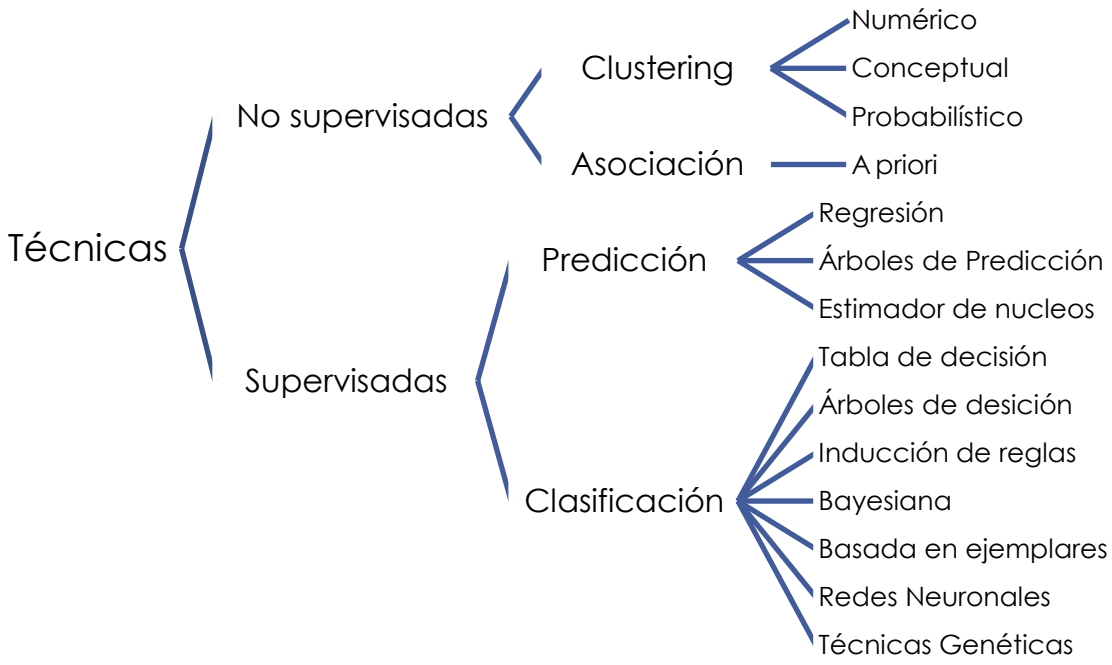
TEMA 2: Técnicas de minería de datos

2.1. Introducción

En la actualidad, existen diversos campos que se enfrentan al problema de disponer de bases de datos con un volumen de información almacenada que no puede ser analizada mediante técnicas estadísticas tradicionales. La extracción de conocimiento de los datos puede resultar de gran interés en diversos ámbitos, tanto para llegar a un conocimiento profundo de la estructura de los datos, como para servir de base en la toma de decisiones. Por otra parte, estas bases de datos se encuentran, en muchas ocasiones, en un proceso de crecimiento continuo, lo cual hace más patente aún la necesidad del uso de técnicas de análisis automatizadas.

Las técnicas de minería de datos comprenden una serie de algoritmos que son capaces de obtener relaciones entre distintos atributos. Pretenden obtener patrones o modelos a partir de los datos recopilados. En este sentido, los datos recogen un conjunto de hechos (una base de datos) y los patrones son expresiones que describen un subconjunto de los datos (un modelo aplicable a ese subconjunto).

Las técnicas de minería de Datos se clasifican en dos grandes categorías: supervisadas o predictivas y no supervisadas o descriptivas.



Cada técnica proporciona un enfoque para extraer la información de los datos y, en general, puede implementarse por varios algoritmos. Cada algoritmo representa la forma de desarrollar en la práctica una determinada técnica paso a paso. De esta forma, será necesario establecer en cada caso particular cual es la técnica más apropiada.

El aprendizaje no supervisado aborda el aprendizaje sin supervisión. En este caso, se trata de ordenar los ejemplos en una jerarquía según las regularidades en la distribución de los pares atributo-valor sin la guía del atributo especial clase.

En el caso del aprendizaje supervisado existe un atributo especial normalmente denominado clase, presente en todos los ejemplos que especifica si el ejemplo pertenece o no a un cierto concepto, que será el objetivo del aprendizaje. Mediante una generalización del papel del atributo clase, cualquier atributo puede desempeñar ese papel, convirtiéndose la clasificación de los ejemplos según el atributo seleccionado en el objeto del aprendizaje.

Supongamos que partimos de un conjunto de ejemplos denominados de entrenamiento de un cierto dominio D, el objetivo del aprendizaje supervisado es construir criterios para determinar el valor del atributo clase en un ejemplo cualquiera del dominio. Estos criterios están basados en los valores de uno o varios de los otros pares (atributo, valor) que intervienen en la definición de los ejemplos.

En esta asignatura nos centraremos en la exposición del último grupo de técnicas que aparecen en la tabla. En concreto, estudiaremos parte de las técnicas de clasificación incluidas en el apartado de aprendizaje supervisado.

2.2. La clasificación

Aprender como clasificar objetos a una de las categorías o clases previamente establecidas, es una característica de la inteligencia de máximo interés para investigadores, dado que la habilidad de realizar una clasificación y de aprender a clasificar otorga el poder de tomar decisiones.

La clasificación es el proceso de dividir un conjunto de datos en grupos mutuamente excluyentes, de tal forma que cada miembro de un grupo este lo más cerca posible de otros y grupos diferentes estén lo más lejos posible de otros, donde la distancia se mide con respecto a las variables especificadas como objeto de predicción.

La tabla (2.1) representa un ejemplo sencillo de clasificación consistente en, a partir de los atributos que modelan el tiempo (vista, temperatura, humedad y viento), determinar si se puede o no jugar al tenis.

El ejemplo empleado tiene dos atributos, temperatura y humedad, que pueden emplearse como simbólicos o numéricos. Entre paréntesis se presentan sus valores numéricos.

Ejemplo	Vista	Temperatura	Humedad	Viento	Jugar
1	Soleado	Alta (85)	Alta (85)	No	No
2	Soleado	Alta(80)	Alta (90)	Si	No
3	Nublado	Alta (83)	Alta (86)	No	Si
4	Lluvioso	Media (70)	Alta (96)	No	Si
5	Lluvioso	Baja (68)	Normal (80)	No	Si
6	Lluvioso	Baja (65)	Normal (70)	Si	No
7	Nublado	Baja (64)	Normal (65)	Si	Si
8	Soleado	Media (72)	Alta (95)	No	No
9	Soleado	Baja (69)	Normal (70)	No	Si
10	Lluvioso	Media (75)	Normal (80)	No	Si
11	Soleado	Media (75)	Normal (70)	Si	Si
12	Nublado	Media (72)	Alta (90)	Si	Si
13	Nublado	Alta (81)	Normal (75)	No	Si
14	Lluvioso	Media (71)	Alta (91)	Si	No

Tabla 2.1. Ejemplo de clasificación

En los siguientes apartados se presentan y explican las principales técnicas de clasificación. Además se mostraran ejemplos basados en la tabla anterior que permiten observar el funcionamiento del algoritmo.

2.2.1. Tabla de decisión

La tabla de decisión constituye la forma más simple y rudimentaria de representar la salida de un algoritmo de aprendizaje, que es justamente representarlo como la entrada.

Estos algoritmos consisten en seleccionar subconjuntos de atributos y calcular su precisión para predecir o clasificar los ejemplos. Una vez seleccionado el mejor de los subconjuntos, la tabla de decisión estará formada por los atributos seleccionados (más la clase), en la que se insertarán todos los ejemplos de entrenamiento únicamente con el subconjunto de atributos elegido. Si hay dos ejemplos con exactamente los mismos pares atributo-valor para todos los atributos del subconjunto, la clase que se elija será la media de los ejemplos (en el caso de una clase numérica) o la que mayor probabilidad de aparición tenga (en el caso de una clase simbólica).

Evaluar el comportamiento de los algoritmos de aprendizaje es un aspecto fundamental; no solo es importante para comparar algoritmos entre sí, sino que en muchos casos forma parte del propio algoritmo de aprendizaje. La forma más habitual de medir la eficiencia de un clasificador es la *precisión predictiva*. Cada vez que se presenta un nuevo caso a un clasificador, este debe tomar una decisión sobre la etiqueta que se le va a asignar. Considerando un error como una clasificación incorrecta de un ejemplo, se puede calcular fácilmente la tasa de error, o su complementaria, la tasa de acierto.

La precisión de un subconjunto, *S*, de atributos para todos los ejemplos de entrenamientos se calcula mediante la primera ecuación para el caso de que la clase sea simbólica o mediante la segunda ecuación si la clase es numérica.

$$\text{precisión (S)} = \frac{\text{ejemplos bien clasificados}}{\text{ejemplos totales}}$$

$$\text{precisión (S)} = \text{-RMSE} = -\sqrt{\frac{\sum_{i \in I}(y_i - \hat{y}_i)^2}{n}}$$

Donde en la segunda ecuación, RMSE es la raíz cuadrada del error cuadrático medio, n es el número de ejemplos totales, y_i el valor de la clase *i* y $\hat{y}_i$ el valor predicho por el modelo para el ejemplo *i*.

Como ejemplo de tabla de decisión, simplemente se puede utilizar la propia tabla 2.1., dado que si se comenzase a combinar atributos y a probar la precisión de dicha combinación, se obtendría como resultado que los cuatros atributos deben emplearse, con lo que la tabla de salida sería la misma. Esto no tiene por qué ser así, ya que en otros problemas no serán necesarios todos los atributos para generar la tabla de decisión, como ocurre en el ejemplo de la tabla 2.2 donde se dispone de un conjunto de entrenamiento en el que aparecen los atributos sexo y tipo (tipo de profesor) y la clase a determinar es si el tipo de contrato es o no de funcionario.

	Atributos		**Clase**
Ejemplo Nº	**Sexo**	**Tipo**	**Funcionario**
1	Hombre	Contratado	No
2	Mujer	Catedrático	Si
3	Hombre	Titular	Si
4	Mujer	Contratado	No
5	Hombre	Catedrático	Si
6	Mujer	Contratado	No
7	Hombre	Ayudante	No
8	Mujer	Titular	Si
9	Hombre	Contratado	No
10	Mujer	Ayudante	No
11	Hombre	Contratado	No

Tabla 2.2. Determinación del tipo de contrato

Si se toma como primer subconjunto el formado por el atributo sexo y se eliminan las repeticiones, resulta la tabla 2.3.

Ejemplo Nº	**Sexo**	**Funcionario**
1	Hombre	No
2	Mujer	Si
3	Hombre	Si
4	Mujer	No

Tabla 2.3. Subconjunto 1

Con lo que se pone de manifiesto que la probabilidad de clasificar bien es del 50%. Si por el contrario se elimina el atributo sexo, quedará la tabla 2.4.

Ejemplo Nº	Tipo	Funcionario
1	Contratado	No
2	Catedrático	Si
3	Titular	Si
4	Ayudante	No

Tabla 2.4. Subconjunto 2

Que tiene una precisión de aciertos del 100%. Por tanto, debería tomarse esta tabla y no la anterior como tabla de decisión.

2.2.2. Árboles de decisión

Los árboles de decisión constituyen una herramienta de aprendizaje supervisado bastante habitual en la minería de datos. Son una técnica muy versátil que puede utilizarse en áreas de muy diversa índole, diagnóstico médico, predicción meteorológica, seguridad vial…

En minería de datos un árbol de decisión es un modelo predictivo que puede ser utilizado para tareas de clasificación o para tareas de regresión, según la naturaleza de la variable de la clase: discreta (árbol de decisión) o continua (árbol de regresión).

Un árbol de decisión puede utilizarse para el análisis de datos con uno o varios de los siguientes fines:

- Descripción: reducir una gran cantidad de datos transformándolos en una forma más compacta que preserva las características esenciales y proporciona un resumen preciso.
- Clasificación: descubrir si los datos contienen clases de objetos bien diferenciadas que puedan ser interpretadas de manera significativa en el contexto de una teoría sustantiva.
- Generalización: descubrir una relación entre variables independientes y dependientes que sea útil para predecir el valor de la variable dependiente en el futuro.

Un árbol de decisión es una forma de representar el conocimiento obtenido en el proceso de aprendizaje inductivo. Puede verse como un conjunto de condiciones organizadas en una estructura jerárquica en forma de árbol, formada por diferentes nodos que se conectan con arcos (o ramas) dirigidos, diferenciándose:

- Nodo raíz: Es el nodo inicial, sólo tiene ramas salientes y en él queda recogida la totalidad de la población o datos de estudio.
- Nodos intermedios (hijos): Poseen ramas entrantes que provienen de los nodos padre y ramas salientes que apuntan a los nodos hijo. Cada uno de

estos nodos contiene una pregunta sobre un atributo concreto, que es una unidad que evalúa una función de decisión para determinar cuál es el próximo nodo hijo por el cual la rama se divide (existe un nodo hijo por cada posible respuesta).

- Nodo terminal (hoja): Representan la partición final, sólo tienen ramas entrantes y se asocia con una etiqueta o valor que caracteriza a los datos que llegan al nodo. De este modo, cada nodo hoja se refiere a una decisión (etiquetada con una de las clases del problema)

Así, dentro del árbol, cada nodo representa una variable atributo, y cada rama representa un estado de esa variable. Normalmente, cada nodo terminal representa el valor esperado de la variable clase o variable en estudio según la información contenida en el conjunto de datos utilizado para construir el modelo. La clasificación de una nueva instancia se realiza en base a una serie de preguntas sobre los valores de sus atributos, empezando por el nodo raíz y siguiendo el camino determinado por las respuestas a las preguntas de los nodos intermedios, hasta llegar a un nodo hoja. La etiqueta asignada a esta hoja es la que se asignará a la instancia a clasificar.

Construcción de árboles de decisión

Los árboles de decisión se construyen recursivamente siguiendo una estrategia descendente, desde los conceptos generales hasta los ejemplos particulares. Por ello a la familia de algoritmos de construcción de árboles de decisión se la conoce como "*Top-Down induction of decisión tres, TDIDT*".

La generación de la estructura de un árbol se fundamenta en el principio de "divide y vencerás". A partir del conjunto completo de datos (que forma el nodo raíz) y dado un criterio de partición determinado, se divide el conjunto en subconjuntos cada vez más pequeños (dando lugar a los nodos intermedios del árbol). Recursivamente se va dividiendo cada uno de los subconjuntos creados hasta que todos ellos son puros (cuando los casos del nodo son de una misma clase) o hasta que su "pureza" no puede incrementarse; se forman así los nodos hoja del árbol. Y finalmente, a los nodos hoja se les asigna una etiqueta o valor determinado de la variable clase.

Si no se establece ningún límite, la construcción del árbol se detiene cuando el nodo es puro. Sin embargo, este criterio puede dar lugar a un sobreajuste de los datos, que reduce la aplicabilidad del modelo de clasificación aprendido. Hace que el modelo construido sea muy específico, poco general, y por tanto malo para otros conjuntos de datos. Pero además, si los datos contienen ruido (errores en los atributos o en las clases), el modelo intentará ajustarse a los errores, perjudicando el comportamiento global del modelo aprendido. Para evitar el sobreajuste existen numerosas estrategias, tales como reglas de parada y los métodos de poda.

El objetivo habitual al construir un árbol de decisión es alcanzar el máximo grado de pureza en los nodos usando el menor número de particiones posibles, así el árbol resultante debe ser pequeño y el número de instancias en cada subconjunto grande.

Lo deseable es que la complejidad del árbol, que puede ser medida según diferentes métricas, sea la menor posible para lograr la máxima comprensión del modelo. Para controlar la complejidad se pueden utilizar determinadas reglas de parada y métodos

de poda. Además, hay que tener en cuenta que la complejidad del árbol tiene un efecto crucial en la precisión del método.

Criterios de partición

Los criterios de partición o ramificación se basan generalmente en medidas de la impureza del nodo. Y la bondad de la partición se mide como el decrecimiento de la impureza que se consigue con ella. Por tanto, la mayoría de los criterios de partición tratan de maximizar la bondad de la partición, lo cual equivale a minimizar la bondad de la impureza del árbol generado por la partición.

Cada uno de los posibles algoritmos que pueden utilizarse para la construcción de un árbol de decisión tiene asociado un criterio de partición. Algunos ejemplos son el índice de Gini, la ganancia de información o la razón de ganancia.

Reglas de parada.

Las reglas de parada tratan de predecir si conviene seguir construyendo el árbol por una determinada rama o no. En general, la fase de crecimiento del árbol continúa hasta que un criterio de parada se activa. Las condiciones más comúnmente utilizadas como criterios de parada son las siguientes:

- Pureza del nodo. Cuando un nodo solamente contiene casos de una misma clase, el proceso de construcción del árbol finaliza, ya que el nodo es puro. Sin embargo, también puede utilizarse un umbral de pureza para detener la ramificación, cuando ésta no suponga una disminución significativa de la pureza del nodo (según alguna medida estadística de pureza). El grado de pureza está relacionado con el de información sobre la variable en estudio.
- Cota de profundidad. Se puede establecer de antemano una determinada cota de profundidad para no construir árboles excesivamente complejos, es decir, excesivamente grandes. Así, cuando un nodo se halle a más de cierta profundidad, se detiene el proceso de generación del árbol.
- Umbral de soporte. Cuando hay un nodo con menos de N casos, también se puede detener el proceso de construcción del árbol, ya que no se considera fiable una clasificación avalada por menos de un número determinado de casos (menos de ese número de casos se consideran insuficientes para estimar probabilidades adecuadamente)

Las reglas de parada también se conocen como reglas de pre-poda porque reducen la complejidad del árbol durante su construcción. Son establecidas a priori por el propio investigador en función de investigaciones anteriores.

Métodos de poda

En general, el método recursivo de construcción de árboles divide el conjunto de casos hasta que se encuentra un nodo puro o no se puede seguir ramificando el árbol, pudiendo producirse un sobreajuste de los datos. La poda permite realizar una simplificación del árbol que permite mejorar tanto la precisión del método como la capacidad predictiva, además, evita el sobreajuste. Los métodos de poda dependen del algoritmo empleado, siendo los más comunes:

- Poda por estimación del error.
- Poda por coste-complejidad
- Poda pesimista

Reglas de decisión obtenidas de árboles de decisión

Una de las principales características de los árboles de decisión es que su estructura puede transformarse en conocimiento directo en forma de reglas de decisión. Las reglas de decisión que se obtienen de árboles de decisión se caracterizan porque el consecuente es una caso concreto de la variable de estudio; por tanto, son del tipo "SI (condición) → ENTONCES (clase)". Las reglas de decisión así obtenidas equivalen completamente al árbol original. Este hecho facilita la comprensión del modelo, y es aún más importante cuando los árboles son grandes, ya que al aumentar el tamaño disminuye su inteligibilidad.

En un árbol, cada regla se obtiene siguiendo el camino desde el nodo raíz a cada nodo hoja del árbol. De modo que desde el nodo raíz se deriva un conjunto de reglas cuyo antecedente es una conjunción de literales relativos a los valores de los atributos situados en los nodos intermedios del árbol, y cuyo consecuente es la decisión a la que hace referencia la hoja del árbol (la clasificación realizada).

Ventajas y desventajas de los árboles de decisión

Las principales ventajas de los árboles de decisión son:

- Son auto-explicativos, y cuando su tamaño no es muy grande son muy fáciles de interpretar.
- Permiten la extracción del conocimiento de un modo comprensible, en forma de reglas de decisión del tipo "SI-ENTONCES". Y estas reglas pueden ser usadas para descubrir determinados patrones de comportamiento que ocurren dentro de un conjunto de datos para una variable objeto de estudio.
- Los árboles de decisión son normalmente un método no paramétrico por lo que no establecen hipótesis sobre su distribución espacial ni de la estructura del clasificador.
- Pueden trabajar con un gran número de variables predictivas sin problemas de multicolinealidad.
- Son flexibles para trabajar con atributos tanto numéricos como nominales.
- Adaptabilidad en el pre-procesamiento de la base de datos, la cual puede contener errores o valores perdidos.
- Alto rendimiento predictivo con relativo esfuerzo computacional. Pueden trabajar con grandes bases de datos y descubrir fácilmente complejas interacciones entre los datos.

Y las principales desventajas son:

- El exceso de sensibilidad sobre el conjunto de entrenamiento.

- La presencia de atributos irrelevantes y el ruido, que pueden provocar que el modelo aprendido sea especialmente inestable, de modo que un pequeño cambio en los datos puede cambiar toda la estructura del árbol.
- Los árboles de decisión no permiten realizar análisis de elasticidades o de sensibilidades, los cuales permitirían examinar los efectos marginales de las variables dependientes sobre las variables objeto de estudio.
- Las reglas de decisión que se pueden obtener de un solo árbol de decisión están muy condicionadas por la variable que se usa como raíz, dando lugar a pérdida de información sobre la variable de estudio.

2.2.3. Algoritmos de construcción de árboles de decisión.

Existen numerosos algoritmos que permiten la construcción de árboles de decisión. Dentro de la familia de los algoritmos TDIDT se tienen, el algoritmo CHAID (Chi-squared Automatic Interaction Detection) implementado por Kass (1980); el método CART (Clasification ans regresión Trees) desarrollado por Breiman et al. (1984); el algoritmo ID3 (Quinlan, 1986) y sus posteriores evoluciones C4.5 (Quinlan 1993) y C5.0 (Quinlan 1997). Las principales diferencias entre los mismos radican en el criterio adoptado para realizar las particiones, en el tipo de variables que pueden manejar, así como en las restricciones que se imponen en el número de ramas en las que se puede dividir cada nodo, o el criterio de poda adoptado.

A continuación se describen en profundidad tres algoritmos de árboles de decisión, los sistemas ID3 y C4.5; y un árbol de decisión muy sencillo con un único nivel de decisión.

2.2.3.1. El algoritmo ID3

El algoritmo ID3 es un algoritmo simple y, sin embargo, potente, cuya misión es la elaboración de un árbol de decisión. El procedimiento para generar un árbol de decisión consiste, como ya se ha comentado, en seleccionar un atributo como raíz del árbol y crear una rama con cada uno de los valores de dicho atributo. Con cada rama resultante (nuevo nodo del árbol), se realiza el mismo proceso, esto es, se selecciona otro atributo y se genera una nueva rama para cada posible valor del atributo. Este procedimiento continúa hasta que los ejemplos se clasifiquen a través de uno de los caminos del árbol. El nodo final de cada camino será un nodo hoja, al que se le asignará la clase correspondiente. Así, el objetivo de los árboles de decisión es obtener reglas o relaciones que permitan clasificar a partir de los atributos.

En cada nodo del árbol de decisión se debe seleccionar un atributo para seguir dividiendo, y el criterio que se toma para elegirlo es: se selecciona el atributo que mejor separe (ordene) los ejemplos de acuerdo a las clases. Para ello se emplea la entropía, medida definida por Shannon (1948) y que representa una medida de cómo está ordenado el universo. La teoría de la información basada en la entropía calcula el número de bits (información, preguntas sobre atributos) que hace falta suministrar para conocer la clase a la que pertenece un ejemplo. Cuanto menor sea el valor de la entropía, menor será la incertidumbre y más útil será el atributo para la clasificación. La definición de entropía que da Shannon es: Dado un conjunto de eventos $A = \{A_1, A_2, \ldots, A_n\}$, con probabilidades

$\{p_1, p_2, \ldots, p_n\}$, la información en el conocimiento de un suceso A_i (bits) se define en la ecuación (1), mientras que la información media de A (bits) se muestra en la ecuación (2).

$$I(A_i) = log_2\left(\frac{1}{p_i}\right) = -log_2(p_i) \tag{1}$$

$$I(A) = \sum_{i=1}^{n} p_i I(A_i) = -\sum_{i=1}^{n} p_i log_2(p_i) \tag{2}$$

El criterio de partición utilizado en el algoritmo ID3, denominado ganancia de información está basado en la entropía. La idea es medir la ganancia de información asociada al elegir particionar por un determinado atributo (o variable). Se define como la diferencia de entropía del nodo actual y la suma ponderada de las entropías correspondientes a bifurcar por ese atributo. De esta forma, se define la ganancia de información al usar el atributo A_i como:

$$G(A_i) = I - I(A_i) \tag{3}$$

Siendo I la información antes de utilizar el atributo e $I(A_i)$ la información después de usarlo. Sus expresiones vienen dadas por las ecuaciones (4) y (5).

$$I = -\sum_{c=1}^{nc} \frac{n_c}{n} log_2\left(\frac{n_c}{n}\right) \tag{4}$$

$$I(A_i) = \sum_{j=1}^{nv(A_i)} \frac{n_{ij}}{n} I_{ij} \;;\; I_{ij} = -\sum_{k=1}^{nc} \frac{n_{ijk}}{n_{ij}} log_2\left(\frac{n_{ijk}}{n_{ij}}\right) \tag{5}$$

En estas ecuaciones *nc* representa el número de clases y n_c el número de ejemplares de la clase c, siendo n el número total de ejemplos. El número de valores del atributo A es $nv(a_i)$, n_{ij} el número de ejemplos con el valor j en A_i y n_{ijk} el número de ejemplos con valor j en A_i y que pertenecen a la clase k. Una vez explicada la heurística empleada para seleccionar el mejor atributo en un nodo del árbol de decisión, se muestra el algoritmo ID3:

Seleccionar el atributo A que maximice la ganancia $G(A_i)$

Crear un nodo para ese atributo con tantos sucesores como valores tenga.
Introducir los ejemplos en los sucesores según el valor que tenga el atributo A_i.
Por cada sucesor:
Si sólo hay ejemplos de una clase, C_k, entonces etiquetarlo con C_k.
Si no, llamar a ID3 con una tabla formada por los ejemplos de ese nodo, eliminado la columna del atributo A_i.

El algoritmo ID3 tiene cierta preferencia implícita a bifurcar los atributos nominales con muchas categorías, produciendo árboles que desprecian de forma prematura el resto de atributos, ya que llegan muy rápidamente a ramas con pocos casos. Otra de las características de este algoritmo es que no presenta ningún proceso de post-poda.

Ejemplo: A continuación se expone el proceso de generación del árbol de decisión para el problema planteado en la tabla 2.1.

El problema planteado en la tabla 2.1 consistía en decidir si jugar o no al tenis atendiendo a los atributos que modelan el tiempo (vista, temperatura, humedad y viento).

Para generar el árbol de decisión será necesario decidir en cada caso cual es el atributo que genera la rama. Para ello, comenzaremos seleccionando el atributo que maximice la ganancia.

En primer lugar, debemos calcular *I*,

$$I = -\frac{5}{14} log_2\left(\frac{5}{14}\right) - \frac{9}{14} log_2\left(\frac{9}{14}\right)$$

donde

nc = número de clases = 2 (Jugar SI o NO)
n_c = número de ejemplares en la clase (9 en SI, 5 en NO)
n = número de ejemplos = 14

Utilizando la ecuación (3) podemos calcular G para cada uno de los atributos vista, temperatura, humedad y viento.

$$G(Vista) = I - I(Vista)$$

$$I_{Vista,Soleado} = -\left(\frac{2}{5} log_2\left(\frac{2}{5}\right) + \frac{3}{5} log_2\left(\frac{3}{5}\right)\right) = 0.9709$$

$$I_{Vista,Nublado} = -\left(\frac{4}{4} log_2\left(\frac{4}{4}\right) + \frac{0}{4} log_2\left(\frac{0}{4}\right)\right) = 0$$

$$I_{Vista,Lluvioso} = -\left(\frac{3}{5} log_2\left(\frac{3}{5}\right) + \frac{2}{5} log_2\left(\frac{2}{5}\right)\right) = 0.9709$$

Por tanto,

$$I(Vista) = \frac{5}{14}0.9709 + \frac{4}{14}0 + \frac{5}{14}0.9709 = 0.6935$$

$$G(Vista) = 0.9403 - 0.6935 = 0.2468$$

Para estos cálculos se debe tener en cuenta que:

nv_{Vista}=número de valores de vista = 3
n_{ij} = número de ejemplos con el valor j (Soleado 5, Nublado 4, Lluvioso 5)
n_{ijk} = número de ejemplos con valor *j* en A_i y que pertenecen a la clase *k* (Soleado: SI 2, NO 3; Nublado: SI 4, NO 0; Lluvioso: SI 3, NO 2)

De forma equivalente podríamos calcular G para el resto de atributos y se obtiene que

$$G(Temperatura) = 0.0292$$
$$G(Humedad) = 0.1519$$
$$G(Viento) = 0.0481$$

Por tanto, en el primer nodo del árbol se decide que el mejor atributo para el nodo es Vista. A continuación, se generan nodos para cada valor del atributo y, en el caso de *Vista=Nublado* se llega a un nodo hoja, ya que todos los ejemplos de entrenamiento que llegan a dicho nodo son de clase SI. Sin embargo, para los otros dos casos se repite el proceso de elección con el resto de atributos y con los ejemplos de entrenamiento que se clasifican a través de ese nodo.

La figura 2.1. muestra el árbol de decisión que se generaría con el algoritmo ID3. Puede observarse que en la representación del árbol, todos los nodos hojas se han representado con forma de cuadrado.

Actividad 2.1. Realiza los cálculos para el resto de pasos del algoritmo hasta llegar a que todos los nodos sean de tipo hoja.

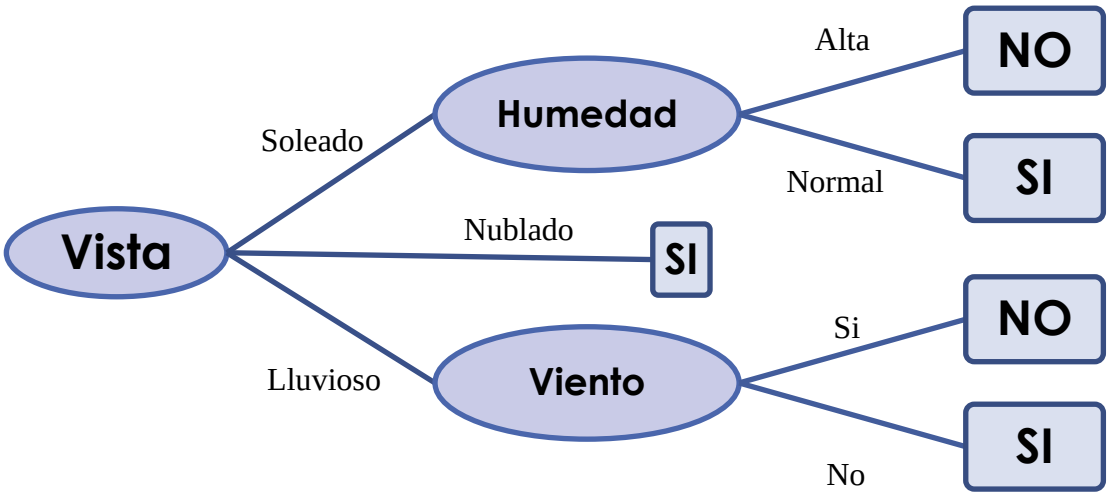


Figura 2.1. Ejemplo de clasificación con ID3

2.2.3.2. El algoritmo IC4.5

El algoritmo ID3 es capaz de tratar con atributos cuyos valores sean discretos o continuos. En el primer caso, el árbol de decisión generado tendrá tantas ramas como valores posibles tome el atributo. Si los valores del atributo son continuos, el ID3 no clasifica correctamente los ejemplos. El algoritmo C4.5 desarrollado por Quilan (1993) es una extensión del ID3 que surge para resolver algunas de sus limitaciones, en concreto se tiene que:

- Como criterio de partición utiliza la razón de ganancia de información. La maximización de la ganancia de información utilizada en el algoritmo ID3 puede dar lugar a errores en atributos nominales con muchas categorías, ya que se crean ramas para cada una de ellas recogiendo así un menor número de casos, y por tanto es más fácil que esas ramas tiendan a ser puras. El algoritmo C4.5 introduce una mejora para evitar este efecto, utilizando como criterio la razón de ganancia en lugar de la ganancia de información.
- Permite trabajar con atributos continuos y con valores perdidos.
- El algoritmo C4.5 incorpora una poda del árbol de decisión.
- Permite obtener reglas de clasificación.

Razón de Ganancia

El test basado en el criterio de maximizar la ganancia tiene como sesgo la elección de atributos con muchos valores. Esto es debido a que cuanto más fina sea la partición producida por los valores del atributo, normalmente, la incertidumbre o entropía en cada nuevo nodo será menor, y por lo tanto también será menor la media de la entropía a ese nivel. El algoritmo C4.5 modifica el criterio de selección del atributo empleando en lugar de la ganancia la razón de ganancia, cuya definición se muestra en la ecuación (6).

$$GR(A_i) = \frac{G(A_i)}{I(División\ A_i)} = \frac{G(A_i)}{-\sum_{j=1}^{nv(A_i)} \frac{n_{ij}}{n} log_2 \left(\frac{n_{ij}}{n}\right)} \quad (6)$$

Al término $I(División\ A_i)$ se le denomina información de ruptura. En esta medida cuando n_{ij} tiende a n, el denominador se hace 0. Por tanto, se ha de escoger el atributo que maximice el cociente siendo su ganancia, al menos, tan grande como la ganancia media de todas las alternativas analizadas.

Valores desconocidos

El sistema C4.5 admite ejemplos con atributos desconocidos tanto en el proceso de aprendizaje como en el de validación. Para calcular, durante el proceso de aprendizaje, la razón de ganancia de un atributo con valores desconocidos, se redefinen sus dos términos, la ganancia mediante la ecuación (7) y la información de ruptura mediante la ecuación (8).

$$G(A_i) = \frac{n_{ic}}{n}\ (I - I(A_i)) \quad (7)$$

$$I(División\ A_i) = -\left(\sum_{j=1}^{nv(A_i)} \frac{n_{ij}}{n} log_2 \left(\frac{n_{ij}}{n}\right)\right) - \frac{n_{id}}{n} log_2 \left(\frac{n_{id}}{n}\right) \quad (8)$$

En estas ecuaciones, n_{ic} es el número de ejemplos con el atributo i conocido, y n_{id} el número de ejemplo con valor desconocido en el mismo atributo. Además, para el cálculo de la entropía $I(Ai)$ se tendrán en cuenta únicamente los ejemplos en los que el atributo A_i tenga un valor definido.

No se toma el valor desconocido como significativo, sino que se supone una distribución probabilística del atributo de acuerdo con los valores de los ejemplos en la muestra de entrenamiento. Cuando se entrena, los casos con valores desconocidos se distribuyen con pesos de acuerdo a la frecuencia de aparición de cada posible valor del atributo en el resto de ejemplos de entrenamiento. El peso ω_{ij} con el que se distribuiría un ejemplo i desde un nodo etiquetado con el atributo A hacia el hijo con valor j en dicho atributo se calcula mediante la ecuación (9), en la que ω_{ij} es el peso del ejemplo i al llegar al nodo, esto es, antes de distribuirse, y $p(A=j)$ la suma de pesos de todos los ejemplos del nodo con valor j en el atributo A entre la suma total de pesos de todos los ejemplos del nodo.

$$\omega_{ij} = \omega_i\, p(A = j) = \omega_i\, \frac{\omega_{A=j}}{\omega} \quad (9)$$

En cuanto a la clasificación de un ejemplo de test, si se alcanza un nodo con un atributo que el ejemplo no tiene (desconocido), se distribuye el ejemplo (divide) en tantos casos como valores tenga el atributo, y se da un peso a cada resultado con el mismo criterio que en el caso del entrenamiento: la frecuencia de aparición de cada posible valor del atributo en los ejemplos de entrenamiento. El resultado de esta técnica es una clasificación con probabilidades, correspondientes a la distribución de ejemplos en cada nodo hoja.

Atributos continuos

El tratamiento que realiza C4.5 de los atributos continuos está basado en la ganancia de información, al igual que ocurre con los atributos discretos. Si un atributo continuo A_i presenta los valores ordenados $v_1, v_2, \ldots, v_n$, se comprueba cual de los valores $z_i = \frac{v_i + v_{i+1}}{2}$; $1 \leq j < n$, supone una ruptura del intervalo $[v_1, v_n]$ en dos subintervalos $[v_1, z_i]$ y $[z_j, v_n]$ con mayor ganancia de información. El atributo continuo, ahora con dos únicos valores posibles, entrará en competencia con el resto de los atributos disponibles para expandir el nodo.

Ejemplos ordenados

Valores	64	65	68	69	70	71	72	72	75	75	80	81	83	85
Clases	A	B	A	A	A	B	B	A	A	A	B	A	A	B

2 ejemplos por intervalo

Valores	64	65	68	69	70	71	72	72	75	75	80	81	83	85
Clases	A	B	A	A	A	B	B	A	A	A	B	A	A	B

Ejemplos adyacentes con la misma clase

Valores	64	65	68	69	70	71	72	72	75	75	80	81	83	85
Clases	A	B	A	A	A	B	B	A	A	A	B	A	A	B

Ejemplos adyacentes con el mismo valor

Valores	64	65	68	69	70	71	72	72	75	75	80	81	83	85
Clases	A	B	A	A	A	B	B	A	A	A	B	A	A	B

Int adyacentes con la misma clase mayoritaria

Valores	64	65	68	69	70	71	72	72	75	75	80	81	83	85
Clases	A	B	A	A	A	B	B	A	A	A	B	A	A	B

$$I = \sum_{c=1}^{nc} \frac{n_c}{n} log_2\left(\frac{n_c}{n}\right) = -\frac{5}{14} log_2\left(\frac{5}{14}\right) - \frac{9}{14} log_2\left(\frac{9}{14}\right) = 0.9403$$

$$I(A_{66.5}) = \sum_{j=1}^{2} \frac{n_j}{n} I_{A=j} = \frac{2}{14} I_{A \leq 66.5} + \frac{12}{14} I_{A>66.5} = 0.93$$

$$I_{A \leq 66.5} = -\frac{1}{2} log_2\left(\frac{1}{2}\right) - \frac{1}{2} log_2\left(\frac{1}{2}\right) = 1$$

$$I_{A>66.5} = -\frac{8}{12} log_2\left(\frac{8}{12}\right) - \frac{4}{12} log_2\left(\frac{4}{12}\right) = 0.9183$$

$$G(A_{66.5}) = I - I(A_{66.5}) = 0.0103$$

$$G(A_{77.5}) = I - I(A_{77.5}) = 0.0251$$

Para mejorar la eficiencia del algoritmo no se consideran todos los posibles puntos de corte, sino que se tienen en cuenta las siguientes reglas:

- Cada subintervalo debe tener un número mínimo de ejemplo (por ejemplo, 2)
- No se divide el intervalo si el siguiente ejemplo pertenece a la misma clase que el actual.
- No se divide el intervalo si el siguiente ejemplo tiene el mismo valor que el actual.
- Se unen subintervalos adyacentes si tienen la misma clase mayoritaria.

Como se ve en el ejemplo anterior, aplicando las reglas anteriores sólo es preciso probar dos puntos de corte (66.5 y 77.5), mientras que si no se empleara ninguna de las mejoras que se comentan previamente se deberían haber probado un total de trece puntos. En el ejemplo anterior, finalmente se tomaría como punto de ruptura el 77,5 dado que se obtiene una mejor ganancia. Una vez seleccionado el punto de corte, este atributo numérico competiría con el resto de atributos. Si bien aquí se ha empleado la ganancia, realmente se emplearía la razón de ganancia, pero no afecta a la elección del punto de corte. Es importante mencionar que este atributo no deja de estar disponible en niveles inferiores como en el caso de los discretos, aunque con sus valores restringidos al intervalo que domina el camino.

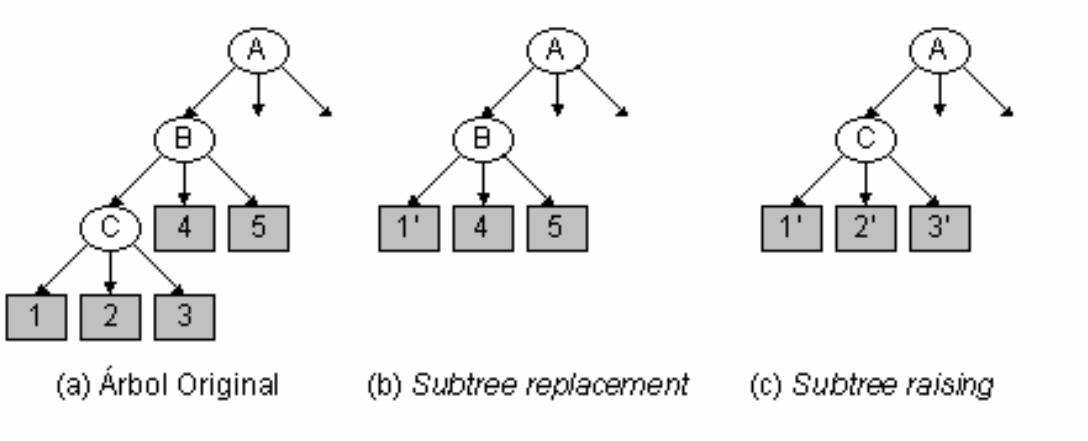
Actividad 2.2. Resuelve el problema anterior desarrollando los cálculos y explicando cada uno de los valores que aparecen en las expresiones para los diferentes cálculos.

Poda del árbol de decisión

El árbol de decisión ha sido construido a partir de un conjunto de ejemplos, por tanto, reflejará correctamente todo el grupo de casos. Sin embargo, como esos ejemplos pueden ser muy diferentes entre sí, el árbol resultante puede llegar a ser bastante complejo, con trayectorias largas y muy desiguales. Para facilitar la comprensión del árbol puede realizarse una poda del mismo. C4.5 efectúa la poda después de haber desarrollado el árbol completo (post-poda), a diferencia de otros sistemas que realizan la construcción del árbol y la poda a la vez (pre-poda); es decir, estiman la necesidad de seguir desarrollando un nodo aunque no posea el carácter de hoja. En C4.5 el proceso de podado comienza en los nodos hoja y recursivamente continúa hasta llegar al nodo raíz.

El algoritmo C4.5 considera dos operaciones de poda: reemplazo de sub-árbol por hoja (subtree replacement) y elevación de sub-árbol (subtree raising).

En la siguiente figura se muestra en lo que consiste cada tipo de poda.



En esta figura tenemos el árbol original antes del podado (a), y las dos posibles acciones de podado a realizar sobre el nodo interno C. En (b) se realiza subtree replacement, en cuyo caso el nodo C es reemplazado por uno de sus subárboles. Por último, en (c) se realiza subtree raising: El nodo B es sustituido por el subárbol con raíz C. En este último caso hay que tener en cuenta que habrá que reclasificar de nuevo los ejemplos a partir del nodo C. Además, subtree raising es muy costoso computacionalmente hablando, por lo que se suele restringir su uso al camino más largo a partir del nodo (hasta la hoja) que estamos podando.

Como se comentó anteriormente, el proceso de podado comienza en las hojas y continúa hacia la raíz pero, la cuestión es cómo decidir reemplazar un nodo interno por una hoja (replacement) o reemplazar un nodo interno por uno de sus nodos hijo (raising). Para ello se compara el error estimado de clasificación del árbol antes y después de una operación de poda, de modo que sólo se lleva a cabo cuando esta diferencia resulta favorable a la poda. Naturalmente, el problema está en que el árbol sin podar tendrá un error de entrenamiento cero o muy próximo a cero. Por tanto, sería muy optimista asumir que el error antes de la poda es el error de entrenamiento sin más. La solución más sencilla consistiría en retirar un pequeño número de instancias del conjunto de entrenamiento antes de construir el árbol, y hacer la estimación de los errores con dicho conjunto. Sin embargo, esta aproximación plantea un problema para aquellos conjuntos de datos que cuenten con pocas instancias de entrenamiento.

La técnica empleada en C4.5 consiste en estimar el error de clasificación basándose en los propios ejemplos de entrenamiento. Para ello, en el nodo donde queramos estimar el error de clasificación, se toma la clase mayoritaria de sus ejemplos como clase representante. Esto implica que habrá E errores de clasificación de un total de N ejemplos que se clasifican a través de dicho nodo. El error observado será $f=E/N$, siendo q la probabilidad de error de clasificación del nodo y $p=1-q$ la probabilidad de éxito. Se supone que la función f sigue una distribución binomial de parámetro q. Y lo que se desea obtener es el error e, que será la probabilidad del extremo superior con un intervalo $[f-z, f+z]$ de confianza c. Dado que se trata de una distribución binomial, se obtendrá e mediante las ecuaciones 10 y 11.

$$P\left[\frac{f-q}{q(1-q)/N} \leq z\right] = c \qquad (10)$$

$$e = \left(\frac{f + \frac{z^2}{2N} + z\sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \right) \quad (11)$$

Como factor *c* (factor de confianza) se suele emplear el 25%, dado que es el que mejores resultados suele dar y que corresponde a un z=0.69.

Obtención de Reglas de Clasificación

Cualquier árbol de decisión se puede convertir en reglas de clasificación, entendiendo como tal una estructura del tipo Si <Condición> Entonces <Clase>. El algoritmo de generación de reglas consiste básicamente en, por cada rama del árbol de decisión, las preguntas y sus valores estarán en la parte izquierda de las reglas y la etiqueta del nodo hoja correspondiente en la parte derecha (clasificación). Sin embargo, este procedimiento generaría un sistema de reglas con mayor complejidad de la necesaria. Por ello, el sistema C4.5 realiza un podado de las reglas obtenidas.

2.2.3.3 Decision Stump (Árbol de un solo nivel)

Por último, se expone un algoritmo más sencillo que genera un árbol de decisión de un único nivel. Se trata de un algoritmo, *decision stump*, que utiliza un único atributo para construir el árbol de decisión. La elección del único atributo que formará parte del árbol se realiza mediante la ganancia de información, y a pesar de su simplicidad, en algunos problemas puede llegar a conseguir resultados interesantes. No tiene opciones de configuración, pero la implementación es muy completa, dado que admite tanto atributos numéricos como simbólicos y clases de ambos tipos también. El árbol de decisión tendrá tres ramas: una de ellas será para el caso de que el atributo sea desconocido, y las otras dos serán para el caso de que el valor del atributo del ejemplo de test sea igual a un valor concreto del atributo o distinto a dicho valor, en caso de los atributos simbólicos, o que el valor del ejemplo de test sea mayor o menor a un determinado valor en el caso de atributos numéricos. En el caso de los atributos simbólicos se considera cada valor posible del mismo y se calcula la ganancia de información con el atributo igual al valor, distinto al valor y valores desconocidos del atributo. En el caso de atributos simbólicos se busca el mejor punto de ruptura, tal y como se vio en el sistema C4.5. Deben tenerse en cuenta cuatro posibles casos al calcular la ganancia de información: que sea un atributo simbólico y la clase sea simbólica o que la clase sea numérica, o que sea un atributo numérico y la clase sea simbólica o que la clase sea numérica. A continuación se comenta cada caso por separado.

Atributo Simbólico y Clase Simbólica

Se toma cada vez un valor v_x del atributo simbólico A_i como base y se consideran únicamente tres posibles ramas en la construcción del árbol: que el atributo A_i sea igual a v_x, que el atributo A_i sea distinto a v_x o que el valor del atributo A_i sea desconocido. Con ello, se calcula la entropía del atributo tomando como base el valor escogido tal y como se muestra en la ecuación 12, en la que el valor de *j* en el sumatorio va desde 1 a 3 porque los valores del atributo se restringen a tres: igual a v_x, distinto de v_x o valor desconocido. En cuanto a los parámetros, n_{ij} es el número de ejemplos con valor *j* en el atributo *i*, *n* el

número total de ejemplos y n_{ijk} el número de ejemplos con valor j en el atributo i y que pertenece a la clase k.

$$I(A_{iv_x}) = \frac{\sum_{j=1}^{3} n_{ij}\log(n_{ij}) - I_{ij}}{n};\ I_{ij} = -\sum_{k=1}^{nc} n_{ijk}\log(n_{ijk}) \qquad (12)$$

Atributo Numérico y Clase Simbólica

Se ordenan los ejemplos según el atributo A_i y se considera cada valor v_x del atributo como posible punto de corte. Se consideran entonces como posibles valores del atributo el rango menor o igual a v_x, mayor a v_x y valor desconocido. Se calcula la entropía del rango tomando como base esos tres posibles valores restringidos del atributo.

Atributo Simbólico y Clase Numérica

Se vuelve a tomar como base cada vez cada valor del atributo, tal y como se hacía en el caso Atributo Simbólico y Clase Simbólica, pero en este caso se calcula la varianza de la clase para los valores del atributo mediante la ecuación 13, donde S_j es la suma de los valores de la clase de los ejemplos con valor j en el atributo i, SS_j es la suma de los valores de la clase al cuadrado y W_j es la suma de los pesos de los ejemplos (número de ejemplos si no se incluyen pesos) con valor j en el atributo.

$$Varianza\ (A_{iv_x}) = \sum_{j=1}^{3}\left(SS_j - \frac{S_j}{W_j}\right) \qquad (13)$$

Atributo Numérico y Clase Numérica

Se considera cada valor del atributo como punto de corte tal y como se hacía en el caso Atributo Numérico y Clase Simbólica. Posteriormente, se calcula la varianza tal y como se muestra en la ecuación 13.

En cualquiera de los cuatro casos que se han comentado, lo que se busca es el valor mínimo de la ecuación calculada, ya sea la entropía o la varianza. De esta forma se obtiene el atributo que será raíz del árbol de decisión y sus tres ramas. Lo único que se hará por último es construir dicho árbol: cada rama finaliza en un nodo hoja con el valor de la clase, que será la media o la moda de los ejemplos que se clasifican por ese camino, según se trate de una clase numérica o simbólica.

2.2.4. Reglas de Clasificación

Las técnicas de Inducción de Reglas surgieron hace más de dos décadas y permiten la generación y contraste de árboles de decisión, o reglas y patrones a partir de los datos de entrada. La información de entrada será un conjunto de casos donde se ha asociado una clasificación o evaluación a un conjunto de variables o atributos. Con esa información estas técnicas obtienen el árbol de decisión o conjunto de reglas que soportan la evaluación o clasificación. En los casos en que la información de entrada posee algún tipo de “ruido" o defecto (insuficientes atributos o datos, atributos irrelevantes o errores u

omisiones en los datos) estas técnicas pueden hacer uso de métodos estadísticos de tipo probabilístico para generar árboles de decisión recortados o podados. También en estos casos pueden identificar los atributos irrelevantes, la falta de atributos discriminantes o detectar huecos de conocimiento. Esta técnica suele llevar asociada una alta interacción con el analista de forma que éste pueda intervenir en cada paso de la construcción de las reglas, bien para aceptarlas, bien para modificarlas.

La inducción de reglas se puede lograr fundamentalmente mediante dos caminos:

Generando un árbol de decisión y extrayendo de él las reglas, como puede hacer el sistema C4.5 o bien mediante una estrategia de *covering*, consistente en tener en cuenta cada vez una clase y buscar las reglas necesarias para cubrir todos los ejemplos de esa clase; cuando se obtiene una regla se eliminan todos los ejemplos que cubre y se continúa buscando más reglas hasta que no haya más ejemplos de la clase.

A continuación se expone una técnica de inducción de reglas basada en árboles de decisión, otra basada en *covering* y una más que mezcla las dos estrategias.

2.2.4.1. Algoritmo 1R

El algoritmo más simple de reglas de clasificación para un conjunto de ejemplos es el 1R. Este algoritmo genera un árbol de decisión de un nivel expresado mediante reglas. Consiste en seleccionar un atributo (nodo raíz) del cual nace una rama por cada valor, que va a parar a un nodo hoja con la clase más probable de los ejemplos de entrenamiento que se clasifican a través suyo.

A continuación, se muestra el algoritmo 1R,

Para cada atributo (A) y para cada valor del atributo (A_i):

Contar el número de apariciones de cada clase con A_i
Obtener la clase más frecuente (C_j)
Crear una regla del tipo Ai -> C_j
Calcular el error de las reglas del atributo A
Escoger las reglas con menor error

La clase debe ser simbólica, mientras los atributos pueden ser simbólicos o numéricos. También admite valores desconocidos, que se toman como otro valor más del atributo. En cuanto al error de las reglas de un atributo, consiste en la proporción entre los ejemplos que cumplen la regla y los ejemplos que cumplen la premisa de la regla. En el caso de los atributos numéricos, se generan una serie de *puntos de ruptura* [breakpoint], que discretizarán dicho atributo formando conjuntos. Para ello, se ordenan los ejemplos por el atributo numérico y se recorren. Se van contando las apariciones de cada clase hasta un número *m* que indica el mínimo número de ejemplos que pueden pertenecer a un conjunto, para evitar conjuntos demasiado pequeños. Por último, se unen a este conjunto ejemplos con la clase más frecuente y ejemplos con el mismo valor en el atributo.

Este algoritmo es tremendamente sencillo, tanto es así que 1R no tiene ningún elemento de sofisticación y genera para cada atributo un árbol de profundidad 1, donde una rama está etiquetada por *missing* si aparecen valores desconocidos en ese atributo en el conjunto de entrenamiento. El resto de las ramas tienen como etiqueta un intervalo construido de una manera muy simple, como se ha explicado antes, o un valor nominal, según el tipo de atributo del que se trate.

A continuación se presenta un ejemplo de 1R, basado en los ejemplos de la tabla 2.1.

Atributo	Reglas	Errores	Error Total
Vista	Soleado → no Nublado → si Lluvioso → si	2/5 0/4 2/5	4/14
Temperatura	Alta → no Media → si Baja → si	2/4 2/6 1/4	5/14
Humedad	Alta → no Normal → si	3/7 1/7	4/14
Viento	Falso → si Cierto → no	2/8 3/6	5/14

Para clasificar según la clase jugar, 1R considera cuatro conjuntos de reglas, Uno por cada atributo, que son las mostradas en la tabla anterior, en las que además aparecen los errores que se cometen. De esta forma se concluye que como los errores mínimos corresponden a las reglas generadas por los atributos vista y humedad, cualquiera de ellas es válida, de manera que arbitrariamente se puede elegir cualquiera de estos dos conjuntos de reglas como generador de *1R*.

2.2.4.2. Algoritmo PRISM

PRISM es un algoritmo básico de aprendizaje de reglas que asume que no hay ruido en los datos.

Sea *t* el número de ejemplos cubiertos por la regla y *p* el número de ejemplos positivos cubiertos por la regla. Lo que hace PRISM es añadir condiciones a reglas que maximicen la relación *p/t* (relación entre los ejemplos positivos cubiertos y ejemplos cubiertos en total). De esta forma, el algoritmo en forma de pseudocódigo sería:

```
PRISM (ejemplos) {
Para cada clase (C)
E = ejemplos
Mientras E tenga ejemplos de C
Crea una regla R con parte izquierda vacía y clase C
Hasta R perfecta Hacer
Para cada atributo A no incluido en R y cada valor v de A
Considera añadir la condición A=v a la parte izquierda de R
```

Selecciona el par A=v que maximice p/t (en caso de empates, escoge la que tenga p mayor) Añadir A=v a R Elimina de E los ejemplos cubiertos por R

Este algoritmo va eliminando los ejemplos que va cubriendo cada regla, por lo que las reglas tienen que interpretarse en orden. Se habla entonces de listas de reglas.

Tomando el ejemplo de la tabla 2.1, el algoritmo actúa de la siguiente forma:

REGLA 1. Clase "Sí"		**REGLA 2. Clase "Sí"**			
Añadir a "if <vacío> **Then** SÍ"	**p/t**	**Añadir a "if** <vacío> **Then** SÍ"	**p/t**	**Añadir a "if** <vacío> **Then** SÍ"	**p/t**
Vista=Soleado	2/5	Vista=Soleado	2/5	Vista=Soleado	2/2
Vista=Nublado	4/4	Vista=Lluvioso	3/5	Vista=Lluvioso	2/3
Vista=Lluvioso	3/5	Temperatura=Alta	0/2	Temperatura=Alta	0/0
Temperatura=Alta	2/4	Temperatura=Media	3/5	Temperatura=Media	2/2
Temperatura=Media	4/6	Temperatura=Baja	2/3	Temperatura=Baja	2/3
Temperatura=Baja	3/4	Humedad=Alta	1/5	Viento=Sí	1/2
Humedad=Alta	3/7	Humedad=Normal	4/5	Viento=No	3/3
Humedad=Normal	6/7	Viento=Sí	1/4		
Viento=Sí	3/6	Viento=No	4/6		
Viento=No	6/8				
If Vista=Nublado **Then** Sí		**If** Humedad=Normal **and** Viento=No **Then** Sí			

Lista completa de decision

- **If** Vista=Nublado **Then** Sí
- **If** Humedad=Normal **and** Viento=No **Then** Sí
- **If** Temperatura=Media **and** Humedad=Normal **Then** Sí
- **If** Vista=Lluvioso **and** Viento=No **Then** Sí
- **If** Vista=Soleado **and** Humedad=Alta **Then** No
- **If** Vista=Lluvioso **and** Viento=Sí **Then** Sí

Puede observarse cómo el algoritmo toma en primer lugar la clase *Sí*. Partiendo de todos los ejemplos de entrenamiento (un total de catorce) calcula el cociente *p/t* para cada par atributo-valor y escoge el mayor. En este caso, dado que la condición escogida hace la regla perfecta (*p/t = 1*), se eliminan los cuatro ejemplos que cubre dicha regla y se busca una nueva regla. En la segunda regla se obtiene en un primer momento una condición que no hace perfecta la regla, por lo que se continúa buscando con otra condición. Finalmente, se muestra la lista de decisión completa que genera el algoritmo.

2.2.4.3. Algoritmo PART

Uno de los sistemas más importantes de aprendizaje de reglas es el proporcionado por C4.5, explicado anteriormente. Este sistema, al igual que otros sistemas de inducción de reglas, realiza dos fases: primero, genera un conjunto de reglas de clasificación y después refina estas reglas para mejorarlas, realizando así una proceso de optimización global de dichas reglas. Este proceso de optimización global es siempre muy complejo y costoso computacionalmente hablando.

El algoritmo PART es un sistema que obtiene reglas sin dicha optimización global. Recibe el nombre PART por su modo de actuación: *obtaining rules from PARTial decision trees*, y fue desarrollado por el grupo neozelandés que construyó el entorno WEKA .

El sistema se basa en las dos estrategias básicas para la inducción de reglas: el *covering* y la generación de reglas a partir de árboles de decisión. Adopta la estrategia del *covering* (con lo que se obtiene una lista de decisión) dado que genera una regla, elimina los ejemplares que dicha regla cubre y continúa generando reglas hasta que no queden ejemplos por clasificar. Sin embargo, el proceso de generación de cada regla no es el usual. En este caso, para crear una regla, se genera un árbol de decisión podado, se obtiene la *hoja* que clasifique el mayor número de ejemplos, que se transforma en la regla, y posteriormente se elimina el árbol. Uniendo estas dos estrategias se consigue mayor flexibilidad y velocidad. Además, no se genera un árbol completo, sino un árbol parcial. Un árbol parcial es un árbol de decisión que contiene *brazos* con subárboles no definidos. Para generar este árbol se integran los procesos de construcción y podado hasta que se encuentra un subárbol *estable* que no puede simplificarse más, en cuyo caso se para el proceso y se genera la regla a partir de dicho subárbol.

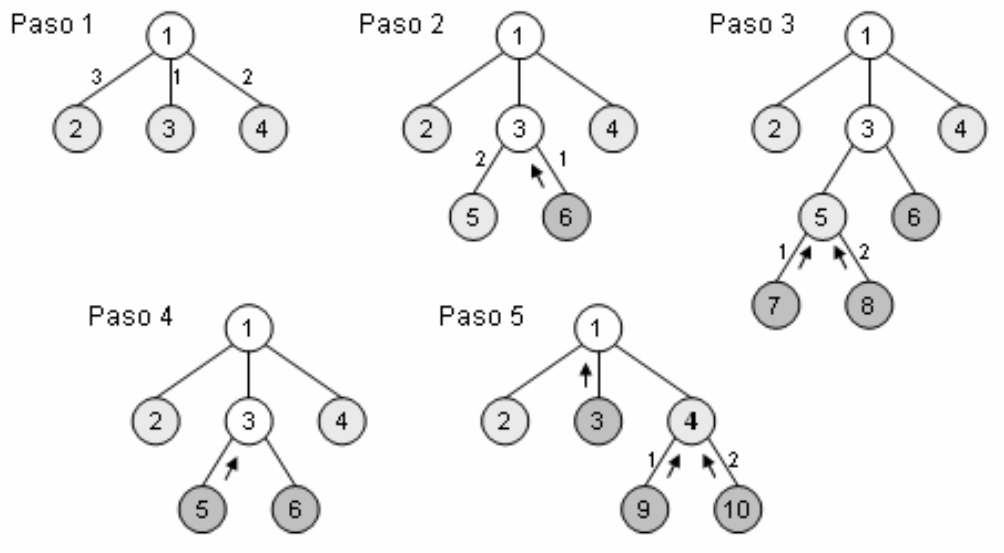
El proceso sería el siguiente:

```
Expandir (ejemplos) {
elegir el mejor atributo para dividir en subconjuntos
Mientras (subconjuntos No expandidos)
Y (todos los subconjuntos expandidos son HOJA)
Expandir (subconjunto)
Si (todos los subconjuntos expandidos son HOJA)
Y (errorSubárbol >= errorNodo)
deshacer la expansión del nodo y nodo es HOJA
```

El proceso de elección del mejor atributo se hace como en el sistema C4.5, es decir, utilizando la razón de ganancia. La expansión de los subconjuntos generados se realiza en orden, comenzando por el que tiene menor entropía y finalizando por el que tiene mayor. La razón de realizarlo así es porque si un subconjunto tiene menor entropía hay más probabilidades de que se genere un subárbol menor y consecuentemente se cree una regla más general. El proceso continúa recursivamente expandiendo los subconjuntos hasta que se obtienen *hojas*, momento en el que se realizará una vuelta atrás. Cuando se realiza dicha vuelta atrás y los hijos del nodo en cuestión son *hojas*, comienza el podado

tal y como se realiza en C4.5 (comparando el error esperado del subárbol con el del nodo), pero únicamente se realiza la función de reemplazamiento del nodo por hoja. Si se realiza el podado se realiza otra vuelta atrás hacia el nodo *padre*, que sigue explorando el resto de sus *hijos*, pero si no se puede realizar el podado el *padre* no continuará con la exploración del resto de nodos hijos. En este momento finalizará el proceso de expansión y generación del árbol de decisión.

A continuación se muestra un ejemplo de generación de árbol parcial con PART.



Junto a cada brazo de un nodo, se muestra el orden de exploración (orden ascendente según el valor de la entropía). Los nodos con relleno gris claro son los que aún no se han explorado y los nodos con relleno gris oscuro los nodos *hoja*. Las flechas ascendentes representan el proceso de *vuelta atrás*. Por último, en el paso 5, cuando el nodo 4 es explorado y los nodos 9 y 10 pasan a ser hoja, el nodo *padre* intenta realizar el proceso de podado, pero no se realiza el reemplazo (representado con el 4 en negrita), con lo que el proceso, al volver al nodo 1, finaliza sin explorar el nodo 2. Una vez generado el árbol parcial se extrae una regla del mismo. Cada *hoja* se corresponde con una posible regla, y lo que se busca es la mejor *hoja*. Si bien se pueden considerar otras heurísticas, en el algoritmo PART se considera mejor hoja aquella que cubre un mayor número de ejemplos. Se podría haber optado, por ejemplo, por considerar mejor aquella que tiene un menor error esperado, pero tener una regla muy precisa no significa lograr un conjunto de reglas muy preciso. Por último, PART permite que haya atributos con valores desconocidos tanto en el proceso de aprendizaje como en el de validación y atributos numéricos, tratándolos exactamente como el sistema C4.5.

2.2.5. Clasificación Bayesiana

Los clasificadores Bayesianos son clasificadores estadísticos, que pueden predecir tanto las probabilidades del número de miembros de clase, como la probabilidad de que una muestra dada pertenezca a una clase particular. La clasificación Bayesiana se basa en

el teorema de Bayes, y los clasificadores Bayesianos han demostrado una alta exactitud y velocidad cuando se han aplicado a grandes bases de datos

A continuación se explican los fundamentos de los clasificadores bayesianos y, más concretamente, del clasificador *naive* Bayesiano. Tras esta explicación se comentará otro clasificador que, si bien no es un clasificador bayesiano, está relacionado con él, dado que se trata también de un clasificador basado en la estadística.

2.2.5.1. Clasificador Naive Bayesiano

Lo que normalmente se quiere saber en aprendizaje es cuál es la mejor hipótesis (más probable) dados los datos. Si denotamos *P(D)* como la probabilidad a priori de los datos (i.e., que datos son más probables que otros), *P(D|h)* la probabilidad de los datos dada una hipótesis, lo que queremos estimar es: *P(h|D)*, la probabilidad a posteriori de *h* dados los datos. La ecuación 14 muestra cómo se puede estimar usando el teorema de Bayes.

$$P(h\backslash D) = \frac{P(D\backslash h)P(h)}{P(D)} \tag{14}$$

Para estimar la hipótesis más probable (MAP, maximum a posteriori hipótesis) se busca el mayor *P(h|D)* como se muestra en la ecuación 15.

$$\begin{aligned} h_{MAP} &= argmax_{h\in H}\left(P(h\backslash D)\right) \\ &= argmax_{h\in H}\frac{P(D\backslash h)P(h)}{P(D)} \\ &= argmax_{h\in H}\, P(D\backslash h)P(h) \end{aligned} \tag{15}$$

Ya que *P(D)* es una constante independiente de *h*. Si se asume que todas las hipótesis son igualmente probables, entonces resulta la hipótesis de máxima verosimilitud (ML, [maximum likelihood]) de la ecuación 16.

$$h_{ML} = argmax_{h\in H}P(D\backslash h) \tag{16}$$

El clasificador *naive* Bayesiano se utiliza cuando se quiere clasificar un ejemplo descrito por un conjunto de atributos (a_i's) en un conjunto finito de clases (V). SE pretende clasificar un nuevo ejemplo de acuerdo con el valor más probable dados los valores de sus atributos. Si se aplica la ecuación 16 al problema de la clasificación se obtendrá la ecuación 17.

$$\begin{aligned} v_{MAP} &= argmax_{v_j\in V}\left(P\left(v_j\backslash a_1,\dots,a_n\right)\right) \\ &= argmax_{v_j\in V}\left(\frac{P\left(a_1,\dots,a_n\backslash v_j\right)P\left(v_j\right)}{P(a_1,\dots,a_n)}\right) \\ &= argmax_{v_j\in V}\left(P\left(a_1,\dots,a_n\backslash v_j\right)P\left(v_j\right)\right) \end{aligned} \tag{17}$$

Además, el clasificador *naive* Bayesiano asume que los valores de los atributos son condicionalmente independientes dado el valor de la clase, por lo que se hace cierta la ecuación 18 y, con ella, la 19.

$$P(a_1, \dots, a_n \backslash v_j) P(v_j) = \prod_i P(a_i \backslash v_j) P(v_j) \tag{18}$$

$$argmax_{v_j \in V} P(v_j \backslash a_1, \dots, a_n) = argmax_{v_j \in V} P(v_j) \prod_i P(a_i \backslash v_j) \tag{19}$$

Los clasificadores *naive* Bayesianos asumen que el efecto de un valor del atributo en una clase dada es independiente de los valores de los otros atributos. Esta suposición se llama "independencia condicional de clase". Ésta simplifica los cálculos involucrados y, en este sentido, es considerado "ingenuo" de ahí su nombre *naive*. Esta suposición es una simplificación de la realidad. A pesar del nombre del clasificador y de la simplificación realizada, el *naive* Bayesiano funciona muy bien, sobre todo cuando se filtra el conjunto de atributos seleccionado para eliminar redundancia, con lo que se elimina también dependencia entre datos.

A continuación, se muestra un ejemplo de aprendizaje con el clasificador *naive Bayesiano*, así como una muestra de cómo se clasificaría un ejemplo de test. Como ejemplo se empleará el de la tabla 2.1.

Proceso de Aprendizaje													
Vista			**Temperatura**			**Humedad**			**Viento**			**Jugar**	
	Sí	**No**		**Sí**	**No**		**Sí**	**No**		**Sí**	**No**	**Sí**	**No**
Soleado	2	3	**Alta**	2	2	**Alta**	3	4	**Sí**	3	3	9	5
Nublado	4	0	**Media**	4	2	**Normal**	6	1	**No**	6	2		
Lluvioso	3	2	**Baja**	3	1								
Soleado	2/9	3/5	**Alta**	2/9	2/5	**Alta**	3/9	4/5	**Sí**	3/9	3/5	9/14	5/14
Nublado	4/9	0/5	**Media**	4/9	2/5	**Normal**	6/9	1/5	**No**	6/9	2/5		
Lluvioso	3/9	2/5	**Baja**	3/9	1/5								

Proceso de Aprendizaje				
Vista	**Temperatura**	**Humedad**	**Viento**	**Jugar**
Soleado	Fría	Alta	Sí	?

$$P(Sí \backslash E) = P(Sí) \times \prod_i P(a_i \backslash Sí) = \frac{9}{14} \times \frac{2}{9} \times \frac{3}{9} \times \frac{3}{9} \times \frac{3}{9} = 0.0053$$

$$P(No \backslash E) = P(No) \times \prod_i P(a_i \backslash No) = \frac{5}{14} \times \frac{3}{5} \times \frac{1}{5} \times \frac{4}{5} \times \frac{3}{5} = 0.0206$$

$$\textbf{Normalizado} \begin{cases} P(Si \backslash E) = \dfrac{0.0053}{0.0053 + 0.0206} = 20.5\% \\ P(No \backslash E) = \dfrac{0.0206}{0.0053 + 0.0206} = 79.5\% \end{cases}$$

Actividad 2.3. Para el ejemplo anterior, realiza la clasificación para el siguiente ejemplo de test:

Vista	Temperatura	Humedad	Viento	Jugar
Lluvioso	Media	Alta	Sí	?

En este ejemplo se observa que en la fase de aprendizaje se obtienen todas las probabilidades condicionadas $P(a_i|v_j)$ y las probabilidades $P(v_j)$. En la clasificación se realiza el producto y se escoge como clase del ejemplo de entrenamiento la que obtenga un mayor valor. Algo que puede ocurrir durante el entrenamiento con este clasificador es que para cada valor de cada atributo no se encuentren ejemplos para todas las clases. Supóngase que para el atributo a_i y el valor j de dicho atributo no hay ningún ejemplo de entrenamiento con clase k. En este caso, $P(a_{ij}|k)=0$. Esto hace que si se intenta clasificar cualquier ejemplo con el par atributo-valor a_{ij}, la probabilidad asociada para la clase k será siempre 0, ya que hay que realizar el producto de las probabilidades condicionadas para todos los atributos de la instancia. Para resolver este problema se parte de que las probabilidades se contabilizan a partir de las frecuencias de aparición de cada evento o, en nuestro caso, las frecuencias de aparición de cada terna atributo-valor-clase. El *estimador de Laplace*, consiste en comenzar a contabilizar la frecuencia de aparición de cada terna a partir del 1 y no del 0, con lo que ninguna probabilidad condicionada será igual a 0.

Una ventaja de este clasificador es la cuestión de los valores perdidos o desconocidos: en el clasificador *naive* Bayesiano si se intenta clasificar un ejemplo con un atributo sin valor simplemente el atributo en cuestión no entra en el productorio que sirve para calcular las probabilidades. Respecto a los atributos numéricos, se suele suponer que siguen una distribución Normal o Gaussiana. Para estos atributos se calcula la media μ y la desviación típica σ obteniendo los dos parámetros de la distribución $N(\mu, \sigma)$, que sigue la expresión de la ecuación 20, donde el parámetro x será el valor del atributo numérico en el ejemplo que se quiere clasificar.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \qquad (20)$$

2.2.5.2. Votación por intervalos de características

Este algoritmo es una técnica basada en la proyección de características. Se le denomina "votación por intervalos de características" (VFI, Voting Feature Interval) porque se construyen intervalos para cada característica o atributo en la fase de aprendizaje y el intervalo correspondiente en cada característica "vota" para cada clase en la fase de clasificación. Al igual que en el clasificador *naive* Bayesiano, cada característica es tratada de forma individual e independiente del resto. Se diseña un sistema de votación para combinar las clasificaciones individuales de cada atributo por separado.

Mientras que en el clasificador *naive* Bayesiano cada característica participa en la clasificación asignando una probabilidad para cada clase y la probabilidad final para cada clase consiste en el producto de cada probabilidad dada por cada característica, en el algoritmo VFI cada característica distribuye sus votos para cada clase y el voto final de cada clase es la suma de los votos obtenidos por cada característica. Una ventaja de estos clasificadores, al igual que ocurría con el clasificador *naive* Bayesiano, es el tratamiento de los valores desconocidos tanto en el proceso de aprendizaje como en el de clasificación: simplemente se ignoran, dado que se considera cada atributo como independiente del resto.

En la fase de aprendizaje del algoritmo VFI se construyen intervalos para cada atributo contabilizando, para cada clase, el número de ejemplos de entrenamiento que aparecen en dicho intervalo. En la fase de clasificación, cada atributo del ejemplo de test añade votos para cada clase dependiendo del intervalo en el que se encuentre y el conteo de la fase de aprendizaje para dicho intervalo en cada clase. El algoritmo sería,

```
Aprendizaje (ejemplos) {
Para cada atributo (A) Hacer
Si A es NUMÉRICO Entonces
Obtener mínimo y máximo de A para cada clase en ejemplos
Ordenar los valores obtenidos (I intervalos)
Si no /* es SIMBÓLICO */
Obtener los valores que recibe A para cada clase en ejemplos
Los valores obtenidos son puntos (I intervalos)
Para cada intervalo I Hacer
Para cada clase C Hacer
contadores [A, I, C] = 0
Para cada ejemplo E Hacer
Si A es conocido Entonces
Si A es SIMBÓLICO Entonces
contadores [A, E.A, E.C] += 1
Si no /* es NUMÉRICO */
Obtener intervalo I de E.A
Si E.A = extremo inferior de intervalo I Entonces
contadores [A, I, E.C] += 0.5
contadores [A, I-1, E.C] += 0.5
Si no
contadores [A, I, E.C] += 1
Normalizar contadores[] /* Σc contadores[A, I, C] = 1 */
}
clasificar (ejemplo E) {
Para cada atributo (A) Hacer
Si E.A es conocido Entonces
Si A es SIMBÓLICO
Para cada clase C Hacer
voto[A, C] = contadores[A, E.A, C]
Si no /* es NUMÉRICO */
Obtener intervalo I de E.A
Si E.A = límite inferior de I Entonces
Para cada clase C Hacer
```

```
voto[A, C] = 0.5*contadores[A,I,C] +
0.5*contadores[A,I-1,C]
Si no
Para cada clase C Hacer
voto[A, C] = contadores [A, I, C]
voto[C] = voto[C] + voto[A, C]
Normalizar voto[]/* Σc voto[C] = 1 */
```

A continuación se presenta un ejemplo de entrenamiento y clasificación con el algoritmo VFI.

Ejemplos de entrenamiento		
Atributo1	**Atributo2**	**Clase**
Verde	4	A
Azul	1	B
Azul	5	A
Azul	2	B
Rojo	5	B
Rojo	3	B
Azul	7	A

→

Fase de Aprendizaje		
Atributo1	**A**	**B**
Verde	1 (1)	0 (0)
Azul	0.5 (2)	0.5 (2)
Rojo	0 (0)	1 (2)

Atributo2	**A**	**B**
7-∞	1 (0.5)	0 (0)
5-7	0.67 (1)	0.33 (0.5)
4-5	0.67 (1)	0.33 (0.5)
1-4	0.25 (0.5)	0.75 (2.5)
-∞-1	0 (0)	1 (0.5)

Clasificación de un ejemplo de Test

Atributo1	**Atributo2**	**Clase**
Azul	6	?

Clase	**Voto 1**	**Voto 2**	**Votos**
A	0.5	0.67	0.585 (1.17)
B	0.5	0.33	0.415 (0.83)

En el ejemplo se muestra una tabla con los ejemplos de entrenamiento y cómo el proceso de aprendizaje consiste en el establecimiento de intervalos para cada atributo con el conteo de ejemplos que se encuentran en cada intervalo. Se muestra entre paréntesis el número de ejemplos que se encuentran en la clase e intervalo concreto, mientras que fuera de los paréntesis se encuentra el valor normalizado. Para el atributo simbólico simplemente se toma como intervalo (punto) cada valor de dicho atributo y se cuenta el número de ejemplos que tienen un valor determinado en el atributo para la clase del ejemplo en cuestión. En el caso del atributo numérico, se obtiene el máximo y el mínimo valor del atributo para cada clase que en este caso son *4* y *7* para la clase *A*, y *1* y *5* para la clase *B*. Se ordenan los valores formándose un total de cinco intervalos y se cuenta el número de ejemplos que se encuentran en un intervalo determinado para su clase, teniendo en cuenta que si se encuentra en el punto compartido por dos intervalos se contabiliza la mitad para cada uno de ellos.

También se muestra un ejemplo de clasificación: en primer lugar, se obtienen los votos que cada atributo por separado concede a cada clase, que será el valor normalizado

del intervalo (o punto si se trata de atributos simbólicos) en el que se encuentre el valor del atributo, y posteriormente se suman los votos (que se muestra entre paréntesis) y se normaliza. La clase con mayor porcentaje de votos (en el ejemplo la clase *A*) *gana*.

2.2.6. Aprendizaje Basado en Ejemplares

El aprendizaje basado en ejemplares o instancias tiene como principio de funcionamiento, en sus múltiples variantes, el almacenamiento de ejemplos. En unos casos todos los ejemplos de entrenamiento, en otros solo los más representativos, en otros los incorrectamente clasificados cuando se clasifican por primera vez, etc. La clasificación posterior se realiza por medio de una función que mide la proximidad o parecido. Dado un ejemplo para clasificar se le clasifica de acuerdo al ejemplo o ejemplos más próximos.

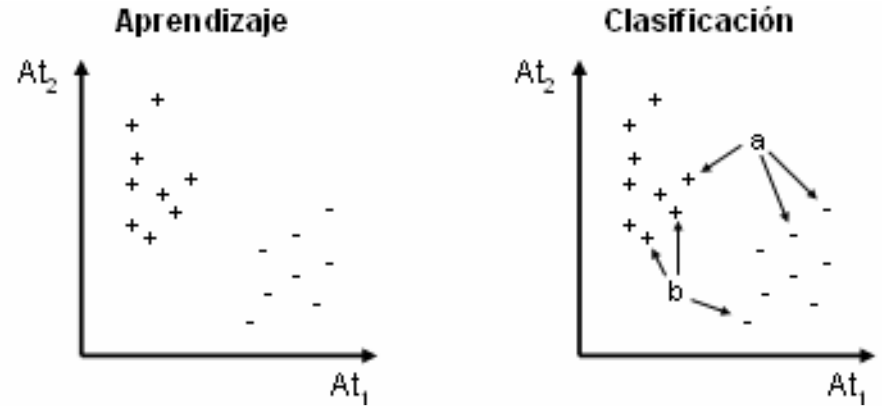
Se han enumerado ventajas e inconvenientes del aprendizaje basado en ejemplares, pero se suele considerar no adecuado para el tratamiento de atributos no numéricos y valores desconocidos. Las mismas medidas de proximidad sobre atributos simbólicos suelen proporcionar resultados muy dispares en problemas diferentes. A continuación se muestran dos técnicas de aprendizaje basado en ejemplares: el método de los *k*-vecinos más próximos y el *k* estrella.

2.2.6.1. Algoritmo de los k-vecinos más próximos

El método de los *k*-vecinos más próximos (KNN, k-Nearest Neighbor) está considerado como un buen representante de este tipo de aprendizaje, y es de gran sencillez conceptual. Se suele denominar método porque es el esqueleto de un algoritmo que admite el intercambio de la función de proximidad dando lugar a múltiples variantes. La función de proximidad puede decidir la clasificación de un nuevo ejemplo atendiendo a la clasificación del ejemplo o de la mayoría de los *k* ejemplos más cercanos. Admite también funciones de proximidad que consideren el peso o coste de los atributos que intervienen, lo que permite, entre otras cosas, eliminar los atributos irrelevantes. Una función de proximidad clásica entre dos instancias x_i y x_j , si suponemos que un ejemplo viene representado por una *n*-tupla de la forma ($a_1(x)$, $a_2(x)$, ... , $a_n(x)$) en la que $a_r(x)$ es el valor de la instancia para el atributo a_r, es la distancia euclídea, que se muestra en la ecuación 21.

$$d(x_i, x_j) = \sqrt{\sum_{i=1}^{n} (x_{il} - x_{jl})^2} \qquad (21)$$

La siguiente figura muestra un ejemplo del algoritmo KNN para un sistema de dos atributos, representándose por ello en un plano.



En este ejemplo se ve cómo el proceso de aprendizaje consiste en el almacenamiento de todos los ejemplos de entrenamiento. Se han representado los ejemplos de acuerdo a los valores de sus dos atributos y la clase a la que pertenecen (las clases son + y -). La clasificación consiste en la búsqueda de los *k* ejemplos (en este caso 3) más cercanos al ejemplo a clasificar. Concretamente, el ejemplo *a* se clasificaría como -, y el ejemplo *b* como +.

Dado que el algoritmo k-NN permite que los atributos de los ejemplares sean simbólicos y numéricos, así como que haya atributos sin valor, el algoritmo para el cálculo de la distancia entre ejemplares se complica ligeramente. El algoritmo que calcula la distancia entre dos ejemplares cualesquiera es:

```
Distancia (E1, E2) {
dst = 0
n = 0
Para cada atributo A Hacer {
dif = Diferencia(E1.A, E2.A)
dst = dst + dif * dif
n = n + 1
}
dst = dst / n
Devolver dst
}
Diferencia (A1, A2) {
Si A1.nominal Entonces {
Si SinValor(A1) O SinValor(A2) O A1 <> A2 Entonces
Devolver 1
Si no
Devolver 0
} Si no {
Si SinValor(A1) O SinValor(A2) Entonces {
Si SinValor(A1) Y SinValor(A2) Entonces
Devolver 1
Si SinValor(A1) Entonces
dif = A2
Si no Entonces
dif = A1
Si dif < 0.5 Entonces
```

```
Devolver 1 – dif
Si no
Devolver dif
} Si no
Devolver abs(A1 – A2)
}
}
```

Además de los distintos tipos de atributos hay que tener en cuenta también, en el caso de los atributos numéricos, los rangos en los que se mueven sus valores. Para evitar que atributos con valores muy altos tengan mucho mayor peso que atributos con valores bajos, se normalizarán dichos valores con la ecuación 22.

$$\frac{x_{il} - min_l}{Max_l - min_l} \tag{22}$$

En esta ecuación x_{if} será el valor *i* del atributo *f*, siendo min_f el mínimo valor del atributo *f* y Max_f el máximo. Por otro lado, el algoritmo permite dar mayor preferencia a aquellos ejemplares más cercanos al que deseamos clasificar. En ese caso, en lugar de emplear directamente la distancia entre ejemplares, se utilizará la ecuación 23.

$$\frac{1}{1 + d(x_i, x_j)} \tag{23}$$

2.2.6.2. Algoritmo k-estrella

El algoritmo K-estrella es una técnica de data mining basada en ejemplares en la que la medida de la distancia entre ejemplares se basa en la teoría de la información. Una forma intuitiva de verlo es que la distancia entre dos ejemplares se define como la complejidad de transformar un ejemplar en el otro. El cálculo de la complejidad se basa en primer lugar en definir un conjunto de transformaciones $T=\{t_1, t_2, ..., t_n, \sigma\}$ para pasar de un ejemplo (valor de atributo) *a* a uno *b*. La transformación σ es la de parada y es la transformación identidad ($\sigma(a)=a$). El conjunto *P* es el conjunto de todas las posibles secuencias de transformaciones descritos en *T** que terminan en σ, y t(a) es una de estas secuencias concretas sobre el ejemplo *a*. Esta secuencia de transformaciones tendrá una probabilidad determinada p(t), definiéndose la función de probabilidad $P^*(b|a)$ como la probabilidad de pasar del ejemplo *a* al ejemplo *b* a través de cualquier secuencia de transformaciones, tal y como se muestra en la ecuación 24.

$$P^*(b\backslash a) = \sum_{t \in P, t(a)=b} p(\bar{t}) \tag{24}$$

Esta función de probabilidad cumplirá las propiedades que se muestran en la ecuación 25.

$$\sum_{b} P^*(b\backslash a) = 1; \;\; 0 \leq P^*(b\backslash a) \leq 1 \tag{25}$$

La función de distancia *K** se define entonces tomando logaritmos, tal y como se muestra en la ecuación 26.

$$K^*(b\backslash a) = -log_2 P^*(b\backslash a) \tag{26}$$

Realmente *K** no es una función de distancia dado que, por ejemplo *K*(a|a)* generalmente no será exactamente *0*, además de que el operador | no es simétrico, esto es, *K*(a|b)* no es igual que *K*(b|a)*. Sin embargo, esto no interfiere en el algoritmo K*. Además, la función *K** cumple las propiedades que se muestran en la ecuación 27.

$$\text{K*}\ (\text{b}|\ \text{a}) \geq 0\ ;\ \text{K*}\ (\text{c}\ |\text{b}) + \text{K*}\ (\text{b}|\ \text{a}) \geq \text{K*}\ (\text{c}\ |\ \text{a}) \tag{27}$$

Una vez explicado cómo se obtiene la función *K** y cuáles son sus propiedades, se presenta la expresión concreta de la función *P**, de la que se obtiene *K**, para los tipos de atributos admitidos por el algoritmo: numéricos y simbólicos.

Probabilidad de transformación para los atributos permitidos

En cuanto a los atributos numéricos, las transformaciones consideradas serán restar del valor *a* un número *n* o sumar al valor *a* un número *n*, siendo *n* un número mínimo. La probabilidad de pasar de un ejemplo con valor *a* a uno con valor *b* vendrá determinada únicamente por el valor absoluto de la diferencia entre *a* y *b*, que se denominará *x*. Se escribirá la función de probabilidad como una función de densidad, tal y como se muestra en la ecuación 28, donde x_0 será una medida de longitud de la escala, por ejemplo, la media esperada para *x* sobre la distribución *P**. Es necesario elegir un x_0 razonable. Posteriormente se mostrará un método para elegir este factor.

Para los simbólicos, se considerarán las probabilidades de aparición de cada uno de los valores de dicho atributo.

$$P^*(x) = \frac{1}{2x_0} e^{-x/x_0} d_x \tag{28}$$

Si el atributo tiene un total de *n* posibles valores, y la probabilidad de aparición del valor *i* del atributo es p_i (obtenido a partir de las apariciones en los ejemplos de entrenamiento), se define la probabilidad de transformación de un ejemplo con valor *i* a uno con valor *j* como se muestra en la ecuación 29.

$$P^*(j\backslash i) = \begin{cases} (1-s)p_j & si \quad i \neq j \\ s + (1-s)p_i & si \quad i = j \end{cases} \tag{29}$$

En esta ecuación *s* es la probabilidad del símbolo de parada (σ). De esta forma, se define la probabilidad de cambiar de valor como la probabilidad de que no se pare la transformación multiplicado por la probabilidad del valor de destino, mientras la probabilidad de continuar con el mismo valor es la probabilidad del símbolo de parada más la probabilidad de que se continúe transformando multiplicado por la probabilidad del valor de destino. También es importante, al igual que con el factor x_0, definir correctamente la probabilidad *s*. Y como ya se comentó con x_0, posteriormente se comentará un método para obtenerlo. También debe tenerse en cuenta la posibilidad de los atributos con valores desconocidos. Cuando los valores desconocidos aparecen en los ejemplos de entrenamiento se propone como solución el considerar que el atributo desconocido se determina a través del resto de ejemplares de entrenamiento.

Esto se muestra en la ecuación 30, donde *n* es el número de ejemplos de entrenamiento.

$$P^*(?\backslash a) = \sum_{b=1}^{n} \frac{P * (b\backslash a)}{n} \tag{30}$$

Combinación de atributos

Ya se han definido las funciones de probabilidad para los tipos de atributos permitidos. Pero los ejemplos reales tienen más de un atributo, por lo que es necesario combinar los resultados obtenidos para cada atributo. Y para combinarlos, y definir así la distancia entre dos ejemplos, se entiende la probabilidad de transformación de un ejemplar en otro como la probabilidad de transformar el primer atributo del primer ejemplo en el del segundo, seguido de la transformación del segundo atributo del primer ejemplo en el del segundo, etc. De esta forma, la probabilidad de transformar un ejemplo en otro viene determinado por la multiplicación de las probabilidades de transformación de cada atributo de forma individual, tal y como se muestra en la ecuación 32. En esta ecuación *m* será el número de atributo de los ejemplos. Y con esta definición la distancia entre dos ejemplos se define como la suma de distancias entre cada atributo de los ejemplos.

$$P^*(E_1, E_2) = \prod_{i=1}^{m} P * (v_{2i} \backslash v_{1i}) \tag{31}$$

Selección de los parámetros aleatorios

Para cada atributo debe determinarse el valor para los parámetros *s* o x_0 según se trate de un atributo simbólico o numérico respectivamente. Además, el valor de este atributo es muy importante. Por ejemplo, si a *s* se le asigna un valor muy bajo las probabilidades de transformación serán muy altas, mientras que si s se acerca a *0* las probabilidades de transformación serán muy bajas. Y lo mismo ocurriría con el parámetro x_0. En ambos casos se puede observar cómo varía la función de probabilidad P^* según se varía el número de ejemplos incluidos partiendo desde *1* (vecino más cercano) hasta *n* (todos los ejemplares con el mismo peso). Se puede calcular para cualquier función de probabilidad el número efectivo de ejemplos como se muestra en la ecuación 32, en la

que *n* es el número de ejemplos de entrenamiento y n_0 es el número de ejemplos con la distancia mínima al ejemplo *a* (para el atributo considerado). El algoritmo K* escogerá para x_0 (o *s*) un número entre n_0 y *n*.

$$n_0 = \frac{(\sum_{b=1}^{n} P^*(b\backslash a))^2}{\sum_{b=1}^{n} P^*(b\backslash a)^2} \leq n \qquad (32)$$

Por conveniencia se expresa el valor escogido como un *parámetro de mezclado* (blending) *b*, que varía entre *b=0%* (n_0) y *b=100%* (*n*). La configuración de este parámetro se puede ver como una *esfera de influencia* que determina cuantos vecinos de *a* deben considerarse importantes. Para obtener el valor correcto para el parámetro x_0 (o *s*) se realiza un proceso iterativo en el que se obtienen las esferas de influencia máxima (x_0 o *s* igual a *0*) y mínima (x_0 o *s* igual a *1*), y se aproximan los valores para que dicha esfera se acerque a la necesaria para cumplir con el parámetro de mezclado.

A continuación se presenta un ejemplo práctico de cómo obtener los valores para los parámetros x_0 o *s*. Se va a utilizar para ello el problema que se presentó en la tabla 2.1, y más concretamente el atributo *Vista* con el valor igual a *Lluvioso*, de dicho problema.

- **Objetivo:**

$$esfera = \frac{b}{100} n_i + n_{lluvioso} \ , \forall i \neq lluvioso = 0.2 * 9 + 5 = 6.8$$

- **Iteración 0 (Inicio):**

$$v_{inferior0} = 0 + \frac{EPSILON}{2} = 0.005 \ \rightarrow esfera = 13.99928$$

$$v_{superior0} = 1 - \frac{EPSILON}{2} = 0.995 \ \rightarrow esfera = 5.03012$$

$$valor_0 = 1 - \frac{b}{100} = 0.8 \rightarrow esfera = 6.41218$$

- **Iteración 1:**

$$v_{inferior1} = v_{inferior0;} \ v_{superior1} = valor_0$$

$$v_1 = \frac{v_{inferior1} + v_{superior1}}{2} = 0.4025 \rightarrow esfera = 10.63580$$

- **Iteración 2:**

$$v_{inferior2} = valor_1; v_{superior2} = v_{superior1}$$

$$valor_2 = \frac{v_{inferior1} + v_{superior1}}{2} = 0.60125 \rightarrow esfera = 8.29876$$

$$esfera = \frac{(\sum P(b\backslash lluv))^2}{\sum P(b\backslash lluv)^2}$$
$$= \frac{(P(sol\backslash lluv) * n_{sol} + P(nub\backslash lluv) * n_{nub} + P(lluv\backslash lluv) * n_{lluv})^2}{P(sol\backslash lluv)^2 * n_{sol} + P(nub\backslash lluv)^2 * n_{nub} + P(lluv\backslash lluv)^2 * n_{lluv}}$$
$$= \frac{(0.00949 * 5 + 0.00949 * 4 + 0.05244 * 5)^2}{0.00949^2 * 5 + 0.00949^2 * 4 + 0.05244^2 * 5} = 8.29876$$

$$P(sol\backslash lluv) = \frac{1-s}{nv}\Big/_m = \frac{1-0.60125}{3}\Big/_{14} = 0.00949$$
$$P(nub\backslash lluv) = \frac{1-0.60125}{3}\Big/_{14} = 0.00949$$
$$P(lluv\backslash lluv) = s + \frac{1-s}{nv}\Big/_m = 0.60125 + \frac{1-0.60125}{3}\Big/_{14} = 0.05244$$

- **Iteración 3:**

$$v_{inferior3} = valor_2;\ v_{superior3} = v_{superior2}$$
$$valor_3 = \frac{v_{inferior3} + v_{superior3}}{2} = 0.70062 \rightarrow esfera = 7.29193$$

- **Iteración 8:**

$$v_{inferior8} = v_{inferior7} = 0.75031;\ v_{superior8} = v_{superior7} = 0.75652$$
$$valor_8 = \frac{v_{inferior8} + v_{superior8}}{2} = 0.75341 \rightarrow esfera = 6.80871$$

$$abs(esfera - objetivo) < EPSILON \rightarrow Conseguido! \rightarrow s = 0.75341$$

En el ejemplo se muestra cómo el objetivo es conseguir un valor para *s* tal que se obtenga una esfera de influencia de *6,8* ejemplos. Los parámetros de configuración necesarios para el funcionamiento del sistema son: el parámetro de mezclado *b*, en este caso igual a *20%*; una constante denominada *EPSILON*, en este caso igual a *0,01*, que determina entre otras cosas cuándo se considera alcanzada la esfera de influencia deseada. En cuanto a la nomenclatura empleada, *n* será el número total de ejemplos de entrenamiento, *nv* el número de valores que puede adquirir el atributo, y se han empleado abreviaturas para denominar los valores del atributo: *lluv* por lluvioso, *nub* por nublado y *sol* por soleado.

Las ecuaciones empleadas para el cálculo de la esfera y de *P** no son exactamente las definidas en las ecuaciones definidas anteriormente. Sin embargo, en el ejemplo se han empleado las implementadas en la herramienta WEKA por los creadores del algoritmo. En cuanto al ejemplo en sí, se muestra cómo son necesarias 8 iteraciones para llegar a conseguir el objetivo planteado, siendo el resultado de dicho proceso, el valor de *s*, igual a *0,75341*.

Actividad 2.4. Repetir el ejemplo anterior considerando en este caso el atributo temperatura con el valor igual a Media

Clasificación de un ejemplo

Se calcula la probabilidad de que un ejemplo *a* pertenezca a la clase *c* sumando la probabilidad de *a* a cada ejemplo que es miembro de *c*, tal y como se muestra en la ecuación 33.

$$P^*(c\backslash a) = \sum_{b \in c} P^*(b\backslash a) \tag{33}$$

Se calcula la probabilidad de pertenencia a cada clase y se escoge la que mayor resultado haya obtenido como predicción para el ejemplo.

El proceso que se sigue para determinar a qué clase pertenece un ejemplo de *test* determinado es el siguiente: en primer lugar, habría que calcular los parámetros x_0 y *s* que aún no se conocen para los pares *atributo-valor* del ejemplo de *test*. Posteriormente se aplican las ecuaciones. Una vez obtenidas las probabilidades, se normalizan y se escoge la mayor de las obtenidas

2.2.7. Redes de Neuronas

Las redes de neuronas constituyen una técnica inspirada en los trabajos de investigación, iniciados en 1930, que pretendían modelar computacionalmente el aprendizaje humano llevado a cabo a través de las neuronas en el cerebro. Posteriormente se comprobó que tales modelos no eran del todo adecuados para describir el aprendizaje humano.

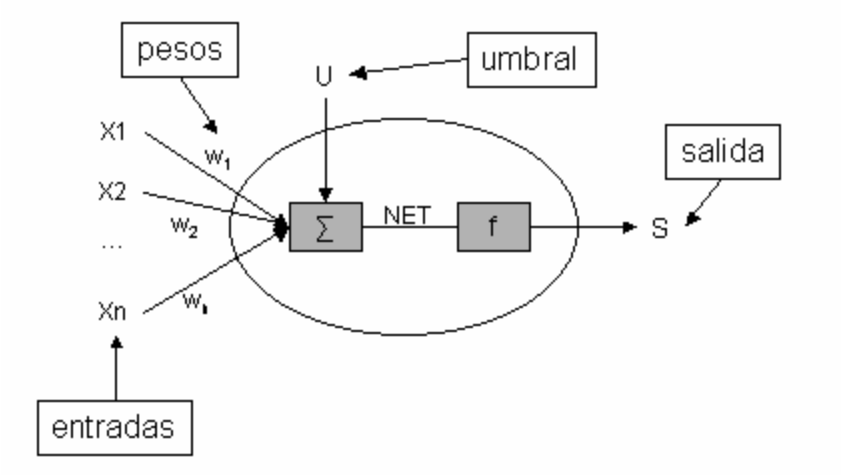
Las redes de neuronas constituyen una nueva forma de analizar la información con una diferencia fundamental con respecto a las técnicas tradicionales: son capaces de detectar y aprender complejos patrones y características dentro de los datos. Se comportan de forma parecida a nuestro cerebro aprendiendo de la experiencia y del pasado, y aplicando tal conocimiento a la resolución de problemas nuevos. Este aprendizaje se obtiene como resultado del adiestramiento ("*training*") y éste permite la sencillez y la potencia de adaptación y evolución ante una realidad cambiante y muy dinámica. Una vez adiestradas las redes de neuronas pueden hacer previsiones, clasificaciones y segmentación. Presentan además, una eficiencia y fiabilidad similar a los métodos estadísticos y sistemas expertos, si no mejor, en la mayoría de los casos. En aquellos casos de muy alta complejidad las redes neuronales se muestran como especialmente útiles dada la dificultad de modelado que supone para otras técnicas. Sin embargo las redes de neuronas tienen el inconveniente de la dificultad de acceder y comprender los modelos que generan y presentan dificultades para extraer reglas de tales modelos. Otra característica es que son capaces de trabajar con datos incompletos e, incluso contradictorios, lo que, dependiendo del problema, puede resultar una ventaja o un inconveniente. Las redes de neuronas poseen las dos formas de aprendizaje: supervisado

y no supervisado; derivadas del tipo de paradigma que usan: el no supervisado (usa paradigmas como los ART "*Adaptive Resonante Theory*"), y el supervisado que suele usar el paradigma del "*Backpropagation*".

Las redes de neuronas están siendo utilizadas en distintos y variados sectores como la industria, el gobierno, el ejército, las comunicaciones, la investigación aeroespacial, la banca y las finanzas, los seguros, la medicina, la distribución, la robótica, el marketing, etc. En la actualidad se estudia la posibilidad de utilizar técnicas avanzadas y novedosas como los Algoritmos Genéticos para crear nuevos paradigmas que mejoren el adiestramiento y la propia selección y diseño de la arquitectura de la red (número de capas y neuronas), diseño que ahora debe realizarse en base a la experiencia del analista y para cada problema concreto.

2.2.7.1. Estructura de las Redes de Neuronas

Las redes neuronales se construyen estructurando en una serie de niveles o capas (al menos tres: entrada, procesamiento u oculta y salida) compuestas por nodos o "neuronas", que tienen la estructura que se muestra a continuación,



Tanto el umbral como los pesos son constantes que se inicializarán aleatoriamente y durante el proceso de aprendizaje serán modificados. La salida de la neurona se define tal y como se muestra en las ecuaciones 34 y 35.

$$NET = \sum_{i=1}^{N} X_i \, w_i + U \qquad (34)$$

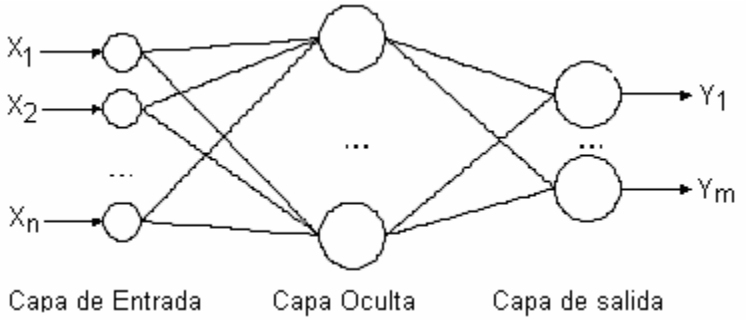
$$S = f(NET) \qquad (35)$$

Como función f se suele emplear una función sigmoidal, bien definida entre *0* y *1* (ecuación 36) o entre *–1* y *1* (ecuación 37).

$$f(x) = \frac{1}{1 + e^{x}} \qquad (36)$$

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad (37)$$

Cada neurona está conectada a todas las neuronas de las capas anterior y posterior a través de los pesos o "dendritas", tal y como se muestra en la figura.



Cuando un nodo recibe las entradas o "estímulos" de otras los procesa para producir una salida que transmite a la siguiente capa de neuronas. La señal de salida tendrá una intensidad fruto de la combinación de la intensidad de las señales de entrada y de los pesos que las transmiten. Los pesos o dendritas tienen un valor distinto para cada par de neuronas que conectan pudiendo así fortalecer o debilitar la conexión o comunicación entre neuronas particulares. Los pesos son modificados durante el proceso de adiestramiento.

El diseño de la red de neuronas consistirá, entre otras cosas, en la definición del número de neuronas de las tres capas de la red. Las neuronas de la capa de entrada y las de la capa de salida vienen dadas por el problema a resolver, dependiendo de la codificación de la información. En cuanto al número de neuronas ocultas (y/o número de capas ocultas) se determinará por prueba y error. Por último, debe tenerse en cuenta que la estructura de las neuronas de la capa de entrada se simplifica, dado que su salida es igual a su entrada: no hay umbral ni función de salida.

2.2.7.2. Proceso de adiestramiento (retropropagación)

Existen distintos métodos o paradigmas mediante los cuales estos pesos pueden ser variados durante el adiestramiento de los cuales el más utilizado es el de retropropagación (Backpropagation). Este paradigma varía los pesos de acuerdo a las diferencias encontradas entre la salida obtenida y la que debería obtenerse. De esta forma, si las diferencias son grandes se modifica el modelo de forma importante y según van siendo menores, se va convergiendo a un modelo final estable. El error en una red de neuronas para un patrón [x= (x_1, x_2, …, x_n), t(x)], siendo *x* el patrón de entrada, *t(x)* la salida deseada e *y(x)* la proporcionada por la red, se define como se muestra en la ecuación 38 para *m* neuronas de salida y como se muestra en la ecuación 39 para *1* neurona de salida.

$$e(x) = \|t(x) - y(x)\|^2 = \frac{1}{2}\sum_{i=1}^{m}(t_i\,(x) - y_i(x))^2 \tag{38}$$

$$e(x) = \frac{1}{2}(t(x) - y(x))^2 \tag{39}$$

El método de descenso de gradiente consiste en modificar los parámetros de la red siguiendo la dirección negativa del gradiente del error. Lo que se realizaría mediante la ecuación 40.

$$w^{nuevo} = w^{anterior} + \alpha\left(-\frac{\delta e}{\delta w}\right) = w^{anterior} - \alpha\frac{\delta e}{\delta w} \tag{40}$$

En la ecuación 40, *w* es el peso a modificar en la red de neuronas (pasando de $w^{anterior}$ a w^{nuevo}) y α es la razón de aprendizaje, que se encarga de controlar cuánto se desplazan los pesos en la dirección negativa del gradiente. Influye en la velocidad de convergencia del algoritmo, puesto que determina la magnitud del desplazamiento.

El algoritmo de retropropagación es el resultado de aplicar el método de descenso del gradiente a las redes de neuronas. El algoritmo completo es el siguiente:

Paso 1: Inicialización aleatoria de los pesos y umbrales.
Paso 2: Dado un patrón del conjunto de entrenamiento *(x, t(x))*, se presenta el vector *x* a la red y se calcula la salida de la red para dicho patrón, *y(x)*.
Paso 3: Se evalúa el error *e(x)* cometido por la red.
Paso 4: Se modifican todos los parámetros de la red utilizando la ecuación 40.
Paso 5: Se repiten los pasos 2, 3 y 4 para todos los patrones de entrenamiento, completando así un ciclo de aprendizaje.
Paso 6: Se realizan n ciclos de aprendizaje (pasos 2, 3, 4 y 5) hasta que se verifique el criterio de parada establecido.

En cuanto al criterio de parada, se debe calcular la suma de los errores en los patrones de entrenamiento. Si el error es constante de un ciclo a otro, los parámetros dejan de sufrir modificaciones y se obtiene así el error mínimo. Por otro lado, también se debe tener en cuenta el error en los patrones de validación, que se presentarán a la red tras *n* ciclos de aprendizaje. Si el error en los patrones de validación evoluciona favorablemente se continúa con el proceso de aprendizaje. Si el error no desciende, se detiene el aprendizaje.

2.2.8. Lógica borrosa ("Fuzzy logic")

La lógica borrosa surge de la necesidad de modelar la realidad de una forma más exacta evitando precisamente el determinismo o la exactitud. En otras palabras, permite el tratamiento probabilístico de la categorización de un colectivo.

Así, para establecer una serie de grupos, segmentos o clases en los cuales se puedan clasificar a las personas por la edad, lo inmediato sería proponer unas edades límite para establecer tal clasificación de forma disjunta. Así los niños serían aquellos cuya edad fuera menor a los 12 años, los adolescentes aquellos entre 12 y 17 años, los jóvenes aquellos entre 18 y 35, las personas maduras entre 36 y 45 años y así sucesivamente. Se habrían creado unos grupos disjuntos cuyo tratamiento, a efectos de clasificación y procesamiento, es muy sencillo: basta comparar la edad de cada persona con los límites establecidos. Sin embargo enseguida se observa que esto supone una simplificación enorme dado que una persona de 16 años 11 meses y veinte días pertenecería al grupo de los adolescentes y, seguramente, es más parecido a una persona de 18 (miembro de otro grupo) que a uno de 12 (miembro de su grupo). Lógicamente no se puede establecer un grupo para cada año, dado que sí se reconocen grupos, y no muchos, con comportamientos y actitudes similares en función de la edad. Lo que implícitamente se está descubriendo es que las clases existen pero que la frontera entre ellas no es clara ni disjunta sino “difusa” y que una persona puede tener aspectos de su mentalidad asociados a un grupo y otros asociados a otro grupo, es decir que implícitamente se está distribuyendo la pertenencia entre varios grupos. Cuando esto se lleva a una formalización matemática surge el concepto de distribución de posibilidad, de forma que lo que entendería como función de pertenencia a un grupo de edad serían unas curvas de posibilidad. Por tanto, la lógica borrosa es aquella técnica que permite y trata la existencia de barreras difusas o suaves entre los distintos grupos en los que se categoriza un colectivo o entre los distintos elementos, factores o proporciones que concurren en una situación o solución.

Para identificar las áreas de utilización de la lógica difusa basta con determinar cuántos problemas hacen uso de la categorización disjunta en el tratamiento de los datos para observar la cantidad de posibles aplicaciones que esta técnica puede tener. Sin embargo, el tratamiento ortodoxo y purista no siempre está justificado dada la complejidad que induce en el procesamiento (pasamos de valores a funciones de posibilidad) y un modelado sencillo puede ser más que suficiente. Aun así, existen problemáticas donde este modelado sí resulta justificado, como en el control de procesos y la robótica, entre otros.

2.2.9. Técnicas Genéticas: Algoritmos Genéticos

Los Algoritmos Genéticos son otra técnica que tiene su inspiración, en la Biología como las Redes de Neuronas. Estos algoritmos representan el modelado matemático de como los cromosomas en un marco evolucionista alcanzan la estructura y composición más óptima en aras de la supervivencia. Entendiendo la evolución como un proceso de búsqueda y optimización de la adaptación de las especies que se plasma en mutaciones y cambios de los de reproducción (mutación y cruce) para ser utilizadas en todo tipo de problemas de búsqueda y optimización. Se da la mutación cuando alguno o algunos de los genes cambian bien de forma aleatoria o de forma controlada vía funciones y se obtiene el cruce cuando se construye una nueva solución a partir de dos contribuciones procedentes de otras soluciones "padre". En cualquier caso, tales transformaciones se realizan sobre aquellos especímenes o soluciones más aptas o mejor adaptadas.

Dado que los mecanismos biológicos de evolución han dado lugar a soluciones realmente idóneas, cabe esperar que la aplicación de tales mecanismos a la búsqueda y

optimización de otro tipo de problemas tenga el mismo resultado. De esta forma los Algoritmos Genéticos transforman los problemas de búsqueda y optimización de soluciones en un proceso de evolución de unas soluciones de partida. Las soluciones se convierten en cromosomas, transformación que se realiza pasando los datos a formato binario, y a los mejores se les van aplicando las reglas de evolución (funciones probabilísticas de transición) hasta encontrar la solución óptima. En muchos casos, estos mecanismos brindan posibilidades de convergencia más rápidos que otras técnicas.

El uso de estos algoritmos no está tan extendido como otras técnicas, pero van siendo cada vez más utilizados directamente en la solución de problemas, así como en la mejora de ciertos procesos presentes en otras herramientas. Así, por ejemplo, se usan para mejorar los procesos de adiestramiento y selección de arquitectura de las redes de neuronas, para la generación e inducción de árboles de decisión y para la síntesis de programas a partir de ejemplos.