# ALGORITMO ID3 EN HASKELL

**Introducción:**

Hemos elegido el algoritmo ID3 para tratar de entender cómo programar un algoritmo de clasificación en un lenguaje declarativo como Haskell.

El primer código que tratamos de probar proviene de una tesis guardada en la colección de tesis electrónicas de la Universidad Autónoma Metropolitana de México (Unidad Iztapalapa). El enlace es <http://148.206.53.84/tesiuami/UAMI14542.pdf> (pag. 32). Al final fue descartado, ya que el código estaba incompleto.

En su lugar, utilizamos el código de la página oficial de haskell, el cual divide la creación del árbol de decisión en 3 módulos diferentes. Lo descargamos desde este enlace <https://hackage.haskell.org/package/DecisionTree> .

**Funcionamiento:**

El desarrollo de este algoritmo en haskell consta de 3 módulos diferentes: un **setup.lhs** encargado de inicializar haskell, un **DecisionTree.hs** con la estructura y las funciones para la creación de un árbol de decisión con ID3 y **test.hs** que contiene los datos de prueba para un árbol de decisión en haskell basado en la posibilidad de jugar o no un partido de tenis según el tiempo.

Nos vamos a centrar en el funcionamiento de las funciones y estructuras de **DecisionTree.hs**:

**DecisionTree** es el tipo del árbol de decisión, que puede ser una hoja con una etiqueta o un nodo que representa un tipo de atributo y que tiene hijos que son a su vez árboles de decisión.

**Attribute** representa uno de los atributos del árbol a crear (en el caso del test puede ser “Outlook”, “temperature”, “humidity” o “windy”) y los valores que puede tomar el atributo.

**Datum** es la estructura que representa una instancia del problema, la cual puede tener un nombre y que tiene una lista con los atributos y uno de los valores que puede tomar cada uno.

**Show** tiene la función de mostrar, bien una hoja con una etiqueta determinada, bien un nodo con esa etiqueta así como sus hijos (para ambos **showTree**), o bien el nombre de un atributo.

**Eq** compara los nombres de dos atributos.

**build** nos construye el árbol con las siguientes condiciones:

* Si la ganancia de información es 0, el árbol es una hoja con la etiqueta que más veces aparece en el conjunto de etiquetas.
* Si la ganancia de información es distinta de 0, será entonces un nodo con el mejor atributo (el atributo que tiene la ganancia obtenida) y una función que busca los hijos de ese nodo que tengan un atributo dado.
* Como información acerca de la función tenemos lo siguiente:
  + El mejor atributo **bAtt** y su ganancia de información **inf** los obtenemos mediante la función **bestAttribute,** que explicaremos más adelante, sobre el dataset y el conjunto de atributos.
  + Obtenemos una división del dataset en varias particiones en “**p**” en base al mejor atributo con la función **partition.**
  + Construimos nuevos subárboles recursivamente con la instrucción **(build atts)** y luego mediante **Map.map** mapeamos los subárboles según las particiones anteriormente obtenidas. Los resultados se guardan en **children**.
  + En **dominantLabel** obtenemos cual es la etiqueta de la clase con mayor número de ocurrencias, mapeando las etiquetas del dataset, agrupándolas, utilizando **Map.findMax** para encontrar la de mayor clave (que será la que más ocurrencias tenga) y quedándonos con la primera parte del par que será el valor de la etiqueta.
  + **Safelookup** es la función que se guarda en **child** y que busca el sub árbol hijo en base a un atributo **a**. Por eso, busca con **Map.lookup** en **children** un sub árbolcuyo nodo tenga como atributo **a**. Después, extraemos ese hijo con **fromJust**.

**getValue** obtiene el valor de un atributo **att** en un Datum **d**.

**label** obtiene la etiqueta de una hoja.

**decide** se encarga de decidir qué etiqueta corresponde a un Datum.

* Si llegamos a una hoja, se le asigna la etiqueta de esa hoja.
* Si estamos en un nodo, buscamos qué valor hay en el Datum para el atributo de ese nodo con **getValue** y decidimos sobre el hijo del nodo que corresponda a ese valor (usando la función almacenada en **child** anteriormente descrita).

**partition** divide el dataset según los posibles valores del atributo **att**.

**entropy** calcula la entropía del dataset para luego calcular la ganancia de información de un atributo.

**groupLabels** agrupa y cuenta las etiquetas del dataset.

**information** calcula la ganancia de información de un atributo respecto a la entropía del dataset.

**bestAttribute** devuelve el mejor atributo así como su ganancia de información usando la función **information.**

**groupWith** agrupa instancias del dataset utilizando **Map**.

El fichero **test.hs** importa todo el fichero **DecisionTree.hs** y utiliza la función **build** en la función **decideProp**.

**Conclusiones:**

Cuando probamos el algoritmo para el problema definido en test, vemos que nos construye el árbol correctamente. Si proporcionamos una instancia de la cual ya conocemos el resultado pero el programa no, vemos que la clasificación se efectúa correctamente.

El estudio y análisis de este código nos ha servido para comprender una implementación posible del algoritmo ID3 en un lenguaje declarativo como haskell. Comparándolo con otras implementaciones, consideramos este como el ejemplo más completo y didáctico de todos los encontrados hasta ahora.