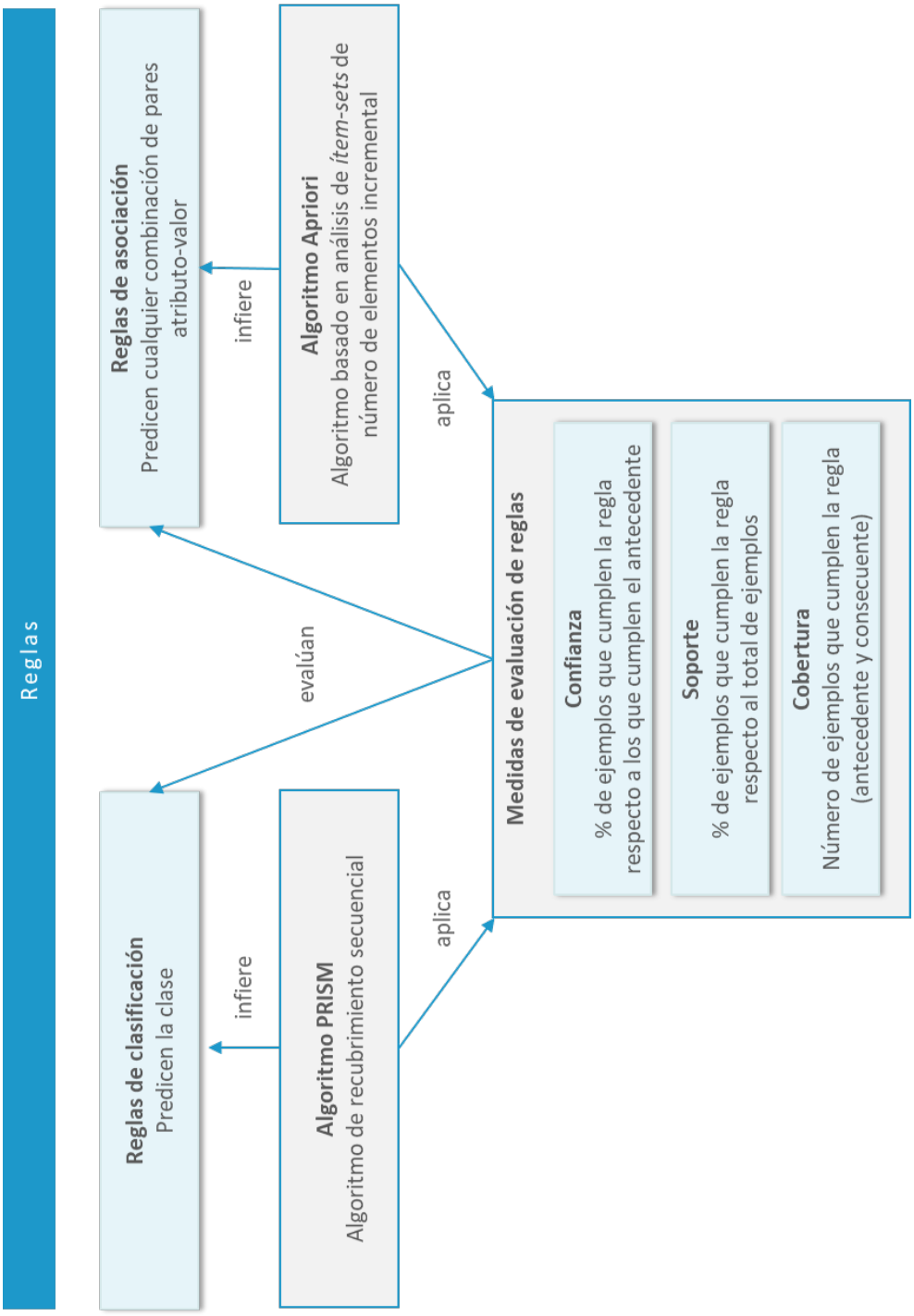
Técnicas de Inteligencia Artificial

Tema 4. Reglas

Índice

Esquema



4.1. ¿Cómo estudiar este tema?

Para estudiar este tema deberás leer las **Ideas clave** que se presentan a continuación. Puedes completar el estudio visualizando la lección magistral, revisando referencias y bibliografía, así como accediendo a los recursos adicionales que se facilitan. Es muy recomendable ver la lección magistral antes de realizar las actividades propuestas.

Al finalizar el estudio de este tema serás capaz de:

- Representar conocimiento mediante reglas de clasificación y de asociación.
- Aplicar algoritmos básicos de construcción de reglas para resolver problemas de aprendizaje.
- Identificar aplicaciones prácticas de las técnicas de aprendizaje de reglas de clasificación o asociación.

4.2. Reglas de clasificación y reglas de asociación

La representación del conocimiento, junto con el razonamiento, son unos de los aspectos más importantes de la inteligencia artificial. El objetivo principal de ambos es la búsqueda de una representación de este conocimiento que facilite la inferencia de nuevo conocimiento. Algunas de las características que una buena representación del conocimiento debe cumplir son (Ruiz, 2016):

- **Comprensible**: fácil de entender por humanos, soportando modularidad y jerarquía (por ejemplo, una Vespa es una moto, que a su vez en un vehículo).
- **Eficiente**: cumple con el objetivo perseguido utilizando el menor número de recursos posibles.
- **Adaptable**: facilita la modificación y actualización del conocimiento.
- **Consistente**: capaz de gestionar y eliminar conocimiento redundante o conflictivo (por ejemplo, «Juan ha comprado una barra de pan» y «la barra de pan ha sido comprada por Juan» implicarían redundancia).
- **Cobertura**: cubre la información en anchura y en profundidad permitiendo resolver conflictos y eliminar redundancias.
- **Completitud**: incluye toda la información necesaria.

Ideas clave

Los sistemas de reglas son uno de los métodos más extendidos para representar conocimiento. Algunas ventanas de los sistemas de reglas:

- Modularidad.
- El conocimiento puede ser ampliado y modificado.
- Fáciles de entender.
- Separación entre control y conocimiento.
- Permiten explicar las decisiones.

De forma general, una representación del conocimiento en forma de reglas estará formada por dos partes:

- Un **antecedente** que incluye las **condiciones de aplicación** del conocimiento.
- Un **consecuente** en el que se indica la **conclusión, respuesta o acción** que ha de llevarse a cabo cuando se cumple el antecedente.

La sintaxis básica de una regla es:

```
SI <antecedente>

    ENTONCES <consecuente>
```

Ideas clave

Las reglas pueden presentar múltiples condiciones unidas por los operadores lógicos AND (conjunción) y OR (disyunción). Asimismo, el consecuente puede presentar múltiples conclusiones. De cualquier manera, es recomendable no mezclar en la misma regla conjunciones y disyunciones.

Por ejemplo, la siguiente regla presenta únicamente conjunciones:

```
SI <antecedente 1>

AND <antecedente 2>

AND <antecedente 3>


ENTONCES <consecuente 1>

<consecuente 2>
```

Los **antecedentes** de una regla incorporan dos partes: **un objeto y su valor, que van asociados por un operador**. Este operador puede ser matemático para dar un valor numérico al objeto o puede ser lingüístico, con lo que se asigna un valor lingüístico al objeto. Por ejemplo:

```
SI edad < 25

    AND “años con carné de conducir” < 2

    AND “número de siniestros previos” > 0

ENTONCES “riesgo de siniestro” es alto
```

En el ejemplo anterior, los objetos en el antecedente son numéricos mientras que el objeto del consecuente tiene un valor lingüístico. También se le puede asignar al consecuente, valores numéricos e incluso expresiones aritméticas como, por ejemplo:

```
SI “edad” < 25

    AND “años con carné de conducir” < 2

    AND “número de siniestros previos” > 0

ENTONCES “riesgo de siniestro” = “edad” * 1.5
```

La representación del conocimiento mediante reglas de clasificación es una alternativa a los árboles de decisión. De hecho, la representación mediante árboles de decisión se puede mapear a la representación mediante reglas de clasificación y viceversa.

A veces, es interesante trasformar un árbol en un conjunto de reglas como, por ejemplo, en los casos en que se tienen árboles grandes difíciles de interpretar. El antecedente de la regla contiene una serie de restricciones de valores que han de tener los atributos, mientras que el consecuente determina un valor de la clase.

En el árbol la serie de restricciones viene representada por las ramas mientras que el consecuente corresponde a la hoja.

Mientras que las reglas de clasificación predicen la clase, las reglas de asociación predicen valores de atributos, combinaciones de valores de atributos, o la propia clase.

Ideas clave

Dado que el consecuente de una regla de asociación puede contener cualquier combinación de valores de atributos, la cantidad de reglas que habría que considerar es muy grande y, por tanto, algunas técnicas que se utilizan para obtener reglas de clasificación no se pueden utilizar para inducir reglas de asociación. Se ha de aplicar, por tanto, métodos que obtengan únicamente aquellas reglas de interés que se apliquen a un número de ejemplos grande y sean precisas.

El interés de las reglas de asociación es descubrir combinaciones de pares atributo-valor que ocurren con frecuencia en un conjunto de datos.

¿En qué casos se puede querer descubrir este tipo de combinaciones? Por ejemplo, en un comercio en línea puede ser muy interesante conocer los productos que los clientes adquieren conjunta y habitualmente con el fin de identificar clientes con patrones de compra similares y así poder predecir posibles compras futuras de estos clientes y realizar ofertas personalizadas.

Como previamente se ha indicado, es posible encontrar un gran número de reglas correspondiente al gran número de posibles combinaciones de valores de atributos. Por tanto, surge la necesidad de utilizar alguna **medida** que indique, por ejemplo, la probabilidad de que el consecuente de la regla se cumpla si se da el antecedente, determinando así la relevancia o interés de la regla.

Específicamente dos medidas populares que se suelen utilizar con el fin de evaluar una regla son:

Confianza	Soporte

Ideas clave

La confianza es la probabilidad condicional de que dado un evento A se produzca un evento B.

$$Confianza\,(A \rightarrow B) = P\,(B|A) = \frac{P\,(A|B)\,P\,(B)}{P\,(A)}$$

Al hablar de reglas, la confianza se puede expresar como el porcentaje de ejemplos que satisfacen el antecedente y consecuente de la regla entre aquellos que satisfacen el antecedente.

Por ejemplo, en la tienda en línea antes mencionada, mediante la medida de la confianza se pretende conocer la probabilidad de que la compra de un detergente para ropa conduzca a la compra de un suavizante para ropa.

El soporte se refiere al cociente del número de ejemplos que cumplen el antecedente y el consecuente de la regla entre el número total de ejemplos. En notación probabilística se puede expresar como:

$$Soporte\,(A \rightarrow B) = P\,(A \cup B)$$

Esta medida de soporte es interesante para detectar aquellas reglas que, aunque se cumplen en algún caso y aunque puedan tener alta confianza, no son relevantes porque cubren casos poco frecuentes. A continuación, se muestra con un ejemplo en qué consisten las reglas de asociación y cómo se aplican estas dos medidas. Específicamente, se van a utilizar los datos del conocido problema «Jugar al aire libre» mostrados en la Tabla 1.

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

Tabla 1. Datos del problema «Jugar al aire libre».

Dado que se trata de un problema con un bajo número de instancias a simple vista se puede extraer algunas reglas de asociación:

```
Regla 1: SI Ambiente es nublado

        ENTONCES jugar = sí


Regla 2: SI Temperatura es baja

        ENTONCES humedad es normal


Regla 3: SI Temperatura es media

        ENTONCES humedad es alta
```

La regla de asociación 1 se puede considerar una regla de clasificación, ya que el consecuente se refiere al atributo de salida o a la clase. Sin embargo, las otras reglas relacionan atributos de entrada.

Se calculan a continuación los valores de confianza y soporte para estas reglas:

$$Confianza\,(Regla\,1{:}ambiente = nublado \rightarrow jugar = sí) = \frac{4}{4} = 1$$

$$Confianza\,(Regla\,2{:}temperatura = baja \rightarrow humedad = normal) = \frac{4}{4} = 1$$

$$Confianza\,(Regla\,3{:}temperatura = media \rightarrow humedad = alta) = \frac{4}{6} = 0.67$$

$$Soporte\,(Regla\,1{:}ambiente = nublado \rightarrow jugar = sí) = \frac{4}{14} = 0.29$$

$$Soporte\,(Regla\,2{:}temperatura = baja \rightarrow humedad = normal) = \frac{4}{14} = 0.29$$

$$Soporte\,(Regla\,3{:}temperatura = media \rightarrow humedad = alta) = \frac{4}{14} = 0.29$$

Para los casos de la regla 1 y la regla 2 la confianza es 1, ya que todos los ejemplos que satisfacen el antecedente de las reglas satisfacen también el consecuente. Sin embargo, para el caso de la regla 3, no siempre que se tiene una temperatura = media , se tiene una humedad = alta y, por lo tanto, la confianza de esa regla es menor.

De los seis ejemplos que cumplen el antecedente de la regla 3, hay dos que no cumplen el consecuente. Respecto a la medida de soporte, todas las reglas presentan la misma medida ya que antecedente y consecuente se da en el mismo número de ejemplos.

Este ejemplo contiene pocos datos, pero, cuando se manejan grandes cantidades de datos, el número de posibles asociaciones puede ser muy elevado. **Por lo tanto, se establecen valores mínimos de confianza y soporte para considerar cuáles de las reglas aprendidas son relevantes.**

Es importante tener en cuenta tanto la confianza como el soporte, ya que una regla puede tener una confianza con valor 1 y, sin embargo, representar una relación rara o inhabitual, con lo cual no es relevante.

Además, existen otras métricas que nos proporcionan información adicional a la que proporcionan la confianza o el soporte. Una de estas medidas es el *lift*, que nos indica la relación entre la probabilidad de que el consecuente de la regla se cumpla si se da el antecedente y la probabilidad de que se cumpla el consecuente de la regla. Más formalmente:

$$lift(A,B) = \frac{P(B \vee A)}{P(B)} = \frac{confianza(A \rightarrow B)}{P(B)}$$

El *lift* mide la correlación entre la ocurrencia de un hecho A y un hecho B:

- Si *lift = 1*, entonces el hecho A es independiente del hecho B.
- Si *lift > 1*, entonces existe correlación entre A y B y, por lo tanto, A probablemente implica B. Nos indica que la regla es útil.
- Si *lift < 1*, entonces existe correlación negativa entre A y B y, por lo tanto, A y B se comportan de forma opuesta (A probablemente implica no B). Nos indica que la regla no es útil.

Ideas clave

Generalmente, el aprendizaje de reglas de asociación comprenderá dos fases:

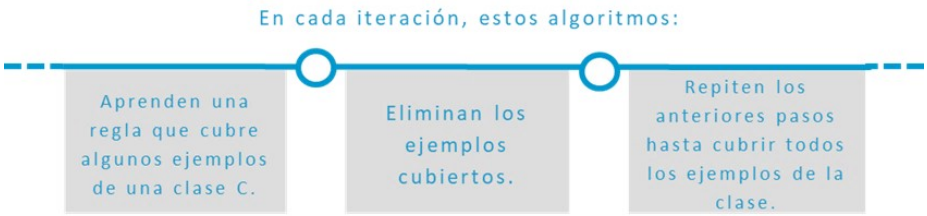
1. Encontrar aquellas reglas cuya frecuencia sea superior a un valor de soporte establecido.
2. De las reglas extraídas en el paso 1, seleccionar aquellas cuya confianza es superior a un valor determinado.

En el apartado 4.4 de este tema se explica de forma detallada el algoritmo *apriori* como procedimiento para generar reglas de asociación.

4.3. Algoritmos de aprendizaje de reglas de clasificación

Como se ha visto con anterioridad, una forma posible de aprender reglas de clasificación es a través de la generación de un árbol de decisión en un primer paso, utilizando uno de los métodos explicados en el Tema 2, y, posteriormente, mapear el árbol generado a un conjunto de reglas equivalente, dando lugar a una regla por cada nodo hoja generado en el árbol.

En este tema se van a explicar los **algoritmos de recubrimiento secuencial** para el aprendizaje directo de conjuntos de reglas de clasificación.



Por tanto, se aprende una regla en cada iteración hasta alcanzar el conjunto final de reglas. La Figura 1 muestra un algoritmo básico de recubrimiento secuencial:

```
PROCEDIMIENTO Recubrimiento_secuencial (Clases, atributos, ejemplos)
COMIENZO
       Reglas ← {}
       Para cada clase C de Clases
       COMIENZO
              E ← Ejemplos
              Mientras (E contenga ejemplos de la clase C)
              COMIENZO
                    Regla ← AprenderUnaRegla (C, E, Atributos)
                    Reglas ← Reglas + {Regla}
                    E ← E – {Ejemplos de E clasificados correctamente por Regla}
              FIN
       FIN
       Devolver Reglas
FIN
```

Figura 1. Algoritmo básico de recubrimiento secuencial.

El algoritmo de recubrimiento secuencial se basa en el uso iterativo de un procedimiento que seleccione una única regla de buena precisión, pero sin necesidad de que cubra todos los ejemplos positivos.

El algoritmo básico presentado en la Figura 1 podría incluir una condición adicional que evalúe la calidad de la regla aprendida para considerar o descartar esta regla.

Por otro lado, la Figura 2 muestra un **algoritmo básico de aprendizaje de una regla** que realiza una búsqueda codiciosa (*greedy*), sin retroceso, que **busca de lo general a lo específico**. Dado que se trata de un método codicioso, existe el riesgo de no encontrar la mejor regla. Esto no implica que no se pueda conseguir una precisión alta, aun presentando una cobertura incompleta.

La cobertura es otra medida utilizada para evaluar el interés de las

reglas y se define como el número de ejemplos que cumplen la regla (antecedente y consecuente).

```
PROCEDIMIENTO AprenderUnaRegla (Clase, Ejemplos, Atributos)
COMIENZO
            Regla ← regla con antecedente A vacío y con consecuente Clase
            MIENTRAS (regla cubre algún ejemplo negativo AND Atributos ≠ ∅)
            COMIENZO
                    Restricciones ←{}
                    Para cada atributo A  no utilizado en la regla
                             COMIENZO
                                  Para cada valor v de A
                                  COMIENZO
                                        Restricciones ← Restricciones + {A=v}
                                  FIN
                             FIN
                    Restriccion ← mejorRestriccion (Restricciones, regla)
                    Regla ← añadir restricción al antecedente
                    Atributos ← atributos – {atributo de Restriccion}
            FIN
            Devolver Regla
FIN
```

Figura 2. Algoritmo básico de aprendizaje de una regla.

Ideas clave

El **algoritmo PRISM** (Cendrowska, 1987) es uno de los algoritmos más simples de recubrimiento secuencial. Utiliza los algoritmos básicos de recubrimiento secuencial y de aprendizaje de una regla expuestos en la Figuras 1 y en la Figura 2, respectivamente.

De forma más específica, el procedimiento MejorRestricción utilizado en el algoritmo de aprendizaje de una regla que emplea **PRISM se basa en la medida de precisión** denominada **confianza**, tal y como se ha definido previamente.

Enfatizando en su definición, la confianza vendrá dada por el cociente entre el número de ejemplos que satisfacen antecedente y consecuente y el número de ejemplos que satisfacen solo el antecedente.

Para comprender mejor el aprendizaje de una regla utilizando el algoritmo PRISM se utilizará como ejemplo el conocido problema «Jugar al aire libre» (cuyos datos están contenidos en la Tabla 1). En este sentido, la Figura 3 ilustra en forma de árbol para facilitar su comprensión, el aprendizaje de una regla para el caso de la clase jugar=no .

Desarrollemos entonces cómo se lleva a cabo la ejecución del algoritmo:

- En primer lugar, se escoge la regla más general, aquella que no tiene ninguna restricción en el antecedente.
- A continuación, se utiliza la medida de la confianza para seleccionar la mejor restricción de todas las restricciones posibles. El cálculo de esta medida, para el ejemplo seguido, se muestra en la Figura 3 entre paréntesis junto a las diferentes reglas. Se observa que el mejor valor de confianza es 0.60 y se da para el atributo ambiente y valor soleado. Por tanto, se escoge en este paso el par atributo ambiente y valor soleado.

- Se añade esta restricción a la regla, eliminando este atributo de la lista de atributos a considerar en la siguiente iteración.
- El siguiente nivel de árbol muestra la siguiente iteración del algoritmo. Se calcula la confianza para cada una de las posibles restricciones y se encuentra que la confianza es igual a 1 tanto para el atributo temperatura y valor alta como para el atributo humedad y valor alta.
- Cuando se obtiene un empate entre restricciones, se escoge la restricción de mayor cobertura. Por tanto, en este paso se escoge el par atributo humedad y valor alta porque tiene cobertura 3 frente a la cobertura 2 del par atributo temperatura y valor alta.
- Se añade la restricción seleccionada a la regla y se elimina este atributo de la lista de atributos a considerar en la siguiente iteración.
- En la siguiente iteración no se cumple la condición de que existan ejemplos negativos cubiertos por la regla. Entonces, el algoritmo devuelve la regla obtenida mediante las dos iteraciones descritas en los pasos anteriores. En concreto, la regla devuelta es la siguiente:

SI ambiente=soleado AND humedad=alta

ENTONCES jugar=no

ENTONCES jugar=no

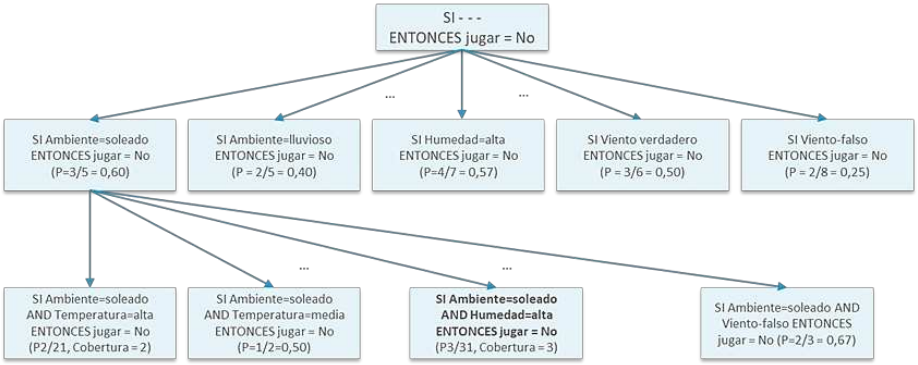


Figura 3. Representación gráfica del aprendizaje de una regla mediante el algoritmo básico para el ejemplo de la Tabla 1 y la clase jugar=no.

Mediante la llamada a este procedimiento de aprendizaje de una regla, PRISM va obteniendo incrementalmente el conjunto de reglas de clasificación para todas las clases existentes.

Otros algoritmos de reglas de clasificación

Existen otros algoritmos de reglas de clasificación además de PRISM, entre ellos:

- ***OneRule* y *ZeroRule***: son los algoritmos más simples de reglas de clasificación para un conjunto de ejemplos (Holte, 1993). Estos algoritmos generan un árbol de decisión expresado mediante reglas de un solo nivel. Los algoritmos predicen simplemente la clase principal clasifican a través suyo.

- *ZeroRule* asigna un valor único de probabilidad a todas las instancias utilizando la media o la moda de la clase de salida, dependiendo de si trabaja con variables numéricas o nominales. La salida sería la clase más probable.

- *OneRule* utiliza particiones de un solo atributo y asigna valores a las instancias que tienen ese atributo, basándose en la media o la moda, al igual que hace *ZeroRule*, para asignar un valor único de probabilidad a todas las instancias de ese conjunto. De forma más sencilla, para cada valor de atributo el algoritmo calculará la

probabilidad para cada clase.

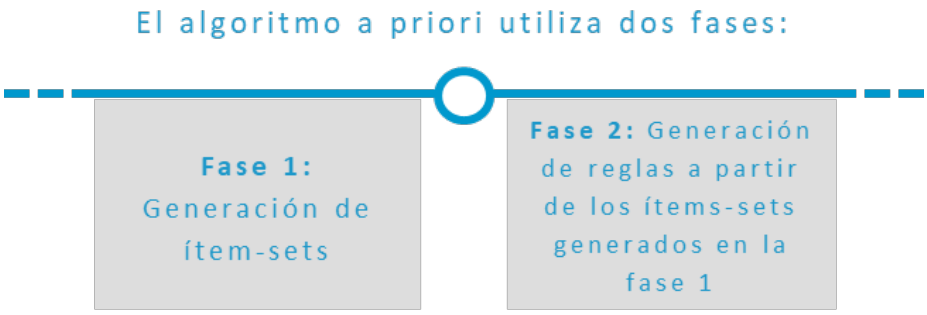
Tomando como errores las menores probabilidades se calculará el error total de cada atributo. Las reglas que se formen vendrás dadas por el atributo con un menor error total.

- **RIPPER** *(Repeated Incremental Pruning Produce Error Reduction)* (Cohen, 1995): algoritmo de reglas de clasificación basado en el algoritmo IREP (*Incremental Reduced Error Pruning)* (Fürnkranz & Widmer, 1994), basado, a su vez, en la técnica REP (*Reduced Error Pruning*) (Bagallo & Haussler, 1990) y en el algoritmo de aprendizaje de reglas *Separate-And-Conquer*.

4.4. Algoritmos de aprendizaje de reglas de asociación

Existen diversos algoritmos para generar reglas de asociación. En esta sección se profundiza en uno de los algoritmos más populares que se denomina **algoritmo *apriori*** (Agrawal et al, 1993).

El algoritmo apriori pretende generar ítem-sets que cumplan una cobertura mínima de manera eficiente. Un ítem es un par atributo-valor mientras que un ítem-set es un conjunto de pares atributo-valor. Un k-ítem-set es un conjunto de k pares atributo-valor. La cobertura de un ítem-sets se refiere al número de instancias que cumplen los valores en el ítem-set y va a determinar la cobertura de las reglas generadas a partir de dicho ítem-set.



Mediante un ejemplo se explica a continuación el funcionamiento de este algoritmo. El primer paso es determinar una cobertura mínima, por ejemplo, 3. Seguidamente, se comienza a trabajar con *ítem-sets* de 1 par atributo-valor, escogiendo aquellos que cumplen como mínimo la cobertura escogida. Para el ejemplo del problema del tiempo de la Tabla 1 se tiene por tanto en la primera iteración del algoritmo los ítem sets de 1 elemento con cobertura superior o igual a 3, tal y como se muestra en la Tabla 2.

En este caso concreto todos los *ítem-sets* de 1 elemento cumplen la condición de la cobertura.

Ítem-sets de 1 elemento	Cobertura
Ambiente soleado	5
Ambiente nublado	4
Ambiente lluvioso	5
Temperatura=alta	4
Temperatura=media	6
Temperatura=baja	4
Humedad-alta	7
Humedad=normal	7
Viento=falso	8
Viento-verdadero	6
Jugar=si	9
Jugar=no	5

Tabla 2. *Ítem-sets* de 1 elemento del problema «Jugar al aire libre».

En la siguiente iteración el algoritmo combina los *ítem-sets* encontrados en la primera iteración para generar *ítem-sets* de 2 elementos que cumplan la condición de cobertura igual o superior a 3. Estos *ítem-sets* se incluyen en la Tabla 3.

Ítem-sets de 2 elementos	Cobertura
Ambiente lluvioso, temperatura=media	3
Ambiente lluvioso, humedad=normal	3
Ambiente soleado, humedad=alta	3
Ambiente lluvioso, viento=falso	3
Ambiente soleado, viento=falso	3
Ambiente lluvioso, jugar=si	3
Ambiente=nublado, jugar=si	4
Ambiente=soleado, jugar=no	3
Temperatura=alta, humedad=alta	3
Temperatura=baja, humedad=normal	4
Temperatura=media, humedad=alta	4
Temperatura=alta, viento=falso	3
Temperatura=media, viento=falso	3
Temperatura=media, viento verdadero	3
Temperatura=baja, jugar=sí	3
Temperatura=media, jugar=sí	4
Humedad=alta, viento=falso	4
Humedad=alta, viento verdadero	3
Humedad=normal, viento=falso	4
Humedad=normal, viento verdadero	3
Humedad=normal, jugar=si	6
Humedad=alta, jugar=si	3
Humedad=alta, jugar=no	4
Viento=falso, jugar=si	6
Viento verdadero, jugar=si	3
Viento-verdadero, jugar=no	3

Tabla 3. *Ítem-sets* de 2 elementos del problema «Jugar al aire libre».

En la siguiente iteración se parte de los *ítem-sets* encontrados de 2 elementos para generar *ítem-sets* de 3 elementos. Por ejemplo, si se consideran las dos primeras entradas de la Tabla 3 se puede generar el siguiente *ítem-set* de 3 elementos:

```
Ambiente=lluvioso, temperatura=media, humedad=normal
```

Existe un único ejemplo que cumple este ítem-set, luego, al no cumplir la cobertura mínima, no se añade a la tabla de ítem-sets de 3 elementos. Así, se procede combinando el resto de las entradas e incluyendo en la tabla aquellas combinaciones que cumplen la condición de cobertura mínima, obteniendo los *ítem-sets* presentes en la Tabla 4.

Ítem-sets de 3 elementos	Cobertura
Ambiente soleado, humedad alta, jugar=no	3
Ambiente lluvioso, viento=falso, jugar=si	3
Temperatura=baja, humedad=normal, jugar=si	3
Humedad=normal, viento=falso, jugar=si	4

Tabla 4. Ítem-sets de 3 elementos del problema «Jugar al aire libre».

El siguiente paso es obtener *ítem-sets* de 4 elementos que cumplan la condición de cobertura de 3 a partir de aquellos presentes en la Tabla 4. En este paso ya no se obtiene ningún *ítem-set* que cumpla dicha condición.

Una vez obtenidos los *ítem-sets*, se procede a la siguiente fase del algoritmo que consiste en generar las reglas de asociación a partir de los *ítem-sets* encontrados de 3 y 2 elementos. De las reglas generadas, se descartan aquellas reglas que no superan un mínimo valor de confianza.

Por ejemplo, se establece un valor de confianza de 0.9. Para el *ítem-set* (Ambiente=soleado, humedad=alta, jugar=no) se pueden generar las reglas siguientes:

```
SI ambiente = soleado

ENTONCES humedad = alta AND jugar = no          (P=3/5=0.6)

SI ambiente = soleado AND humedad = alta
```

```
ENTONCES jugar = no                                (P=3/3=1)

SI ambiente = soleado AND jugar = no

ENTONCES humedad = alta                            (P=3/3=1)

SI humedad = alta

ENTONCES jugar = no AND ambiente = soleado         (P=3/7=0.43)

SI humedad = alta AND jugar = no

ENTONCES ambiente = soleado                        (P=3/4=0.75)

SI jugar = no

ENTONCES humedad = alta AND ambiente = soleado (P=3/5=0.6)
```

Entre paréntesis, tras cada regla, se indica el valor de la confianza. Únicamente hay dos reglas que superan el valor de confianza mínimo establecido de 0.9 y, por tanto, son las reglas que serán consideras en el conjunto de reglas final:

```
SI ambiente = soleado AND humedad = alta

ENTONCES jugar = no                              (P=3/3=1)

SI ambiente = soleado AND jugar = no

ENTONCES humedad = alta                          (P=3/3=1)
```

De la misma manera se procederá a generar las posibles reglas por cada ítem set encontrado en la primera fase, seleccionando únicamente aquellas con las que se obtiene un valor de confianza superior a 0.9. Por ejemplo, otras reglas que se generarían serían:

```
SI humedad = normal AND viento = falso
```

```
ENTONCES jugar = si                          (P=4/4=1)

SI ambiente = lluvioso AND viento falso

ENTONCES jugar = si                          (P=3/3=1)

SI temperatura = baja

ENTONCES humedad = normal                    (P=4/4=1)
```

De este algoritmo se dice que genera las reglas eficientemente, puesto que en cada fase solo tiene en cuenta un subconjunto de posibles *ítem-sets*, aquellos considerados en la iteración previa por superar una cobertura mínima, y un *ítem-set* de k elementos no va a cumplir la norma de la mínima cobertura a no ser que los *ítem-sets* de *k-1* elementos candidatos para combinar cumplan también esa condición.

De cualquier manera, para conjuntos de datos grandes este algoritmo puede suponer una alta carga computacional dependiendo de la cobertura especificada.

Por último, se debe remarcar que no siempre una regla de asociación con alta confianza y soporte resulta útil. Por ejemplo, en el caso del ejemplo de la tienda en línea mencionado previamente, si se da el hecho habitual de que los clientes que compran detergente de la lavadora también compran suavizante, esta información puede resultar poco útil a efectos de *marketing*, no siendo necesario promocionar ninguno de los dos productos.

Sin embargo, sí puede resultar útil cuando se encuentra el hecho de que asociando la venta de dos productos se vende más de un producto, o cuando se da el hecho opuesto en el que se encuentra que dos productos, si se asocian, compiten entre sí, con lo cual la regla que les asocia tiene una confianza baja.

Otros algoritmos de reglas de asociación

Existen otros algoritmos de reglas de asociación además de apriori, entre ellos:

- Part.
- FP-Growth y TD-FP-Growth.
- ECLAT (Equivalent CLAss Transformation)

PART

Algoritmo que obtiene reglas de asociación utilizando el algoritmo de árboles de decisión C4.5. PART no necesita realizar una optimización global.

FP-Growth* y *TD-FP-Growth

Para reducir el coste que la generación de los conjuntos de ítem-sets implica, Han *et al.* (2000) propusieron el algoritmo *FP-Growth*. Al igual que apriori, este el algoritmo permite obtener reglas de asociación a partir de *ítem-sets* frecuentes, pero sin generar las diferentes reglas candidatas para cada iteración de *k* elementos. Este algoritmo propone una nueva estructura de patrones frecuentes (FP – *Frequent Patterns*), extendida del árbol de prefijos, que permite almacenar toda la información de las transacciones comprimida.

La eficiencia del algoritmo se logra gracias a tres técnicas: en primer lugar, la compresión de la base de datos; en segundo lugar, limitando la generación de ítem-sets; en tercer lugar, utilizando un método de «*divide y* vencerás» para descomponer la tarea de búsqueda de patrones en varias bases de datos condicionales, reduciendo drásticamente el espacio de búsqueda (Han *et al.*, 2000). Los resultados obtenidos del análisis de cada base de datos se concatenarán en el paso final. La

versión *TD-FP-Growth* cambia el orden de búsqueda de arriba hacia abajo, en oposición al orden de abajo hacia arriba del *FP-Growth*, lo cual ahorra espacio y tiempo (Wang *et al*., 2002).

ECLAT *(Equivalent CLAss Transformation):*

Algoritmo basado en la búsqueda de *ítem-sets* frecuentes. La diferencia principal con a priori es que éste almacena las transacciones de forma horizontal (elementos que forman una transacción en la misma línea), mientras que ECLAT analiza los datos de forma vertical, conteniendo cada línea un ítem y las transacciones en las que aparece ese ítem (Zaki *et al*., 1997).

4.5. Aplicaciones y ejemplos de implementación

Algunas aplicaciones de las reglas de asociación en la literatura

El interés de las reglas de asociación es descubrir combinaciones de pares atributo-valor que ocurren frecuentemente en un conjunto de datos. De hecho, también son conocidas como técnicas de patrones de búsqueda (*pattern search*) o *association rule learning* (Fournier-Viger *et al.*, 2017). Entre sus posibles aplicaciones, una de las más interesantes es analizar los carritos de la compra, tanto en los supermercados físicos, para saber cómo distribuir los productos en las estanterías, o en las tiendas *online*.

Es decir, en una tienda *online* puede ser muy interesante conocer los productos que los clientes compran juntos, con el fin de identificar a los clientes con patrones de compra similares, y así poder predecir posibles compras futuras de estos clientes y hacer ofertas personalizadas (por ejemplo, los padres que compran pañales también compran fórmula, así como cerveza cuando no pueden salir a tomar una copa).

Otras aplicaciones similares en las que son útiles incluyen el análisis de patrones de navegación en la web. También se han empleado en aplicaciones médicas para analizar las relaciones entre la recuperación después de operaciones y la posible cronificación de secuelas (Hui *et al.* 2014).

Las reglas de clasificación en la literatura y otros algoritmos clasificadores

Tal y como se ha explicado, los árboles de decisión están muy relacionados con las técnicas de reglas de clasificación, como el algoritmo PRISM (Liu, Gegov y Cocea, 2016). Sin embargo, además de la regresión logística, los árboles de decisión clasificadores y las reglas de clasificación**, existen otros algoritmos clasificadores, incluyendo k-NN, los clasificadores Naïve Bayes y los SVM**, además de, por supuesto, las redes neuronales, con las cuales podemos resolver problemas tanto de regresión como de la clasificación, y que veremos en los siguientes temas.

Ideas clave

El ***algoritmo k-NN*** (*k nearest neighbors* o *k vecinos más cercanos*) se incluye, al igual que los árboles de decisión, en lo que se conoce como «aprendizaje no paramétrico» (Kanj *et al.* 2016). Es decir, a diferencia de algoritmos de «aprendizaje paramétrico», este tipo de método no requiere una función paramétrica predefinida ***Y*** *= f(**X**)*. Esto hace que este tipo de algoritmo sea adecuado para aquellas situaciones en las que la relación entre ***X*** e ***Y*** es demasiado compleja para ser expresada como un modelo lineal. En el algoritmo k-NN, cada instancia se representa como un vector y para clasificar o hacer una predicción sobre un dato de entrada, se toman los *k* más cercanos y se calcula la media de sus valores, si estamos trabajando con datos continuos, como el valor estimado de una casa, o su moda, si estamos trabajando con datos categóricos, como la determinación de la raza de un perro. La selección del *k* se hace por validación cruzada, eligiendo el *k* que tiene el menor error, en promedio, a lo largo de las diferentes iteraciones. Este algoritmo se utiliza como método de clasificación, como detección de fraudes (Hossain y Uddin, 2018), como método de regresión, como predicción del precio de la vivienda (Nawaz *et al.*, 2019), o para imputar los datos de entrenamiento que faltan, imputando el promedio o el modo de los vecinos en lugar de un valor que falta (Liu *et al.,* 2016).

Los clasificadores ***Naïve Bayes*** (*clasificadores Bayesianos ingenuos*) son en realidad uno de los casos más sencillos de redes bayesianas o redes de creencias (Gupta *et al.,* 2019; Li, Corchado y Prieto, 2016). Las redes bayesianas son redes acíclicas dirigidas en las que cada nodo representa un estado o condición y cada arco entre dos nodos representa la probabilidad de que, dado el estado o condición del nodo fuente, se produzca el estado del nodo destino, teniendo en cuenta el teorema de Bayes:

$$P(B|A) = \frac{P(A|B)\,P(B)}{P(A)}$$

La particularidad de los clasificadores ingenuos de Bayes es considerar una fuerte independencia entre las diferentes características. Este tipo de clasificadores han sido ampliamente utilizados como filtros antispam.

Para ello, tienen en cuenta la frecuencia con la que las diferentes palabras aparecen en los correos electrónicos deseados y en los correos spam. En este sentido, con el tiempo al final de los correos spam se introdujeron una serie de palabras comunes en los correos deseados, atacando así al clasificador Bayes ingenuo. Esto se conoce como *envenenamiento bayesiano*. Así, desde 2010, se utilizan otros tipos de algoritmos para el filtrado antispam (Bhowmick y Hazarika, 2018).

Uno de los métodos utilizados, de hecho, actualmente para el filtrado antispam es el de las **máquinas de soporte vectorial (SVM – Support Vector Machine** (Rana *et al.*, 2018). Las SVM son clasificadores basados en la idea de buscar dos líneas entre los puntos de datos de entrenamiento bidimensionales y con el máximo margen posible entre estas líneas (es decir, un problema de optimización, normalmente utilizando la optimización Lagrangiana). Cuando nuestros datos no son bidimensionales, se buscan hiperplanos en lugar de líneas. Si no es posible dibujar tal línea o hiperplano, tenemos que suavizar la condición de separación añadiendo una función de coste o pérdida, o aumentando el número de dimensiones de los datos (con términos como x^2, x^3 o incluso $cos(x)$). Las aplicaciones de las SVM incluyen la clasificación de imágenes (aunque este tipo de aplicación se lleva a cabo actualmente con redes neuronales pre-entrenadas, más adecuadas para imágenes y vídeo), el análisis de sentimientos, la clasificación de texto y contenido en redes sociales (Dang *et al.*, 2016) o la detección de *outliers* (Liu, White y Newell, 2018).

Ejemplos de implementación

En este ejemplo, vamos a aplicar el algoritmo *apriori* utilizando en esta ocasión un pequeño *dataset* en **Kaggle**, proporcionado por Shazad Udwadia (llamado «Grocery Store Data Set») con licencia de dominio público CC0 – Creative Commons 0) y en el

cual se incluyen 20 transacciones de cestas de la compra en una tienda de comestibles de diferentes productos (11 ítems posibles en total). Es un *dataset* muy pequeño, pero nos sirve para efectos ilustrativos.

En el mismo Kaggle podemos ver información sobre el contenido de los datos y algunos detalles estadísticos. Podemos descargar el *dataset* desde el siguiente enlace. Nos solicitará registrarnos si no tenemos ya una cuenta en Kaggle. Para ello podemos utilizar una cuenta de Google, correo electrónico, etc., pero no tiene coste alguno. Disponible en: https://www.kaggle.com/shazadudwadia/supermarket

Aunque Kaggle nos permite utilizar *notebooks online* bajo Python o R para trabajar con los *dataset* que comparten otros usuarios sin necesidad de disponer de un sistema Python *offline*, por el momento vamos a seguir trabajando *offline*, para así también seguir aprendiendo a gestionar las librerías de terceros.

En el enlace indicado nos mostrará, en realidad, información sobre el *dataset*, de forma similar a OpenML, y nos permitirá descargarlo a través del botón «*Download*» (o incluso crear un nuevo *notebook* basado en dichos datos). De esta forma, descargaremos un archivo " supermarket.zip ". Descomprimiéndolo obtendremos el archivo " GroceryStoreDataSet.csv " con el que vamos a trabajar. Si abrimos su contenido con cualquier editor de texto, como el propio Visual Studio Code, veremos que no tiene *header* y que cada línea representa una compra de un cliente, conteniendo la lista de artículos adquiridos.

```
"MILK,BREAD,BISCUIT"
"BREAD,MILK,BISCUIT,CORNFLAKES"
"BREAD,TEA,BOURNVITA"
"JAM,MAGGI,BREAD,MILK"
"MAGGI,TEA,BISCUIT"
"BREAD,TEA,BOURNVITA"
"MAGGI,TEA,CORNFLAKES"
"MAGGI,BREAD,TEA,BISCUIT"
"JAM,MAGGI,BREAD,TEA"
"BREAD,MILK"
"COFFEE,COCK,BISCUIT,CORNFLAKES"
"COFFEE,COCK,BISCUIT,CORNFLAKES"
"COFFEE,SUGER,BOURNVITA"
"BREAD,COFFEE,COCK"
"BREAD,SUGER,BISCUIT"
"COFFEE,SUGER,CORNFLAKES"
"BREAD,SUGER,BOURNVITA"
"BREAD,COFFEE,SUGER"
"BREAD,COFFEE,SUGER"
"TEA,MILK,COFFEE,CORNFLAKES"
```

Como CSV, en realidad, solo existe una única columna, de modo que habrá que realizar algo de tratamiento en los datos.

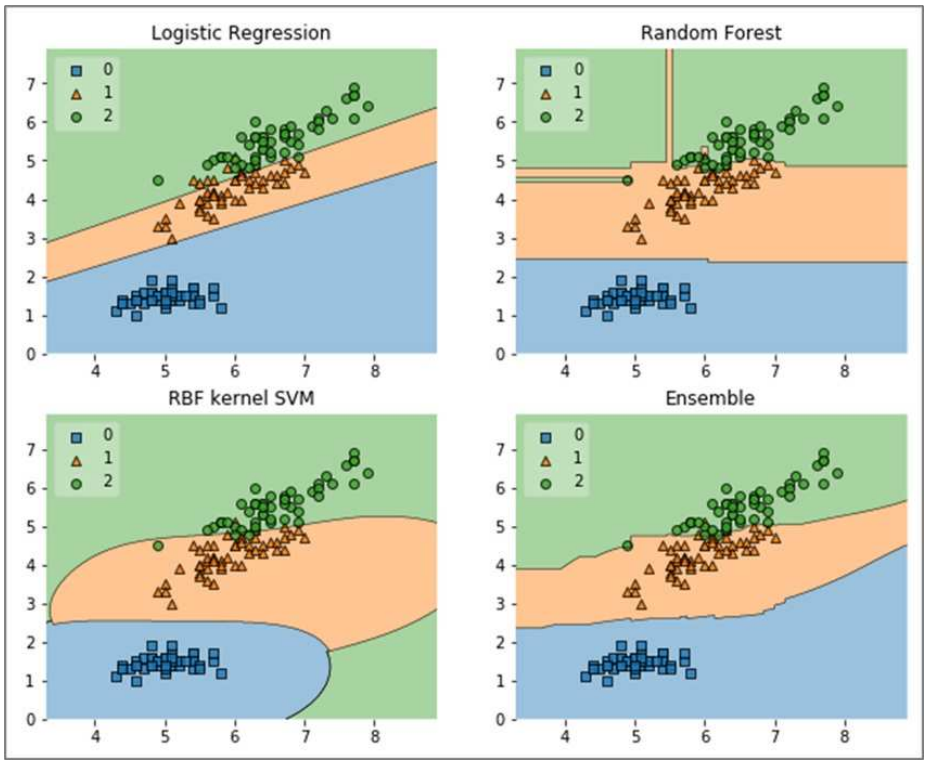
Sin embargo, antes de nada, vamos a instalar una nueva librería que no hemos usado hasta ahora, mlxtend, utilizando nuestro gestor de paquetes, por ejemplo:

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

Fuente: http://rasbt.github.io/mlxtend/

MLxtend (*Machine Learning extensions*) incluye extensiones útiles para realizar técnicas de *machine learning*, incluyendo, por ejemplo, el algoritmo *apriori*. Además, incluye interesantes ayudas para utilizar gráficos como regiones de decisión, como

se puede ver a continuación, que podemos utilizar en otro momento para comparar algoritmos de clasificación (aunque no es el caso que nos ocupa, pues ahora no estamos trabajando con clasificadores, que se encontrarían dentro del aprendizaje supervisado, sino con reglas de asociación / búsqueda de patrones, que se encontrarían dentro del aprendizaje no supervisado).



Una vez instalado el módulo, continuemos con el ejemplo que nos ocupa. En primer lugar, utilicemos este código para ver el contenido de los datos una vez cargados:

```
# Librerías necesarias
import pandas as pd
# recordad emplear aquí la ruta absoluta o relativa hasta nuestr
o dataset descargado
# en función de en que ruta estamos ejecutando nuestro script
df = pd.read_csv('./GroceryStoreDataSet.csv', names=['products']
, header=None)

print("---df---")
print(df)
print("---df.columns---")
print(df.columns)
print("---df.values---")
print(df.values)
```

Resultando la siguiente salida:

```
---df---
                          products
0               MILK,BREAD,BISCUIT
1    BREAD,MILK,BISCUIT,CORNFLAKES
2              BREAD,TEA,BOURNVITA
3             JAM,MAGGI,BREAD,MILK
4               MAGGI,TEA,BISCUIT
5              BREAD,TEA,BOURNVITA
6             MAGGI,TEA,CORNFLAKES
7          MAGGI,BREAD,TEA,BISCUIT
8              JAM,MAGGI,BREAD,TEA
9                      BREAD,MILK
10 COFFEE,COCK,BISCUIT,CORNFLAKES
11 COFFEE,COCK,BISCUIT,CORNFLAKES
12          COFFEE,SUGER,BOURNVITA
13              BREAD,COFFEE,COCK
14             BREAD,SUGER,BISCUIT
15         COFFEE,SUGER,CORNFLAKES
16           BREAD,SUGER,BOURNVITA
```

```
17              BREAD,COFFEE,SUGER

18              BREAD,COFFEE,SUGER

19      TEA,MILK,COFFEE,CORNFLAKES

---df.columns---

Index(['products'], dtype='object')

---df.values---

[['MILK,BREAD,BISCUIT']

['BREAD,MILK,BISCUIT,CORNFLAKES']

['BREAD,TEA,BOURNVITA']

['JAM,MAGGI,BREAD,MILK']

['MAGGI,TEA,BISCUIT']

['BREAD,TEA,BOURNVITA']

['MAGGI,TEA,CORNFLAKES']

['MAGGI,BREAD,TEA,BISCUIT']

['JAM,MAGGI,BREAD,TEA']

['BREAD,MILK']

['COFFEE,COCK,BISCUIT,CORNFLAKES']

['COFFEE,COCK,BISCUIT,CORNFLAKES']

['COFFEE,SUGER,BOURNVITA']

['BREAD,COFFEE,COCK']
```

```
['BREAD,SUGER,BISCUIT']

['COFFEE,SUGER,CORNFLAKES']

['BREAD,SUGER,BOURNVITA']

['BREAD,COFFEE,SUGER']

['BREAD,COFFEE,SUGER']

['TEA,MILK,COFFEE,CORNFLAKES']]
```

Preprocesamos los datos para que puedan ser usados por mlxtend, primero separando los elementos por las comas y posteriormente utilizando el objeto TransactionEncoder .

```
# Librerías necesarias
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

# recordad emplear aquí la ruta absoluta o relativa hasta nuestr
o dataset descargado
# en función de en que ruta estamos ejecutando nuestro script
df = pd.read_csv('./GroceryStoreDataSet.csv', names=['products']
, header=None)

# preprocesamos los datos usando las comas como separadores
data = list(df["products"].apply(lambda x:x.split(',')))
print("---data---")
print(data)
print()

# convertimos los datos a un formato que entienda mlxtend
te = TransactionEncoder()
te_data = te.fit(data).transform(data)
df = pd.DataFrame(te_data,columns=te.columns_)
print("---df.head()---")
print(df.head())
print()
```

Obteniendo la salida (hemos reducido la segunda salida para que se vea mejor como una tabla):

```
---data---
```

```
[['MILK', 'BREAD', 'BISCUIT'],
['BREAD', 'MILK', 'BISCUIT', 'CORNFLAKES'],
['BREAD', 'TEA', 'BOURNVITA'],
['JAM', 'MAGGI', 'BREAD', 'MILK'],
['MAGGI', 'TEA', 'BISCUIT'],
['BREAD', 'TEA', 'BOURNVITA'],
['MAGGI', 'TEA', 'CORNFLAKES'],
['MAGGI', 'BREAD', 'TEA', 'BISCUIT'],
['JAM', 'MAGGI', 'BREAD', 'TEA'],
['BREAD', 'MILK'],
['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
['COFFEE', 'COCK', 'BISCUIT', 'CORNFLAKES'],
['COFFEE', 'SUGER', 'BOURNVITA'],
['BREAD', 'COFFEE', 'COCK'],
['BREAD', 'SUGER', 'BISCUIT'],
['COFFEE', 'SUGER', 'CORNFLAKES'],
['BREAD', 'SUGER', 'BOURNVITA'],
['BREAD', 'COFFEE', 'SUGER'],
['BREAD', 'COFFEE', 'SUGER'],
['TEA', 'MILK', 'COFFEE', 'CORNFLAKES']]
```

```
---df.head()---

   BISCUIT BOURNVITA BREAD   COCK COFFEE CORNFLAKES    JAM MAGGI   MILK
SUGER    TEA

0     True      False   True False   False       False False False   True
False False

1     True      False   True False   False        True False False   True
False False

2    False       True   True False   False       False False False False
False   True

3    False      False   True False   False       False   True   True   True
False False

4     True      False False False   False       False False   True False
False   True
```

Apliquemos ahora el algoritmo *apriori* y veamos el resultado de las asociaciones ordenadas de forma descendente, en base a su soporte:

```
# Librerías necesarias
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

# recordad emplear aquí la ruta absoluta o relativa hasta nuestr
o dataset descargado
# en función de en que ruta estamos ejecutando nuestro script
df = pd.read_csv('./GroceryStoreDataSet.csv', names=['products']
, header=None)

# preprocesamos los datos usando las comas como separadores
data = list(df["products"].apply(lambda x:x.split(',')))

# convertimos los datos a un formato que entienda mlxtend
te = TransactionEncoder()
te_data = te.fit(data).transform(data)
df = pd.DataFrame(te_data,columns=te.columns_)

# procedemos a aplicar el algoritmo apriori
df1 = apriori(df, min_support=0.01, use_colnames=True)
df1 = df1.sort_values(by="support",ascending=False)
print(df1)
```

```
    support                        itemsets
2      0.65                         (BREAD)
4      0.40                        (COFFEE)
0      0.35                       (BISCUIT)
10     0.35                           (TEA)
5      0.30                     (CORNFLAKES)
..      ...                             ...
55     0.05      (CORNFLAKES, MILK, BISCUIT)
57     0.05        (SUGER, BREAD, BOURNVITA)
17     0.05                 (SUGER, BISCUIT)
```

```
37      0.05              (CORNFLAKES, MAGGI)

82      0.05 (COFFEE, MILK, TEA, CORNFLAKES)
```

Por último, creemos una función para predecir el siguiente elemento que escogerá con mayor probabilidad un cliente a su cesta en función del estado actual de elementos en la misma:

```
# Librerías necesarias
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

# recordad emplear aquí la ruta absoluta o relativa hasta nuestr
o dataset descargado
# en función de en que ruta estamos ejecutando nuestro script
df = pd.read_csv('./GroceryStoreDataSet.csv', names=['products']
, header=None)

# preprocesamos los datos usando las comas como separadores
data = list(df["products"].apply(lambda x:x.split(',')))

# convertimos los datos a un formato que entienda mlxtend
te = TransactionEncoder()
te_data = te.fit(data).transform(data)
df = pd.DataFrame(te_data,columns=te.columns_)

# procedemos a aplicar el algoritmo apriori
df1 = apriori(df, min_support=0.01, use_colnames=True)
df1 = df1.sort_values(by="support",ascending=False)

# función auxiliar
def check_in_list(ruleitem, basketitem):
    ret = all(t in list(ruleitem) for t in basketitem)
    return ret
```

```
# función para predecir el próximo elemento más probable que int
roducirá el cliente en la cesta
def next_item(basketitems):
    if basketitems is None:
        return df1["itemsets"][0]
    max_len_apriori=len(basketitems) + 1
    # lista de solo 1 item:
    df2 = apriori(df,min_support=0.01, max_len=max_len_apriori,
use_colnames=True)
    df2 = df2[df2["itemsets"].apply(len) > max_len_apriori-1]
    df2["check_in_list"] = df2["itemsets"].apply(check_in_list,
args=(basketitems,))
    df2 = df2[df2["check_in_list"] == True]
    df2 = df2.sort_values(by="support",ascending=False)

    if(len(df2) > 0):
        return list(df2["itemsets"])[0]

# Probamos ahora a predecir el próximo valor en la cesta en func
ión de los items actuales en ella:

# partimos de una cesta vacía
item = next_item(None)
print(list(item))
#>['BISCUIT']

# añadimos ese elemento a la cesta y predecimos el segundo:
item = next_item(["BISCUIT"])
print(list(item))
#>['BREAD', 'BISCUIT']

# añadimos ese elemento a la cesta y predecimos el tercero:
item = next_item(["BREAD", "BISCUIT"])
print(list(item))
#>['MILK', 'BREAD', 'BISCUIT']
```

Obviamente podemos elegir un estado de la cesta aleatorio, no hay necesidad de seguir esta secuencia.

4.6. Referencias bibliográficas

Bagallo, G. & Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, *5*(1), 71-99. Disponible en https://doi.org/10.1007/BF00115895

Bhowmick, A. & Hazarika, S. M. (2018). E-mail spam filtering: a review of techniques and trends. In *Advances in Electronics, Communication and Computing* (pp. 583-590). Singapore: Springer.

Cendrowska, J. (1987). PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, *27*(4), 349-370. Disponible en: https://doi.org/10.1016/S0020-7373(87)80003-2

Cohen, W. W. (1995). *Fast Effective Rule Induction*. In Proceedings of the Twelfth International Conference on Machine Learning, 115–123.

Dang, N. C., De la Prieta, F., Corchado, J. M. & Moreno, M. N. (2016, June). *Framework for retrieving relevant contents related to fashion from online social network data*. In International Conference on Practical Applications of Agents and Multi-Agent Systems (pp. 335-347). Cham: Springer.

Fournier-Viger, P., Lin, J. C. W., Vo, B., Chi, T. T., Zhang, J. & Le, H. B. (2017). A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *7*(4), e1207.

Fürnkranz, J. & Widmer, G. (1994). Incremental Reduced Error Pruning. En W. W. Cohen & H. Hirsh (Eds.), *Machine Learning Proceedings 1994* (pp. 70-77). Morgan Kaufmann. Disponible en https://doi.org/10.1016/B978-1-55860-335-6.50017-9

Gupta, A., Slater, J. J., Boyne, D., Mitsakakis, N., Béliveau, A., Druzdzel, M. J. & Arora, P. (2019). Probabilistic Graphical Modeling for Estimating Risk of Coronary Artery Disease: Applications of a Flexible Machine-Learning Method. *Medical*

Decision Making, *39*(8), 1032-1044.

Han, J., Pei, J. & Yin, Y. (2000). Mining frequent patterns without candidate generation. Association for Computing Machinery. Disponible en https://doi.org/10.1145/335191.335372

Holte, R. C. (1993). Very Simple Classification Rules Perform Well on Most Commonly Used Datasets. *Machine Learning*, *11*(1), 63-90. Disponible en https://doi.org/10.1023/A:1022631118932

Hossain, M. A. & Uddin, M. N. (2018, October). *A Differentiate Analysis for Credit Card Fraud Detection*. In 2018 International Conference on Innovations in Science, Engineering and Technology (ICISET) (pp. 328-333). IEEE.

Hui, L., Shih, C. C., Keh, H. C., Yu, P. Y., Cheng, Y. C. & Huang, N. C. (2014, May). *The application of association rules in clinical disease: the relationship between recovery after operation of endovascular aneurysm repairing and chronic*. In Pacific-Asia Conference on Knowledge Discovery and Data Mining (pp. 712-721). Cham: Springer.

Kanj, S., Abdallah, F., Denoeux, T. & Tout, K. (2016). Editing training data for multi-label classification with the k-nearest neighbor rule. *Pattern Analysis and Applications*, *19*(1), 145-161.

Li, T., Corchado, J. M., & Prieto, J. (2016). *Convergence of distributed flooding and its application for distributed Bayesian filtering*. IEEE Transactions on Signal and Information Processing over Networks, 3(3), 580-591.

Liu, C., White, M. & Newell, G. (2018). Detecting outliers in species distribution data. *Journal of Biogeography*, *45*(1), 164-176.

Liu, H., Gegov, A. & Cocea, M. (2016). Rule-based systems: a granular computing perspective. *Granular Computing*, *1*(4), 259-274.

Liu, Z. G., Pan, Q., Dezert, J. & Martin, A. (2016). Adaptive imputation of missing values for incomplete pattern classification. *Pattern Recognition*, 52, 85-95.

Nawaz, M., Javaid, N., Mangla, F. U., Munir, M., Ihsan, F., Javaid, A. & Asif, M. (2019, July). *An Approximate Forecasting of Electricity Load and Price of a Smart Home Using Nearest Neighbor.* In Conference on Complex, Intelligent, and Software Intensive Systems (pp. 521-533). Cham: Springer.

Rana, S. P., Prieto, J., Dey, M., Dudley, S. & Corchado, J. M. (2018). A Self Regulating and Crowdsourced Indoor Positioning System through Wi-Fi Fingerprinting for Multi Storey Building. *Sensors*, *18*(11), 3766.

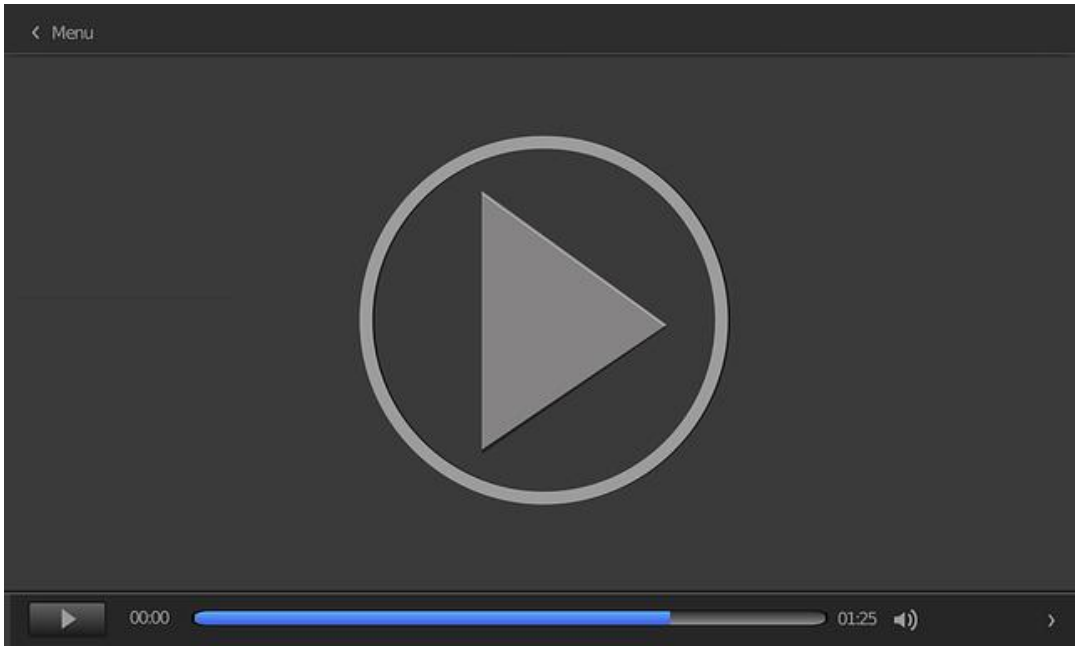
Ruiz, A. (2016, mayo 23). Representación del Conocimiento [Educación]. Disponible en https://es.slideshare.net/Alva_Ruiz/representacin-del-conocimiento-62308123

Wang, K., Tang, L., Han, J., & Liu, J. (2002). Top Down FP-Growth for Association Rule Mining. En M.S. Chen, P. S. Yu, & B. Liu (Eds.), *Advances in Knowledge Discovery and Data Mining* (pp. 334-340). Springer. Disponible en https://doi.org/10.1007/3-540-47887-6_34

Zaki, M. J., Parthaasarathy, S., Ogihara, M. & Li, W. (1997). *New Algorithms for Fast Discovery of Association Rules*. In 3rd Intl. Conf. on Knowledge Discovery and Data Mining, 283–286.

Aprendizaje de reglas de clasificación y asociación con la herramienta Weka

En esta lección magistral se mostrará cómo se puede utilizar Weka para obtener reglas de clasificación y de asociación a partir de un conjunto de datos empleando los algoritmos explicados en este tema.



04. Aprendizaje de reglas de clasificación y asociación con la herramienta Weka

Accede al vídeo:

https://unir.cloud.panopto.eu/Panopto/Pages/Embed.aspx?id=b26c847e-1c49-4d8c-a7d8-aff800f9df21

La contribución de las reglas de asociación a la minería de datos

De Moya Amaris, M.E. & Rodríguez Rodríguez, J.E. (2003). La contribución de las reglas de asociación a la minería de datos. *Tecnura,* 7(13), 94-109. http://revistas.udistrital.edu.co/ojs/index.php/Tecnura/article/view/6175.

El artículo comienza con una introducción a la minería de datos y a las reglas de asociación, incluyendo ejemplos ilustrativos. Explica cómo se generan las reglas de asociación mediante el algoritmo *apriori,* incluyendo un pseudocódigo del mismo. Además, profundiza en el tema abordando las reglas de asociación multinivel.

Curva ROC

Accede al artículo desde el aula virtual o a través de la siguiente dirección web: http://es.wikipedia.org/wiki/Curva_ROC.

Uno de los datos de evaluación que muestra Weka en las salidas de los algoritmos de clasificación es el área ROC. El artículo disponible en Wikipedia sobre la curva ROC explica concisa y claramente en qué consiste esta curva y para qué se utiliza. Además, incluye muchas referencias y enlaces de interés relacionados.

Talleres de aprendizaje estadístico

Accede a los talleres desde el aula virtual o a través de las siguientes direcciones web:

https://docplayer.es/44657219-Taller-1-grupo-9-tecnicas-de-aprendizaje-estadistico-profesora-claudia-jimenez-r.html

https://docplayer.es/66175720-Tecnicas-de-aprendizaje-estadistico-taller-1.html

Para profundizar en el conocimiento de las técnicas de aprendizaje estadístico se recomienda la lectura de los siguientes talleres.

1. Si en un problema se desea identificar los síntomas correspondientes a tres enfermedades conocidas, las técnicas apropiadas para resolver el problema son (selecciona las opciones adecuadas):

A. Árboles de decisión.

B. Algoritmo apriori.

C. Algoritmo de recubrimiento secuencial.

D. Algoritmo PRISM.

2. Si en un problema se desea identificar relaciones entre síntomas de personas que presentan ciertas enfermedades, las técnicas apropiadas son:

A. Árboles de decisión.

B. Algoritmo apriori.

C. Algoritmo de recubrimiento secuencial.

D. Algoritmo PRISM.

3. Indica cuáles de las siguientes afirmaciones son verdaderas:

A. Las reglas de clasificación predicen la clase.

B. Las reglas de asociación predicen combinaciones de atributos o la propia clase.

C. Los algoritmos que aprenden reglas de asociación buscan combinaciones de pares atributo-valor que ocurren con cierta frecuencia.

D. Las reglas de asociación tienen el mismo objetivo que las reglas de clasificación.

4. Si se quiere conocer el porcentaje de ejemplos que cumplen una regla respecto del total de ejemplos, se ha de aplicar la medida de:

A. Cobertura.

B. Soporte.

C. Confianza.

D. Cubierta.

5. Si se quiere conocer el porcentaje de ejemplos que cumplen una regla respecto de todos los ejemplos que sólo cumplen el antecedente, se ha de aplicar la medida de:

A. Cobertura.

B. Soporte.

C. Confianza.

D. Cubierta.

6. ¿Cuáles de los siguientes algoritmos se puede emplear para el aprendizaje de reglas de clasificación?

A. PRISM.

B. C4.5.

C. Apriori.

D. ID3.

7. Indica cuál de las siguientes afirmaciones es correcta:

A. No es posible mapear árboles de decisión a reglas de clasificación.

B. Los algoritmos de recubrimiento secuencial aprenden una regla en cada iteración.

C. En cada iteración, el algoritmo de recubrimiento secuencial exige que la regla cubra todos los ejemplos positivos.

D. Apriori es un algoritmo de recubrimiento secuencial.

8. Indica cuáles de las siguientes afirmaciones son correctas:

A. El procedimiento básico, para aprender una regla, utilizado en los algoritmos de recubrimiento secuencial tiene como parámetro el conjunto de todas las clases.

B. El procedimiento, básico para aprender una regla, utilizado en los algoritmos de recubrimiento secuencial añade a la regla un único par atributo-valor en cada iteración.

C. El procedimiento de recubrimiento secuencial devuelve una única regla.

D. El algoritmo de recubrimiento secuencial elimina los ejemplos cubiertos por la regla generada en cada iteración.

9. Indica cuáles de las siguientes afirmaciones son correctas respecto al algoritmo PRISM:

A. Es un algoritmo de recubrimiento secuencial.

B. Utiliza la medida de precisión o confianza para generar las reglas.

C. Parte de la regla más específica alcanzando la más general.

D. Es un algoritmo de generación de conjuntos de reglas de los más simples.

10. Indica cuáles de las siguientes afirmaciones son verdaderas respecto al algoritmo apriori:

A. Genera ítem-sets.

B. Utiliza la medida de confianza para evaluar las reglas obtenidas.

C. No genera reglas sino ítem-sets.

D. Valora los ítem-sets generados mediante una medida de confianza.