Realizado por:

· Beatriz Espinar Aragón
· Steven Mallqui Aguilar
· Rogger Huayllasco De la Cruz

# ALGORITMO ID3

## PRÁCTICA 2

11 de abril de 2023
Ingeniería del Conocimiento
Grupo E – Ingeniería del Software

UNIVERSIDAD COMPLUTENSE MADRID

# CONTENIDO

# 1. Introducción

La finalidad de este documento es presentar un informe detallado sobre la práctica, consistente en implementar una versión reducida y preliminar del algoritmo ID3. Se explica el proceso y las decisiones tomadas para lograr una aplicación que permita visualizar el funcionamiento del algoritmo.

ID3 es un algoritmo de aprendizaje automático que permite construir árboles de decisión. Para ello, se parte de un conjunto de datos iniciales correspondientes a un conjunto de atributos y se construye el árbol que permitirá posteriormente clasificar nuevos datos. Existen una serie de atributos cuyos valores representan los parámetros iniciales de la situación, y un atributo especial que representa la decisión que se ha tomado en función de los parámetros anteriores.

De este modo, utilizando el ID3 se puede tomar una decisión de forma eficiente a partir de nuevos datos introducidos. A pesar de que el algoritmo se explicará más adelante cuando se profundice en la implementación, su funcionamiento general es el siguiente:

- En cada paso, el algoritmo selecciona el atributo *mejor* (ya se verá qué significa esto de "mejor"), dividiendo los datos en clases distintas.
- Así, se divide el conjunto de datos en subconjuntos a partir de los valores que toma el atributo seleccionado, y se repite el proceso.
- El proceso continúa recursivamente hasta llegar al caso base: sólo se puede tomar una decisión (todos los valores de los atributos restantes llevan asociado un valor concreto del atributo decisión).

Para esta práctica, el conjunto de datos inicial se almacena en dos ficheros únicos incorporado en el proyecto, en el directorio `/input_files`, aunque el programa podría ser fácilmente refactorizado para permitir al usuario introducir su propio fichero de datos. En el fichero `Juego.txt`, cada ejemplar se incluye en una línea del fichero, con los valores de los atributos (incluidos los del atributo decisión) separados por ','. Por otro lado, los atributos se incluyen en otro fichero `AtributosJuego.txt` en una única línea, también separados por ','. A partir de ellos se construye el árbol que, evidentemente, será siempre el mismo.

Por último, es importante explicar el parámetro utilizado en esta versión del algoritmo para seleccionar el atributo que mejor discrimina: el **mérito**. Sean

- $N = nº\ de\ ejemplos\ totales$
- $a_i = nº\ de\ ejemplos\ en\ la\ rama$ i
- $p_i = \%\ de\ ejemplos + en\ la\ rama$ i
- $n_i = \%\ de\ ejemplos - en\ la\ rama$ i
- $r_i = \%\ de\ ejemplos\ en\ la\ rama$ i $= {}^{a_i}/_{N}$

NOTA: En esta versión del algoritmo el atributo decisión sólo toma 2 posibles valores, que asociamos con "positivo" y "negativo".

Así pues, el mérito se calcula aplicando la siguiente fórmula:

$$merito = \sum_{i=1}^{n} (r_i * infor(p_i, n_i))$$

Donde $infor(p,n) = -p * \log_2(p) - n * \log_2(n)$.

Ahora que ya se ha planteado el objetivo del algoritmo, se presentan las secciones que componen este documento para facilitar su lectura:

- Entorno. Se expone brevemente el entorno utilizado para la implementación del algoritmo, así como el lenguaje de programación elegido y otras herramientas de interés.
- Implementación. Se explica con mayor detalle las decisiones relativas al diseño e implementación, tales como las estructuras de datos empleadas o el funcionamiento específico del algoritmo.
- Ampliaciones. Se presentan las ampliaciones realizadas sobre el algoritmo básico (un único nivel de recursividad), incluyendo la lógica implementada y los cambios aplicados sobre el apartado anterior.
- Ejemplos. Se propone al usuario un ejemplo ilustrativo para mostrar el funcionamiento del algoritmo, incluyendo los cambios con respecto a las ampliaciones.

# 2. Entorno

Se explica en este primer apartado el entorno de programación empleado para el desarrollo del algoritmo, con las herramientas y librerías utilizadas.

En primer lugar, el lenguaje de programación escogido para implementar el algoritmo ha sido Python. En particular, la versión más reciente de este lenguaje: 3.11.2. Es la misma decisión que se tomó en la práctica anterior (Implementación del Algoritmo A∗), y se han aplicado los mismos criterios: el amplio rango de librerías muy útiles que proporciona Python y el interés personal de los miembros del equipo por familiarizarse con este lenguaje.

Para el desarrollo del código, de igual manera, se ha optado por utilizar el entorno de desarrollo PyCharm, por estar especializado en Python y por las herramientas que ofrece para detectar errores rápidamente, así como para mantener buenas prácticas de programación.

Por otro lado, en lo relativo a las librerías utilizadas, se listan a continuación las más relevantes para este proyecto:

- `Tkinter` → Se trata de una librería preinstalada de Python que proporciona una interfaz gráfica de usuario, y es una de las más populares para el desarrollo de aplicaciones de escritorio por su simplicidad y flexibilidad. En esta práctica ha sido imprescindible para el desarrollo de la GUI (véase Clase MainWindow y Clase Table), pues todas las ventanas, los botones, la tabla con los datos de entrada, el árbol de decisión, etc, han sido dibujados con herramientas de esta librería.
- `Math` → Se trata de un módulo que proporciona acceso a las funciones matemáticas definidas en el estándar de C. En esta práctica se ha utilizado en la clase del algoritmo (véase Clase ID3) para implementar la función $infor(p, n)$, puesto que era necesario usar logaritmos.
- `NumPy` → Se trata de una librería que permite trabajar con matrices multidimensionales, proporcionando además una gran cantidad de funciones matemáticas y otras herramientas de trabajo con matrices. En esta práctica se ha utilizado para representar la tabla de valores (ejemplos) que utiliza como entrada el algoritmo ID3, y poder obtener los parámetros necesarios de manera eficiente (véase Clase ID3).

Además, se ha utilizado `pyinstaller`, una herramienta que permite convertir el programa de Python en un ejecutable independiente empaquetando todas las dependencias necesarias para poder ejecutarse en cualquier máquina sin tener que instalar el lenguaje o las bibliotecas utilizadas.

Por último, el equipo ha hecho uso de otras tecnologías que han permitido desarrollar la práctica, entre las cuales cabe destacar: GitHub y GitHub Desktop, para el control de versiones y facilitar el trabajo en equipo mediante la creación de un repositorio común; Discord o Whatsapp, como plataformas de comunicación para trabajar simultáneamente sobre el código; Word (Office 365) con OneDrive, para la elaboración de la memoria; y Paint, para el diseño de la interfaz gráfica.

# 3. Implementación

En esta sección se profundiza en la implementación del algoritmo ID3, explicando los módulos en los que se ha dividido el proyecto y las estructuras de datos utilizadas. Además, se explican los pasos que sigue el algoritmo y cómo ha sido plasmado en la interfaz para que el usuario visualice todo el proceso.

El proyecto se ha dividido en 10 módulos principales, que se explicarán a lo largo de este apartado. En «Main» se explicará el funcionamiento inicial de la aplicación, procesando los datos e inicializando la interfaz gráfica con la clase «MainWindow». Esta incluye toda la generación de la GUI, haciendo uso de los módulos «Table», «Tree», «Info», «Rules» y «Decision» para mostrar el contenido del algoritmo. Para la implementación del algoritmo se ha creado la clase «ID3» (en esta sección es donde se explica el flujo principal del algoritmo), que construye el árbol de decisión mediante la clase «Node», que representa el estado del algoritmo en cada momento. Por último, en «Utilities» se incluyen las constantes empleadas por los demás módulos para realizar sus funciones.

Es relevante destacar que, además de las explicaciones más detalladas de este informe, el código de cada clase está documentado, de forma que se incluye al principio de cada clase un breve comentario descriptivo del objetivo de la clase y los atributos que necesita. Por supuesto, el resto del código está igualmente comentado.

NOTA: Todos los conceptos relativos a las ampliaciones se omiten en este apartado, pues se explican ya en los apartados correspondientes (véase Ampliaciones). Por lo tanto, se dejarán las clases «Info», «Rules» y «Decision» para más adelante.

## 3.1. Clase Main

En primer lugar, contamos con el módulo `Main` (realmente no se ha creado una clase, sólo un método `main()`, pues no requiere de atributos ni otros métodos), que inicia la aplicación desde donde se crea la interfaz gráfica. Para dar comienzo a la aplicación, la función principal debe seguir dos pasos:

1) Leer y procesar los datos de entrada
2) Iniciar la interfaz gráfica

El segundo paso sólo consiste en crear una instancia de la Clase MainWindow, que se encargará de inicializar la GUI, invocando al constructor con los datos de entrada. Veamos pues cómo se procesan, recordando el formato de los ficheros explicado en el apartado de Introducción:

- `attributes` → Se trata de una lista (array) donde se guardan los nombres de los atributos, incluido el correspondiente a la decisión (en este caso, "Jugar"). Para guardarlos, basta procesar la línea del fichero `AtributosJuego.txt` obteniendo las palabras utilizando como carácter separador la ','.
- `data` → Se trata de una matriz de la librería `NumPy` donde se guardan los valores de todos los atributos, siendo cada fila un ejemplar y cada columna un atributo. Ahora bien, para crear esta matriz `NumPy` exige dar unas dimensiones iniciales, pero no conocemos de antemano el número de ejemplares que aparecen en el fichero. Por ello, se procesan primero los datos almacenándolos en una lista de listas y, posteriormente, se genera la matriz. Los datos se procesan de igual manera que los atributos, pero repitiendo el proceso para cada línea existente en el fichero.

## 3.2. Clase Node

La clase «Node» es la estructura de datos básica para representar el estado del algoritmo en un momento concreto. Así pues, cada nodo del árbol lleva asociado una serie de atributos:

- `num` → Es el número asociado al nodo. Se utiliza para representarlo (véase Clase Tree) y distinguirlo al mostrar su información (véase Clase Info). Los nodos se irán numerando conforme se vayan creado (véase Clase ID3).
- `attributes` → Es la lista que guarda los nombres de los atributos disponibles[1] en el nivel del árbol donde se encuentra el nodo.
- `data` → Es la matriz que representa los valores de los atributos disponibles en el nivel del árbol donde se encuentra el nodo.
- `merits` → Se trata de un diccionario que guarda para cada atributo disponible (clave) el mérito que tiene (valor). Este diccionario estará ordenado crecientemente por valor, para poder obtener rápidamente el atributo *mejor*, es decir, el que tenga menor mérito.
- `value` → Es el nombre del atributo *mejor*, que es el seleccionado para expandir el nodo. Es, como su propio nombre indica, el parámetro que da valor al nodo a la hora de representarlo.
- `children` → Se trata de otro diccionario que representará los hijos del nodo (vacío, si es nodo hoja). La clave será cada posible valor que toma el atributo *mejor*, que representará la rama, y a esa clave estará asociado como valor el nodo «Node» hijo.

La lógica de esta clase es absolutamente sencilla, pues sólo requiere de los correspondientes getters y setters para controlar el acceso a los atributos que se acaban de presentar.

## 3.3. Clase ID3

La clase «ID3» es la más importante, pues implementa la lógica del algoritmo. Veamos lo primero de todo los atributos que tiene, recordando que aquello relativo a las ampliaciones se estudiará con mayor detalle en dicho apartado:

- `num` → Se trata de una variable global que permite numerar los nodos según se van generando. Es un parámetro útil para dibujar y representar el árbol, como se verá más adelante.
- `attributes_input` → Es la lista inicial de los atributos de entrada.
- `data_input` → Es la matriz de valores iniciales correspondientes a los atributos de entrada.
- `basic` → Es un booleano que indica si el algoritmo debe ejecutarse en su versión básica (un único nivel de recursividad) o completa (todos los niveles). Se utiliza en la ampliación del Algoritmo completo.
- `info` → Es la instancia de la Clase Info, utilizada para poder ir dibujando la información que genera el algoritmo.
- `rules` → Es la instancia de la Clase Rules, utilizada para poder mostrar y calcular las reglas que se deducen del algoritmo una vez este ha terminado.

Ahora, antes de pasar a los pasos que sigue el algoritmo, recordemos que la selección del atributo en un determinado momento se hace en función de su mérito. Ya vimos en el apartado de Introducción que se calculaba así:

[1] Los atributos disponibles son aquellos entre los que se decide cuál es el que mejor discrimina. Inicialmente serán todos los atributos de entrada.

$$merito = \sum_{i=1}^{n} (r_i * infor(p_i, n_i))$$

Donde $infor(p, n) = -p * \log_2(p) - n * \log_2(n)$.

Así pues, la implementación de la función `infor()` es bastante intuitiva, pero veamos cómo se calculan los méritos de los atributos. La función `calculate_merits()` recibe dos argumentos, que son, como siempre, la lista de atributos (`attributes`) disponibles en ese momento, y la matriz de valores (`data`) correspondientes a esos atributos.

- Los méritos se guardan en un diccionario: para cada atributo (clave) se almacena su mérito (valor).
- $N = nº\ total\ de\ ejemplares$: se calcula viendo el número de filas de la matriz `data`.
- Los pasos a seguir para cada atributo $i$ son los siguientes:
  1) Obtener los distintos valores que toma $i$.
  2) Inicializar los valores de $a, p, n, r$. Estos serán diccionarios que guarden para cada valor $v$ del atributo (clave) su correspondiente $a_v, p_v, n_v, r_v$.
  3) Como el mérito es un sumatorio, la variable local `merit` se inicializa a 0.
  4) Se calcula el valor de los 4 parámetros:
     - $a_v = cuántas\ veces\ aparece$ v $en\ la\ columna\ del\ atributo$ i
     - $p_v = \frac{cuántas\ veces\ (el\ atributo\ \text{i}\ tiene\ el\ valor\ \text{v} \wedge el\ atributo\ decisión\ es\ \text{"si"})}{a_v}$
     - $n_v = \frac{cuántas\ veces\ (el\ atributo\ \text{i}\ tiene\ el\ valor\ \text{v} \wedge el\ atributo\ decisión\ es\ \text{"no"})}{a_v}$
     - $r_v = \frac{a_v}{N}$
  5) Actualizar `merit` acumulando el valor anterior $+ r_v * infor(p_v, n_v)$
- Así, el último paso una vez rellenado el diccionario con el mérito de cada atributo, consiste en ordenar el diccionario por valor creciente. Es decir, asegurar que queda ordenado de menor a mayor mérito, para luego conocer fácilmente el atributo *mejor*.

Ahora sí, sólo queda analizar los pasos que sigue el algoritmo. Antes de nada, es importante recordar que los nodos se representan mediante la Clase Node:

1. Lo primero es crear la instancia del nodo raíz con los datos de los que disponemos al principio (número del nodo, atributos disponibles y valores de los atributos).
2. Caso base 1: La columna correspondiente a la decisión no contiene el valor "si". Entonces significa que todos los ejemplos son "no" y por tanto el algoritmo termina, dando valor "no" al nodo hoja.
3. Caso base 2: Análogo al caso base 1, pero cuando la columna no contiene el valor "no" y, por tanto, todos los ejemplos son "si".
4. Caso recursivo: Hay ejemplares tanto positivos como negativos. Entonces:
   a. Calcular el mérito de todos los atributos con `calculate_merits()`.
   b. Elegir el primer elemento del diccionario `merits`, que contendrá el atributo con un mérito menor (es decir, el atributo *mejor*).
   c. Ese atributo seleccionado es ahora el valor del nodo raíz, y lo llamamos `best_attr`.
   d. Preparar llamadas recursivas. Para ello hacemos una copia de los atributos y eliminamos `best_attr`, y después obtenemos los distintos valores que toma.

e. Para cada valor `v`, se hace este paso y los que siguen. Actualizar la variable global `num` para numerar los siguientes nodos que se vayan generando.
f. De la matriz `data`, filtramos las filas cuyo valor de `best_attr` sea `v`.
g. De la matriz `data`, eliminamos la columna de datos correspondiente a `best_attr`.

Siguiendo estos pasos se habrá ejecutado el primer nivel de recursividad del algoritmo ID3, y habremos obtenido el primer atributo que se selecciona por discriminar mejor, junto a las tablas de valores que quedan preparadas para las llamadas recursivas que se harán cuando se implemente el Algoritmo completo.

Nótese que para todas las operaciones realizadas sobre la matriz `data` (ver cuántas veces aparece un valor, obtener los distintos valores que hay en una columna, etc) son necesarios métodos y notaciones que proporciona la biblioteca `NumPy`, pues hacerlas manualmente resultaría notablemente más costoso.

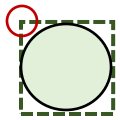
## 3.4. Clase Tree

«Tree» es la clase desde la cual se dibuja el árbol de decisión, esto es, el resultado tras haber ejecutado el algoritmo. Para ello, cuenta con dos atributos principales:

- `root` → Es el nodo raíz del árbol, a partir del cual puede recorrer los demás nodos de manera recursiva.
- `canvas` → Es el lienzo sobre el que se dibuja el árbol.

Así pues, como todo árbol, la manera más intuitiva de recorrerlo es mediante una función recursiva, que en este caso recibe 3 parámetros:

- `node` → Nodo raíz del subárbol que se va a dibujar
- `pos` → Posición del lienzo en la que se debe dibujar la raíz. Cada nodo se representa mediante un círculo y su `pos` es la siguiente:


(esquina superior izquierda del cuadrado que contiene al círculo)

- `right` → Booleano que indica si `node` es el hijo derecho de su padre (`false` si es la raíz del árbol). Se utiliza para saber dónde escribir la información del nodo sin que se superponga con él.

Nótese que este último parámetro y la recursión en sí misma no son necesarios para la versión básica del algoritmo, pues únicamente se entrará al caso base:

→ El árbol consiste en un único nodo que tiene el valor del atributo escogido en el primer nivel de recursividad.

La lógica de esta funcionalidad tiene mayor interés cuando se explique el Algoritmo Completo.

## 3.5. Clase Table

La clase «Table» se ha creado para dibujar las tablas de datos utilizadas por el programa. Se utiliza tanto para los datos de entrada iniciales, como para la información que se va mostrando del algoritmo (véase Clase Info), y dispone de los siguientes atributos:

- `attributes` → Es la lista de los atributos que formarán los nombres de las columnas de la tabla.
- `data` → Es la matriz de los valores de los atributos, que se utilizará para rellenar el contenido de la tabla.
- `frame` → Es el frame en el que se dibuja la tabla (frame izquierdo del panel central de contenido, véase Clase MainWindow).

Para dibujarla se utilizan las `Entry`'s de `tkinter`, que representan las celdas de la tabla. En la primera fila se colocan los atributos, y después se insertan los valores.

### 3.6. Clase MainWindow

En esta sección presentaremos una de las grandes clases que se han implementado en esta práctica: la clase «MainWindow». Es uno de los módulos más importantes porque es la que genera toda la interfaz gráfica, y da funcionalidad a los botones que permiten ejecutar el algoritmo.

Así pues, la clase cuenta con los siguientes atributos:

- `main_window` → Es la ventana de tkinter donde se pinta la interfaz.
- `attributes_input` → Es la lista de los atributos de entrada que ha procesado la Clase Main. Se pasará como parámetro cuando se construya la Clase ID3.
- `data_input` → Es la matriz de datos de entrada correspondiente a los valores de los atributos y, de igual manera, será uno de los parámetros que recibirá la clase del algoritmo.
- `main_frame` → Es el frame principal de la aplicación, el que ocupa toda la ventana.
- `middle_frame` → Es el frame central donde se dibuja el árbol. Tanto este como el `main_frame` son necesarios como atributos para poder ser utilizados desde otros métodos de la clase.
- `canvas` → Es el lienzo donde se dibuja el árbol de decisión. Hace falta guardarlo como atributo para poder pasarlo al constructor de la Clase Tree.
- `basic_button` y `complete_button` → Son los botones que permiten ejecutar las dos versiones del algoritmo. De nuevo, son necesarios como atributos porque hay que modificarlos desde otros métodos de la clase.
- `info_alg`, `rules` y `decision` → Se trata de atributos que representan instancias de las clases Info, Rules y Decision, respectivamente.

Antes de ver los métodos que contiene la clase, veamos la disposición general de la interfaz. La pantalla se divide (horizontalmente) en tres grandes zonas: el `header`, que contiene el título de la aplicación, el `footer`, que contiene los botones, y el `content`, que contiene el resto de elementos.

Dentro del `content_frame`, subdividimos a su vez en tres grandes zonas verticales. La primera (`left_frame`) incluye la tabla de los datos de entrada y las reglas que se deducen del algoritmo. La segunda (`middle_frame`) muestra el árbol de decisión y la sección donde el usuario puede introducir sus propios valores y calcular la decisión correspondiente. La última (`right_frame`) contiene la información del algoritmo, es decir, los pasos y variables que va utilizando el ID3. De este modo, la disposición quedará así:

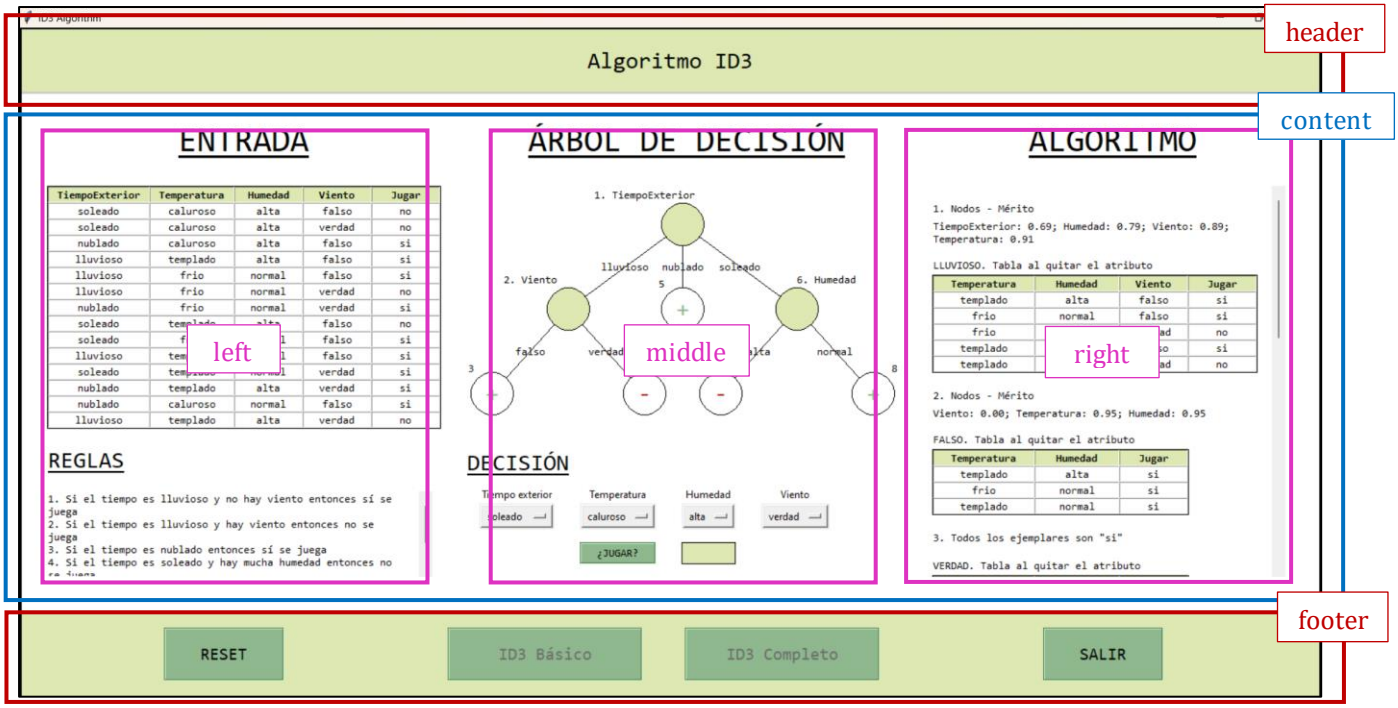



*Ilustración 1. Disposición paneles interfaz*

En cuanto a los botones, aparecen tres funcionalidades principales:

- Reset. Permite resetear la aplicación (importante recordar que una vez ejecutado el algoritmo no puede volverse a ejecutar ninguna de sus versiones, hay que reiniciar).
- Exit. Permite salir y cerrar la aplicación.
- Algoritmo. Un botón para cada versión. Ambas versiones implican la creación de la Clase ID3, pero el algoritmo completo indica el booleano `basic=false`, para que se ejecuten todos los niveles de recursividad. A continuación, dibujan el árbol llamando a la Clase Tree y desactivan los botones del algoritmo para que no pueda volverse a ejecutar. Por último, si se ha ejecutado el algoritmo completo, se activa la funcionalidad de decisión.

Por otro lado, en cuanto a los métodos (aparte de las funcionalidades de los botones), la clase tiene una función `init_gui()` que inicializa la interfaz gráfica, llamando a su vez a los métodos que crean todos los frames explicados anteriormente. El panel más relevante es el `content_frame`, pues crea la tabla de datos de entrada, así como las instancias de Rules, Decision e Info.

### 3.7. Clase Utilities

Por último, se presenta la clase «Utilities», donde se definen las constantes utilizadas por el resto de las clases. Todos los parámetros refieren a la interfaz gráfica, incluyendo la dimensión de la pantalla, las fuentes de letra y los colores utilizados. Además, incluye las medidas básicas de distancia que se emplean para construir (dibujar) el árbol de decisión de manera clara. Tal y como se explicará en la ampliación del Algoritmo Completo, estas distancias permiten distinguir cada nodo con su valor, cada rama, y asegurar que ninguna de esta información se superpone.

# 4. Ampliaciones

En este apartado se presentan las ampliaciones realizadas sobre la práctica básica, que incluye únicamente un nivel de recursividad en el algoritmo, y que no permite analizar su funcionamiento.

En este caso, se ha optado por implementar las dos ampliaciones propuestas: ejecutar todos los niveles de recursividad del algoritmo, y mostrar y comprobar el funcionamiento de este para los ejemplos de la tabla.

Estas ampliaciones sobre el enunciado básico se explican en los siguientes apartados, junto a una explicación de la refactorización del código que han provocado y cómo han sido implementadas.

## 4.1. Algoritmo completo

La primera ampliación consiste en ejecutar el algoritmo completo, esto es, todos los niveles de recursividad. Para implementarla se han realizado cambios en dos módulos principales del código, que se explican a continuación. Además, nótese que algunas de las refactorizaciones y funcionalidades añadidas en la segunda ampliación sólo tienen sentido cuando se ha implementado el algoritmo al completo.

### Clase ID3

Por supuesto, la clase principal que se ha visto modificada por esta ampliación ha sido la clase que implementa el algoritmo. Ahora bien, la práctica básica contenía prácticamente toda la funcionalidad del algoritmo, por lo que únicamente ha sido necesario añadir 3 líneas de código al original:

- La primera es el condicional, que indica que las dos siguientes líneas sólo deben ejecutarse si el usuario ha indicado que desea ejecutar el algoritmo completo.
- La segunda línea es la **llamada recursiva**, que debe hacerse con los parámetros calculados anteriormente (véase Clase ID3). Esta llamada repetirá el proceso para cada uno de los posibles valores (hijos) del atributo seleccionado hasta que llegue a los casos básicos, en los que todos los ejemplares son positivos o negativos.
- El último paso es añadir el hijo (la raíz del subárbol hijo, en realidad) calculado por la función recursiva al nodo raíz, asignándolo en el diccionario a la clave del valor del atributo que ha generado dicho hijo.

### Clase Tree

Ya se introdujo brevemente a la Clase Tree en apartados anteriores, pero se vio que la funcionalidad de esta clase tiene mayor interés cuando el algoritmo se ha ejecutado completamente, pues permite aplicar la recursividad. Recordemos que la función recursiva recibía los parámetros `node`, `pos` y `right`, para indicar el nodo raíz del subárbol, dónde debe dibujarse, y si el nodo es el hijo derecho de su padre. Veamos pues los pasos que sigue el algoritmo:

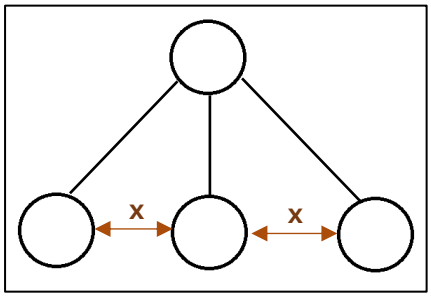
1. Caso base: Si el nodo es hoja, basta dibujarlo con el valor correspondiente: + o –, para “si” y “no”, respectivamente.
2. Caso recursivo: Si el nodo tiene hijos, entonces:
    a. Calcular la posición inicial desde donde se irán dibujando los hijos. Para esto se tiene en cuenta el número de hijos que tiene el nodo (= el número de valores que toma el atributo).
    b. Hacer para cada hijo:

i. Dibujar la rama que unirá al padre con el hijo, y que va desde el centro de uno hasta el centro del otro (hay que dibujar primero la línea para que quede por debajo de los nodos). La rama está formada por la línea, con el valor del atributo correspondiente al hijo que se va a dibujar.
ii. Llamada recursiva, primero se dibujan los hijos y después el padre

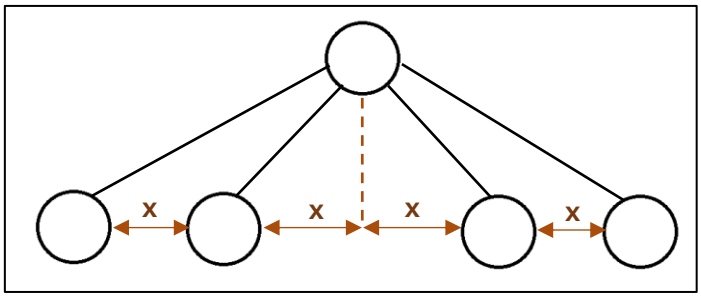
c. Dibujar el nodo raíz, con el valor del atributo escogido en ese nivel

NOTA: Todos los nodos están numerados por su atributo `num`, y los "títulos" de cada nodo se dibujarán a la izquierda o a la derecha en función de si son hijos derechos o izquierdos.

Además, una tarea imprescindible para dibujar de manera clara el árbol de decisión ha sido realizar el cálculo de las distancias entre los nodos. En primer lugar, se ha escogido una constante que determina la distancia vertical entre los niveles del árbol, igual para todos. Ahora bien, para las distancias horizontales, se ha tomado un valor inicial que se modifica en función del número de hijos. A mayor número de hijos, mayor distancia los separa. Después, para calcular sus posiciones, se ha seguido el siguiente esquema (ejemplo de 3 hijos y 4 hijos):




*Tres hijos*




*Cuatro hijos*

Siendo **x** la distancia entre los nodos ya ponderada en función del número de hijos. Así, se calcula la posición del nodo más a la izquierda y a partir de ahí basta sumar **x**, saltándose la posición del padre en caso de que el número de hijos sea par.

## 4.2. Funcionamiento del algoritmo

Esta ampliación, tal y como se ha interpretado, consiste en comprobar el funcionamiento del algoritmo. Para ello, se han considerado varias opciones o alternativas que ayudan a ilustrar el comportamiento de ID3 para los ejemplos dados.

En primer lugar, se comprobarán los cálculos matemáticos que realiza el algoritmo en su versión básica, para asegurar que las decisiones que toma son correctas. Este desarrollo manual del algoritmo se realizará en el apartado de Ejemplos.

Por otro lado, se ha optado por permitir al usuario visualizar el proceso y resultado del algoritmo en la propia aplicación, de manera más visual y práctica. Con este objetivo, se han creado tres clases adicionales que mostrarán los valores de los parámetros durante la ejecución del algoritmo y las reglas que se deducen de este, así como un ejemplo básico en el que el usuario puede introducir nuevos valores y obtener la decisión resultante.

### 4.2.1. Clase Info

La primera clase, denominada «Info», se encarga de mostrar en el panel derecho de la pantalla (ver Clase MainWindow) la información del algoritmo según se va ejecutando. Esta información consiste en los valores que toman las variables, así como notificaciones de cuándo se ha llegado al caso base.

La clase tiene un único atributo, que es el `frame` donde se imprime la información. Para cada "porción" de información se crea un nuevo frame interno que se añade al `frame` principal, de manera que cuando no caben más, el scrollbar permite desplazarse verticalmente. Ahora bien, ¿qué son estas "porciones" de información? ¿qué parámetros del algoritmo son relevantes para visualizarlos?

Se han considerado 3 elementos de interés:

- Lista {Atributo: Mérito} → Ya se ha visto que en un determinado nodo (estado) del algoritmo se calcula el mérito de los atributos disponibles y el atributo seleccionado es aquel que tenga un mérito menor. Por ello, para cada nodo intermedio se muestra la lista (diccionario) de cada atributo con su mérito, ordenados de menor a mayor mérito, para comprobar que, de hecho, se ha escogido el atributo *mejor*.
- Casos base → Los nodos hoja, que representan nodos base, se muestran en la zona de información indicando que todos los ejemplares son positivos/negativos, dependiendo del valor del nodo.
- Tabla de atributos → Cuando se expande un nodo y se crea una rama, el algoritmo está eliminando el atributo seleccionado de la lista de atributos. Esto equivale a eliminar una columna de la tabla de valores. Además, una rama concreta equivale a un valor que toma el atributo seleccionado, de manera que la tabla de valores del hijo está formada únicamente por las filas de la tabla original que tienen ese valor para el atributo *mejor*. En conclusión, este proceso da como resultado una tabla de datos que actúa como entrada para la llamada recursiva del hijo. Esta tabla es la que se muestra en la sección de información, para ver cómo van evolucionando los datos de entrada del algoritmo, y para dibujarla utiliza la Clase Table.

Para poder mostrar estos elementos, desde la Clase ID3 ha sido necesario añadir las correspondientes líneas de código a la función que implementa el algoritmo para mostrar la información según se va ejecutando.

### 4.2.2. Clase Rules

Otra de las clases que muestra el resultado del algoritmo es «Rules», que calcula e imprime las reglas que se deducen tras construir el árbol de decisión. La clase cuenta con los siguientes atributos:

- `frame` → Es el frame en el que se imprimen las reglas.
- `rules_text` → Es el texto que contiene las reglas, y que se va construyendo recursivamente.
- `num_rule` → Es una variable global útil para saber el número de regla que se está construyendo en cada momento.

Aparte de pintar las reglas en la interfaz, es necesaria una función previa que calcule las reglas a partir del árbol. Como hay que recorrerlo, de nuevo se implementa una función recursiva, que va construyendo un `string` que describe la regla desde la raíz hasta cada nodo hoja. De este modo, se implementa la recursión que sigue:

- Caso base: si el nodo es hoja, es porque ya se toma la decisión, así que terminamos y completamos la regla, añadiendo el número de regla (e incrementando la variable para la

siguiente), la regla parcial construida por las llamadas anteriores, y el final de la oración: "entonces sí/no se juega".

- Caso recursivo: si el nodo tiene hijos, entonces empezamos o continuamos la regla en función de si estamos el nodo raíz (para saberlo se utiliza un booleano) o no. Después, se añade el texto correspondiente al valor de la rama que se va a recorrer, y se hace la llamada recursiva con cada rama/hijo.

Nótese que el texto se calcula con un `switch` en función de cada valor de los atributos. Esta no es la manera idónea de implementarlo, pues si los datos de entrada fueran distintos habría que adaptar el código. Sin embargo, dado que los miembros del equipo conocen los valores de los atributos de antemano, se ha optado por simplificar esta funcionalidad e implementarla de forma estática en lugar de dinámica.

La clase que invoca al método que calcula y dibuja las reglas es la Clase ID3, de forma que, en el método que llama a la función recursiva que ejecuta el algoritmo, se invoca una vez haya terminado.

### 4.2.3. Clase Decision

Por último, se ha creado la clase «Decision», que permite introducir datos al usuario y ver qué decisión se toma siguiendo el árbol. Supone una manera más directa e interactiva de comprobar los resultados del algoritmo, y necesita los siguientes atributos:

- `frame` → Es el frame en el que se insertan los campos de entrada, el botón para ejecutar la funcionalidad y el campo resultado.
- `button` → Es el botón para ejecutar la funcionalidad. Hay que guardarlo como atributo porque estará activado o desactivado dependiendo de si se ha ejecutado o no el algoritmo completo (si no se ha ejecutado, no se puede usar esta funcionalidad).
- `selected_options` → Se trata de una lista que permite obtener para cada `OptionMenu` (explicado posteriormente) la opción seleccionada por el usuario en un momento concreto. Se inicializa con 4 posiciones porque la entrada contiene 4 atributos.
- `input` → Consiste en un diccionario que guarda para cada atributo (clave), el valor seleccionado por el usuario (valor), obteniéndolo de `selected_options`. Se utiliza para calcular la decisión.
- `output_label` → Es la etiqueta en la que se escribe el output. Se necesita guardarla como atributo para poder escribir el resultado en la función que calcula la decisión.
- `root` → Es el nodo raíz del árbol, necesario para poder recorrerlo y calcular la decisión.

Si bien es cierto que la clase cuenta con diversos métodos para dibujar la interfaz u obtener los valores de entrada seleccionados por el usuario, veamos la función principal, que calcula la decisión. Como siempre, para recorrer el árbol se utiliza la recursividad, y en este caso la recursión es bastante sencilla (sólo hay que bajar por las ramas que correspondan a los valores de entrada):

- Caso base: Si el nodo es hoja, entonces ya sabemos la decisión que se toma: sí o no.
- Caso recursivo: Si el nodo es interno, es decir, tiene hijos, entonces el nodo tendrá el correspondiente atributo *mejor*, con los distintos valores que toma. Por tanto, basta seleccionar la rama que tenga el valor seleccionado por el usuario para ese atributo (que se puede saber con el diccionario `input`) y hacer la llamada recursiva únicamente al nodo hijo de esa rama.

# 5. Ejemplos

En esta última sección se proponen dos ejemplos principales que ilustran el comportamiento del algoritmo. Primero se presenta la versión básica del algoritmo y, posteriormente, la versión completa junto a las funcionalidades extra que la acompañan.

- EJEMPLO BÁSICO

El primer ejemplo consiste en presentar brevemente la implementación básica del algoritmo, que sólo ejecuta un nivel de recursividad. Veamos la interfaz inicial de la aplicación:

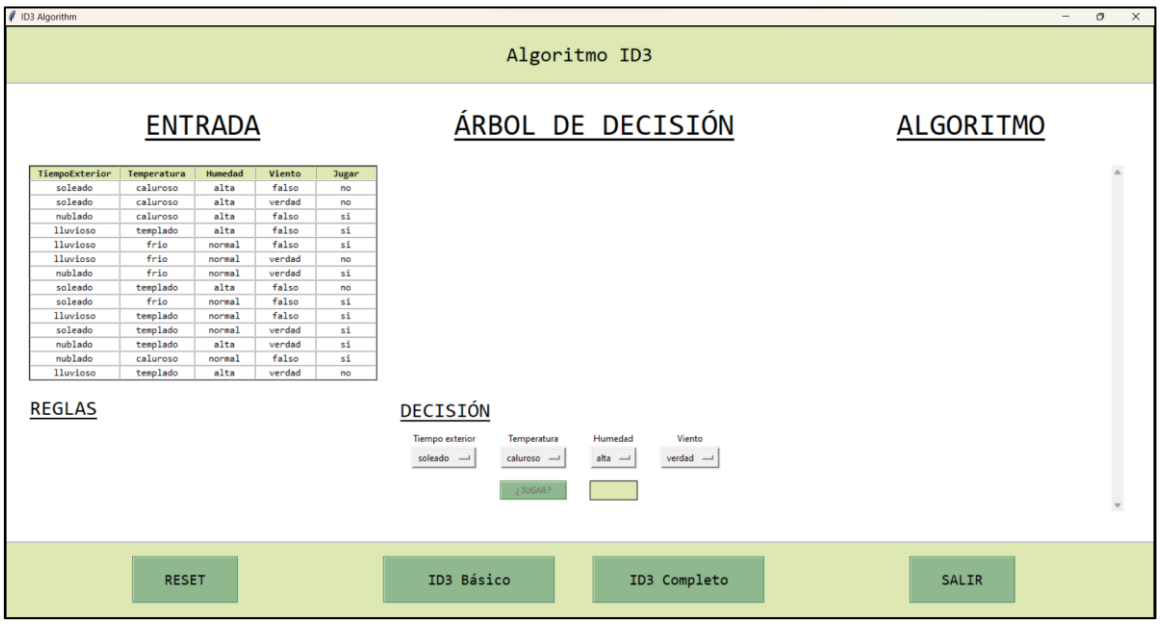



*Ilustración 2. Interfaz inicial*

En la zona izquierda se pueden ver los valores de entrada con los que trabaja el algoritmo. Así pues, si pulsamos en el botón "`ID3 Básico`" obtendremos el siguiente resultado:

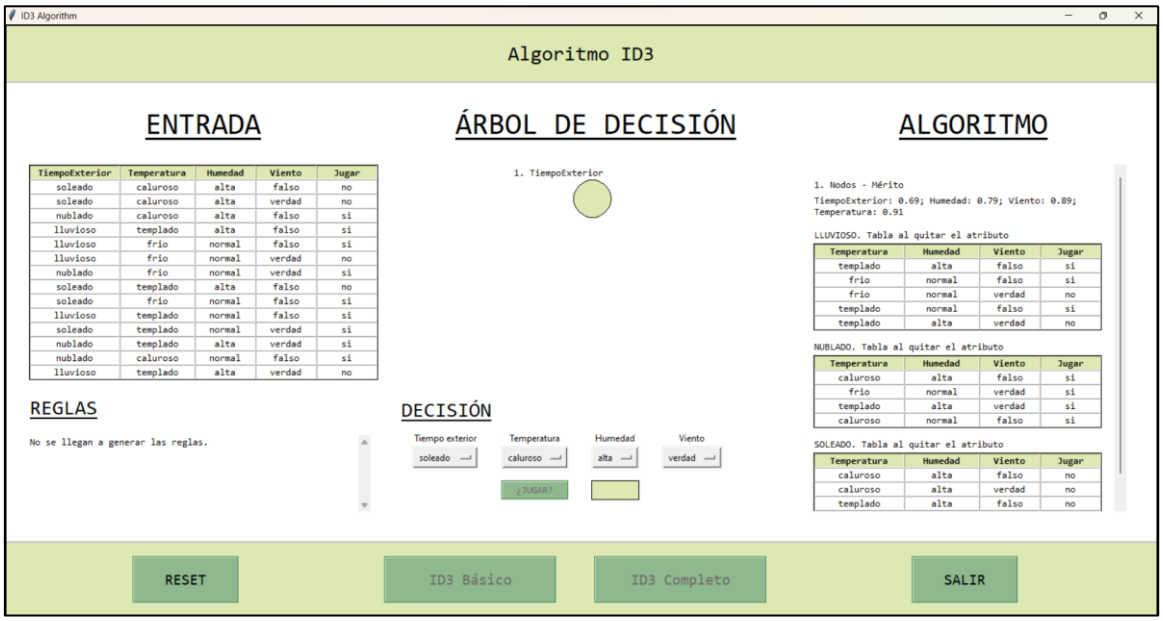
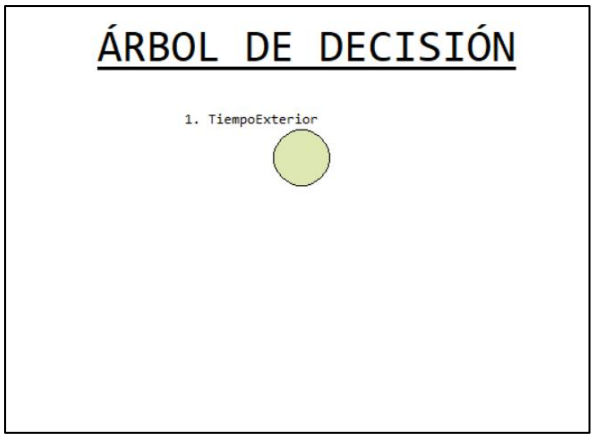
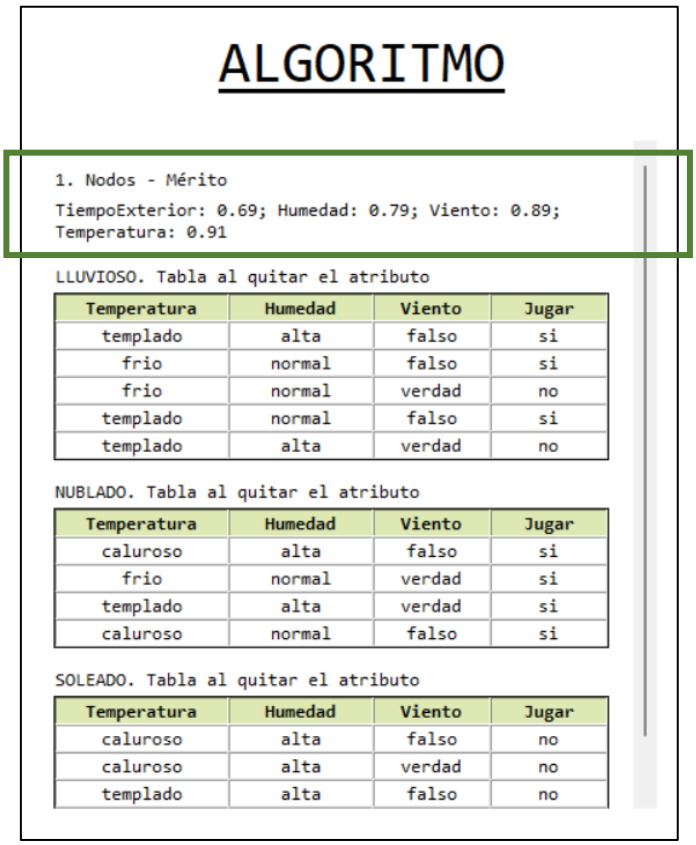



*Ilustración 3. Resultado ID3 básico*

Las secciones de reglas y decisión son irrelevantes, pues sólo tienen sentido cuando se ejecuta el algoritmo completo. Ahora bien, en el árbol de decisión se puede ver que sólo hay un nodo, correspondiente al estado en el que se elige el primer atributo y se preparan los valores para las llamadas recursivas:




Vemos pues, que el atributo seleccionado por el algoritmo es "TiempoExterior". De hecho, si observamos la zona de la información del algoritmo, podemos analizar este primer paso mejor:




La lista de cada atributo con su mérito permite comprobar que `TiempoExterior` es el de menor mérito y, por tanto, el que mejor discrimina. Es por ello que, para cada valor de `TiempoExterior` (en este caso, `lluvioso`, `nublado` y `soleado`), se elimina la columna de `TiempoExterior` de la tabla, y el algoritmo se queda únicamente con las filas en las que el atributo toma el valor `lluvioso`, `nublado` o `soleado` respectivamente. Este resultado es fácil comprobarlo observando la tabla de la zona izquierda de la pantalla. Veamos entonces que los méritos se calculan correctamente:

- $merito(TiempoExterior) = \frac{5}{14} * infor\left(\frac{2}{5},\frac{3}{5}\right) + \frac{4}{14} * infor\left(\frac{4}{4},\frac{0}{4}\right) + \frac{5}{14} * infor\left(\frac{3}{5},\frac{2}{5}\right) = 0{,}69$
- $merito(Humedad) = \frac{7}{14} * infor\left(\frac{3}{7},\frac{4}{7}\right) + \frac{7}{14} * infor\left(\frac{6}{7},\frac{1}{7}\right) = 0{,}79$
- $merito(Viento) = \frac{8}{14} * infor\left(\frac{6}{8},\frac{2}{8}\right) + \frac{6}{14} * infor\left(\frac{3}{6},\frac{3}{6}\right) = 0{,}89$
- $merito(Temperatura) = \frac{4}{14} * infor\left(\frac{2}{4},\frac{2}{4}\right) + \frac{6}{14} * infor\left(\frac{4}{6},\frac{2}{6}\right) + \frac{4}{14} * infor\left(\frac{3}{4},\frac{1}{4}\right) = 0{,}91$

Efectivamente, todos los resultados coinciden con los que muestra la aplicación, por lo que podemos dar paso al siguiente ejemplo.

- EJEMPLO COMPLETO

Ejecutaremos ahora el algoritmo en su totalidad con todos los niveles de recursividad, con su respectivo árbol de decisión y la información detallada de cada paso dado en la ejecución del algoritmo. Además, se incluyen las reglas resultantes de dicho árbol, así como un apartado en el que se muestra si es posible jugar o no, según los tipos de atributos que el usuario seleccione.

Este es el resultado al pulsar el botón de "`ID3 Completo`":

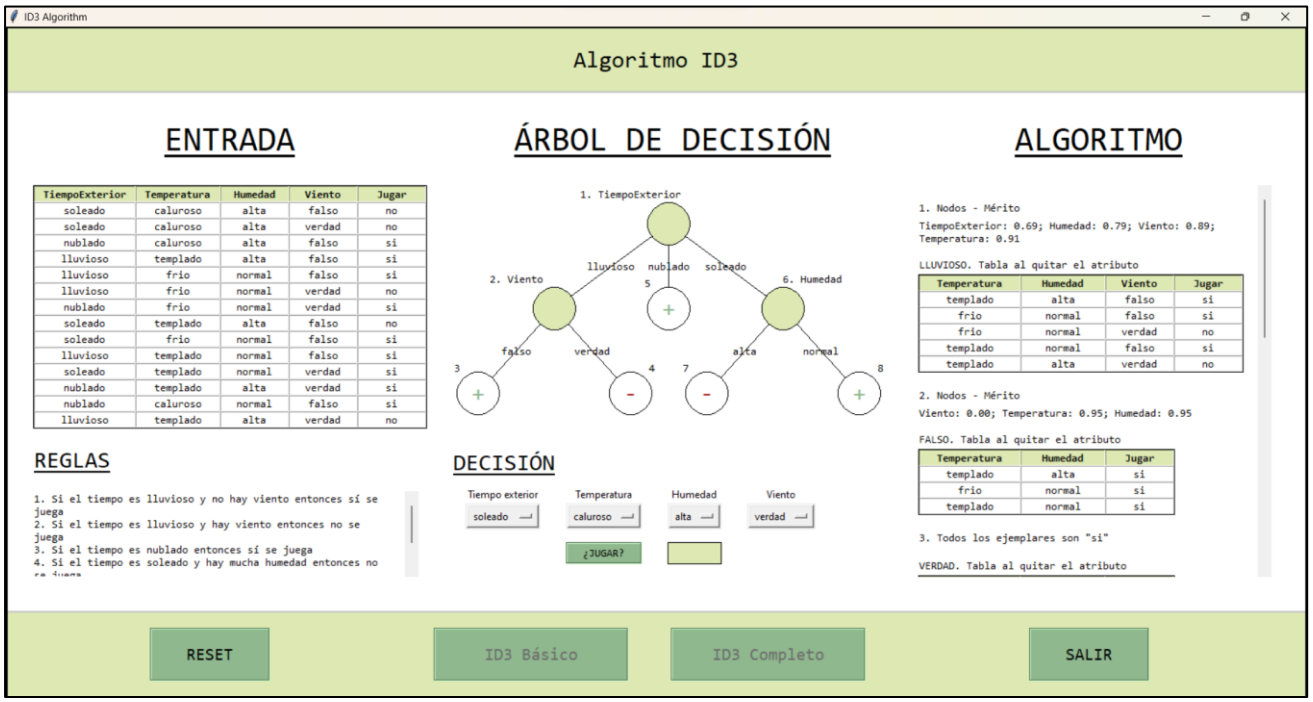
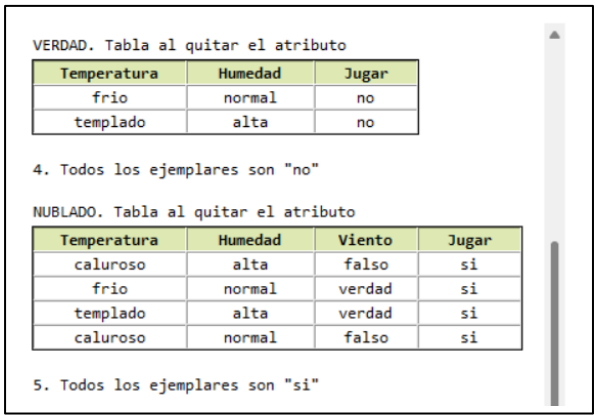



*Ilustración 4. Resultado ID3 completo*

En la sección "Árbol de decisión" se muestra el árbol resultante tras ejecutar el algoritmo. Como raíz tiene el atributo "`TiempoExterior`", ya que es el de menor mérito con respecto a los demás, como se ha visto en el ejemplo anterior. Este a su vez tiene tres hijos, correspondientes a cada uno de los valores que puede tomar.

El primer nodo que se recorre es el correspondiente al valor `lluvioso`, como se puede apreciar en la tabla resultante al quitar el atributo. Siguiendo por esa rama, se vuelven a calcular los méritos de los atributos (todos menos "`TiempoExterior`"), obteniendo en este caso que "`Viento`" es el que tiene menor mérito. Ocupa por tanto el nuevo nodo, que se expandirá empezando por el valor "`falso`". Ahora bien, en la tabla de la zona de la información se puede observar que todos los ejemplares son "`si`", llegando a un caso base y finalizando así esta rama.



VERDAD. Tabla al quitar el atributo

| Temperatura | Humedad | Jugar |
|---|---|---|
| frio | normal | no |
| templado | alta | no |

4. Todos los ejemplares son "no"

NUBLADO. Tabla al quitar el atributo

| Temperatura | Humedad | Viento | Jugar |
|---|---|---|---|
| caluroso | alta | falso | si |
| frio | normal | verdad | si |
| templado | alta | verdad | si |
| caluroso | normal | falso | si |

5. Todos los ejemplares son "si"

Para el hijo derecho del nodo correspondiente al valor "`verdad`", obtenemos la tabla siguiente, donde veremos que todos los ejemplares tienen el valor de la decisión "`no`":

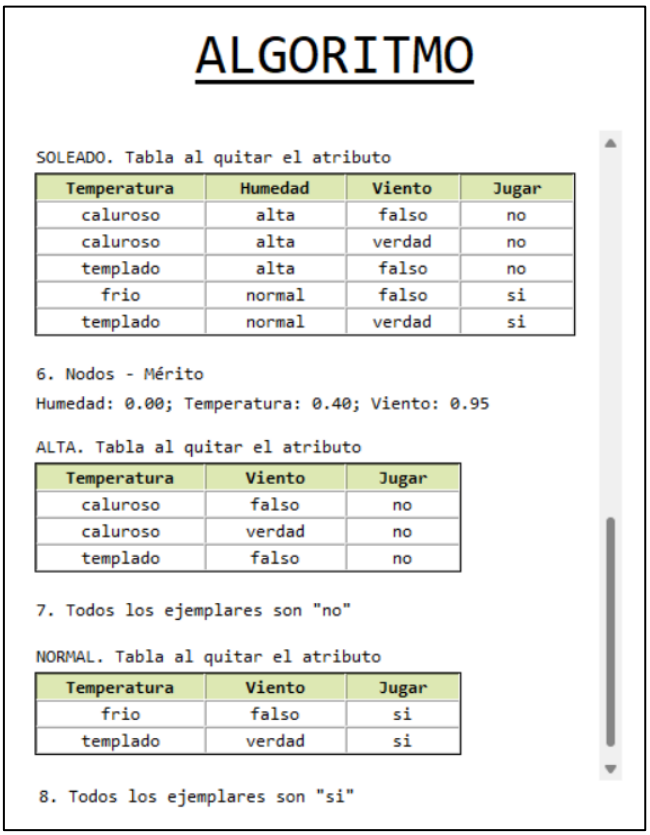
Pasamos entonces al segundo hijo del nodo raíz, corresponde al valor `nublado`, en el que todos sus ejemplares son "`si`", llegando de nuevo a un caso base y saltando al último hijo del nodo raíz correspondiente al atributo `soleado`.

En este caso, tal y como se ve en la siguiente captura, hay ejemplares tanto positivos como negativos, por lo que será necesario volver a calcular los méritos de los atributos restantes (`Humedad`, `Temperatura` y `Viento`).

Después de calcular los méritos, se asigna el atributo *mejor* al nuevo nodo que es, en este caso, "Humedad". Este tiene dos hijos: `alta` y `normal`, pero en ambos casos nos encontraremos en un caso base, en el que todos los ejemplares son "`no`" y "`si`", respectivamente. Así termina el algoritmo.

Damos paso entonces a la sección "Reglas", donde se muestran las 5 reglas deducidas tras la ejecución del algoritmo (una por cada nodo hoja):

1. Si el tiempo es lluvioso y no hay viento entonces sí se juega.
2. Si el tiempo es lluvioso y hay viento entonces no se juega.
3. Si el tiempo es nublado entonces si se juega.
4. Si el tiempo es soleado y hay mucha humedad entonces no se juega.
5. Si el tiempo es soleado y hay humedad normal entonces sí se juega.




Es inmediato ver cómo cada regla se obtiene a partir del recorrido de cada rama del árbol de decisión.

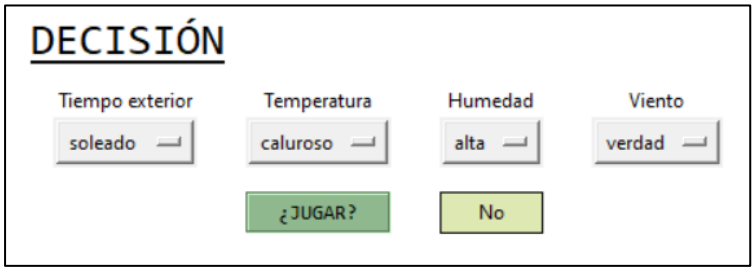
Por último, el apartado "Decisión", dispone de 4 campos de entrada de selección correspondiente a los atributos de la tabla de entrada, para que el usuario pueda introducir nuevos valores que desee y comprobar el efecto del algoritmo, viendo si se puede jugar o no dependiendo de los factores iniciales.

Se han propuesto, pues, dos ejemplos para ilustrar el comportamiento de esta funcionalidad. El primero consiste en introducir valores correspondientes a una de las filas de la tabla de entrada, para comprobar que obtenemos el resultado esperado (el de la tabla):

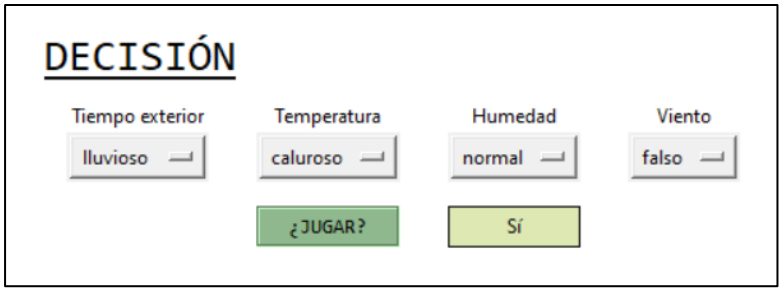
La fila de la tabla es:

| soleado | caluroso | alta | verdad | no |
|---|---|---|---|---|

Y, efectivamente, al introducir esos parámetros obtenemos:




Veamos entonces un ejemplo más interesante, en el que los valores introducidos no coinciden con ninguna fila de la tabla inicial. Por ejemplo:




Y es fácil comprobar que la decisión tomada es acertada, pues basta seguir el árbol de decisión con los valores en cuestión: