**FUNDAMENTOS DE COMPUTACIÓN**
**TRABAJO ENTREGABLE**
**NOVIEMBRE 2017**

El objetivo de este trabajo es construir árboles de decisión a aprtir de ejemplos, utilizandoel algoritmo de Machine Learning ID3. El mismo tiene un puntaje de 10 puntos y debe ser realizado en grupos de hasta dos estudiantes.
La entrega se realizará por Aulas **antes del 29 de noviembre a las 23 hrs.**

Recomendamos utilizar las funciones del Preludio de Haskell vistas en clase para booleanos y listas, así como definir funciones auxiliares cuando sea necesario, para hacer más legible el código.
Se tendrá en cuenta el estilo de programación y el nivel de abstracción utilizado en la definición de las funciones.

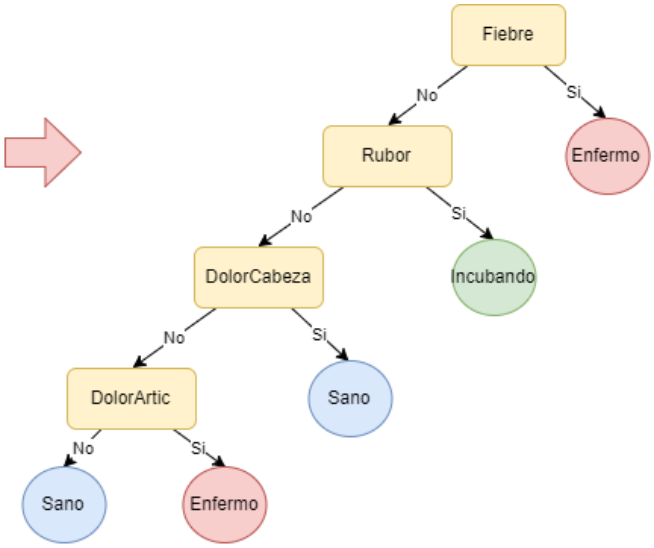
## 1. Descripción general

El algoritmo ID3 puede clasificarse como un algoritmo de aprendizaje inductivo, que se basa en el descubrimiento de patrones a partir de ejemplos.
El mismo funciona como sigue:
Supongamos que disponemos de un determinado número de ejemplos observados en el mundo real, {e1,. . . ,en}, donde cada uno está definido a partir de un conjunto de propiedades (atributos), y pertenece a una clase determinada.
La tarea del aprendizaje inductivo consiste en inducir, a partir de estos datos, un mecanismo que permita inferir las clases de cada uno de los ejemplos a partir únicamente de sus atributos.
Una vez realizado esto, será posible utilizar este mecanismo para deducir la clase a la que pertenecen nuevos ejemplos habiendo observado únicamente sus atributos.

Es evidente que obtener un mecanismo como este sería muy útil debido a que podríamos predecir comportamientos futuros a partir de los comportamientos observados en el pasado. Por ejemplo, a partir de los síntomas que tenemos observados en pacientes, y sabiendo ya si han desarrollado o no cierta enfermedad, podríamos extraer patrones que nos permitieran predecir si un paciente nuevo, aquejado de ciertos síntomas, desarrollará o no la misma enfermedad.

El algoritmo ID3 construye un *árbol de decisión* que permite determinar a qué clase pertenece un elemento determinado. Un árbol de decisión está formado por un conjunto de nodos de decisión (internos) y de nodos de clasificación (hojas):
Los nodos de decisión están asociados a los atributos y tienen 2 o más ramas que salen de él, cada una de ellas representando los posibles valores que puede tomar el atributo asociado. De alguna forma, un nodo de decisión es como una pregunta que se le hace al ejemplo analizado, y dependiendo de la respuesta que dé, el flujo tomará una de las ramas salientes.
Un nodo de clasificación está asociado a la clase que se quiere proporcionar, y nos devuelve la decisión del árbol con respecto al ejemplo de entrada.

| Diagnostico de gripe | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | Síntomas | | | | | |
| | | Fiebre | Moco | Rubor | DolorArtic | DolorCabeza | Diagnóstico |
| Pacientes | Diego | TRUE | TRUE | TRUE | TRUE | TRUE | Enfermo |
| | Juan | FALSE | FALSE | FALSE | FALSE | FALSE | Sano |
| | Manuela | FALSE | TRUE | TRUE | FALSE | TRUE | Incubando |
| | Nora | FALSE | TRUE | FALSE | FALSE | TRUE | Sano |
| | JPablo | TRUE | TRUE | FALSE | FALSE | TRUE | Enfermo |
| | Lucia | TRUE | FALSE | FALSE | FALSE | TRUE | Enfermo |
| | Valeria | TRUE | FALSE | FALSE | TRUE | FALSE | Enfermo |
| | Joaquin | FALSE | TRUE | FALSE | TRUE | FALSE | Enfermo |




El ejemplo anterior muestra la tabla de ejemplos que hemos observado respecto a los síntomas que tienen los docentes de Fundamentos y la posibilidad de tener gripe, estar incubándola o estar sanos. El árbol de la derecha muestra un posible mecanismo aprendido para poder tomar decisiones para esta tarea de clasificación.

## 2. El algoritmo ID3

El algoritmo ID3 fue presentado en 1979 por J. Ross Quinlan. Las entradas son un conjunto de ejemplos descritos mediante una serie de pares atributo-valor. Podemos pensar en ellos como una tabla en la que cada fila representa un ejemplo completo, y cada columna tiene el valor almacenado de cada uno de sus posibles atributos. Además, para cada ejemplo se tiene la información de a qué clase pertenece.
La salida es un árbol de decisión que separa a los ejemplos de acuerdo a las clases a las que pertenecen.

**Entropía de Shannon.** En cada paso del algoritmo se decide qué atributo permite separar mejor los ejemplos entre sí respecto a la clasificación final. Para ello, hace uso de la Teoría de la Información desarrollada por C. Shannon en 1948. Para entender cómo se mide la ganancia de información se introduce el concepto de Entropía, que de alguna forma mide el grado de incertidumbre de una muestra.
Supongamos que miramos cómo de homogéneos son los ejemplos de los que queremos aprender respecto a la clasificación: Una muestra completamente homogénea (es decir, en la que todos los ejemplos pertenecen a la misma clase) tiene incertidumbre mínima, es decir, no tenemos dudas de cuál es la clase a la que pertenece cualquiera de sus elementos (si elegimos al azar cualquier de ellos, sabremos qué resultado tendremos). En este caso, fijaremos la incertidumbre (entropía) en 0. En cambio, una muestra igualmente distribuida, es decir, que tiene el mismo número de ejemplos de cada posible clase, muestra una incertidumbre máxima, en el sentido de que es la peor situación para poder saber a priori cuál sería la clase a la que pertenece uno de sus ejemplos elegido al azar. Así pues, en este caso fijaremos la entropía en 1.

Para decidir qué atributo crea las ramas más homogéneas y, por tanto, proporciona una ganancia de información mayor, se procede del siguiente modo:

- Se divide el conjunto de datos en función de los diferentes atributos.

- Se calcula la entropía para cada atributo $a$, utilizando la siguiente fórmula:

$$E(a) = (-1) \times \sum_{c \in Clases} (p(c) \times log_2\ p(c))$$

  donde $p(c)$ es la proporción de elementos que pertenecen a la clase $c$ para los cuales el atributo $a$ se cumple. Si $p(c)$ es 0, el sumando $p(c) \times log_2\ p(c)$ se reemplaza por 0.
- Se elige el atributo de menor entropía, y se parten los datos] según ese atributo, que será la raíz del árbol de decisión.
- El algoritmo continúa recursivamente analizando cada uno de los subconjuntos así obtenidos, eliminando el atributo elegido del conjunto de atributos a considerar, como se explicará más adelante.

## 3. Datos

Como ejemplo utilizaremos los siguientes Datos, que se encuentran en el módulo `Datos`:

```
data Atributo = Fiebre | Moco | Rubor | DolorArtic | DolorCabeza
  deriving (Eq,Ord,Enum,Show)

data Clase = Enfermo | Sano | Incubando
  deriving (Eq,Ord,Enum,Show)
```

Además, se definirán los siguientes tipos:

```
type Nombre = String

type Registro = (Nombre,Clase,[Bool])
```

Cada `Registro` se corresponde con una fila de la tabla de ejemplos:

- El `Nombre` corresponde al nombre del ejemplo.
- La `Clase` corresponde a la clase a la que pertenece el `Nombre`.
- Para simplificar el problema utilizaremos atributos del tipo SI/NO, por lo que alcanzará con utilizar una lista de booleanos que indican si el atributo correspondiente se cumple o no para el `Nombre`.

En el módulo `Datos` también se encuentran los ejemplos para constuir el árbol de decisión para diagnosticar gripe, y son los siguientes:

```
ejs :: [Registro]
ejs = [ ("Diego",Enfermo,[True,True,True,True,True]),
        ("Juan",Sano,[False,False,False,False,False]),
        ("Manuela",Incubando,[False,True,True,False,True]),
        ("Nora",Sano,[False,True,False,False,True]),
        ("JPablo",Enfermo,[True,True,False,False,True]),
        ("Lucia",Enfermo,[True,False,False,False,True]),
        ("Valeria",Enfermo,[True,False,False,True,False]),
        ("Joaquin",Enfermo,[False,True,False,True,False])]
```

De esta lista de ejemplos sabemos que Diego está enfermo y tiene fiebre, moco, rubor, dolor articular y dolor de cabeza, pero Manuela está incubando y y no tiene fiebre, tiene moco, tiene rubor, no tiene dolor articular y tiene dolor de cabeza.

## 4. Funciones Auxiliares

El módulo `Auxiliares` será donde definamos nuestras funciones auxiliares. El mismo deberá importar al módulo `Datos`.

**1. Se pide:** Definir las siguientes funciones auxiliares sobre los Datos

Para generar las listas con los nombres de los atributos y de las clases del módulo de `Datos` utilizaremos las siguientes funciones:

```
atributos :: [Atributo]
atributos = [toEnum 0 ..]

clases :: [Clase]
clases = [toEnum 0 ..]
```

La función `fromEnum::Enum a => a -> Int` devuelve la posición de un constructor en la definición de un tipo enumerado (finito). Así, por ejemplo tenemos que, en el tipo `Atributo` se cumple que `fromEnum Fiebre = 0`, `fromEnum Moco = 1` .

(1) Evaluar las constantes `atributos` y `clases` para verificar sus valores.
Poner los resultados como comentarios en el código.

(2) `clase :: Nombre -> [Registro] -> Clase`, que devuelve la clase de un nombre una lista de Registros. Se asumirá que no hay nombres repetidos.
Ejemplo: `clase "Nora" ejs = Sano`

(3) `valorAtr :: Atributo -> Registro -> Bool`, que devuelve el valor de un atributo en un registro. Sugerencia: usar la función `fromEnum` para obtener la posición del atributo en la lista de booleanos del registro.
Ejemplos:
```
valorAtr Fiebre ("JPablo",Enfermo,[True,True,False,False,True]) = True
valorAtr Rubor (ejs!!3) = False
```

(4) `regsAtrB :: Atributo -> Bool -> [Registro] -> [Registro]`, tal que `regsAtrB a b rs` devuelve la lista de registros de `rs` cuyos elementos tienen el valor `b` para el atributo `a`.
Ejemplos:
```
regsAtrB Rubor True ejs =
[("Diego",Enfermo,[True,True,True,True,True]),
 ("Manuela",Incubando,[False,True,True,False,True])]
regsAtrB Moco False ejs =
[("Juan",Sano,[False,False,False,False,False]),
 ("Lucia",Enfermo,[True,False,False,False,True]),
 ("Valeria",Enfermo,[True,False,False,True,False])]
```

(5) `regsClase :: Clase -> [Registro] -> [Registro]`, tal que `regsClase c rs` devuelve la lista de registros de `rs` cuyos elementos pertenecen a la clase `c`.
Ejemplo: `regsClase Incubando ejs = ("Manuela",Incubando,[False,True,True,False,True])]`

(6) `prop :: Clase -> Atributo -> [Registro] -> Float`, tal que `prop c a rs` devuelve la proporción (número entre 0 y 1) de registros de `rs` que pertenecen a la clase `c` y para los cuales el atributo `a` es verdadero. O sea, la cantitdad de registros de `rs` que pertenecen a la clase `c` y para los cuales el atributo `a` es verdadero dividida la cantidad de registros de `rs` que pertenecen a la clase `c`
Sugerencia: utilizar la función `fromIntegral` para castear una expresión de tipo `Int` a tipo `Float` y poder utilizar la división `(/)` de números en punto flotante.
Ejemplos:
```
prop Sano Moco ejs = 0.5
prop Incubando Moco ejs = 1.0
prop Sano Fiebre ejs = 0.0
```

(7) `entrAtrClase :: Clase -> Atributo -> [Registro] -> Float`, tal que `entrAtrClase c a rs` calcula la entropía para el atributo `a` como `(prop c a rs * log(prop c a rs))`, si es que `prop c a rs` $\neq$ `0`, y en caso contrario devuelve `0`.
Ejemplos:
```
entrAtrClase Sano Moco ejs = -0.3465736
entrAtrClase Incubando Moco ejs = 0.0
entrAtrClase Sano Fiebre ejs = 0.0
entrAtrClase Enfermo DolorArtic ejs = -0.30649537
```

(8) `entrAtr :: [Atributo] -> [Registro] -> Atributo -> Float`, tal que `entrAtr ats rs a` devuelve la entropía de la lista de registros `rs` para el atributo `a` si éste pertenece a `ats`, o `1.0` si `a` no pertenece a `ats`.
La entropía para un atributo se calcula como: `-` $\sum_{c \in \texttt{clases}}$ `entrAtr ats rs a`, donde `entrAtr ats rs a` es la entropía del atributo `a` calculada con la función del punto anterior. Observar que se debe calcular esta sumatoria considerando todas las clases del conjunto de datos.
Ejemplos:
```
entrAtr [Fiebre, Rubor] ejs Rubor = 0.32188758
entrAtr [Fiebre, DolorCabeza, Moco] ejs Fiebre = 0.17851482
entrAtr [Fiebre, DolorCabeza, Moco] ejs Rubor = 1.0
```

(9) `minAtr :: [Atributo] -> [Registro] -> Atributo`, que devuelve el atributo de menor entropía en la lista de registros con respecto a la lista de atributos recibida. Si hay varios posibles, se devuelve el que aparece primero en la lista.
Sugerencia: definir una función auxiliar que calcule el mínimo de `(f x)` para los `x` de una lista dada.
Ejemplos:
```
minAtr [Rubor, DolorCabeza, Moco] ejs = Rubor
minAtr atributos ejs = Fiebre
```

(10) `partAtr :: Atributo -> [Registro] -> ([Registro], [Registro])`, parte una lista de registros en dos, la primera conteniendo aquellos para los cuales el atributo es falso y la segunda para los cuales es verdadero.

Ejemplo: `partAtr DolorArtic ejs =`

```
        ([("Juan",Sano,[False,False,False,False,False]),
          ("Manuela",Incubando,[False,True,True,False,True]),
          ("Nora",Sano,[False,True,False,False,True]),
          ("JPablo",Enfermo,[True,True,False,False,True]),
          ("Lucia",Enfermo,[True,False,False,False,True])] ,
        [("Diego",Enfermo,[True,True,True,True,True]),
          ("Valeria",Enfermo,[True,False,False,True,False]),
          ("Joaquin",Enfermo,[False,True,False,True,False])])
```

(11) `maxClase :: [Registro] -> Clase` que devuelve la clase más representativa de una lista de registros (o sea, la que aparece más veces).
Sugerencia: definir una función auxiliar que calcule las ocurrencias de cada elemento en una lista, y otra para después encontrar el elemento que aparece más veces en ésta.
Ejemplo: `maxClase ejs = Enfermo`

## 5. Árbol de decisión

El objetivo del algoritmo ID3 es construir un árbol de decisión a patir de un conjunto inicial de ejemplos.
Los árboles de decisión están formados por un conjunto de nodos de decisión (internos) y de nodos de clasificación (hojas):
Los nodos de internos se corresponden con los atributos y tienen en nuestro caso dos hijos, representando los posibles valores `True` y `False` que puede tomar el atributo para un dato dado. Las hojas se corresponden con las clases y retornan la decisión del árbol con respecto al ejemplo de entrada.

En el módulo `ID3` se definen los siguientes tipos:

```
data ArBin a b where {Hoja :: b -> ArBin a b ;
                      Nodo :: a -> (ArBin a b) -> (ArBin a b) -> (ArBin a b) }
  deriving Show

type ArDec = ArBin Atributo Clase
```

Para construir el árbol de decisión se procede del siguiente modo:

- Inicialmente se tiene una lista no vacía de registros (ejemplos) que contienen los datos de entrenamiento, y otra con los atributos a evaluar.
- Se elige el atributo de menor entropía de esta lista, que será la raíz del árbol de decisión, y se parten los registros según sean `True` o `False` para ese atributo.
- El algoritmo continúa recursivamente analizando cada uno de las sublistas de registros así obtenidas, eliminando el atributo elegido de la lista de atributos a considerar.

- La recursión termina si se da alguno de los siguientes casos:
  i. La lista de registros está vacía. Esto puede pasar cuando luego de una llamada recursiva una de las dos listas de registros quedó vacía porque en la lista anterior no se encontraron registros con un cierto valor de un atributo. En este caso, se construye una hoja con la clase más representativa en el padre, que será recibida como parámetro.
  ii. La lista de registros no está vacía, pero todos los elementos en la lista de registros pertenecen a la misma clase. En este caso, se construye una hoja con la clase correspondiente.
  iii. La lista de registros no está vacía, pero el conjunto de atributos es vacío y en la lista de registros hay elementos pertenecientes a más de una clase. En este caso, se construye una hoja con la clase más representativa de la lista de registros.

**2. Se pide:** Definir las siguientes funciones para construir el árbol de decisión:

(12) `id3 :: [Registro] -> [Atributo] -> Clase -> ArDec` que construye el árbol de decisión a partir de una lista de registros `rs`, una lista de atributos `ats` y una clase `c`. Esta última será la clase más representativa de la lista `rs` y será utilizada en las llamadas recursivas (observar que esta clase se utilizará en caso que la recursión termine en el caso (i) explicado anteriormente). Ejemplo: para los ejemplos y atributos del módulo de datos, deberá obtenerse el árbol del dibujo mostrado arriba.

(13) `arDec :: ArDec`, que es el árbol construido usando la función `id3` para el conjunto de ejemplos del módulo de Datos. La clase inicial para construir este árbol será la más representativa en los ejemplos.

Finalmente, utilizando el árbol construido, se pueden clasificar nuevos datos a partir de sus atributos.

**3. Se pide:** Definir las siguientes funciones para clasificar datos:

(14) `clasificar :: ArDec -> (Nombre,[Bool]) -> (Nombre,Clase)`, que devuelve la clase a la que pertenece un elemento a partir de sus atributos y un árbol dado.

(15) `clasificacion :: [(Nombre,Clase)]`, que es la clasificación obtenida para el conjunto de tests del módulo de Datos.
La misma debe dar el siguiente resultado: `[("Mauricio",Incubando),("Federico",Sano),("Igor",Sano),("Bruno",Enfermo),("Ignacio",Enfermo)]`

(16) Finalmente, entregamos dos juegos más con datos que deberán utilizarse para construir árboles de decisión de ejemplos más complejos. Los mismos se utilizarán simplemente cambiando el archivo `Datos` por estos nuevos. Los resultados de los árboles y tests están comentados en los archivos entregados.

## 6. Entregables

- El trabajo deberá realizarse en grupos de hasta dos estudiantes.
- Se pueden utilizar las funciones del Preludio de Haskell definidas para booleanos y listas. Cualquier otra función auxiliar que se necesite utilizar aparte de las de Preludio debe ser definida y se debe explicar qué hace (en forma de comentario en el código).
- La entrega deberá realizarse por Aulas antes del **29/11/2017 a las 23:00 hs.**
- Deberán subirse a Aulas dos archivo Haskell (`Auxiliares.hs` e `ID3.hs`) con el código fuente de la solución. El archivo debe incluir los nombres y números de estudiantes como comentarios al principio del mismo.
- En Aulas se encuentran estos archivos con las funciones que deben implementarse. Para facilitar la corrección deberá usarse el mismo como template.
- **No se corregirán archivos que no compilen, por lo que recomendamos comentar el código que no compile y dejar como `undefined` las funciones no implementadas.**

**Defensas (a confirmar):**

- **Grupos nocturnos: Jueves 30/11 a las 17.30 hs**
- **Grupos matutinos: Viernes 1/12 a las 10.00 hs**

**Referencias.**

Fernando Sancho Caparrini. Aprendizaje Inductivo: Árboles de Decisión.
(http://www.cs.us.es/∼fsancho/?e=104).

## 7. Anexo: algunas funciones útiles del Preludio de Haskell

```
elem :: Eq a => a -> [a] -> Bool

(!!) :: [a] -> Int -> a

map :: (a->b) -> [a] -> [b]

filter :: (a->Bool) -> [a] -> [a]

lenght :: [a] -> Int

sum :: [Int] -> Int

maximum :: Ord a => [a] -> a
```