Técnicas de Inteligencia Artificial

Tema 3. Árboles de decisión

Índice

Esquema

Ideas clave

3.1. ¿Cómo estudiar este tema?

3.2. Introducción. Representación del conocimiento mediante árboles de decisión

3.3. Descripción de la tarea de inducción

3.4. Algoritmo básico de aprendizaje de árboles de decisión: ID3

3.5. Espacio de búsqueda y bias inductivo

3.6. Métodos de selección de atributos

3.7. Sobreajuste y poda de árboles

3.8. Medidas de la precisión de la clasificación. Curva ROC

3.9. Simplificación de árboles de decisión mediante poda: algoritmo C4.5

3.10. Ensemble Learning y Random Forest

3.11. Aplicaciones y ejemplos de implementación

3.12. Referencias bibliográficas

A fondo

Introducción a la herramienta Weka

Inducción de árboles de decisión mediante el algoritmo ID3

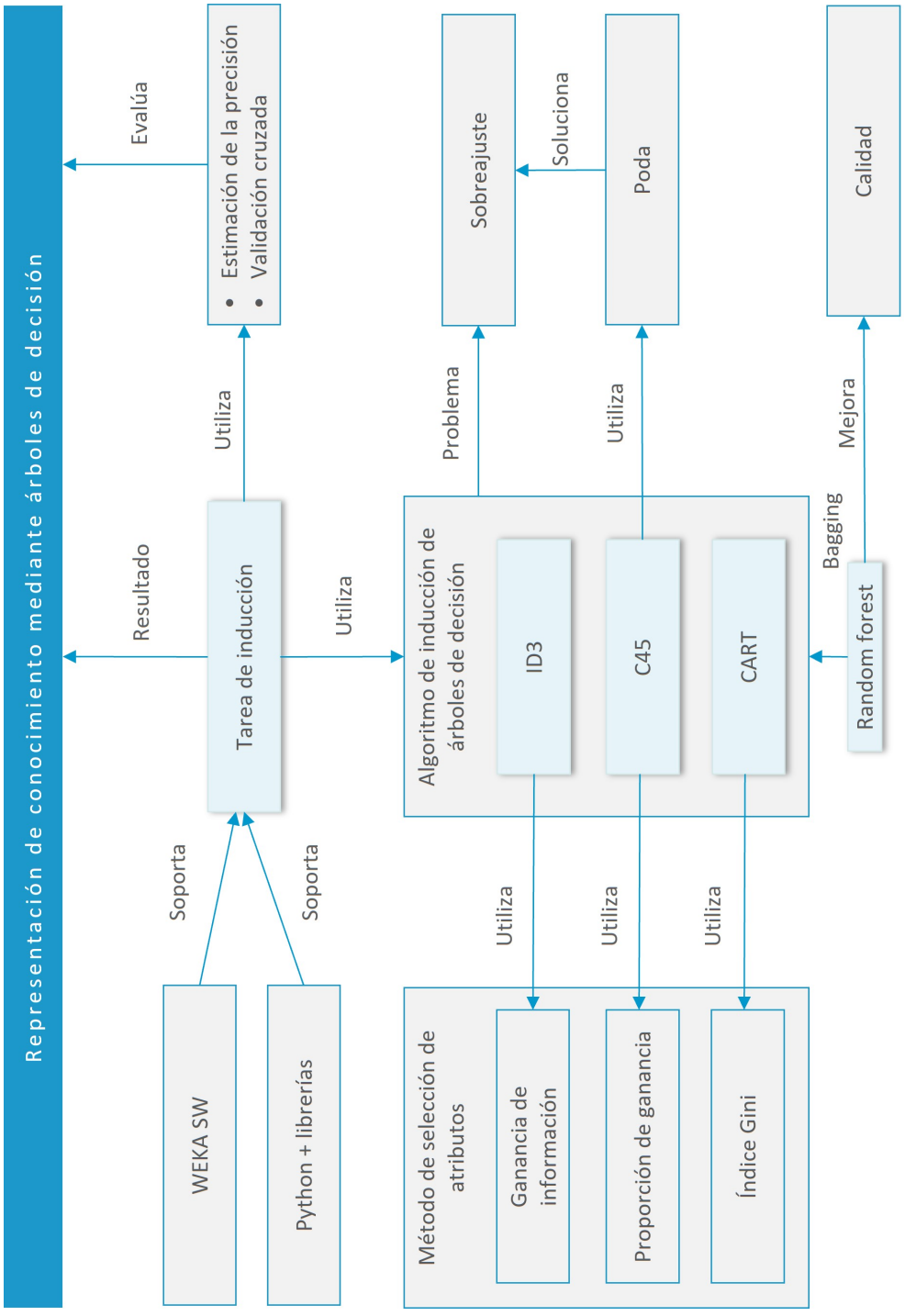
Software de minería de datos Weka

Weka 3: Software de minería de datos en Java

Wiki de documentación sobre Weka

Test

Esquema



3.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral. Revisando referencias y bibliografía, así como accediendo a los recursos adicionales que se facilitan.

Al finalizar el estudio de este tema el alumno será capaz de:

- Implementar árboles de decisiones para representación de conocimiento.
- Aplicar los algoritmos ID3, C4.5 y Random Forest para resolver problemas de aprendizaje.
- Describir el problema de sobreajuste.
- Resolver un problema de sobreajuste aplicando poda.
- Identificar aplicaciones prácticas de la técnica de árboles de decisión.
- Aplicar algoritmos de árboles de decisión utilizando librerías bajo Python.
- Utilizar las funciones básicas de Weka.

3.2. Introducción. Representación del conocimiento mediante árboles de decisión

El aprendizaje de árboles de decisión es una de las técnicas más utilizadas para el aprendizaje inductivo, en concreto como técnica de aprendizaje supervisado, el cual es bastante robusto frente a datos ruidosos. Las entradas y salidas de la función objetivo suelen ser valores discretos, aunque en el caso de las entradas también podrían ser valores continuos. La representación de esta función objetivo toma forma de árbol y es interpretada como una serie de condiciones consecutivas que puede ser fácilmente mapeada a reglas.

Mediante un árbol de decisión será posible clasificar instancias cuya clase sea desconocida. Para llevar a cabo esta tarea de clasificación no habrá mas que comparar los atributos de dicha instancia con las ramas del árbol de decisión, de forma que se recorra de raíz a la hoja correspondiente el camino que no vaya marcando el valor de los atributos.

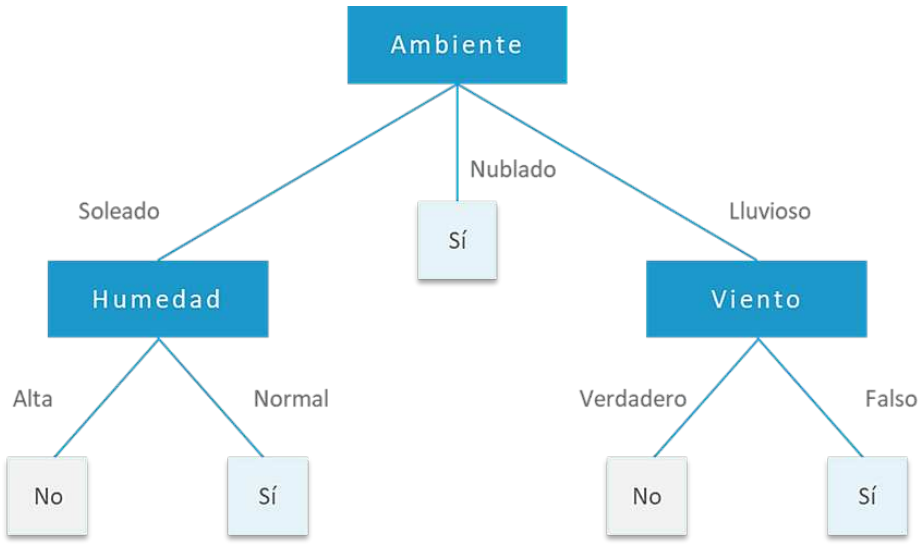


Figura 1. Ejemplo de árbol de decisión para el aprendizaje del concepto «Jugar al aire libre».

La Figura 1 muestra un ejemplo clásico de árbol de decisión. Este problema lo utiliza J.R. Quinlan para ilustrar el algoritmo *ID3* (Quinlan, 1986), que se explica en un apartado posterior, siendo un ejemplo que se puede encontrar adaptado en distintos libros que explican la técnica de árboles de decisión.

Con este árbol se pretende clasificar instancias con atributos relativos a diversos factores del tiempo ambiental con el fin de decidir si se juega al aire libre o no. Por ejemplo, si un sábado amanece lluvioso y con viento (condiciones representadas en las ramas de la derecha) el árbol indica que no es un día adecuado para jugar al aire libre.

¿Cuándo se considera adecuado el aprendizaje mediante árboles de decisión?

En general se siguen los siguientes criterios para responder a esta cuestión (Mitchell, 1997):

- Las instancias se representan por un conjunto de atributos y sus valores (en el ejemplo anterior, el atributo humedad puede tener el valor alta o normal). Estos valores pueden ser nominales, pero también se pueden resolver problemas con atributos de valores numéricos, mediante la aplicación de los algoritmos adecuados.
- La función objetivo tiene valores de salida discretos. En el ejemplo se asigna el valor sí o no.
- Se requieren descripciones disyuntivas. Los árboles de decisión son adecuados para la representación de este tipo de disyunciones. Cada rama desde la raíz a la hoja representa una conjunción lógica (operador AND), mientras que el árbol completo es una disyunción de conjunciones (operador OR). Por ejemplo, la clase sí del ejemplo de la Figura 1 queda representada por tres ramas del árbol que se pueden mapear a la siguiente regla, que es una disyunción de conjunciones que restringen los valores de los atributos de una instancia que pertenece a la clase sí:

```
SI ambiente es soleado AND humedad es normal

OR ambiente es lluvioso AND viento es falso

OR ambiente es nublado

    ENTONCES jugar = sí
```

- Los datos de entrenamiento contienen errores o valores de atributos desconocidos. Los árboles de decisión son robustos frente a errores tanto en la asignación de la clase de una instancia o clasificación de un ejemplo, como frente a si existen valores de los atributos de un ejemplo desconocidos o con errores.

Los árboles de decisión presentan entre otras las siguientes ventajas:

- Son fáciles de comprender y traducir a reglas.
- Pueden trabajar con conjuntos de datos tanto numéricos como nominales.
- Pueden trabajar con datos multidimensionales.
- No requieren conocimiento en un dominio dado ni establecer parámetros.

Sin embargo, existen algunas restricciones:

- Los atributos de salida deben ser categorías.
- No se permiten múltiples atributos de salida. No obstante, se puede emplear un árbol de decisión para cada atributo de salida.
- Los árboles construidos a partir de datos numéricos pueden resultar muy complejos

Los árboles de decisión han sido aplicados con éxito a problemas del mundo real, resultando **muy adecuados para resolver problemas de clasificación**, esto es, cuando la tarea consiste en clasificar ejemplos dentro de un conjunto discreto de posibles categorías.

Por tanto, se pueden utilizar, por ejemplo, para el diagnóstico de enfermedades, el diagnóstico de fallos de sistemas, la clasificación de clientes con el fin de aplicar estrategias de *marketing* o de detectar si hay riesgo en conceder un préstamo.

3.3. Descripción de la tarea de inducción

La definición de inducción nos indica que dicha tarea consiste en extraer el conocimiento general implícito a partir de observaciones y experiencias particulares. En el aprendizaje de árboles de decisión, el espacio de hipótesis es el conjunto de todos los árboles de decisión posibles. La tarea de inducción del árbol de decisión consiste en encontrar el árbol que mejor encaje con los datos de ejemplo, ya clasificados, disponibles. Para cada clase, se ha de encontrar una rama en el árbol que satisfaga la conjunción de valores de atributos representada por la rama.

Cuando se está generando un árbol de decisión un elemento crucial es el **método de selección de atributos**, que determina qué criterio se utiliza para generar las diferentes ramas del árbol, que van determinando la clasificación en las diferentes clases.

El criterio dependerá del tipo de dato que sea el atributo. Por ejemplo, si el tipo de dato del atributo es discreto, se crea una rama para cada valor conocido del atributo. Sin embargo, si el tipo de dato es numérico, hay que establecer algún punto de separación en las ramas como, por ejemplo, si el valor del atributo es mayor o menor a un determinado valor (ver ejemplos en la Figura 2).

Figura 2. Creación de ramas del árbol en función del tipo de atributo.

Para describir el aprendizaje de árboles de decisión se continúa con el popular problema sobre el tiempo. El conjunto de datos de entrenamiento se muestra en la Tabla 1. El ejemplo tiene 14 instancias de entrada con cuatro atributos de entrada y

un atributo de salida (el concepto a aprender). En el artículo de Quinlan simplemente se indican dos valores P y N, siglas de positivo y negativo, para el atributo de salida (Quinlan, 1986).

En el ejemplo que se presenta se utiliza una versión adaptada presente en varias referencias, en los que el problema consiste en clasificar las mañanas de los sábados, en función de tres factores ambientales, según sean adecuadas o no para jugar al aire libre (por ejemplo, un partido de tenis). Se mantienen los atributos de entrada presentes en el artículo de Quinlan con los siguientes valores (Quinlan, 1986).

- Ambiente: soleado, nublado, lluvioso.
- Temperatura: alta, media, baja.
- Humedad: alta, normal.
- Viento: verdadero, falso.

El atributo de salida, la clase, puede tomar también dos valores que se utilizan para clasificar las mañanas de los sábados de acuerdo a si se juega o no al aire libre:

- Clase: sí, no.

Id	Ambiente	Temperatura	Humedad	Viento	Clase
E_1	Soleado	Alta	Alta	Falso	No
E_2	Soleado	Alta	Alta	Verdadero	No
E_3	Nublado	Alta	Alta	Falso	Sí
E_4	Lluvioso	Media	Alta	Falso	Sí
E_5	Lluvioso	Baja	Normal	Falso	Sí
E_6	Lluvioso	Baja	Normal	Verdadero	No
E_7	Nublado	Baja	Normal	Verdadero	Sí
E_8	Soleado	Media	Alta	Falso	No
E_9	Soleado	Baja	Normal	Falso	Sí
E_{10}	Lluvioso	Media	Normal	Falso	Sí
E_{11}	Soleado	Media	Normal	Verdadero	Sí
E_{12}	Nublado	Media	Alta	Verdadero	Sí
E_{13}	Nublado	Alta	Normal	Falso	Sí
E_{14}	Lluvioso	Media	Alta	Verdadero	no

Tabla 1. Instancias con sus valores de atributos y clases del problema «Jugar al aire libre».

Un algoritmo básico para construir un árbol de decisión es sencillo.

En la Figura 3 se muestra este algoritmo para el caso en que los atributos de los datos de entrenamiento son nominales. Los parámetros para generar el árbol de decisión son los siguientes:

- Los datos de entrenamiento E, las instancias ejemplo clasificadas en clases.
- La lista de atributos de las instancias que van a ser candidatas para determinar la decisión.
- El método de selección de los atributos. Especifica una heurística para seleccionar el atributo que mejor discrimina los ejemplos para una clase.

```
        PROCEDIMIENTO Inducir_Arbol (Ejemplos E, Lista_Atributos,
        Método_Selección_Atributos)
        COMIENZO
P1      Crear un nodo N;
P2      SI todos los elementos de E pertenecen a la misma clase, C
        ENTONCES retornar N como nodo hoja etiquetado con la clase C,
P3      SINO SI la lista de atributos (Lista_Atributos) está vacía
        ENTONCES retornar N como nodo hoja etiquetado con la clase más numerosa
        en los ejemplos
P4      SINO aplicar Método_Selección_Atributos(E, Lista_Atributos) para
        seleccionar el atributo A que mejor particiona E
P5                          Borrar Atributo A de la lista de Atributos
P6      Lista_Atributos
P7      Etiquetar N con el atributo seleccionado
                            PARA CADA valor V de A
        Siendo Ev el subconjunto de elementos en E con valor V en el atributo A.
P8      SI Ev está vacío
        ENTONCES unir al nodo N una hoja etiquetada con la clase mayoritaria en
        E.
P9      SINO unir al nodo N el nodo retornado de Inducir_Arbol (Ev,
        Lista_Atributos, Método_Selección_Atributos)
                            FIN PARA CADA
        FIN SI-SINO
        FIN
```

Figura 3. Algoritmo básico de construcción de árboles de decisión.

En primer lugar, se crea un nodo raíz que representa a todos los ejemplos en E. Si todas las instancias en E son de la misma clase, el nodo se convierte en una hoja del árbol que es etiquetada con esa clase (P2). Si esto no es así y la lista de atributos no está vacía, se llama al procedimiento Método_Selección_Atributos (paso P4) para determinar el atributo que se ha de comprobar en el nodo N, que será el atributo que implica la mejor división de los ejemplos en clases. Crecerán tantas ramas del nodo N como posibles valores del atributo. Se trata de que los subconjuntos de ejemplos resultantes de la división sean lo más homogéneos posibles (la homogeneidad máxima se da cuando todos los ejemplos de cada subconjunto pertenecen a la misma clase).

En el paso P6 se etiqueta el nodo N con el atributo A seleccionado que servirá como condición para la clasificación. El nodo se ramifica con los diversos valores del atributo y se distribuyen los ejemplos en las diversas ramas de acuerdo al valor del atributo que cumplen.

En la Figura 4 se muestra un ejemplo del resultado de la primera iteración del algoritmo sobre los datos del problema del tiempo mostrados en la Tabla 1. El método de selección del atributo ha seleccionado ambiente como el mejor atributo para dividir los ejemplos teniendo en cuenta específicamente la ganancia de información que se consigue dividiendo los ejemplos con este atributo (en la siguiente sección se explicará este método de selección de atributos que utiliza el algoritmo ID3 basado en la medida de ganancia de información).

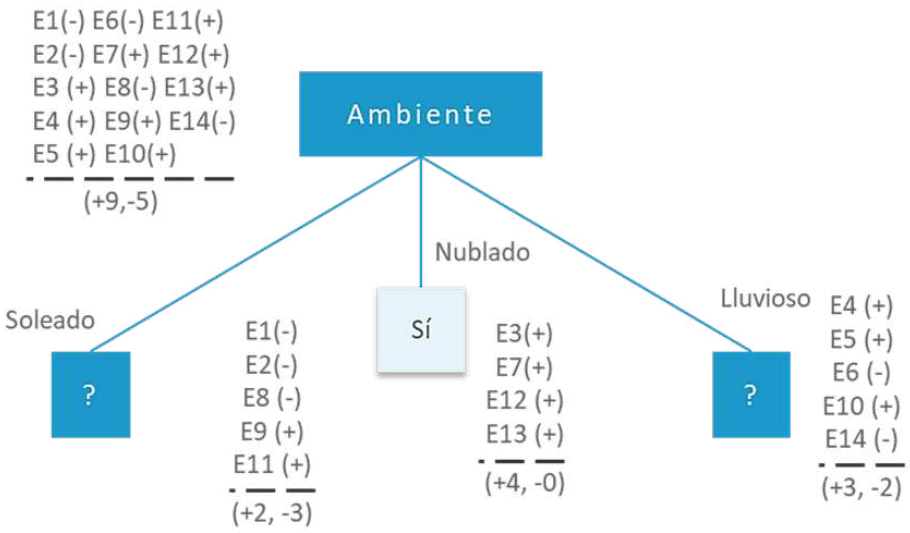


Figura 4. Árbol parcial generado en la primera iteración del algoritmo sobre los datos de ejemplo de la Tabla 1.

Una vez particionados los ejemplos, se vuelve a iniciar el procedimiento en cada nodo hijo, partiendo del subconjunto de ejemplos asignados a ese nodo hijo Ev y de la lista de atributos no utilizados previamente para particionar.

En el ejemplo concreto, para el nodo hijo generado que se une al nodo raíz con la rama nublado, se va a cumplir la condición del paso 2 (P2), es decir, todos los ejemplos en esa partición pertenecen a la clase sí , con lo cual ese nodo se convierte en un nodo hoja etiquetado con la clase sí .

Para los otros dos nodos hijos generados es necesario seleccionar de nuevo un atributo en base a los nuevos subconjuntos generados, en ambos casos compuestos por cinco ejemplos. Este proceso es iterativo hasta que se cumple una de las siguientes condiciones:

- Todos los ejemplos de Ev pertenecen a la misma clase, con lo cual el nodo se convierte en un nodo hoja (P2).
- No hay más atributos para particionar ejemplos (P3). En este caso se puede convertir el nodo en una hoja y etiquetarlo con la clase mayoritaria en los ejemplos de Ev . Esta decisión se denomina **votación mayoritaria**.
- Si no hay ejemplos para una rama, la partición Ev está vacía (P8), de modo que se crea un nodo hoja con la clase mayoritaria en E . Existen variaciones del algoritmo que toman otras decisiones como simplemente etiquetar el nodo hoja como desconocido .

Ideas clave

El algoritmo básico aquí descrito es del tipo «**divide-y-vencerás**», construido sin retroceder en ningún caso para volver a reconsiderar una decisión tomada en un paso previo. Esto último relativo a siempre avanzar hacia adelante es denominado **método codicioso** (***greedy*** en inglés).

En cada iteración de este algoritmo básico se aplica el método de selección de atributos en base a los atributos y ejemplos, y esto puede suponer un alto requisito de computación.

3.4. Algoritmo básico de aprendizaje de árboles de decisión: ID3

El algoritmo ID3 construye lo árboles *top-down* (de arriba a abajo) utilizando un método de selección de atributos basado en la **teoría de la información**. ID3 considera como heurística que el atributo cuyo conocimiento aporta mayor información en la clasificación es el más útil.

El algoritmo ID3 sigue los pasos del algoritmo básico mostrado en la Figura 4. En primer lugar, el algoritmo, para cada atributo, realiza un cálculo para medir cuán bien ese atributo clasifica los ejemplos disponibles. El atributo mejor clasificador se convierte en la condición del nodo raíz que da lugar a distintas ramas, una por cada valor posible del atributo.

Los ejemplos se distribuyen en los nodos descendientes de acuerdo con su valor del atributo mejor clasificador. Este proceso que se ha aplicado al nodo raíz se repite para los nodos descendientes. Este algoritmo nunca da marcha atrás para reconsiderar decisiones previas.

Como previamente se ha comentado un aspecto importante es el método de selección de atributos, que determina el rendimiento del algoritmo. El algoritmo ID3 utiliza la ganancia de información para seleccionar en cada paso según se va generando el árbol aquel atributo que mejor distribuye los ejemplos de acuerdo a su clasificación objetivo.

¿Cómo se mide la mejor distribución de ejemplos o se selecciona aquel atributo cuyo conocimiento aporta mayor cantidad de información?

Para contestar a esta pregunta ID3 utiliza conceptos que forman parte de la teoría de la información. En concreto, como previamente se ha indicado, utiliza la ganancia de

información que a su vez mide la reducción esperada de entropía. Para comprender esto se va a explicar a continuación el concepto de entropía y, posteriormente, la medida de ganancia de información.

La entropía caracteriza la heterogeneidad de un conjunto de ejemplos. Cuando una clase C puede tomar *n* valores, la entropía del conjunto de ejemplos E respecto a la clase C se define como:

$$Entropía\,(E) = \sum_{i=1}^{n} -p_i \log_2 p_i \,(1)$$

Siendo pi la proporción de ejemplos de E que pertenecen a la clase i.

Si todos los miembros del conjunto *E* pertenecen a la misma clase, la entropía es nula, es decir, tendrá un valor igual a 0. Sin embargo, esta será igual a 1 si se tiene un mismo número de ejemplos positivos y negativos (cuando se trata de una clasificación booleana), y tendrá un valor comprendido entre 0 y 1 cuando no es igual el número de ejemplos positivos y negativos.

Dado que la entropía mide la heterogeneidad de un conjunto de ejemplos, la ganancia de información utiliza la entropía para medir la efectividad de un atributo para clasificar ejemplos. Específicamente mide la reducción de entropía cuando se distribuyen los ejemplos de acuerdo a un atributo concreto.

Siendo un atributo *A* con *Va* posibles valores y un conjunto de ejemplos *E*, la fórmula para calcular la ganancia de información viene dada por la expresión (2):

$$Ganancia\,(E,A) = Entropía\,(E) - \sum_{v \in Va} \frac{|E_v|}{E} Entropía\,(E_v)\,(2)$$

Ev es el subconjunto de ejemplos para los que el atributo *A* toma el valor *v* dentro de los posibles valores de *v* (especificados en *Va*).

Por tanto, en el ejemplo previo del artículo de Quinlan, si se utiliza la ganancia de información para seleccionar los atributos que particionan en cada paso el árbol, para el caso del nodo raíz se tiene:

$$Entropía\,(E) = \frac{-9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = +0{,}94$$

$$\text{Ganancia (E, ambiente)} = \text{Entropía(E)} - \frac{5}{14}\text{Entropia}(E_{soleado}) - \frac{4}{14}\text{Entropia}(E_{nublado}) - \frac{5}{14}\text{Entropia}(E_{lluvioso}) = +0{,}9402 - \frac{5}{14}0{,}9709 - \frac{4}{14}0 - \frac{5}{14}0{,}9709 = 0{,}247$$

$$\text{Ganancia (E, humedad)} = \text{Entropía(E)} - \frac{7}{14}\text{Entropia}(E_{alta}) - \frac{7}{14}\text{Entropia}(E_{normal}) = +0{,}9402 - \frac{7}{14}0{,}985 - \frac{7}{14}0{,}591 = 0{,}151$$

$$\text{Ganancia (E, viento)} = \text{Entropía(E)} - \frac{8}{14}\text{Entropia}(E_{falso}) - \frac{6}{14}\text{Entropia}(E_{verdadero}) = +0{,}9402 - \frac{8}{14}0{,}811 - \frac{6}{14}1 = 0{,}048$$

$$\text{Ganancia (E, temperatura)} = \text{Entropía(E)} - \frac{4}{14}\text{Entropia}(E_{alta}) - \frac{6}{14}\text{Entropia}(E_{media}) - \frac{4}{14}\text{Entropia}(E_{baja}) = +0{,}9402 - \frac{4}{14}1 - \frac{6}{14}0{,}918 - \frac{4}{14}0{,}811 = 0{,}029$$

De acuerdo con la medida de ganancia de información que aplica el algoritmo ID3 el atributo ambiente es el que proporciona mayor cantidad de información a la hora de predecir la clase. Por lo tanto, con este método de selección de atributos el ambiente es seleccionado como el atributo a comprobar en el nodo raíz y tres ramas se crean, correspondientes a los tres valores que este atributo toma.

Los ejemplos se dividen en tres grupos según su valor de este atributo y se llega por tanto al árbol parcial mostrado en la Figura 4. Dado que todos los ejemplos con ambiente igual a nublado pertenecen a la clase sí (entropía es 0), el nodo unido a través de la rama nublado se convierte en un nodo hoja.

Partiendo de los nodos hijos el procedimiento se repite hasta que o todos los ejemplos pertenecen a la misma clase, o todos los atributos están incluidos en un camino del árbol, o no quedan más ejemplos.

3.5. Espacio de búsqueda y bias inductivo

En el problema de construcción de árboles de decisión, el espacio de hipótesis o posibles soluciones es el conjunto de todos los posibles árboles de decisión. En concreto, el algoritmo ID3 busca por el espacio de hipótesis hasta encontrar el árbol que encaja con los ejemplos de entrenamiento.

ID3 realiza una búsqueda en escalada, guiada por la medida de ganancia de información, desde árboles más sencillos a árboles más complejos, buscando aquel que clasifica correctamente los datos de entrenamiento.

ID3 tiene la ventaja de que **trabaja en un espacio de hipótesis completo**, con lo cual no se da el caso de estar buscando en un espacio de hipótesis en que no se encuentra la solución. Sin embargo, ID3 no trabaja con varias posibles soluciones simultáneamente, aquellas que son consistentes con los ejemplos, sino que solo trabaja con una posible solución. En este caso se **corre el riesgo de no construir el árbol que representa la mejor solución.**

ID3 no da marcha atrás en su búsqueda para reconsiderar elecciones previas. Esto puede dar lugar al problema típico de métodos de búsqueda de escalada sin vuelta atrás**, el poder converger hacia una solución óptima local, que no es la mejor solución en global.**

ID3 tiene la ventaja de ser **robusto frente a errores**, dado que en cada paso que se da utiliza todos los ejemplos para hacer los cálculos de la ganancia de información. Sin embargo, esto supone **más carga computacional** que una solución incremental en la que en cada paso solo se tienen en cuenta parte de los ejemplos.

Ideas clave

En esta búsqueda por el espacio de hipótesis, **¿en qué se basa ID3 para generalizar el árbol de decisión, esto es, considerar que el árbol clasificará correctamente instancias no utilizadas en la etapa de aprendizaje? En definitiva, entendemos el *bias* como los criterios de selección de una hipótesis en función de una serie de factores y supuestos.**

De los posibles árboles que podrían servir para clasificar un conjunto de ejemplos, ID3 escoge como mejor árbol al primero que encuentra en una búsqueda en la que se va generando un árbol desde lo más sencillo a lo más complejo.

ID3, por tanto, funciona siguiendo estas dos premisas:

- Da preferencia a árboles cortos frente a los largos.
- Sitúa los atributos que proporcionan mayor ganancia de información más cerca de la raíz.

Con ID3 no se garantiza encontrar el árbol más corto, puesto que no considera en cada paso todos los árboles posibles que se valorarían en una búsqueda exhaustiva en amplitud. Sin embargo, ID3 realiza una búsqueda *greedy* sin retroceso, que pretende encontrar el árbol más corto que a su vez sitúa los atributos que mayor ganancia de información aportan en la zona del árbol más cerca de la raíz.

El *bias* inductivo de ID3 se puede, por tanto, considerar como una preferencia por árboles cortos frente a largos y se prefieren árboles que sitúan cerca de la raíz a los atributos que aportan mayor ganancia de información.

Ideas clave

El preferir árboles cortos puede mejorar la **generalización**, puesto que hay hipótesis complejas que encajan muy bien con los datos de entrenamiento pero que no generalizan correctamente datos futuros.

3.6. Métodos de selección de atributos

Existen diversos métodos de selección de atributos empleados en diversos algoritmos de construcción de árboles de decisión como la **tasa de ganancia**, el **índice Gini** o la previamente explicada **ganancia de información** utilizada por el algoritmo ID3. El usar un método u otro dependerá tanto de los datos de entrenamiento disponibles, como del algoritmo específico que se emplee así como de los supuestos y suposiciones que se realizan para generalizar la mejor solución encontrada (*bias*).

Por ejemplo, el **índice Gini**, que **mide la impureza de los datos**, es típicamente utilizado en algoritmos CART (Classification and Regression Trees).

El índice Gini se calcula con la expresión (3):

$$Gini(E) = 1 - \sum_{i=1}^{n} p_i^2 \qquad (3)$$

Siendo *n* el número de clases totales y siendo *pi* el resultado de dividir el número de ejemplos que pertenecen a la clase *Ci* entre el número de ejemplos totales. El sumatorio se refiere a la suma que comprende dicho cociente para las *n* clases.

El atributo seleccionado para dividir los ejemplos en subconjuntos será aquel que maximice la reducción de impureza (el que tenga un índice *Gini* menor).

Otro método de selección de atributos se basa en la **proporción de ganancia**, utilizada por el algoritmo C4.5, que se explicará más adelante. Cuando se manejan atributos con muchos valores, la proporción de ganancia puede dar mejores resultados que la medida de ganancia de información, que favorece aquellos atributos con mayor número de valores.

Si, por ejemplo, se da el caso extremo de un atributo con un valor diferente para cada ejemplo, la medida de ganancia de información determinará que este atributo

sea escogido puesto que aporta la mayor información en la clasificación, ya que determina la clase sin ninguna ambigüedad. Sin embargo, realmente no es un clasificador útil para nuevas instancias.

La proporción de ganancia compensa el hecho de que un atributo pueda tener muchos valores dividiendo por la medida denominada información de la división, tal y como se indica en la fórmula (4).

$$Información\,de\,la\,División\,(E,A) = -\sum_{i=v_i}^{v_n} Ei \vee \frac{|Ei|}{E\vee} \log_2 \frac{|E_i|}{|E|} \qquad (4)$$

Siendo *Ei, Ei+1, …, En* las diferentes particiones de ejemplos que resultan de dividir el conjunto *E* de ejemplos teniendo en cuenta los valores *vi, vi+1, …, vn* que toma el atributo, respectivamente.

Así la proporción de ganancia se calcula como el cociente entre la ganancia de información y la información de la división, tal y como se muestra en la expresión (5).

$$Proporción\,de\,Ganancia\,(E,A) = \frac{Ganancia\,de\,Información(E,A)}{Información\,de\,la\,División(E,A)} \qquad (5)$$

Dividiendo por la información de la división se consigue penalizar aquellos atributos con muchos valores que se distribuyen muy uniformemente entre los ejemplos. Si un atributo toma dos valores y distribuye los ejemplos en dos grupos de igual tamaño, la *Información de la División* toma el valor 1.

Sin embargo, si un atributo toma un gran número de valores y distribuye los ejemplos uniformemente en estos valores se obtiene un valor de Información de la División de *log2n*.

Otro método es denominado **Longitud de Descripción Mínima** o Minimum Description Length (MDL). Este método considera el mejor árbol de decisión aquel que requiere un menor número de bits para o bien codificar el propio árbol o bien codificar los casos no clasificados por el árbol, seleccionando la opción más simple de estas.

Otros métodos consideran mejor particionar el árbol basándose en múltiples atributos y no solo en uno.

Realmente los métodos de selección indicados funcionan bien. Utilizan diferentes supuestos y suposiciones sobre la mejor solución encontrada (*bias*) pero no está claro qué método es mejor que otro. En general la complejidad de árbol aumentará conforme la profundidad sea mayor, pero, por otra parte, árboles más cortos pueden generar más errores en las decisiones.

3.7. Sobreajuste y poda de árboles

ID3 crea el árbol de decisión de tal manera que clasifica correctamente los datos de entrenamiento y, aunque parece contradictorio, clasificar demasiado bien no es siempre adecuado ya que los datos de entrenamiento pueden contener ruido o simplemente pueden no ser una muestra suficientemente numerosa de datos que permita generalizar.

El sobreajuste se produce cuando existe una hipótesis H del espacio de hipótesis que se ajusta mejor a los datos de entrenamiento que otra hipótesis H' del espacio de hipótesis, pero, al mismo tiempo, H' se ajusta mejor a todas las instancias (comprendiendo datos de entrenamiento e instancias futuras).

Construir hipótesis con demasiadas condiciones, demasiado complejas, reflejadas en árboles con demasiados nodos, a veces supone sobreajustar la hipótesis a los datos de entrenamiento.

Utilizando el ejemplo previo del problema del tiempo, si el Ejemplo 7 (E7) mostrado en la Tabla 1 tiene asignada la clase `No` erróneamente, esto daría lugar a la generación de un árbol distinto y más complejo, ya que, en el segundo paso del algoritmo no se generaría el nodo hoja, tal y como se muestra en la Figura 4 (todos los ejemplos con el atributo `nublado` pertenecían a la misma clase), sino que se seguiría ramificando, dando lugar a un árbol más complejo. Queda como ejercicio del alumno el ejecutar el algoritmo para comprobar qué árbol se generaría.

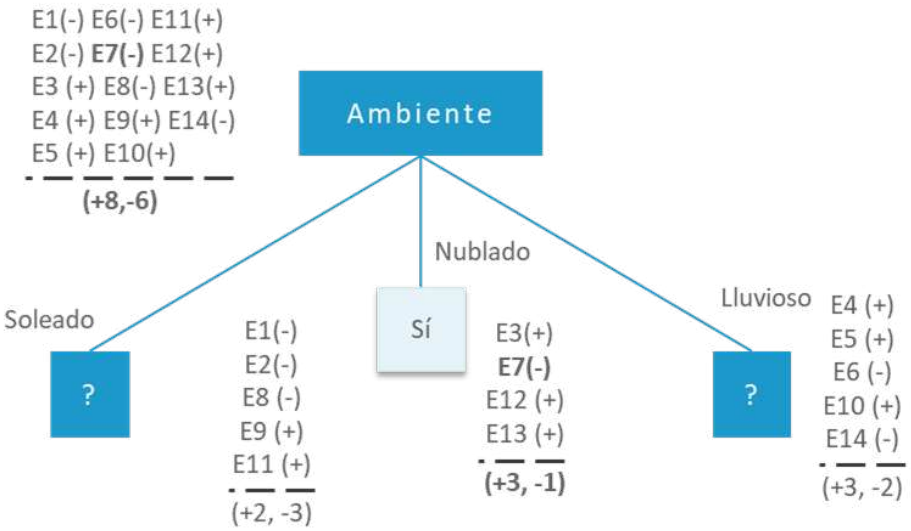


Figura 5. Árbol parcial generado en la primera iteración del algoritmo sobre los datos de ejemplo de la Tabla 1 (con la clase del Ejemplo 7 (E7) cambiada a valor no.

El sobreajuste es un problema práctico en la construcción de árboles de decisión y generalmente se ha de aplicar alguna estrategia para mitigar el problema.

Dos tipos de estrategias para evitar el sobreajuste son:

Pospoda

Podar el árbol una vez generado. Se pueden llegar a tener en cuenta combinaciones de atributos antes de realizar la poda. Existen ocasiones en que dos o más atributos combinados aportan bastante información en la clasificación mientras que los atributos por sí solos, no.

Prepoda

Limitar el crecimiento del árbol. Tiene la ventaja de que ahorra los costes de procesamiento debidos a generar nodos y ramas que posteriormente serían podados (utilizando un método de pospoda)

Si el árbol se mapea a reglas, se puede realizar una poda más eficaz, siguiendo los pasos que se indican a continuación:

- Generación del árbol de decisión a partir de los datos de entrenamiento.
- Mapeado del árbol a un conjunto de reglas. Cada camino desde la raíz hasta la hoja corresponde a una regla (una serie de condiciones enlazadas con el operador AND).

- Poda de cada regla eliminando las condiciones en el antecedente que suponen mejorar la precisión de la clasificación.
- Ordenación de las reglas en función de la precisión estimada y se clasifican los ejemplos futuros empleando las reglas en este orden.

Una diferencia entre podar el árbol directamente o trabajar en la poda de reglas es que cuando se elimina un nodo de un árbol se están eliminando diferentes reglas que resultan de la condición testeada en ese nodo, mientras que, si se trabaja con reglas, la decisión de poda se realiza de una manera aislada para cada camino que pasa por ese nodo. Además, si se trabaja con reglas en la poda es indiferente si el atributo testeado se encuentra cerca de la raíz o cerca de la hoja.

¿Cómo se puede saber cuál es el tamaño del árbol adecuado? Es difícil determinar cuándo se ha de parar el crecimiento del árbol o hasta cuánto se ha de podar.

Cuando se poda un nodo de un árbol, se eliminan las ramificaciones y se convierte a ese nodo en una hoja a la que se puede asignar la clase mayoritaria de los ejemplos vinculados a ese nodo. La decisión de podar típicamente se tomará en función de medidas estadísticas que permiten conocer cuáles son las ramas menos fiables.

Una práctica frecuente es utilizar la técnica de validación cruzada (cross-validation). La validación cruzada permite estimar el ajuste del modelo a un hipotético conjunto de datos de prueba cuando no se dispone de este conjunto de datos de prueba de manera explícita. Consiste en dividir el conjunto de ejemplos disponibles en un conjunto de datos de entrenamiento y un conjunto de datos de validación:

- **Datos de entrenamiento**: se utilizan para generar el árbol.
- **Datos de validación**: se utilizan para validar la precisión del árbol generado sobre datos futuros.

La etapa de validación nos puede servir para evaluar la efectividad de la poda del árbol a la hora de clasificar instancias futuras. Se puede, por ejemplo, valorar si podar cada nodo del árbol en base a si el árbol podado resultante clasifica al conjunto de validación igual o mejor que el árbol original. Iterativamente se van eliminando nodos según este criterio, eligiendo como nodo al que se poda a aquel que mejora la clasificación del conjunto de datos de validación. De esta manera, los nodos creados debido a datos ruidosos o erróneos aislados son eliminados.

La desventaja de este método es que se dispone de menos datos para el entrenamiento y es preciso disponer de suficientes datos, tanto en el conjunto de datos de entrenamiento como en el conjunto de datos de validación, de tal manera que las muestras sean significantes desde un punto de vista estadístico. Es típico utilizar 2/3 de los datos disponibles para el entrenamiento y 1/3 de los datos disponibles para la validación.

Cuando el conjunto de datos disponible es pequeño se suele emplear la técnica conocida como **validación cruzada de *k* iteraciones *(K-fold cross-validation*)**. Mediante esta técnica los datos se dividen en *k* subconjuntos de igual tamaño. Un subconjunto es utilizado como datos de prueba (o validación), mientras que el resto de los subconjuntos, (los *k-1* subconjuntos restantes), son utilizados como datos de entrenamiento. La validación cruzada se repite *k* veces, y en cada una de las repeticiones se utiliza uno de los subconjuntos como datos de prueba.

Para obtener el resultado final, se realiza la media de los resultados obtenidos en cada iteración. El número de iteraciones conveniente dependerá de los datos disponibles y del número de atributos, pero es habitual utilizar una validación cruzada de 10 iteraciones, ya que suele ofrecer buenos resultados.

3.8. Medidas de la precisión de la clasificación. Curva ROC

Como previamente se ha indicado, se puede utilizar un conjunto de ejemplos de los disponibles (datos de validación) para evaluar la hipotética efectividad de la clasificación de instancias futuras por un árbol de decisión inducido y por los diversos árboles que resultan de la poda. Sin embargo, para esta evaluación se está teniendo en cuenta un conjunto de datos de prueba limitados y, por ello, surge la siguiente pregunta: **¿Cómo podemos estimar la precisión real en la clasificación de futuras instancias?** Es evidente que no es lo mismo contar con conjuntos de datos de entrenamiento y validación numerosos que no contar con ellos.

Es lógico que se confíe más en una clasificación inducida a partir de 1 000 000 de datos de entrenamiento que a partir de un conjunto de 100 instancias. Igualmente será más lógico confiar en una validación que utiliza un numeroso conjunto de datos de validación. Si, por ejemplo, se están utilizando *N=100* ejemplos de prueba y, de esos 100 ejemplos, 90 están bien clasificados de acuerdo con una hipótesis *h*, se tiene una tasa de éxito *te* de 90 %. ¿Cómo se puede extrapolar este hecho a futuras instancias? ¿Cuál es la mejor estimación de la precisión de *h* a la hora de clasificar futuras instancias?

En principio se podría considerar que una buena estimación de la tasa de éxito de clasificación de futuras instancias es 0.9. Sin embargo, también se sabe que, dado que la muestra de futuras instancias no va a ser exactamente igual que la muestra de instancias de prueba *N*, se debería aplicar un intervalo de confianza en valores alrededor de *te* con el fin de tener en cuenta esta desigualdad en las muestras.

Para calcular dicho intervalo se asumen ciertas condiciones con el fin de aplicar teorías estadísticas que simplifican el problema. Por ejemplo, se asume que la predicción de la clase de cada una de las instancias es un evento independiente del

resto de predicciones. Este evento tiene dos resultados posibles: éxito o error en la predicción.

Por tanto, las predicciones de las clases de las diferentes instancias constituyen una serie de eventos independientes, con dos posibles resultados, siendo un proceso de Bernoulli. Así, consideramos la tasa de éxito esperada como una variable aleatoria *te* para un conjunto de *N* muestras, con una media de p y una varianza *p(1-p)/N*.

Si tenemos un gran conjunto de muestras, esto es, si *N* es grande (*N>100*), la distribución de la tasa de éxito será próxima a una distribución normal. Para una distribución normal, la probabilidad de que una variable aleatoria *X*, con una media igual a 0, se inscriba dentro de un cierto rango de confianza de anchura *2z*, $P[-z \leq X \leq z] = c$, se puede obtener a partir de tablas de distribución de probabilidad normal *N(0,1)*, que permiten relacionar *z* con el nivel de confianza deseado.

En la Tabla 2 se muestran algunos valores de *z* que se pueden extraer de dichas tablas, en este caso se relaciona *z* con la probabilidad de que la variable X sea superior a *z*, $P[X \geq z]$. Dado que se asume en dicha tabla que la variable X tiene una media de cero y una varianza de 1, podemos considerar que las cifras de *z* se miden en desviaciones estándar de la media. Por tanto, un valor de $P[X \geq z]$ = 0.01 implica que hay un 1 % de probabilidad de que el valor de X sea superior a 2.33 veces la desviación estándar sobre el valor de la media. Dado que tenemos una distribución simétrica, existe también un 1 % de probabilidad de que el valor de *X* sea inferior a -2.33 veces la desviación estándar sobre el valor de la media, luego se da la probabilidad de que *X* se encuentre en el intervalo de confianza *[-2.33, 2.33]* de $P[-2.33 \leq X \leq 2.33] = 1 - 0.01 - 0.01 = 0.98$.

$P[X \geq z]$	z
1%	2.33
5%	1.65
10%	1.28
25%	0.68
40%	0.25

Tabla 2. Límite del intervalo de confianza *z* para diferentes valores de confianza.

Para poder utilizar estas tablas cuando la media no es igual a 0 ni la varianza es igual a 1, tenemos que restar a la variable aleatoria te la media *p* y dividir por la desviación estándar $\sqrt{p(1-p)/N}$:

$$P\left[-z < \frac{t_e - p}{\sqrt{\frac{p(1-p)}{N}}} < z\right] = c \qquad (6)$$

A partir de la anterior expresión, se obtienen las siguientes fórmulas para calcular los intervalos de confianza (**Witten & Frank, 2005).** El límite superior del intervalo viene dado por:

$$p = \frac{\left(t_e + \frac{z^2}{2N} + z\sqrt{\frac{t_e}{N} - \frac{t_e^2}{N} + \frac{z^2}{4N^2}}\right)}{\left(1 + \frac{z^2}{N}\right)} \qquad (7)$$

El límite inferior del intervalo viene dado por:

$$p = \frac{\left(t_e + \frac{z^2}{2N} - z\sqrt{\frac{t_e}{N} - \frac{t_e^2}{N} + \frac{z^2}{4N^2}}\right)}{\left(1 + \frac{z^2}{N}\right)} \qquad (7)$$

Por ejemplo, si tenemos una tasa de éxito con 1 000 datos de prueba de 0.75 y queremos una confianza *c = 0.80*, utilizamos el valor de la tabla $P\left[X \geq z\right] = 10$, siendo *z = 1.28*. Así, el intervalo que se obtendría para *p* es [0.732, 0.767]. Podemos afirmar con un 80% de confianza que la tasa de éxito real se encontrará en dicho intervalo.

El intervalo será más estrecho según *N* aumente. Lógicamente es más fiable un resultado de validación cuando se tienen más muestras.

Curva ROC

La medida más popular que se utiliza para determinar el rendimiento de un sistema de clasificación es su precisión (*accuracy*): el porcentaje de instancias clasificadas correctamente. La precisión es una medida fácil de entender y de comparar, pero obvia algunos aspectos importantes en el rendimiento como pueden ser, por ejemplo, el número de verdaderos positivos (TP – *True* Positive), falsos positivos (FP – *False Positive*), verdaderos negativos (TN – *True Negative*) y falsos negativos (FN – *False Negative*). Estos aparecen siempre que clasifiquemos una clase para la que tengamos instancias pertenecientes a esa clase e instancias que no pertenecen a ella. A la hora de clasificar, la mayoría de los clasificadores producen una puntuación sobre la que deciden si la instancia pertenece a una clase o no que varía entre 0 (definitivamente negativo) y 1 (definitivamente positivo). La Figura 6 muestra cómo se clasificaría un conjunto de datos de pacientes sanos y enfermos en función del umbral elegido para la toma de la decisión.

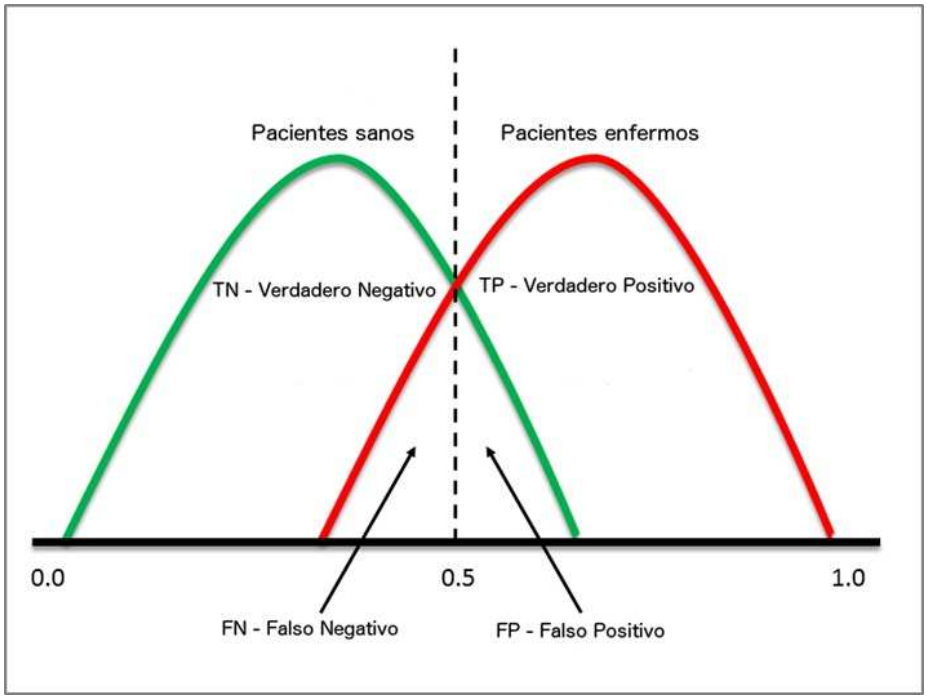


Figura 6. Conjunto de datos de pacientes sanos y enfermos y su clasificación en función del umbral elegido.

Las medidas mencionadas anteriormente dan lugar a otras medidas también interesantes a la hora de evaluar el clasificador:

- **Sensibilidad** (*sensitivity*): también conocido como ratio de verdaderos positivos (TPR – *True Positive Rate*). Mide la capacidad del clasificador para detectar la clase en una población con instancias de esa clase, la proporción de casos positivos bien detectados. Se calcula como TP / (TP + FN).

- **Especificidad** (*specificity*): mide la capacidad del clasificador para descartar instancias que no pertenecen a la clase en una población con instancias de esa clase, la proporción de casos negativos correctamente detectados. Se calcula como TN / (TN + FP).

- **Ratio de falsos positivos** (FPR – *False Positive Rate*): Proporción de casos negativos que el clasificador detecta como positivos. De forma más concreta FP / (FP + TN).

En base a estas dos medidas, sensibilidad y especificidad, se forma la curva ROC (*Receiver Operating Characteristic*). Para ello se representa el TPR o sensibilidad en el eje de ordenadas y el FPR en el eje de abcisas (*1 - especificidad*). La curva ROC traza el TPR frente al FPR a diferentes umbrales de clasificación. Al disminuir el umbral de clasificación clasifica más elementos como positivos, aumentando así tanto los falsos positivos como los verdaderos positivos.

El área de la curva ROC tomará valores entre 0,5 (cuando no el sistema no distinga entre un positivo y un falso positivo) y 1 (el sistema clasifica perfectamente para esta clase). La Figura 7 muestra la curva ROC para dos clasificadores diferentes.

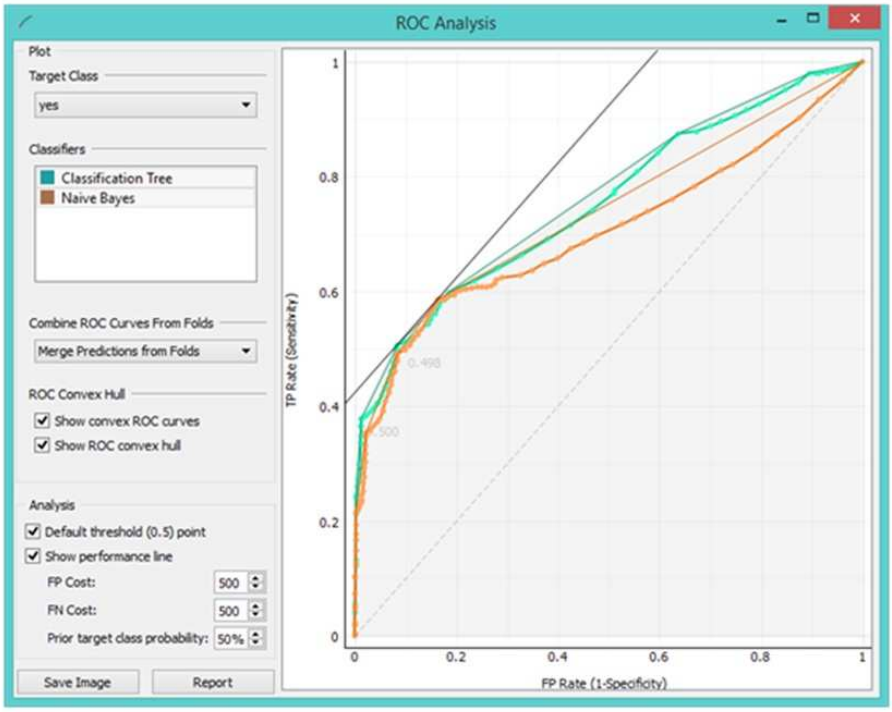


Figura 7. Análisis de la curva ROC de los clasificadores efectuado con la herramienta Orange3. Fuente: https://orange3.readthedocs.io/en/3.4.0/widgets/evaluation/rocanalysis.html

3.9. Simplificación de árboles de decisión mediante poda: algoritmo C4.5

El algoritmo $C_{4.5}$ es un método de inducción de árboles de decisión basado en ID3. Este algoritmo fue propuesto por J.R. Quinlan que escribió un libro sobre el mismo (Quinlan, 1993).

Mientras que el algoritmo ID3 se aplica a atributos con valores discretos, el algoritmo $C_{4.5}$ se puede aplicar tanto a atributos con valores discretos como a atributos con valores continuos. En este último caso, en cada nodo se ha de establecer un umbral de valor del atributo para realizar la partición de ejemplos. Además, $C_{4.5}$ puede trabajar con datos ausentes, que no son tenidos en cuenta a la hora de calcular las métricas para seleccionar los atributos. El método de selección de atributos que utiliza $C_{4.5}$ es la medida de proporción de ganancia.

$C_{4.5}$ realiza una **poda** tras la generación del árbol (*pospoda*) con el fin de mejorar la generalización del modelo. Una vez que el árbol ha sido generado, $C_{4.5}$ elimina aquellos nodos del árbol para los cuales el resultado de podar es una mejora en la precisión en la clasificación.

¿Cómo el algoritmo $C_{4.5}$ estima la precisión de la clasificación? La estadística en la que se basa el algoritmo $C_{4.5}$ se considera débil, pero, al mismo tiempo, ofrece buenos resultados en la práctica. $C_{4.5}$ no utiliza un conjunto de ejemplos para la validación separado de los datos de entrenamiento, sino que **la validación la realiza a partir de los mismos datos de entrenamiento.**

Tras la poda, $C_{4.5}$ asigna a un nodo la clase mayoritaria en el conjunto de instancias que alcanzan ese nodo, y esto da lugar a un determinado número de errores *e* debido a que hay instancias que no pertenecen a esa clase. Para dicho nodo la tasa de error tendría un valor *terror = e/N*, siendo *N* el número de instancias que lo alcanzan.

Esta estimación, al ser calculada sobre los mismos datos de entrenamiento, resulta en una estimación claramente optimista y la predicción del error está por tanto bastante sesgada.

Para compensar el hecho favorable que supone el emplear los propios datos del entrenamiento para el cálculo de la tasa de error, C4.5 realiza lo que se denomina una **poda pesimista**, ajustando esa tasa de error con una penalización al valor obtenido.

Para un determinado nivel de confianza, toma el nivel más desfavorable del intervalo de confianza como medida del error (lo que correspondería al límite superior del intervalo expresado en la Ecuación 7 de la sección anterior) y no se tiene en cuenta la posibilidad de que el valor pueda encontrarse dentro del intervalo de confianza. Por defecto $C_{4.5}$ toma un valor de confianza *c = 0.25*.

Para estimar la tasa real del error *terror_real* a partir de la tasa observada *terror*, obtiene el límite superior del intervalo de confianza a partir de las siguientes expresiones (Witten & Frank, 2005):

$$P\left[\frac{t_{error} - t_{error_{real}}}{\sqrt{\frac{t_{error_{real}}\left(1 - t_{error_{real}}\right)}{N}}} > z\right] = c \qquad (9)$$

Tomando el límite superior del intervalo obtenemos:

$$t_{error_{real}} = \frac{\left(t_{error} + \frac{z^2}{2N} + z\sqrt{\frac{t_{error}}{N} - \frac{t_{error}{}^2}{N} + \frac{z^2}{4N^2}}\right)}{\left(1 + \frac{z^2}{N}\right)} \qquad (10)$$

Siendo z=0.68 para un valor de c=0.25 (ver Tabla 2).

Para conjuntos de datos muy grandes la desviación estándar tenderá a ser pequeña, con lo cual la estimación pesimista se acercará al error observado (Mitchell, 1997).

3.10. Ensemble Learning y Random Forest

En la Figura 1 del Tema 1 vimos que una de las ramas del aprendizaje automático o *machine learning* era el ***ensemble learning***, también conocido como ***integrated learning*.** En castellano las traducciones habituales son ***aprendizaje ensemble*** o ***aprendizaje integrado***.

Actualmente, los métodos más modernos y maduros utilizados para obtener los resultados más precisos en entornos de producción, además de las redes neuronales, que veremos en los Tema 5 y 6, son precisamente los métodos *ensemble learning*, a pesar de que las primeras son más populares.

En los métodos de aprendizaje integrado, la idea es unir un conjunto de algoritmos ineficientes en los que cada uno colabora corrigiendo los errores del resto del conjunto (Krawczyk et al., 2017). De esta manera, se consigue una calidad general más alta que la de los mejores algoritmos individuales que trabajan de forma aislada.

De hecho, se obtienen resultados aún mejores cuando los algoritmos que componen el conjunto son más inestables, en el sentido de predecir resultados completamente diferentes con poco ruido en los datos de entrada. Por lo tanto, es común utilizar algoritmos como la regresión y los árboles de decisión como parte de los métodos de aprendizaje integrado, porque estos algoritmos son muy sensibles a los valores atípicos en los datos de entrada. Por el contrario, algoritmos muy estables como el Naïve Bayes o el k-NN (*k Nearest Neighbors* o los *k vecinos más cercanos*) son utilizados raramente en los métodos *ensemble learning*.

Como vemos, el aprendizaje integrado no está exclusivamente relacionado con los árboles de decisión. Sin embargo, lo abordamos en este tema con el fin de

aprovechar para describir uno de los algoritmos *ensemble learning* más populares: ***el random forest.***

Existen tres métodos principales dentro del aprendizaje integrado en función de cómo se combinan los diferentes algoritmos que componen cada método: el ***stacking*** (apilamiento), el ***bagging*** (embolsamiento) y el ***boosting*** (refuerzo). Estos tipos de métodos se utilizan actualmente para cualquier aplicación en la que se puedan aplicar los algoritmos supervisados clásicos, aunque también existen métodos de aprendizaje integrado dentro del aprendizaje no supervisado, así como los sistemas de búsqueda, la visión por ordenador o la detección de objetos.

Stacking

Los métodos *stacking* (o *stack generalization*) son quizás los métodos menos utilizados de todos (comparados con los métodos *bagging* y *boosting*). El concepto en el que se basan es muy simple (ver Figura 8): entrenar diferentes algoritmos (por ejemplo, un k-NN, un árbol de decisión y un SVM – *Support Vector Machine* o Máquina de vectores de soporte, creadas por Vapnik) (Noble, 2006) utilizando los mismos datos y utilizarlos en paralelo para clasificar la misma entrada.

Finalmente, se promedian las salidas de los diferentes algoritmos para obtener un valor de salida final, normalmente utilizando un algoritmo de regresión (Cao y Wang, 2018).

Un ejemplo de este tipo de técnica es el **FWLS** (*Feature-Weighted Linear Stacking* o Apilamiento lineal ponderado por características), utilizado como método de recomendación de contenido en plataformas de comercio en línea (Henriques y Mendes-Moreira, 2016).

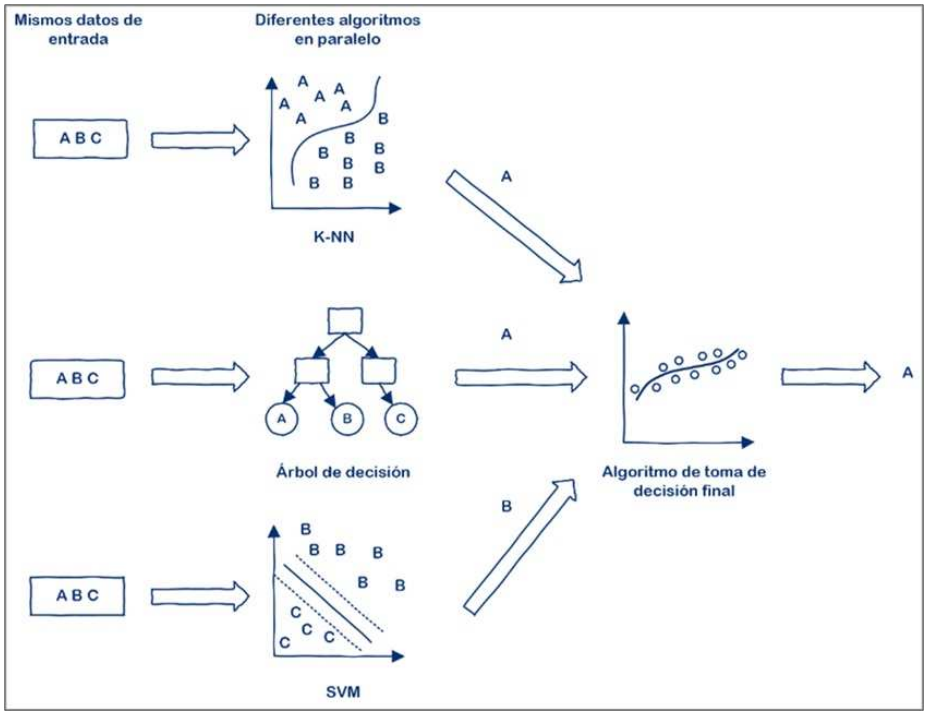


Figura 8. Esquema de funcionamiento en los métodos stacking.

Bagging (y el Random Forest o bagging con árboles de decisión)

Por su parte, en los métodos *bagging* (*Bootstrap AGGregatING*) se utiliza un conjunto de algoritmos en paralelo en los que todos los algoritmos son iguales (por ejemplo, todos son árboles de decisión), pero cada uno de ellos se entrena con un subconjunto aleatorio de datos extraídos del mismo conjunto de datos de entrenamiento (Wu et al., 2018).

Por último, y al igual que en los métodos *stacking*, se promedian las salidas de los diferentes algoritmos para obtener un valor de salida final, de nuevo normalmente utilizando un algoritmo de regresión. La Figura 9 muestra el esquema de funcionamiento de este tipo de métodos.

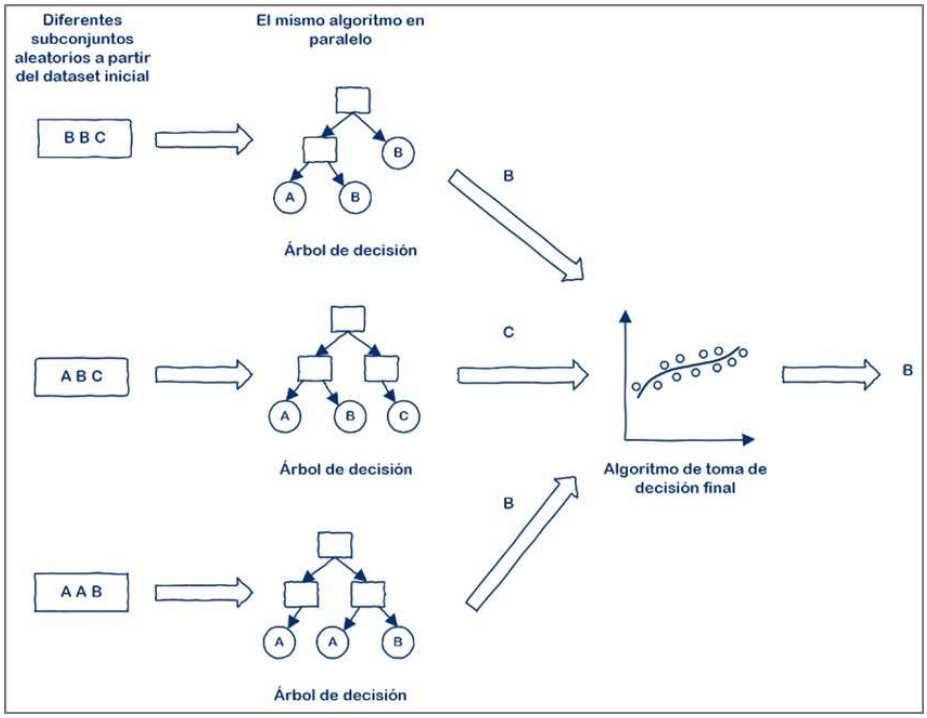


Figura 9. Esquema de funcionamiento en los métodos bagging (usando como ejemplo el random forest).

De hecho, uno de los métodos de apilamiento más conocidos es el *random forest* (o árbol aleatorio), en el que se utilizan precisamente árboles de decisión como algoritmos a aplicar en paralelo. El algoritmo *random forest* se utiliza ampliamente, por ejemplo, en los teléfonos móviles para reconocer dónde están las caras en una imagen cuando se utiliza una aplicación de cámara y mostrar el encuadre de las mismas en la fotografía (Kremic y Subasi, 2016). Aunque puede perderse cierta precisión con respecto a una red neuronal, el bosque aleatorio requiere muchos menos cálculos y es más factible utilizarlo en aplicaciones en tiempo real en dispositivos móviles debido a sus menores requisitos computacionales.

Boosting

En el caso de los métodos ***boosting***, los diferentes algoritmos elegidos se entrenan uno por uno de forma secuencial en serie (en lugar de trabajar en paralelo, como sucede en los métodos *stacking* y *bagging*) (Al Daoud, 2019). Es decir, el primer algoritmo se entrena utilizando todos los datos de entrenamiento elegidos como entrada. Después, se observa qué casos han dado peores resultados después del primer algoritmo y se les da prioridad a la hora de elegir el subconjunto de datos de entrenamiento del segundo algoritmo, y así sucesivamente.

Todos los algoritmos podrían ser, por ejemplo, árboles de decisión. La principal ventaja de este tipo de algoritmos es que son muy precisos cuando se aplican como métodos de clasificación. Por el contrario, y como podría deducirse, su principal desventaja es que los algoritmos no se ejecutan en paralelo, sino en serie, y requieren un mayor tiempo de cálculo. No obstante, siguen siendo más rápidos que las redes neuronales.

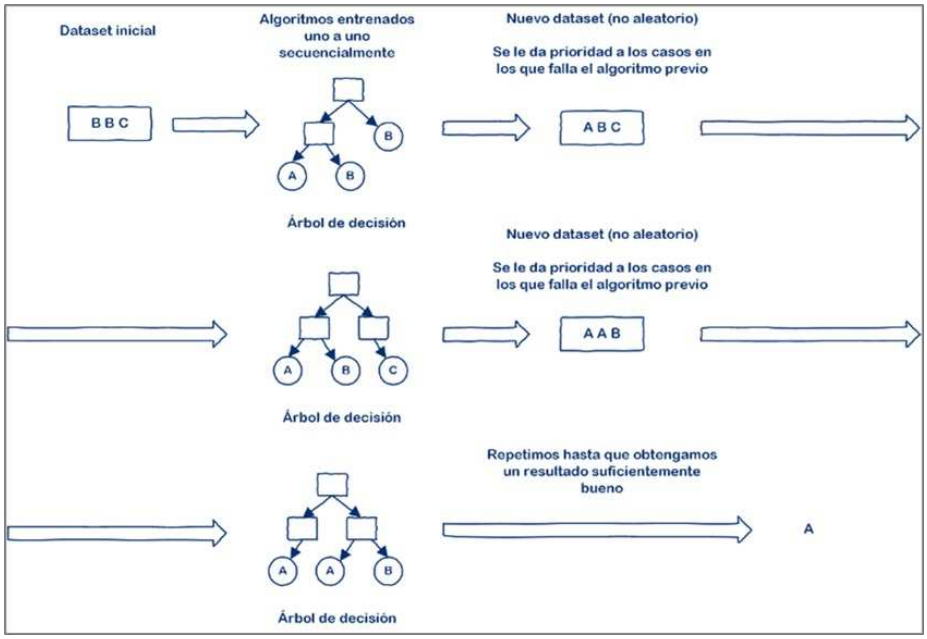


Figura 10. Esquema de funcionamiento en los métodos boosting.

Ideas clave

Entre los principales algoritmos *boosting* se encuentran AdaBoost, CatBoost, LigthGBM o XGBoost. Una de las principales aplicaciones de este tipo de algoritmos ha sido ordenar los resultados de las búsquedas en Google o en redes sociales (Wu et al. 2010).

3.11. Aplicaciones y ejemplos de implementación

Algunas aplicaciones de los árboles de decisión en la literatura

El aprendizaje del árbol de decisión es una de las técnicas más utilizadas para el aprendizaje inductivo, siendo un método bastante robusto frente a los ruidosos datos comparativos. Los árboles de decisión se utilizan como modelo predictivo para vincular los valores conocidos de los atributos de los elementos, representados como ramas en el árbol, con las conclusiones sobre un valor de atributo de elemento normalmente desconocido, por ejemplo, una etiqueta para la clase de elemento, representada como las hojas del árbol (Shalev-Schwartz y Ben-David, 2014).

Las entradas y salidas de la función objetivo suelen ser valores discretos, aunque también podrían ser continuos en el caso de las entradas. La representación de esta función objetivo toma la forma de un árbol y se interpreta como una serie de condiciones consecutivas que pueden ser fácilmente mapeadas a las reglas de clasificación y pueden ser fácilmente explicadas por un humano. Los árboles cuyas hojas toman valores discretos se denominan **árboles de clasificación**. En cambio, aquellos cuyas hojas pueden tomar valores continuos se denominan **árboles de regresión**. Los árboles de decisión se utilizan generalmente para el análisis de decisiones y también para describir datos en la extracción de datos.

Se utilizan en el diagnóstico de enfermedades en la medicina (Song y Ying, 2015) o en la determinación de la concesión de un préstamo, entre muchas otras posibilidades (Namazkhan et al., 2020). Otros posibles usos de los árboles de decisión incluyen la implementación de clasificadores para filtrado de *sitios webs spam* (Silva et al., 2016).

U n *sitio web spam* es una web creada para deliberadamente engañar y dañar potencialmente a los usuarios con el fin de beneficiarse (el contenido no es real, tratan de suplantar a otras webs, utilizan redireccionamientos engañosos, etc.).

Además, tal y como ya se ha explicado, el algoritmo *random forest* se utiliza ampliamente, por ejemplo, en los teléfonos móviles para reconocer dónde están las caras en una imagen cuando se utiliza una aplicación de cámara y mostrar el encuadre de las mismas en la fotografía (Kremic y Subasi, 2016). Aunque puede perderse cierta precisión con respecto a una red neuronal, el bosque aleatorio requiere muchos menos cálculos y es más factible utilizarlo en aplicaciones en tiempo real en dispositivos móviles debido a sus menores requisitos computacionales.

Ejemplos de implementación

En el anterior tema, vimos un ejemplo de cómo utilizar Python y sus librerías para llevar a cabo una regresión lineal, polinómica o una regresión lineal múltiple. Sin embargo, en esta ocasión estamos trabajando con algoritmos supervisados de clasificación, de modo que no podemos aplicar este tipo de técnicas. Sí que podemos utilizar una ***regresión logística***, que, a pesar de su nombre, es un **clasificador.**

La regresión logística se utiliza para dar una probabilidad entre 0 y 1 de que los datos de entrada correspondan a una determinada clase de salida, pero también puede utilizarse para clasificar cualquier número de categorías (por ejemplo, para reconocer qué dígito manuscrito entre 0 y 9 se observa en una imagen). En lugar de utilizar los mínimos cuadrados ordinarios que podrían dar probabilidades inferiores a 0 o superiores a 1, se utilizan ***modelos logit***, como la ***función sigmoidea***:

$$S(x) = \frac{1}{1 + e^{-x}}$$

¿Y por qué contamos esto? Porque vamos a comparar los resultados obtenidos con u n *dataset* determinado cuando utilizamos un árbol de decisión y la regresión logística. **Así aprenderemos también a comparar algoritmos entre sí.**

Nota: más adelante, en el Tema 5, cuando veamos las redes neuronales, aprovecharemos para compararlas con el algoritmo *random forest* en problemas similares, viendo, así, también, un ejemplo de su aplicación con Python.

A continuación, vamos a aplicar tanto la regresión logística como una técnica de árbol de decisión empleando Python y librerías apropiadas. Para ello, vamos a emplear `scikit-learn`, que implementa una versión optimizada del algoritmo CART, aunque sólo empleando variables numéricas. Si necesitamos emplear variables categóricas, podemos transformarlas previamente en numéricas empleando el método `map()` de `pandas`. Este método de esta librería nos permite clasificar de forma binaria (para valores -1 o 1) o como categorías (0, 1, 2, …).

Vamos a emplear uno de los *datasets* más conocidos en *machine learning*, el **iris (el conjunto de datos *flor Iris* o *iris de Fisher***), introducido por Ronald Fisher en 1936 en un artículo como ejemplo de análisis de discriminante lineal. Este *dataset* es un conjunto de datos multivariante de 150 muestras, 50 de cada una de las tres especies de flor Iris (*Iris setosa, Iris virginica* e *Iris versicolor*), con medidas del largo y ancho del sépalo y el pétalo de cada una de ellas. Este *dataset* viene incluido en `scikit-learn`, de modo que lo podemos importar de dicha librería, como ya hemos visto en el Tema 1. No obstante, lo importaremos de OpenML, que, como vemos, tiene una URL de acceso al CSV mediante: https://www.openml.org/data/get_csv/61/dataset_61_iris.arff

Largo de sépalo	Ancho de sépalo	Largo de pétalo	Ancho de pétalo	Especies
5.1	3.5	1.4	0.2	I. setosa
4.9	3.0	1.4	0.2	I. setosa
4.7	3.2	1.3	0.2	I. setosa

Tabla 3. Conjunto de datos flor Iris o iris de Fisher (muestra de las primeras filas).

En primer lugar, y dando por sentado que seguimos manteniendo instalados los módulos de los ejemplos del Tema 2, instalaremos los módulos graphviz (para dibujar diagramas, entre otros, los diagramas de árbol de decisión) y pydotplus, si no los tuviéramos ya en nuestro sistema, con la orden habitual:

```
PS C:\Users\xxx> pip install graphviz
```

```
PS C:\Users\xxx> pip install pydotplus
```

Nota: es necesario también instalar los ejecutables de graphviz para que el ejecutable dot esté en la ruta de búsqueda del sistema y así poder renderizar diagramas:

Fuente: https://www.graphviz.org/download/

También será necesario incluir la ruta al ejecutable en nuestro PATH, por ejemplo, en Windows 10 esto se realizaría en Panel de Control → Sistema → Configuración avanzada del sistema → Variables de entorno, y ahí editar la variable PATH añadiendo, por ejemplo:

```
C:\Program Files (x86)\Graphviz2.38\bin
```

Ideas clave

O la ruta de instalación que hayamos empleado. **Hemos de reiniciar Visual Studio Code si estamos realizando este proceso con el IDE abierto.**

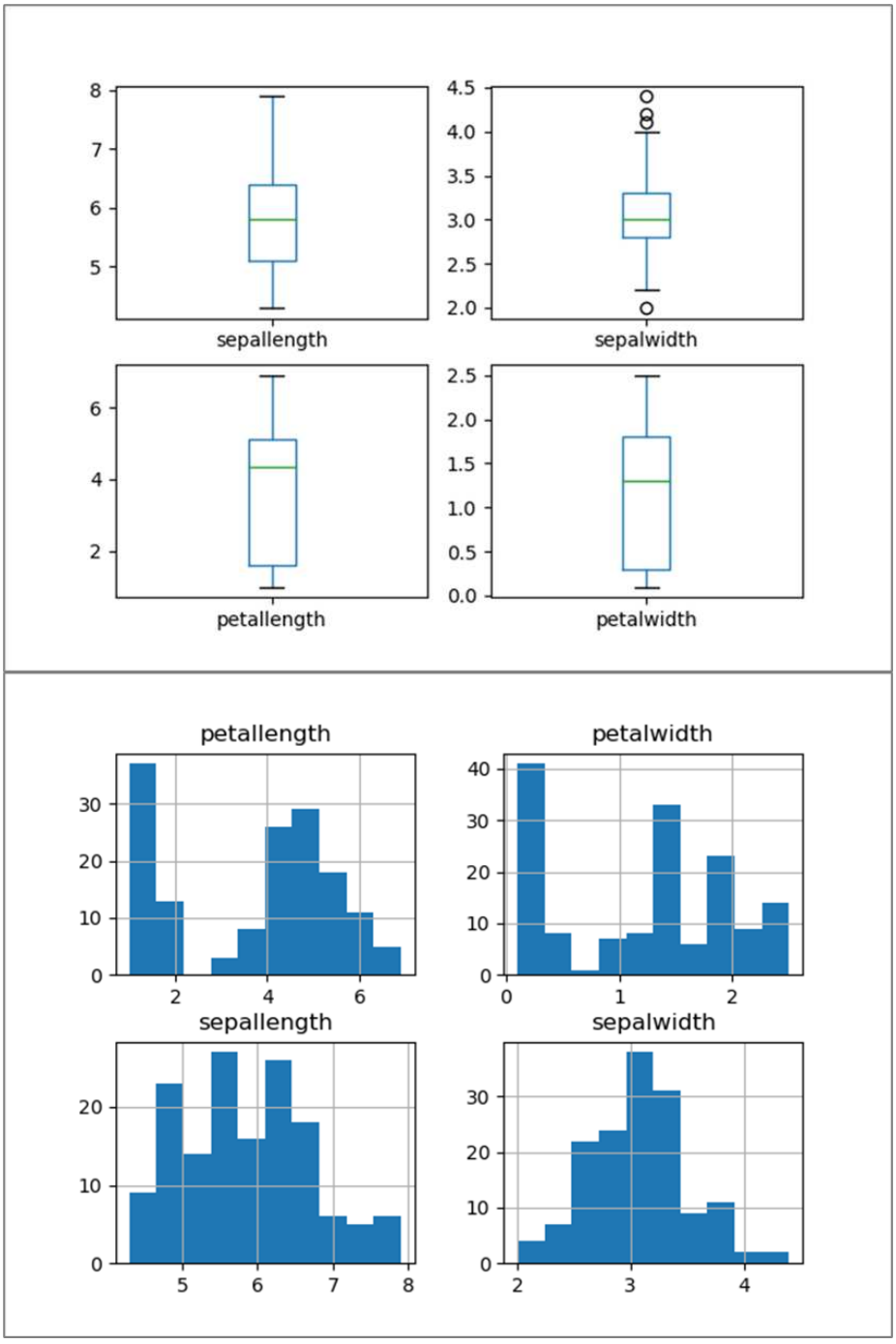
Proseguimos. Primero echamos un vistazo a los datos del *dataset* en modo texto, tal y como se explica en los comentarios del código:

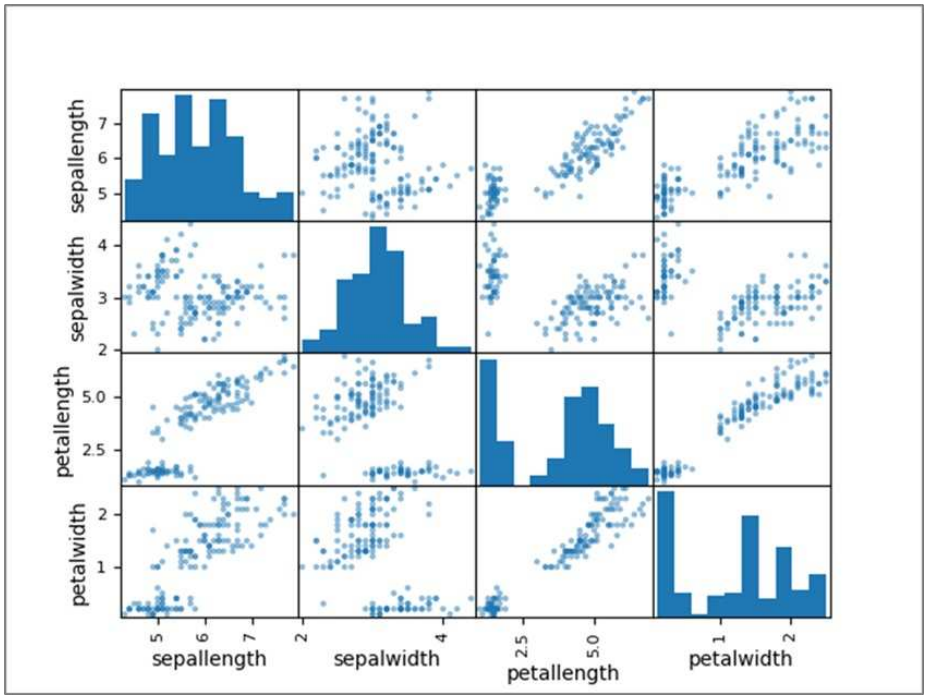
```
# Load libraries
from pandas import read_csv

url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
# La siguiente línea no es necesaria usando esta fuente, pues ya incluye la cabecera
#names = ['sepallength', 'sepalwidth', 'petallength', 'petalwidth', 'class']
#dataset = read_csv(url, names=names)
dataset = read_csv(url)

# mostramos la "forma", debería haber 150 entradas con 5 atributos cada una
print(dataset.shape)
#> (150 , 5)

# mostramos las 3 primeras entradas para echar un vistazo
print(dataset.head(3))
#>   sepallength  sepalwidth  petallength  petalwidth        class
#>0          5.1         3.5          1.4         0.2  Iris-setosa
#>1          4.9         3.0          1.4         0.2  Iris-setosa
#>2          4.7         3.2          1.3         0.2  Iris-setosa

# mostramos un resumen estadístico de los datos
print(dataset.describe())
#>       sepallength  sepalwidth  petallength  petalwidth
#>count   150.000000  150.000000   150.000000  150.000000
#>mean      5.843333    3.054000     3.758667    1.198667
#>std       0.828066    0.433594     1.764420    0.763161
#>min       4.300000    2.000000     1.000000    0.100000
#>25%       5.100000    2.800000     1.600000    0.300000
#>50%       5.800000    3.000000     4.350000    1.300000
#>75%       6.400000    3.300000     5.100000    1.800000
#>max       7.900000    4.400000     6.900000    2.500000

# distribución por clases
print(dataset.groupby('class').size())
#>Iris-setosa        50
#>Iris-versicolor    50
#>Iris-virginica     50
#>dtype: int64
```

Echemos un vistazo a los datos ahora en modo gráfico, de nuevo tal y como se explica en los comentarios del código de ejemplo:

```
# Load libraries
from pandas import read_csv
from pandas.plotting import scatter_matrix
from matplotlib import pyplot

url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.arff"
# La siguiente línea no es necesaria usando esta fuente, pues ya incluye la cabecera
#names = ['sepallength', 'sepalwidth', 'petallength', 'petalwidth', 'class']
#dataset = read_csv(url, names=names)
dataset = read_csv(url)

# gráficos univariable:
 # diagramas de caja (box and whisker)
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
pyplot.show()

 # histogramas
dataset.hist()
pyplot.show()

# gráficos multivariable
 # matriz de dispersión
scatter_matrix(dataset)
pyplot.show()
```

Obteniendo los siguientes gráficos consecutivamente:





Ahora vamos a evaluar dos algoritmos *machine learning* y compararlos entre sí: la regresión logística frente al árbol de decisión CART.

Para ello, separaremos nuestro *dataset*: usaremos un 80 % de los datos para entrenar los algoritmos y un 20 % de los datos para hacer los test de predicción. Esta suele ser una proporción habitual.

Además, utilizaremos una **validación cruzada estratificada de 10 veces (*k-fold*)** para estimar la precisión del modelo.

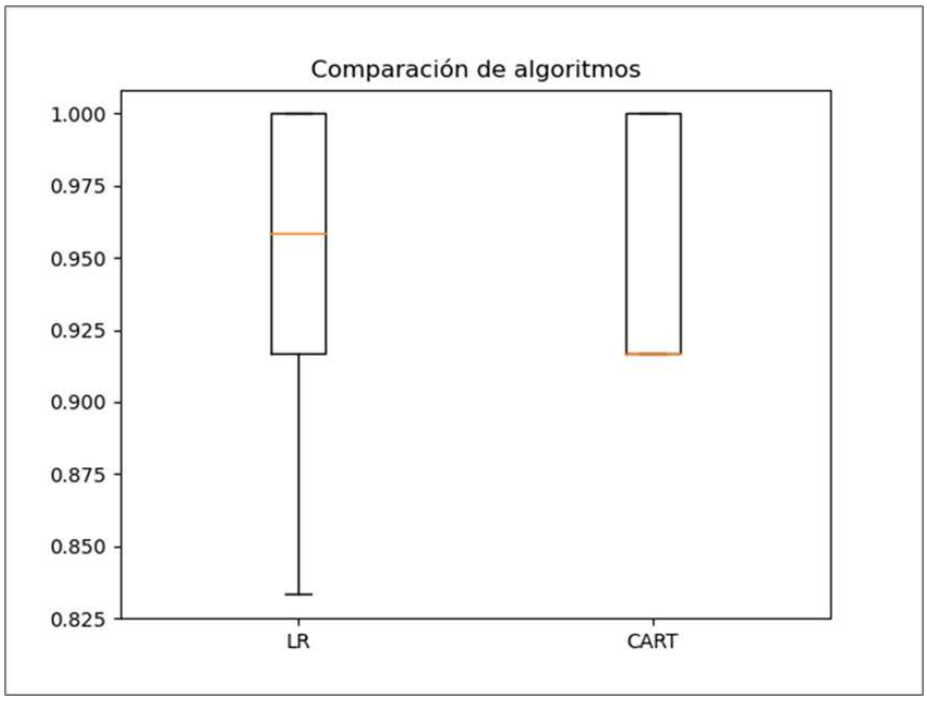
Esto dividirá nuestro conjunto de datos en 10 partes, entrenando en 9 partes y testeando en 1 parte y repetirá para todas las combinaciones de divisiones de entrenamiento-test.

Estratificado, significa que cada pliegue o división del conjunto de datos tendrá como objetivo tener la misma distribución de ejemplo por clase que existe en todo el conjunto de datos de entrenamiento.

```
# Load libraries
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.ar
ff"
dataset = read_csv(url)

# Dividimos el dataset en 80% de datos para entrenar y 20% para
testear
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(
X, y, test_size=0.20, random_state=1, shuffle=True)

# Cargamos los algoritmos
models = []
models.append(('LR', LogisticRegression(solver='liblinear', mult
i_class='ovr')))
models.append(('CART', DecisionTreeClassifier()))

# evaluamos cada modelo por turnos
results = []
names = []
for name, model in models:
    kfold = StratifiedKFold(n_splits=10, random_state=1)
    cv_results = cross_val_score(model, X_train, Y_train, cv=kfo
ld, scoring='accuracy')
    results.append(cv_results)
    names.append(name)
    print('%s: %f (%f)' % (name, cv_results.mean(), cv_results.s
td()))

# Comparación de algoritmos
pyplot.boxplot(results, labels=names)
pyplot.title('Comparación de algoritmos')
pyplot.show()

#>LR: 0.941667 (0.065085)
#>CART: 0.950000 (0.040825)
```

Obtendremos la siguiente gráfica comparativa, además. Como vemos en los resultados por consola (mostrados en los comentarios del código) y en la gráfica, el CART tiene una precisión media del 95 % frente al 94 % de la regresión logística, en estas circunstancias. Nótese que hemos evaluado el algoritmo frente en una

validación cruzada estratificada de 10 veces, de ahí que tengamos una media y una desviación para el valor de la precisión. En ambos el extremo superior del bigote está al 100 % (máximo), además del extremo superior de la caja (percentil 75 o cuartil superior). Esto sucede porque tenemos varias pruebas en las cuales la precisión es del 100 %. Sin embargo, en la regresión logística tenemos precisiones mínimas más bajas, por debajo del 85 %, mientras que en el árbol de decisión tanto la mediana (línea horizontal), como el mínimo (extremo inferior del bigote), como el cuartil inferior (percentil 25) permanecen por encima del 90 %.



Vamos a quedarnos, por el interés de este tema, solo con el algoritmo CART. Vamos a realizar predicciones con el algoritmo de árbol, obteniendo resultados por consola como vemos en los comentarios en el código:

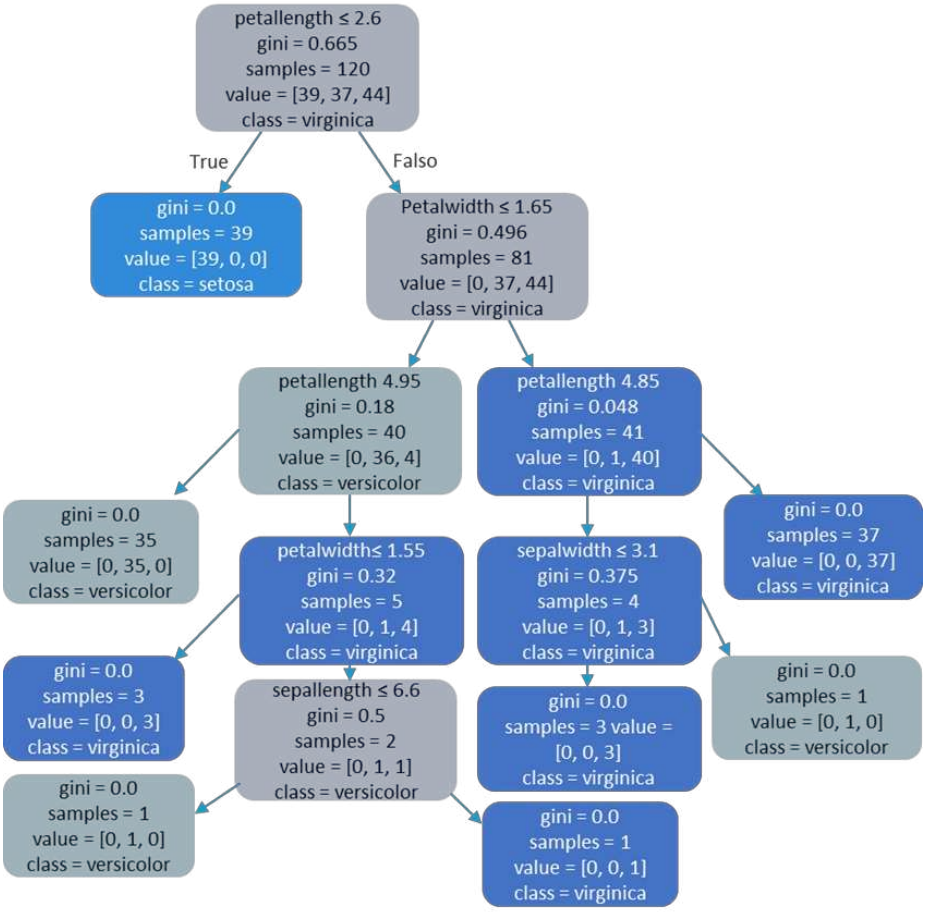
```
# Load libraries
from pandas import read_csv
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.ar
ff"
dataset = read_csv(url)

# Dividimos el dataset en 80% de datos para entrenar y 20% para
testear
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(
X, y, test_size=0.20, random_state=1, shuffle=True)

# Realizamos predicciones con el dataset de validación
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

# Evaluamos las predicciones, en primer lugar la precisión obten
ida
print(accuracy_score(Y_validation, predictions))
#> 0.9666666666666667

# ahora la matriz de confusión (vemos en este ejemplo que sólo h
emos cometido un fallo)
print(confusion_matrix(Y_validation, predictions))
#>[[11  0  0]
#> [ 0 12  1]
#> [ 0  0  6]]

# y finalmente un informe de clasificación que ofrece un desglos
e de cada clase por precisión, recuerdo, puntuación f1 y apoyo,
mostrando excelentes resultados (dado que el conjunto de datos d
e validación era pequeño).
print(classification_report(Y_validation, predictions))
#>                 precision    recall  f1-score   support
#>
#>    Iris-setosa       1.00      1.00      1.00        11
```

```
#>Iris-versicolor       1.00      0.92      0.96        13
#> Iris-virginica       0.86      1.00      0.92         6
#>
#>       accuracy                           0.97        30
#>      macro avg       0.95      0.97      0.96        30
#>   weighted avg       0.97      0.97      0.97        30

# realizamos una predicción de ejemplo:
print(model.predict([[6.0, 3.0, 5.0, 2.0]]))
#>['Iris-virginica']
```

Y, por último, vamos a visualizar el aspecto del árbol:

```
# Load libraries
from pandas import read_csv
import matplotlib.pyplot as plt
import matplotlib.image as pltimg
import pydotplus
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# Cargamos el dataset
url = "https://www.openml.org/data/get_csv/61/dataset_61_iris.ar
ff"
dataset = read_csv(url)

# Dividimos el dataset en 80% de datos para entrenar y 20% para
testear
array = dataset.values
X = array[:,0:4]
y = array[:,4]
X_train, X_validation, Y_train, Y_validation = train_test_split(
X, y, test_size=0.20, random_state=1, shuffle=True)
```

```
# Realizamos predicciones con el dataset de validación
model = DecisionTreeClassifier()
model.fit(X_train, Y_train)
predictions = model.predict(X_validation)

# damos detalles sobre el modelo
print(model)

# mostramos el árbol gráifcamente
data = tree.export_graphviz(model, out_file=None, feature_names=
dataset.columns.values[0:4], class_names=["setosa", "versicolor"
, "virginica"], filled=True, rounded=True, special_characters=Tr
ue)

graph = pydotplus.graph_from_dot_data(data)
graph.write_png('mydecisiontree.png')

img = pltimg.imread('mydecisiontree.png')
imgplot = plt.imshow(img)
plt.show()
```



Obtenemos también los parámetros empleados en el árbol de decisión, que pueden ser modificados para realizar pruebas alternativas al construirlo:

```
DecisionTreeClassifier(ccp_alpha=0.0,
class_weight=None,
criterion='gini',
max_depth=None,
max_features=None,
max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
min_samples_leaf=1,
min_samples_split=2,
min_weight_fraction_leaf=0.0,
presort='deprecated',
random_state=None,
splitter='best')
```

3.12. Referencias bibliográficas

Al Daoud, E. (2019). Comparison between XGBoost, LightGBM and CatBoost Using a Home Credit Dataset. *International Journal of Computer and Information Engineering*, *13*(1), 6-10.

Cao, C. & Wang, Z. (2018). IMCStacking: Cost-sensitive stacking learning with feature inverse mapping for imbalanced problems. *Knowledge-Based Systems*, *150*, 27-37.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I. H. (2009). The WEKA data mining software: An update. Association for Computing Machinery. Disponible en https://doi.org/10.1145/1656274.1656278

Henriques, P. M., & Mendes-Moreira, J. (2016, September). *Combining recommendation systems with a dynamic weighted technique.* In 2016 Eleventh International Conference on Digital Information Management (ICDIM) (pp. 203-208). IEEE.

Krawczyk, B., Minku, L. L., Gama, J., Stefanowski, J. & Woźniak, M. (2017). Ensemble learning for data stream analysis: A survey. *Information Fusion*, *37*, 132-156.

Kremic, E. & Subasi, A. (2016). Performance of random forest and SVM in face recognition. *The International Arab Journal of Information Technology, 13*(2), 287-293.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Namazkhan, M., Albers, C. & Steg, L. (2020). A decision tree method for explaining household gas consumption: The role of building characteristics, socio-demographic variables, psychological factors and household behaviour. *Renewable and*

Sustainable Energy Reviews, 119, 109542.

Noble, W. S. (2006). What is a support vector machine? *Nature biotechnology*, *24*(12), 1565-1567.

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81-106. Disponible en https://doi.org/10.1007/BF00116251

Quinlan, J.R. (1993). *C4.5: programs for Machine Learning*. San Francisco: Morgan Kauffmann.

Shalev-Shwartz, S. & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Silva, R. M., Almeida, T. A. & Yamakami, A. (2016, December). *Towards web spam filtering using a classifier based on the minimum description length principle*. In 2016 15th IEEE International.

Song, Y. Y. & Ying, L. U. (2015). Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, *27*(2), 130.

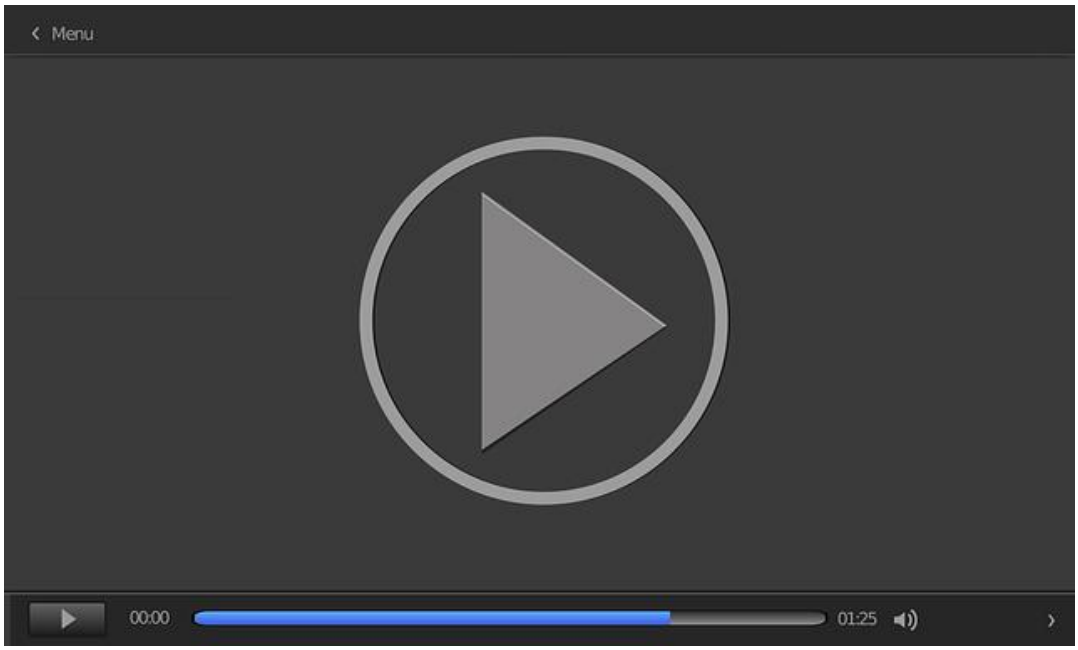
Witten, I.H. & Frank, E. (2005). *Data Mining.* USA: Morgan Kaufmann Publishers.

Wu, Q., Burges, C. J., Svore, K. M. & Gao, J. (2010). Adapting boosting for information retrieval measures. *Information Retrieval*, *13*(3), 254-270.

Wu, Z., Li, N., Peng, J., Cui, H., Liu, P., Li, H. & Li, X. (2018). Using an ensemble machine learning methodology-Bagging to predict occupants' thermal comfort in buildings. *Energy and Buildings*, 173, 117-127.

Introducción a la herramienta Weka

Esta lección magistral consiste en un tutorial introductorio al uso de la herramienta Weka. Se enseñarán funciones básicas tales como la carga de datos y la ejecución de un algoritmo de clasificación, así como la interpretación de los resultados obtenidos.



03. Introducción a la herramienta Weka

Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=aa424a9a-03b6-4e94-9c58-aff800f9dd27

Inducción de árboles de decisión mediante el algoritmo ID3

Quinlan, J.R. (1986). Induction of Decision Trees. Machine Learning, 1, 81-106. http://link.springer.com/article/10.1007%2FBF00116251

Artículo muy ilustrativo escrito por el propio creador del algoritmo ID3, J.R. Quinlan. Resume el problema de aprendizaje de árboles de decisión y propone el algoritmo ID3, exponiendo sus ventajas, limitaciones y dificultades, así como propuestas para solventar estas dificultades.

Software de minería de datos Weka

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I.H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1), 10-18. http://www.cms.waikato.ac.nz/~ml/publications/2009/weka_update.pdf

Artículo escrito por los creadores de Weka proporcionando una introducción a esta herramienta y contando la historia del proyecto.

UCI Machine Learning Repository

Accede a la página desde el aula virtual o a través de la siguiente dirección web: http://archive.ics.uci.edu/ml/

Esta web contiene casi 300 conjuntos de datos disponibles para ser utilizados en tareas de investigación y aprendizaje de la minería de datos. Tiene una interfaz muy útil para filtrar los conjuntos de datos disponibles en función del tipo de aprendizaje que se quiere realizar (clasificación —aprendizaje supervisado, clustering— aprendizaje no supervisado, etc.), los tipos de atributos que se desea manejar (nominales, numéricos, etc.), número de instancias de ejemplo o el área de conocimiento de los datos.

Weka 3: Software de minería de datos en Java

Accede a la página desde el aula virtual o a través de la siguiente dirección web: http://www.cs.waikato.ac.nz/ml/weka/

Sitio web del grupo *Machine Learning Group* de la Universidad de Waikato dedicado a la herramienta Weka. Dispone de una sección para descarga del software, una lista de publicaciones relacionadas con el aprendizaje automático y la minería de datos, así como enlaces a páginas web relacionadas con estos campos y la inteligencia artificial.

Wiki de documentación sobre Weka

Accede a la página desde el aula virtual o a través de la siguiente dirección web: http://weka.wikispaces.com/

Wiki de contenido público cuya finalidad es documentar la herramienta Weka. Contiene una sección muy interesante sobre las preguntas más frecuentes (FAQs), incluyendo cuestiones de uso básico y avanzado.

Test

1. Completa esta sentencia con la opción correcta: «La clasificación mediante árboles de decisión es una tarea de aprendizaje de tipo [...]»:

A. Supervisado.

B. No-supervisado.

C. Supervisado y no supervisado.

D. Ninguna de las anteriores.

2. Indica cuáles de las siguientes afirmaciones son verdaderas:

A. El algoritmo ID3 permite valores numéricos en los atributos.

B. El algoritmo C4.5 permite valores numéricos en los atributos de salida.

C. Los algoritmos ID3 y C4.5 permiten valores nominales en los atributos de salida.

D. El algoritmo C4.5 permite valores numéricos y nominales en los atributos de entrada.

3. Marca de las siguientes opciones aquellas en las que resulta adecuado utilizar un árbol de decisión en la tarea de aprendizaje:

A. No se conoce la clase de las instancias disponibles.

B. La función objetivo tiene valores de salida discretos.

C. Existen varios atributos de salida.

D. Los datos de entrenamiento contienen errores.

4. Indica cuál de las siguientes afirmaciones es correcta:

A. El método de selección de los atributos especifica una heurística para seleccionar el atributo que mejor discrimina los ejemplos para una clase.

B. Un algoritmo básico de construcción de árboles de decisión recibe como entrada los datos de entrenamiento y los atributos de las instancias, y obtiene como salida un método de selección de atributos.

C. El método codicioso es un método de selección de atributos.

D. C4.5 no utiliza un método de selección de atributos.

5. Cuando se tienen atributos con un gran número de posibles valores ¿qué método de selección de atributos conviene utilizar?

A. Ganancia de información.

B. Proporción de ganancia.

C. Indice Gini.

D. Longitud de descripción mínima.

6. Indica cuáles de las siguientes afirmaciones son correctas respecto a ID3:

A. ID3 se basa en el método codicioso o greedy.

B. ID3 considera como heurística que el atributo cuyo conocimiento aporta mayor información en la predicción de la clase es el más útil.

C. ID3 utiliza la medida de proporción de ganancia.

D. ID3 trabaja con un espacio de hipótesis incompleto.

7. Indica las respuestas correctas en relación con los métodos de aprendizaje integrado:

A. En los métodos stacking se utilizan los mismos datos de entrada a diferentes algoritmos en paralelo. Finalmente se utiliza un algoritmo decisor para dar la respuesta final.

B. En los métodos bagging se utilizan el mismo tipo de algoritmos en paralelo, a los cuales se pasa como entrada diferentes subconjuntos aleatorios a partir del dataset inicial.

C. Random forest es un tipo de método boosting utilizando árboles de decisión en serie.

D. En los métodos boosting se parte de un dataset inicial y se entrenan diferentes algoritmos uno a uno secuencialmente, introduciendo como entrada de cada algoritmo un conjunto aleatorio del dataset inicial.

8. Indica cuáles de las siguientes afirmaciones son correctas respecto a C4.5:

A. C4.5 utiliza la medida de proporción de ganancia.

B. C4.5 utiliza un método de prepoda.

C. C4.5 poda el árbol utilizando datos de prueba.

D. C4.5 realiza una poda una vez se ha generado el árbol.

9. Indica cuáles de las siguientes afirmaciones son correctas respecto al método de validación cruzada:

A. Utiliza un conjunto de datos de validación y un conjunto de datos de entrenamiento.

B. La validación cruzada divide los datos disponibles en dos subconjuntos.

C. La validación cruzada no se puede utilizar para evaluar la efectividad de una poda.

D. Cuando se tienen pocos datos no conviene utilizar la validación cruzada con k iteraciones.

10. Indica cuáles de las siguientes afirmaciones son correctas respecto a las medidas de precisión en la clasificación:

A. La tasa de error de una muestra es el cociente entre el número de instancias erróneamente clasificadas y el número total de instancias.

B. Cuanto mayor sea el número de instancias, el intervalo de confianza será mayor.

C. C4.5 utiliza los propios datos de entrenamiento para calcular la tasa de error.

D. Cuanto mayor nivel de confianza se requiere se obtiene un intervalo de confianza menor.