RNPS No. 2143
ISSN 2072-6260
Versión Digital

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada
MINISTERIO DE LA INDUSTRIA BÁSICA

REPORTE TÉCNICO
Minería de Datos

SERIE GRIS

Clasificación basada en reglas de asociación de clase para flujos de datos: un estado del arte

Yaniela Fernández Mena,
Raudel Hernández León y
José Hernández Palancar

RT_021 **noviembre 2013**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

RNPS No. 2143
ISSN 2072-6260
Versión Digital

SERIE GRIS

REPORTE TÉCNICO
Minería de Datos

Clasificación basada en reglas de asociación de clase para flujos de datos: un estado del arte

Yaniela Fernández Mena,
Raudel Hernández León y
José Hernández Palancar

RT_021 **noviembre 2013**

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada
MINISTERIO DE LA INDUSTRIA BÁSICA

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

Tabla de contenido

Lista de tablas

Clasificación basada en reglas de asociación de clase para flujos de datos: un estado del arte

Yaniela Fernández Mena[1], Raudel Hernández León[2] y José Hernández Palancar[2]

[1] Universidad de las Ciencias Informáticas (UCI).
La Habana, Cuba.
yaniela@uci.cu
[2] Dpto. de Minería de Datos. Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV).
La Habana, Cuba.
{rhernandez,jpalancar}@cenatav.co.cu

RT_021 CENATAV
Aceptado: 19 de agosto de 2013

Resumen. En la actualidad se ha incrementado el número de aplicaciones que generan datos constantemente (flujos de datos). Varias son las técnicas de Minería de Datos que se han adaptado para el procesamiento de flujos de datos. En particular, la clasificación en flujos de datos ha sido muy abordada mediante árboles de decisión y métodos de ensamble de clasificadores. Otras técnicas han recibido menor atención, tal es el caso de los métodos basados en reglas, particularmente los basados en Reglas de Asociación de Clase (CARs), las cuales son preferidas por muchos especialistas debido a la fácil comprensión de las mismas. Adicionalmente, los clasificadores basados en CARs han demostrado ser competitivos con respecto a otras técnicas como los árboles de decisión.
Este reporte aborda los aspectos teóricos para la construcción de clasificadores basados en CARs, resume los clasificadores más representativos y ofrece una comparativa entre estos teniendo en cuenta sus principales características.

Palabras clave: clasificación, flujos de datos, reglas de asociación de clase.

Abstract. Currently, the number of applications generating data streams has been increased. Several Data Mining techniques have been adapted to process data streams. Specifically, classification has been widely discussed by means of decision trees and ensemble classifiers. Other techniques have received less attention, such is the case of rule-based methods, particularly those based on Class Association Rules (CARs), which are preferred for many researchers due to their easy understanding. Additionally, the CAR based classifiers have shown to be very competitive with respect to other techniques such as decision trees.
This report discusses the theoretical aspects for building CAR based classifiers, summarizes the most representative ones and provides a comparison among them taking into account their main features.

Keywords: classification, data stream, class association rules.

1. Introducción

La Clasificación Supervisada o simplemente Clasificación, es una técnica de Minería de Datos cuyo objetivo es construir, a partir de un conjunto etiquetado de objetos, instancias o transacciones, un modelo o clasificador que permita asignar una clase a nuevas transacciones. Al construir un

clasificador, los datos pueden ser finitos y estar todos disponibles (datos estáticos), o bien pueden ser infinitos y arribar de forma prácticamente constante (flujos de datos).

Los clasificadores para flujos de datos, a diferencia de los clasificadores para datos estáticos, no disponen de todo el conjunto de transacciones etiquetadas (conjunto de entrenamiento) a priori, sino que estas arriban de forma incremental a lo largo del tiempo. Por otro lado, dado que los flujos de datos son, en teoría, infinitos, resulta imposible cargar todas las transacciones en memoria y se requiere de un algoritmo incremental que actualice el clasificador con la información de las nuevas transacciones, reutilizando la información extraída de las transacciones previas. En general, los clasificadores para flujos de datos [1,8,23] deben garantizar que:

1. Cada transacción del flujo de datos sea procesada a lo sumo una vez.
2. Los resultados de clasificación estén disponibles en todo momento.
3. El modelo resultante sea consistente con las nuevas transacciones que llegan, pues los datos pueden variar a lo largo del tiempo debido a cambios en su distribución de probabilidad.

La clasificación en flujos de datos tiene diversas aplicaciones como: detección de software malicioso [46], clasificación de paquetes en el área de las telecomunicaciones [53], monitoreo de procesos industriales [4,37], monitoreo de datos de navegación en vehículos automotrices [38], detección de fraudes en transacciones bancarias [65], entre otros.

Varios han sido los clasificadores adaptados al entorno de flujos de datos. Entre los más reportados se encuentran los árboles de decisión[19,25,26,34,36,59,61] y los métodos de ensamble de clasificadores [9,39,47,51,57]. Otros clasificadores que han recibido menor atención son los basados en el vecino más cercano [5,54], las máquinas de vectores de soporte [43,66] y los basados en reglas. Particularmente, los clasificadores basados en Reglas de Asociación de Clase (CARs), son preferidos por muchos especialistas debido a su interpretabilidad, aspecto que los hace más expresivos y fáciles de comprender. Su interpretabilidad permite a los especialistas modificar las reglas con base en su experiencia y así mejorar la eficacia del clasificador. Además de los clasificadores basados en CARs, los árboles de decisión también generan reglas comprensibles. Para construir un clasificador utilizando árboles de decisión se sigue una estrategia voraz seleccionando en cada momento la característica que mejor separa las clases. Sin embargo, esta estrategia voraz puede podar reglas interesantes. En [55], los autores probaron que las reglas obtenidas de los árboles de decisión son un subconjunto de las reglas generadas por los clasificadores basados en CARs, asumiendo un umbral relativamente bajo de concurrencia (Soporte) de los elementos que componen la regla.

En general, los clasificadores basados en CARs calculan un conjunto de reglas a partir de un conjunto de entrenamiento. Una vez extraídas las CARs, se ordenan y se determina un criterio de decisión que permite asignar una clase a una nueva transacción. Varios trabajos [42,62] han proporcionado evidencias de que los clasificadores basados en CARs son competitivos con los clasificadores probabilísticos [20], con los basados en árboles de decisión [49] y con los basados en inducción de reglas [17,50].

El cálculo de las CARs en un entorno de flujos de datos es una extensión de los trabajos realizados para calcular conjuntos frecuentes de ítems (FIs por sus siglas en inglés) en flujos de datos, siendo el consecuente de la regla (la clase) el último ítem del FI calculado.

En este reporte se describirán los principales algoritmos desarrollados para calcular FIs (son los mismos que se utilizan para calcular las CARs) en flujos de datos, así como los clasificadores reportados, basados en CARs. El reporte consta de 4 sesiones. En la Sección 2 se presentan los

conceptos básicos necesarios para el resto del documento. En la Sección 3 se aborda el trabajo relacionado y finalmente, en la Sección 4, se muestran las conclusiones del reporte.

2. Marco teórico

En esta sección se presenta el marco teórico necesario para abordar el problema de la clasificación basada en CARs en flujos de datos. Debido a que la mayoría de los clasificadores desarrollados basados en CARs trabajan con datos estáticos, muchos de los conceptos que se presentan son los utilizados en datos estáticos. Como se observará, estos conceptos son extensibles para trabajar con flujos de datos.

2.1. Conceptos preliminares

Sean $I = \{i_1, i_2, ..., i_n\}$ un conjunto de n ítems y T un conjunto de transacciones, donde cada transacción $t \in T$ está formada por un conjunto de ítems X tal que $X \subseteq I$.

Definición 2.1. *Un conjunto de ítems X es un subconjunto de I cuyo tamaño está dado por su cardinalidad. Un conjunto de ítems de cardinalidad k se denomina k-itemset.*

Es importante aclarar que cuando se haga referencia a un conjunto de ítems X se estará hablando de un subconjunto de I y se supondrá, sin pérdida de generalidad, que existe un orden lexicográfico entre los ítems.

Definición 2.2. *Un flujo de datos es una secuencia ordenada, continua e infinita de transacciones T, que arriban en un instante de tiempo t_i.*

Para procesar y minar los flujos de datos se han propuesto diferentes modelos de ventana.

Definición 2.3. *Un modelo de ventana es un mecanismo que permite establecer límites o fronteras en un flujo de datos para procesar una secuencia finita de transacciones. Existen diferentes modelos de ventana:*

- *modelo Landmark: Los límites (extremos) de la ventana se establecen entre un punto específico (llamado Landmark) y el presente (las últimas transacciones que están arribando). En este modelo todas las transacciones que se procesan tienen el mismo nivel de importancia.*
- *modelo Sliding: Ambos límites de la ventana se desplazan, el tamaño de la ventana puede o no variar. Este tipo de ventana es muy similar a las estructuras FIFO (First In First Out), cuando se inserta una transacción por el extremo final de la ventana, la transacción presente en el extremo inicial se elimina.*
- *modelo Time-Fading: Tiene características similares al modelo Landmark, pero le brinda mayor importancia a las transacciones más recientes de la ventana. Introduce un peso a las transacciones, que decrece a medida que estas se vuelven obsoletas en el tiempo hasta ser eliminadas.*

El contenido de una ventana se define en términos de transacciones (Counted-based), o en intervalos de tiempo (Time -based) donde se procesan las transacciones que arriban en cada intervalo. La ventana modifica su contenido progresivamente ya sea por el arribo de nuevas transacciones

o por el paso del tiempo. La transición puede ser mediante un paso o mediante múltiples pasos. En el primer caso, los límites se desplazan cada una transacción (o bloque de transacciones), o cada un intervalo de tiempo. En el segundo caso los límites se desplazan una cantidad específica de transacciones (o bloques de transacciones), o varios intervalos de tiempo.

Definición 2.4. *La Frecuencia de un conjunto de ítems X respecto a una ventana V, denotada por $frec(X)$, es el número de transacciones en V que contienen a X. El Soporte de un conjunto de ítems X respecto a una ventana V, denotado por $sop(X)$, se define como $frec(X)/|V|$, donde $|V|$ es el total de transacciones de la ventana V.*

Definición 2.5. *X es un conjunto frecuente de ítems (FI) en V, si $sop(X) \geq s$ con $(0 < s \leq 1)$, siendo s el umbral de Soporte definido por el usuario.*

Nótese que las definiciones anteriores de Frecuencia y Soporte de un conjunto de ítems son relativas a una ventana. Por ejemplo, suponiendo que se trabaja sobre un modelo *Landmark* de ventana, definido por intervalos de tiempo, donde en cada unidad de tiempo se recibe un conjunto de transacciones T; un conjunto de ítems X puede ser frecuente en un tiempo t_1 y dejar de serlo en un período de tiempo mayor t_2.

Definición 2.6. *X es un conjunto frecuente-maximal de ítems (MFI por sus siglas en inglés) en V si X es frecuente en V y ninguno de sus superconjuntos propios es frecuente en V. Es decir, no existe un conjunto frecuente de ítems Y en V tal que $X \subset Y$.*

Definición 2.7. *X es un conjunto frecuente-cerrado de ítems (CFI por sus siglas en inglés) en V si X es frecuente en V y ninguno de sus superconjuntos propios posee su misma frecuencia. Es decir, que no existe un conjunto de ítems Y en V tal que $X \subset Y$ y $sop(X) = sop(Y)$.*

Para calcular los FIs/CFIs/MFIs en un flujo de datos es necesario almacenar, además de estos, a los conjuntos infrecuentes de ítems, los cuales son candidatos a frecuentes ante el arribo de nuevas transacciones. Es por ello que muchos de los algoritmos aproximados de cálculo de FIs usan un parámetro de error ϵ, con $0 \leq \epsilon \leq s \leq 1$, para calcular los conjuntos de ítems candidatos a frecuentes.

Definición 2.8. *Se denomina conjunto sub-frecuente de ítems (sub-FI) a aquel cuyo Soporte es mayor que ϵ.*

Debido a las características de los flujos de datos, muchos estudios se han enfocado en la obtención de resultados aproximados, garantizando un límite de error en función del tiempo empleado para procesar las transacciones.

Definición 2.9. *La frecuencia aproximada de un conjunto de ítems X, denotada por $Efrec(X)$, es un valor aproximado de la frecuencia real $frec(X)$.*

Para extender las definiciones anteriores al problema de clasificación basada en CARs, además del conjunto I, se tiene un conjunto de clases C y un conjunto de transacciones etiquetadas T_C (conjunto de entrenamiento). Las transacciones del conjunto T_C están formadas por un conjunto de ítems X y una clase $c \in C$. Esta extensión no afecta las definiciones de Soporte y Frecuencia enunciadas previamente.

Definición 2.10. *Una regla de asociación de clase (CAR por sus siglas en inglés) es una implicación $X \Rightarrow c$ tal que X es un conjunto de ítems y $c \in C$*

Definición 2.11. *El Soporte de una CAR $X \Rightarrow c$ en una ventana V es igual a $sop(X \cup \{c\})$*

Definición 2.12. *La probabilidad de que esté presente c en las transacciones que contienen a X se conoce como la Confianza de la regla $X \Rightarrow c$, y se define en función del Soporte como: $conf(X \Rightarrow c) = \frac{sop(X \bigcup \{c\})}{sop(X)}$. La Confianza toma valores en el intervalo [0,1].*

Los umbrales de Soporte y Confianza utilizados para calcular las CARs se deben determinar con mucho cuidado. Si se utilizan umbrales de Soporte muy pequeños (cercanos a 0) se puede generar un gran volumen de CARs, mientras que si se utilizan umbrales muy altos (cercanos a 1) se pueden dejar de generar muchas CARs interesantes. En [31], se propone utilizar como umbral de Soporte el mínimo valor que evita la ambigüedad al momento de clasificar. Dos CARs se consideran ambiguas si tienen el mismo antecedente implicando diferentes clases.

Definición 2.13. *El tamaño de una CAR $X \Rightarrow c$ está dado por su cardinalidad,* i.e. $|X|+1$*; una CAR de cardinalidad* k *se denomina k-CAR.*

Definición 2.14. *Una CAR $X \Rightarrow c$ satisface o cubre totalmente (de manera exacta) a una transacción t si $X \subseteq t$.*

El problema de la construcción de clasificadores basados en CARs para flujos de datos puede definirse formalmente como:

Sea I un conjunto de ítems, C un conjunto de clases, F_C un flujo de transacciones, donde cada transacción es un par $< X, c >$ tal que $X \subseteq I$ y $c \in C$, R un conjunto ordenado de reglas de la forma $X \Rightarrow c$, W una medida de calidad para evaluar cada regla $r \in R$ y D un criterio de decisión que utiliza a R para asignar una clase a cada transacción t que se desee clasificar. Dados I, C y F_C, construir un clasificador basado en CARs para flujos de datos, consiste en: (1) calcular R de manera incremental, (2) ordenar R según la medida de calidad W y (3) definir el criterio de decisión D.

2.2. Medidas de calidad

Para calcular el conjunto de CARs, se deben considerar dos problemas fundamentales, la cantidad de reglas a obtener y la calidad de las mismas. Para evaluar la calidad de las reglas se han reportado varias medidas de interés. En [28,31,41,48], los autores sugieren tres propiedades que debería cumplir una buena medida de calidad (QM por sus siglas en inglés) para evaluar las reglas de asociación de clase. Estas propiedades son las siguientes:

Propiedad 2.1. Si $Sop(X \Rightarrow Y) = Sop(X)Sop(Y)$ entonces $QM(X \Rightarrow Y) = 0$.

Propiedad 2.2. $QM(X \Rightarrow Y)$ es monótona creciente con respecto a $Sop(X \Rightarrow Y)$ cuando el resto de los parámetros permanece constante.

Propiedad 2.3. $QM(X \Rightarrow Y)$ es monótona decreciente cuando $Sop(X)$ (o $Sop(Y)$) crece y el resto de los parámetros permanece constante.

La Propiedad 2.1 establece que toda buena medida de calidad debe reflejar la independencia estadística [6]. La Propiedad 2.2 establece que entre dos reglas $X \Rightarrow Y$ y $X' \Rightarrow Y'$, tales que $Sop(X) = Sop(X')$ y $Sop(Y) = Sop(Y')$, la mejor regla va a ser aquella cuyo Soporte es mayor. La Propiedad 2.3 evita obtener reglas engañosas porque su valor no aumenta por solo aumentar

el Soporte del consecuente (o del antecedente). La generación de reglas engañosas se pone de manifiesto cuando se procesan conjuntos de datos con muchos ítems propensos a ocurrir con y sin la presencia de otros ítems [31].

El Soporte y la Confianza han sido las medidas más reportadas para evaluar la calidad de las reglas obtenidas a partir de los FIs en el contexto de los flujos de datos [52]. No obstante, varios autores han señalado un conjunto de limitaciones de estas medidas [6,31], por ejemplo: la selección de un umbral de Soporte muy alto puede generar CARs que contengan conocimiento demasiado evidente y a su vez dejar de generar CARs interesantes, por el contrario, la selección de un umbral de Soporte muy bajo puede generar un gran volumen de CARs, las cuales pueden ser redundantes o introducir ruido. Por tanto, el Soporte no es una medida de calidad apropiada para evaluar las CARs y resulta difícil determinar un umbral adecuado. De manera similar al Soporte, la Confianza presenta varias limitaciones: (1) no detecta independencia estadística, (2) no detecta dependencias negativas lo que trae como consecuencia que se generen reglas engañosas, y (3) no considera en su definición al consecuente, en nuestro caso, la clase.

En el contexto de los datos estáticos, se han reportado diferentes medidas de calidad alternativas al Soporte y la Confianza como $Lift$, $Conviction$ y $CertaintyFactor$. En [7,31], los autores presentaron un análisis de estas medidas señalando algunas de sus limitaciones. Las medidas $Lift$ y $Conviction$ tienen la limitación de ser no acotadas, por tanto, las diferencias de los valores de estas medidas no resultan significativas siendo difícil definir un umbral para ellas. Además, la medida $Lift$ es simétrica, lo cual casi no sucede en la práctica. La medida $CertaintyFactor$ es acotada, sin embargo no es una buena medida para evaluar dependencias negativas y además, ante pequeñas variaciones del Soporte alcanza valores muy diferentes.

En [31], se propone la medida de calidad $NetConf$ (ver Ec. 1) como una alternativa que no presenta las limitaciones de las medidas descritas anteriormente. Esta medida permite estimar la fortaleza de una regla y tiene entre sus principales ventajas que detecta las reglas engañosas (o dependencias negativas) ignoradas por la Confianza.

$$NetConf(X \Rightarrow Y) = \frac{Sop(X \Rightarrow Y) - Sop(X)Sop(Y)}{Sop(X)(1 - Sop(X))} \quad . \tag{1}$$

La medida $Netconf(X \Rightarrow Y)$, al igual que $CertaintyFactor$ toma valores en el intervalo [-1,1]. Valores positivos del Netconf representan dependencias positivas, valores negativos representan dependencias negativas (reglas engañosas) y el valor cero representa independencia.

En [31], los autores demuestran que cuando se tienen solo dos clases, cero es el mínimo valor de $Netconf$ que evita la ambigüedad al momento de clasificar, mientras que si se tienen más de dos clases el mínimo valor de $Netconf$ que evita la ambigüedad es 0,5.

En la Tabla 1 se presentan las propiedades que cumplen las medidas de calidad antes mencionadas.

Tabla 1. Propiedades que cumplen las medidas de calidad descritas.

Medida	**Propiedad 1**	**Propiedad 2**	**Propiedad 3**
Soporte	No	Sí	No
Confianza	No	Sí	No
Lift	No	Sí	Sí
Conviction	No	Sí	No
Certainty Factor	Sí	Sí	Sí
Netconf	Sí	Sí	Sí

Como se puede observar, son varias las medidas utilizadas para evaluar la calidad de las reglas en datos estáticos; en cambio, en flujos de datos, prevalece el Soporte y la Confianza.

2.3. Estrategias de ordenamiento

Como se enunció en la sección 2.1, luego de calculado el conjuntos de CARs, las mismas deben ordenarse quedando las de mayor interés en las primeras posiciones del orden. En la literatura existen seis estrategias fundamentales de ordenamiento de CARs para datos estáticos.

1. **Estrategias que combinan un conjunto de criterios para ordenar**. Estas estrategias aplican uno o varios criterios de ordenamiento basados en alguna medida de calidad o en alguna característica de las reglas, por ejemplo, la longitud del antecedente. Se diferencian principalmente en el orden en que se aplican dichos criterios.
   - **CSA (Confianza - Soporte - Longitud del antecedente):** Esta estrategia combina la Confianza, el Soporte y la longitud del antecedente. CSA ordena las CARs en forma descendente de acuerdo con la Confianza, las CARs que tengan valores iguales de Confianza se ordenan en forma descendente de acuerdo con el Soporte, y en caso de empate, se ordenan en forma ascendente de acuerdo con la longitud del antecedente [42,44].
   - **ACS (Longitud del antecedente - Confianza - Soporte):** ACS es una variante de la estrategia CSA, pero considera primero la longitud del antecedente, seguida de la Confianza y el Soporte [16].
   - **SR - QM (*Specific Rules - Quality Measure*):** Es una variante de las estrategias anteriores pero en este caso, se ordenan las CARs de manera descendente por la longitud del antecedente, y en caso de empate, se ordenan de manera descendente por el valor de la medida de calidad que se utilice para evaluar la regla. Esta estrategia es consecuente con la estrategia de poda del espacio de búsqueda usada por los autores, que garantiza la obtención de CARs específicas (grandes) con altos valores de la medida de calidad utilizada [31].
2. **Estrategias que asignan un peso a cada regla**. Las estrategias WRA (del inglés *Weighted Relative Accuracy*) [16,40,58,60], LAP (del inglés *LAPlace expected error estimate*) [15,58] y $\chi^2(Chi-Cuadrado)$ [42] asignan un peso a cada CAR, calculado en función del Soporte y la Confianza, y después ordenan el conjunto de CARs en forma descendente de acuerdo al valor de los pesos asignados.

Para todas las estrategias, en caso de persistir el empate entre dos o más reglas, prevalece el orden en que fueron generadas. Los escasos clasificadores desarrollados (basados en CARs) para flujos de datos, utilizan las mismas estrategias de ordenamiento que los clasificadores desarrollados para datos estáticos. En [52], se ordenan las reglas descendentemente de acuerdo con el valor de confianza, y en [27] se ordena de acuerdo con la ganancia de información, mientras que en [22,24], las reglas se mantienen en el mismo orden que son generadas.

2.4. Criterios de decisión

En la mayoría de los trabajos reportados, luego de construido el clasificador, para clasificar una nueva transacción t, se determina el subconjunto de CARs que cubren a t de manera exacta (ver Def.2.14) y se utiliza un criterio de decisión para asignar una clase a t. En la literatura se reportan cuatro criterios de decisión para el trabajo con datos estáticos:

1. La Mejor Regla: Se selecciona la primera regla en el orden establecido (la mejor regla) que cubra a t y se asigna a t la clase de la regla seleccionada [44].
2. Las Mejores K Reglas: Se seleccionan, por cada clase, las primeras K reglas en el orden establecido que cubran a t, se promedian los valores de calidad de las reglas en cada clase y se asigna a t la clase para la que se obtenga mayor promedio [58].
3. Todas las Reglas: Se seleccionan, para cada clase, todas las reglas que cubran a t, se promedian los valores de calidad de las reglas en cada clase y se asigna a t la clase para la que se obtenga mayor promedio [42].
4. Dynamic K: Este criterio es una variación del criterio "Las mejores K reglas", en este criterio se seleccionan, por cada clase, las CARs maximales que cubren a t y se asigna la clase c tal que todas las CARs seleccionadas con consecuente c, que cubran a la nueva transacción t (supóngase que son K), tengan mayor promedio de los valores de calidad que las primeras K CARs seleccionadas de las clases restantes, en caso de empate se asigna la clase de menos CARs [31].

En el caso de la clasificación en flujos de datos se ha reportado el uso de estos criterios combinados con técnicas de clasificación, como el vecino más cercano [22] y las redes bayesianas [24].

En [22], los autores utilizan el criterio de "La Mejor Regla", si esta es una regla consistente (solo cubrió transacciones de la clase positiva en su construcción), entonces se asigna la clase que contiene la regla; en caso de ser una regla inconsistente (cubrió transacciones de clase negativa y positiva), se clasifica mediante el vecino más cercano.

En [24], los autores aplican dos criterios para clasificar una nueva transacción, "La mejor Regla" y "Todas las reglas". En ambos casos aplican el Teorema de Bayes para asignar a la transacción la clase que maximice la probabilidad a posteriori dada por la regla de Bayes.

Varios autores han reportado algunas limitaciones de los criterios de decisión antes mencionados , por ejemplo: Los clasificadores que siguen el criterio "La Mejor Regla" apuestan por una sola regla para clasificar, y como se menciona en [16,31], no se puede esperar que una sola regla prediga exactamente la clase de cada transacción que ésta cubra. El criterio "Las Mejores K Reglas" puede afectar la eficacia del clasificador cuando 1) existe desbalance en el número de CARs con altos valores de la medida de calidad, por clase, que cubren a la nueva transacción; o cuando 2) la mayoría de las mejores K reglas se obtuvieron a partir del mismo ítem, dando lugar a cierta redundancia [31]. Por último, al utilizar el criterio "Todas las Reglas" pueden incluirse reglas con bajos valores de la medida de la calidad para clasificar [62].

3. Generación de CARs

Al igual que el minado de conjuntos frecuentes de ítems sobre flujos de datos [10,18,29,45], el cálculo de todas las CARs es una tarea costosa debido a su complejidad exponencial [3,31]. El costo de calcular las CARs es igual al costo de calcular los FIs (las clases se consideran como un ítem más) pues las CARs se obtienen a partir de los FIs de forma inmediata, por ejemplo, el conjunto $\{i_1, i_2, c_1\}$ da lugar a la CAR $\{i_1, i_2 \Rightarrow c_1\}$ (en lo adelante en este epígrafe utilizaremos indistintamente cálculo de FIs y cálculo de CARs). Por lo general, el cálculo de las CARs se corresponde con un recorrido por el espacio de búsqueda formado por las CARs. Supóngase un conjunto de tres ítems $I = \{i_1, i_2, i_3\}$ y dos clases $C = \{c_1, c_2\}$. En la Figura 1 se observa el

retículo (espacio de búsqueda) de las CARs que se pueden formar con los ítems del conjunto I y las clases del conjunto C.

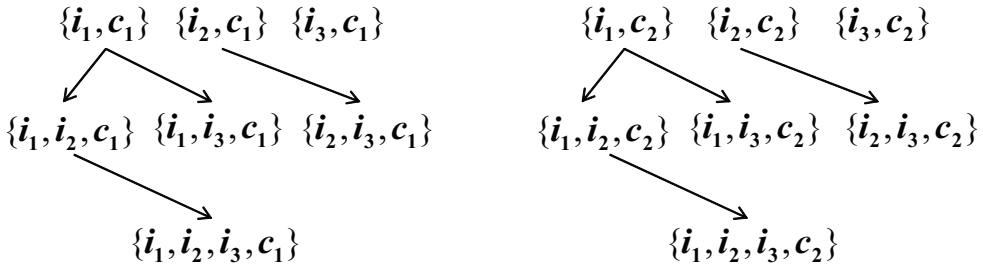

Fig. 1. Espacio de búsqueda de las CARs que se forman con tres ítems y dos clases.

El primer nivel del retículo está compuesto por las CARs de tamaño dos; noten que en este nivel, con el objetivo de ser más intuitivos, se repitieron los ítems de tamaño uno y se intercalaron las clases entre ellos. También buscando simplicidad se omitió el conjunto vacío y se representó cada CAR como un conjunto con los ítems que forman el antecedente al principio y la clase como último elemento, *e.g.* el conjunto $\{i_1, i_2, c_1\}$ representa a la CAR $\{i_1, i_2 \Rightarrow c_1\}$.

Si se tienen n ítems y m clases, el espacio de búsqueda de las CARs es de $m*2^n$ elementos. Para podar el espacio de búsqueda y evitar el cálculo del Soporte de las CARs que no son frecuentes, los algoritmos sobre datos estáticos aplican la propiedad de clausura descendente del Soporte [3], que plantea que todo subconjunto de un FI es frecuente, o lo que es igual, todo superconjunto de un conjunto no frecuente es no frecuente.

Al trabajar con flujos de datos se hace muy difícil el proceso de minado, ya que las CARs pueden variar a medida que arriben nuevas transacciones. Dado que los flujos de datos son en teoría infinitos, resulta imposible almacenar todas las transacciones y se requiere de un algoritmo incremental que actualice las CARs frecuentes con la información de las nuevas transacciones.

En general, los algoritmos que trabajan sobre flujos de datos, en un primer paso, procesan el flujo de transacciones siguiendo un modelo de ventana (ver Def. 2.3), obtienen los FIs (recuerden que las CARs se obtienen de forma inmediata a partir de los FIs) y los almacenan en una estructura arbórea. A medida que arriban nuevas transacciones a la ventana, el algoritmo recorre la estructura arbórea actualizando los FIs con los nuevos datos. Las operaciones de actualización sobre la estructura que almacena los FIs pueden ser por inserción, modificación o eliminación.

Algorithm 1: Pasos generales del proceso de cálculo de FIs en flujos de datos

```
Input: Flujo de transacciones F
Output: FIs
FIs ⟵ ∅;
Inicializar la ventana W con transacciones de F;
W' ⟵ ∅;
while true do
    if W ≠ W' then
        if FIs ≠ ∅ then
            FIs ⟵ Actualizar FIs con W
        else
            Calcular FIs con W
        end
    end
    if petición de respuesta then
        return FIs
    end
    W' ← W;
    Actualizar W con transacciones de F;
end
```

Los algoritmos de cálculo de CARs (o FIs) definen diferentes estrategias para recorrer el espacio de búsqueda. Estas estrategias pueden clasificarse atendiendo a la dirección del recorrido [31]:

- Descendentes: Si el recorrido se realiza desde el segundo nivel (CARs de tamaño 2) hacia el último nivel, deteniéndose en el nivel donde no se genere ninguna CAR.
- Ascendentes: Si el recorrido se realiza en el sentido opuesto, desde un nivel aproximado a los últimos niveles donde se generen CARs hacia el primer nivel.

Al mismo tiempo, dentro de estas estrategias se pueden generar las CARs de dos formas:

1. En amplitud (*Breadth-First Search*): Se generan todas las CARs de tamaño k antes de generar las CARs de tamaño $k+1$. Esta estrategia necesita realizar múltiples recorridos por el conjunto de datos para calcular el Soporte de los conjuntos de ítems candidatos. Un clasificador que sigue esta estrategia es el AC-DS [52].
2. En profundidad (*Depth-First Search*): Se generan las CARs por cada rama del espacio de búsqueda. Es decir, por cada clase, se toma como antecedente el primer ítem y se extiende, tanto como sea posible, con los ítems lexicográficamente mayores que él, generando todos los superconjuntos frecuentes y retrocediendo (*backtracking*) cuando no se pueda extender más. Después de terminar con un ítem se toma el siguiente y se realiza el mismo proceso. Algunos de los algoritmos para el cálculo de CARs sobre flujos de datos que siguen esta estrategia son Moment [14] y StreamGen [27].

La mayoría de los algoritmos para el cálculo de FIs sobre datos estáticos se basan en el algoritmo Apriori [2], el cual sigue una estrategia de recorrido en amplitud realizando múltiples recorridos por el conjunto de datos, una alternativa no adecuada cuando se trabaja con flujos de datos ya que en cada momento solo se tiene acceso a una parte del flujo. Como alternativa, algoritmos como Eclat [64] y FP-Growth [30] reducen a solo dos los recorridos del conjunto de datos, generando los conjuntos de candidatos siguiendo un recorrido en profundidad. Por lo

general, la mayoría de los algoritmos de minería de FIs para flujos de datos se basan en el algoritmo FP-Growth.

Un problema al que se enfrentan los algoritmos de minería de FIs para flujos de datos es el tratamiento de los conjuntos de ítems que no son frecuentes pero pueden convertirse en frecuentes en algún momento. Si la información de estos conjuntos de ítems no se almacena es imposible calcular su Soporte real en el momento en que se vuelven frecuentes.

Por ello, algunos algoritmos [29,33,45] clasifican los conjuntos de ítems en: frecuentes, sub-frecuentes e infrecuentes. Los segundos, son los candidatos potenciales a convertirse en frecuentes. Estos algoritmos mantienen de manera eficiente los FIs y sub-FIs en una estructura arbórea similar a un FP-Tree [30], la cual se actualiza periódicamente cada vez que llegan nuevas transacciones.

Otro problema que sufren los algoritmos de cálculo de FIs es la explosión combinatoria. Dos soluciones han sido propuestas para aliviar este problema, la primera se enfoca en calcular solo los conjuntos maximales (ver Def.2.6), cuyo número por lo general es mucho menor que los FIs; sin embargo, a partir de los MFIs no se puede obtener el Soporte de los restantes FIs, lo cual puede ser necesario [14]. La segunda solución, consiste en calcular los cerrados (ver Def.2.7), que son menos que los FIs, y además, contienen la información necesaria para calcular los Soportes de los restantes FIs.

Para la construcción de los algoritmos de cálculo de FIs/CFIs/MFIs se pueden identificar tres enfoques basados en el modelo de ventana utilizado: (1) enfoques que trabajan sobre todo el flujo "modelo *LandMark*", (2) enfoques que asignan mayor importancia a las transacciones más recientes "modelo *Time-Fading*", y (3) enfoques que trabajan sobre una parte del flujo "modelo *Sliding Windows*". A su vez, los algoritmos se pueden clasificar según su naturaleza en exactos o aproximados, y dentro de los aproximados, pueden clasificarse según el tipo de resultados en: orientados a falsos positivos u orientado a falsos negativos.

Los algoritmos aproximados orientados a falsos positivos, incluyen en el conjunto final algunos sub-FIs (ver Def. 9); es decir, a partir de un umbral de Soporte s y un parámetro de error ϵ que controla el consumo de memoria, proporcionan como salida los conjuntos de ítems (falsos positivos) con Soporte entre $s - \epsilon$ y s, los cuales se consideran frecuentes.

Los algoritmos aproximados orientados a falsos negativos pierden algunos FIs en el conjunto final. Estos algoritmos, mediante los parámetros s y ϵ, usan $s + \epsilon$ como umbral de Soporte y calculan FIs con Soporte mayor que s, dejando de considerar como frecuentes los conjuntos de ítems (falsos negativos) con Soporte entre s y $s + \epsilon$.

A continuación se describen los algoritmos más representativos de la literatura de acuerdo a los tres enfoques anteriores.

3.1. Algoritmos que procesan todo el flujo

En esta subsección se describen cuatro algoritmos aproximados que usan un modelo de ventana *LandMark*. En este enfoque el cálculo de los FIs se realiza teniendo en cuenta todas las transacciones que arriban entre un punto específico y el presente. Dentro de estos algoritmos, tres son orientados a falsos positivos y uno a falsos negativos.

3.1.1. Lossy-Counting

En [45], Manku y Motwani proponen Lossy-Counting, un algoritmo aproximado de cálculo de FIs en flujos de datos, orientado a falsos positivos. Lossy-Counting recibe dos parámetros de

entrada: un umbral de Soporte $s \in (0, 1]$ y un parámetro de error $\epsilon \in (0, 1)$. Lossy-Counting procesa bloques de transacciones de tamaño $w = \lceil \frac{1}{\epsilon} \rceil$ y calcula los FIs aplicando una estrategia de recorrido en profundidad. Como salida este algoritmo garantiza que:

- Se calculen todos los conjuntos de ítems cuya frecuencia real exceda sN, donde N es el total de transacciones recibidas. No existen falsos negativos.
- Se calculen además los conjuntos de ítems cuya frecuencia real sea mayor que $(s-\epsilon)N$ (Falsos positivos).
- Las frecuencias calculadas se diferencien de las frecuencias reales a lo sumo en ϵN.

Lossy-Counting necesita almacenar aquellos conjuntos de ítems (sub-FIs) con Soporte no menor que ϵ para garantizar que las frecuencias calculadas se diferencien de las frecuencias reales a lo sumo en ϵN. Como el valor de ϵ es por lo general mucho más pequeño que el umbral de Soporte, la aproximación de los resultados es mucho más exacta. Sin embargo, la cantidad de sub-FIs que se tiene que almacenar es muy grande implicando un alto consumo de memoria. Este problema está presente en la mayoría de los algoritmos que utilizan un parámetro de error para controlar la aproximación del resultado.

3.1.2. DSM-FI

De forma similar al algoritmo Lossy-Counting, DSM-FI (*Data Stream Mining for Frequent Itemsets*) [33] lee un segmento de transacciones hacia la memoria en cada unidad de tiempo y las ordena lexicográficamente. Seguidamente, DSM-FI construye y mantiene en memoria un árbol de prefijos que implementa una estructura llamada SFI-Forest (*Summary Frequent Item set Forest*).

La estructura SFI-Forest almacena las transacciones procesadas de manera compacta mediante un bosque de árboles, donde cada árbol guarda la información de las transacciones de igual prefijo. Cada camino de un árbol desde la raíz hasta un nodo hoja almacena un sub-FI. Para calcular la frecuencia de un sub-FI se necesita recorrer todo el árbol. DSM-FI propone un mecanismo eficiente para el cálculo de los FIs sobre el SFI-Forest, llamado ToDoFIS (*Top Down Frequent Item set Search*), que inicia con la búsqueda de los conjuntos frecuentes maximales presentes en la estructura de datos y genera los FIs a partir de estos.

En [33], los autores mostraron que *DSM-FI* resulta muy eficiente para datos densos o dispersos. Por otro lado, aunque la estructura SFI-Forest es más compacta que un árbol de prefijos, tiene un mayor costo computacional ya que se requieren más recorridos sobre el árbol para obtener la frecuencia de los conjuntos de ítems.

3.1.3. hMine

El algoritmo hMine [56] es orientado a falsos positivos y almacena las transacciones procesadas en una tabla hash de tamaño fijo. En este algoritmo, una transacción se representa como un vector binario, cuyo número expresado en el sistema decimal constituye una entrada en la tabla hash.

Además de las transacciones, cada entrada de la tabla hash almacena una lista de nodos (llamados nodos frecuentes) que poseen la información (frecuencia real y estimada) de los FIs asociados a dicha entrada.

Cuando arriba una nueva transacción, se actualizan en la tabla hash los soportes de todos los subconjuntos de ítems que conforman a la transacción. Para ello, hMine verifica si cada subconjunto de ítems tiene un nodo frecuente en su entrada de la tabla hash, en ese caso se incrementa en 1 su contador de frecuencia real. En caso contrario se verifica si es un conjunto

candidato mediante la propiedad de clausura descendente del Soporte. Luego, el algoritmo poda los nodos no frecuentes de los conjuntos de ítems que fueron actualizados en la tabla hash.

Al igual que ocurre con el algoritmo Lossy-Counting, hMine se ve afectado por el parámetro de error; mientras menor es el error mayor es la cantidad de sub-FIs que deben almacenarse en la tabla hash, lo que implica un mayor consumo de memoria y un mayor tiempo de actualización. Sin embargo, debido al uso de un mecanismo (no se describe el mismo en [56]) que permite estimar el Soporte de los conjuntos infrecuentes de ítems, sin necesidad de almacenarlos en la tabla hash, se logra un gran ahorro de memoria.

3.1.4. FDPM

En [63], los autores proponen el algoritmo FDPM (*Frecuent Data Stream Pattern Mining*), que calcula un conjunto aproximado de FIs considerando el Soporte de un conjunto de ítems X en las primeras n transacciones. Para calcular el valor de n, FDPM se basa en el límite de *Chernoff* [13].

El algoritmo recibe como parámetros un umbral de Soporte y una confiabilidad δ. FDPM establece que el Soporte de un conjunto de ítems X en las primeras n transacciones del flujo de datos se encuentra acotado por un error ϵ respecto al Soporte de X en todo el flujo de datos, con una probabilidad de al menos $(1 - \delta)$. El cálculo de los FIs se realiza aplicando a cada segmento de transacciones procesadas, un algoritmo de cálculo de FIs para datos estáticos.

El algoritmo FDPM garantiza que:

- La frecuencia real de los FIs es mayor que $s * N$ con una probabilidad de $1 - \delta$.
- Ningún FI con frecuencia real menor que $s * N$ pertenece al conjunto final (orientado a falsos negativos).
- Las frecuencias calculadas de los FIs son cercanas a sus frecuencias reales con una probabilidad no menor de $(1 - \delta)$.

A diferencia del algoritmo Lossy-Counting, en este algoritmo el consumo de memoria se mantiene constante y no depende de un parámetro de error. Sin embargo, como FDPM es orientado a falsos negativos, puede ganar en rendimiento y perder en exactitud ya que se podan FIs que afectan la calidad del resultado.

3.2. Algoritmos que asignan mayor importancia a las transacciones más recientes

En esta subsección serán descritos dos algoritmos aproximados, de minado de FIs, orientados a falsos positivos, que usan un modelo *Time-Fading* (ver Def. 2.3) que favorece a las transacciones más recientes, asignándole un peso que decrece a medida que éstas se vuelven obsoletas con el paso del tiempo.

3.2.1. FP-Stream

El algoritmo FP-Stream [29], orientado a falsos positivos, permite descubrir FIs a diferentes granularidades de tiempos. Este algoritmo utiliza un modelo de ventana especial llamado *tilted-time window* basado en unidades de tiempo, en el que la frecuencia de un conjunto de ítems se almacena en diferentes granularidades de tiempo, donde las más pequeñas se corresponden con las unidades de tiempo más recientes (por ejemplo, los últimos n minutos) y las más grandes con unidades de tiempo más lejanas (por ejemplo, últimos n meses).

El algoritmo trabaja con una estructura de almacenamiento llamada FP-Stream que tiene dos componentes: un árbol de prefijos y una tabla compuesta por varias ventanas de tiempo a diferentes escalas (llamada *tilted-time windows*). Cada nodo hoja del árbol almacena la tabla que mantiene la frecuencia del conjunto de ítems en diferentes escalas de tiempo.

FP-Stream procesa cada bloque de transacciones mediante el algoritmo FP-growth para ir obteniendo los FIs y sub-FIs (ver Def. 2.8). Los FIs y sub-FIs obtenidos se almacenan en el árbol de prefijos y la frecuencia de cada conjunto de ítems se distribuye en la tabla de ventanas de tiempo. Luego se elimina el número de registros de la tabla, a partir de las ventanas con escalas de tiempo más viejas, y se podan aquellas cuya frecuencia acumulada sea menor que $s * N$, donde s es el umbral de Soporte y N la cantidad de transacciones procesadas hasta el momento. Si la tabla de un conjunto de ítems X queda vacía, producto de la poda, FP-growth deja de considerar los super-conjuntos de X con base en la propiedad de clausura descendente del Soporte y X se elimina del árbol de prefijos.

El uso de múltiples ventanas de tiempo a diferentes escalas es una buena opción para aplicaciones en las que se requiera obtener un historial completo del comportamiento de los conjuntos frecuentes de ítems, con niveles de detalles expresados en función del tiempo. Sin embargo, el consumo de memoria es excesivo ya que para cada conjunto de ítems se requiere almacenar la misma información varias veces, por lo que la estructura FP-Stream puede volverse muy compleja, lo que trae consigo un alto costo computacional en las operaciones de actualización y poda.

3.2.2. estDec

Al igual que los algoritmos anteriores, estDec [10] es un algoritmo aproximado, orientado a falsos positivos. El algoritmo estDec examina cada transacción del flujo de datos sin generar candidatos y mantiene un registro del conteo de frecuencia de cada conjunto (mediante un árbol de prefijos) en las transacciones analizadas hasta ese momento.

El algoritmo estDec emplea una estructura especial para almacenar los conjuntos frecuentes potenciales y su conteo de frecuencia. Adicionalmente, esDec solo necesita actualizar el conteo de frecuencia para aquellos conjuntos que son sub-conjunto de las transacciones más recientes.

En estDec, el uso de una tasa de decremento disminuye el efecto de las transacciones más antiguas en el resultado global del proceso de minado. Sin embargo, estimar la frecuencia de un conjunto a partir de la frecuencia de los subconjuntos que lo forman puede propagar un error desde los conjuntos de tamaño 2 hasta los super-conjuntos de tamaño n. Por lo tanto, es difícil definir un límite para el error en la frecuencia calculada del conjunto resultante, y se obtendrán muchos falsos positivos. Además, la actualización de la estructura creada luego de la llegada de cada transacción puede no ser adecuada en flujos a altas velocidades.

3.3. Algoritmos que procesan una parte del flujo

En esta subsección se describen seis algoritmos que usan un modelo *Sliding* de ventana para minar FIs/CFIs/MFIs. En este enfoque resultan más interesantes las transacciones recientes que las más antiguas. Por cada desplazamiento de la ventana se agregan nuevas transacciones y a su vez se elimina la misma cantidad de transacciones agregadas. Dentro de los algoritmos a analizar, cinco son exactos, y uno es aproximado y orientado a falsos negativos. Por otra parte, dos de estos algoritmos son para minar FIs, otros tres para minar CFIs y uno para calcular MFIs.

3.3.1. Moment

En [14], Chi *et al.* proponen un algoritmo denominado Moment, que calcula los CFIs usando una ventana deslizante, que mantiene las transacciones más recientes del flujo de datos. Las transacciones de la ventana son almacenadas en un FP-Tree, para acelerar su procesamiento. Los CFIs y sub-FIs son calculados a partir de las transacciones almacenadas en el FP-Tree, mediante un recorrido "Primero en profundidad". Los CFIs y sub-FIs son almacenadas empleando una estructura llamada *Closed-Enumeration-Trees* (CTE por sus siglas en inglés), similar a un árbol de prefijos, donde cada nodo representa un conjunto de ítems.

Este método distingue los CFIs y sub-CFIs en 4 tipos de nodos, llamados: "*Infrequent Gateway Node*", que son nodos infrecuentes cuyos padres y hermanos son frecuentes; "*Unpromising Gateway Node*" que son nodos infrecuentes que tienen un superconjunto con su mismo Soporte; "*Intermediate Node*" que son nodos frecuentes que tienen un superconjunto con su mismo Soporte y "*Closed Node*" que son aquellos nodos que no tienen hijos con igual o mayor Soporte que él.

Cada vez que se procesa una transacción, Moment clasifica los sub-FIs presentes en la transacción en uno de los 4 tipos de nodos mencionados anteriormente. Cada vez que un usuario requiere obtener los FCIs, hay que recorrer la estructura CET (la cual solo contiene los nodos cercanos y los nodos límites) para obtener los n conjuntos frecuentes cercanos asumiendo que todos los cambios de interés han ocurrido en los límites de los conjuntos frecuentes cercanos y el resto de los conjuntos se encuentran fuera la mayoría del tiempo.

A diferencia de los árboles de prefijos, la estructura CET que *Moment* propone solo mantiene los CFIs y aquellos nodos que marcan el límite entre los CFIs y el resto de los conjuntos, esto reduce el número de nodos comparado con los árboles de prefijos. No obstante, las operaciones sobre la estructura CET son costosas ya que recorren el árbol siguiendo una estrategia "Primero en profundidad", y por lo tanto, los costos en almacenamiento y tiempo son elevados.

3.3.2. CFI-Stream

En [35], los autores proponen el algoritmo CFI-Stream para el cálculo de CFIs. Este algoritmo chequea cada conjunto de ítems de manera *online* sin usar ninguna estructura de datos auxiliar, y actualiza el soporte de los conjuntos de ítems cerrados, los que almacena en una estructura arbórea similar a un árbol de prefijos llamada DIrect Update tree (DIU). Cada nodo en el DIU almacena un FCI y su conteo de frecuencia. Cuando llega una nueva transacción, *CFI-Stream* genera todos los subconjuntos de ítems de dicha transacción, y chequea si éstos son cerrados o no mediante un mecanismo basado en el uso del operador de clausura (operador de Galois). En caso de ser cerrados y estar en el DIU, actualiza sus contadores de frecuencia, así como el de sus subconjuntos de ítems presentes en el DIU. De la misma manera ocurre cuando se desliza la ventana y se elimina una transacción, en este caso se chequea si el conjunto de ítems de la transacción y sus subconjuntos dejan de ser cerrados, o si es necesario decrementar en uno el valor de sus contadores de frecuencia.

Con el mecanismo de chequeo mediante el uso del operador de clausura, solo se mantienen los conjuntos cerrados en la estructura DIU, lo que reduce el número de nodos almacenados en memoria comparado con la estructura CET del algoritmo *Moment*. Por otra parte, al igual que el algoritmo *Moment*, *CFI-Stream* realiza operaciones de actualización cada vez que se desliza la ventana. Esa forma de procesamiento es poco eficiente cuando se trabaja con flujos de datos con altas tasas de llegadas. Sin embargo estos métodos devuelven de manera exacta los FCIs presentes en el momento actual, lo cual es una ventaja para aplicaciones que requieran de este tipo de respuestas.

3.3.3. IncMine

En [12], Cheng *et al.* proponen el algoritmo IncMine para obtener un conjunto aproximado de CFIs, usando una ventana deslizante compuesta por unidades de tiempo de tamaño fijo (ver Def. 2.3), donde retienen un nuevo tipo de conjunto de ítems, denominados semi-CFIs (*semi - Closed Frequent Itemsets*). Los semi-CFIs son FIs que tienen superconjuntos con su misma frecuencia, en alguna unidad de tiempo de la ventana. A partir de las propiedades de los semi-CFIs, los autores determinan que el estado de un conjunto de ítems puede ser conocido, a partir de la relación entre los semi-CFIs sobre dos deslizamientos consecutivos de la ventana de tiempo (o dos ventanas de tiempo consecutivas). Esto indica que solo una pequeña parte de los sub-conjuntos de un semi-CFI en la ventana actual puede convertirse en semi-CFI en el próximo deslizamiento y como consecuencia, no es necesario actualizar los CFIs.

Con base en lo anterior, los semi-CFIs se actualizan mediante la poda de una gran cantidad de subconjuntos de los semi-CFIs que no tienen posibilidades de convertirse en semi-CFIs. Los semi-CFIs se almacenan en una estructura de datos llamada *Inverted Index Structure*, basada en arreglos, que posee dos componentes. El primer componente particiona la primera ventana de acuerdo al tamaño de los semi-CFIs presentes en ella, cada semi-CFIs de tamaño n pertenece a la misma partición. Cada partición se almacena en un CFI-array que guarda los semi-CFIs de la misma partición, con un identificador asociado y su conteo de frecuencia, este último se calcula de manera aproximada en cada unidad de tiempo de la ventana. El segundo componente es una estructura llamada *Inverted CFI Index* que se construye a partir de los CFI-arrays, y se utiliza para comprobar si un FI es semi-CFI o no. Además mediante esta estructura, se garantiza un recorrido eficiente por el espacio de búsqueda de los súper-conjuntos de los semi-FCIs analizados.

En este algoritmo, el umbral de Soporte es relativo a la ventana, y se calcula en cada unidad de tiempo para los conjuntos de ítems que se retienen más tiempo en la ventana, de manera que el Soporte de un semi-CFI expira a lo largo del tiempo, con lo que se logra reducir el número de elementos que deben mantenerse y procesar. Además, mediante el *Inverted Index Structure* se logra una mejor eficiencia sobre las operaciones de actualización de los semi-CFIs y un bajo consumo de memoria, comparado con el uso de un árbol de prefijos. No obstante, las operaciones de inserción y eliminación siguen siendo costosas a pesar de que ocurran con menor frecuencia. Por otra parte, incMine pierde en exactitud por ser orientado a falsos positivos, aunque es válido resaltar que la cantidad de falsos positivos que genera es pequeña.

3.3.4. StreamGen

El algoritmo StreamGen [27] calcula unos FIs, denominados FI-generadores (FGIs). Esta nueva concepción de FIs se basa en una forma muy interesante de definir los CFIs. Si se particiona el espacio de búsqueda en clases de equivalencia, agrupando en cada clase los FIs de igual Soporte, los conjuntos maximales (los de mayor longitud) de cada clase de equivalencia son los CFIs. De forma análoga se pueden seleccionar los conjuntos minimales (los de menor longitud) de cada clase de equivalencia, y a esos se les llaman conjuntos generadores de esa clase de equivalencia, lo que equivale a ser generadores del espacio de búsqueda.

Un FGI es un FI que no posee un sub-conjunto de él con su mismo soporte. Según el principio MDL (*Minimum Description Length Principle*) [32], los FGIs son muy útiles con respecto a los CFIs para problemas de selección de rasgos y para construir clasificadores basados en reglas de asociación de clase.

El algoritmo StreamGen utiliza un árbol de prefijos para almacenar las transacciones que deben ser procesadas. Al igual que el algoritmo Moment, usa una estructura parecida al CET

denominada "*Enumeration Tree* (ET)", para mantener los FGIs y los conjuntos de ítems que tienen posibilidades de convertirse en generadores. Para ello, los autores eliminan el "*Gateway Node*" del CET y solo distinguen tres tipos de nodos en el árbol: (1) "*Infrequent Node*", son nodos que representan conjuntos de ítems infrecuentes, (2) "*Unpromising Node*", aquellos nodos que representan FIs no generadores, y (3) "*Generator Node*", son los nodos que representan a un FGI. Para actualizar el ET se realizan operaciones de actualización, inserción y eiminación cada vez que se desliza la ventana. Para hacer menos costoso este proceso solo se podan los "*Infrequent Node*" o "*Unpromising Node*" que posean nodos hermanos del padre, clasificados como "Infrequent Node" o "Unpromising Node".

La estructura ET el número de nodos con respecto a estructura CET, propuesta por Moment. No obstante, las operaciones sobre la estructura ET son costosas en cuanto a tiempo, ya que por cada transacción que arriba o expira, se recorre el árbol mediante la estrategia "Primero en profundidad" para actualizar los nodos asociados.

3.3.5. MFI-Trans-SW

MFI-Trans-SW [32] es un algoritmo exacto, que calcula los FIs utilizando un modelo *Sliding* de ventana. Este algoritmo almacena las transacciones mediante secuencias de bits siguiendo un proceso denominado *bit-sequence transform*. MFI-Trans-SW calcula todos los FIs siguienda una estrategia "en amplitud" y consta de tres fases:

1. Fase de inicialización de la ventana: En esta fase se llena la ventana con una cantidad de transacciones definida por el usuario. Mediante un proceso llamado *bit-sequence transform*, cada item X presente en las transacciones de la ventana se transforma en una secuencia de bits (denominada $Bit(X)$). Si un ítem X se encuentra en la transacción *i-ésima* de la ventana, el bit *i-ésimo* de la estructura $Bit(X)$ se hace 1, de lo contrario se hace 0.
2. Fase de desplazamiento de la ventana: En esta fase se agregan y eliminan transacciones a la ventana, y la información de los ítems presentes en las transacciones eliminadas se actualiza mediante un desplazamiento de bits hacia la izquierda en todos los ítems afectados.
3. Fase de generación de FIs: En esta fase se realiza el proceso de generación de candidatos usando la propiedad de clausura descendente del Soporte. En este proceso se calcula el Soporte de los conjuntos de ítems de tamaño k, generados a partir del Soporte de los conjuntos de ítems de tamaño $k-1$.

A pesar de ser un algoritmo exacto, el procesamiento de las transacciones transformadas en secuencias de bits, hace que MFI-Trans-SW se ejecute mucho más rápido y consuma menos memoria que los demás algoritmos de su tipo. Sin embargo, presenta el problema de una alta generación de candidatos al igual que los restantes algoritmos que siguen una estrategia de recorrido en "en amplitud".

3.3.6. Max-FISM

El algoritmo *Max-FISM* [21] calcula los MFIs utilizando un modelo de ventana *Sliding*. Por cada transacción recibida en la ventana, el algoritmo ordena sus ítems con el objetivo de realizar un procesamiento más eficiente, y construye una estructura con las mismas características de un árbol de prefijos llamada Max-Set, con la particularidad de que solo se almacenan los conjuntos de ítems denominados Max-Itemsets. Se llama Max-Itemset al subconjunto de ítems de mayor longitud de una transacción previamente ordenada. Cada nodo del Max-Set almacena el Max-

Itemset de la transacción y su frecuencia en la ventana de manera independiente, es decir, sin tener en cuenta el conteo de frecuencia de sus subconjuntos o superconjuntos.

Para determinar los MFIs luego de construído el Max-Set, se realiza un recorrido por el árbol de manera ascendente y de izquierda a derecha. Para el proceso, se usan dos listas auxiliares llamadas MFI-List y MIFI-List donde se almacenan los FIs y los conjuntos infrecuentes de ítems encontrados durante el recorrido. Estas listas se actualizan a medida que se analiza cada uno de los nodos del árbol. Por cada nodo analizado, si este no pertenece a ninguna de las dos listas auxiliares, se calcula su conteo de frecuencia real, acumulando a su contador de frecuencia independiente los valores de frecuencia de sus superconjuntos presentes en el Max-Set. Los FIs obtenidos se insertan en la MFI-List y se eliminan de la misma todos los subconjuntos de los FIs insertados. Con ello se garantiza que en la MFI-List se encuentren solamente los MFIs. Si el conjunto de ítems analizado es infrecuente, se inserta en la MIFI-List y se eliminan todos los conjuntos de ítems que lo contienen a él en la lista, luego se analizan sus subconjuntos siguiendo el mismo proceso de manera recursiva.

Debido a que la cantidad de nodos puede aumentar considerablemente a medida que arriben nuevas transacciones a la ventana, el algoritmo aplica una poda cada cierto período de tiempo, usando una medida llamada *Maximum Validity Time* (MVT) que indica cuánto tiempo un conjunto de ítems puede permanecer siendo frecuente, asumiendo que éste no esté presente en las futuras transacciones.

El algoritmo es muy eficiente en cuanto a tiempo de procesamiento y uso de memoria, al procesar cada transacción solo tiene en cuenta el conjunto Max-Itemset y no es necesario monitorear sus subconjuntos de ítems. A pesar de ser un algoritmo exacto, al tener en cuenta los MFIs, estos no contienen la información del soporte de cada FI, a no ser que el conjunto de ítems sea también maximal, lo que puede conllevar a perder ítems que pueden ser importantes.

3.4. Síntesis y conclusiones

En la sección anterior se analizaron doce algoritmos que permiten calcular FIs/CFIs/MFIs en flujos de datos. De estos se describieron sus características fundamentales, sus ventajas y desventajas. En la Tabla 3.4 se muestra un resumen comparativo de los algoritmos de acuerdo a sus características.

Como se observa en la Tabla 3.4, seis de los algoritmos utilizan un modelo de ventana *Landmark* y seis un modelo *Sliding* para procesar las transacciones. De los seis algoritmos que emplean un modelo *Landmark*, los cuatro primeros (*Lossy Counting*, *FDPM*, *DSM-FI* y *hMine*) no realizan distinción entre la transacciones, mientras que el algoritmo *estDec*, que usa un modelo *Time-Fading* con características similares al modelo *Landmark*, favorece a las transacciones más recientes disminuyendo gradualmente el efecto de las transacciones obsoletas. El algoritmo *FP-Stream*, propuesto por Gianella *et al.* trabaja sobre el modelo *Time-Fading* particionando la ventana en diferentes granularidades, con la frecuencia de los conjuntos de ítems correspondientes a cada nivel de granularidad.

De los doce algoritmos, tres usan un modelo de ventana compuesto por unidades de tiempo (*time-based*) y el resto usa ventanas compuestas por secuencias o bloques de transacciones (*count-based*). Las ventanas de tipo *time-based* se caracterizan por ser más flexibles que las de tipo *count-based* debido a que no es necesario llenar la ventana con un número fijo de transacciones para para comenzar a procesar sus elementos. Por otra parte, en las ventanas de tipo *count-based*, el proceso de actualización tiene lugar por cada transacción que llega o expira, lo que deteriora el

Tabla 2. Características de los algoritmos de cálculo de conjuntos frecuentes/cerrados/maximales de ítems analizados.

Algoritmo	**Tipos de conjuntos de ítems**	**Modelo de procesamiento**	**Forma de procesamiento**	**Aproximación del resultado**
Lossy Counting	FIs	LandMark/count-based	por bloques	falsos-positivos
FDPM	FIs	Landmark/Count-based	por bloques	falsos-negativos
DSM-FI	FIs	Landmark/Time-based	por bloques	falsos-positivos
hMine	FIs	Landmark/Count-based	por transacciones	falsos-positivos
FP-Stream	FIs	Time-Fading/Time-based	por bloques	falsos-positivos
estDec	FIs	Time-Fading/Count-based	por transacciones	falsos-positivos
Moment	CFIs	Sliding/Count-based	por transacciones	exacto
CFI-Stream	CFIs	Sliding/Count-based	por transacciones	exacto
IncMine	CFIs	Sliding/Time-based	por bloques	falsos-negativos
StreamGen	CFIs	Sliding/Count-based	por transacciones	exacto
MFI-Trans-SW	FIs	Sliding/Count-based	por transacciones	exacto
Max-FISM	MFIs	Sliding/Count-based	por transacciones	exacto

rendimiento del algoritmo cuando se deben procesar grandes flujos de datos en los que la tasa de llegada de transacciones es alta. En general, es más recomendable el procesamiento por bloques de transacciones, especialmente cuando se trabaja con todo el flujo de datos (modelo *LandMark*) o con ventanas deslizantes de gran tamaño como propone Cheng *et al.* [12].

La mayoría de los algoritmos descritos son aproximados, también la mayoría son orientados a falsos positivos, excepto los propuestos por Yu *et al.* [63] y Cheng *et. al.* [12] que son orientados a falsos negativos. Los algoritmos orientados a falsos positivos usan un parámetro de error para obtener una mejor aproximación en los resultados, sin embargo mientras menor es el valor del error, mayor es el número de sub-FIs que deben ser almacenados, lo que trae consigo un alto consumo de memoria. Una posible solución a este problema la proponen Yu *et al.* [63] y Cheng *et al.* [12] cuando establecen un umbral de Soporte variable, que se incrementa gradualmente y que permite reducir el número de sub-FIs en memoria.

Los algoritmos exactos requieren almacenar el Soporte de todos los conjuntos de ítems que han pasado por la ventana de procesamiento, haciendo altamente costosa la tarea de minería y prácticamente intratable; por ello se recomienda utilizar este tipo de algoritmos cuando se trabaja con pequeñas ventanas deslizantes que procesan una parte del flujo, o cuando se trabaja con flujos de datos en los que la tasa de arribo de transacciones es baja.

El minado de FIs es una tarea altamente costosa y la mayoría de los algoritmos que trabajan sobre un modelo *LandMark* aplican algoritmos aproximados para establecer un adecuado balance entre el rendimiento del algoritmo y la calidad del conjunto de salida.

Una alternativa es calcular los CFIs, que son menos y tienen toda la información necesaria para calcular los soportes de todos los FIs. Sin embargo, en dependencia del tipo procesamiento y el tipo de resultado que se desea obtener, se puede degradar el proceso de minado como ocurre con los algoritmos Moment y CFI-Stream, que procesan el flujo transacción por transacción para dar un resultado exacto sacrificando el rendimiento del algoritmo, especialmente cuando la ventana deslizante es de gran tamaño. Con el algoritmo IncMine, los autores proponen un estudio que muestra como establecer un balance entre la calidad del conjunto resultante y el rendimiento del algoritmo, calculando CFIs lo más exacto posible, con un gran número de transacciones procesadas por segundo y un consumo de memoria constante, sobre una ventana deslizante.

Cuando el número de CFIs es muy grande, se puede considerar calcular los MFIs, que es muy eficiente en términos de uso de memoria y CPU. Sin embargo el mayor problema que presenta esta variante es la pérdida de información de los subconjuntos frecuentes que pueden ser calculados a partir de los MFIs.

Una posible solución sería el minado de conjuntos frecuentes-generadores de ítems (FGIs), con los que es posible obtener una representación más concisa de los FIs, y según el principio MDL (*Minimum Description Length Principle*) son muy útiles respecto a los CFIs para construir modelos de selección y clasificación. Estos tipos de conjuntos minimales, presentes en determinadas clases de equivalencia que forman los FIs, pueden ser muy tolerantes al ruido ya que se caracterizan por generar conjuntos de ítems de pequeña longitud.

4. Clasificadores

A pesar de la diversidad de técnicas propuestas para clasificar, hasta la fecha solo un reducido número de ellas, basadas en reglas de decisión se han adaptado para clasificar en flujos de datos. En esta sección se presentan cuatro clasificadores basados en reglas. El método general que siguen estos clasificadores es el siguiente:

Dado un flujo de transacciones F, un modelo de ventana W, una medida de calidad M y un criterio de decisión D: (1) Calcular el conjunto de reglas de manera incremental a partir de W, (2) Ordenar el conjunto de reglas según la medida de calidad M y (3) definir el criterio de decisión D para clasificar

Estos algoritmos se pueden dividir en dos tipos según la estrategia que usan para calcular las reglas: (1) los clasificadores basados en CARs, que usan algoritmos de cálculo de FIs y (2) los clasificadores basados en técnicas de inducción de reglas.

4.1. Clasificadores basados en CARs

Los clasificadores basados en CARs, por lo general, adaptan un algoritmo de cálculo de FIs para obtener el conjunto de CARs en un flujo de datos. Estos clasificadores siguen dos etapas para obtener el conjunto de CARs. En una primera etapa, durante el proceso de generación de las CARs o después de generadas, se aplican estrategias de poda para reducir el número de CARs. Luego, en una segunda etapa, se ordenan las CARs y se selecciona, según el orden establecido, un subconjunto de CARs que cubra a las transacciones de la ventana. Con este subconjunto de CARs se construye el clasificador.

4.1.1. StreamGenRules

El clasificador StreamGenRules, presentado en [27], genera un conjunto de CARs a partir de transacciones procesadas en un modelo *Sliding*. Este algoritmo recibe un conjunto de FGIs candidatos a CARs, calculados con el algoritmo StreamGen [27].

StreamGenRules ordena las CARs candidatas por los valores de ganancia (*gain*), con el objetivo de discriminar mejor las reglas. Luego, se seleccionan y almacenan las reglas que cubren a las transacciones de la ventana. A su vez se eliminan de la ventana las transacciones que fueron cubiertas por las reglas almacenadas. Con esta estrategia de cubrimiento, StreamGenRules trata de obtener reglas que separen o discriminen mejor el conjunto de transacciones de la ventana.

De forma similar a varios clasificadores del estado del arte, los autores construyen un modelo de clasificación basado en SVM (*Support Vector Machine*) cuyos vectores de características se forman con las reglas almacenadas.

Los experimentos realizados sobre seis conjuntos de datos del repositorio UCI, muestran que StreamGenRules obtiene, en promedio, mejor eficacia que el clasificador DDPMine [11], uno de los mejores clasificadores basados en CARs para datos estáticos.

4.1.2. AC-DS

El clasificador AC-DS [52] usa un modelo *LandMark* para procesar las transacciones y calcular las CARs. Para ello acepta dos parámetros: el umbral de Soporte y la cantidad de bloques de transacciones de la ventana. Por cada bloque de transacciones el algoritmo calcula el conjunto de reglas, las ordena en memoria y poda las que presenten valores de Soporte por debajo del umbral de Soporte establecido. El algoritmo permite clasificar una nueva transacción en cualquier momento.

Para calcular las reglas AC-DS aplica una estrategia de recorrido en Amplitud. Para ello realiza diferentes recorridos sobre cada bloque de transacciones procesado, en el primer recorrido calcula las CARs de tamaño 1 (ver Def. 2.13), en el segundo recorrido calcula las de tamaño 2 y en los siguientes recorridos utiliza las $(k-1)-CARs$ del recorrido anterior para calcular las $k-CARs$ del recorrido actual. Las CARs calculadas en cada recorrido se podan aplicando la propiedad de clausura descendente del Soporte.

Además, AC-DS utiliza una estructura en memoria con varias entradas para almacenar la reglas calculadas durante el proceso de generación de candidatos. Después de obtenido el conjunto de CARs, AC-DS las ordena de acuerdo con su valor de Confianza y elimina las reglas con valores de Soporte por debajo del umbral de Soporte establecido. Los autores no especifican cual es el criterio de decisión que utilizan para asignar una clase a una nueva transacción.

A pesar de que el clasificador AC-DS alcanza altos valores de eficacia en la clasificación, esta puede verse afectada por el uso de la medida Confianza atendiendo a sus limitaciones (véase subsección 2.2). Por otra parte, al seguir una estrategia de recorrido en amplitud, se producen muchas reglas, especialmente ante bajos umbrales de Soporte. Además, el tiempo que se emplea en el proceso de generación de reglas es elevado, debido a que se debe recorrer varias veces el bloque de transacciones.

4.2. Clasificadores basados en técnicas de inducción de reglas

En esta subsección se describen los clasificadores que generan reglas de decisión. A diferencia de los algoritmos anteriores, estos métodos aplican diferentes heurísticas para inducir las reglas.

4.2.1. FACIL

En [22], los autores proponen un clasificador denominado FACIL, induce reglas formadas por conjuntos de intervalos cerrados definidos por hiperrectángulos. Este clasificador recibe cada transacción del flujo como un vector normalizado en el intervalo $[0, 1]$, seguido de la clase. FACIL no utiliza una ventana global para procesar las transacciones, sino que usa por cada regla una ventana que contiene las transacciones que cubrió dicha regla en su proceso de construcción.

El cálculo de las reglas se realiza comprobando, para cada nueva transacción, si se cumple alguna de las tres situaciones siguientes: (1) "Cobertura Positiva", si la transacción es cubierta

por una o más reglas asociadas a su misma clase, en ese caso se estima la nueva región del espacio que sería ocupada por la regla a expandir, favoreciendo a aquella que supone el menor número de cambios en términos de su antecedente; (2) "Cobertura Negativa", si la transacción es cubierta por una regla asociada a una etiqueta distinta, esta se añade a la ventana de la regla; (3) "Nueva Descripción", si la transacción no es cubierta por ninguna regla, se crea una nueva que describa a la transacción.

Simultáneamente al proceso de actualización, FACIL lleva a cabo una estrategia de poda a medida que visita las reglas del modelo con el objetivo de refinarlo. Antes de comprobar si una regla cubre a una nueva transacción, ésta se elimina si la última regla generalizada asociada a su misma clase la cubre en su totalidad. Si no es el caso, tras comprobar que la regla visitada no cubre a la nueva transacción ésta se elimina si satisface al menos una de las dos condiciones siguientes: (1) Es una regla dudosa con un Soporte inferior al de cualquiera de las reglas generadas a partir de las antiguas transacciones de su ventana, (2) El número de veces que dicha regla impidió a otras de distinta etiqueta ser generalizadas (expandirse) es mayor que su Soporte.

Las reglas que conforman el modelo de FACIL pueden ser consistentes o inconsistentes. Las primeras son las que no cubren en su proceso de construcción transacciones con distinta clase; las segundas, son aquellas reglas que cubren transacciones con valores muy cercanos en su antecedente pero con distinta clase (transacciones de frontera). *FACIL* usa un mecanismo de olvido que permite desechar transacciones de las ventanas cuando estas se vuelven obsoletas o dejan de ser relevantes.

Para clasificar una nueva transacción, en el caso de las reglas consistentes usa el criterio de "La Mejor Regla" (ver subsección 2.4), las reglas inconsistentes clasifican las transacciones mediante el vecino más cercano.

Debido a la estrategia que sigue para calcular las reglas y al criterio de clasificación flexible, las reglas pueden ser inconsistentes sin dañar la exactitud del modelo, con lo que se consiguen varios beneficios, entre ellos, reducir el coste computacional y refinar el modelo simultáneamente al proceso de actualización sin comprometer la eficiencia en el aprendizaje. Sin embargo, al seguir el criterio "La Mejor Regla", este clasificador apuesta a una sola regla para clasificar y no se puede esperar que una sola regla prediga correctamente la clase de cada transacción que ésta cubra. Los resultados experimentales obtenidos mediante bases de datos del almacén UCI y flujos de datos dinámicos a partir de planos rotantes demuestran el buen rendimiento de FACIL como clasificador multi-clase de propósito general.

4.2.2. VFDR

En [24], los autores proponen el clasificador VFDR, basado en reglas de decisión. VFDR utiliza una estructura en memoria L_r que permite almacenar incrementalmente las reglas calculadas.

Cada regla es una conjunción de literales formados por condiciones de la forma: $(at > v)$ o $(at \leq v)$ para atributos continuos, y $(at = v_j)$ en el caso de los atributos nominales, donde at es un atributo, v alguna constante y v_j algún valor discreto del conjunto de valores de at.

El proceso del cálculo de las reglas es muy parecido al proceso de construcción de los árboles de decisión generados por los algoritmos que usan la estrategia del *VFDT* (*Very Fast Decision Tree*)[19]. A medida que se reciben nuevas transacciones, se verifican las reglas presentes en la estructura que cubren a la transacción y se almacenan las estadísticas necesarias. Inicialmente la estructura se encuentra vacía, por lo que se crea una nueva entrada con la regla por defecto (regla vacía) y se acumula la información de las primeras N transacciones necesarias y suficientes para expandir la regla sin dañar la precisión del modelo. Por cada regla que tenga suficiente

información para ser expandida, se realiza su crecimiento por el literal que minimize la entropía de la clase de las transacciones cubiertas por la regla.

La desigualdad de Hoeffding se utiliza como cota de error para determinar N. Cada entrada de la estructura que almacena las reglas contiene: un valor entero que almacena la cantidad de transacciones cubiertas por la regla, un vector que almacena información para calcular la probabilidad $p(c_i)$ de recibir transacciones en cada clase c_i, una matriz $p(at = v_j|c_i)$,con los valores necesarios para calcular la probabilidad de recibir determinados valores vj de atributos nominales, por cada etiqueta de clase c_i, y un "btree"que almacena información para calcular la probabilidad $p(at > v_j|c_i)$, de recibir valores mayores a un valor v_j para atributos continuos, por cada etiqueta de clase c_i.

VFDR genera dos tipos de reglas: ordenadas y no ordenadas, según la forma en que fueron calculadas. En el primer grupo, por cada transacción que llega solo se actualizan las estadísticas necesarias de la primera regla que fue cubierta por dicha transacción. En el segundo grupo, cada transacción sirve para actualizar las estadísticas de todas las reglas que la cubren. Si alguna transacción no fue cubierta por ninguna regla entonces se actualiza la regla por defecto.

Para clasificar una nueva transacción los autores aplican dos criterios, "La mejor Regla" y "Todas las reglas". En ambos casos, para asignar la clase a la nueva transacción, aplican el Teorema de Bayes, de forma que se asigna la clase que maximice la probabilidad a posteriori dada por la regla de Bayes, asumiendo la independencia de los atributos dada la clase. De esta forma se usa mejor la información estadística almacenada por cada regla.

En los experimentos, los autores evaluaron el poder de predicción del clasificador usando dos estrategias de clasificación, la primera asignando la clase mayoritaria y la segunda aplicando la regla de Bayes a partir de las estadísticas almacenadas en L_r. En promedio, el clasificador que siguió la segunda estrategia presentó mejor poder de predicción, especialmente en presencia de datos ruidosos. Por otra parte, los conjuntos de reglas no ordenadas para construir el clasificador son más competitivas en cuanto a poder de predicción e interpretabilidad respecto a los conjuntos de reglas ordenadas, sin embargo sufren el problema de tener un alto número de reglas afectando de manera moderada el rendimiento del algoritmo. Además los criterios de decisión aplicados pueden afectar la eficacia del clasificador al tener en cuenta a una sola regla (La mejor Regla) o al incluir reglas de baja calidad (Todas las reglas).

4.3. Síntesis y conclusiones

En la sección anterior fueron expuestos los principales trabajos del estado del arte relacionados con la temática. Los clasificadores presentados se dividieron en dos grupos de acuerdo a la estrategia que siguen para calcular las reglas: (1) Clasificadores basados en CARs, y (2) Clasificadores que inducen reglas de decisión. En todos los casos, se describieron las etapas fundamentales que siguen los clasificadores: (1) cálculo de las reglas, (2) estrategias de ordenamiento y (3) criterios de decisión para clasificar una nueva transacción. En la Tabla 4.3 se muestra un estudio comparativo de los algoritmos presentados.

Como se puede observar, los clasificadores basados en CARs reportados siguen dos etapas. En la primera etapa durante el proceso de generación de las CARs o después de generadas, se aplican estrategias de poda para reducir el número de CARs. Luego, en la segunda etapa, se ordenan las CARs y se seleccionan las que conformarán el clasificador final mediante un proceso de cubrimiento. Este mecanismo es altamente costoso debido a que se deben recorrer las transacciones de la ventana en más de una ocasión, antes de pasar a procesar un nuevo bloque de transacciones.

Tabla 3. Características de los clasificadores analizados.

Algoritmo	**Estrategia de cálculo de reglas / Algoritmo**	**Estrategia de ordenamiento**	**Criterio de decisión**
StreamGenRules	cálculo de FGIs / StreamGen	Ganancia de información	SVM
AC-DS	cálculo de FIs / Lossy Counting	Confianza	-
FACIL	inducción de reglas	-	Mejor Regla / Vecino más cercano
VFDR	inducción de reglas	-	Mejor Regla / Todas las reglas

Por otro lado los criterios de ordenamiento usados en estos clasificadores, mediante el uso de las medidas Confianza y Ganancia no permiten hacer una mejor selección de las reglas en caso de existir empate entre ellas.

Los algoritmos de inducción de reglas: FACIL y VFDR, trabajan de forma similar a los clasificadores integrados que define Hernández *et. al.*[31] en su trabajo. Estos utilizan diferentes heurísticas que incluyen estrategias de poda y estrategias de ordenamiento para calcular directamente el conjunto final de reglas. De esta forma construyen el clasificador en una sola etapa, evitando el costoso proceso de cubrimiento de los clasificadores basados en CARs.

5. Conclusiones

En este reporte se realizó un análisis de los clasificadores basados en reglas para flujos de datos. Estos clasificadores se dividieron en dos grupos teniendo en cuenta la estrategia usada para calcular las reglas. Estos grupos están compuestos por: los basados en reglas de asociación de clases y los basados en técnicas de inducción de reglas.

Los clasificadores basados en CARs reportados siguen dos etapas con un mecanismo altamente costoso, dado por el algoritmo utilizado para calcular las CARs. De ahí que la mayoría de los algoritmos analizados se enfocaron en este sentido (sección 3). De estos algoritmos, una gran cantidad son de aproximación de resultados, y muchos orientados a falsos positivos, excepto dos que son orientados a falsos negativos. Los primeros consumen mucha memoria debido a la gran cantidad de conjuntos de ítems sub-frecuentes que es necesario mantener en memoria para obtener el resultado final. Por otro lado los algoritmos exactos analizados requieren almacenar la informació de todos los FIs que han pasado por la ventana de procesamiento, haciendo altamente costoso el cálculo y prácticamente intratable.

Una posible solución que puede acelerar el proceso de minado y disminuir el consumo de memoria es el minado de conjuntos de ítems de otros tipos como los CIs, MIs, FIGs y otros que permitan obtener una representación más concisa de los FIs.

Los algoritmos de inducción de reglas trabajan de forma similar a los clasificadores integrados, que utilizan diferentes heurísticas que incluyen estrategias de poda y estrategias de ordenamiento para calcular directamente el conjunto final de reglas. De esta forma construyen el clasificador en una sola etapa, evitando el costoso proceso de cubrimiento de los clasificadores basados en CARs.

Referencias bibliográficas

1. C. C. Aggarwal. *Data Streams Models and Algorithms*. Springer Science+Business Media, LLC, 2007.
2. R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of ítems in large databases. *SIGMOD Rec.*, 22(2):207–216, 1993.
3. R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, 1994.
4. A. Alzghoul and M. Lofstrand. Increasing availability of industrial systems through data stream mining. *Comput. Ind. Eng.*, 60(2):195–205, 2011.
5. J. Beringer and E. Hüllermeier. Efficient instance-based learning on data streams. *Intell. Data Anal.*, 11(6):627–650, 2007.
6. F. Berzal, I. J. Blanco, D. Sánchez, and M. A. Vila. Measuring the accuracy and interest of association rules: A new framework. *Intell. Data Anal.*, 6(3):221–235, 2002.
7. F. Berzal, J. C. Cubero, N. Marín, D. Sánchez, J. M. Serrano, and A. Vila. Association rule evaluation for classification purposes. In *III Taller Nacional de Minería de Datos y Aprendizaje*, pages 135–144, 2005.
8. A. Bifet. Adaptive stream mining: Pattern learning and mining from evolving data streams. In *Proceedings of the 2010 conference on Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*, pages 1–212, 2010.
9. A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 139–148, 2009.
10. J. H. Chang and W. S. Lee. Finding recent frequent ítemsets adaptively over online data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 487–492, New York, NY, USA, 2003. ACM.
11. H. Cheng, X. Yan, J. Han, and P. S. Yu. Direct discriminative pattern mining for effective classification. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 169–178, 2008.
12. J. Cheng, Y. Ke, and W. Ng. Maintaining frequent closed ítemsets over a sliding window. *J. Intell. Inf. Syst.*, 31(3):191–215, 2008.
13. H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4), 1952.
14. Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz. Catch the moment: maintaining closed frequent ítemsets over a data stream sliding window. *Knowl. Inf. Syst.*, 10(3):265–294, 2006.
15. P. Clark and R. Boswell. Rule induction with cn2: Some recent improvements. In *Proceedings of the European Working Session on Machine Learning*, EWSL '91, pages 151–163, 1991.
16. F. Coenen, P. Leng, and S. Ahmed. Data structures for association rule mining: T-trees and p-trees. In *IEEE Transactions on Knowledge and Data Engineering*, pages 774–778, 2004.
17. W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, 1995.
18. G. Cormode and M. Hadjieleftheriou. Finding frequent ítems in data streams. *Proc. VLDB Endow.*, 1(2):1530–1541, 2008.
19. P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 71–80, 2000.
20. R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. John Willey & Sons, 1973.
21. Z. Farzanyar, M. R. Kangavari, and N. Cercone. Max-fism: Mining (recently) maximal frequent ítemsets over data streams using the sliding window model. *Computers & Mathematics with Applications*, 64(6):1706–1718, 2012.
22. F. J. Ferrer, J. S. Aguilar, and J. C. Riquelme. Incremental rule learning and border examples selection from numerical data streams. *Journal of Universal Computer Science*, 11(8):1426–1439, 2005.
23. J. Gama. *Knowledge Discovery from Data Streams*. Chapman and Hall CRC, 2010.
24. J. Gama and P. Kosina. Learning decision rules from data streams. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Volume Two*, pages 1255–1260, 2011.
25. J. Gama, P. Medas, and R. Rocha. Forest trees for on-line data. In *Proceedings of the 2004 ACM symposium on Applied computing*, SAC '04, pages 632–636, 2004.
26. J. Gama, R. Rocha, and P. Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 523–528, 2003.

27. C. Gao and J. Wang. Efficient ítemset generator discovery over a stream sliding window. In *Proceedings of the 18th ACM conference on Information and knowledge management*, CIKM '09, pages 355–364, 2009.
28. L. Geng and H. Hamilton. Choosing the right lens: Finding what is interesting in data mining. In *Quality Measures in Data Mining*, volume 43 of *Studies in Computational Intelligence*, pages 3–24. Springer Berlin / Heidelberg, 2007.
29. C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu. Mining frequent patterns in data streams at multiple time granularities, 2002.
30. J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, 2000.
31. R. Hernández. *Desarrollo de Clasificadores basados en Reglas de Asociación de Clase*. PhD thesis, Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla, 2011.
32. L. Hua-Fu, H. Chin-Chuan, S. Man-Kwan, and L. Sun-Yin. Efficient maintenance and mining of frequent Ítemsets over online data streams with a sliding window. In *Systems, Man and Cybernetics, 2006. SMC '06. IEEE International Conference on*, volume 3, pages 2672–2677, 2006.
33. L. Hua-fu and L. Suh yin andS. Man-kwan. An efficient algorithm for mining frequent Ítemsets over the entire history of data streams. In *In Proc. of First International Workshop on Knowledge Discovery in Data Streams*, 2004.
34. G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *IN PROC. OF THE 2001 ACM SIGKDD INTL. CONF. ON KNOWLEDGE DISCOVERY AND DATA MINING*, pages 97–106, 2001.
35. N. Jiang and L. Gruenwald. Cfi-stream: mining closed frequent ítemsets in data streams. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '06, pages 592–597, 2006.
36. R. Jin. Efficient decision tree construction on streaming data. In *In 9th ACM International Conference on Knowledge Discovery and Data Mining (SIGKDD*, pages 571–576, 2003.
37. S. C. Karacal. Data stream mining for machine reliability. In *IIE Annual conference and exhibition*, 2006.
38. H. Kargupta, R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa, and D. Handy. Vedas: A mobile and distributed data stream mining system for real-time vehicle monitoring. In *SDM*. SIAM, 2004.
39. J. Z. Kolter and M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res.*, 8:2755–2790, 2007.
40. N. Lavrac, P. Flach, and B. Zupan. Rule evaluation measures: A unifying view. In *9th International Workshop on Inductive Logic Programming*, pages 174–185, 1999.
41. P. Lenca, B. Vaillant, P. Meyer, and S. Lallich. Association rule interestingness measures: Experimental and theoretical studies. In *Quality Measures in Data Mining*, volume 43 of *Studies in Computational Intelligence*, pages 51–76. Springer Berlin / Heidelberg, 2007.
42. W. Li, J. Han, and J. Pei. Cmar: accurate and eficient classification based on multiple class-association rules. In *IEEE International Conference on Data Mining*, pages 369–376, 2001.
43. Z. Li, J. Yang, and J. Zhang. Dynamic incremental svm learning algorithm for mining data streams. In *Proceedings of the The First International Symposium on Data, Privacy, and E-Commerce*, ISDPE '07, pages 35–37, 2007.
44. B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *4th International Conference on Knowledge Discovery and Data Mining*, pages 80–86, 1998.
45. G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 346–357, 2002.
46. M. M. Masud, T. M. Al-Khateeb, K. W. Hamlen, J. Gao, L. Khan, J. Han, and B. Thuraisingham. Cloud-based malware detection for evolving data streams. *ACM Trans. Manage. Inf. Syst.*, 2(3):16:1–16:27, 2008.
47. C. O. Nikunj and R. Stuart. Online bagging and boosting. In *In Artificial Intelligence and Statistics 2001*, pages 105–112, 2001.
48. G. Piatetsky-Shapiro. Discovery, analysis and presentation of strong rules. In *Knowledge Discovery in Databases*, pages 229–248. AAAI Press, 1991.
49. J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., 1993.
50. J. R. Quinlan and R. M. Cameron-Jones. Foil: A midterm report. In *Proceedings of the European Conference on Machine Learning*, 1993.
51. W. N. Street and Y. Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 377–382, 2001.
52. L. Su, H. Liu, and Z. Song. A new classification algorithm for data stream. *I.J. of Modern Education and Computer Science*, 3(4):32–39, 2007.

53. X. Tian, Q. Sun, X. Huang, and Y. Ma. Dynamic online traffic classification using data stream mining. In *Proceedings of the 2008 International Conference on MultiMedia and Information Technology*, pages 104–107, 2008.
54. D. Torres, J. Aguilar, and Y. Rodríguez. Classification model for data streams based on similarity. In *Proceedings of the 24th international conference on Industrial engineering and other applications of applied intelligent systems conference on Modern approaches in applied intelligence - Volume Part I*, IEA/AIE'11, pages 1–9, 2011.
55. A. Veloso, W. Meira Jr., and M. J. Zaki. Lazy associative classification. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM'06, pages 645–654, 2006.
56. E. T. Wang and A. L. Chen. A novel hash-based approach for mining frequent ítemsets over data streams requiring less memory space. *Data Min. Knowl. Discov.*, 19(1):132–172, 2009.
57. H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 226–235, 2003.
58. Y. J. Wang, X. Qin, and F. Coenen. A novel rule weighting approach in classification association rule mining. In *Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, pages 271–276, 2007.
59. W. Xu, Z. Qin, H. Hu, and N. Zhao. Mining uncertain data streams using clustering feature decision trees. In *Proceedings of the 7th international conference on Advanced Data Mining and Applications - Volume Part II*, ADMA'11, pages 195–208, 2011.
60. J. W. Yanbo, X. Qin, and F. Coenen. Hybrid rule ordering in classification association rule mining. *Trans. MLDM*, 1(1):1–15, 2008.
61. H. Yang and S. Fong. Moderated vfdt in stream mining using adaptive tie threshold and incremental pruning. In *Proceedings of the 13th international conference on Data warehousing and knowledge discovery*, DaWaK'11, pages 471–483, 2011.
62. X. Yin and J. Han. Cpar: Classification based on predictive association rules. In *SDM'03*, 2003.
63. J. X. Yu, Z. Chong, H. Lu, and A. Zhou. False positive or false negative: mining frequent ítemsets from high speed transactional data streams. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 204–215, 2004.
64. M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.
65. M. Zareapoor, K. R. Seeja, and M. A. Alam. Analysis on credit card fraud detection techniques: Based on certain design criteria. *International Journal of Computer Applications*, 52(3):35–42, 2012.
66. P. Zhang, X. Zhu, and L. Guo. Mining data streams with labeled and unlabeled training examples. In *Proceedings of the 2009 Ninth IEEE International Conference on Data Mining*, ICDM '09, pages 627–636, 2009.

Notaciones

$\Rightarrow$	Operador lógico de implicación
$\in$	Pertenencia
$\subseteq$	Subconjunto
$sop(X)$	Soporte del conjunto de ítems X
$frec(X)$	Frecuencia del conjunto de ítems X
$Efrec(X)$	Valor aproximado de la frecuencia real X
$conf(X)$	Confianza del conjunto de ítems X
$\|\ \|$	Operador de cardinalidad
$\cup$	Unión de conjuntos
$conf(X \Rightarrow Y)$	Confianza de la regla $X \Rightarrow Y$
$k-$itemset	Conjunto de ítems de tamaño k
$k-$CAR	CAR de tamaño k
BD	Conjunto de datos
$Netconf(X \Rightarrow Y)$	Netconf de la regla $X \Rightarrow Y$

Acrónimos

CAR	Regla de Asociación de Clase
QM	*Quality Measure*
CSA	Confianza - Soporte - Longitud del Antecedente
ACS	Longitud del Antecedente - Confianza - Soporte
WRA	*Weighted Relative Accuracy*
LAP	*Laplace Expected Error Estimate*
SR-QM	*Specific Rules - Quality Measure*
FI	*Frequent Itemset*
sub-FI	*sub-Frequent Itemset*
FIFO	*First In First Out*
MFI	*Maximal Frequent Itemset*
CFI	*Closed Frequent Itemset*
DSM-FI	*Data Stream Mining for Frequent Itemset*
FDPM	*Frequent Data Stream Pattern Mining*
FP-stream	*Frequent Pattern - stream*
CTE	*Closed Enumeration Tree*
FP-tree	*Frequent Pattern - tree*
CFI-stream	*Closed Frequent Itemset - stream*
DIU	*Direct Update Tree*
semi-CFI	*semi-Closed Frequent Itemset*
MDL	*Minimum Description Principle*
VFDT	*Very Fast Decision Tree*

RT_021, noviembre 2013
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Lucía González Bayona
Diseño de Portada: Di. Alejandro Pérez Abraham
RNPS No. 2143
ISSN 2072-6260
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. No. 21812 e/218 y 222, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba