



# ANÁLISIS EXPLORATORIO DE DATOS

EXTRACCIÓN DE REGLAS

**Bruno Díaz**  
**Mónica Rodríguez**  
**Karina López**

## Contenido

CONTEXTO E INTRODUCCIÓN A LA EXTRACCIÓN DE REGLAS .....	3
REGLAS DE CLASIFICACIÓN.....	5
ALGORITMOS DE CLASIFICACIÓN.....	6
ALGORITMO ZERO-R.....	7
ALGORITMO ONE-R (ONE RULE) .....	9
ALGORITMO PART .....	12
ALGORITMO JRIP (RIPPER).....	18
ALGORITMO CONJUNCTIVE RULES .....	22
ALGORITMO NNGE.....	25
DTNB (DECISION TABLES AND NAIVE BAYES) .....	32
REGLAS DE ASOCIACIÓN .....	35
ALGORITMOS DE APRENDIZAJE DE REGLAS DE ASOCIACIÓN.....	41
ALGORITMO APRIORI .....	42
VARIANTES DEL ALGORITMO APRIORI .....	46
EJEMPLOS DE EJECUCIÓN DEL ALGORITMO A PRIORI .....	48
EJEMPLO A PRIORI CON WEKA .....	49
EJEMPLO A PRIORI CON PAQUETE ARULES DE R .....	54
ANEXOS .....	57
ANEXO 1 : DESCRIPCIÓN BASE DE DATOS CREDIT SCORING .....	57
ANEXO 2: SCRIPT DE EJECUCIÓN DEL APRIORI CON R.....	59
BIBLIOGRAFÍA .....	60

## CONTEXTO E INTRODUCCIÓN A LA EXTRACCIÓN DE REGLAS

El proceso de extracción de conocimiento tiene como objetivo el descubrimiento de nueva información a partir de bases de datos. Este proceso es una secuencia de varias fases:

- Preparación de datos
- Minería de datos
- Evaluación
- Difusión
- Uso de Modelos

Este proceso es iterativo, ya que los resultados de una fase puede conllevar a retroceder a otra anterior y para conseguir buenos resultados puede que haya que repetir varias veces el proceso. Además, la extracción de conocimiento es un proceso interactivo ya que es necesaria la participación del usuario o de un experto para ayudar en la preparación de los datos, validación del conocimiento...

Nosotros nos vamos a centrar en la fase de minería de datos, la cuál es la más característica de este proceso. El objetivo de esta fase es producir nuevo conocimiento que pueda utilizar el usuario para lo cual se construye un modelo basado en los datos recopilados. Este modelo es una descripción de los patrones y las relaciones entre los datos. Para poder construir dicho modelo hay que elegir entre los diferentes algoritmos y tipos de modelos y determinar el tipo de tarea de minería que se desea realizar.

Los tipos de tareas principales de la minería de datos son:

Clasificación. Todas las instancias de la base de datos pertenecen a una clase, la cual se indica mediante el valor de un atributo de clase. El objetivo es predecir la clase de las nuevas instancias, de las cuales no se conoce la clase, a partir del resto de los atributos. Probablemente sea la tarea más utilizada.

Regresión. Consiste en obtener una función que asigna a cada instancia un valor real. La principal diferencia con la clasificación es que el valor a predecir es numérico.

Agrupamiento. Consiste en obtener grupos a partir de los datos. La principal diferencia con la clasificación es que en el agrupamiento en vez de analizar los datos etiquetados con una clase los analiza para generar dicha etiqueta. Lo que se intenta es crear grupos compuestos por instancias que se parezcan lo máximo posible, al mismo tiempo que sean muy diferentes de las instancias de los otros grupos.

Correlaciones. Se usa para examinar el grado similitud de los valores de dos variables numéricas. Lo que se desea estudiar es lo que ocurre con el valor de una de las variables cuando varía el valor de la otra.

Reglas de asociación. El objetivo de las reglas de asociación es encontrar relaciones no explícitas entre atributos categóricos. La formulación más común de dichas reglas es del estilo “si el atributo X toma el valor d entonces el atributo Y toma el valor b”.

## REGLAS DE CLASIFICACIÓN

La clasificación es probablemente la tarea más utilizada en la minería de datos. En ella, cada instancia pertenece a una clase, la cual se indica mediante el valor de un atributo que llamamos la clase de la instancia. Este atributo puede tomar diferentes valores discretos, cada uno de los cuales corresponde a una clase. El resto de los atributos de la instancia se utilizan para predecir la clase. Es decir, el objetivo va a ser predecir la clase de las instancias que se desconoce la clase.

La razón de precisión de un algoritmo de clasificación es el cociente entre las predicciones correctas y el número total de predicciones. Por tanto el objetivo del algoritmo de clasificación será maximizar la razón de precisión.

Hay muchas técnicas de la minería de datos que nos permiten hacer clasificación como por ejemplo las redes neuronales o los árboles de decisión. En el siguiente apartado entraremos en detalle en algunos de estos algoritmos.

## ALGORITMOS DE CLASIFICACIÓN

Se va a proceder a la presentación de los diferentes algoritmos de clasificación que aparecen en el software Weka (*Waikato Environment for Knowledge Analysis*). Éste, es un software de libre distribución para aprendizaje automático y minería de datos desarrollado por la Universidad de Waikako (Nueva Zelanda).

Weka soporta varias tareas estándar de minería de datos, como son el preprocesamiento de datos, la clasificación, el clustering, las reglas de asociación, la selección de atributos, la regresión y la visualización. En este caso, se procederá a una descripción teórica de los algoritmos de clasificación contenidos en Weka, complementada con una descripción detallada de los diferentes parámetros de cada algoritmo. De la misma forma, la utilización de los algoritmos se ilustrará con el análisis de unos datos reales sobre un problema de credit scoring (archivo credit.xls). Esta base de datos se describe en el anexo número uno.

Los distintos algoritmos serán ejecutados sobre los datos anteriores, intentando introducir modificaciones relevantes en los correspondientes parámetros de los algoritmos que repercutan en una mayor eficacia en la clasificación, para finalmente mostrar la salida producida por el programa Weka. Como parte final, se mostrará una tabla comparativa de los resultados de la mejor ejecución de cada algoritmo sobre los mismos datos, donde se podrá comparar la eficacia en la clasificación de cada algoritmo, mediante los indicadores de calidad de la clasificación.

## ALGORITMO ZERO-R

El ZeroR es uno de los algoritmos base de la tarea de clasificación, su comportamiento sencillo hace que sea rápido en ejecución, pero provoca, por otro lado, unos resultados poco deseables.

Tras la aplicación del método ZeroR, se obtienen como salida del algoritmo resultados basados en la clase mayoritaria, es decir, devuelve la clase con el mayor número de ocurrencias. Por ejemplo, si en un conjunto de datos hay un atributo con posibles valores SI y NO y la clase SI ocurre más veces que la clase NO, entonces se selecciona la clase SI.

Al hacer esta predicción tan simple, se considera el caso base para data mining y deberá ser el de menor tiempo de ejecución, además de que toda aplicación que trabaje con los principios del data mining no debe proporcionar resultados peores que los obtenidos mediante este algoritmo. Debido a su simplicidad, no es usado como predictor, pero sí como punto de referencia para el resto de algoritmos, se mide la eficiencia de los algoritmos comparándolos con la del ZeroR.

Su utilización en Weka es bastante sencilla, al aplicarse sobre los datos se obtiene como salida la clase mayoritaria, entendiendo por clase mayoritaria la moda en el caso de clases nominales y la media en el caso de clases numéricas. Su simplicidad hace que no sea posible modificar ningún parámetro que pueda mejorar, ni tan siquiera modificar, la clasificación.

Al usar el clasificador ZeroR sobre los datos del credit scoring, se han obtenido los resultados mostrados a continuación. En este caso y previsiblemente, se ha obtenido como predicción de clase el valor 1 para la decisión, puesto que la muestra contenía un 70% de valores igual a uno para el atributo clase decisión.

### === Run information ===

Scheme: weka.classifiers.rules.ZeroR  
Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
Instances: 1000  
Attributes: 21  
Test mode: 10-fold cross-validation

### === Classifier model (full training set) ===

ZeroR predicts class value: 1

Time taken to build model: 0 seconds

### === Stratified cross-validation ===

#### === Summary ===

Correctly Classified Instances	700	70 %
Incorrectly Classified Instances	300	30 %
Kappa statistic	0	
Mean absolute error	0.4202	
Root mean squared error	0.4583	
Relative absolute error	100 %	
Root relative squared error	100 %	
Total Number of Instances	1000	

### === Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0	0	0	0	0	0.5	0
	1	1	0.7	1	0.824	0.5	1
Weighted Avg	0.7	0.7	0.49	0.7	0.576	0.5	

### === Confusion Matrix ===

a b <-- classified as  
0 300 | a = 0  
0 700 | b = 1



## ALGORITMO ONE-R (ONE RULE)

El algoritmo OneR crea una regla para cada atributo en los datos de entrenamiento y escoge luego la regla con la tasa de error (número de instancias de los datos de entrenamiento en los que la clase del valor de un atributo no concuerda con la asociación que la que la regla le da a ese atributo) más pequeña como su "one rule". Para crear una regla para cada atributo debe determinarse la clase más frecuente para cada valor del atributo.

Se trata, por tanto, de un árbol de decisión de un único nivel que clasifica en base a un único atributo.

Es un algoritmo de clasificación simple, pero muy efectivo, que, del mismo modo que el algoritmo ZeroR, puede emplearse para determinar lo bien que funcionan otros clasificadores más complejos.

- Ordena los atributos de acuerdo a la tasa de error (sobre el conjunto de entrenamiento)
- Selecciona aleatoriamente un atributo cuando las tasas de error son iguales, y una clase cuando dos o más clases dan la misma tasa de error con un atributo.
- Los valores indefinidos los trata como si pertenecieran al conjunto indefinido. En el caso de existir valores numéricos los trata como si fueran valores numéricos continuos, usando un método directo para agrupar esos valores en un conjunto de intervalos.
- Para evitar el sobreajuste de las muestras, es decir, la generación de demasiados intervalos, el algoritmo requiere que todos los intervalos contengan más de un determinado número mínimo de muestras (por defecto 6).
- En el caso de atributos nominales que no cumplan ese criterio (situación bastante infrecuente) el algoritmo elimina dichos atributos.

### **Pseudocódigo:**

1. En el conjunto de entrenamiento contar el número de ejemplos en la clase C que tiene el valor V para el atributo A y almacenar dicho valor en una matriz de tres dimensiones: COUNT[C,V,A].

2. Se toma como clase por defecto la que contenga el mayor número de ejemplos en el conjunto de entrenamiento (la precisión de la clase por defecto es el número de ejemplos de entrenamiento en la clase por defecto dividido por el número total de ejemplos).
3. Por cada atributo numérico A, crear una versión nominal de A definiendo un número finito de intervalos de valores. Estos intervalos serán los valores de la versión nominal de A, con lo que se tendrá la matriz  $COUNT[C, I, A]$ , cuyo valor para I será la suma de todos los valores de V que entren dentro del intervalo I.
4. Por cada atributo, A, (usando la versión normalizada):
  - a. Construir una hipótesis que incluya el atributo A seleccionando, por cada valor de V de A (y también para los indefinidos), una clase óptima para V. Si varias clases son óptimas, seleccionar una de forma aleatoria.
  - b. Añadir la hipótesis a un conjunto llamado Hipótesis que contendrá una hipótesis por cada atributo.
5. Elegir la regla del conjunto Hipótesis que obtenga mayor precisión sobre el conjunto de entrenamiento. En caso de empate, elegir una al azar.

### Opciones del clasificador OneR en Weka

- **minBucketSize:** número mínimo de valores que debe tener un atributo. Esta opción es relevante cuando se trabaja con datos numéricos. Cuando se tienen datos nominales esta opción se puede obviar en la mayoría de los casos.

### === Run information ===

Scheme: weka.classifiers.rules.OneR -B 6  
Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
Instances: 1000  
Attributes: 21  
Test mode: 10-fold cross-validation

### === Classifier model (full training set) ===

Anteriores:

0 -> 0  
1 -> 0  
2 -> 1  
3 -> 1  
4 -> 1

(717/1000 instances correct)

Time taken to build model: 0.02 seconds

### === Stratified cross-validation ===

#### === Summary ===

Correctly Classified Instances	712	71.2 %
Incorrectly Classified Instances	288	28.8 %
Kappa statistic	0.1304	
Mean absolute error	0.288	
Root mean squared error	0.5367	
Relative absolute error	68.5425 %	
Root relative squared error	117.108 %	
Total Number of Instances	1000	

### === Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.15	0.047	0.577	0.15	0.238	0.551	0
	0.953	0.85	0.723	0.953	0.822	0.551	1
Weighted Avg	0.712	0.609	0.679	0.712	0.647	0.551	

### === Confusion Matrix ===

a b <-- classified as  
45 255 | a = 0  
33 667 | b = 1

## ALGORITMO PART

Este algoritmo utiliza la técnica divide y vencerás para enfrentarse al problema de clasificación. La estrategia del algoritmo Part, consiste en construir una regla, borrar las instancias que la regla cubre y continuar construyendo reglas recursivamente para las restantes instancias hasta que no queden ya más instancias.

Su gran diferencia, respecto a las aproximaciones estándar, reside en la forma que cada regla es creada. Para crear una única regla, se crea un árbol podado para el actual conjunto de instancias, la hoja que posea una cobertura mayor se convierte en regla y el árbol es descartado. De esta forma se consigue evitar la generalización apresurada o indebida (hasty generalization), generalizando una vez que las implicaciones sean conocidas. El hecho de construir un árbol podado para la obtención de una única regla, en lugar de crear un árbol incrementalmente añadiendo una conjunción de cada vez, hace que se evite el problema de árboles sobrepodados, típico de los algoritmos que usan técnicas del tipo divide y vencerás.

La idea principal es construir un árbol parcial de decisión en lugar de uno completamente explorado. Un árbol parcial de decisión es un árbol de decisión ordinario con la característica de poseer como ramas árboles indefinidos. Para lograr generar un árbol de este tipo, se integran las operaciones de construcción y poda para lograr un subárbol estable que no pueda simplificarse más. Una vez encontrado este árbol, el proceso de construcción cesa y se crea la regla.

A continuación se muestra el algoritmo para la construcción de un árbol parcial de decisión. En él se observan los pasos a seguir, primeramente se divide el conjunto dado de ejemplos en subconjuntos. Seguidamente y siempre y cuando haya subconjuntos que todavía no hayan sido expandidos y los que sí han sido expandidos sean hojas, se elige un nuevo subconjunto a expandir y se expande. Una vez expandido se evalúa si todos los árboles que están expandidos son hojas y que el error relativo del subárbol es menor que el error relativo del nodo, en caso afirmativo se debe deshacer la expansión en subconjuntos y marcar el nodo como nodo hoja.

### **Procedure Expand Subset**

1. choose split of given set of examples into subsets

2. **while** there are subsets that have not been expanded  
    **and** all the subsets expanded so far are leaves  
        choose next subset to be expanded and expand it
3. **if** all the subsets expanded are leaves **and**  
    estimated error for subtree  $\geq$  estimated error for node  
        undo expansion into subsets and make node a leaf

Una vez que el árbol de decisión está construido, se extrae una única regla de él. Cada hoja corresponde a una posible regla, por lo que tratamos de encontrar la mejor hoja de entre esos subárboles que han sido expandidos en hojas y que representan a una minoría del total. Se trata por lo tanto, de buscar la regla que tenga una cobertura que incluya al mayor número posible de instancias y seleccionarla como única regla.

Al usar este algoritmo con Weka, se observan diferentes parámetros que pueden ser modificados para una configuración más específica del algoritmo:

- **binarySplits**: Si se desea usar una partición binaria en atributos nominales al construir el árbol parcial. En el caso del credit scoring, no se realizará modificaciones con respecto a este parámetro. Valor por defecto: FALSE.
- **confidenceFactor**: Nivel de confianza usado en la poda (cuanto más pequeños más poda será realizada). Este parámetro sí será susceptible de ser modificado. Valor por defecto: 25%.
- **minNumObj**: Número mínimo de instancias que incluya la cobertura de una regla para poder ser candidata. No se considera necesario incluir una cobertura mínima al usar el caso de credit scoring como un caso ilustrativo, pero pueden darse situaciones reales donde esta cobertura mínima sea deseable, por lo que haremos modificaciones. Valor por defecto: 2.
- **numFolds**: Determina la cantidad de datos usados para el reduced-error pruning, una carpeta será para la poda y el resto para las reglas. No será variado, se mantendrá el valor por defecto. Valor por defecto: 3.
- **reducedErrorPruning**: Se decidirá si se usa el reduced-error pruning o por el contrario el C.4.5 pruning. Se observará cuál de las opciones produce mejores resultados. Valor por defecto: FALSE.

- **seed:** En el caso de usarse el reduced-error pruning, se usará esta semilla para aleatorizar los datos. Se tomará la dada por defecto. Valor por defecto: 1.
- **unpruned:** Si se realizará o no la poda. En el caso del credit scoring, se evaluarán las dos posibilidades. Valor por defecto: FALSE.

Se empieza el análisis con los parámetros por defecto, obteniendo un 71% de instancias correctamente clasificadas. Pero se observa que al aumentar a 10 el número mínimo de instancias se obtiene una mejora de hasta un 72.9% de instancias correctamente clasificadas. Se muestra continuación la salida obtenida con Weka para un número mínimo de instancias igual a 10:

#### === Run information ===

```
Scheme:   weka.classifiers.rules.PART -M 10 -C 0.25 -Q 1
Relation:  credit_weka-weka.filters.unsupervised.attribute.Reorder-
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last
Instances: 1000
Attributes: 21
Test mode: 10-fold cross-validation
```

#### === Classifier model (full training set) ===

PART decision list

```
-----
saldoCta = 4: 1 (394.0/46.0)
meses = 9 AND saldoCta = 2: 1 (83.0/26.0)
meses = 10: 1 (44.0/8.0)
meses = 3: 0 (30.0/9.0)
avales = 1 AND pagoAnteriores = 4: 1 (83.0/22.0)
avales = 1 AND saldoCta = 1 AND ahorros = 1: 0 (123.0/43.0)
meses = 8: 1 (108.0/40.0)
saldoCta = 3: 1 (27.0/3.0)
objeto = 2: 1 (25.0/8.0)
otrosCreditos = 6: 1 (28.0/11.0)

: 0 (55.0/20.0)
Number of Rules :      11
```

Time taken to build model: 0.14 seconds

=== Stratified cross-validation ===

=== Summary ===

Correctly Classified Instances	729	72.9 %
Incorrectly Classified Instances	271	27.1 %
Kappa statistic	0.3245	
Mean absolute error	0.3453	
Root mean squared error	0.4265	
Relative absolute error	82.1838 %	
Root relative squared error	93.0793 %	
Total Number of Instances	1000	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.47	0.16	0.557	0.47	0.51	0.728	
	0.84	0.53	0.787	0.84	0.813	0.728	
Weighted Avg	0.729	0.419	0.718	0.729	0.722	0.728	

=== Confusion Matrix ===

a b <-- classified as

141 159 | a = 0

112 588 | b = 1

Se empieza a variar el nivel de confianza usado para la poda, si lo subimos al 50% se obtienen resultados peores que con el 25% (72.2% de instancias correctamente clasificadas, frente al 72.9% con la confianza al 25%). Este hecho sugiere que se ha elegido una confianza demasiado alta, al comprobarlo con un nivel de confianza del 40%, se obtiene el mejor resultado: un 75.7% de instancias correctas. A continuación se muestra la salida obtenida con Weka para un nivel de confianza del 40%:

**=== Run information ===**

Scheme: weka.classifiers.rules.PART -M 10 -C 0.4 -Q 1  
Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
Instances: 1000  
Attributes: 21  
Test mode: 10-fold cross-validation

**=== Classifier model (full training set) ===**

PART decision list

-----  
saldoCta = 4: 1 (394.0/46.0)  
meses = 10 AND pagoAnteriores = 2: 1 (21.0/3.0)  
pagoAnteriores = 4: 1 (140.0/40.0)  
avales = 1 AND saldoCta = 3: 1 (40.0/9.0)  
avales = 1 AND ahorros = 5 AND saldoCta = 2: 1 (36.0/6.0)  
avales = 3: 1 (32.0/5.0)  
ahorros = 1 AND meses = 7: 0 (35.0/9.0)  
ahorros = 1 AND otrosCreditos = 9: 0 (22.0/5.0)  
pertenencias = 4 AND numCredsAntsBanco = 1 AND antigEmpleo = 5: 0 (17.0/4.0)  
edad = 3: 0 (45.0/19.0)  
edad = 1 AND ocupacion = 3 AND ingresosDisp = 2: 0 (15.0/2.0)  
ingresosDisp = 2: 1 (38.0/8.0)  
pertenencias = 4: 0 (26.0/5.0)  
ahorros = 2 AND pertenencias = 3: 1 (13.0/6.0)  
ahorros = 1 AND ocupacion = 3 AND antigCasa = 4: 0 (20.0/7.0)  
objeto = 2: 1 (25.0/8.0)  
objeto = 0: 0 (24.0/8.0)  
antigEmpleo = 3 AND edad = 1: 1 (12.0/3.0)  
antigEmpleo = 2: 0 (14.0/7.0)  
antigEmpleo = 3: 0 (13.0/4.0)  
  
: 1 (18.0/5.0)  
Number of Rules : 21  
Time taken to build model: 0.07 seconds

**=== Stratified cross-validation ===**



### === Summary ===

Correctly Classified Instances	757	75.7 %
Incorrectly Classified Instances	243	24.3 %
Kappa statistic	0.3845	
Mean absolute error	0.3316	
Root mean squared error	0.4149	
Relative absolute error	78.9079 %	
Root relative squared error	90.5494 %	
Total Number of Instances	1000	

### === Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.49	0.129	0.62	0.49	0.547	0.744	0
	0.871	0.51	0.799	0.871	0.834	0.744	1
Weighted Avg	0.757	0.396	0.746	0.757	0.748	0.744	

### === Confusion Matrix ===

```
a  b  <-- classified as
147 153 | a = 0
90 610 | b = 1
```

Cabe destacar que activando los parámetros `unpruned` y `reducedErrorPruning` no se obtienen mejores resultados, por lo que la configuración de parámetros que proporciona mejores resultado para este modelo es el anteriormente descrito.

## ALGORITMO JRIP (RIPPER)

El algoritmo RIPPER (Repeated Incremental Pruning Produce Error Reduction) fue introducido por Cohen (1995) como mejora del algoritmo IREP (Incremental Reduced Error Pruning).

El algoritmo IREP considera un conjunto de reglas en forma normal disyuntiva: un conjunto formado por una disyunción de reglas, cada una de las cuales estará constituida por una conjunción de literales, siendo una regla parcial de una determinada regla la intersección de un subconjunto de los literales a partir de los cuales se forma esa regla.

Considerando el caso en el que la variable clase tome dos posibles valores (ejemplos positivos y negativos), el algoritmo IREP etiqueta el conjunto de casos o patrones en dos subconjuntos, el de los patrones positivos y el de los negativos, subdividiendo cada uno de estos subconjuntos en otros dos subconjuntos relacionados, respectivamente, con la construcción y el podado de las reglas.

A partir de aquí, el algoritmo induce el conjunto de reglas utilizando un procedimiento (GrowRule) que lo que hace es añadir repetidamente a la regla en construcción aquel literal que da origen a la regla parcial con mayor valor del criterio siguiente:

$$v(Rpar; R'par; Dgrow_{pos}; Dgrow_{neg}) = cu \left[ -\log_2 \left( \frac{pos}{pos + neg} \right) + \log_2 \left( \frac{pos'}{pos' + neg'} \right) \right] \quad [10]$$

Siendo:

- $cu$ : porcentaje de ejemplos positivos en  $Dgrow_{pos}$  que siendo cubiertos por  $Rpar$  están también cubiertos por  $R'par$  (al ser  $Rpar$  más específica que  $R'par$  no todos los ejemplos positivos cubiertos por  $R'par$  lo van a estar por  $Rpar$ ).
- $pos$ : número de ejemplos positivos cubiertos por  $Rpar$  en  $Dgrow_{pos}$ .
- $neg$ : número de ejemplos negativos cubiertos por  $Rpar$  en  $Dgrow_{neg}$ .
- $pos'$ : número de ejemplos positivos cubiertos por  $R'par$  en  $Dgrow_{pos}$ .
- $neg'$ : número de ejemplos negativos cubiertos por  $R'par$  en  $Dgrow_{neg}$ .

Cuando no se encuentra ningún literal cuya inclusión en la regla permita que la regla especializada mejore el criterio, comienza el proceso de podado de dicha regla, por medio de un procedimiento (PruneRule) que lo que plantea es un borrado secuencial, empezando por el último literal introducido a la regla en su fase de crecimiento, podando literales mientras se mejore el criterio:

$$v(Rule; Dprune_{pos}; Dprune_{neg}) = Pos \left( \frac{Neg - neg}{Pos + neg} \right)$$

Siendo:

- *Pos*: número de ejemplos en  $Dprune_{pos}$ .
- *Neg*: número de ejemplos en  $Dprune_{neg}$ .
- *pos*: número de ejemplos positivos en  $Dprune_{pos}$  cubiertos por la regla.
- *neg*: número de ejemplos negativos en  $Dprune_{neg}$  cubiertos por la regla.

#### **Pseudocódigo:**

##### **Begin**

Ruleset= $\emptyset$

**while**  $D_{pos} = Dgrow_{pos} \cup \neq \emptyset$  **do**

/\*construir y podar una nueva regla\*/

Dividir D en  $Dgrow_{pos} \cup Dgrow_{neg} \cup Dprune_{pos} \cup Dprune_{neg}$

Rule:=GrowRule( $Dgrow_{pos} \cup Dgrow_{neg}$  D)

Rule:=PruneRule(Rule,  $Dprune_{pos} \cup Dprune_{neg}$ )

**if** la tasa de error de Rule en  $Dprune_{pos} \cup Dprune_{neg} > 50\%$

**then**

return RuleSet

**else**

Añadir Rule a RuleSet

Borrar ejemplos abiertos por Rule de D

**end if**

**end while**

return RuleSet

##### **End**

Las mejoras que introduce el algoritmo RIPPER son:

1. Métrica alternativa para la fase de poda, basando la poda en el criterio siguiente:

$$v(\text{Rule}; D_{\text{prune}_{\text{pos}}}; D_{\text{prune}_{\text{neg}}}) = \frac{\text{Pos} - \text{Neg}}{\text{Pos} + \text{Neg}}$$

2. Incorporación de un heurístico para determinar cuándo parar el proceso de añadir reglas.
3. Posteriormente a todo el proceso visto para el algoritmo IREP, el algoritmo RIPPER efectúa una búsqueda local para optimizar el conjunto de reglas (rule set) de dos maneras diferentes:
  - a. Reemplazando una regla que forma parte del conjunto de reglas por otra siempre y cuando el conjunto de reglas correspondiente tenga un menor error de clasificación en  $D_{\text{prune}_{\text{pos}}} \cup D_{\text{prune}_{\text{neg}}}$ .
  - b. Revisar una determinada regla añadiendo literales para que así se consiga un menor error en  $D_{\text{prune}_{\text{pos}}} \cup D_{\text{prune}_{\text{neg}}}$ .

#### Opciones del clasificador JRIP en Weka:

- **checkErrorRate:** tasa de error usada como criterio para detener el proceso de poda.
- **folds:** cantidad de datos usado para la poda.
- **minNo:** peso mínimo total de las instancias de una regla.
- **optimizations:** número de procesos de optimización.
- **seed:** semilla de aleatorización.
- **usePruning:** para realizar/omitir proceso de poda.

### === Run information ===

Scheme: weka.classifiers.rules.JRip -F 3 -N 2.0 -O 2 -S 1  
Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
Instances: 1000  
Attributes: 21  
Test mode: 10-fold cross-validation

### === Classifier model (full training set) ===

JRIP rules:

=====

(saldoCta = 1) and (ocupacion = 3) and (ahorros = 1) => decision=0 (134.0/53.0)  
(saldoCta = 2) and (otrosCreditos = 3) and (pagoAnteriores = 2) => decision=0 (10.0/0.0)  
(saldoCta = 2) and (edad = 1) and (antigCasa = 2) => decision=0 (23.0/6.0)  
(antigEmpleo = 2) and (meses = 5) => decision=0 (7.0/1.0)  
(saldoCta = 2) and (ahorros = 2) and (ecivil\_sexo = 2) => decision=0 (9.0/1.0)  
=> decision=1 (817.0/178.0)

Number of Rules : 6

Time taken to build model: 0.28 seconds

### === Stratified cross-validation ===

#### === Summary ===

Correctly Classified Instances	690	69 %
Incorrectly Classified Instances	310	31 %
Kappa statistic	0.1994	
Mean absolute error	0.388	
Root mean squared error	0.4585	
Relative absolute error	92.3372 %	
Root relative squared error	100.0496 %	
Total Number of Instances	1000	

#### === Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.347	0.163	0.477	0.347	0.402	0.593	0
	0.837	0.653	0.749	0.837	0.791	0.593	1
Weighted Avg	0.69	0.506	0.668	0.69	0.674	0.593	

#### === Confusion Matrix ===

a b <-- classified as  
104 196 | a = 0  
114 586 | b = 1

## ALGORITMO CONJUNCTIVE RULES

Un single conjunctive rule learner (aprendizaje simple de reglas conjuntivas) es un aprendizaje basado en reglas que puede predecir tanto para clases numéricas como nominales.

Una regla consiste en antecedentes, que puede ir unidos por “AND” y el consecuente, valor de la clase. En el caso del conjuntives rules, el consecuente será la distribución de las clases disponibles en el conjunto de datos o su media, en el caso de un valor numérico. Si el test de instancia no está cubierto por esta regla, entonces se predice usando la distribución o media por defecto del dato, no cubierto por la regla, en el conjunto de datos de entrenamiento. Este aprendizaje selecciona un antecedente calculando la Information Gain ( $\text{Information gain} = \text{information before splitting} - \text{information after splitting}$ ) de cada antecedente y poda la regla generada usando el REP-Reduced Error Prunning (Error reducido de poda) o el prepodamiento simple basado en el número de antecedentes.

En el caso de la clasificación, la información de un antecedente es la media ponderada tanto de los datos cubiertos como no cubiertos por la regla y se usa en la poda

Al usar este algoritmo con Weka, se observan diferentes parámetros que pueden ser modificados para una configuración más específica del algoritmo:

- **exclusive:** Se establece si se considera una expresión exclusiva para la división de atributos nominales. Valor por defecto: True.
- **folds:** Determina la cantidad de datos usados para el reduced-error pruning, una carpeta será para la poda y el resto para las reglas. No será variado, se mantendrá el valor por defecto. Valor por defecto: 3.
- **minNo:** Se establece el peso mínimo total de las instancias en una regla. Valor por defecto: 2.0
- **numAntds:** número de antecedentes permitidos en la regla si se usa el prepodado. Valor por defecto -1.
- **seed:** La semilla para aleatorizar los datos.

Al realizar el análisis de los datos de credit scoring con el Weka, no se han obtenido resultados positivos con este algoritmo. Se ha intentado modificar parámetros para mejorar los resultados, pero las modificaciones no han causado efecto en la salida, obteniéndose siempre resultados semejantes a los del ZeroR como se muestra a continuación:

#### === Run information ===

Scheme: weka.classifiers.rules.ConjunctiveRule -N 3 -M 2.0 -P -1 -S 1  
 Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
 R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
 weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
 Instances: 1000  
 Attributes: 21  
 Test mode: 10-fold cross-validation

#### === Classifier model (full training set) ===

Single conjunctive rule learner:

-----

=> decision = 1

Class distributions:

Covered by the rule:

0	1
0.29985	0.70015

Not covered by the rule:

0	1
0	0

Time taken to build model: 0.07 seconds

#### === Stratified cross-validation ===

#### === Summary ===

<b>Correctly Classified Instances</b>	700	70.0 %
<b>Incorrectly Classified Instances</b>	300	30.0 %
<b>Kappa statistic</b>	0	
<b>Mean absolute error</b>	0.42	
<b>Root mean squared error</b>	0.4583	
<b>Relative absolute error</b>	99.9578 %	
<b>Root relative squared error</b>	100 %	
<b>Total Number of Instances</b>	1000	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0	0	0	0	0	0.5	0
	1	1	0.7	1	0.824	0.5	1
Weighted Avg	0.7	0.7	0.49	0.7	0.576	0.5	

=== Confusion Matrix ===

a b <-- classified as

0 300 | a = 0

0 700 | b = 1



## ALGORITMO NNGE

El algoritmo del vecino más cercano usa una métrica que mide la distancia entre el nuevo ejemplo y un conjunto de ejemplares en memoria. De esta manera, se clasifica el nuevo ejemplo de acuerdo a la clase vecina más cercana. Para la determinación de la similitud, se usa una función distancia. En el caso de atributos numéricos, esta función distancia estará basada en la distancia euclídea; donde cada ejemplo será tratado como un punto en un espacio  $n$ -dimensional y asumiendo que, para cada punto dado, su área circundante compartirá su misma clase.

Al considerar los atributos simbólicos, éstos no encajan en el modelo espacial euclídeo, por lo que se determina su similitud contando las características comunes que poseen los dos puntos que se están considerando, dando lugar a una métrica mucho menos robusta. En el caso de dominios que presenten tanto atributos numéricos como categóricos, se ha de considerar como métrica la distancia euclídea, pero considerando una distancia cero para los atributos simbólicos que presenten una misma característica y uno si no la presentan. Esto hace que los sistemas puros basados en el vecino más cercano funcionan generalmente mejor en dominios numéricos que en dominios simbólicos. El problema de estos algoritmos es encontrar una función distancia que funcione bien tanto para atributos discretos como continuos, además de los problemas introducidos por el ruido y los atributos irrelevantes.

La técnica de Ejemplares Generalizados (Generalised exemplars) ofrece una solución para los problemas que presenta el algoritmo del vecino más cercano. Así los ejemplares que compartan la misma clase se agrupan, representando así más completamente grandes reglas; este hecho reduce el papel de la función distancia para clasificar cuando no existe una regla que cubra el nuevo ejemplar y, por lo tanto, los errores en la clasificación provocados por la imprecisión de la función distancia.

Non-Nested Generalised Exemplars (NNGE), es un algoritmo que generaliza ejemplares sin anidamiento o solapamiento. Es una extensión del NGE (Salzberg, 1991), que realiza la generalización uniendo ejemplares y formando hiperrectángulos en el espacio que representan las conjunctive rules con disyunción interna. NNGE forma una generalización cada vez que un nuevo ejemplar se añade a la base de datos, uniéndolo a su vecino más cercano de la misma clase. A diferencia de su predecesor, el

NNGE, NNGE no permite a los hiperrectángulos el anidamiento o solapamiento debido a que verifica en cada generalización para asegurarse que no se cubra ningún ejemplar negativo y modifica cualquier generalización que lo haga en un momento posterior.

NNGE aprende incrementalmente clasificando en un primer momento y luego generalizando cada nuevo ejemplar. Usa una función distancia Euclídea modificada que maneja los hiperrectángulos, las características simbólicas, los ejemplares y los pesos. Las características con valores numéricos se normalizan dividiendo cada valor por el rango de los valores observados.

La clase predicha inicialmente es la proporcionada por el método del vecino más cercano y, después de que cada nuevo ejemplo sea clasificado, se usa un feedback dinámico para ajustar el nuevo ejemplar y sus pesos. El NNGE evita el problema que puede producirse cuando, al realizarse la clasificación de un ejemplar, le correspondan un, o más de un, hiperrectángulo al que pertenezca, pero que no sea de la clase correcta. Una vez clasificado, el nuevo ejemplar se generaliza uniéndolo con el vecino más cercano de la misma clase, que puede ser tanto un hiperrectángulo como un ejemplar simple. De producirse el primer caso, se crearía un nuevo hiperrectángulo, en el segundo caso, el vecino más cercano crecerá hasta cubrir al nuevo ejemplo.

No se permite la sobregeneralización causada por el anidamiento y el solapamiento de hiperrectángulos, debido a que antes de que se generalice un nuevo ejemplar, se verifica si existe algún ejemplar en el espacio correspondiente al nuevo hiperrectángulo propuesto.

Se muestra a continuación el pseudocódigo del algoritmo, donde podemos observar las tres grandes partes de este algoritmo la clasificación del ejemplar, el ajuste del modelo y la generalización del nuevo ejemplar:

#### **Mientras (existan nuevos ejemplares)**

Leer ejemplar

Almacenar ejemplar

Ajustar el rango de atributos

**Clasificar el ejemplar:**

mientras (existan más rectángulos)

calcular la distancia del nuevo ejemplar al rectángulo  
 si (distancia < la mínima distancia a la clase del rectángulo)  
     establecer la distancia mínima a esa distancia  
     establecer el contador de clase a 1  
 si(distancia = la mínima distancia a la clase del rectángulo)  
     incrementar el contador de clase en 1  
 mientras(existan más ejemplares sin generalizar)  
     calcular distancia del nuevo ejemplar al ejemplar  
     si (distancia < la mínima distancia a la clase del rectángulo)  
         establecer la distancia mínima a esa distancia  
         establecer el contador de clase a 1  
     si(distancia = la mínima distancia a la clase del rectángulo)  
         incrementar el contador de clase en 1  
 devolver la clase con la menor distancia y el mayor contador  
 devolver el primer ejemplar/hiperrectángulo encontrado con esa clase/distancia

#### **ajustar modelo:**

si (es correcta la predicción)  
     incrementar positivamente el contador para ese ejemplar/hiperrectángulo  
 si no  
     incrementar positivamente el contador para ese ejemplar/hiperrectángulo  
     si (ejemplar se encuentra dentro de un hiperrectángulo de otra clase)  
         podar el hiperrectángulo sobregeneralizado  
     si no  
         ajustar los pesos para los atributos con valores diferentes

#### **generalizar el nuevo ejemplar:**

si (el vecino más cercano es un hiperrectángulo)  
     extender cada rango característico para incluir al nuevo ejemplar  
     si (el rectángulo extendido cubre ejemplares/rectángulos conflictivos)  
         restablecer el hiperrectángulo a su tamaño original  
         almacenar el nuevo ejemplar en verbatim  
     si no  
         conservar las modificaciones  
         descartar el ejemplar

si (el vecino más cercano es un ejemplar)  
    crear un hiperrectángulo que cubra los dos ejemplares  
    si (el nuevo rectángulo cubre ejemplares/rectángulos conflictivos)  
        descartar el nuevo hiperrectángulo  
        almacenar el nuevo ejemplar en verbatim  
    si no  
        conservar las modificaciones  
        descartar el ejemplar

Al usar la implementación que Weka nos ofrece de este algoritmo, se observan diferentes parámetros que pueden ser modificados para una configuración más específica del algoritmo:

- numAttemptsOfGeneOption: Establece el número de tentativas para generalización. Valor por defecto: 5.
- numFoldersMIOption: Establece el número de carpetas para información mutua. Valor por defecto: 5.

Al realizar el estudio de los datos del credit scoring con Weka y el algoritmo NNGE se obtienen unos resultados que no mejoran a los del ZeroR, incluso si se varían los parámetros que controlan el número de tentativas de la generalización o los que establecen el número de carpetas para información mutua, se pueden obtener resultados peores que los obtenidos por el ZeroR.

En un primer lugar, se muestra la salida de Weka producida al ejecutar el algoritmo NNGE sobre los datos de credit scoring con los parámetros con sus valores por defecto:

### === Run information ===

Scheme: weka.classifiers.rules.NNge -G 5 -I 5  
Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
Instances: 1000  
Attributes: 21  
Test mode: 10-fold cross-validation

### === Classifier model (full training set) ===

NNGE classifier

#### Rules generated :

(...)

Stat :

class 0 : 167 exemplar(s) including 70 Hyperrectangle(s) and 97 Single(s).  
class 1 : 218 exemplar(s) including 76 Hyperrectangle(s) and 142 Single(s).  
Total : 385 exemplars(s) including 146 Hyperrectangle(s) and 239 Single(s).

Feature weights : [0.09473884155263944 0.04411075852920106 0.04361779931044024  
0.024893540002036224 0.019106490808355733 0.028114675087585177 0.013102322536286123  
0.003972131851941858 0.006810549736432131 0.004797020933391814 5.425012434176397E-4  
0.016985185935771783 0.011540940040073498 0.008875070307598633 0.013076961913416726  
0.001978331300512697 0.0013373568583454568 6.5676192783540406E-6 9.636600149094416E-4  
0.005822991014286552]

Time taken to build model: 1.22 seconds

### === Stratified cross-validation ===

#### === Summary ===

Correctly Classified Instances	700	70.0 %
Incorrectly Classified Instances	300	30.0 %
Kappa statistic	0.2331	
Mean absolute error	0.3	
Root mean squared error	0.5477	
Relative absolute error	71.3984 %	
Root relative squared error	119.5228 %	
Total Number of Instances	1000	

#### === Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.38	0.163	0.5	0.38	0.432	0.609	0
	0.837	0.62	0.759	0.837	0.796	0.609	1
<b>Weighted Avg</b>	0.7	0.483	0.681	0.7	0.687	0.609	

#### === Confusion Matrix ===

a b <-- classified as

114 186 | a = 0

114 586 | b = 1

En un segundo lugar, se muestra la salida de Weka producida al ejecutar el algoritmo NNGE sobre los datos de credit scoring con los parámetros modificados (en este caso 10 tentativas), donde se pueden observar resultado peores que los del ZeroR :

#### === Run information ===

Scheme: [weka.classifiers.rules.NNge -G 10 -I 5](#)

Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-

weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last

Instances: 1000

Test mode: 10-fold cross-validation

#### === Classifier model (full training set) ===

[NNGE classifier](#)

[Rules generated :](#)

class 0 IF : saldoCta in {1} ^ meses in {9} ^ pagoAnteriores in {0,2} ^ objeto in {2,3,9} ^ otrosCreditos in {3,5} ^ ahorros in {1,5} ^ antigEmpleo in {1,3,4,5} ^ ingresosDisp in {2,4} ^ ecivil\_sexo in {3} ^ avales in {1} ^ antigCasa in {2,4} ^ pertenencias in {2,3} ^ edad in {2,3} ^ otrosPrestatarios in {3} ^ vivienda in {1,2} ^ numCredsAntsBanco in {1,2} ^ ocupacion in {3} ^ unidadFamiliar in {1} ^ telef in {1,2} ^ extranjero in {1} (5)

(.....)

class 1 IF : saldoCta in {2} ^ meses in {10} ^ pagoAnteriores in {4} ^ objeto in {8} ^ otrosCreditos in {9} ^ ahorros in {5} ^ antigEmpleo in {4} ^ ingresosDisp in {1} ^ ecivil\_sexo in {2} ^ avales in {1} ^ antigCasa in {3} ^ pertenencias in {2} ^ edad in {2} ^ otrosPrestatarios in {3} ^ vivienda in {2} ^ numCredsAntsBanco in {2} ^ ocupacion in {2} ^ unidadFamiliar in {1} ^ telef in {1} ^ extranjero in {1} (1)

Stat :

class 0 : 158 exemplar(s) including 72 Hyperrectangle(s) and 86 Single(s).

class 1 : 176 exemplar(s) including 79 Hyperrectangle(s) and 97 Single(s).

Total : 334 exemplars(s) including 151 Hyperrectangle(s) and 183 Single(s).

Feature weights : [0.09473884155263944 0.04411075852920106 0.04361779931044024  
0.024893540002036224 0.019106490808355733 0.028114675087585177 0.013102322536286123  
0.003972131851941858 0.006810549736432131 0.004797020933391814 5.425012434176397E-4  
0.016985185935771783 0.011540940040073498 0.008875070307598633 0.013076961913416726  
0.001978331300512697 0.0013373568583454568 6.5676192783540406E-6 9.636600149094416E-4  
0.005822991014286552]

Time taken to build model: 1.3 seconds

#### === Stratified cross-validation ===

##### === Summary ===

<b>Correctly Classified Instances</b>	697	69.7 %
<b>Incorrectly Classified Instances</b>	303	30.3 %
<b>Kappa statistic</b>	0.2215	
<b>Mean absolute error</b>	0.303	
<b>Root mean squared error</b>	0.5505	
<b>Relative absolute error</b>	72.1124 %	
<b>Root relative squared error</b>	120.1189 %	
<b>Total Number of Instances</b>	1000	

##### === Detailed Accuracy By Class ===

	<b>TP Rate</b>	<b>FP Rate</b>	<b>Precision</b>	<b>Recall</b>	<b>F-Measure</b>	<b>ROC Area</b>	<b>Class</b>
	0.367	0.161	0.493	0.367	0.421	0.603	0
	0.839	0.633	0.755	0.839	0.795	0.603	1
<b>Weighted Avg</b>	0.697	0.492	0.677	0.697	0.683	0.603	

##### === Confusion Matrix ===

a b <-- classified as

110 190 | a = 0

113 587 | b = 1

## DTNB (DECISION TABLES AND NAIVE BAYES)

Se trata de un algoritmo de clasificación que divide el conjunto de atributos en dos grupos, asignando a un grupo probabilidades de clase basadas en naive bayes y a otro grupo en una tabla de decisión, combinando las estimaciones de probabilidad resultantes.

Una tabla de decisión almacena los datos de entrada de forma condensada en base a una colección seleccionada de atributos usándola como tabla de consulta para hacer predicciones. Cada entrada de la tabla se asocia con estimaciones de probabilidad de clase basadas en frecuencias observadas. La clave es la selección de un subconjunto de atributos sumamente discriminatorios, que normalmente se realiza mediante validación cruzada.

El algoritmo DTNB continúa del mismo modo que lo hacen las tablas de decisión simple, evaluando en cada punto de la búsqueda el mérito asociado dividiendo los atributos en dos subconjuntos inconexos: uno para DT y otro para NB.

En cada paso los atributos seleccionados son modelados por NB, y el resto por DT, siendo todos los atributos modelados inicialmente por la tabla de decisión.

Las estimaciones de probabilidad de clase del DT y NB deben ser combinadas para generar estimaciones de probabilidad de clase totales. Asumiendo que  $X^I$  es el conjunto de atributos en el DT y  $X^\perp$  lo es en el NB, la probabilidad de clase total se calcula como:

$$Q\left(\frac{y}{X}\right) = \frac{\alpha \times Q_{DT}(y|X^I) \times Q_{NB}(y|X^\perp)}{Q(y)}$$

Donde  $Q_{DT}(y|X^I)$  y  $Q_{NB}(y|X^\perp)$  son las estimaciones de probabilidad de clase obtenidas de la tabla de decisión y del naive bayes respectivamente,  $\alpha$  es una constante de normalización y  $Q(y)$  es la probabilidad previa de la clase.



### **Opciones del algoritmo DTNB en weka:**

- **crossVal**: número de grupos para validación cruzada (1 = uno fuera).
- **displayRules**: para mostrar o no las reglas resultantes.
- **evaluationMeasure**: medida utilizada para evaluar el rendimiento de la combinación de atributos usada en la tabla de decisión.
- **search**: método de búsqueda utilizado para encontrar la combinación de atributos de la tabla de decisión.
- **useIBk**

### === Run information ===

Scheme: weka.classifiers.rules.DTNB -X 1  
Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
Instances: 1000  
Attributes: 21  
Test mode: 10-fold cross-validation

### === Classifier model (full training set) ===

Decision Table: (...)  
Number of training instances: 1000  
Number of Rules : 980  
Non matches covered by Majority class.  
Evaluation (for feature selection): CV (leave one out)  
Feature set: 4,5,7,8,11,12,13,14,17,19,21  
  
Time taken to build model: 5.71 seconds

### === Stratified cross-validation ===

#### === Summary ===

Correctly Classified Instances	751	75.1 %
Incorrectly Classified Instances	249	24.9 %
Kappa statistic	0.3602	
Mean absolute error	0.3289	
Root mean squared error	0.4151	
Relative absolute error	78.2665 %	
Root relative squared error	90.5715 %	
Total Number of Instances	1000	

### === Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
	0.457	0.123	0.614	0.457	0.524	0.767	0
	0.877	0.543	0.79	0.877	0.831	0.767	1
Weighted Avg	0.751	0.417	0.737	0.751	0.739	0.767	

### === Confusion Matrix ===

a b <-- classified as  
137 163 | a = 0  
86 614 | b = 1

## REGLAS DE ASOCIACIÓN

Inicialmente, las reglas de asociación surgieron para analizar la composición de las cestas de la compra. El objetivo inicial era observar las combinaciones de productos que suelen comprar los clientes, para mejorar la distribución de los mismos en las estanterías.

El primer paso es crear una base de datos donde se almacenarán los datos referentes a nuestro problema y a continuación se trata de extraer información útil a partir de ellos mediante un algoritmo de aprendizaje de reglas de asociación.

Para poder entender mejor el proceso nos vamos a apoyar en un ejemplo del problema de la cesta de la compra. Como hemos dicho anteriormente, nos interesa observar qué productos se compren conjuntamente. Comenzamos creando una base de datos con una tabla, en la que almacenamos en cada fila una cesta de la compra y en cada columna cada producto que hay disponible en la tienda. En la tabla vamos a guardar valores binarios de manera que, tendremos un 1 en la posición  $(i,j)$  en el caso de que en la cesta  $i$  se haya incluido el producto  $j$  y un 0 en el caso contrario. En la *figura\_1* podemos observar una tabla con estas características.

	Pan	Leche	Cerveza	Pañales	Galletas	Cacao
C1	1	0	0	0	1	1
C2	0	1	0	1	1	0
C3	1	1	0	0	0	1
C4	1	0	1	1	0	0
C5	1	0	1	0	1	0
C6	1	1	0	1	0	0
C7	1	0	1	1	0	0
C8	0	1	0	0	1	1
C9	0	1	1	1	1	1

*Figura 1. Tabla de la cesta de la compra.*

Las reglas de asociación son proposiciones de la forma “*Si X entonces Y*” siendo X e Y subconjuntos de todas las combinaciones posibles de pares atributo-valor. Como en nuestro ejemplo solo tenemos valores binarios, X e Y serán subconjuntos de los productos existentes en la tienda. Una regla de asociación sacada de nuestro ejemplo podría ser: **SI Pan Y Leche ENTONCES Cacao**, ya que observamos en la fila C3 que tanto Pan como Leche y Cacao tienen valor 1.

Supongamos ahora que tenemos una base de datos (*figura 2*) donde almacenamos las condiciones climáticas y si se puede disputar un partido o no. Ahora nos encontramos con que los valores de los atributos no son de tipo binario.

<b>Cielo</b>	<b>Temp</b>	<b>Humedad</b>	<b>Viento</b>	<b>Jugar</b>
Soleado	Alta	Alta	Falso	No
Soleado	Alta	Alta	Cierto	No
Cubierto	Alta	Alta	Falso	Si
Lluvioso	Suave	Alta	Falso	Si
Lluvioso	Fría	Normal	Falso	Si
Lluvioso	Fría	Normal	Cierto	No
Cubierto	Fría	Normal	Cierto	Si
Soleado	Suave	Alta	Falso	No
Soleado	Fría	Normal	Falso	Si
Lluvioso	Suave	Normal	Falso	Si
Soleado	Suave	Normal	Cierto	Si
Cubierto	Suave	Alta	Cierto	Si
Cubierto	Alta	Normal	Falso	Si
Lluvioso	Suave	Alta	Cierto	No

*Figura 2.- Tabla del clima y deporte.*

En este caso los subconjuntos X e Y si deben de ser subconjuntos de los posibles pares atributo-valor. A continuación podemos observar algunos ejemplos de reglas de asociación obtenidas a partir de los datos de la *figura 2*.

**SI** “*humedad=normal*” **Y** “*viento=falso*” **ENTONCES** “*jugar=si*”

**SI** “*humedad=normal*” **Y** “*jugar=si*” **ENTONCES** “*viento=falso*”

**SI** “*viento=falso*” **ENTONCES** “*humedad=normal*” **Y** “*jugar=si*”

**SI** “*jugar=si*” **ENTONCES** “*humedad=normal*” **Y** “*viento=falso*”

Si nos fijamos en las reglas creadas anteriormente podemos observar que en la parte derecha de la regla puede aparecer cualquier atributo e incluso pueden aparecer más de uno. Debido a esto vemos que se generan muchas reglas a considerar aunque nuestra base de datos sea pequeña.

Ahora que ya sabemos intuitivamente lo que es una regla de asociación vamos a dar una definición más formal.

Consideremos  $I=\{i_1, i_2, \dots, i_n\}$  el conjunto de literales, llamados items. Los literales son el conjunto de posibles pares atributo-valor y serán los objetos base de este modelo.

En el primer ejemplo (*figura 1*) como los datos son de tipo binario tendríamos que I sería el conjunto de atributos ya que al ser datos binarios solamente existe la posibilidad de que cada atributo aparezca o no en cada cesta de la compra. En concreto el conjunto quedaría de la siguiente manera:

$I = \{“Pan”, “Leche”, “Cerveza”, “Pañales”, “Galletas”, “Cacao”\}$

En el segundo ejemplo tenemos que para cada atributo hay un conjunto de posibles valores que mostramos a continuación:

**Cielo** = {“Soleado”| “Cubierto”| “Lluvioso”}

**Temp** = {“Fría”| “Suave”| “Alta”}

**Humedad** = {“Normal”| “Alta”}

**Viento** = {“Cierto”| “Falso”}

$$\mathbf{Jugar} = \{ "Si" | "No" \}$$

Debido a lo mencionado anteriormente, el conjunto I sería el conjunto de todos los pares atributo-valor que se puedan formar con los datos anteriores quedándonos de la siguiente manera:

$$\cdot I = \{ "Cielo=Soleado", "Cielo=Cubierto", "Cielo=Lluvioso", "Temp=Fria", "Temp=Suave", "Temp=Alta", "Humedad=Normal", "Humedad=Alta", "Viento=Cierto", "Viento=Falso" \}$$

Llamaremos  $\mathbf{D}=\{t_1, t_2, \dots, t_n\}$  al conjunto de transacciones almacenadas en la base de datos. Cada transacción es la representación de cada uno de los datos del problema. Dicha transacción no es más que un conjunto de items, es decir, es un subconjunto de I y posee un identificador único (**TID**). A continuación vamos a ver cuáles son los conjuntos de las bases de datos de los ejemplos.

En el ejemplo de la cesta de compra cada transacción indicará si el producto forma parte de la cesta o no:

$$\mathbf{D1} = \{$$

$$\begin{aligned} t_1 &= \{ "Pan", "Galletas", "Cacao" \}, \\ t_2 &= \{ "Leche", "Pañales", "Galletas" \}, \\ t_3 &= \{ "Pan", "Leche", "Cacao" \}, \\ t_4 &= \{ "Pan", "Cerveza", "Pañales" \}, \\ t_5 &= \{ "Pan", "Cerveza", "Galletas" \}, \\ t_6 &= \{ "Pan", "Leche", "Pañales" \}, \\ t_7 &= \{ "Pan", "Cerveza", "Pañales" \}, \\ t_8 &= \{ "Leche", "Galletas", "Cacao" \}, \\ t_9 &= \{ "Leche", "Cerveza", "Pañales", "Galletas", "Cacao" \} \\ &\} \end{aligned}$$

Para el segundo ejemplo tenemos que cada transacción será el conjunto de valores que toma cada atributo:

$$\mathbf{D2} = \{$$

$$t_1 = \{ "Cielo=Soleado", "Temp=Alta", "Humedad=Alta", "Viento=Falso", "Jugar=No" \},$$

$t_2 = \{ \text{"Cielo=Soleado"}, \text{"Temp=Alta"}, \text{"Humedad=Alta"}, \text{"Viento=Cierto"}, \text{"Jugar=No"} \},$   
 $t_3 = \{ \text{"Cielo=Cubierto"}, \text{"Temp=Alta"}, \text{"Humedad=Alta"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_4 = \{ \text{"Cielo=Lluvioso"}, \text{"Temp=Suave"}, \text{"Humedad=Alta"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_5 = \{ \text{"Cielo=Lluvioso"}, \text{"Temp=Fría"}, \text{"Humedad=Normal"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_6 = \{ \text{"Cielo=Lluvioso"}, \text{"Temp=Fría"}, \text{"Humedad=Normal"}, \text{"Viento=Falso"}, \text{"Jugar=No"} \},$   
 $t_7 = \{ \text{"Cielo=Cubierto"}, \text{"Temp=Fría"}, \text{"Humedad=Normal"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_8 = \{ \text{"Cielo=Soleado"}, \text{"Temp=Suave"}, \text{"Humedad=Alta"}, \text{"Viento=Falso"}, \text{"Jugar=No"} \},$   
 $t_9 = \{ \text{"Cielo=Soleado"}, \text{"Temp=Fría"}, \text{"Humedad=Normal"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_{10} = \{ \text{"Cielo=Lluvioso"}, \text{"Temp=Suave"}, \text{"Humedad=Normal"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_{11} = \{ \text{"Cielo=Soleado"}, \text{"Temp=Suave"}, \text{"Humedad=Normal"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_{12} = \{ \text{"Cielo=Cubierto"}, \text{"Temp=Suave"}, \text{"Humedad=Alta"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_{13} = \{ \text{"Cielo=Cubierto"}, \text{"Temp=Alta"}, \text{"Humedad=Alta"}, \text{"Viento=Falso"}, \text{"Jugar=Si"} \},$   
 $t_{14} = \{ \text{"Cielo=Lluvioso"}, \text{"Temp=Suave"}, \text{"Humedad=Alta"}, \text{"Viento=Falso"}, \text{"Jugar=No"} \}$   
 $\}$

Consideramos **X** un conjunto de items de I si es un subconjunto de I. Debemos tener en cuenta que en caso de que los datos no sean binarios en el conjunto X no debe contener varios items que hagan referencia a un mismo atributo. Por ejemplo sea  $X_1$  un conjunto de items de tamaño 2 para el primer ejemplo (*figura 1*) y  $X_2$  uno para el segundo ejemplo (*figura 2*):

$X_1 = \{ \text{"Pan"}, \text{"Leche"}, \text{"Galletas"} \}$  ya que  $X_1 \subset I$  y  $X_1 \subset t_1$ .

$X_2 = \{ \text{"Cielo=Soleado"}, \text{"Viento=Cierto"} \}$  ya que  $X_2 \subset I$  y  $X_2 \subset t_2$ .

Por lo tanto, una regla de asociación se puede definir como una regla de la forma **SI X ENTONCES Y** ( $X \rightarrow Y$ ) donde X e Y son conjuntos de items disjuntos. En este tipo de reglas llamamos a X el predecesor y a Y el sucesor o consecuente. El sentido intuitivo de esta regla es que una transacción que contiene a X tiende a contener a Y.

En un problema real vamos a intentar generar reglas de asociación que nos muestren implicaciones entre items dentro de las transacciones existentes. Debemos tener en cuenta que hay un gran número de reglas a considerar aunque la base de datos sea pequeña. Trabajar con todas las reglas posibles sería inviable, por tanto, debemos utilizar alguna técnica que nos permita centrarnos solamente en las que nos interese. Para ello vamos a utilizar dos medidas para conocer la calidad de las mismas.

La primera es la **cobertura o soporte** de una regla (support) se define como el número de instancias que la regla predice correctamente (también se puede dar en porcentaje), es decir, el número de transacciones que contienen a X y a Y. La otra medida es la **confianza o precisión** (confidence) que consiste en medir el porcentaje de veces que la regla se cumple cuando se puede aplicar. Las transacciones en las que se puede aplicar la reglas son aquellas que contienen a X y las que cumplen la regla son las que contienen tanto a X como a Y.

Para nuestro problema de la cesta de la compra (*figura 1*), vamos a calcular la cobertura y la confianza de la regla “*SI Pan Y Leche ENTONCES Cacao*” para el donde  $X = \{“Pan”, “Leche”\}$  e  $Y = \{“Cacao”\}$ . Buscamos en  $D_I$  las transacciones que contienen a X y a Y, es decir, que contienen el conjunto {“Pan”, “Leche”, “Cacao”}. Obtenemos que esta regla tiene de cobertura 1 ya que solamente  $t_3$  contiene a X y a Y. Para calcular la confianza observamos que 2 transacciones contienen a X ( $t_3$  y  $t_6$ ), por tanto, la confianza de la regla será del 50% (se aplica correctamente una vez de las dos veces que se puede aplicar).

A parte de la cobertura y confianza, existen mas medidas de calidad de las reglas de asociación. A continuación hablaremos de algunas de ellas.

Sea  $X \rightarrow Y$  una regla de asociación, definimos el **lift** de la regla como la probabilidad de que ocurra X e Y cuando ocurren X e Y por separado:

$$lift = Pr(X,Y) / (Pr(X)*Pr(Y))$$

Si su valor es 1 entonces X e Y son independientes.

El **leverage** mide la proporción de casos adicionales cubiertos tanto por X como por la Y, sobre los casos en que se esperaba que X e Y fueran independientes:

$$leverage = Pr(X,Y) - Pr(X)*Pr(Y)$$

Los valores deseados de leverage son los que sean mayores que 0.

Para finalizar, la medida **conviction** hace referencia a la proporción de casos en los que se cumple X y no Y entre los casos que se cumplen ambos:

$$conviction = ( pr(X) * pr(not Y) ) / Pr(X,Y)$$



## ALGORITMOS DE APRENDIZAJE DE REGLAS DE ASOCIACIÓN

El conjunto de items a ser analizado en un problema crece exponencialmente con respecto al número de variables de los datos. Por este motivo y para optimizar el proceso de generación e interpretación de reglas podemos establecer ciertas restricciones sobre las reglas:

- Restricciones sintácticas. Estas restricciones limitan los items que pueden aparecer en una regla. Por ejemplo, podemos estar interesados sólo en las reglas que tengan un item específico en el consecuente o en el antecedente.
- Restricciones de soporte. Nuestro interés puede recaer en las reglas cuyos items aparezcan en un porcentaje de las transacciones de  $D$  con un soporte mínimo. Es decir, la regla debe de aparecer con una frecuencia mínima en la base de datos.
- Restricciones de cumplimiento. Normalmente interesa que la confianza de las reglas supere un mínimo.

Muchos de los algoritmos de aprendizaje de reglas se basan en estas restricciones para seleccionar las reglas que les interesa analizar. En los problemas reales nos encontramos con que existen pocos conjuntos de ítems frecuentes, de lo que se benefician los métodos que exigen una cobertura y confianza mínima.

## ALGORITMO APRIORI

El algoritmo Apriori quizás sea el algoritmo más popular. El funcionamiento de este algoritmo se basa en la búsqueda de los conjuntos de ítems con determinada cobertura. En primer lugar se crean los conjuntos de un ítem que superan la cobertura mínima. Este conjunto de conjuntos se utiliza para construir el conjunto de conjuntos de dos ítems, y así sucesivamente hasta que se llegue a un tamaño en el cual no existan conjuntos de ítems con la cobertura requerida.

Vamos a ver como sería el proceso en nuestro ejemplo de la cesta de la compra (*figura 1*) tomando como cobertura mínima 3.

En primer lugar creamos los conjuntos de sólo un ítem. En esta tabla existen 6 conjuntos y todos ellos tienen cobertura igual o superior a 3, por lo tanto tomaremos los 6. En la figura 3 observamos la cobertura de los conjuntos con 2 ítems y podemos comprobar que hay 6 conjuntos que poseen la cobertura mínima.

U	{“Pan”}	{“Leche”}	{“Cerveza”}	{“Pañales”}	{“Galletas”}	{“Cacao”}
{“Cacao”}	2	<b>3</b>	1	1	<b>3</b>	
{“Galletas”}	2	<b>3</b>	2	2		
{“Pañales”}	<b>3</b>	2	<b>3</b>			
{“Cerveza”}	<b>3</b>	0				
{“Leche”}	2					
{“Pan”}						

*Figura 3.- Cobertura de los conjuntos de 2 ítems.*

Ahora construimos los conjuntos de 3 ítems. Para ello, por cada columna de la tabla hacemos conjuntos de 3 elementos con los ítems filas que tengan cobertura mínima. En este caso, sólo tenemos 2 columnas que posean al menos 2 elementos con cobertura 3, por tanto, tendremos 2 conjuntos: {“Pan”, “Pañales”, “Cerveza”} y {“Leche”, “Cacao”, “Galletas”}. Ambos conjuntos poseen cobertura 2, por tanto, se finaliza el proceso.

Una vez se han seleccionado los conjuntos de ítems que cumplen la cobertura mínima, se continúa extrayendo de estos conjuntos las reglas que tengan un nivel de

confianza mínimo. Para ilustrar este procedimiento, vamos a suponer que nuestra cobertura mínima es de 2 y vamos a utilizar como ejemplo el conjunto {"Pan", "Pañales", "Cerveza"}. En la siguiente tabla (figura 4), podemos apreciar las reglas extraídas, acompañadas de su confianza.

Regla	Confianza
SI "Pan" Y "Pañales" ENTONCES "Cerveza"	2/3
SI "Pan" Y "Cerveza" ENTONCES "Pañales"	2/3
SI "Cerveza" Y "Pañales" ENTONCES "Pan"	2/3
SI "Pan" ENTONCES "Cerveza" Y "Pañales"	2/6
SI "Pañales" ENTONCES "Pan" Y "Cerveza"	2/5
SI "Cerveza" ENTONCES "Pan" Y "Pañales"	2/4
SI $\emptyset$ ENTONCES "Pan" Y "Pañales" Y "Cerveza"	2/9

Figura 4.- Reglas extraídas del conjunto {"Pan", "Pañales", "Cerveza"}

Ahora quedaría seleccionar las reglas que superen la confianza mínima que hayamos determinado. Si fijásemos la confianza mínima en un 50% seleccionaríamos las 3 primeras reglas.

Resumiendo, el aprendizaje de reglas de asociación se divide normalmente en dos fases: la extracción de los conjuntos de ítems que cumplan con la cobertura mínima requerida desde los datos, y la generación de las reglas a partir de estos conjuntos.

Para acelerar el proceso de búsqueda de los conjuntos de ítems se emplea la siguiente propiedad: "un conjunto de ítems formado por  $X$  ítems es frecuente si y sólo si cada uno de los  $X$  ítems es frecuente por sí solo. Ésto permite, crear los conjuntos de ítems de una manera incremental. En la figura 5 se muestra algoritmo.

```

ALGORITMO Apriori ( D: datos; MinC: cobertura mínima)
i=0
Rellena_Item(Ci) //Incluye en Co todos los ítems de tamaño 1
MIENTRAS Ci  $\neq \emptyset$ 
    PARA CADA x=elemento de Ci
        SI Cobertura(x)  $\geq$  MinC ENTONCES  $L_i = L_i \cup x$ 
    FINPARA
     $C_{i+1} = \text{Selecciona\_Candidatos}(L_i)$ 
    i=i+1;
FINMIENTRAS
DEVUELVE C
FIN ALGORITMO

```

*Figura 5.- Algoritmo de búsqueda de conjuntos de ítems (Apriori).*

A la hora de crear las reglas se puede comenzar creando reglas en las que el consecuente sólo contenga un ítem. De este modo, si el conjunto contiene  $i$  ítems, generaremos  $i$  reglas. Si a continuación deseamos crear reglas con más de un ítem en la parte derecha, podemos aprovechar la siguiente propiedad. Si tenemos una regla con varios ítems en la parte derecha, por ejemplo la siguiente regla obtenida a partir del ejemplo 1:

**SI** “Galletas” **Y** “Pan” **ENTONCES** “Cerveza” **Y** “Pañales”

Esta regla puede dividirse en dos reglas con un ítem en la parte derecha:

**SI** “Galletas” **Y** “Pan” **ENTONCES** “Cerveza”

**SI** “Galletas” **Y** “Pan” **ENTONCES** “Pañales”

La cobertura de ambas reglas será idéntica a la regla anterior, por tener la misma parte izquierda. En cuanto a la confianza, se puede asegurar que será igual o mayor a la regla con dos ítems en la parte derecha, ya que el número de registros que cumplan la parte izquierda en los que aparezca “Cerveza” será mayor o igual al número de registros que cumplan la parte izquierda en los que aparezcan “Cerveza” y “Pañales”.

De la misma manera, podemos observar que para que una regla con dos ítems en la parte derecha (  $X \rightarrow Z \ Y \ S$  ) cumpla los requisitos de cobertura y confianza es necesario que las reglas con la misma parte izquierda y cada uno de los ítems de la derecha también los cumplan. Este hecho facilita la generación de reglas con más de un ítem en la parte derecha, ya que podemos hacerlo de manera incremental.

Existen variantes de este algoritmo que permiten mejorar la búsqueda de reglas de asociación basándose en el perfeccionamiento de los punto clave que constituyen un mayor consumo de recursos. Para reducir el número de accesos a la base de datos se suelen utilizar estructuras de datos complejas como por ejemplo tablas hash o introducir estructuras de árbol. De este último tipo podemos mencionar el uso de la estructura árbol-FP "*Frequent Pattern Tree*" [Han et al. 2000] que permite mejorar la generación y prueba de los conjuntos de ítems candidatos.

Otra técnica que nos permite optimizar el tiempo podría ser, el uso de una técnica de muestreo. En este caso, en lugar de trabajar con todas las transacciones de la base de datos, trabajaríamos con una muestra representativa de la misma.

Algo similar al muestreo, sería realizar un filtro sobre los atributos para rechazar aquellos que se consideren poco relevantes. Esta técnica recibe el nombre de selección de atributos (*feature selection*) que se puede emplear para reducir la complejidad de cualquier problema de minería de datos.

En caso de tener que trabajar con una base de datos muy grande, existe la posibilidad de trabajar con técnicas de paralelización. Estas técnicas modifican los algoritmos de manera que se puedan ejecutar en varios procesadores.

Finalmente, las reglas de asociación en su versión original sólo trabajan con atributos discretos, es decir, no soportan el uso de atributos numéricos. Para subsanar esta limitación, existen varias técnicas que normalmente tratan de convertir a tipo categórico los atributos de tipo numéricos (discretizar). Una técnica puede ser la de discretizar a partir de unos rangos preestablecidos. Otra más completa, sería usar técnicas de agrupamiento asignando a cada grupo un valor.

## VARIANTES DEL ALGORITMO APRIORI

El algoritmo Apriori-TID, es un algoritmo derivado del Apriori. Este algoritmo utiliza la misma función de generación de candidatos. La diferencia con el Apriori reside en que, en lugar de acceder a la base de datos cada vez que se va a crear un conjunto de items candidato, se realiza un escaneado principal. A partir de este escaneado, se crean unos conjuntos auxiliares que van a poseer un identificador y después se van a crear los conjuntos de items a partir ellos. Como podemos deducir, la mejora que proporciona este algoritmo es computacional ya que disminuye el tiempo de acceso a la base de datos.

A parte de las reglas de asociación existen las reglas de asociación secuenciales. Este tipo de reglas expresan patrones de comportamiento secuenciales, es decir, que se dan en instantes distintos de tiempo. Este tipo de información se puede aplicar en muchas campos y tiene especial utilidad en el análisis de navegación sobre páginas web donde nos puede interesar estudiar relaciones del tipo: el 55% de personas que entran en la pagina de consultar precios a los 2 días realizan una compra. También es útil en distribución y en marketing ya que pueden interesar reglas del estilo: si un cliente compra un microondas y un frigorífico entonces en menos de seis meses compra una lavadora.

Para este tipo de problemas se va a trabajar con una base de datos temporal, es decir, vamos a guardar para cada transacción el identificador del cliente, un identificador de tiempo y el conjunto de items de la transacción. El aprendizaje de las reglas de asociación secuenciales se basa en encontrar las secuencias más comunes. Una secuencia se define formalmente como una lista de items de un mismo cliente ordenada por el tiempo. En este ámbito, la cobertura se define como la proporción de clientes que poseen esa determinada secuencia. Por tanto, el aprendizaje de reglas de asociación secuenciales se basa en encontrar el conjunto de secuencias que cumplan una cobertura mínima.

Uno de los algoritmos más popular, para el aprendizaje de este tipo de reglas, es el *AprioriAll*. Este algoritmo se divide en cinco fases:

- Ordenación: se construye una secuencia de items por cada cliente. Es decir, vamos a ordenar la base por cliente y dentro de cada cliente por el identificador de tiempo.
- Selección de conjuntos de items: se seleccionan los conjuntos de items que cumplen con una cobertura mínima utilizando la tabla construida anteriormente.
- Transformación y renombramiento: se le asigna a cada conjunto de items frecuente un identificador. Cada secuencia se transforma de manera que sólo contenga sus items frecuentes. A continuación, se renombra cada conjunto por su identificador en cada una de las secuencias.
- Construcción de secuencias frecuente: a partir del conjunto de items frecuente se construye incrementalmente el conjunto de secuencias que cumplan con el criterio de cobertura.
- Selección de secuencias máximas: filtra el conjunto de secuencias frecuentes de manera que no haya subsecuencias partiendo desde las secuencias de mayor tamaño.

## EJEMPLOS DE EJECUCIÓN DEL ALGORITMO A PRIORI

A continuación se procederá a la ejecución del algoritmo a priori con diferentes implementaciones. En un primer lugar, se utilizará el software Weka y en un segundo lugar el paquete arules que proporciona el lenguaje de programación R.

Para la ejecución de este algoritmo se usará nuevamente la base de datos del credit scoring, cuyas características se detallan en el anexo 1.

Se presentará inicialmente, tanto en el caso del paquete arules de R como del software Weka, una descripción detallada de los diferentes parámetros que pueden ser modificados en la ejecución del algoritmo y que, previsiblemente producirán salidas diferentes.

A continuación, se comprobará sobre los datos como, para diferentes valores tanto de confianza como de soporte, se obtendrá un número de reglas diferente.



## EJEMPLO A PRIORI CON WEKA

Al iniciar el análisis con Weka, aparecen los siguientes parámetros:

- **car**: para extraer reglas de asociación para la clase en lugar de reglas generales.
- **classIndex**: índice del atributo clase.
- **delta**: reduce iterativamente el soporte hasta que el soporte mínimo es alcanzado o se alcanza un número mínimo de reglas.
- **lowerBoundMinSupport** – límite inferior para el soporte mínimo
- **metricType**: tipo de medida para clasificar las reglas; **confidence** (proporción de casos cubiertos por la premisa que son cubiertos por el consecuente; las reglas de asociación de clase sólo pueden ser extraídas en base a este criterio de la confianza), **lift** (confianza dividida por la proporción de ejemplos cubiertos por el consecuente), **leverage** (proporción de ejemplos adicionales cubiertos tanto por premisa como por consecuente sobre los considerados para premisa y consecuente independientemente) y **conviction**.
- **minMetric**: se consideran sólo aquellas en las que el criterio establecido supera este valor.
- **numRules**: número de reglas.
- **outputItemSets**: para mostrar también el conjunto de items.
- **removeAllMissingCols**: elimina columnas con datos faltantes.
- **significanceLevel**: nivel de significación (solo si el criterio es la confianza)
- **upperBoundMinSupport**: límite superior para el soporte mínimo.
- **verbose**

A continuación se muestra la salida del weka, producida con diferentes valores de confianza y de soporte, además se varía el parámetro CAR, con el que se buscan reglas generales o reglas para la toma de decisión sobre la concesión o no concesión de crédito, según adopte el valor verdadero o falso.

**Car=FALSE -> 5 reglas con confianza mínima del 90 %**

**=== Run information ===**

Scheme: weka.associations.Apriori -N 5 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -c -1

Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-

R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-

weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last

Instances: 1000

Attributes: 21

**=== Associator model (full training set) ===**

Apriori

=====

Minimum support: 0.75 (750 instances)

Minimum metric <confidence>: 0.9

Number of cycles performed: 5

Generated sets of large itemsets:

Size of set of large itemsets L(1): 4

Size of set of large itemsets L(2): 4

Size of set of large itemsets L(3): 1

Best rules found:

1. avales=1 unidadFamiliar=1 767 ==> extranjero=1 750 conf:(0.98)

2. avales=1 907 ==> extranjero=1 881 conf:(0.97)

3. unidadFamiliar=1 845 ==> extranjero=1 819 conf:(0.97)

4. otrosPrestatarios=3 814 ==> extranjero=1 783 conf:(0.96)

5. unidadFamiliar=1 extranjero=1 819 ==> avales=1 750 conf:(0.92)

Car=**TRUE** -> 5 reglas con confianza mínima del 90 %

=== **Run information** ===

Scheme: weka.associations.Apriori -N 5 -T 0 -C 0.9 -D 0.05 -U 1.0 -M 0.1 -S -1.0 -A -c -1

Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-

R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-

weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last

Instances: 1000

Attributes: 21

=== **Associator model (full training set)** ===

Apriori

=====

Minimum support: 0.25 (250 instances)

Minimum metric <confidence>: 0.9

Number of cycles performed: 15

Generated sets of large itemsets:

Size of set of large itemsets L(1): 20

Size of set of large itemsets L(2): 65

Size of set of large itemsets L(3): 94

Size of set of large itemsets L(4): 50

Size of set of large itemsets L(5): 8

Best rules found:

1. saldoCta=4 avales=1 otrosPrestatarios=3 317 ==> decision=1 290 conf:(0.91)
2. saldoCta=4 avales=1 otrosPrestatarios=3 extranjero=1 308 ==> decision=1 281 conf:(0.91)
3. saldoCta=4 otrosPrestatarios=3 334 ==> decision=1 303 conf:(0.91)
4. saldoCta=4 otrosPrestatarios=3 unidadFamiliar=1 288 ==> decision=1 261 conf:(0.91)
5. saldoCta=4 otrosPrestatarios=3 extranjero=1 323 ==> decision=1 292 conf:(0.9)

Car=**TRUE** -> 5 reglas con confianza mínima del 80 % pero soporte mínimo del 33%

=== **Run information** ===

Scheme: weka.associations.Apriori -N 5 -T 0 -C 0.8 -D 0.05 -U 1.0 -M 0.33 -S -1.0 -A -c -1  
Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
Instances: 1000  
Attributes: 21

=== **Associator model (full training set)** ===

Apriori

=====

Minimum support: 0.33 (330 instances)

Minimum metric <confidence>: 0.8

Number of cycles performed: 14

Generated sets of large itemsets:

Size of set of large itemsets L(1): 13

Size of set of large itemsets L(2): 34

Size of set of large itemsets L(3): 24

Size of set of large itemsets L(4): 9

Size of set of large itemsets L(5): 1

Best rules found:

1. saldoCta=4 avales=1 373 ==> decision=1 331 conf:(0.89)

2. saldoCta=4 394 ==> decision=1 348 conf:(0.88)

3. saldoCta=4 extranjero=1 382 ==> decision=1 336 conf:(0.88)

Car=**TRUE** -> 5 reglas con confianza mínima del 95 % pero soporte mínimo del 12,5%

=== **Run information** ===

Scheme: weka.associations.Apriori -N 5 -T 0 -C 0.95 -D 0.05 -U 1.0 -M 0.125 -S -1.0 -A -c -1  
Relation: credit\_weka-weka.filters.unsupervised.attribute.Reorder-  
R2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1-  
weka.filters.unsupervised.attribute.NumericToNominal-Rfirst-last  
Instances: 1000  
Attributes: 21

=== **Associator model (full training set)** ===

Apriori

=====

Minimum support: 0.13 (125 instances)

Minimum metric <confidence>: 0.95

Number of cycles performed: 18

Generated sets of large itemsets:

Size of set of large itemsets L(1): 53

Size of set of large itemsets L(2): 314

Size of set of large itemsets L(3): 689

Size of set of large itemsets L(4): 768

Size of set of large itemsets L(5): 430

Size of set of large itemsets L(6): 111

Size of set of large itemsets L(7): 8

Best rules found:

1. saldoCta=4 ecivil\_sexo=3 avales=1 otrosPrestatarios=3 vivienda=2 148 ==> decision=1 142  
conf:(0.96)

2. saldoCta=4 ecivil\_sexo=3 avales=1 otrosPrestatarios=3 vivienda=2 extranjero=1 143 ==> decision=1  
137 conf:(0.96)

3. saldoCta=4 pagoAnteriores=4 otrosPrestatarios=3 137 ==> decision=1 131 conf:(0.96)

4. saldoCta=4 pagoAnteriores=4 otrosPrestatarios=3 extranjero=1 134 ==> decision=1 128 conf:(0.96)

5. saldoCta=4 ecivil\_sexo=3 otrosPrestatarios=3 vivienda=2 155 ==> decision=1 148 conf:(0.95)

## EJEMPLO A PRIORI CON PAQUETE ARULES DE R

El paquete `arules` de R proporciona una infraestructura para representar, manipular y analizar datos transaccionales y patrones (normalmente conjuntos de ítems y reglas de asociación). Así mismo, proporciona interfaces para las implementaciones en C de los algoritmos de asociación de reglas A priori y Eclat realizadas por C. Borglet. A continuación se hará una descripción detallada de la función `a priori`, contenida en el paquete `arules`, para luego analizar los datos del credit scoring.

La función `arules` permite la inclusión de cuatro posibles parámetros, que se explicarán a continuación:

*`apriori(data, parameter = NULL, appearance = NULL, control = NULL)`*

- **data:** objeto de la clase transacción o que se pueda convertir en la clase transacción (por ejemplo: matrices binarias o `data.frames`)
- **parameter:** objeto de la clase `APparameter` o una lista nombrada. Por defecto se construyen las reglas con un soporte del 0.1, una confianza del 0.8 y longitud máxima igual a 5.
- **appearance:** objeto de la clase `APappearance` o una lista nombrada, con este parámetro conseguimos que la apariencia de los ítems pueda ser restringida. Por defecto todos los ítems aparecen con la `appearance` restringida.
- **control:** objeto de la clase `Apcontrol` o una lista nombrada, que permite controlar la ejecución del algoritmo; como por ejemplo el ordenamiento de ítems.

A continuación se hará un análisis de los datos del credit scoring, para ello se usará el script mostrado en el anexo número dos.

Primeramente, se intenta con una confianza del 0.9 y un soporte del 0.25. Se muestra a continuación el código R usado y su salida:

```
> a1=apriori(credit,parameter = list(supp = 0.25, conf = 0.9,target="rules"))
```

parameter specification:

confidence	minval	smax	arem	aval	originalSupport	support	minlen	maxlen
0.9	0.1	1	none	FALSE	TRUE	0.25	1	5

target ext

rules FALSE

algorithmic control:

filter tree heap memopt load sort verbose

0.1 TRUE TRUE FALSE TRUE 2 TRUE

apriori - find association rules with the apriori algorithm

version 4.21 (2004.05.09) (c) 1996-2004 Christian Borgelt

set item appearances ...[0 item(s)] done [0.00s].

set transactions ...[94 item(s), 1000 transaction(s)] done [0.00s].

sorting and recoding items ... [31 item(s)] done [0.00s].

creating transaction tree ... done [0.00s].

checking subsets of size 1 2 3 4 5 done [0.01s].

writing ... [743 rule(s)] done [0.00s].

creating S4 object ... done [0.00s].

```
> a1.sub <- subset(a1, subset = rhs %pin% "decision" )#para coger solo los
antecedentes o consecuentes(lhs,rhs) que contengan Kredit
```

```
> a1.sub
```

set of 7 rules

```
> inspect(SORT(a1.sub)[1:7])
```

	lhs	rhs	support	confidence	lift
1	{saldoCta=4, otrosPrestatarios=3}	=> {decision=1}	0.303	0.9071856	1.295979
2	{saldoCta=4, otrosPrestatarios=3, extranjero=1}	=> {decision=1}	0.292	0.9040248	1.291464
3	{saldoCta=4,avales=1,otrosPrestatarios=3}	=> {decision=1}	0.290	0.9148265	1.306895
4	{saldoCta=4,avales=1,otrosPrestatarios=3,extranjero=1}	=> {decision=1}	0.281	0.9123377	1.303340
5	{saldoCta=4,otrosPrestatarios=3,unidadFamiliar=1}	=>{decision=1}	0.261	0.90625	1.294643
6	{saldoCta=4, otrosPrestatarios=3, unidadFamiliar=1, extranjero=1}	=> {decision=1}	0.254	0.9039146	1.291307
7	{saldoCta=4,avales=1,otrosPrestatarios=3,unidadFamiliar=1}	=>{decision=1}	0.250	0.91240	1.303441

Para finalizar, se intenta con una confianza del 0.95 y un soporte del 0.25. Se muestra a continuación el código R usado y su salida, viendo como se reduce el número de reglas:

```
> a2=apriori(credit,parameter = list(supp = 0.15, conf = 0.8,target="rules"))
```

parameter specification:

confidence	minval	smax	arem	aval	originalSupport	support	minlen	maxlen
0.8	0.1	1	none	FALSE	TRUE	0.15	1	5
target	ext							
rules	FALSE							

algorithmic control:

filter	tree	heap	memopt	load	sort	verbose
0.1	TRUE	TRUE	FALSE	TRUE	2	TRUE

```
apriori - find association rules with the apriori algorithm
version 4.21 (2004.05.09)      (c) 1996-2004  Christian Borgelt
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[94 item(s), 1000 transaction(s)] done [0.00s].
sorting and recoding items ... [49 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 done [0.03s].
writing ... [6552 rule(s)] done [0.00s].
creating S4 object ... done [0.00s].
```

```
> a2.sub <- subset(a2, subset = rhs %pin% "decision" )#para coger solo los
antecedentes o consecuentes(lhs,rhs) que contengan posicion
```

```
> a2.sub
set of 161 rules
```

```
> inspect(SORT(a2.sub)[1:4])
```

	lhs	rhs	support	confidence	lift
1	{saldoCta=4}	=> {decision=1}	0.348	0.8832487	1.261784
2	{saldoCta=4, extranjero=1}	=> {decision=1}	0.336	0.8795812	1.256545
3	{saldoCta=4, avales=1}	=> {decision=1}	0.331	0.8873995	1.267714
4	{saldoCta=4, avales=1, extranjero=1}	=> {decision=1}	0.321	0.8842975	1.263282



## ANEXOS

### ANEXO 1 : DESCRIPCIÓN BASE DE DATOS CREDIT SCORING

Variable	Description	Categories	Score	rel. frequency in % for	
				good credits	bad credits
decision	Creditability: 1 : credit-worthy 0 : not credit-worthy				
saldoCta	Balance of current account	no balance or debit 0 <= ... < 200 DM ... >= 200 DM or checking account for at least 1 year no running account	2 3 4 1	36 4,67 15,33 46	23,43 7 49,71 19,86
meses	Duration in months (metric)				
meses_Categ	Duration in months (categorized)	<=6 6 < ... <= 12 12 < ... <= 18 18 < ... <= 24 24 < ... <= 30 30 < ... <= 36 36 < ... <= 42 42 < ... <= 48 48 < ... <= 54 > 54	10 9 8 7 6 5 4 3 2 1	3 22,33 18,67 22 6,33 12,67 1,67 10,67 0,33 2,33	10,43 30 18,71 22,57 5,43 6,86 1,71 3,14 0,14 1
pagoAnteriores	Payment of previous credits	no previous credits / paid back all previous credits paid back previous credits at this bank no problems with current credits at this bank hesitant payment of previous credits problematic running account / there are further credits running but at other banks	2 4 3 0 1	56,33 16,67 9,33 8,33 9,33	51,57 34,71 8,57 2,14 3
objeto	Purpose of credit	new car used car items of furniture radio / television household appliances repair education vacation retraining business other	1 2 3 4 5 6 7 8 9 10 0	5,67 19,33 20,67 1,33 2,67 7,33 0 0,33 11,33 1,67 29,67	12,29 17,57 31,14 1,14 2 4 0 1,14 9 1 20,71
otrosCreditos	Amount of credit in "Deutsche Mark" (metric)				
otrosCreditos_Categ	Amount of credit in DM (categorized)	<=500 500 < ... <= 1000 1000 < ... <= 1500 1500 < ... <= 2500 2500 < ... <= 5000 5000 < ... <= 7500 7500 < ... <= 10000 10000 < ... <= 15000 15000 < ... <= 20000 > 20000	10 9 8 7 6 5 4 3 2 1	1 11,33 17 19,67 26 11,33 6,67 7 1 0	2,14 9,14 19,86 24,57 28,57 9,71 3,71 2 0,29 0
ahorros	Value of savings or stocks	< 100,- DM 100,- <= ... < 500,- DM 500,- <= ... < 1000,- DM >= 1000,- DM not available / no savings	2 3 4 5 1	11,33 3,67 2 10,67 72,33	9,86 7,43 6 21,57 65,14

antigEmpleo	Has been employed by current employer for	unemployed	1	7,67	6,57
		<= 1 year	2	23,33	14,57
		1 <= ... < 4 years	3	34,67	33,57
		4 <= ... < 7 years	4	13	19,29
		>= 7 years	5	21,33	27
ingresosDisp	Installment in % of available income	>= 35	1	11,33	14,57
		25 <= ... < 35	2	20,67	24,14
		20 <= ... < 25	3	16	16
		< 20	4	53	45,29
ecivil_sexo	Marital Status / Sex	male: divorced / living apart	1	6,67	4,29
		female: divorced / living apart / married	2	11,33	10,29
		male: single	2	26	18,43
		male: married / widowed	3	48,67	57,43
		female: single	4	8,33	9,57
avales	Further debtors / Guarantors	none	1	90,67	90,71
		Co-Applicant	2	6	3,29
		Guarantor	3	3,33	6
antigCasa	Living in current household for	< 1 year	1	12	13,43
		1 <= ... < 4 years	2	32,33	30,14
		4 <= ... < 7 years	3	14,33	16,14
		>= 7 years	4	41,33	41,29
pertenencias	Most valuable available assets	Ownership of house or land	4	22,33	12,43
		Savings contract with a building society / Life insurance	3	34	32,86
		Car / Other	2	23,67	23
		not available / no assets	1	20	31,71
edad	Age in years (metric)				
edad_categ	Age in years (categorized)	0 <= ... <= 25	1	26,67	15,71
		26 <= ... <= 39	2	47,33	52,72
		40 <= ... <= 59	3	21,67	26,14
		60 <= ... <= 64	5	2,33	3
		>= 65	4	2	2,43
otrosPrestatarios	Further running credits	at other banks	1	19	11,71
		at department store or mail order house	2	6,33	4
		no further running credits	3	74,67	84,29
Vivienda	Type of apartment	rented flat	2	62	75,43
		owner-occupied flat	3	14,67	9,14
		free apartment	1	23,33	15,57
numCredsAntsBanco	Number of previous credits at this bank (including the running one)	one	1	66,67	61,86
		two or three	2	30,67	34,43
		four or five	3	2	3,14
		six or more	4	0,67	0,57
ocupacion	Occupation	unemployed / unskilled with no permanent residence	1	2,33	2,14
		unskilled with permanent residence	2	18,67	20,57
		skilled worker / skilled employee / minor civil servant	3	62	63,43
		executive / self-employed / higher civil servant	4	17	13,86
unidadFamiliar	Number of persons entitled to maintenance	0 to 2	2	84,67	84,43
		3 and more	1	15,33	15,57
telef	Telephone	no	1	62,33	58,43
		yes	2	37,67	41,57
extranjero	Foreign worker	yes	1	1,33	4,71
		no	2	98,67	95,29

## ANEXO 2: SCRIPT DE EJECUCIÓN DEL APRIORI CON R

```
#carga credito
credit<-read.table("C:/Users/karina/Desktop/master/anal expl
dat/proyecto/DATOS/credit_weka_categF.txt",header=TRUE)
attach(credit)

credit$decision <- as.factor(credit$decision);credit$decision
credit$saldoCta <- as.factor(credit$saldoCta);credit$saldoCta
credit$meses_Categ <- as.factor(credit$meses_Categ);credit$meses_Categ
credit$pagoAnteriores <- as.factor(credit$pagoAnteriores);credit$pagoAnteriores
credit$objeto <- as.factor(credit$objeto);credit$objeto
credit$otrosCreditos_Categ <- as.factor(otrosCreditos_Categ);otrosCreditos_Categ
credit$ahorros <- as.factor(ahorros);ahorros
credit$antigEmpleo <- as.factor(antigEmpleo);antigEmpleo
credit$ingresosDisp <- as.factor(ingresosDisp);ingresosDisp
credit$ecivil_sexo <- as.factor(ecivil_sexo);ecivil_sexo
credit$avales <- as.factor(avales);avales
credit$antigCasa <- as.factor(antigCasa);antigCasa
credit$pertenencias <- as.factor(pertenencias);pertenencias
credit$edad_Categ <- as.factor(edad_Categ);edad_Categ
credit$otrosPrestatarios <- as.factor(otrosPrestatarios);otrosPrestatarios
credit$vivienda <- as.factor(vivienda);vivienda
credit$numCredsAntsBanco<- as.factor(numCredsAntsBanco);numCredsAntsBanco
credit$ocupacion <- as.factor(ocupacion);ocupacion
credit$unidadFamiliar <- as.factor(unidadFamiliar);unidadFamiliar
credit$telef <- as.factor(telef);telef
credit$extranjero <- as.factor(extranjero);extranjero

library(arules)

a1=apriori(credit,parameter = list(supp = 0.25, conf = 0.9,target="rules"))
a1.sub <- subset(a1, subset = rhs %pin% "decision" )#para coger solo los antecedentes o
consecuentes(lhs,rhs) que contengan Kredit
a1.sub
inspect(SORT(a1.sub)[1:7])

a2=apriori(credit,parameter = list(supp = 0.15, conf = 0.8,target="rules"))
a2.sub <- subset(a2, subset = rhs %pin% "decision" )#para coger solo los antecedentes o
consecuentes(lhs,rhs) que contengan Kredit
a2.sub
inspect(SORT(a2.sub)[1:4])
```

## BIBLIOGRAFÍA

- “Reglas de Asociación” de Carlos Alonso González del Grupo de Sistemas Inteligentes, Departamento de Informática, de la Universidad de Valladolid.  
<http://www.infor.uva.es/~calonso/MUITIC/MineriaDatos/01IntroduccionMDyAA.pdf>
- “Minería de Datos” de José L. Balcázar del Departamento Matesco de la Universidad de Cantabria.  
<http://personales.unican.es/balcazarjl/181109.pdf>
- “Implantación de Primitivas SQL para el Descubrimiento de Reglas de Asociación y Clasificación al interior del motor PostgreSQL” dirigido por Ricardo Timarán Pereira de la Universidad de Manizales.  
<http://www.umanizales.edu.co/programs/ingenieria/Ventana/ventana12/articulo24.pdf>
- “Reglas de asociación aplicadas a la detección de fraude con tarjetas de créditos” de María-- Amparo Vila Miranda, Daniel Sánchez Fernández y Luis Cerda Leiva.  
<http://www.google.es/url?sa=t&source=web&ct=res&cd=1&ved=0CAkQFjAA&url=http%3A%2F%2Fdecsai.ugr.es%2F~castro%2Fdoctocsi%2FLuis%2520Cerde%2Fp79.pdf&ei=DrdiS4LvEsir4QbImLjrBg&usq=AFQjCNH73x8hoTcJxFdVXZ7t9m6zr9PSLg&sig2=crGKjtR12cEOmMdN0tVhWA>
- “Propuesta de proyecto Algoritmos A priori y A priori TID. Minería de Datos” de Claudia Bello P. y Maryurith Inzulza C.  
[http://educnet.decomuv.cl/educnet/uploads/inzulza\\_bello.pdf?nombre=p373/inzulza\\_bello.pdf](http://educnet.decomuv.cl/educnet/uploads/inzulza_bello.pdf?nombre=p373/inzulza_bello.pdf)
- “Data Mining. Theory and Practice” de K.P. Soman, Shyam Diwakar y V. Ajay.  
[http://books.google.es/books?id=IYc2muhCbmEC&pg=PT205&lpg=PT205&dq=conviction+data+mining&source=bl&ots=jnorabrJD&sig=MkN4na5TIILbSsqk28REB5n8EgI&hl=es&ei=x8xhS7PNG4bv4gbb5qT1Cw&sa=X&oi=book\\_result&ct=result&resnum=6&ved=0CCQQ6AEwBTgK#v=onepage&q=leverage&f=false](http://books.google.es/books?id=IYc2muhCbmEC&pg=PT205&lpg=PT205&dq=conviction+data+mining&source=bl&ots=jnorabrJD&sig=MkN4na5TIILbSsqk28REB5n8EgI&hl=es&ei=x8xhS7PNG4bv4gbb5qT1Cw&sa=X&oi=book_result&ct=result&resnum=6&ved=0CCQQ6AEwBTgK#v=onepage&q=leverage&f=false)
- J. Furnkranz, G. Widner (1994). Incremental Reduced Error Pruning, Proceedings of the Eleventh International Conference on Machine Learning.

- W.W. Cohen (1995). Fast effective rule induction, Proceedings of the Twelfth International Conference on Machine Learning.
- Mark Hall and Eibe Frank (2008). Combining Naive Bayes and Decision Tables, Proceedings of the 21st Florida Artificial Intelligence Society Conference (FLAIRS).
- Arun George Eapen (2004). Application of Data mining in Medical Applications.
- John O'Donovan (2003). A Comparison of Collaborative Recommendation Algorithms Over Diverse Data Types.
- Eibe Frank, Ian H. Witten (1998). Generating Accurate Rule Sets Without Global Optimization. In: Fifteenth International Conference on Machine Learning, 144-151.
- Brent Martin (1995). Instance-Based Learning Nearest Neighbour with Generalisation.
- Rajeev Kohli, Ramesh Krishnamurti and kamel Jedidi (2004). Subset-conjunctive rules for breast cancer diagnosis
- Documentación sobre el paquete arules de R en la web:  
<http://cran.r-project.org/>
- Documentación del propio paquete Weka