# Clasificación con árboles de decisión

Ramon Caihuelas Quiles

B2.332 - M2

**Ramon Caihuelas Quiles**

Licenciado en Ciencias de la Información UAB
Postgrado ETSEIB UPC
Máster Gestión Tecnologías Información Ingeniera La Salle URL
Doctorando Informática Ingeniera La Salle URL
Responsable del grupo de bases de datos y soporte al desarrollo del Área de Tecnologías de la Universidad de Barcelona
Colaborador docente de los estudios de Informática de Ingeniería La Salle URL y Universitat Oberta de Catalunya

El encargo y la creación de este material docente han sido coordinados por el profesor:

Julià Minguillón Alfonso (2016)

Primera edición: Octubre 2016

Av. Tibidabo, 39-43, 08035 Barcelona

# Índice

# Introducción

Cuando nos planteamos un proyecto de Minería de Datos, la aproximación al problema del descubrimiento de conocimiento en grandes volúmenes de datos es diferente a la aproximación científica clásica de formular una hipótesis y contrastarla. En nuestro caso vamos a presentar datos a un algoritmo sin ningún tipo de hipótesis o idea preconcebida, sino que queremos descubrir conocimiento desconocido previamente.

En el ámbito de la minería de datos existen dos tipos de enfoques muy diferentes cuando se parte de un objetivo de negocio y una gran cantidad de datos a procesar. El primer enfoque son los algoritmos que nos descubren las relaciones ocultas entre los datos y nos permiten generar modelos que ayudan a comprender un determinado escenario. Hablaríamos de modelos **descriptivos o explicativos**. El segundo enfoque son los algoritmos cuyo objetivo es poder predecir cuándo se producirá el hecho contenido en una variable llamada clasificadora. Estos modelos se llaman **clasificadores o predictivos**. Ubicados en esta segunda tipología se hallan los árboles de decisión. A los árboles de decisión que clasifican los datos del conjunto de entrada en función de una variable clasificadora categórica (es decir, que toma un conjunto finito de valores) se les llama árboles de clasificación. Si la variable clasificadora es continua, hablaríamos de árboles de regresión. Ambos tipos de árboles de decisión dan el nombre común CART al trabajo descrito por Breiman y Friedman en 1984.

**Bibliografía**

Breiman, Leo; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software. ISBN 978-0-412-04841-8.

## Objetivos

- Comprender el ciclo de vida, es decir, los pasos iterativos para poder generar un modelo clasificador de minería de datos mediante algoritmos de árboles de decisión.
- Conocer los conceptos de un modelo clasificador y en qué tipos de proyectos de minería de datos se utilizan.
- Conocer los conceptos y uso de los arboles de decisión: hojas, ramas, entrenamiento, poda.
- Preparación de los datos para su uso con un clasificador.
- Identificar las problemáticas más comunes de los arboles de decisión: el sesgo y el sobreentreno.
- Saber cómo evaluar la calidad del modelo generado.
- Extraer conocimiento del modelo construido: generación de reglas e interpretación.

# 1. Árboles de decisión

Los árboles de decisión son un tipo de algoritmos clasificadores/predictivos que tienen como características una estructura jerárquica recursiva formada por hojas, que son las etiquetas de clase que crean el modelo (toman una decisión), y las ramas que son los valores o condiciones que pueden tomar las variables y que llevan a otra rama u hoja.

**Lectura complementaria**

Usama Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. (1996). From Data Mining to Knowledge Discovery in Databases. *AI Magazine, 17(3).* `http://www.aaai.org/ojs/index.php/aimagazine/article/download/1230/1131`

Básicamente, un árbol de decisión toma como entrada un conjunto de datos, y sobre él aplica dos operaciones básicas. La primera, clasificar el conjunto de datos dando como resultado una etiqueta o valor de la clase que minimiza el error cometido, de acuerdo a algún criterio. Por ejemplo, si tenemos un conjunto de datos con diez personas, ocho mujeres y dos hombres, y el objetivo es clasificar una persona como hombre o mujer de acuerdo a ciertas variables (peso, altura, edad, color de cabello, etc.) el algoritmo podría decidir clasificar dicho conjunto con la etiqueta o clase "Mujer", cometiendo un error del 20 %, si es que dicho valor satisface cierto criterio preestablecido. La segunda operación consiste en dividir el conjunto de datos de entrada en dos subconjuntos disjuntos, seleccionando para ello el mejor criterio posible, usando las variables que describen cada elemento del conjunto de datos. Así por ejemplo, el algoritmo podría decidir usar alguna variable del conjunto de datos de personas para separar el conjunto de diez personas en dos (p.e. la altura), uno con siete mujeres, y el otro con dos hombres y una mujer. Entonces, para cada subconjunto, se vuelve a aplicar el mismo procedimiento recursivamente, es decir, decidir si etiquetarlo o no como una hoja terminal, o bien subdividirlo en dos subconjuntos para seguir con el mismo procedimiento. En el ejemplo proporcionado, el subconjunto con siete mujeres sería etiquetado con la clase "Mujer", no cometiendo ningún error, mientras que el otro subconjunto podría ser subdividido para intentar diferenciar la mujer restante de los dos hombres, mediante el uso de alguna otra variable. Este proceso se repite hasta que no hay ningún conjunto de datos que pueda o deba ser dividido.

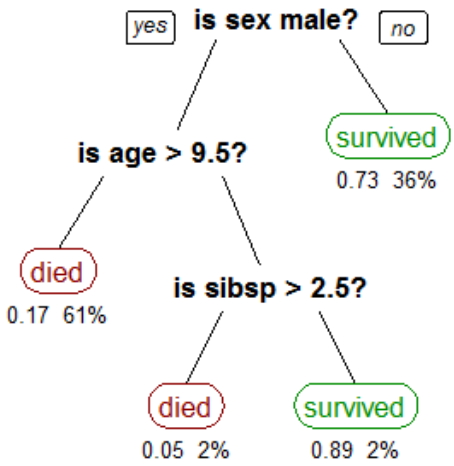
Es decir, existe un criterio global de parada, un criterio local de parada (es decir, decidir si un nodo debe ser subdividido o no), un criterio de etiquetado (asignar la clase que define mejor a un conjunto de datos) y, finalmente, un criterio de partición (decidir cómo subdividir un conjunto en dos o más subconjuntos, en el caso general). Normalmente se trabaja con árboles binarios, indicando que en cada paso el conjunto a dividir se subdivide en dos subconjuntos disjuntos. Además, lo habitual es que el criterio de partición utilice solamente una variable del conjunto de entrada en cada paso, creando hiperplanos ortogonales, no oblicuos, por cuestiones de simplicidad.

Ejemplo: El siguiente árbol representa un modelo clasificador aplicado a ciertas variables correspondientes a los pasajeros del Titanic, con el objetivo de decidir si un pasajero en concreto sobrevivió o no al hundimiento.

**Figura 1**

Observemos que cada hoja del árbol está acompañada de unos números que reflejan el porcentaje de supervivencia, y el porcentaje de registros asignados a dicha hoja, así como la decisión sobre la supervivencia o no (indicada por el color verde o rojo, respectivamente). Estos tres elementos nos ayudan a comprender el modelo creado y, sobre todo, permitirán la evaluación posterior de su calidad.

Figura 1. Clasificación de los pasajeros del Titanic en función de su supervivencia.




Fuente: Wikipedia.

Descripción de los elementos del árbol:

**Variables**: sex, age, sibsp

**Condiciones**: sex=male ?, age >9.5 ?, sibsp >2.5 ?

**Clasificación**: died / survived

En este simple ejemplo de clasificador ya podemos observar algunos conceptos que serán comunes en los árboles de decisión. Observemos la estructura del árbol: leemos el árbol de arriba a abajo, empezando por la hoja superior o raiz. La ubicación de las hojas es relevante, ya que indica un orden y determina su importancia en el modelo. Es decir, la primera variable en la construcción del modelo que determina el hecho de supervivencia o no es el sexo (variable sex), la segunda es la edad (age) y la última en importancia es la presencia de mujeres y hermanas (sibsp, de *siblings and parents*). Este orden se determina mediante el cálculo de la relevancia de cada variable con respecto al objetivo propuesto, usando diferentes medidas basadas en la entropía o cantidad de información capturada por cada variable. Supongamos que el fichero utilizado para construir este clasificador contiene otras variables, como por ejemplo la nacionalidad del pasajero. Si dicha variable no aparece en el árbol construido es que el algoritmo la ha descartado por su escasa capacidad predictiva, optando por otras variables más relevantes. En cada condición se evalúan todas las variables disponibles y se busca el valor que optimiza un criterio de partición de los datos de entrada, con el objetivo de

minimizar el error cometido en cada nivel.

Es importante destacar que, como pasa en la mayoría de modelos basados en árboles de decisión, sólo se evalúa una variable en cada condición, es decir, todos los hiperplanos que separan el conjunto de datos de entrada en dos son ortogonales a alguno de los ejes o dimensiones. Esto simplifica la interpretación del modelo pero puede dificultar la creación de modelos en casos que los datos siguen distribuciones no ortogonales u oblicuas, como suele suceder en la mayoría de casos. Para solventar este problema pueden usarse algoritmos de extracción de características para incorporar nuevas variables en el conjunto de datos original que representen mejor la estructura interna de dicho conjunto de datos.

Observemos también que cada hoja está acompañada de unos números que reflejan la probabilidad de supervivencia y el porcentaje de observaciones que çaen.en dicha hoja, es decir, que son clasificadas de acuerdo a las condiciones que llevan a dicha hoja. Por ejemplo, hay una hoja que clasifica como no supervivientes al 61 % de las personas que viajaban en el Titanic, concretamente los hombres con una edad mayor de 9 años y medio.

Finalmente, observemos la rama derecha de la primera condición, es decir, las mujeres. El modelo directamente ya clasifica como supervivientes a las mujeres, acumulando un 36 % de observaciones y con una capacidad predictiva del 0.73 (cometiendo un error en el 27 % de los casos). ¿Por qué no aparece ninguna condición más como la edad, que sí aparece en el caso de los hombres? Existe un concepto en los árboles de decisión llamado poda, mediante el cual si una rama tiene escasa capacidad predictiva se elimina (se poda). El objetivo es presentar un modelo más compacto, legible y claro, estableciendo un compromiso entre precisión y complejidad. No obstante, esto también puede ser un indicador de que el conjunto de entrada no dispone de todas las variables relevantes para responder la pregunta que se desea responder mediante el clasificador.

Estos conceptos aquí citados los profundizaremos con el caso de uso que se utilizará a lo largo de este módulo.

## 2. Caso de uso

**Objetivo de negocio:** Con el objetivo de mejorar los resultados académicos docentes, el Ministerio de Educación de un supuesto país nos encarga crear un modelo para poder predecir con antelación cuándo un alumno no va a superar una asignatura. Con este modelo predictivo funcionando en sus escuelas, los docentes contarían con una herramienta que les advertiría de cuándo se producen ciertos patrones que podrían llevar a un alumno a suspender una asignatura.

Para alcanzar el objetivo de negocio, se propone crear diversos modelos con diferentes implementaciones de árboles de decisión en R. Dichos modelos se analizarán y estudiarán para encontrar la mejor solución de acuerdo a ciertos criterios.

## 3. Preparación de los datos

Los datos con los que realizaremos los experimentos han sido escogidos y descargados del repositorio de *Machine Learning* de la Universidad de California, Irvine (UCI). Se trata de un repositorio abierto donde los donantes (investigadores) comparten ficheros de datos para usos académicos y docentes relacionados con *machine learning*.

El conjunto de datos está dividido en dos ficheros, donde cada uno recoge una serie de variables socio-demográficas y académicas de los alumnos. Cada fichero se corresponde a una asignatura.

**Enlace de los datos**

Para más información consultar: `https://archive.ics.uci.edu/ml/datasets/STUDENT+ALCOHOL+CONSUMPTION`

El primer bloque de tareas se realiza con un simple editor de texto. Este bloque de tareas está orientado a modificar los datos de origen para que encajen con el objetivo de esta guía. Por lo tanto, no los consideraremos como tarea propiamente dicha del proyecto de minería de datos, sino de la fase de preprocesado.

- El primer paso consiste en crear un único fichero csv a partir de los dos existentes, concatenándolos.
- El segundo paso consiste en cambiar los valores de algunas variables que, aunque son nominales, están representadas con un valor numérico (usado como índice). Con ello buscamos mejorar la lectura de los diferentes análisis de los datos y evitar que algunos algoritmos las traten como valores numéricos por omisión.

En nuestro caso queremos estudiar el hecho de superar o no el curso. Procedemos a generar una variable nueva a partir de la variable *G3*. Esta variable recoge la nota final de la asignatura en una escala de 0 a 20. La nueva variable contiene el valor Si/No aplicado a cada registro, donde Si implica un valor en *G3* igual o superior a 10 y No para un valor inferior a 10. Esta nueva variable la llamaremos *Supera* y nos indicará si un alumno ha superado o no el curso.

## 4. Creación de un árbol de decisión con el algoritmo C5.0

Existen múltiples algoritmos que construyen un árbol de decisión a partir de un conjunto de datos de entrada que contiene una variable que hace de objetivo de clasificación. Estos algoritmos se diferencian en función de los diferentes criterios de parada y partición que utilizan. Entre otros, en R disponemos de una implementación del algoritmo C5.0 de Ross Quinlan. Este algoritmo tiene un elevado grado de madurez al cual se ha llegado a través de sus diferentes versiones en el tiempo, como ID3, posteriormente C4.5 y hasta la actual llamada C5.0.

**Lecturas complementarias**

Algoritmo C4.5/C5.0 de Ross Quinlan: `https://es.wikipedia.org/wiki/C4.5`

Tutorial de C5: `https://www.rulequest.com/see5-unix.html`

Packages de R relacionados con árboles de decisión: `https://cran.r-project.org/view=MachineLearning`

Veamos cómo construir un árbol de decisión con el algoritmo C5.0. Primero cargamos los datos y la librería necesaria:

```
# Carga del fichero. El camino de acceso al fichero es relativo en cada ordenador
Alumnos_usos_sociales <-read.csv("/run/media/ramon/My Passport/BXXXXArbresDdecisio/student/Students_v20160908_2.csv",
                                 comment.char = "#")
# Carga C5.0
library(C50)
```

Código 4.1: Fichero C50.R

```
> library(C50)
```

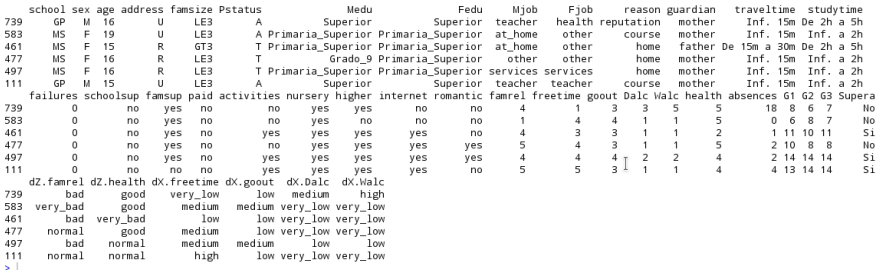
Código 4.2: Fichero C50.txt

Como precaución reordenamos los datos al azar. Muchas veces los registros en los ficheros pueden tener un cierto orden inherente que a nosotros no nos interesa para construir el modelo, dado que lo podría desvirtuar, especialmente cuando el conjunto de datos se divide en dos, uno de entrenamiento y otro de test (o a veces también llamado de validación). La reordenación la hacemos con las siguientes líneas de código:

```
# Reordenar al azar los datos
Alumnos_usos_sociales <- Alumnos_usos_sociales[sample(nrow(Alumnos_usos_sociales)),]
# Visualizamos el resultado
head(Alumnos_usos_sociales)
```

Código 4.3: Fichero C50.R

Figura 2.



```
    school sex age address famsize Pstatus            Medu            Fedu     Mjob     Fjob     reason guardian   traveltime  studytime
739     GP   M  16       U     LE3       A         Superior         Superior  teacher   health reputation   mother     Inf. 15m De 2h a 5h
583     MS   F  19       U     LE3       A Primaria_Superior Primaria_Superior  at_home    other     course   mother     Inf. 15m  Inf. a 2h
461     MS   F  15       R     GT3       T Primaria_Superior Primaria_Superior  at_home    other       home   father De 15m a 30m De 2h a 5h
477     MS   F  16       R     LE3       T          Grado_9 Primaria_Superior    other    other       home   mother     Inf. 15m  Inf. a 2h
497     MS   F  16       R     LE3       T Primaria_Superior Primaria_Superior services services       home   mother     Inf. 15m  Inf. a 2h
111     GP   M  15       U     LE3       A         Superior         Superior  teacher  teacher     course   mother     Inf. 15m  Inf. a 2h
    failures schoolsup famsup paid activities nursery higher internet romantic famrel freetime goout Dalc Walc health absences G1 G2 G3 Supera
739        0        no    yes   no         no     yes    yes       no       no      4        1     3    3    5      5        18  8  6  7      No
583        0        no    yes   no         no     yes     no       no       no      1        4     4    1    1      5         0  6  8  7      No
461        0        no    yes   no        yes     yes    yes      yes       no      4        3     3    1    1      2         1 11 10 11     Si
477        0        no    yes   no         no     yes    yes      yes      yes      5        4     3    1    1      5         2 10  8  8      No
497        0        no    yes   no        yes     yes    yes      yes      yes      4        4     4    2    2      4         2 14 14 14     Si
111        0        no     no   no        yes     yes    yes      yes       no      5        5     3    1    1      4         4 13 14 14     Si
    dZ.famrel dZ.health dX.freetime dX.goout  dX.Dalc  dX.Walc
739       bad      good     very_low      low   medium     high
583  very_bad      good       medium   medium very_low very_low
461       bad  very_bad          low      low very_low very_low
477    normal      good       medium      low very_low very_low
497       bad    normal       medium   medium      low      low
111    normal    normal         high      low very_low very_low
> 
```

.

El primer paso es escoger qué variables utilizaremos para construir el árbol. No es buena idea incluir campos con valores únicos para cada registro, como los identificadores, ni variables categóricas con valores muy dispersos, a menos que se preprocesen y se agrupen valores, reduciendo su variedad. Como siempre, una inspección del conjunto de datos original es muy recomendable, mediante el uso de descriptores estadísticos y visualizaciones sencillas. Es preferible también ensayar con diferentes variables y analizar los resultados obtenidos, mejor que presentar todas las variables a la vez para construir el modelo. El uso de demasiadas variables al mismo tiempo generalmente produce árboles muy complejos y difíciles, cuando no imposible, de interpretar. En nuestro caso escogeremos de entrada las siguientes variables: *sex, age, studytime, failures, schoolsup, famsup, absences*. Esta elección es arbitraria y con el fin de obtener un árbol adecuado para una finalidad docente, no la de resolver el problema originalmente propuesto. De hecho, algunas de las decisiones tomadas persiguen mostrar las limitaciones del proceso, obteniendo árboles de decisión que no serían adecuados para su utilización en un entorno real. Así:

```
# Asignamos a X las variables para construir el arbol
X <- Alumnos_usos_sociales[,c("sex","age","studytime","
    failures","schoolsup","famsup","absences")]
head(X)
```

Código 4.4: Fichero C50.R

```
  sex age  studytime failures schoolsup famsup absences
1   F  18  De 2h a 5h        0       yes     no        4
2   F  17  De 2h a 5h        0        no    yes        2
3   F  15  De 2h a 5h        0       yes     no        6
4   F  15 De 5h a 10h        0        no    yes        0
5   F  16  De 2h a 5h        0        no    yes        0
6   M  16  De 2h a 5h        0        no    yes        6
```

Código 4.5: Fichero C50.txt

El algoritmo para poder generar el modelo precisa de dos variables. Una es la matriz o tabla X que contiene los datos de generación del modelo, y la otra es la variable Y que contiene el objetivo con el cual clasificamos. En este caso se trata de la variable *Supera* y que toma los valores Si/No, la cual indica si el estudiante ha aprobado o no la asignatura.

Recordemos que la distribución de los valores de la variable *Supera* es de 230 alumnos que no superan la asignatura y 814 que sí la superan, existiendo por lo tanto un sesgo importante. Si decidiéramos que todos los estudiantes superan la asignatura (usando un clasificador trivial), estaríamos cometiendo un error del 22,0 %, es decir, para todos aquellos casos en los que no superan la asignatura.

```
# Asignamos a Y si supera o no el curso
Y <- Alumnos_usos_sociales[,34]
```

Código 4.6: Fichero C50.R

```
> Y <- Alumnos_usos_sociales[,34]
```

Código 4.7: Fichero C50.txt

El siguiente paso es repartir los datos en dos conjuntos disjuntos. El primer conjunto, llamado de entrenamiento, se lo proporcionaremos al algoritmo para crear el modelo. El segundo, llamado de test o de validación, lo reservamos para evaluar la calidad del modelo generado, es decir, para comprobar cómo clasifica datos que no han sido utilizados durante el entrenamiento y poder medir si el árbol construido sufre de sobreentrenamiento. La proporción de los registros asignados a cada conjunto es arbitraria y depende, entre otros factores, del número de datos total disponible. Normalmente se utiliza la conocida como regla de los dos tercios, donde el conjunto original se particiona en dos, uno de entrenamiento con dos tercios de los datos de entrada y el resto para el conjunto de test. Este porcentaje es arbitrario y puede oscilar alrededor de dicho valor. Se trata de un compromiso entre disponer del máximo de observaciones para construir un árbol de decisión lo más preciso posible y disponer de suficientes observaciones para validar el árbol de decisión construido.

**Lectura complementaria**

Cross Validation: `https://es.wikipedia.org/wiki/Validación_cruzada`

En este caso, de acuerdo con el tamaño del conjunto de datos original de 1044 elementos, usaremos 800 elementos para el entrenamiento y el resto (244) para validación, seleccionados aleatoriamente. Este método de partición es sencillo pero presenta problemas, dada la dificultad de que las variables del conjunto original y sobre todo la variable clasificadora estén presentes en las proporciones correctas (o al menos parecidas) en los dos conjuntos. Si los conjuntos de entrenamiento y de test son muy diferentes, el árbol construido con el primero seguramente será muy ineficiente para datos nuevos, como los del segundo (el conjunto de test).

Para evitar este problema, una de las técnicas más utilizadas y con mayores garantías es el método llamado *cross-validation*. Este método consiste en dividir el conjunto de datos en diferentes subconjuntos y utilizarlos todos menos uno para crear el modelo y validarlo con el restante, promediando el error cometido en cada caso. La generalización se conoce como *N-fold cross-validation*, donde $N$ es el número de subconjuntos en los cuales se subdivide el conjunto de entrada (a veces también se usa $K$ para este parámetro), usando $N-1$ para el entrenamiento y 1 para la validación. Entonces se construyen $N$ modelos diferentes y se promedian los resultados obtenidos. Es típico usar $N = 3$, pero para conjuntos grandes con muchas observaciones se puede llegar hasta valores de $N = 10$. En el caso que $N$ coincida con el número de elementos del conjunto de entrada,

se da el caso de lo que se conoce como *Leave-One-Out cross-validation*, recomendada para conjuntos de entrada pequeños.

Por simplicidad, en este ejemplo usaremos una única partición del conjunto original en dos, de acuerdo a lo comentado anteriormente:

```
# conjunto de entreno
trainX <- X[1:731,]
trainY <- Y[1:731]
# conjunto de test
testX <- X[732:1044,]
testY <- Y[732:1044]
```

Código 4.8: Fichero C50.R

```
> # Asignamos a Y si supera o no el curso
> Y <- Alumnos_usos_sociales[,34]
> # conjunto de entreno
> trainX <- X[1:800,]
> trainY <- Y[1:800]
> # conjunto de test
> testX <- X[801:1044,]
> testY <- Y[801:1044]
```

Código 4.9: Fichero C50.txt

Creamos el modelo y observamos el resultado:

```
# Generamos el modelo y observamos el resultado
model <- C50::C5.0(trainX,trainY)
summary(model)
```

Código 4.10: Fichero C50.R

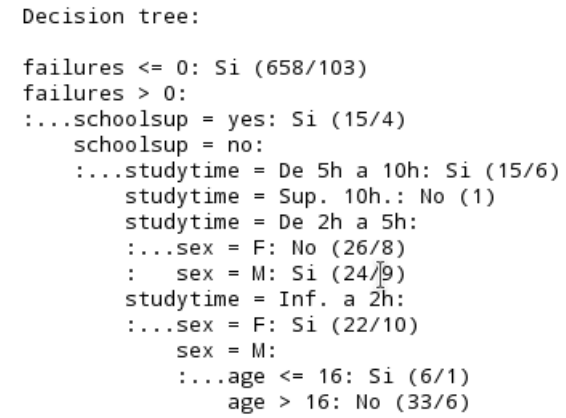
Figura 3.

```
C5.0 [Release 2.07 GPL Edition]
-------------------------------

Class specified by attribute `outcome'

Read 800 cases (8 attributes) from undefined.data

Decision tree:

failures <= 0: Si (658/103)
failures > 0:
:...schoolsup = yes: Si (15/4)
    schoolsup = no:
    :...studytime = De 5h a 10h: Si (15/6)
        studytime = Sup. 10h.: No (1)
        studytime = De 2h a 5h:
        :...sex = F: No (26/8)
        :   sex = M: Si (24/9)
        studytime = Inf. a 2h:
        :...sex = F: Si (22/10)
            sex = M:
            :...age <= 16: Si (6/1)
                age > 16: No (33/6)


Evaluation on training data (800 cases):

           Decision Tree
         ----------------
         Size      Errors

            9  147(18.4%)   <<


          (a)   (b)    <-classified as
         ----  ----
           46   133    (a): class No
           14   607    (b): class Si


        Attribute usage:

        100.00% failures
         17.75% schoolsup
         15.88% studytime
         13.88% sex
          4.88% age


Time: 0.0 secs
```

### 4.1. Lectura del modelo

Analicemos el resultado por cada bloque de información. Empecemos por el árbol generado en forma textual:

Figura 4.




.

¿Qué nos dice nuestro árbol? El primer nodo o condición utiliza la variable *failures*, así que las primeras ramas se toman en función de los valores de *failures* ≤ 0 o *failures* > 0:

```
failures <= 0: Si (658/103)
```

Código 4.11: Fichero C50.txt

En el primer caso, la rama *failures* ≤ 0 nos dice directamente que si el estudiante no ha suspendido anteriormente superará la asignatura. De este hecho hemos de extraer el conocimiento de que la variable con mayor capacidad de discriminación o mayor capacidad predictiva con respecto al problema propuesto es *failures*, la primera escogida por el algoritmo. También hemos de pensar que todas las hojas y ramas por debajo de *failures* ≤ 0 no son importantes, ya que no tienen capacidad discriminante fuerte y por eso el algoritmo ha decidido no continuar por dicha rama. Tenemos, pues, una primera rama que nos lleva a una hoja que ya clasifica directamente. Es importante remarcar que esto no es lo habitual, siendo necesarios diversos niveles de profundidad para conseguir una hoja suficientemente pura que clasifique a un porcentaje relevante de elementos del conjunto de entrenamiento.

¿Qué más información tenemos al respecto? Nuestro árbol nos muestra que en esta primera hoja se acumulan 658 observaciones sobre 800 que tenemos en el conjunto de entrenamiento, y que de éstas, 103 sobre 658 no coinciden con la clasificación propuesta (Si), lo cual es una tasa de error del 15,6 %. Se trata, por lo tanto, de una primera decisión importante que reduce el error cometido si clasificáramos todo el conjunto de datos como Si (recordemos que era del 22,0 %). Es decir, la hoja etiquetada como Si a la cual se llega con *failures* ≤ 0 clasifica el 82,2 % del conjunto de entrada cometiendo un error del 15,6 %.

Acabemos de leer el árbol. Profundizamos en la rama de *failures* > 0, es decir, aquellos estudiantes que son repetidores de la asignatura. En este caso la siguiente variable con capacidad predictiva es *schoolsup*. Recordemos que se corresponde a tener soporte de

la escuela, o algún tipo de actividad de refuerzo. Si la rama toma el valor Si, nuestro árbol indica que se supera la asignatura, para las 15 observaciones asignadas de las cuales no se clasifican bien 4. Por el contrario, si el estudiante no tiene un refuerzo escolar pero la dedicación en estudio (*studytime*) es de 5 a 10 horas, entonces supera la asignatura, aunque cometiendo 6 errores para los 15 casos contemplados en dicha hoja. La siguiente rama es un poco extraña, ya que con mayor esfuerzo (horas de estudio superior a 10) suspende. Cuando vemos el número de observaciones asignadas (solamente 1) comprendemos que es una respuesta malintencionada o errónea por parte del estudiante en el momento de responder la encuesta. De hecho, un análisis estadístico descriptivo de esta variable muestra ya valores extraños (posibles *outliers*). La siguiente rama es la de una cantidad de horas de estudio media, de 2 a 5 horas. En este caso el árbol nos indica que la siguiente condición a preguntar es el sexo del estudiante. Las 26 mujeres en esta hoja no superan la asignatura excepto en 8 casos. Los 26 estudiantes hombres sí que la superan, siendo 9 mal clasificados. Finalmente, si tomamos la rama de tiempo de estudio inferior a dos horas, la siguiente rama sigue utilizando el sexo como decisión. Esto nos indica que existe alguna relación entre sexo y horas de estudio, ya que el árbol está replicando la misma estructura de condiciones a diferentes niveles. Es un indicador de que quizás se podrían combinar las dos variables en una sola, o al menos realizar un análisis estadístico preliminar para observar su distribución conjunta. Si tomamos esta rama volvemos a tener unos resultados un poco extraños, como en el caso anterior. En este caso, si se trata de una estudiante mujer (22 observaciones), se supera la asignatura, cometiendo 10 errores (casi la mitad, un error muy elevado). En cambio, si se trata de un hombre, aparece una nueva condición usando la variable edad, donde la rama de edad inferior a 16 años conlleva superar la asignatura, mientras que si la edad es superior a 16 no se superará.

De esta lectura del árbol es interesante comprender diversos aspectos importantes comentados con anterioridad: primero, el algoritmo decide si seguir subdividiendo o no una hoja, con el objetivo de mejorar la clasificación realizada hasta el momento. Esto es un criterio de parada, el cual puede ser ajustado mediante el uso de parámetros, como por ejemplo el número mínimo de elementos en una hoja para ser subdividida o el error cometido. Segundo, el algoritmo decide qué variable utilizar en cada nivel de acuerdo a algún criterio (llamado de partición) relacionado con la capacidad discriminativa de cada variable. Finalmente, para una hoja, el algoritmo decide cuál es la mejor clasificación posible, teniendo en cuenta diferentes criterios como la proporción de elementos de cada clase, su probabilidad *a priori* o el coste de cometer un error de clasificación. En el ejemplo mostrado, con solo dos clases, se escoge como etiqueta o clase resultante la más popular en cada hoja, minimizando el error cometido.

En lenguaje natural, el árbol de decisión construido nos dice que, si se repite curso, el estudiante lo tiene mal para superar la asignatura. Con refuerzo escolar lo puede superar. Y si no tiene refuerzo escolar, cuanto mayor sea y menos estudie, peor lo tendrá, excepto en algunos casos detectados.

### 4.2. Calidad del modelo generado

Revisamos el árbol para comprobar que la suma de las observaciones en cada hoja del árbol coincide con la suma de las observaciones utilizadas para construir el modelo *658+15+15+1+26+24+22+6+33=800,* de forma que todos los elementos del conjunto de entrenamiento han quedado clasificados, resultado del proceso seguido durante la creación del árbol de decisión. También realizamos un simple cálculo manual para tener presente la tasa de error de cada hoja y la global, que queda de la siguiente forma:

Cuadro 2: Estudio de error cometido en cada hoja.

| Rama | Asignadas a la hoja | Mal clasificadas | Error (%) |
|---|---|---|---|
| 1 | 658 | 103 | 15,65 |
| 2 | 15 | 4 | 26,67 |
| 3 | 15 | 6 | 40,00 |
| 4 | 1 | 0 | 0,00 |
| 5 | 26 | 8 | 30,77 |
| 6 | 24 | 9 | 37,50 |
| 7 | 22 | 10 | 45,45 |
| 8 | 6 | 1 | 16,67 |
| 9 | 33 | 6 | 18,18 |
| | **800** | **147** | **18,38** |

Seguimos revisando el modelo generado:

```
Evaluation on training data (800 cases):

      Decision Tree
    ----------------
    Size      Errors

       9  147(18.4%)   <<
```

Código 4.12: Fichero C50.txt

Coincide con el cálculo hecho. El árbol tiene nueve hojas. La tasa de error puede parecer no muy grande, pero si observamos el cálculo de error por hoja podemos tener porcentajes inadmisibles. Además, el conjunto original tiene 814 estudiantes de una clase (los que superan) y 230 de la otra (los que no). Ya hemos visto que un árbol de decisión trivial formado por una sola hoja con la etiqueta Si (los que superan, mayoritarios) cometería un error del 22.0 %, no mucho peor. Por lo tanto, ¿Cómo de bueno es realmente el árbol de clasificación construido? Esto se puede calcular a partir de la matriz de confusión

que genera el algoritmo, formada por el número de elementos de la clase $i$ que se clasifican como $j$, para todos los valores posibles de $i$ y $j$ (en nuestro caso, $i$ y $j$ pueden tomar los valores Si y No):

```
     (a)    (b)    <-classified as
    ----   ----
      46    133    (a): class No
      14    607    (b): class Si
```

Código 4.13: Fichero C50.txt

Además, ¿Dónde se está cometiendo el error? ¿Es equivalente para las dos clases? Para saberlo utilizamos la matriz de confusión que nos dice cuantos elementos de cada clase son clasificados como elementos de otras clases. Si tomamos los estudiantes de la clase No, hay 133 casos mal clasificados como Si y 46 casos bien clasificados como No. Si tomamos la clase Si, hay 607 casos bien clasificados como Si y 46 mal clasificados como No. Por lo tanto, nuestro clasificador no puede considerarse adecuado, dado que no es capaz de discriminar correctamente los casos de la clase No, los cuales son mayoritariamente clasificados como Si:

| | Todos | Correctos | Incorrectos |
|---|---|---|---|
| NO | 1 | 1 | 0 |
| | 26 | 18 | 8 |
| | 33 | 27 | 6 |
| Sumatorio | **60** | **46** | **14** |
| SI | 658 | 555 | 103 |
| | 15 | 11 | 4 |
| | 15 | 9 | 6 |
| | 24 | 15 | 9 |
| | 22 | 12 | 10 |
| | 6 | 5 | 1 |
| Sumatorio | **740** | **607** | **133** |

Cuadro 4: Desglose de la matriz de confusión.

De la matriz de confusión se pueden extraer diferentes indicadores, muchos de ellos derivados de la práctica clínica, que explican como de sensible (*sensitivity*) y específico (*specificity*) es un clasificador, cuando una de las dos clases se asocia a la presencia de una condición, hablando también de falsos positivos (elementos que no satisfacen la condición pero son clasificados como tales) y falsos negativos (elementos que satisfacen la condición pero no son clasificados como tales). En nuestro caso, si la condición

**Lectura complementaria**

Sensibilidad y especificidad: `https://es.wikipedia.org/wiki/Sensibilidad_y_especificidad_(estadistica)`

fuera el hecho de no superar la asignatura, tendríamos 133 falsos negativos y 14 falsos positivos. Un buen clasificador presenta valores elevados para ambos indicadores al mismo tiempo, lo cual no es cierto en nuestro caso.

Terminamos de revisar el modelo generado:

```
Attribute usage:

100.00% failures
 17.75% schoolsup
 15.88% studytime
 13.88% sex
  4.88% age
```

Código 4.14: Fichero C50.txt

¿Qué nos indica esta información? Primero de todo, se puede observar que las variables *absences* o *famsup* son irrelevantes para la construcción del modelo y han sido descartadas (o, mejor dicho, no seleccionadas) por el algoritmo. La información que se aporta es el uso de las variables que ha utilizado el algoritmo con respecto al número de elementos evaluados de acuerdo a dicha variable. Así, la raiz del árbol, en este caso *failures*, siempre estará al 100 %, mientras que el resto es un indicador de la importancia en la construcción del modelo. Existen diferentes métodos para determinar la relevancia de cada variable, siendo éste el más sencillo.

### 4.3. Visualización del árbol de decisión

Vamos a generar ahora otro árbol distinto. Utilizaremos el mismo fichero de datos y tendremos la misma variable clasificadora, pero en este caso prescindiremos arbitrariamente de las variables *absences* y *famsup*, con el objetivo de generar un modelo más compacto y fácil de leer. Volvemos a repetir todos los pasos anteriores, generamos el modelo y lo mostramos en formato textual.

```
# ATENCION PARA PODER GENERAR EL ARBOL ELIMINA famsup y absences y repite pasos
X <- Alumnos_usos_sociales[,c("sex","age","studytime","failures","schoolsup")]
# Redefinimos el conjunto de entreno y el conjunto de test
trainX <- X[1:800,]
testX <- X[801:1044,]
trainY <- Y[1:800]
# Generacion modelo
model <- C50::C5.0(trainX,trainY)
```

Código 4.15: Fichero C50.R

Figura 5.

```
Decision tree:

failures <= 0: Si (675/81)
failures > 0:
:...schoolsup = yes: Si (18/5)
    schoolsup = no:
    :...failures > 1: No (39/13)
        failures <= 1:
        :...age <= 16: Si (17/6)
            age > 16: No (51/24)


Evaluation on training data (800 cases):

           Decision Tree
         ----------------
         Size      Errors

            5  129(16.1%)   <<


          (a)   (b)    <-classified as
         ----  ----
           53    92    (a): class No
           37   618    (b): class Si


        Attribute usage:

        100.00% failures
         15.62% schoolsup
          8.50% age
```

.

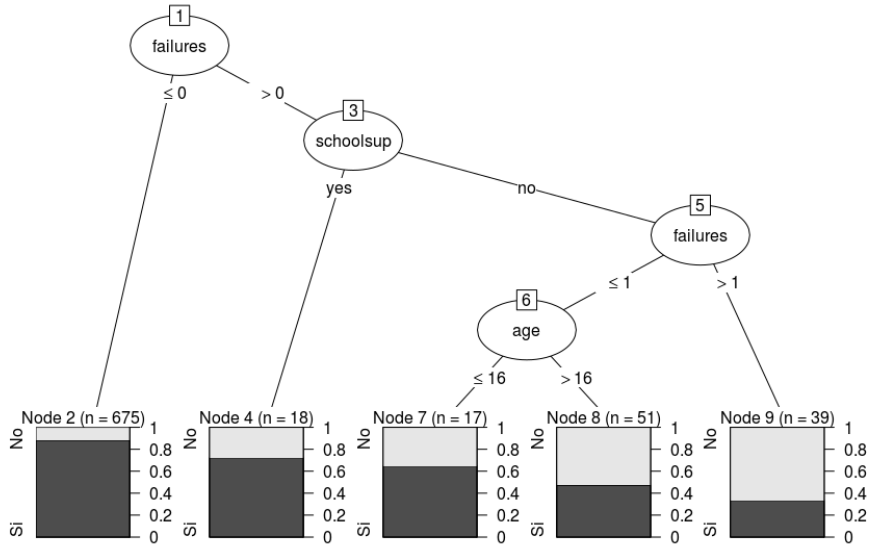
Ahora dibujamos el árbol obtenido:

```
summary(model)
# Dibujamos el arbol
```

Código 4.16: Fichero C50.R

La representación gráfica permite ver mejor la estructura en árbol de las decisiones tomadas para clasificar cada elemento hasta llegar a su clase final decidida en cada hoja: R/C50.R

Figura 6.




.

En este caso, el árbol obtenido ha variado, siendo más compacto y reduciendo el error global, aunque sigue siendo ineficaz para clasificar correctamente los elementos de la clase No. Observamos que el árbol en su forma gráfica es otra representación de la versión textual que facilita su lectura y comprensión, aunque puede ser complicado visualizar árboles de decisión grandes.

### 4.4. Extracción de reglas

Una de las bondades de los árboles de decisión es la capacidad de poder extraer los resultados obtenidos en forma de reglas fáciles de implementar en un sistema experto, simplificando la estructura del árbol de decision que puede llegar a ser muy compleja. Sobre el modelo que acabamos de crear en la figura **??** ejecutaremos el siguiente comando para extraer las reglas:

```
1
2 # Generacion del modelo con reglas
3 model <- C50::C5.0(trainX,trainY,rules=TRUE)
```

Código 4.17: Fichero C50.R

Figura 7.

```
Rule 1: (80/34, lift 3.2)
        age > 16
        failures > 0
        schoolsup = no
        ->  class No  [0.573]

Rule 2: (125/61, lift 2.8)
        failures > 0
        ->  class No  [0.512]

Rule 3: (575/81, lift 1.1)
        failures <= 0
        ->  class Si  [0.879]

Rule 4: (655/93, lift 1.0)
        failures <= 1
        schoolsup = no
        ->  class Si  [0.857]

Default class: Si


Evaluation on training data (800 cases):

             Rules
       ----------------
         No      Errors

          4  137(17.1%)   <<


        (a)   (b)    <-classified as
       ----  ----
         58    87    (a): class No
         50   605    (b): class Si


       Attribute usage:

       100.00% failures
        85.50% schoolsup
        10.00% age
```

Examinemos una regla con detalle:

```
Rule 1: (80/34, lift 3.2)
  age > 16
  failures > 0
  schoolsup = no
  ->  class No  [0.573]
```

Código 4.18: Fichero C50.txt

El número que identifica cada la regla es arbitrario y solo sirve para identificarla. Para la regla 1, sabemos que 80 ($n$) son las observaciones asignadas a esta regla y 34 ($m$) el número de observaciones que no se clasifican bien. Para calcular la fiabilidad de la regla cuando se aplique a una nueva observación, la precisión de la regla se calcula mediante la regla de sucesión, también llamada a veces el ratio de Laplace: $(n - m + 1)/(n + k)$, donde $k$ es el número de clases. En nuestro caso (80-34+1)/(80+2)=0,573.

**Lectura complementaria**

Regla de Sucesión: `https://es.wikipedia.org/wiki/Regla_de_sucesión`

Es importante observar que las reglas generadas no se corresponden exactamente con las hojas del árbol, sino que combinan diferentes condiciones. Además, para un mismo dato de entrada, pueden activarse más de una regla, por lo que el orden de aplicación es importante, siendo necesario tomar una decisión en función de su confianza, usando una votación ponderada, por ejemplo.

Las reglas se presentan ordenadas por clase y, dentro de cada clase, por confianza.

### 4.5. Podado del árbol

El algoritmo C5.0 construye el árbol en dos fases. En la primera fase se construye un árbol completo, intentando clasificar correctamente todos los elementos del conjunto de entrenamiento. Esto siempre será posible excepto en caso de que existan elementos del conjunto que sean iguales pero que estén clasificados usando una clase diferente, lo cual debe ser corregido antes de realizar el entrenamiento. En la segunda fase se revisa si alguna de las ramas generadas tiene una elevada tasa de error y no aporta nada seguir realizando particiones en dicha rama. Si ese fuese el caso, se colapsan los registros como una sola hoja en ese nivel y se recalcula la tasa de error. Con ello queremos evitar el sobreentrenamiento u overfitting, un problema típico de los árboles de decisión. Dicho problema consiste en que los modelos se sobreentrenan con los datos de construcción del modelo y no generalizan bien cuando se les presentan nuevos datos. Por ello es necesario siempre disponer de un conjunto de test con datos que no haya visto el modelo durante su construcción, para medir su capacidad de generalización y evitar el sobreentrenamiento.

En el siguiente ejemplo generamos nuestro árbol de decisión con la opción de poda activada (por defecto) y con la opción de poda desactivada y compararemos los resultados:

```
# Carga del fichero. El camino de acceso al fichero es
    relativo en cada ordenador
Alumnos_usos_sociales <-read.csv("/run/media/ramon/My
    Passport/BXXXXArbresDdecisio/student/Students_v20160908_
    2.csv",comment.char = "#")
# Carga C5.0
library(C50)
set.seed(5813)
# Reordenar al azar los datos
Alumnos_usos_sociales <- Alumnos_usos_sociales[sample(nrow(
    Alumnos_usos_sociales)),]
X <- Alumnos_usos_sociales[,c("sex","age","studytime","
    failures","schoolsup")]
# Asignamos a Y si supera o no el curso
Y <- Alumnos_usos_sociales[,34]
# conjunto de entreno
trainX <- X[1:800,]
trainY <- Y[1:800]
```

```
# conjunto de test
testX <- X[801:1044,]
testY <- Y[801:1044]
#Modelo
model <- C50::C5.0(trainX,trainY)
summary(model)
plot(model)
model <- C50::C5.0(trainX,trainY,rules=TRUE)
summary(model)
# Poda
modelP <- C50::C5.0(trainX,trainY,control=C5.0Control(
    noGlobalPruning=TRUE))
summary(modelP)
plot(modelP)
modelP <- C50::C5.0(trainX,trainY,rules=TRUE, control=C5.0
    Control(noGlobalPruning=TRUE))
summary(modelP)
```

Código 4.19: Fichero C50P.R

Veamos el modelo generado con poda:

```
failures <= 0: Si (667/91)
failures > 0:
:...failures > 2: No (18/3)
    failures <= 2:
    :...schoolsup = no: No (101/43)
        schoolsup = yes: Si (14/4)


Evaluation on training data (800 cases):

      Decision Tree
    ----------------
    Size      Errors

       4  141(17.6%)   <<


     (a)   (b)    <-classified as
    ----  ----
      73    95    (a): class No
      46   586    (b): class Si
```

R/C50P.txt

Se trata de un árbol con solamente 4 hojas. Ahora veamos el mismo árbol generado sin poda:

```
failures <= 0: Si (667/91)
failures > 0:
:...failures > 2: No (18/3)
    failures <= 2:
    :...schoolsup = no:
        :...age <= 18: No (76/28)
        :   age > 18: Si (25/10)
        schoolsup = yes:
        :...age <= 15: No (3/1)
            age > 15: Si (11/2)
```

```
Evaluation on training data (800 cases):

      Decision Tree
    ----------------
    Size      Errors

       6  135(16.9%)   <<


     (a)   (b)    <-classified as
    ----  ----
      65   103    (a): class No
      32   600    (b): class Si
```

R/C50P.txt

En este caso el árbol generado tiene 6 hojas. Observemos que la ramas podadas se corresponden en ambos casos a la edad. Este es un síntoma típico de los árboles de decisión, la replicación de condiciones en un nivel, indicando que la variable es también relevante en el nivel superior. Seguramente una combinación de ambas variables (en este caso, *age* y *schoolsup*) sería más eficiente, eliminando dicha replicación.

Comparamos las reglas extraídas con poda:

```
Rules:

Rule 1: (118/45, lift 2.9)
  failures > 0
  schoolsup = no
  ->  class No  [0.617]

Rule 2: (133/56, lift 2.8)
  failures > 0
  ->  class No  [0.578]

Rule 3: (782/153, lift 1.0)
  failures <= 2
  ->  class Si  [0.804]

Default class: Si
```

R/C50P.txt

Reglas extraídas sin poda:

```
Rules:

Rule 1: (8/2, lift 3.3)
  age <= 15
  failures > 0
  failures <= 2
  ->  class No  [0.700]

Rule 2: (88/30, lift 3.1)
  age <= 18
  failures > 0
```

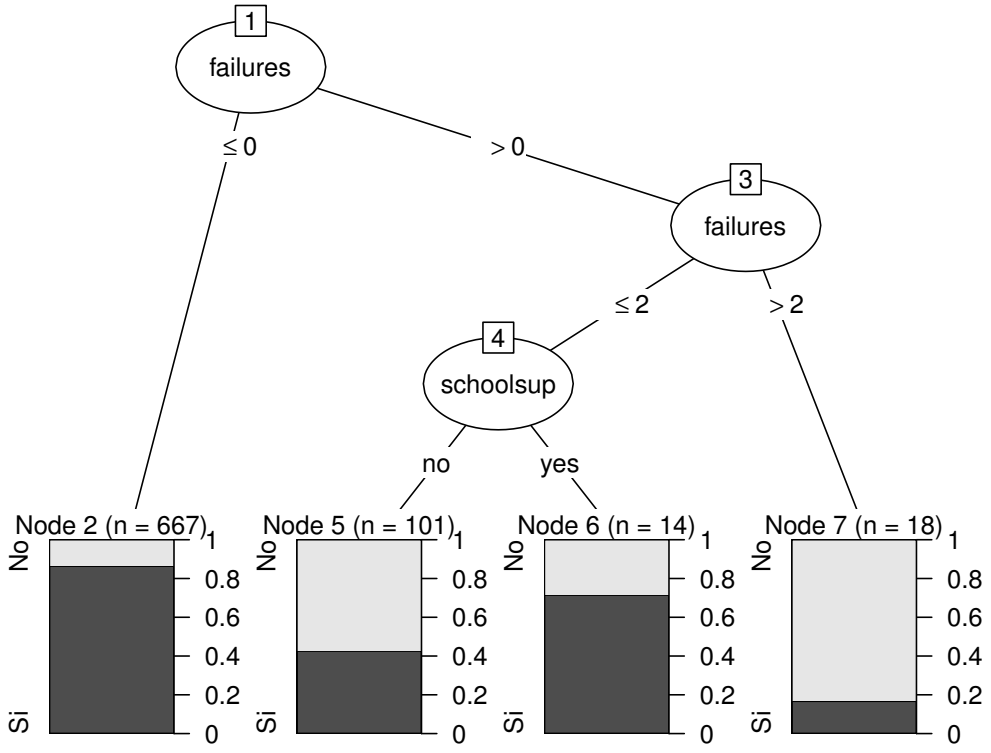
```
  schoolsup = no
  ->  class No  [0.656]

Rule 3: (133/56, lift 2.8)
  failures > 0
  ->  class No  [0.578]

Rule 4: (782/153, lift 1.0)
  failures <= 2
  ->  class Si  [0.804]

Default class: Si
```

R/C50P.txt

Podemos observar que el árbol podado generaba 4 reglas y el árbol no podado 5. Con la poda desaparecen las dos primeras reglas, fusionándose en una nueva.

Finalmente mostramos los arboles gráficos de cada modelo. Primero el podado:

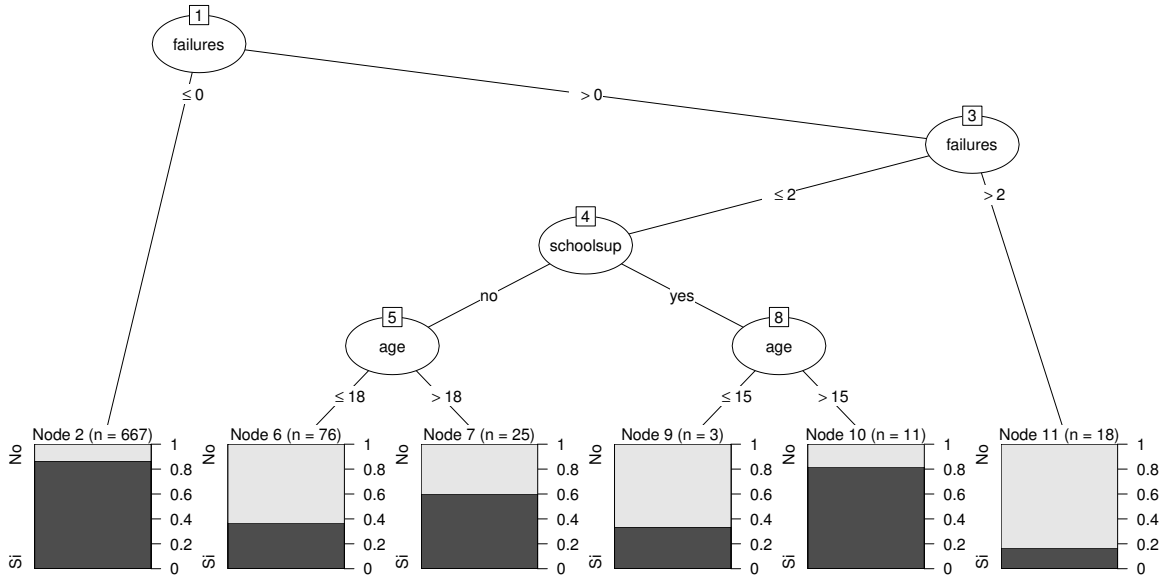
Figura 8.




.

El árbol sin podar queda así:

Figura 9.




.

Podemos comprobar que el árbol ha pasado de 4 hojas a 6, al no podar aquellas ramas que hacen referencia a la edad.

Que la poda sea la opción por defecto del algoritmo utilizado para construir el modelo es un indicador de su importancia, ya que para conjuntos grandes de datos y con muchas variables es muy fácil acabar construyendo árboles de decisión enormes muy difíciles de interpretar y con poca capacidad de generalización si no se podan.

### 4.6. Capacidad predictiva del modelo generado

Vamos a comprobar la capacidad predictiva del modelo que hemos generado. Recordemos que hemos creado el modelo con un conjunto de datos de entrenamiento y hemos reservado un subconjunto de datos de test que no hemos usado, es decir, el árbol de decisión no los ha utilizado en ningún cálculo. Ahora, con el modelo creado, vamos a dejar que el clasificador asigne cada observación del conjunto de test a su clase y que evalúe si lo ha hecho bien o mal:

```
1
2 # Calculo de la capacidad predictiva del modelo a partir del
      conjunto de test
3 p <-predict(model,testX,type="class")
```

Código 4.20: Fichero C50.R

```
> # Calculo de la capacidad predictiva del modelo a partir
    del conjunto de test
> p <-predict(model,testX,type="class")
> sum(p==testY)/length(p)
[1] 0.6680328
```

Código 4.21: Fichero C50.txt

En este caso, el valor de $p$ nos informa que los 244 elementos han sido clasificado correctamente en una proporción del 66,8 %, es decir, cometiendo un error muy superior al obtenido con el conjunto de entrenamiento. Esta situación es típica, especialmente si los conjuntos de entrenamiento y test no están bien balanceados.

Vamos a intentar corregir este problema aplicando una técnica de validación cruzada, K-fold cross validation. Dividiremos nuestros datos en $K = 10$ subconjuntos. Reservaremos un subconjunto de datos de test y construiremos el árbol con el resto de los datos de entreno $K - 1$. Valoraremos su capacidad predictiva contra el subconjunto de datos de test. Iteraremos tantas veces como $K$. Terminado el proceso calcularemos la media de error global de todos los árboles construidos.

```
library(C50)
library(plyr)

Alumnos_usos_sociales <-read.csv("/run/media/ramon/My
    Passport/BXXXXArbresDdecisio/student/Students_v20160908_
    2.csv",comment.char = "#")

X <- Alumnos_usos_sociales[,c("sex","age","studytime","
    failures","schoolsup","famsup","absences","Supera")]

form <- "Supera ~ sex + age + studytime + failures +
    schoolsup"
folds <- split(X, cut(sample(1:nrow(X)),10))
errs.c50 <- rep(NA, length(folds))
for (i in 1:length(folds)) {
  test <- ldply(folds[i], data.frame)
  train <- ldply(folds[-i], data.frame)
  tmp.model <- C5.0(as.formula(form), train)
  tmp.predict <- predict(tmp.model, newdata=test)
  conf.mat <- table(test$Supera, tmp.predict)
  errs.c50[i] <- 1 - sum(diag(conf.mat))/sum(conf.mat)
  }

print(sprintf("Media de error con k-fold cross validation:
    %.3f percent", 100*mean(errs.c50)))
```

Código 4.22: Fichero C50K.R

```
> print(sprintf("Media de error con k-fold cross validation:
     %.3f percent", 100*mean(errs.c50)))
[1] "Media de error con k-fold cross validation: 19.724
    percent"
```

Código 4.23: Fichero C50K.txt

Observando el error de clasificación global vemos que hemos corregido el problema producido por asignar los datos de entreno y test de forma arbitraria sin revisar el balan-

ceo correcto de la variables contenidas y en especial de la variable clasificadora. También podemos observar que el error global tiene valores similares al obtenido mediante el árbol generado originalmente.

La cuestión es qué árbol seleccionar, dado que hemos generado $K$ árboles diferentes y la distribución de errores puede ser muy diferente. Una opción es escoger el árbol que genere la matriz de confusión más similar a la matriz promedio, aunque esto puede ser complicado si el número de árboles construidos y en número de clases es elevado. La manera más sencilla para resolver este problema es utilizar mecanismos que combinen todos los árboles construidos en un solo clasificador, como por ejemplo la técnica llamada *boosting*.

### 4.7. Boosting

Una técnica interesante para mejorar la calidad del árbol obtenido son las técnicas de boosting. La idea es que podemos crear un proceso que itere la creación de modelos, utilizando la información extraída de un modelo en el siguiente. Así, cuando el segundo modelo se genera, ya se dispone de la información de los errores que ha cometido el primero (de hecho, todos los anteriores) y esto permite que el primer modelo (y anteriores) voten (en el sentido de aportar su decisión tomada) en el momento de asignación del valor de la observación, de acuerdo al error cometido. Esto hace que los árboles sean ligeramente diferentes y que, a medida que iteramos, las votaciones se amplían a los nuevos modelos, reduciendo el error global. Se trata de un procedimiento de combinación de clasificadores, pero en este caso usando clasificadores construidos iterativamente, compartiendo el modelo (árboles de decisión), el conjunto y variables de y los parámetros del modelo.

**Lectura complementaria**

Yoav Freund, and Robert E. Schapire. (1999). A short introduction to Boosting. *Journal of Japanese Society for Artificial Intelligence, 14(5):771-780.* `https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf`

Vamos a ver un ejemplo completo para comprender su uso. Primero generamos un árbol de acuerdo a la siguiente configuración:

```
# Carga del fichero.
Alumnos_usos_sociales <-read.csv("/run/media/ramon/My
    Passport/BXXXXArbresDdecisio/student/Students_v20160908_
    2.csv",comment.char = "#")
# Carga C5.0
library(C50)
set.seed(5813)
# Reordenar al azar los datos
Alumnos_usos_sociales <- Alumnos_usos_sociales[sample(nrow(
    Alumnos_usos_sociales)),]
X <- Alumnos_usos_sociales[,c(1:30,34)]
# Asignamos a Y si supera o no el curso
Y <- Alumnos_usos_sociales[,34]
# conjunto de entreno
trainX <- X[1:700,]
# conjunto de test
testX <- X[701:1044,]
#Revisamos % variable clasificatoria
prop.table(table(trainX$Supera))
prop.table(table(testX$Supera))
model <- C50::C5.0(trainX[,c(1:30)],trainX$Supera)
```

```
model
summary(model)
```

Código 4.24: Fichero C50B.R

De la ejecución destaquemos varios puntos:

```
> #Revisamos % variable clasificatoria
> prop.table(table(trainX$Supera))

       No        Si
0.2028571 0.7971429
> prop.table(table(testX$Supera))

      No       Si
0.255814 0.744186
```

Código 4.25: Fichero C50B.txt

Primero revisamos de la proporción de la variables clasificatoria en los conjuntos de entrenaniento y test. No siendo perfecta, es bastante similar, por lo que la consideramos aceptable.

Pasemos ahora a analizar la calidad del modelo construido:

```
Evaluation on training data (700 cases):

      Decision Tree
    ----------------
    Size      Errors

      24   70(10.0%)   <<


     (a)   (b)    <-classified as
    ----  ----
      81    61    (a): class No
       9   549    (b): class Si
```

Código 4.26: Fichero C50B.txt

Revisemos la calidad del modelo. De 700 casos incluidos en el conjunto de entrenamiento, 70 no clasifican bien, lo que significa una tasa de error del 10,0 %. La matriz de confusión nos explica que estos 70 casos se reparten en 9 Si clasificados como No y 61 No clasificados como Si, es decir, la clase No al ser minoritaria tiene tendencia a ser menospreciada por el árbol, favoreciendo la otra.

Ahora le presentaremos al modelo los datos del conjunto de test y revisaremos como los clasifica:

```
library(gmodels)
Supera_pred <- predict(model, testX)
```

```
CrossTable(testX$Supera,Supera_pred,prop.chisq=FALSE,prop.c=
    FALSE,prop.r=FALSE,dnn = c('actual Supera','predicted
    Supera'))
```

Código 4.27: Fichero C50B.R

```
Total Observations in Table:  344


                 predicted Supera
actual Supera          No          Si    Row Total
-----------------------------------------------------
           No          18          70           88
                    0.052       0.203
-----------------------------------------------------
           Si          18         238          256
                    0.052       0.692
-----------------------------------------------------
 Column Total          36         308          344
-----------------------------------------------------
```

Código 4.28: Fichero C50B.txt

De los 344 casos presentados, clasifica bien 18 de la clase No y 238 de la clase Si. Esto es una precisión del 74,4 %. Es decir, clasifica incorrectamente 70 de la clase No (la mayoría) y 18 de la clase Si, cometiendo un error del 25,6 %, muy superior al del conjunto de entrenamiento y especialmente sesgado en el caso de la clase No. Veamos si podemos mejorarlo con técnicas de boosting:

```
# miramos de mejorarlo con tecnicas de boosting
modelB <- C50::C5.0(trainX[,c(1:30)],trainX$Supera,trials
    =100)
modelB

Supera_pred <- predict(modelB, testX)
CrossTable(testX$Supera,Supera_pred,prop.chisq=FALSE,prop.c=
    FALSE,prop.r=FALSE,dnn = c('actual Supera','predicted
    Supera'))
```

Código 4.29: Fichero C50B.R

Construimos el modelo aplicando el parámetro *trials=10*. Este parámetro activa el boosting y marca el numero de arboles a construir. Si el algoritmo detecta que la calidad de los nuevos modelos creados es baja se detendrá antes de llegar al número marcado de iteraciones. El siguiente paso es contrastar con los datos del conjunto de test la calidad de este nuevo modelo creado:

```
Total Observations in Table:  344


                 predicted Supera
actual Supera          No          Si    Row Total
-----------------------------------------------------
           No          26          62           88
                    0.076       0.180
-----------------------------------------------------
```

```
           Si           25          231          256
                     0.073        0.672
-------------------------------------------------
 Column Total           51          293          344
-------------------------------------------------
```

Código 4.30: Fichero C50B.txt

De los 344 casos presentados, clasifica bien 26 de la clase No y 231 de la clase Si, es decir, clasifica incorrectamente 62 elementos de la clase No y 25 de la clase Si, cometiendo un error del 25,3 %, por lo que se obtiene una muy leve mejora. Forcemos entonces las iteraciones a 100:

```
Total Observations in Table:  344


                predicted Supera
actual Supera           No           Si    Row Total
-------------------------------------------------
           No           26           62           88
                     0.076        0.180
-------------------------------------------------
           Si           28          228          256
                     0.081        0.663
-------------------------------------------------
 Column Total           54          290          344
-------------------------------------------------
```

Código 4.31: Fichero C50B.txt

Si analizamos estos datos vemos que ya no estamos mejorando. Podría ser que estuviéramos sobreentrenando el modelo o que simplemente el conjunto de datos a estudiar no aporte más valor. Para poder continuar avanzando deberíamos revisar la población de datos que estamos usando y valor la inclusión de variables que pudieran haberse descartado, así como una mejor partición del conjunto original respetando su distribución original.

Hemos de tener presente que cuando generamos un modelo con las técnicas de boosting se están creando tantos árboles como iteraciones se realizan. Con la sentencia *summary(modelB)* podemos observar todos los arboles creados. El modelo final, por lo tanto, es mucho más complejo y complicado de interpretar, llegando a poder ser considerado un método de caja negra, difícil de comprender pero con una elevada fiabilidad.

## 5. Conclusiones

En general, los árboles de decisión son un modelo de minería de datos que permite construir buenos clasificadores, siendo sus puntos fuertes la facilidad y claridad de extracción de la información contenida en su estructura interna y las reglas asociadas. Esta facilidad en extraer reglas permite su implementación en sistemas ayuda a la toma de decisiones. Permiten averiguar cuales son las variables con mayor peso en el proceso de clasificación y descartar aquellas que no intervienen en el proceso. Por contra, hemos de vigilar que la aparente alta fiabilidad de un modelo no sea un problema de sobreentrenamiento. Pueden ser también demasiado sensibles a las variables mayoritariamente representadas en el conjunto de entrenamiento. Por ello se han desarrollado diversas técnicas que reflejan diferentes aproximaciones al problema de encontrar las variables con mayor capacidad predictiva, cuándo crear una rama nueva o cuándo se debe podar un árbol, así como la generación de diferentes árboles que pueden ser combinados para mejorar la capacidad predictiva.

En un escenario real de uso, siempre hemos de tener presente cruzar y analizar en detalle los modelos generados, recrearlos cada cierto tiempo y tener una visión crítica que intente explicar los resultados obtenidos, huyendo de los modelos de caja negra.

## Bibliografía

**Johannes Ledolter** (2013). *Data Mining and Business Analytics with R*. Wiley, ISBN: 978-1-118-44714-7.

**Yu-Wei, Chiu (David Chiu)** (2015). *Machine Learning with R Cookbook*. Packt, 978-1-783-98204-2.

**Pawel Cichosz** (2015). *Data Mining Algorithms: Explained Using R*. Wiley, 978-1-118-33258-0.