ciertos criterios que impongamos, sea una buena estimación de $f$ en la ecuación (1.1). Una vez hallada esa función $\hat{f}$ (bien mediante una forma paramétrica o no), podremos predecir $Y$ mediante la cantidad

$$\hat{Y} = \hat{f}(X), \tag{1.2}$$

donde $X$, al igual que en caso anterior, será el vector de variables que tengamos sobre el problema. En el campo de la inferencia, cabe preguntarse sobre cuales de esas variables $X_j$ son realmente útiles a la hora de predecir o estudiar el comportamiento de la variable objetivo $Y$, y si lo son, cual es su tipo de relación. En particular, dentro de los métodos paramétricos esa relación podría ser lineal, entrando de lleno en una de las ramas de la estadística más estudiadas a lo largo de la historia. Más concretamente, en una regresión lineal se busca una función $f$ de la forma

$$f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 ..... + \beta_p X_p, \tag{1.3}$$

donde $\beta_0$ es el termino constante, $\beta_i$ para $i > 0$, son los parámetros de cada variable independiente, y $p$ es el número de parámetros independientes. Así pues, la estimación $f$ pasa a ser el problema de la estimación de $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^T$, y si bien es cierto que existen muchas maneras de estimar el vector de coeficientes $\beta$, por popularidad, conocimiento sobre el mismo, propiedades deseables y facilidad de manejo, destaca sobre todas ellas el procedimiento de *ajuste mediante mínimos cuadrados ordinarios*. En particular, se demuestra que este método bajo ciertas hipótesis básicas, coincide con el de máxima verosimilitud cuando $\varepsilon$ es de tipo gaussiano. En este caso, geometricamente la estimación pasa a ser la proyección ortogonal sobre el espacio de funciones posibles según las hipótesis para $f$, y además, satisface el Teorema de Gauss-Markov por el cual estamos antes el mejor estimador lineal insesgado desde el punto de vista de minimización de la varianza (ver por ejemplo el libro de Faraway, [6], 2002).

Sin embargo, aún siendo cierto que el suponer una forma paramétrica para $f$, y más una forma lineal, simplifica mucho el problema de su estimación, esto puede suponer un gran problema si el mecanismo *real* difiere mucho de las formas asumidas para $f$ dando lugar a estimaciones y resultados de los mismos muy pobres. Este puede ser el caso cuando existen efectos multiplicativos o interacciones entre las variables del modelo final. Los modelos de árboles solucionan este inconveniente de modo natural aumentando el número de particiones del espacio cuando sea necesario. Además, los métodos no-paramétricos, que tratan de aproximarse a los datos disponibles de una manera *controlada* sin esa forma funcional, tienen el potencial de adaptarse a un número de escenarios mayor evitando el problema citado anteriormente, aunque con la desventaja de que normalmente requieren un número mayor de datos disponibles para entrenar estos métodos al tener que buscar en un espacio mayor. Entre los métodos no-parámetricos podemos contar por ejemplo los *splines*, *KNN* o *K-vecinos próximos* y los árboles que trataremos en esta memoria.

A la vista de esta introducción, parece claro que los árboles serán una alternativa muy interesante cuando la linealidad de los datos sea una hipótesis alejada de la realidad, pues este procedimiento nos ayudará a capturar esa no-linealidad mediante la división en distintos espacios que proponen los árboles de regresión. Por último, cabe añadir, que el modelo de los árboles de regresión puede verse en cierto modo como una regresión constante a trozos. Mientras que la regresión lineal suponía para $f$ la forma dada en la ecuación (1.3), para los modelos de regresión podemos escribir el siguiente modelo aditivo

$$f(X) = c_1 \cdot 1_{\{X \in R_1\}} + \cdots + c_M \cdot 1_{\{X \in R_M\}}, \tag{1.4}$$

donde $R_1, \ldots, R_M$ representan las diferentes particiones del espacio de las variables independientes. Sin embargo, al ser los $R_1, \ldots, R_M$ regiones del espacio por determinar, para un número $M$ desconocido, también a elegir en base a los datos y los criterios utilizados, el problema no puede reducirse al de regresión lineal y la ecuación (1.3).

Por último, entre las ventajas de los árboles de regresión sobre la regresión lineal podemos encontrar:

- Los árboles pueden representarse gráficamente incluso en altas dimensiones.
- Son mucho más fáciles de interpretar, más que una regresión lineal de una sola variable, incluso para los no expertos. De hecho, el sistema de interpretación de los árboles replica en cierto modo a la toma de decisiones humana.
- Pueden tratar con regresores cualitativos sin crear variables *dummy* o auxiliares.

- No necesitan asumir hipótesis sobre las relaciones entre las variables de estudio.
- Ofrecen m ayor robustez frente a datos atípicos.

Entre los inconvenientes, y además del caso ya mencionado en el que existe una linealidad clara, podemos señalar que existen ciertos métodos que dan normalmente lugar a estimaciones más precisas. Sin embargo, veremos también en esta memoria como corregir esta desventaja con los métodos de agregación o ensamblado. Por otro lado, estos métodos son algo más inestables ante perturbaciones de los datos, perdemos en cierto modo el sentido geométrico de la regresión lineal, y se suele requerir un mayor número de datos para igualar su fiabilidad.

# Capítulo 2

# Árboles de regresión.

## 2.1. Árboles, conceptos generales.

A continuación, damos la definición de árbol y de sus principales características asociadas. Nótese que lo definimos como un grafo dirigido, pues aunque en teoría de grafos no tiene porque serlo, los árboles con los que se trabaja en estadística sí que son necesariamente dirigidos al representar un toma de decisiones sucesiva.

*Definición* **2.1.** Un árbol es un grafo conexo acíclico dirigido formado por un conjunto finito de nodos conectados.

Su representación gráfica toma la forma que podemos observar en la figura (2.1) con los nodos dirigidos hacía abajo. A continuación se enumeran las definiciones asociadas a los árboles.

- Un nodo es la unidad sobre la que se construye un árbol.
- Un nodo puede tener varios nodos a los que apunte llamados **nodos hijo**.
- Se dice que un nodo es un **nodo padre** si posee al menos un nodo hijo.
- Al nodo sin antecesores se le llama **nodo raíz** y es único.
- **Hermanos** son aquellos nodos que comparten el mismo padre.
- Por ultimo están los **nodos terminales** u **hoja** que son aquellos que no tienen ramificaciones o nodos hijo.
- **Rama** es el camino(unión de arcos simples) desde la raíz a un nodo hoja.
- El **grado** de un nodo es el número de descendientes directos que posee, y su **nivel** el número de arcos que han de recorrerse desde la raíz, a la cual se le presupone nivel 1. El **grado de un árbol** es el máximo de los grados de sus nodos, y su **altura** el máximo de los niveles.

Con todo ello, cada nodo no terminal representará en nuestro contexto una proposición lógica acerca de una variable predictora, y cada camino a una hoja una región diferente del espacio que es la intersección de la secuencia de proposiciones lógicas.

Los árboles de regresión toman por tanto su nombre del aspecto del grafo que representa la toma de decisiones entre proposiciones lógicas hasta la llegada a un nodo terminal. Estos, se pueden entender como una regla de análisis no paramétrico que se basan en dividir el espacio predictor en una serie de regiones simples que incluyen observaciones con valores similares para la variable respuesta.

Para hacer una predicción dentro de cada hoja utilizaremos la media de los datos de entrenamiento que se encuentren en la misma región del espacio multidimensional de predictores en caso de los árboles de regresión, y asignaremos las observaciones a la clase más votada en los árboles de clasificación.

La elección de la media en los problemas de regresión se sustenta en el resultado del teorema siguiente.

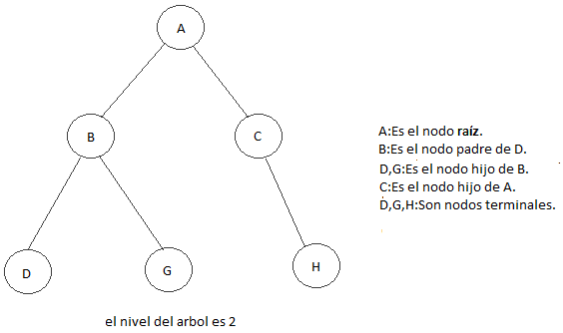


Figura 2.1: Esquema de un árbol.

**Teorema 1.** La constante $k$ que minimiza el valor esperado del error cuadrático es el valor medio.

*Demostración.* Sea $\phi(\cdot)$ la función de perdida cuadrática y $f$ la función de densidad de la variable aleatoria $Y$. Así

$$\phi(k) = E\left[(Y-k)^2\right], \tag{2.1}$$

donde

$$\begin{aligned}\phi(k) = E\left[(Y-k)^2\right] &= \int_{-\infty}^{\infty}(y-k)^2 f(y)dy \\ &= \int_{-\infty}^{\infty}(y^2-2yk+k^2)f(y)dy \qquad (2.2)\\ &= \int_{-\infty}^{\infty}y^2 f(y)dy - 2k\int_{-\infty}^{\infty}yf(y)dy + k^2. \qquad (2.3)\end{aligned}$$

Como queremos minimizar derivamos respecto a $k$ buscamos igualar la derivada a 0:

$$\begin{aligned}\frac{\partial}{\partial(k)}\phi(k) = 0 \Leftrightarrow 0 - 2\int_{-\infty}^{\infty}(y)f(y)dy + 2k = 0 \\ \Leftrightarrow k = \int_{-\infty}^{\infty}yf(y)dy = E[Y] \qquad (2.4)\end{aligned}$$

por definición de $E[Y]$, supuesto finita. □

## 2.2. Construcción de Árboles de regresión.

Para la construcción del árbol utilizaremos la notación:

- $n$: número de casos.
- $n_l$: numero de casos en la hoja $l$.
- $\widetilde{T}$: conjunto de hojas del árbol $T$.
- $T_t$: Subárbol a partir del nodo $t$.
- $T - T_t$: Árbol $T$ podado en el nodo $t$. Por ejemplo, en la figura 2.1, $T - T_c$ sería el árbol compuesto por los nodos A, B, D y G.
- $Y$: Vector de la variable dependiente $Y = (y_1, \ldots, y_n)^T$.

- $X$: Matriz de las variables predictoras $X = (X_1 \mid \cdots \mid X_p)$ con $X_j = (x_1, \ldots, x_n)^t$.
- $m_l := \frac{1}{n_l} \sum_{i \in l} y_i$, media en la hoja $l$.
- $S(T)$: *Sum of Squared Errors* del árbol $T$ o suma total de errores cuadráticos, definida como

$$S = \sum_{l \in \tilde{T}} \sum_{i \in l} (y_i - m_l)^2. \tag{2.5}$$

  y usamos la notación del simplificada $S(t) = S(T - T_t)$

- Corte binario o corte: Es una proposición del tipo $X_j \leq c$ para una constante $c \in \mathbb{R}$. Al trabajar con un número de datos finito, el valor de $c$ no da lugar a posibles problemas de identificación tomando siempre el valor menor (que coincidirá con el valor en esa variable de un dato) a la hora de determinar $c$ en un corte.

La construcción de un árbol de regresión se realiza de manera algorítmica como sigue.

---

ALGORITMO DE CONSTRUCCIÓN DE UN ÁRBOL

---

**Datos:** $Y$, $X$, Criterio de parada, criterio de nodo terminal.
**Inicialización:** El nodo raíz representa todo el espacio de predictores $X$. Se le asigna la media de los valores de $Y$, $m_T$. Calcular $S(T)$.
**Paso 1:** Calcular $S(T)$ para todos los posibles cortes $c$ de los nodos no terminales, si todos los nodos son terminales **FINAL DEL ALGORITMO**.
**Paso 2:** Seleccionar el corte que arroje menor $S$.
**Paso 3:** Si se cumple el criterio de parada , nodo terminal y aplicar **Paso 1**. En caso contrario añadir dos nodos no terminales y aplicar el **Paso 1**.
**FINAL DEL ALGORITMO**

---

Como vemos, el algoritmo toma una forma de búsqueda ávida o vóraz, pues toma decisiones localmente óptimas a cada paso, en este caso el mejor corte posible. Y es descendente, ya que comienza en el nodo raíz y divide sucesivamente el espacio de predictores. Calcular en un primer momento de manera exhaustiva todos los posibles particionamientos del espacio sería inalcanzable computacionalmente para la mayor parte de las aplicaciones.

El criterio de parada suele establecerse como un umbral $\delta$ que es la mejora mínima que se exige cada vez que se realiza un corte, aunque existen otras opciones. En cuanto al criterio de nodos terminales, es común encontrarse con condiciones como un mínimo número de datos en cada nodo para evitar nodos con pocos participantes. De hecho, el $S(T)$ de un árbol con una hoja por cada dato sería cero en este caso. El problema con este razonamiento es la fiabilidad de nuestras estimaciones, que estarían basadas en muestras muy pequeñas, lo que nos lleva a modelos con mala precisión predictiva. Este fenómeno se conoce como *overfitting* o sobreentrenamiento . Existen dos procedimientos alternativos para minimizar este problema:

1. Especificar criterios de parada y nodos terminales como se ha explicado. A estos criterios se les llama criterios pre-poda.
2. Cultivar un árbol muy grande y después podarlo como se verá en la siguiente sección.

## 2.3. Poda de arboles de regresión

Como hemos señalado anteriormente, una manera de corregir el sobreentrenamiento es el llamado podado de un árbol. Normalmente los criterios de parada suelen ser laxos, o incluso inexistentes dejando

solamente el criterio de nodo terminal, pues al ser una búsqueda ávida y no exhaustiva, es posible que un corte no tan bueno desde el punto de vista el del error, permita otro posterior mucho mejor. De ahí la importancia de dejar crecer el árbol para su posterior podado, e imponer criterios de parada laxos en la fase de crecimiento del árbol.

Si bien es cierto que existen muchas maneras de realiza esta poda, en CART se propone el *cost complexity pruning* (podado de coste de complejidad) también llamado *weakest link pruning*, o podado del eslabón más débil. De nuevo, una estimación de todos los posibles subárboles suele estar fuera del alcance en problemas reales.

Esto se basa en estimar mediante validación cruzada el error de un determinado subarbol. La validación cruzada es una técnica de remuestreo que nos permite evaluar la precisión de nuestro modelo. Surge como una mejora del *holdout method* o método de retención, en el que dividíamos el conjunto de datos en dos subconjuntos, uno son los datos de entrenamiento o *training set*, que se usa para validar el análisis en el otro subconjunto (datos de prueba o *test set*), de ahí el nombre de la técnica.

Hay muchas maneras de aplicar esta técnica, pero usaremos la más común, la validación cruzada de k bloques o *K-fold cross-validation*, en la que como su nombre indica, aplicaremos $k$ veces el método holdout sobre una partición aleatoria del training set en $k$ grupos de tamaño igual o similar y promediaremos sus errores.

Así pues, este tipo de podado se hace mediante la minimización sobre una cantidad formada por una estimación de la bondad del ajuste más una penalización de la complejidad del modelo. En el caso de los árboles de regresión trabajaremos con la siguiente cantidad, donde $\alpha$ es una parámetro real a escoger,

$$S_\alpha(T) = S(T) + \alpha \mid \tilde{T} \mid . \tag{2.6}$$

Como vemos en la formula anterior, pretendemos disminuir el error total S, pero sin aumentar de manera drástica el número de hojas del árbol, de ahí la importancia del término de penalización.

Podemos esquematizar este algoritmo de la siguiente manera, para más detalles ver el libro de Breiman [3].

---

ALGORITMO DE CONSTRUCCIÓN Y PODA DE UN ÁRBOL

---

**Datos:** $Y, X$, Criterio de parada, criterio de nodo terminal.

**Inicialización:** Construimos el árbol $T^1$ con el algoritmo de construcción normal ($\alpha^1 = 0$).

**Paso 1:** Seleccionar el nodo $t \in T^1$ que minimice

$$g_1(t) = \frac{S(t) - S(T_t^1)}{\tilde{T}_t^1 - 1},$$

y establecer $t_1 := t,\ \alpha^2 = g_1(t_1)$ y $T^2 = T^1 - T_{t_1}^1$.

**Paso i-ésimo:** Seleccionar el nodo $t \in T^i$ que minimice

$$g_i(t) = \frac{S(t) - S(T_t^i)}{\tilde{T}_t^i - 1},$$

y establecer $t_i := t,\ \alpha^{i+1} = g_i(t_i)$ y $T^{i+1} = T^i - T_{t_i}^i$.

**Paso Final:** Al llegar en el paso $n$ a la raíz ($T^n$ =Nodo Raíz) se tienen la secuencias

- $T^1 \supseteq T^2 \supseteq \cdots \supseteq T^n$.
- $\alpha^1 \leq \alpha^2 \leq \cdots \leq \alpha^n$.

Elegir $\alpha^k$, mediante validación cruzada minimizando $S_\alpha(T^k)$.

**FINAL DEL ALGORITMO**

---

Si bien nosotros optaremos por este modo de poda al ser el más popular y uno de los más estudiados, cabe añadir que existe métodos más simples para ello, como el ir eliminando corte a corte mediante validación cruzada mientras mejore la estimación del error de predicción.

## 2.4. Arboles de clasificacion

Como ya hemos dicho, Los arboles de clasificación se utilizan cuando la respuesta es categórica o cualitativa. Esta vez, en vez de minimizar la suma de los errores cuadráticos tendremos que hacerlo sobre otras cantidades ligadas a la nueva naturaleza de estas variables.

El primer cambio respecto a los árboles de regresión será sobre la cantidad $m_l$ asignada como la media aritmética en la hoja $l$. En este nuevo contexto, la clase con mayor número de participantes será la que ponga etiqueta a la hoja, y será el valor que le demos en la predicción a los individuos que caigan en la misma.

Del mismo modo, la suma de errores cuadráticos dejará de tener sentido, y su papel será tomado por la tasa de error de clasificación ($E$) que se estima mediante la proporción de observaciones que no pertenecen a la clase más común dentro de su hoja, es decir, la proporción de individuos mal clasificados en la hoja. Matemáticamente se puede expresar este error de clasificación en la hoja $l$, $E_l$, como

$$E_l = 1 - \max_k(\hat{p}_{lk}) \tag{2.7}$$

donde $\hat{p}_{lk}$ es la proporción de observaciones de la región u hoja $l$ que pertenecen a la clase $k$. Sin embargo, pueden emplearse también medidas asociadas a esta cantidad ya que suelen tener un mejor desempeño si el enfoque no es solamente el de mejorar la tasa de clasificación. Por ejemplo, sumando en las $K$ clases diferentes de individuos podemos utilizar el índice de Gini $G$:

$$G = \sum_{k=1}^{K} \hat{p}_{lk}(1 - \hat{p}_{lk}), \tag{2.8}$$

también llamado pureza de los nodos, pues un valor bajo indicaría que un nodo contiene principalmente observaciones de una única clase. Y de igual modo, es popular utilizar la entropía $D$ del mismo modo:

$$D = -\sum_{k=1}^{K} \hat{p}_{lk} \log \hat{p}_{lk}. \tag{2.9}$$

En cualquier caso, la nueva regla de asignación de las observaciones, y cualquiera de estos criterios ponderados por el número de observaciones de la hoja a la que pertenecen, $n_l$, pueden hacer el papel de la media y de $S(\cdot)$ en el algoritmo de construcción del árbol explicados en la sección 2.2 .

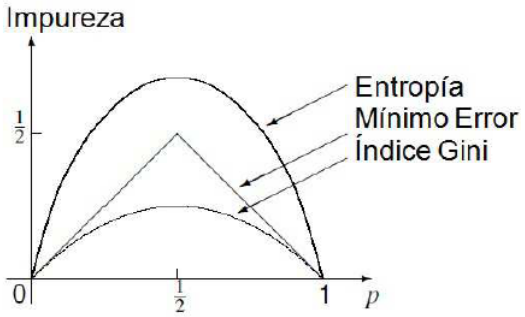


Figura 2.2: Compración de criterios de pureza.

## 2.5. Reglas Subrogadas y el tratamiento de Datos Missing

En determinadas situaciones, bien sea por error en la toma de datos, o por cualquier otro motivo, se debe realizar la predicción de su valor en la variable objetivo para un individuo en el cual una o más variables no están disponibles. Por ejemplo, cuando se mira la expresión génica (gene expression

microarray), los datos suelen contener muchos datos faltantes (missing) al haber una gran cantidad de variables y una probabilidad no despreciable de no medir correctamente dicho campo. No es posible por tanto desechar simplemente los individuos que no posean sus características completas puesto que estaríamos perdiendo gran parte de la muestra. Uno de los enfoques más característicos de las técnicas basadas en árboles para los datos faltantes es el de los cortes o reglas subrogadas.

Los árboles, tanto en clasificación como en regresión, abordan este problema buscando un corte alternativo cuando la variable que se debería utilizar en determinado nodo no se encuentra disponible en el dato de test. Para encontrar otro corte basado en otra variable, se buscará una división que lleve a una partición de los datos similar a la del corte óptimo. Del mismo modo, si dicho corte hiciera uso otra vez de una variable faltante, se procedería de igual manera hasta encontrar un corte posible para el dato en cuestión.

Hay que remarcar que en el proceso de búsqueda de una corte subrogado no se busca minimizar la medida de bondad de ajuste elegida en la construcción del árbol, si no que tratan de aproximar el resultado del mejor corte en cuanto a la división en dos partes de los datos. Es decir, el objetivo es replicar el mejor corte de la mejor forma posible en función de los individuos que van hacia una y otra rama del árbol. El motivo de este procedimiento es el hecho de que lo que se pretende es que los subsiguientes cortes del árbol original sigan creando unas particiones similares para las que fueron creados, ya que nada garantiza el hecho de que el segundo mejor corte divida los datos de una manera similar aunque su medida de bondad de ajuste esté muy cerca del primero.

# Capítulo 3

# Métodos de agregación

Tanto en clasificación como en regresión, existen estrategias basadas en la mezcla de modelos para mejorar la precisión. Estos métodos se denominan *ensemble methods* ya que combinan los resultados de más de un modelo.

Se puede distinguir dos familias:

- Averaging methods: Son métodos en los que se construyen predictores independientes y luego se combinan las predicciones en una etapa final. Como ejemplo, podemos encontrar el bagging, random forest...
- Boosting methods: A diferencia del anterior las estimaciones se crean secuencialmente, es decir, se va aprendiendo de las estimaciones anteriores. Entre estas técnicas se encuentran el boosting o ada boost entre otros.

Si bien un objetivo común de los *ensemble methods* aplicados sobre los árboles es el mejorar la precisión de los mismos, sus características dependen principalmente del tipo exacto de técnica que se aplique. En esta memoria trataremos algunas de ellas.

## 3.1. Bagging

*Bagging* es una técnica introducida por Leo Breiman en 1994 [1]. El nombre deriva de *bootstap aggregation* y fue uno de los primeros métodos efectivos de tipo *ensemble learning*. Aunque se diseñó originalmente para la clasificación y habitualmente se aplica en árboles de decisión, se puede aplicar tanto en modelos de regresión como de clasificación indistintamente. El primer paso de este algoritmo es generar $B$ distintos *boostrap trainning sets* o *boostraping*. El Bootstrap es un procedimiento propuesto inicialmente por Efron (1979) [4], basado en crear un número $B$ de muestras con remplazamiento del mismo tamaño que el conjunto inicial y ajustar un estadístico para cada una de ellas. En nuestro caso, se ajustará un árbol sin podar para reducir el sesgo todo lo posible, y aún siendo la varianza del mismo alta, al combinarlos entre sí la misma se reduzca.

En cada iteración bootstrap, se utilizan cada vez aproximadamente $\frac{2}{3}$ de los datos para ajustar el árbol correspondiente y el tercio restante se consideran observaciones *out of bag* (OOB). Este resultado viene de tomar el límite

$$\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = e^{-1} \approx 0{,}36 \tag{3.1}$$

pues $(1-\frac{1}{n})^n$ es la probabilidad de que un dato no se seleccione para una muestra *boostrap* de tamaño n cuando $n \to \infty$ ya que, a cada elección con remplazamiento la probabilidad de seleccionar un dato concreto es claramente $(1-\frac{1}{n})$, de donde al repetir de manera independiente $n$ veces se obtiene el resultado.

Por ultimo se promedian todas las predicciones de los distintos árboles obteniendo una predicción generalmente mucho más precisa que las de los modelos individuales.

El *OOB* es un método que nos sirve para estimar errores. En cada iteración *boostrap* hay datos que quedan fuera, es decir, que para cada árbol hay un conjunto *OOB* que no se usa para el entrenamiento. De este modo, *OOB* es el error de predicción medio para los datos que quedan fuera del ajuste del árbol en cada iteración, llamando error *OOB* al promedio de esta cantidad a lo largo de las $M$ iteraciones que componen el *bagging*.

La construcción de *bagging* se realiza de manera algorítmica del siguiente modo:

---

ALGORITMO DE BAGGING

---

**Inicialización:** El conjunto de entrenamiento $(X,Y)$ y M el numero de repeticiones *boostrap* que queremos crear.

**Paso 1:** Se construyen M muestras *boostrap* $B_1^*, B_2^*, B_3^* .... B_M^*$.
**Paso 2:** Para cada una de ella se calcula el estimador (árbol) $g_1^*, g_2^*, g_3^* .... g_M^*$.
**Paso 3:** El estimador *bagging* es $f_{bag}(x) = \frac{\sum_{m=1}^{M} g_m^*(x)}{M}$.
**FINAL DEL ALGORITMO**

---

## 3.2. Random forest

*Random forest*, también propuesto por Breiman en 2001 [2] supone un paso más en esta idea, ya que uno de los problemas de *bagging* ocurre cuando hay un predictor demasiado fuerte o dominante. Esto provoca que la varianza no se reduzca de manera notable al hacer el promedio de las predicciones. *Random forest* nos permite aplicar una estrategia que permite reducir la correlación entre árboles.

Al igual que en *bagging* vamos construyendo conjuntos de entrenamiento utilizando remuestreo. Por el contrario, en cada uno de los nodos de los árboles, sólo se permitirá al algoritmo elegir entre los $m$ predictores elegidos de forma aleatoria de entre los $p$ predictores considerados globalmente. En cada nodo se vuelve a calcular el conjunto de $m$ predictores permitidos. En regresión lo usual es elegir $m = p/3$.

Esquemáticamente el algoritmo es muy parecido a *bagging* quedando:

---

ALGORITMO DE RANDOM FOREST

---

**Inicialización:** El conjunto de entrenamiento $(X,Y)$ y M el número de repeticiones *boostrap* que queremos crear.

**Paso 1:** Se construyen M muestras *boostrap* $B_1^*, B_2^*, B_3^* .... B_M^*$.
**Paso 2:** Para cada una de ella se calcula el estimador (árbol) $g_1^*, g_2^*, g_3^* .... g_M^*$, donde en cada nodo que se construye:

- Se escogerá entre los $m$ predictores elegidos de forma aleatoria de entre los $p$ predictores considerados.
- Elegimos el mejor punto de corte para las $m$ variables.
- Añadimos dos nodos terminales no visitados, si se cumple el criterio de subdivisión de un nodo.

**Paso 3:** El estimador *Random Forest* es $f_{RF}(x) = \frac{\sum_{m=1}^{M} g_m^*(x)}{M}$.
**FINAL DEL ALGORITMO**

---

Con este proceso, *random forest* nos permite romper con la correlación entre los árboles intermedios ayudando a tener predicciones más fiables. Y es debido a la ruptura de esta correlación, que un predictor

dominante no va a poder ser elegido en $\frac{p-m}{p}$ de los nodos, con lo que el resto de predictores tendrán más oportunidades de participar en los nodos intermedios como criterio de división.

En este marco, veamos que ocurre con la varianza cuando aumenta el número de árboles, suponiendo varianzas iguales $\sigma$ con correlación $\rho$.

La varianza del estimador de *Random Forest* será

$$Var(\frac{1}{M}\sum_{i=1}^{M} g_i^*(x)) = \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j=1}^{M} Cov(g_i^*(x), g_j^*(x)) = \frac{1}{M^2}\sum_{i=1}^{M}\sum_{j\neq i}^{M} Cov(g_i^*(x), g_j^*(x)) + Var(g_i^*(x)) \quad (3.2)$$

donde sustituyendo obtenemos

$$\begin{aligned} Var(\frac{1}{M}\sum_{i=1}^{M} g_i^*(x)) &= \frac{1}{M^2}\sum_{i=1}^{M}((M-1)\rho\sigma^2 + \sigma^2) \\ &= \frac{M\rho\sigma^2(M-1) + M\sigma^2}{M^2} = \frac{\sigma^2\rho(M-1)}{M} + \frac{\sigma^2}{M} \\ &= \rho\sigma^2 + \frac{\sigma^2(1-\rho)}{M}. \end{aligned} \quad (3.3)$$

De esta manera, cuando $M \to \infty$, la varianza del estimador solo depende de $\rho\sigma^2$. Si la correlación es alta como puede ocurrir en *bagging* cuando hay variables cuya importancia es muy superior al resto, el valor de dicha cantidad será alto indicando una alta variabilidad. Mientras tanto, en *random forest* podemos disminuir artificialmente el valor del número de variables candidatas a formar parte de un corte $m$, y de este modo, al aleatorizar más la selección de variables, el valor de $\rho$ será menor y consecuentemente se reducirá la varianza en virtud de la ecuación anterior.

### 3.2.1. Importancia de las variables

En aplicaciones de minería de datos, las variables de predicción casi nunca son igual de relevantes. A menudo, solo unas pocas variables tienen una influencia sustancial en la respuesta, pero la gran mayoría de variables son irrelevantes y podrían igualmente no tener sentido incluirlas. Por eso a menudo es útil aprender la importancia de cada variable de entrada para predecir la respuesta. El mecanismo de construcción random forest y bagging nos permite establecer un baremo de la importancia de cada variable en la predicción final. Para ello se calcula el error de la muestra OOB. A continuación para cada variable de la muestra OOB, se permuta y se vuelve a calcular el error de la muestra OOB permutada. El resultado debería ser peor que para la muestra OOB original.

El procedimiento anterior se realiza para todos las variable y se calcula el promedio. Así las variables menos importantes deberían alterar menos la diferencia entre el error de la muestra OOB y el error de la muestra OOB permutada, que las variables importantes. Para mas información ver [8].

## 3.3. Boosting

Es un método para mejorar la capacidad predictiva de los modelos basados en árboles propuesto por Schapire [16] . A diferencia del *bagging*, el *boosting* es secuencial. Esto quiere decir que se van ajustando arboles al *training data* de manera iterativa, de tal manera que a cada paso se va poniendo más énfasis en las observaciones que no se han ajustado bien por los árboles anteriores. Para hacer esto, los algoritmos de boosting tratan de optimizar una función de pérdida como la propuesta en (2.5).

A diferencia de los otros modelos ya nombrados, en *boosting* nos interesa que los árboles no crezcan mucho y se adapten a los datos, ya que queremos que nuestro procedimiento vaya aprendiendo lentamente. El algoritmo general para el boosting tiene la siguiente forma:

---

ALGORITMO DE CONSTRUCCIÓN DE BOOSTING

---

**Datos:** $Y, X, 0 < \lambda < 1, M$.

**Inicialización:** El modelo $\hat{F}_0(x) = 0$.

**Paso 1:** Para $m \geq 1$ actualizamos los residuos $U_i = y_i - \lambda \hat{F}_{m-1}(x_i)$ y tomamos de forma aleatoria sin remplazamiento un conjunto de datos de $(X,U)$ al que le ajustaremos un árbol que lo denotaremos con $\hat{f}_m(x) = \hat{g}_{(X,U)}(x)$.

**Paso 2:** Se actualiza $\hat{F}_m(x) = \hat{F}_{m-1}(x) + \lambda \hat{f}_m(x)$.

**Paso 3:** Repetimos los pasos 1 y 2, $M$ veces.

**Paso final:** La salida del modelo es $\hat{F}(x) = \sum_{m=1}^{M} \lambda \hat{f}_m(x)$.

**FINAL DEL ALGORITMO**.

---

# Capítulo 4

# Ejemplo práctico

En esta parte se intentara dar un sentido a lo que se ha estudiado en las secciones anteriores mediante un ejemplo práctico, en el que trataremos con diversos datos de los jugadores de la liga española de 2015/16, toda la información se puede encontrar en https://es.whoscored.com/ .

El principal objetivo en esta parte, además de poner en práctica los métodos explicados anteriormente, será describir qué variables son las más importantes a la hora de formar el rating y entender cómo se podría calcular con un número reducido de variables y cuánto de buena sería esta aproximación. Elegimos esta variable por encima de la otras, a causa de que provoca más interés, ya que cuando fichas un jugador en una aplicación te interesa fichar los que más resultados te van a dar, lo cual esta relacionado con rating.

El conjunto de datos consta de 547 individuos y 38 variables que son:

- **Mins** : minutos jugados por jugador.
- **Position**; posición en la que juega cada jugador.
- **Player_number** : número del jugador.
- **Age** : la edad de cada jugador.
- **Tall** : altura de cada jugador.
- **Weight** : peso de cada jugador.
- **App_start** : veces puesto en la alineación principal.
- **App_dub** : veces puesto como reserva.
- **Goals** : goles marcados por los jugadores.
- **Assists** : asistencias de gol .
- **Yel** : tarjetas amarillas.
- **Red** : tarjetas rojas.
- **Rating** : variable numérica de la puntuación dada por la aplicación y es la variable a predecir.
- **Spg** : media de disparos por partido.
- **Ps_x** : pase más largo.
- **Motm** : hombre del partido.
- **Aw** : media de batallas en el aire ganadas por partido.

- **Tackles** : quitar el balón del jugador por partido.
- **Inter** : robos de balón por partido.
- **Fouls** : media de fallos por partido.
- **Offsides** : caer en posición de fuera de juego que resulta en un tiro libre para el equipo contrario.
- **Clear** : media de balones despejadas por partido.
- **Drb** : media de regates por partido.
- **Blocks** : media de bloqueos por partido.
- **Owng** : goles en propia puerta.
- **Keyp_x** : media de pases claves por partido.
- **Off** : media de fueras de juego por partido.
- **Disp** : media de pérdida de posesión por partido.
- **Unstch** : media de balones mal controladas por partido.
- **Keyp_y** : media de pases clave por partido.
- **Avgp** : media de pases por partido.
- **Ps_y** : pase más largo.
- **Crosses** : media de pases cruzados por partido.
- **Long_b** : media de longitud de pases por partido.
- **Team_name** : nombre de el equipo donde juega el jugador.

Una vez expuestas las variables, se debe comenzar siempre con la depuración de la base de datos a tratar.

## 4.1. Depuración de los datos

En esta sección hablaremos de los principales cambios hechos en las variables y también comentaremos la librería *rpart*, del software estadístico R, que es el que se empleará posteriormente.

Como podemos observar las variables tienen naturalezas muy diversas, existen datos que no tienen el formato deseado. Por eso realizaremos las trasformaciones que se creen convenientes para cada columna, además se aprovecha para corregir los 10 individuos que no tienen *weight*, lo cual es evidentemente un error.

Por otro lado establecemos dos nuevas variables *posinew* que hará el papel de *position*. Este cambio se debe a que hay demasiada diversidad de posiciones, así que las hemos reagrupado en tres grupos:1 son aquellos que juegan únicamente de delantero y de medio, 2 son aquellos que juegan de defensa y de medio, 3 son los porteros.

También hemos creado la variable *teamnew* que hace el papel de *team_ name*, este cambio se debe a que hay equipos con nombres largos y al representar nuestro árbol dificulta la visualización de estos. La regla decidida para simplificar los nombres no sigue una pauta concreta para todos, pero la regla predominante es que aquellos equipos formados por más de una palabra se toma la inicial de cada palabra, las formadas por una palabra se toma la primera letra y la siguiente consonante.

Después de la revisión de los datos se encontraron dos variables repetidas que se eliminaron *ps_x* y *keyp_y*.

La librería *rpart* nos permite construir modelos tanto de regresión como de clasificación y los modelos resultantes se pueden representar como árboles binarios. Sus principales argumentos son:

- *Formula*: Donde se define la variable dependiente y las variables independientes presentes en el conjunto de datos.

- *Method*: Aquí se define el método de particionamiento. Si es de regresión se pone *anova* y si es de clasificación *class*, depende de la naturaleza de la variable dependiente el escoger un método u otro.

- *Control*: Nos permitirá establecer criterios de parada y de nodo terminal, el argumento usado es **cp** con el que se controlara la complejidad del árbol, también tenemos **minsplit** el mínimo de observaciones que debe existir en un nodo para que se intente una división.

En la primera parte del anexo se puede ver como se ha realizado la depuración en el software R. Con todo estos cambios la base estará formada por 36 variables y 537 individuos, con la cual trabajaremos.

## 4.2. Modelado

Como el objetivo principal del trabajo va a ser predecir el *rating* de un jugador a partir del resto de variables, el análisis en esta sección estará claramente marcado por este objetivo.

Ya hemos visto que el conjunto de datos con el que vamos a trabajar consta de 36 variables y 537 individuos, pero antes de entrar en materia, dividiremos el conjunto de datos que tenemos en dos partes. Uno lo usaremos para entrenar, como training set, que constara de 358 jugadores y el otro para validar nuestro modelo formado por 179 jugadores. De igual manera, para evaluar si el desempeño de los árboles es relativamente bueno lo compararemos con un modelo de regresión el cual esta formado con las mismas variables que los árboles que ajustemos.

También comparará el $R^2_{adj}$ de cada modelo, que viene dado por:

$$R^2_{adj} = 1 - \frac{(1-R^2)(n-1)}{n-k-1}, \tag{4.1}$$

donde $n$ es numero de individuos en el conjunto de datos, $k$ es el número de variables usadas y $R^2$ es la variabilidad explicada.

Para crear nuestro primer modelo de árbol de regresión usaremos la librería *rpart* la cual hemos introducido en la sección anterior. Para crear nuestro primer modelo de árbol de regresión, una vez calculado y para hacernos una idea de su funcionamiento y sintaxis, seguiremos la rama que está situada a la derecha. Supongamos que tenemos un jugador que ha jugado más de 1163 minutos y ha metido más de 18 goles, según nuestro árbol su puntuación sería de **7.9** como podemos observar en la figura **4.1**.

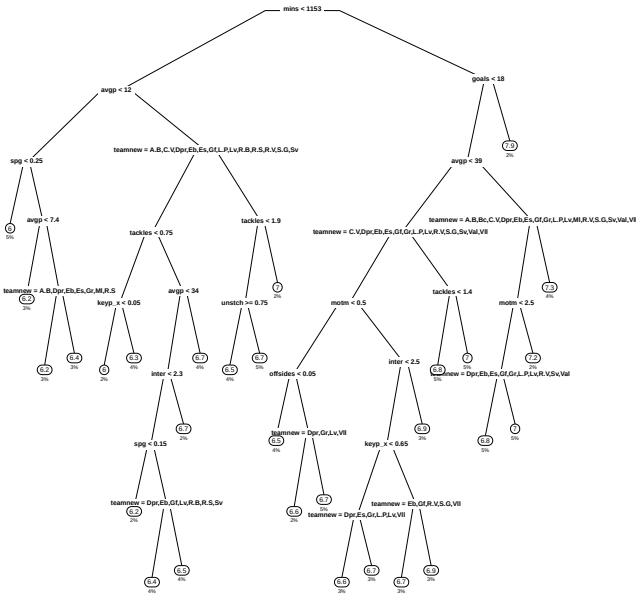


Figura 4.1: Árbol de regresión.

Se puede decir que es un árbol complejo aunque podemos entenderlo igualmente, pero veamos si se puede simplificar aún más. Para ello usaremos el método de poda explicado en **(2.3)**. Calculamos mediante validación cruzada el **cp** óptimo que para este árbol resulta ser de 0,01038142. Podemos ver el árbol podado resultante en la figura **4.2**.

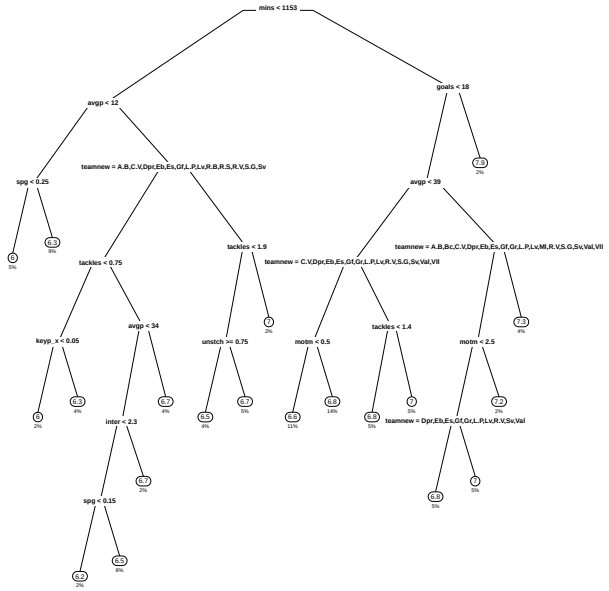


Figura 4.2: Árbol podado.

El árbol actual es mucho más sencillo, habiendo empleado un número de criterios más reducido. Como se puede observar en la figura antes del podado las variables como *inter, unstch* y *offsides* ya no están presentes en el árbol podado.

Aunque el árbol podado es más simple está aun muy lejos de ser bueno en comparación con el modelo de regresión como podemos observar en los siguientes gráficos comparando las predicciones:

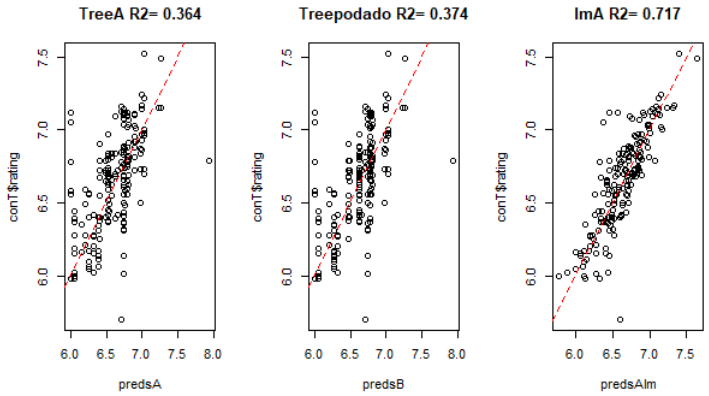


Figura 4.3: Comparación entre árboles y regresión.

Se puede observar que en regresión las predicciones tienen un gran desempeño en comparación con los árboles, es decir, el rendimiento de los árboles está siendo bastante pobre en comparación.

Veamos si los métodos de agregación explicados nos ayudan a tener una mejora predictiva. Empezaremos aplicando bagging. Para poder aplicar bagging usaremos la librería *ipred* cuya función bagging consta de los siguientes argumentos:

- *Formula*: Donde es elegida cual es la variable dependiente y cuales son las predictoras.
- *Method*: Aquí se define el método que queremos usar, *standard* para bagging y *Double* para doble bagging.
- *Control*: Nos permitirá modificar partes importantes del proceso. Solamente citaremos **cp** con el que se controla la complejidad del árbol y que mantenemos en su valor por defecto, en este caso 0.001.
- *Nbag*: El numero de muestras boostrap. En este caso 1000.

Al aplicar bagging se obtiene una mejora sustancial, como se puede ver a continuación en la figura 4.2.

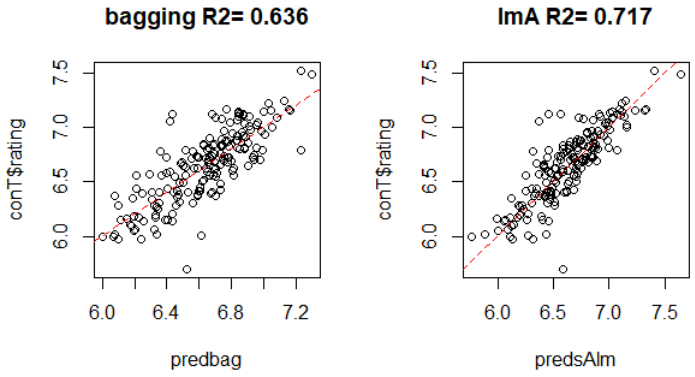


Figura 4.4: Comparación entre bagging y regresión.

Con este cambio se ha dado una gran mejora predictiva en comparación con los árboles de regresión observando como los métodos de agregación funcionan, pero en este caso sin alcanzar aún el desempe-

ño de la regresión lineal.

Para ajustar un método de Random Forest o Bosques Aleatorios, usaremos la librería *randomForests* la cual tiene los siguientes argumentos:

- *Formula*: Donde es elegida cual es la variable dependiente y cuales son las predictoras.
- *Mtry*: Número de variables tomadas al azar como candidatos en cada división. En regresión suele establecerse $\frac{p}{3}$.
- *N.tree*: El número de árboles. En nuestro caso 1000.
- *Importance*: Nos permite evaluar la importancia de los predictores.

Con esta propuesta se consigue de nuevo mejora predictiva. Esto, como se ha explicado en la teoría, puede ser causado por que hay algunas variables cuya importancia es mayor en magnitud que otras siendo esto la causa de las diferencias con el bagging. Como podemos observar en la figura **4.5**, la variabilidad explicada y la calidad de las predicciones se acercan sustancialmente a la de la regresión lineal.

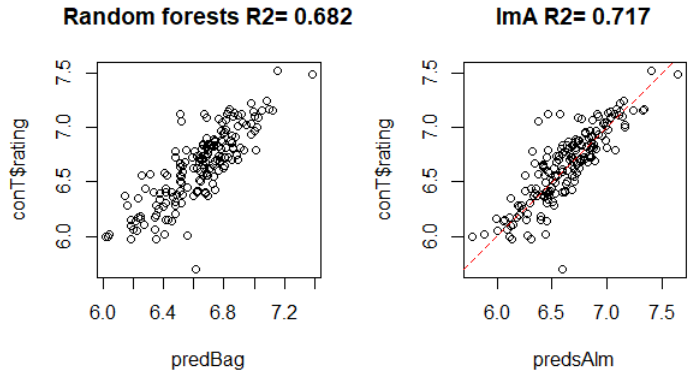


Figura 4.5: Comparación entre random forest y regresión.

En último lugar aplicaremos Boosting mediante la librería *gbm*, la cual contiene los siguientes parámetros en su función principal:

- *Formula*: Donde es elegida cual es la variable dependiente y cuales son las predictoras.
- *Distribution*: Aquí se elige que distribución usaremos. En este caso gaussian, pues es la que establece el error cuadrático como función de pérdida.
- *N.tree*: El número de árboles usados. Establecemos 1000.
- *shrinkage*: Tasa de aprendizaje. Por defecto 0,001.

Los métodos de agregación, por tanto, han supuesto una amplia mejora respecto a los árboles, sobre todo el boosting como se puede observar en la figura **4.6** donde el desempeño ya es notablemente mejor que en la regresión lineal.

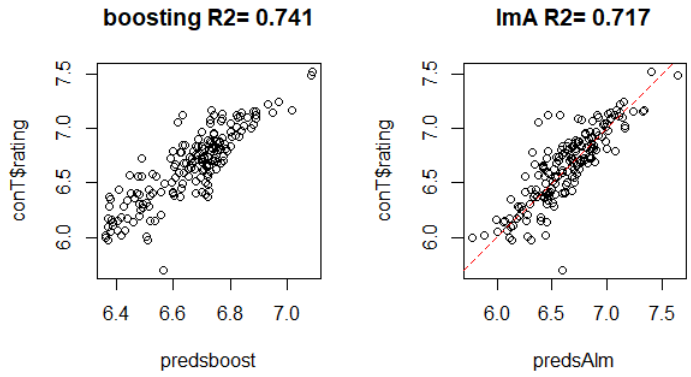


Figura 4.6: Comparación entre boosting y regresión.

Por último y como conclusión pondremos una tabla con las variables más importantes, ya que uno de nuestros objetivos era ver si se podía imitar o aproximar el modelo rating con menos variables además de conocer cuales son los factores que más influyen en la puntación de un jugador. En la tabla siguiente podemos comparar la importancia relativa de las variables más influyentes.

| Variables | Información Relativa |
|---|---|
| mins | 29.231 |
| avgp | 17.427 |
| motm | 17.333 |
| teamnew | 17.045 |
| apps_start | 6.796 |
| keyp_x | 2.075 |
| spg | 2.012 |
| goals | 0.875 |
| inter | 0.851 |
| apps_sub | 0.469 |
| longb | 0.366 |
| clear | 0.097 |
| ps_y | 0.093 |
| aw | 0.070 |

Cuadro 4.1: Tabla de importancia.

Como podemos ver *mins, avgp, motm, teamnew,app_start, goals* y *spg* son de las variables más importante, con ellas comprobaremos si mejora la capacidad predictiva de los árboles en comparación con los antiguos. El código de R para esta parte se puede consultar igualmente en el anexo.

## 4.3. Conclusión

Con todo esto tenemos la siguiente tabla donde podemos comparar los desempeños de las diferentes técnicas empleadas para los modelos con todas las variables, y también quitando las 25 variables menos importantes según la sección anterior.

Como podemos ver al quitar 25 variables, aumenta la bondad relativa al número de variables en el ajuste de los árboles. Al mismo tiempo en bagging, random forest, boosting y regresión se reduce notablemente, pues estos métodos hacen mejor uso de la información global de la base de datos (hay que recordar que la mayor parte de las variables no formaban parte de los árboles de regresión).

| Técnica | 8 Variables | 33 Variables |
|---|---|---|
| arbol | 0.466 | 0.364 |
| árbol podado | 0.477 | 0.374 |
| bagging | 0.609 | 0.636 |
| regresión | 0.605 | 0.718 |
| random forest | 0.651 | 0.682 |
| boosting | 0.665 | 0.741 |

Cuadro 4.2: comparación de $R^2$ ajustado.

Por otro lado, vemos que boosting supera en ambas ocasiones a la regresión lineal y al resto de métodos, mientras que en este caso los dos averaging methods quedan ligeramente por debajo.

En cuanto a la comparación de los dos averaging methods, parece que en este caso son superados en cierto modo por la regresión lineal. Sin embargo, Random Forest se comporta notablemente mejor en presencia de solo 8 variables, casi al nivel del boosting, y bagging supera por muy poco a la regresión lineal, pareciendo que esta incorpora algo mejor la información contenida en el resto de las variables.

Finalmente, en la cuestión relativa a la formación del rating de un jugador, hemos comprabado como las variables con mayor importancia relativa para puntuación de un jugador, tienen que ver con el número de minutos que disputa el mismo, la media de pases por partido así como la de disparos, los goles que mete (como parece ser intuitivo), el equipo que juega, y su número de titularidades.

# Bibliografía


[1] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24 (2): 123–140.

[2] Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32.

[3] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). *Classification And Regression Trees*. Wadsworth, Belmont CA.

[4] Efron, B. (1979): Bootstrap Methods: Another Look at the Jackknife. *The Annals of Statistics*.,7, pages 1–26.

[5] Friedman, J.H. (1991). Multivariate Adaptive Regression Splines, *The Annals of Statistics*.,19, 1–67.

[6] Faraway, J. (2002). Practical regression and ANOVA using R, Reproduction.

[7] He, Y., (2006). *Missing Data Imputation for Tree-Based Models*. Los Angeles, University of California, Tesis de Doctorado.

[8] Hastie, T., Tibsshirani, R., Friedman J.H., (2001).*The Elements of Statiscal Learning*. Springer.

[9] Karalic, A. (1992). *Employing linear regression in regression tree leaves*. In Proceedings of ECAI-92. WileySons.

[10] Kass , G.V. (1980). An Exploratory Technique for Investigating Large Quantities of Categorical Data, *Applied Statistics*.,29, 119–127.

[11] Light, R.J., Margolin, B.H. (1971). An analysis of variance for categorical data. *J. Amer. Statist. Assoc*.,66, 534–544.

[12] Larsen, D.R., Speckman, P.L. (2004). Multivariate Regression Trees for Analysis of Abundance Data. *Biometrics*.,60, 543–549.

[13] Messenger, R., Mandell, L. (1972). A modal search technique for predictive nominal scale multivariate analysis. *J. Amer. Statist. Assoc*.,67, 768–772.

[14] Morgan, J.N., Sonquist, J.A. (1963). Problems in the analysis of survey data, and a proposal. *J. Amer. Statist. Assoc*.,58, 415–434.

[15] Quinlan, J.R. (1992). Learning with continuous classes. In Adams Sterling, editor, *Proceedings of AI'92*,343–348. World Scientific.

[16] Schapire, R.E. (1990).The strenght of weak learnability. *In Machine Learning* ,5, 197–227.

[17] Sonquist, J.A., Baker, E.L., Morgan, J.N. (1974). Searching for structure: an approach to analysis of substantial bodies of micro-data and documentation for a computer program. Survey Research Center, University of Michigan.

# Anexo I: Código en R

## Librerías

```
library(readr)
library(rpart)
library(car)
library (rpart)
library(rpart.plot)
library(ISLR)
library(ipred)
library(randomForest)
library(dismo)
library(gbm)
library(survival)
library(splines)
library(lattice)
library(parallel)
```

## Conjunto de datos y depuración

```
laligaplayer <- read_csv("C:\\Users\\david\\Desktop\\base de datos R\\
    laligaplayer.txt",
col_names = FALSE, locale = locale())
View(laligaplayer)

#añadimos el nombre de las variables (38 variables)
names(laligaplayer)<-c('player_number', 'flag', 'name', 'age', '
    position', 'tall',
'weight', 'apps_start', 'apps_sub', 'mins', 'goals', 'assists',
'yel', 'red', 'spg', 'ps_x', 'motm', 'aw', 'rating', 'tackles',
'inter', 'fouls', 'offsides', 'clear', 'drb', 'blocks', 'owng',
'keyp_x', 'fouled', 'off', 'disp', 'unstch', 'keyp_y', 'avgp',
'ps_y', 'crosses', 'longb', 'team_name')

summary(laligaplayer)

## aqui cambiaremos 3 portero,2 defensa medio ,1 medio delantero
unique(laligaplayer$position)
laligaplayer<- within(laligaplayer, {
 posinew <- Recode(laligaplayer$position, '"FW"="1"; "M(CLR) FW"="1";
      "D(C) DMC"="2"; "M(C)" ="1"; "D(LR) M(CR)"="2"; "M(LR)"="1"; "D(CL
      )"="2"; "D(L)"="2"; "D(CR)"="2"; "D(C)"="2";
                       "D(L) M(L)"="2"; "AM(LR)"="1"; "MF"="1"; "D(C) M(
                          C)"="2"; "AM(CL)"="1"; "GK"="3"; "AM(L)"="1";
```

```
                          "AM(C)"="1"; "AM(L) FW"="1"; "AM(CLR) FW"="1";
                           "M(CLR)"="1"; "D(C) M(CLR)"="2"; "D(R)"="2";
                          "M(CR)"="1"; "D(LR)"="2";
                        "DMC"="2"; "DF"="2"; "AM(CR) FW"="1"; "M(CL)
                           "="1"; "D(R) DMC"="2"; "D(R) M(LR)"="2"; "AM(
                           CR)"="1"; "AM(CLR)"="1"; "D(CLR)"="2"; "D(R)
                           M(R)"="2"; "M(R)"="1"; "AM(LR) FW"="1"; "AM(R
                           )"="1"; "D(L) M(R)"="2"; "AM(C) FW"="1";
                      "D(LR) M(R)"="2"; "AM(CL) FW"="1"; "D(L) M(C)
                         "="2"; "D(CL) M(R)"="2"; "D(L) M(CL)"="2"; "M(L
                         ) FW"="1"; "M(L)"="1"; "M(LR) FW"="1"; "AM(R)
                         FW"="1";  ', as.factor.result=TRUE)
})
unique(laligaplayer$posinew)
unique(laligaplayer$team_name)

##simplificación de nombres
laligaplayer<- within(laligaplayer, {
 teamnew <- Recode(laligaplayer$team_name, '"Deportivo La Coruna"="
     Depor"; "Atletico Madrid"="A.M"; "Barcelona"="Bc"; "Celta Vigo" ="
     C.V"; "Athletic Bilbao"="A.B"; "Eibar"="Eb"; "Espanyol"="Es"; "
     Getafe"="Gf"; "Granada"="Gr"; "Las Palmas"="L.P";
                    "Levante"="Lv"; "Malaga"="Ml"; "Rayo Vallecano"="R.
                       V"; "Real Betis"="R.B"; "Real Madrid"="R.M"; "
                       Real Sociedad"="R.S"; "Sevilla"="Sv"; "Sporting
                       Gijon"="S.G"; "Valencia"="Val"; "Villarreal"="
                       Vill";  ', as.factor.result=TRUE)
})

unique(laligaplayer$teamnew)

##depuración
laligaplayer$position=as.factor(laligaplayer$position)
laligaplayer$team_name=as.factor(laligaplayer$team_name)
laligaplayer$team_name=as.factor(laligaplayer$teamnew)
laligaplayer<-laligaplayer[-c(46,127,261,349,417,425,429,460,462,517),]
laligaplayer<- laligaplayer[ ,!colnames(laligaplayer)=="ps_x"]
laligaplayer<- laligaplayer[ ,!colnames(laligaplayer)=="keyp_y"]
laligaplayer<- laligaplayer[ ,!colnames(laligaplayer)=="Continen"]
laligaplayer<- laligaplayer[ ,!colnames(laligaplayer)=="Continen"]
laligaplayer<- laligaplayer[ ,!colnames(laligaplayer)=="team_name"]
```

## Modelado

```
set.seed(1)
filasT<- sample(1:nrow(laligaplayer), floor(nrow(laligaplayer)/3))

conT<- laligaplayer[filasT,]
sinT<- laligaplayer[-filasT,]


##hacemos ahora un árbol
mi.arbol <- rpart(rating~.,data=sinT[,c(4,6:37)],
                  method= "anova",
                  control= rpart.control( cp=0.001))
```

```
12
13
prp(mi.arbol,extra=100,cex= .42,under=T,yesno=F)
15
16
17
18
##usemos validación cruzada para econtrar el cp optimo
set.seed(1)
matriz.predicciones <- xpred.rpart(fit = mi.arbol, xval = 10)
residuos.c <- (matriz.predicciones - sinT$rating[1:358])^2
apply(residuos.c, 2, sum)
CP.optimo2<-as.numeric(names(which.min(apply(residuos.c, 2, sum))))
CP.optimo2
##cp optimo sale
mi.arbol.podado2 <- prune(mi.arbol,cp=CP.optimo2)
28
prp(mi.arbol.podado2,extra=100,cex= .45,under=T,yesno=F)
30
31
##Las predicciones del modelo sobre las muestras con que se construyó:
par(mfrow=c(1,3))
lmA<- lm(rating~.,data=sinT[,c(4,6:37)])
predsAlm<-predict(lmA, newdata=conT)
36
predsA<-predict(mi.arbol, newdata=conT)
predsB<-predict(mi.arbol.podado2, newdata=conT)
39
modA<- lm(conT$rating~predsA)
plot(conT$rating~predsA, main=paste("TreeA R2=", round(summary(modA)$
    adj.r.squared,3)))
abline(0,1,lty=2,col=2)
43
modB<- lm(conT$rating~predsB)
plot(conT$rating~predsB, main=paste("Treepodado R2=", round(summary(
    modB)$adj.r.squared,3)))
abline(0,1,lty=2,col=2)
47
48
modAlm<- lm(conT$rating~predsAlm)
plot(conT$rating~predsAlm, main=paste("lmA R2=", round(summary(modAlm)$
    adj.r.squared,3)))
abline(0,1,lty=2,col=2)
52
##empezaremos ahora utilizar bagging
library(ipred)
bag<-bagging(rating~.,data=sinT[,c(4,6:37)], nbagg=1000, method=c("
    standard"), control= rpart.control( cp=0.001))
56
predbag<-predict(bag, newdata=conT)
modbag<- lm(conT$rating~predbag)
##dibujos
par(mfrow=c(1,2))
plot(conT$rating~predbag, main=paste("bagging=", round(summary(modbag)$
    adj.r.squared,3)))
abline(0,1,lty=2,col=2)
63
plot(conT$rating~predsAlm, main=paste("lmA R2=", round(summary(modAlm)$
```

```
    adj.r.squared,3)))
abline(0,1,lty=2,col=2)
66
67
68
##random forest
library(randomForest)
Baglaliga<- randomForest(rating~.,data=sinT[,c(4,6:37)],  mtry=11,
    importance=TRUE, ntree=1000)
predBag<- predict (Baglaliga,newdata=conT,n.trees=1000)
moddd<-lm(conT$rating~ predBag)
74
##dibujos
plot(conT$rating~predsAlm, main=paste("lmA R2=", round(summary(modAlm)$
    adj.r.squared,3)))
abline(0,1,lty=2,col=2)
plot(predBag, conT$rating,main=paste("Random forests R2=", round(
    summary(moddd)$adj.r.squared,3)))
79
plot(Baglaliga)
81
##veamos lass variables importantes
importance(Baglaliga)
varImpPlot(Baglaliga,cex= .45)
85
##boosting
laligaplayerboost<-gbm(rating~.,data=laligaplayer[,c(4,6:37)],
    distribution="gaussian",n.trees=1000,interaction.depth=4)
predsboost<-predict(laligaplayerboost, newdata=conT,n.trees=1000)
modboost<- lm(conT$rating~predsboost)
summary(laligaplayerboost)
91
##dibujos
plot(predsboost, conT$rating,main=paste("boosting R2=", round(summary(
    modboost)$adj.r.squared,3)))
94
modAlm<- lm(conT$rating~predsAlm)
plot(conT$rating~predsAlm, main=paste("lmA R2=", round(summary(modAlm)$
    adj.r.squared,3)))
abline(0,1,lty=2,col=2)
```