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

Búsqueda automática de reglas para detección de fraudes en flujos de eventos

Vitali Herrera-Semenets y Andrés Gago-Alonso

RT_029 **enero 2015**

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

Búsqueda automática de reglas para detección de fraudes en flujos de eventos

Vitali Herrera-Semenets y Andrés Gago-Alonso

RT_029 **enero 2015**

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

Siglas y acrónimos

ACS: Sistema de colonia de hormigas artificial (en inglés *Ant Colony System*).
Ant-Miner: Sistema de colonia de hormigas para el descubrimiento de reglas (en inglés *Ant Colony-Based Data Miner*).
AQ21: Algoritmo de inducción natural.
ARFF: Formato de fichero de relación entre atributos (en inglés *Attribute-Relation File Format*).
BACQ: Algoritmo para obtener cláusulas generales basado en ILP.
BPM: Minería de procesos de negocio (en inglés *Business Process Mining*).
C5.0: Algoritmo que permite generar reglas basándose en los árboles de decisión.
CART: Algoritmo que permite la construcción de un árbol de clasificación y regresión.
CEP: Procesamiento de eventos complejos (en inglés *Complex Event Processing*).
Cigol: Algoritmo para obtener cláusulas generales basado en ILP.
Golem: Algoritmo para obtener cláusulas generales basado en ILP.
ILP: Programación lógica inductiva (en inglés *Inductive Logic Programming*).
LEF: Evaluación funcional lexicográfica (en inglés *Lexicographical Evaluation Functional*).
FURIA: Algoritmo de inducción de reglas difusas sin ordenar (en inglés *Fuzzy Unordered Rule Induction Algorithm*).
MDL: Longitud de la descripción mínima (en inglés *Minimum Description Length*).
MXML: Minería de lenguaje de marcado extensible (en inglés *Mining eXtensible Markup Language*).
PART: Algoritmo que permite generar reglas basándose en los árboles de decisión.
Progol: Algoritmo para obtener cláusulas generales basado en ILP.
Progolem: Algoritmo para obtener cláusulas generales basado en ILP.
RIPPER: Poda incremental repetida para producir reducción del error (en inglés *Repeated Incremental Pruning to Produce Error Reduction*).
SBVR: Semántica de vocabulario de negocios y reglas de negocios (en inglés *Semantics of Business Vocabulary and Rules*).
SMS: Servicio de mensajes cortos (en inglés *Short Message Service*).
SVM: Maquinas de soporte vectorial (en inglés *Support Vector Machines*).
X2R: Algoritmo de generación de reglas basado en discretización y selección de características.

Notaciones

a_i	i-ésimo atributo
a^-	Valor mínimo de un atributo
a^+	Valor máximo de un atributo
A_k	k-ésimo conjunto de atributos
b	Cantidad de bits
c_l	l-ésima clase
C	Conjunto de clases
d	Cantidad de divisiones de un conjunto de datos
D	Conjunto de datos
D_i	i-ésimo conjunto de datos
D_t	Conjunto de entrenamiento
D^{c_l}	Conjunto de datos de entrenamiento etiquetados con la clase c_l
$\bar{D}$	Conjunto de datos discretizados
e_i	i-ésimo evento
$\bar{e}$	Contador de eventos de interés
$\overrightarrow{e}$	Traza de evento
$\overrightarrow{e_i}$	i-ésima traza de evento
$\overrightarrow{e}^+$	Traza positiva
$\overrightarrow{E}$	Conjunto de trazas de eventos
$\bar{E}$	Conjunto de tipos de eventos atómicos
$\tilde{E}(I_i)$	Conjunto de eventos explícitos en el intervalo de tiempo I_i
f, g	Fórmulas
$G(m)$	Ganancia de información para una condición m
h	Conjunto de eventos históricos
$\tilde{h}$	Conjunto de eventos estimados
i, k, l	Subíndices (números enteros no negativos)
I_i	i-ésimo intervalo de tiempo
K_i	i-ésimo subconjunto de A_i
m	Condición
n	Instancia
n_i	i-ésima instancia
N	Conjunto de instancias
o	Objeto
O	Conjunto de objetos
p_i	i-ésimo parametro
$\tilde{P}(I_i)$	Conjunto estimado de parámetros para el intervalo de tiempo I_i
$\bar{q}$	Cláusula general
r	Regla
r_i	i-ésima regla
$\bar{r}$	Cláusula
$\overleftarrow{r}, \overrightarrow{r}$	Fórmulas en U
R	Conjunto de reglas
R_i	i-ésimo conjunto de reglas

t	Árbol
T	Conjunto de árboles
u	Valor del umbral correspondiente al nodo de decisión
$\tilde{u}$	Valor de tolerancia asociado al umbral
$\check{u}$	Umbral inferior
$\hat{u}$	Umbral superior
U	Universo
v	Número real
$\bar{v}$	Valor de una variable
V_i	i-ésimo conjunto de valores
V_i'	i-ésimo subconjunto de V_i
W	Ventana de tiempo
$\hat{W}$	Umbral de ventana de tiempo
x_i	i-ésima variable
X	Conjunto de variables
z	Porcentaje de modificación
z_u	Porcentaje de modificación automático asociado a u
$\hat{z}$	Porcentaje máximo permitido de modificación
$\check{z}$	Porcentaje mínimo permitido de modificación
$\alpha[a]$	Nivel de significación del atributo a
δ	Función de asignación de clases a instancias
ε	Conjunto de ejemplos de cláusulas
ε'	Subconjunto de ε
ε^+	Conjunto de ejemplos positivos
ε^-	Conjunto de ejemplos negativos
η	Actividad del negocio
ι	$l-$complejo
λ	Hipótesis
π_i	i-ésima función de proyección
$\Re$	Conjunto de restricciones
$\Re_r$	Conjunto de restricciones de la regla r
$\Re_{\overrightarrow{e}}$	Conjunto de restricciones que satisface la traza $\overrightarrow{e}$
ξ	Función de corrección
ς	Nivel de rendimiento
ρ	Fórmula básica de transición de parámetros
ϱ	Función para calcular la sumatoria del índice Gini de los nodos sucesores
$\hat{\varrho}$	Función para calcular la ganancia Gini
τ	Tendencia
$\check{\phi}$	Límite de soporte inferior
$\hat{\phi}$	Límite de soporte superior
$\check{\varphi}$	Límite central inferior
$\hat{\varphi}$	Límite central superior
ψ	Función de ruido predefinida
$\oplus$	Operador de comparación
$\ominus$	Conjunto de signos de desigualdad

Tabla de contenido

Lista de figuras

Lista de tablas

Búsqueda automática de reglas para detección de fraudes en flujos de eventos

Vitali Herrera-Semenets y Andrés Gago-Alonso

Equipo de Investigaciones de Minería de Datos, Centro de Aplicaciones de Tecnologías de Avanzada (CENATAV), {vherrera,agago}@cenatav.co.cu

RT_029, Serie Gris, CENATAV
Aceptado: 14 de enero de 2015

Resumen. En el campo de la detección de fraude, los métodos basados en evaluación de reglas están considerados entre los más utilizados por su eficiencia. Las técnicas de búsqueda automática de reglas pueden ser vistas como una evolución de aquellas basadas en evaluación de reglas. Donde se sustituye o se complementa la generación manual de reglas por un proceso de descubrimiento de patrones que conlleva a la formación de las mismas. De esta manera se reduce el trabajo manual de los analistas y se obtienen reglas que están implícitas dentro de los grandes volúmenes de datos y que difícilmente serían encontradas por un humano. En este trabajo son analizadas una variedad de técnicas de búsqueda automática de reglas. Además, se analizan métodos de otros campos, tales como: procesos de negocio, procesamiento de eventos complejos y programación lógica inductiva. Con dicho análisis se muestra como estos enfoques pueden ser extendidos con el fin de obtener reglas que puedan aplicarse en la detección de fraude. Finalmente, es presentado un resumen de las bases de datos utilizadas para evaluar las técnicas discutidas.

Palabras clave: búsqueda automática de reglas, descubrimiento de reglas de negocios, detección de fraude.

Abstract. In the fraud detection area, the rule-based evaluation methods are considered one of the most frequently used due to its efficiency. The techniques based on automatic rule search can be viewed as an evolution of the rule-based evaluation methods. Where manual rule generation is replaced or supplemented by a pattern discovery process, leading to the formation of rules. In this way, the manual work of analysts is reduced, and rules which cannot be conceived by humans due to the large volume of data to be processed can be obtained. In this paper, a variety of techniques based on automatic rule search are analyzed. Furthermore, several proposals from other fields are also discussed, including: business processes, complex event processing and inductive logic programming. This complementary discussion shows how they can be extended in order to obtain rules, which can be used in fraud detection. Finally, a summary about the databases used for evaluating the discussed techniques is presented.

Keywords: automatic rule search, discovery of business rules, fraud detection.

1. Introducción

Las actividades delictivas, conocidas como fraude, representan pérdidas millonarias para las compañías que ofrecen servicios de comunicaciones [1]. Los fraudes pueden ser llevados a cabo a través de medios como Internet, computadoras o servicios de telecomunicaciones, lo cual hace que sea

necesaria para su detección, la implementación y puesta en práctica de técnicas automatizadas. La detección de fraude en áreas como subastas en línea [2], tarjetas de crédito [3], mercado de valores [4], servicios de telecomunicaciones [5,6,7,8], entre otras, cuentan con varias propuestas basadas en distintos enfoques [9].

En el trabajo de Herrera-Semenets et al. [9] se presenta un análisis sobre los métodos de detección de fraude en servicios de telecomunicaciones y se describen algunas técnicas utilizadas en otras áreas de aplicación. Algunas de las técnicas más empleadas son las basadas en evaluación de reglas, detección automática de reglas, detección de anomalías, análisis de redes sociales, redes bayesianas, redes neuronales, así como sistemas híbridos.

Las técnicas basadas en evaluación de reglas [10,11,12] son muy eficientes, pero muy difíciles de administrar, ya que requieren de mucho trabajo para especificar reglas en cada caso de fraude posible. En efecto, el conjunto de reglas a utilizar se define con anterioridad por un grupo de analistas. En cambio, en las técnicas basadas en detección automática de reglas, los datos son procesados por un algoritmo que genera el conjunto de reglas. Las reglas obtenidas pueden ser analizadas y filtradas por los analistas para complementar una base de reglas ya existentes.

Este trabajo se centra en los métodos para detectar fraudes basados en búsqueda automática de reglas. Estos métodos se pueden ver como una generalización de los árboles de decisión [13], donde podría aplicarse más de una regla o ninguna, permitiendo abarcar varias condiciones y usar reglas por defecto.

Dentro de los enfoques analizados en este trabajo están aquellos que utilizan algoritmos de aprendizaje basados en árboles de decisión (ver sección 3.2), donde los árboles obtenidos son representados como conjuntos de reglas [14,15,16]. Este tipo de algoritmos utilizan una técnica de partición [13]. Dicha técnica construye las reglas mediante condiciones que particionan el conjunto de datos, donde para cada condición existen condiciones complementarias. En este sentido, un árbol de decisión siguiendo el camino disponible según las condiciones, puede llevar un ejemplo hasta una hoja y asignarle una clase.

Otra forma de obtener un conjunto de reglas es mediante algoritmos voraces (ver sección 3.1), los cuales implementan una técnica de cobertura [13]. Esta técnica consiste en ir añadiendo reglas a medida que se vayan cubriendo los ejemplos de forma consistente. Los ejemplos ya cubiertos por las reglas obtenidas se van descartando y se comienzan a procesar de nuevo los ejemplos restantes.

Tanto los algoritmos que utilizan técnicas de partición, como los que utilizan técnicas de cobertura, presentan características que los hacen útiles para la detección de fraude, por ejemplo:

- Son aplicables para la clasificación de ejemplos como fraudulentos o legítimos.
- Pueden procesar atributos continuos y discretos.
- Por lo general son eficientes y existen variantes escalables para grandes volúmenes de datos.
- Pueden ser usados en contextos donde existan atributos no significativos y valores faltantes.

También son analizadas en este trabajo propuestas enfocadas a otras áreas, pero que pueden ser extendidas para su aplicación en la detección de fraudes. Dentro de los cuales, se incluyen técnicas que posibilitan actualizar reglas existentes (ver secciones 4.1 y 5.2). Además, son presentados métodos que generan elementos como $l-$complejos (ver sección 3.3) o cláusulas (ver sección 6), que dada su estructura es posible representarlos como reglas. También son analizadas las bases de datos utilizadas por varias propuestas para evaluar sus resultados (ver sección 7).

Este trabajo está compuesto por ocho secciones. La primera sección consiste en la introducción al tema tratado. Algunos conceptos básicos para comprender el resto del trabajo son presentados en la segunda sección. En la tercera sección se analizan distintos enfoques de técnicas para la

obtención de reglas de clasificación. Las propuestas para el descubrimiento de reglas en procesos de negocios se abordan en la cuarta sección. En la quinta sección son analizados métodos que descubren reglas mediante el procesamiento de eventos complejos. Los algoritmos basados en programación lógica inductiva que aplican inferencia inductiva para obtener reglas son abordados en la sexta sección. Las bases de datos utilizadas en las experimentaciones son analizadas en la séptima sección. Finalmente, la octava sección presenta las conclusiones generales de la investigación.

2. Conceptos básicos

En este reporte se analizan diversos enfoques que generan reglas. Para comprender el funcionamiento de dichos enfoques es necesario tener en cuanta algunos aspectos, los cuales son presentados a continuación.

2.1. Instancias y reglas simples

Sean $A_1, A_2, \ldots, A_k$ conjuntos cuyos elementos son conocidos como atributos. El número entero k, $k > 1$, representa la cantidad de atributos que caracterizan los datos que se desean procesar; este valor numérico es también conocido como *dimensión del problema*. En efecto, todos los algoritmos para la búsqueda de reglas procesan secuencias o flujos de instancias, donde cada instancia $n = (a_1, a_2, \ldots, a_k)$ es un vector perteneciente al *universo* $U = A_1 \times A_2 \times \ldots \times A_k$, donde $a_i \in A_i$, para cada $1 \leq i \leq k$.

Las instancias pueden ser conocidas como eventos cuando uno de los conjuntos A_i contiene los instantes de tiempo en que ocurre cada instancia. En ese caso, el i-ésimo atributo será conocido como *atributo temporal*.

Sean $\pi_1, \pi_2, \ldots, \pi_k$ funciones tales que $\pi_i : U \mapsto A_i$ tal que dado una instancia $n = (a_1, a_2, \ldots, a_k) \in U$ se tiene que $\pi_i(n) = a_i$, para cada $1 \leq i \leq k$. Estas funciones son conocidas en la literatura como *funciones de proyección*.

Una *condición* en el universo U es una función $m : U \mapsto \{0, 1\}$, definida a partir de tres elementos π_i, $\oplus$, y a_i, donde π_i es una función de proyección, $\oplus$ es un operador de comparación valido para los elementos del conjunto A_i (por ejemplo, si A_i es un conjunto numérico $\oplus \in \{=, <, >, \leq, \geq, \ldots\}$), y $a_i \in A_i$. Una instancia n satisface la condición m cuando la comparación entre $\pi_i(n)$ y a_i realizada mediante $\oplus$ es verdadera, en ese caso $m(n) = 1$; en caso contrario, $m(n) = 0$.

Una *fórmula* en el universo U también es una función que toma valores $\{0, 1\}$ que se define recursivamente a partir de los siguientes principios:

- Si m es una condición en U entonces $f = m$ una fórmula.
- Si g es una fórmula en U entonces $f = (g)$ es otra fórmula, tal que $f(n) = g(n)$ para todo $n \in U$.
- Dada dos fórmulas en U, g_1 y g_2, se tiene que:
  - $f = g_1 \wedge g_2$ es una fórmula, donde $f(n) = g_1(n) \wedge g_2(n)$ para todo $n \in U$.
  - $f = g_1 \vee g_2$ es una fórmula, donde $f(n) = g_1(n) \vee g_2(n)$ para todo $n \in U$.

Una *regla simple* r en el universo U es representada por un par ordenado $\langle \overleftarrow{r}, \overrightarrow{r} \rangle$ donde $\overleftarrow{r}$ y $\overrightarrow{r}$ son fórmulas en U. Decimos que r se cumple en el conjunto $D \subseteq U$ si para todo $n \in D$ se tiene que $\overleftarrow{r}(n) = 1 \implies \overrightarrow{r}(n) = 1$. Decimos r cubre una instancia $n \in U$ si $\overleftarrow{r}(n) = 1$.

2.2. Reglas para clasificación

Sea un universo $U = A_1 \times A_2 \times \ldots \times A_k$ y D un subconjunto de U, $D \in U$, que contiene instancias que deben ser clasificadas en un *conjunto de clases* $C = \{c_1, c_2, \ldots, c_l\}$, siendo l un número entero $l \geq 2$.

Una *regla de clasificación* es simplemente una regla en el universo $U \times A_{k+1}$, representada por $\langle \overleftarrow{r}, \overrightarrow{r} \rangle$, donde $A_{k+1} = C$, $\overleftarrow{r}$ es una fórmula en U y $\overrightarrow{r}$ es una condición de igualdad (usando el operador "=") definida sobre la componente correspondiente a las clases (A_{k+1}). Por simplicidad, una regla de clasificación suele representarse como $\langle \overleftarrow{r}, c_i \rangle$, siendo $c_i \in C$. El significado intuitivo de una regla de clasificación consiste en que el cumplimiento de la premisa $\overleftarrow{r}$ en una instancia implica que dicha instancia pertenece a la clase c_i.

Supongamos que existe $\delta : D \mapsto C$ una función que a cada instancia de D le asigna una clase en C. El *conjunto de entrenamiento* asociado a δ, se define como $D_t = \{(a_1, \ldots, a_k, a_{k+1}) | n = (a_1, \ldots, a_k) \in D \wedge \delta(n) = a_{k+1}\}$.

2.3. Reglas difusas

Las reglas difusas analizadas en este reporte constituyen un tipo especial de reglas de classificación $\langle \overleftarrow{r}, c_i \rangle$, donde la premisa $\overleftarrow{r}$ es una fórmula cuyas condiciones difieren un tanto de las presentadas en la sección 2.1.

Una *condición difusa* en el universo U es una función $m : U \mapsto [0, 1]$, definida a partir de tres elementos π_i, λ, y K_i, donde π_i es una función de proyección, λ es es una función de pertenencia en conjuntos difusos, y $K_i \subseteq A_i$. El valor de m dada una instancia n coincide con el valor de $\lambda(\pi_i(n), K_i)$ la pertenencia difusa de $\pi_i(n)$ al conjunto K_i.

3. Técnicas para búsqueda de reglas de clasificación

En esta sección se analizarán varias propuestas basadas en búsqueda automática de reglas. Las técnicas analizadas están enfocadas a dar solución en un área determinada. Algunas de ellas pudieran extenderse para su posterior aplicación en otras esferas.

Un requerimiento fundamental de dichas técnicas es que los datos del conjunto de entrenamiento tienen que estar etiquetados, es decir debe estar definida en cada instancia del conjunto de entrenamiento la clase a la cual pertenece. Si el conjunto de datos no cumple con el requerimiento mencionado, no se puede ejecutar el aprendizaje para detectar reglas. A continuación se procede al análisis de las técnicas.

3.1. Algoritmos voraces

Los algoritmos voraces analizados en este reporte (AQ21 [17], X2R [18], RIPPER [19], PART [16]) y Ant-Miner [20] cumplen con el esquema general del algoritmo Búsqueda-Voraz descrito en esta sección (ver algoritmo 1). La entrada de este algoritmo es un conjunto de entrenamiento D_t definido según el marco teórico de la sección 2.2.

Algoritmo 1: Búsqueda-Voraz (D_t)

Entrada: D_t - conjunto de entrenamiento
Salida: R - conjunto de reglas encontradas

```
1  D_t ← Preparar-Entrada (D_t);
2  R ← ∅;
3  while Quedan-Instancias(D_t) do
4  |   N ← Seleccionar-Instancias(D_t);
5  |   R_1 ← Calcular-Reglas(N, D_t);
6  |   R_2 ← Filtrar-Reglas(R_1, D_t);
7  |   R ← R ∪ R_2;
8  |   D_t ← Eliminar-Cubrimientos(D_t, R_2);
9  end
10 R ← Ajustar-Reglas(R);
11 return R;
```

La primera acción consiste en preparar D_t para las acciones posteriores, por medio de una función Preparar-Entrada (ver algoritmo 1, línea 1). El uso de esta función no es obligatoria dentro de un algoritmo voraz ya que puede ser que el conjunto de entrada ya esté listo para ser procesado en los siguientes pasos. No obstante, algunas de los trabajos reportados en la literatura usan esta función para discretizar los datos el conjunto de entrenamiento (X2R) o asignarles el valor inicial a determinadas variables de estado (RIPPER).

Luego en el algoritmo Búsqueda-Voraz, se ejecuta un proceso iterativo mientras queden instancias por procesar, por medio de un criterio de parada definido en la función Quedan-Instancias (ver algoritmo 1, línea 3). Generalmente, dicha función verifica solamente que $D_t \neq \emptyset$. No obstante, existen algoritmos voraces que siguen otras estrategias; por ejemplo: AQ21 verifica si D_t contiene instancias positivas y RIPPER, además de comprobar que $D_t \neq \emptyset$, verifica que la última regla encontrada r no sea muy "complicada" (El grado de complicación de una regla está dado en términos de la longitud total de la descripción [21]. La condición de parada se cumple si la longitud de la descripción de r es al menos b bits mayor que la longitud de la descripción más corta encontrada hasta ese momento. El autor sugiere escoger $b = 64$ para la condición de parada.).

Dentro de cada iteración del proceso antes mencionado, se selecciona un conjunto (N) de instancias que se utilizarán como base para generar las reglas candidatas, por medio de la función Seleccionar-Instancias (ver algoritmo 1, línea 4). El segundo paso dentro de cada iteración consiste en generar un primer conjunto R_1 de reglas candidatas, por medio de la función Calcular-Reglas (ver algoritmo 1, línea 5). El tercer paso dentro de cada iteración consiste en generar un segundo conjunto R_2 de reglas filtradas o mejoradas, por medio de la función Filtrar-Reglas (ver algoritmo 1, línea 6). Luego, las reglas de R_2 son adicionadas al conjunto R que contiene las reglas encontradas hasta el momento (ver algoritmo 1, línea 7). Al final de cada iteración se eliminan de D_t aquellas instancias que han sido cubiertas por las reglas en R_2, por medio de la función Eliminar-Cubrimientos (ver algoritmo 1, línea 8). Esta última función es la misma en todas los algoritmos voraces reportados en la literatura. La tabla 1 contiene un resumen de las estrategias de iteración reportadas, mostrando diferentes maneras de implementar las funciones Seleccionar-Instancias, Calcular-Reglas, y Filtrar-Reglas.

Una vez concluidas todas las iteraciones, el algoritmo Búsqueda-Voraz procede a realizar un último filtrado general de las reglas calculadas, por medio de la función Ajustar-Reglas (ver línea 10). Este último paso tampoco es obligatorio dentro de los algoritmos voraces. No obstante,

Tabla 1. Estado del arte de los algoritmos voraces.

	AQ21	Ant-Miner	PART	X2R	RIPPER
Seleccionar-Instancias	Selecciona una instancia positiva n llamada semilla (en inglés *seed*).	No se usa el conjunto N. Por tanto, es como si $N = \emptyset$.		Selecciona un conjunto con las instancias más frecuentes en D_t.	Selecciona aleatoriamente 2/3 de los elementos en D_t.
Calcular-Reglas	Obtiene un conjunto de reglas (estrella aproximada) para la semilla que no cubren ninguno de los ejemplos negativos.	Calcula un conjunto de reglas empleando un método basado en colonia de hormigas [22] (ACS por sus siglas en inglés).	Calcula el árbol de desición simplificado asociado a D_t, usando el algoritmo C4.5 [23]. Luego, se busca la hoja que más instancias cubre y se determina el camino que conduce desde la raíz hasta dicha hoja. El camino obtenido es representado en forma de regla.	Crea una regla vacía y le adiciona condiciones que son aprendidas utilizando el conjunto (N) hasta que la regla no cubra más instancias de otras clases. Cada condición es adicionada de forma tal que se maximice alguna medida de calidad, definida según el contexto de aplicación.	
Filtrar-Reglas	Selecciona las reglas de acuerdo a una medida de calidad, definida según el contexto de aplicación.	Selecciona solamente la mejor regla descartando las restantes, de acuerdo a una medida de calidad, definida según el contexto de aplicación.	Sin acciones, $R_2 = R_1$.		Reemplaza la regla por una más general, eliminando condiciones de la premisa.

algunos trabajos del estado del arte (AQ21) utilizan medidas, como el funcional de evaluación lexicográfica, para eliminar algunas reglas del conjunto R y otros (X2R y RIPPER) eliminan las reglas redundantes en cada clase y reemplazan las reglas específicas por otras más generales.

3.2. Métodos basados en árboles de decisión

Existe varios algoritmos para obtener árboles de decisión, por ejemplo C4.5 [23], C5.0 [14] y CART [24]. Dichos algoritmos siguen el esquema general del algoritmo Árbol-Decisión que se presenta en esta sección (ver algoritmo 2).

Algoritmo 2: Árbol-Decisión (D_t)

Entrada: D_t - conjunto de entrenamiento
Salida: R - conjunto de reglas encontradas

```
1 R ← ∅;
2 T ← ∅; //conjunto de árboles
3 t ← Generar-Árbol-Decisión(D_t);
4 T ← Podar-Árbol-Decisión(t);
5 R ← Optimizar(T, R);
6 return R;
```

El primer paso del algoritmo 2 consiste en generar un árbol de decisión t (ver algoritmo 2, línea 3). El criterio utilizado para construir el árbol de decisión puede variar en dependencia de la propuesta. Al árbol de decisión generado, se le aplica un proceso de poda y se obtiene una colección de árboles T (ver algoritmo 2, línea 4). Según el método utilizado, la colección puede comprender uno o varios árboles. Finalmente, se lleva a cabo un procedimiento de optimización (ver algoritmo 2, línea 5). Dicho procedimiento, puede ser aplicado sobre T, o sobre el conjunto de reglas R obtenido a partir de un árbol de decisión t_1, tal que $t_1 \in T$. La tabla 2 contiene la estrategia utilizada en cada paso por los algoritmos analizados en esta sección.

Tabla 2. Estado del arte de los métodos basados en árboles de decisión.

	C5.0	CART
Generar árbol de decisión	Para determinar cual condición se debe tener en determinado nodo de decisión utilizan el criterio de la máxima ganancia de información [25].	Para evaluar cual división en un nodo resulta más óptima para particionar el conjunto de entrenamiento, se utiliza el criterio de la ganancia Gini [26].
Poda	El método de poda comienza por el final del árbol y examina cada subárbol que no sea hoja. Si reemplazar el subárbol en análisis por una de sus hojas o por su rama más frecuentemente usada reduce el porcentaje de error estimado, entonces se poda el árbol efectuando el reemplazo.	El valor de costo-complejidad se calcula varias veces durante la poda, como resultado de sustituir cada arista del árbol por el nodo raíz de dicha arista. El menor valor de costo-complejidad obtenido en este proceso va a indicar por donde debe ser podado el árbol. Este proceso se repite recursivamente hasta que el subárbol que se vaya a podar solo tenga el nodo raíz.
Optimización	Se extraen las reglas del árbol de decisión y se optimizan mediante el proceso de generalización.	De la colección de subárboles obtenidos del proceso de poda se selecciona el subárbol más óptimo aplicando validación cruzada. A partir del subárbol seleccionado se extraen las reglas.

En el trabajo de Kuhn y Johnson [14] se presenta un método para obtener reglas mediante el uso de este tipo de árboles. Inicialmente, el árbol de decisión que se obtiene suele ser muy grande, por lo cual es posible que sobreajuste el conjunto de entrenamiento. La solución consiste en podarlo buscando una relación óptima de costo-complejidad que varía de acuerdo al tipo de árbol que se esté utilizando. Esto consiste en encontrar un subárbol (árbol simplificado) con un tamaño adecuado que tenga el menor porcentaje de error. Tanto el árbol inicial como el simplificado pueden ser utilizados como base para la obtención de las reglas.

El proceso para generar reglas tiene lugar a partir del árbol de decisión. Una regla es generada para cada hoja, siguiendo el camino que conduce de la raíz a la hoja.

Las reglas obtenidas pueden contener condiciones irrelevantes, por lo cual deben ser generalizadas eliminando las condiciones irrelevantes sin que esto afecte su presición. Para decidir que condición eliminar supongamos que tenemos la siguiente regla $r_1 = \langle \overleftarrow{r_1}, c \rangle$ y una regla más general $r_2 = \langle \overleftarrow{r_2}, c \rangle$, donde $\overleftarrow{r_2}$ es el resultado de eliminar una condición m del conjunto $\overleftarrow{r_1}$. El criterio para seleccionar m puede variar de acuerdo al contexto de aplicación.

Otro trabajo que usa árboles de decisión para detectar reglas es el algoritmo CART [24]. En este algoritmo, del conjunto de subárboles obtenidos durante la poda se selecciona el más óptimo. Para el proceso de selección se utiliza la técnica de *validación cruzada*. La validación cruzada es un método empleado para validar la construcción de un modelo sin necesidad de utilizar un nuevo

conjunto de datos. Como resultado de aplicar dicho método, se obtiene el costo de clasificación incorrecta para cada árbol. El árbol cuyo valor de costo de clasificación incorrecta sea el mínimo es utilizado para seleccionar el árbol final.

El proceso para seleccionar el árbol final en ocasiones suele ser inestable. Esto se debe a que para algunos conjuntos de datos utilizados, el árbol final seleccionado resulta ser significativamente diferente. La solución está en utilizar la regla de un-error-estándar (en inglés one-standard-error) propuesta por Breiman et al. [24]. Lo que se busca con aplicar esta regla es reducir la inestabilidad al escoger el árbol final. Así como encontrar el subárbol podado más simple que su rendimiento sea comparable con el del subárbol más óptimo. Luego de tener el árbol final se genera el conjunto de reglas. Para ello se toma el camino que conduce a cada nodo terminal como una regla y se agrega al conjunto de estas.

3.3. Método basado en agrupamiento conceptual

El agrupamiento conceptual [27] es una forma de aprendizaje no supervisado, que consiste en agrupar las instancias en grupos que representan conceptos simples. Este tipo de agrupamiento consta de dos procesos claves. El proceso de estructuración es donde se agrupan entidades y se determinan grupos a partir de una colección de objetos. El otro proceso es la caracterización, en la cual se determina el concepto de cada grupo de la estructuración. En el trabajo de Suárez y Pagola [28] se hace un análisis de los principales algoritmos de agrupamiento conceptual, describiendo características importantes que deberían estar presentes en un algoritmo conceptual.

En Michalski et al. [17] se presentan varios experimentos que usan inducción natural basada en el algoritmo AQ21 (ver sección 3.1) y agrupamiento conceptual. En el experimento que es aplicado a la detección de fraudes en impuestos, se combina el agrupamiento conceptual y la inducción natural para buscar reglas.

En la figura 1 se muestra el esquema de la técnica en análisis. En este método los datos asociados a impuestos son agrupados por un algoritmo conceptual. Luego se aplica una técnica de aprendizaje supervisado sobre las instancias fraudulentas de cada grupo $(D_1, D_2, ..., D_i)$. La salida son reglas simples que permiten detectar impuestos regulares y fraudulentos. Las reglas obtenidas se aplican sobre los nuevos datos asociados a impuestos. El resultado son diferentes grupos de contribuyentes y entre ellos el grupo con un mayor porcentaje de fraudes con respecto a los otros.

Esta propuesta está basada en la metodología AQ estrella propuesta por Michalski [29]. Mediante dicha metodología es posible construir descripciones de clases. A los objetos de la clase a la cual se le generan descripciones son denominados positivos y los objetos que pertenecen al resto de las clases se denominan negativos. El conjunto de todas las descripciones maximalmente generales que cubren un objeto o y no cubren los restantes objetos negativos, es definido como estrella del objeto o.

La metodología AQ estrella es posible adaptarla, la familia de algoritmos CLUSTER [30,31,29] la adapta como se describe a continuación. Se seleccionan aleatoriamente k objetos semillas de un conjunto de objetos no clasificados. Los objetos seleccionados son tratados como representativos individuales de k clases imaginarias. Para cada semilla el algoritmo genera descripciones maximales generales que no cubran ninguna otra semilla. A partir de las descripciones generadas son seleccionados como objetos representativos aquellos que satisfacen la descripción de las clases formadas. Los objetos representativos seleccionados pasan a ser las semillas de la próxima itera-

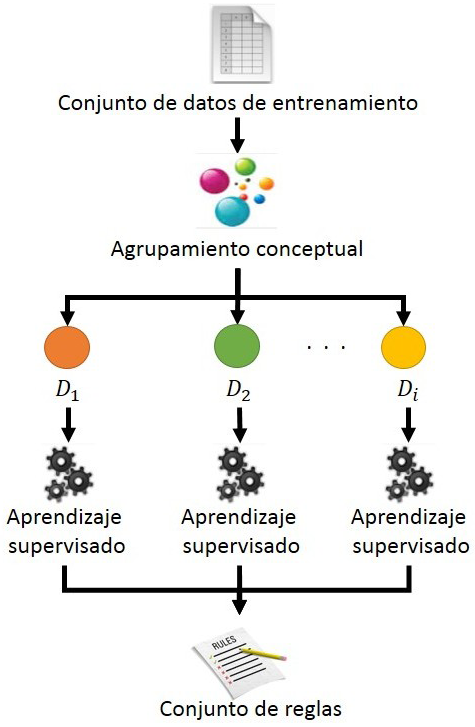

Fig. 1. Agrupamiento y aprendizaje de reglas para la detección de fraudes en impuestos.

ción. Este proceso se repite mientras no se supere un número predefinido de iteraciones donde la clasificación no mejora.

Un conjunto de objetos O puede ser descrito por un conjunto de variables X. Para cada variable $x_i \in X$, se establece un conjunto V_i que contiene todos los valores posibles que puede tomar x_i para cualquier objeto en O. La descripción de un objeto o se puede definir como:

$$o = (x_1(o), x_2(o), ..., x_{|X|}(o)),$$

donde $x_i(o) \in V$, $i = 1, 2, ..., \mid X \mid$. Un selector es definido como $[x_i \oplus V_i']$ donde $\oplus = \{<, \leq, >, \geq, \neq, =\}$ y $V_i' \subseteq V_i$. Luego, un complejo lógico ($l-$complejo [32]) es el producto lógico de selectores. Veamos el siguiente ejemplo donde se muestra cuando un objeto satisface a un $l-$complejo. Dado un objeto $o = (4, 3, 7, 8)$ se dice que este satisface al $l-$complejo $\iota = [x_1 = \{4, 5\}][x_3 \geq 5][x_4 \in [6, ..., 9]]$. El $l-$complejo ι es la representación simbólica de todos los objetos para los cuales x_1 es 4 o 5, x_3 es mayor o igual que 5, y x_4 está entre 6 y 9. El concepto de $l-$complejo es muy utilizado en técnicas de agrupamiento conceptual [33,34] para caracterizar los grupos que se obtienen a partir de algún criterio de agrupamiento. Dadas las características presentadas de los $l-$complejos, estos pueden ser utilizados como una regla para satisfacer determinada clase.

3.4. Una técnica para obtener reglas difusas

En el trabajo de H*ü*hn y H*ü*llermeier [35] se presenta un algoritmo de generación de reglas difusas (FURIA por sus siglas en inglés). En este trabajo se realiza una adaptación del algoritmo

RIPPER [19], con la cual se calcula un primer conjunto de reglas. Estás últimas son transformadas en reglas difusas, mediante el procesamiento de las condiciones asociadas a atributos numéricos.

Las reglas difusas obtenidas por FURIA cumplen con las definiciones planteadas en la sección 2.3. En este caso, la función de pertenencia λ es definida como una función trapesoidal, mientras que los conjuntos K_i constituyen intervalos numéricos. Además, la premisa en estas reglas se conforma por la conjunción lógica de una o varias condiciones. Por tanto, el cubrimiento de una instancia viene dado por el producto de todos los valores de las condiciones que componen la premisa.

En el trabajo de Döring et al. [36] se utiliza FURIA como parte de una herramienta para generar reglas de negocio, a partir de instancias de flujos de trabajo en registros.

3.5. Conclusiones parciales

La posibilidad de representar árboles de decisión como conjuntos de reglas, es una característica en la que se basan algunos de los algoritmos para descubrir reglas. Podemos ver el caso del algoritmo PART, el cual parte de un árbol obtenido como resultado del algoritmo C4.5 que es una versión anterior de C5.0. PART permite obtener cada regla mediante un árbol simplificado. El algoritmo C5.0 obtiene un conjunto de reglas a partir de un árbol complejo, sobre las cuales ejecuta una optimización de forma global. El algoritmo CART también utiliza árboles de decisión para obtener reglas, para los cuales aplica validación cruzada con el propósito de seleccionar al más óptimo.

El principio de divide y vencerás [37] también es aplicado para generar reglas, tal es el caso del algoritmo RIPPER. Los algoritmos basados en dicho principio generan una regla a la vez y de forma iterativa sigue generando otras reglas para las instancias que van restando que no son cubiertas por ninguna de las reglas generadas.

El tratamiento que se le da al conjunto de entrenamiento antes de realizar el proceso de aprendizaje constituye una característica propia de cada técnica vista. El tratamiento puede consistir desde aplicar un algoritmo de agrupamiento conceptual hasta aplicar un algoritmo de discretización para representar los valores numéricos de los atributos como valores discretos. Este último mediante la discretización permite identificar los atributos irrelevantes que solo aportarían ruido en el aprendizaje.

De forma similar el algoritmo Ant-Miner solo procesa atributos categóricos, es decir, necesita discretizar los atributos continuos. Esta carácterística límita un poco a dichos algoritmos en cuanto a los operadores que pueden emplear en las reglas que generan; obligándolos a solo poder utilizar cuando más los operadores $(=, \neq)$.

Las reglas que se obtienen generalmente definen límites claros para los atributos de los datos a analizar. Los límites claros en ocasiones pueden pasar por alto actividades fraudulentas que estén próximas al límite pero no lo sobrepasen. El algoritmo FURIA presenta una propuesta interesante para dar solución a dicho problema. Convertir reglas ordinarias en reglas difusas, permite tener un límite más flexible. De esta manera, el algoritmo alerta cuando existe la posibilidad de estar en presencia de una actividad fraudulenta aunque esta no haya sobrepasado el límite predefinido.

4. Descubrimiento de reglas en procesos de negocio

La gestión de procesos de negocios permite atender grandes cantidades de instancias de procesos, como pueden ser compras en línea, solicitudes de préstamos, actividades generadas por aplicaciones de negocios, entre otras. Con la aplicación de reglas de negocio es posible detectar riesgos para el negocio, como transacciones fraudulentas o tendencias que puedan eventualmente provocar daños. Las reglas de negocios son fáciles de entender por los seres humanos y definen o restringen algunos aspectos del negocio.

También pueden estudiarse algunos trabajos basados en descubrimiento de reglas de negocio con el fin de determinar si pueden ser extendidos a otras áreas. Por ejemplo, la propuesta de Rosso-Pelayo et al. [38] consiste en una técnica para realizar minería de procesos de negocio (BPM por sus siglas en inglés) usando datos no estructurados en lugar de registros. El enfoque se divide en dos partes. En la primera parte se obtiene la asociación entre procesos de negocio y los datos no estructurados [39]. El objetivo de la segunda parte consiste en identificar reglas para procesar las actividades presentes en los documentos.

En este trabajo se demuestra que es factible encontrar evidencias de procesos incompletos en reportes no estructurados de los procesos de negocios conocidos; aplicando detección de reglas asociadas a procesos de negocios [40] mediante la ontología de dominio [41] específico. Por la característica que tienen los datos que se procesan, los cuales son no estructurados y escritos en lenguaje natural; pudiera hacerse un estudio más profundo para determinar si es posible extender dicho enfoque a la detección de fraudes mediante SMS. Hay disponible otro trabajo muy relacionado a este tipo de método aplicado en la industria bancaria, específicamente en textos que describen servicios o productos financieros [42].

Existen varias herramientas que permiten gestionar reglas de negocios [43,44]. Además en la literatura se pueden encontrar muchas investigaciones enfocadas a sistemas para el modelado de reglas de negocios [45,46,47,48], así como otras enfocadas a la extracción de reglas de negocios en códigos de sistemas [49,50,51], pero pocas orientadas al descubrimiento de dichas reglas. En esta sección se analizarán técnicas empleadas para el descubrimiento de reglas de negocios.

4.1. Actualización de reglas de negocio

La propuesta de Francia y Moreno [52] se enfoca en la detección de tendencias en procesos de negocios altamente automatizados. Una tendencia es considerada como una sucesión de instancias de procesos similares que presentan características en común. La adaptación automática del proceso de negocio se pretende realizar mediante la modificación de las reglas, sin requerir intervención humana.

Para comprender mejor como se evidencian las tendencias supongamos que una organización solo puede otorgar una determinada cantidad de dinero por día, contando con filtros para las solicitudes que excedan cierto monto. De ocurrir una rápida secuencia de solicitudes, donde la mayoría estuviera por debajo del monto, pero que en total excedan el dinero disponible para préstamos, se estaría frente a un riesgo de negocio. La detección oportuna de dicha tendencia podría ayudar a tomar medidas para evitar daños en el negocio.

El modelo propuesto consta de tres capas: capa de procesos de negocios, capa de reglas de negocios y una capa de monitoreo de eventos de negocio. La capa de procesos de negocios contiene las definiciones de procesos, que se componen por nodos que representan actividades y por nodos que implican decisiones, los cuales influyen en el camino a seguir en el flujo. A los nodos de

decisión se asocian las reglas de negocios definidas por analistas en la capa de reglas de negocios. En caso de ser detectada una tendencia pueden ser modificadas automáticamente, de forma tal que se alerte sobre la situación actual en los procesos del negocio. La capa de monitoreo de eventos está destinada a detectar los eventos de interés que ocurren en los procesos de negocios y dispara nuevas tareas como respuesta a los eventos detectados.

Los analistas del negocio son los encargados de determinar los umbrales de detección de tendencias y los umbrales de tolerancia para reglas y procesos de negocio. Los analistas también son responsables de definir las variables que se utilizarán en el proceso de negocio. Dichas variables son clasificadas como críticas y no críticas. Las tendencias que contengan variables centrales críticas, son para las cuales el sistema cambiará automáticamente los umbrales en el momento en que son detectadas. En caso de que la tendencia detectada no tenga variables centrales críticas se le notifica a los analistas para que estos sean los encargados de tomar una desición sobre la modificación de los umbrales excedidos.

Para detectar el surgimiento de nuevas tendencias, el analista define un contador de eventos de interés $\bar{e}$ y una ventana de tiempo W. El valor de $\bar{e}$ va a determinar el número de eventos con variables centrales similares, que deben ocurrir, para que una secuencia de eventos sea considerada como una tendencia. Por otra parte, la variable W representa el período de tiempo en el cual las tendencias potenciales serán analizadas, luego de ese período, el valor de $\bar{e}$ será reseteado a cero. De esta forma, una tendencia potencial que no alcanza el umbral establecido en $\bar{e}$ dentro de la ventana de tiempo W, no será considerada como una tendencia real.

Cuando no se cumple la condición de los nodos de decisión se ejecuta un método para determinar si se debe realizar o no un análisis de tendencia. Esto se realiza comprobando si el valor de la variable en análisis en el nodo de decisión, se sitúa dentro del rango de tolerancia definido para ese umbral (ver algoritmo 3, líneas 1-8). Si esto ocurre, el sistema ejecuta el análisis de tendencia.

Algoritmo 3: Decisión_Analizar_Tendencia (u,$\tilde{u}$,$\bar{v}$)

Entrada: u - valor del umbral de la regla de negocio correspondiente al nodo de decisión, $\tilde{u}$ - valor de tolerancia asociado al umbral, $\bar{v}$ - valor de la variable involucrada.

```
1 ŭ = u − u · ũ //Calcular umbral inferior
2 if (v̄ < u) and (v̄ >= ŭ) then
3 |   return true
4 end
5 û = u + u · ũ //Calcular umbral superior
6 if (v̄ > u) and (v̄ <= û) then
7 |   return true
8 end
9 return false
```

Cuando se procede al análisis de tendencias, si $\bar{e}$ alcanza el valor del umbral de detección correspondiente, se efectúan las notificaciones sobre la detección de la tendencia y se toman acciones correspondientes. Para ello primero se buscan las fechas posibles según el valor de la ventana (ver algoritmo 4, línea 1). Luego para cada una de las fechas del arreglo, se cuenta la cantidad de instancias del proceso de negocio relacionadas a la tendencia en análisis, que se encuentran almacenadas en una base de datos (ver algoritmo 4, líneas 3-5). Si la cantidad de instancias obtenidas es igual al umbral de detección significa que se está en presencia de una tendencia (ver algoritmo 4, líneas 6-10).

Algoritmo 4: Analizar_Tendencia ($\bar{e}$,W)

Entrada: $\bar{e}$ - umbral de detección correspondiente, W - ventana de tiempo.

```
1  totalInstancias ← 0
2  arregloFechas[] ← posibles fechas según W
3  for i = 0 to long(arregloFechas[]) do
4  |  totalInstancias ← totalInstancias + arregloFechas[i].cantidadInstancias
5  end
6  if totalInstancias == ē then
7  |  return tendenciaDetectada ← true
8  else
9  |  return tendenciaDetectada ← false
10 end
```

En dependencia del resultado obtenido, en caso de detectarse una tendencia se procede a analizar sus variables centrales. En caso de que la tendencia contenga variables centrales críticas, el sistema automáticamente modificará los umbrales y parámetros de la regla involucrada, según un porcentaje de modificación automático predefinido por el analista (ver algoritmo 5, línea 5) y notificará a los mismos de los cambios realizados (ver algoritmo 5, líneas 4-6). En caso de no presentar variables centrales críticas el sistema muestra el porcentaje de modificación de umbrales dentro de un rango válido al analista (ver algoritmo 5, líneas 7-13). Luego el umbral afectado es modificado según el porcentaje que ingrese el analista.

Algoritmo 5: Analizar_Variables_Tendencia (τ)

Entrada: τ - tendencia detectada.

```
1  u ← Obtener valor actual del umbral de la regla de negocio en cuestión
2  z_u ← Obtener porcentaje de modificación automático asociado al umbral
3  ẑ, ž ← Obtener rango de modificación
   //Porcentaje máximo y mínimo permitido de modificación
4  if τ presenta variables críticas then
5  |  u+ = (z_u/100) · u
6  |  Notificación
7  else
8  |  Solicitar ingreso de porcentaje de modificación z especificando rangos (ẑ, ž)
9  |  if z >= ž and z <= ẑ then
10 |  |  u+ = (z/100) · u
11 |  else
12 |  |  Mensaje de error: El valor ingresado no esta en el rango válido
13 |  end
14 end
```

Como se muestra en la figura 2 el analista define los procesos y las reglas de negocios. A partir de las reglas definidas, mediante el análisis de las tendencias se procede a detectar los eventos de interés para los procesos de negocios. Con el propósito de registrar lo ocurrido se almacenan las tendencias reales detectadas y las tendencias potenciales en la base de datos.

La modificación de los valores en las reglas presentes en las transiciones entre actividades en los procesos conllevan a una modificación en el proceso mismo. Esto se debe a que al cambiar los

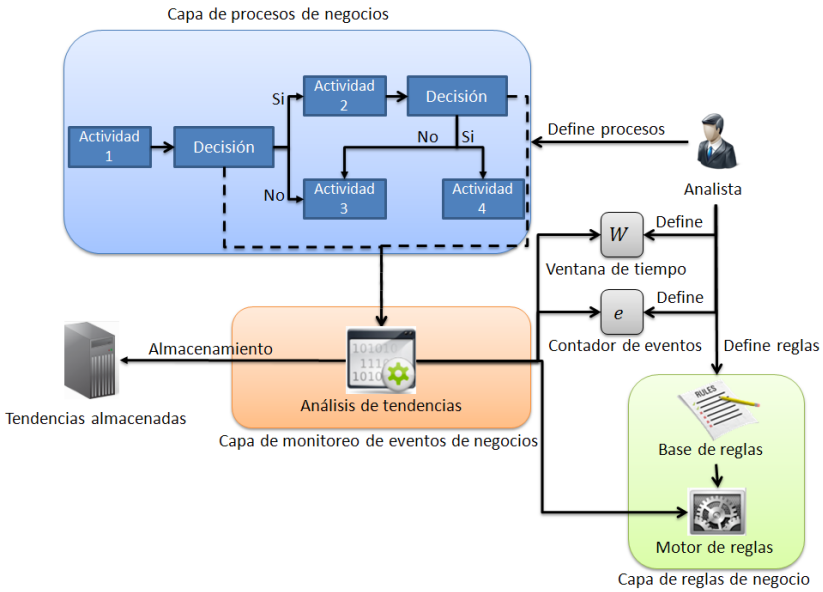

Fig. 2. Esquema de detección de tendencias.

valores de las reglas asociadas a los nodos de decisión, se modifican los caminos a tomar dentro del flujo del proceso, dependiendo de los nuevos valores.

4.2. Extracción de reglas de negocio utilizando agrupamiento multi-agente

La propuesta de Mikanov et al. [53] consiste en un motor multi-agente, el cual realiza agrupamiento de datos y generación de reglas para ser aplicadas en logística de transporte. Los datos se encuentran guardados en una tabla donde cada registro representa una orden de transportación. Las ordenes contienen información como el origen y el destino de la transportación, así como información sobre la carga. Cada información representa un atributo del registro. Los atributos pueden ser de diferentes tipos (enteros, reales, cadenas de caracteres, etc).

Los agentes de registros y los grupos constituyen los principales elementos del método. Un agente es un programa capaz de analizar una situación, tomar decisiones, ejecutar acciones y comunicarse con otros agentes. Dentro del agrupamiento, los agentes forman comunidades virtuales similares a jerarquías temporales, que pueden ser organizadas de distintas formas.

Los agentes de registros pueden emplear métricas como la densidad máxima del grupo, la cantidad de registros, cantidad de sub-dimensiones para un grupo, tiempo de vida en el grupo, entre otras. Dichas métricas se tienen en cuenta para asignar un registro a un grupo.

El proceso de agrupamiento [54] comienza cuando llega un registro nuevo al cual se le asigna un agente. El agente del registro busca los grupos más representativos mediante la fórmula de valoración de grupos, la cual en este caso viene dada por la densidad del grupo. El agente del registro envía una solicitud a los agentes de los grupos seleccionados para pasar a ser miembro de alguno.

Los agentes de grupos que reciben la solicitud realizan una evaluación utilizando la fórmula de valoración de registros, donde si la densidad del grupo se ve maximizada, entonces se acepta la

solicitud. Los agentes de grupos que aceptan la solicitud envían al agente de registro la aceptación y este escoge al más apropiado de los grupos que lo aceptan para unirse. En caso de que ningún agente de grupo acepte la solicitud, el agente de registro intenta formar un nuevo grupo. Para ello envía una propuesta de formación de grupo a los agentes de otros registros que pueden o no formar parte de grupos.

Los agentes de registro a los cuales se les envió la propuesta, solo aceptan si la formación de un nuevo grupo aumenta el valor del sistema. El principal objetivo de la asignación de elementos a los grupos es maximizar el valor del sistema, el cual constituye el valor del proceso general de agrupamiento. Al aceptar la propuesta, los agentes reorganizan todo el sistema. Las relaciones existentes entre los registros y sus grupos se eliminan y se crean nuevas relaciones entre distintos registros incrementando el valor del sistema. Dicho proceso es conocido como auto-organización.

Los agentes de los nuevos grupos creados y los de aquellos grupos que sufrieron modificaciones en sus propiedades (límites, valores, cantidad de registros) como resultado de la auto-organización, repiten el proceso descrito anteriormente para los agentes de los registros seleccionados. El proceso de agrupamiento continua hasta que todos los registros estén relacionados a grupos y que ningún cambio en los grupos pueda incrementar el valor del sistema, o que expire el tiempo disponible para el agrupamiento. Con la constante llegada de nuevos registros al sistema se tiene en cuenta la fecha, lo cual permite eliminar los registros fuera de fecha para futuros agrupamientos. Este proceso es conocido como evolución.

Para comprender mejor el proceso de agrupamiento analicemos el siguiente ejemplo. Supongamos que se tienen cuatro instancias (ver figura 3) de registros que llegan al sistema una tras otra.

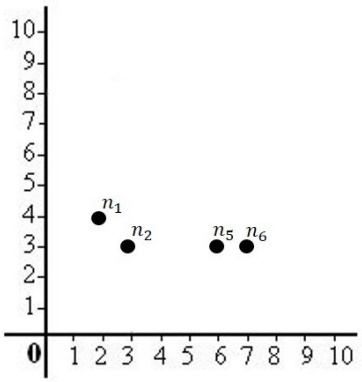

Fig. 3. Instancias disponibles para agrupamiento.

Al llegar la instancia n_1 y seguidamente la instancia n_2, se forma un grupo D_2 (ver figura 4).

Al llegar la instancia n_5 envía una solicitud al grupo D_2, pero es rechazada porque si pasa a integrar el grupo se reduce la densidad de este. Entonces la instancia n_5 envía una solicitud a D_2 para crear un nuevo grupo formado por n_5 y D_2, el agente del grupo acepta y se forma el grupo D_3 (ver figura 5).

Cuando llega la instancia n_6 envía una propuesta a n_5 para que deje de formar parte de D_3 y se una a n_6 en un nuevo grupo. La instancia n_5 acepta ya que el nuevo grupo D_4 tendrá mayor densidad que el grupo D_3. Por tanto el grupo D_3 es eliminado y se forma el grupo D_4 con las instancias n_5 y n_6 (ver figura 6).

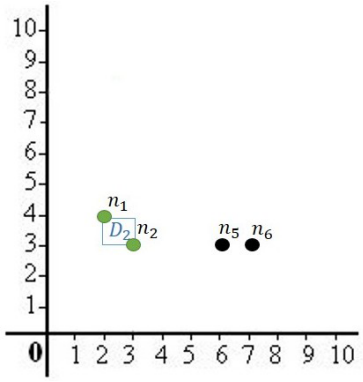

Fig. 4. Las instancias n_1 y n_2 forman el grupo D_2.

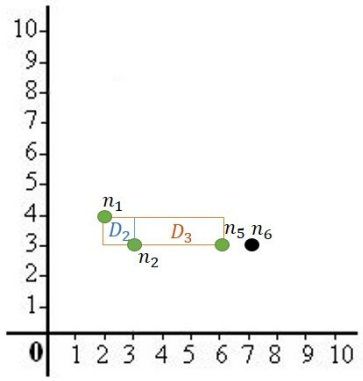

Fig. 5. La instancia n_5 y el grupo D_2 forman el grupo D_3.

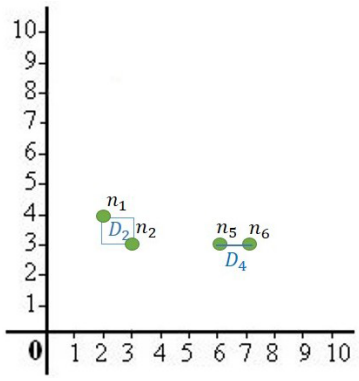

Fig. 6. El grupo D_3 es eliminado y se forma el grupo D_4.

El grupo D_4 le envía una propuesta al grupo D_2 para formar juntos un nuevo grupo. El grupo D_2 acepta y se forma el grupo D_5 (ver figura 7). El grupo D_5 no tiene posibilidades de agrupamiento disponibles porque todas las instancias y los grupos han alcanzado sus pertenencias óptimas y el proceso de agrupamiento concluye.

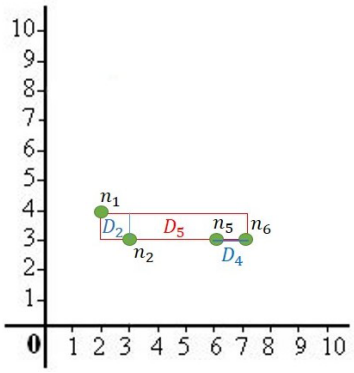

Fig. 7. El grupo D_2 y D_4 forman el grupo D_5.

Luego de obtener los grupos se hace necesario representarlos de forma tal que se puedan interpretar fácilmente por analistas o por sistemas automatizados, la solución está en las reglas. Para generar reglas a partir de grupos se forman dos grupos de características. El primero representa los atributos de la parte antecedente (SI) y el segundo la parte consecuente ($ENTONCES$). Los autores incluyen en un grupo los atributos sobre los cuales no se tiene control (información de la orden de transportación) y en el otro grupo aquellos atributos que se pueden manipular (tipo de camión para transportar, compañía, entre otros). Un ejemplo de una regla sería:

- "SI la carga pesa 5kg, $ENTONCES$ el camión debe ser un ZIL y la compañía Trans-ZIL".

Las dependencias de los atributos de la parte consecuente sobre los atributos de la parte antecedente (patrones) son determinados en el agrupamiento. Las reglas son evaluadas siguiendo tres criterios:

- Representatividad: muestra la cantidad de elementos del grupo que se incluyen en la regla. Este parámetro no depende de los patrones.
- Confianza: representa la cantidad de elementos de un grupo que se incluyen en la parte antecedente de la regla y no cumplen las condiciones de la parte consecuente. Este parámetro depende del patrón definido para el grupo. Por ejemplo, el patrón "todas las que están embarazadas son mujeres" tiene nivel de confianza de 100 %, mientras la regla inversa tiene bajo nivel de confianza ya que "no todas las mujeres están embarazadas".
- Integridad: representa la cantidad de elementos de un grupo que se incluyen en la parte consecuente de la regla y no cumplen las condiciones de la parte antecedente. Por ejemplo, la regla "SI eres un ser humano $ENTONCES$ eres mortal" tiene alto nivel de representatividad, pero bajo nivel de integridad ya que "no todos los mortales son seres humanos".

Mientras mayor sea el valor del nivel de representatividad y confianza, mayor será el valor de interdependencia para el analista. Cuando una regla es detectada se intenta mover algunas condiciones de la parte antecedente a la parte consecuente. Si el movimiento de condiciones no reduce el nivel de confianza, entonces la regla modificada es de mayor utilidad, ya que tiene menos condiciones en la parte antecedente. Por ejemplo la regla:

- "*SI* el origen de la carga es la ciudad Krasnoyarsk, *ENTONCES* el destino es la ciudad de Moscú y el camión debe ser un ZIL",

es de mayor preferencia que la regla:

- "*SI* el origen de la carga es la ciudad Krasnoyarsk y el destino es la ciudad de Moscú, *ENTONCES* el camión debe ser un ZIL",

si el nivel de confianza después de transformada la regla no se reduce. Cualquier regla con un alto nivel de confianza representa un grupo. Por tanto si un algoritmo de agrupamiento encuentra todos los grupos densos, entonces encuentra todas las reglas con un nivel alto de confianza.

4.3. Descubrimiento de reglas de negocios mediante minería de procesos

La propuesta de Crerie et al. [55] representa un método para descubrir reglas de negocios a partir de registros de eventos de sistemas de información. La solución se centra en dos subtipos de reglas de negocios [56]: las de subtipo condición y las de subtipo autorización. Las de subtipo condición sirven para restringir la existencia de algunas reglas de negocios según la aplicación de otras o según condiciones pre-establecidas. Un ejemplo de reglas de negocios de subtipo condición es:

- "la inscripción de un estudiante se realiza *SI* el estudiante tiene documentos; y el estudiante pagó la inscripción".

Las reglas de subtipo autorización restringen quien está autorizado para llevar a cabo una acción determinada. Por ejemplo:

- "Solo el profesor puede aprobar al estudiante".
- "Solo los estudiantes inscritos pueden realizar el examen".

Esta propuesta está implementada sobre la infraestructura de un sistema de minería de procesos (ProM) [57]; el cual es compatible con una amplia variedad de técnicas de minería de procesos implementadas como complementos (*plugins*). La minería de procesos es utilizada en varios estudios [58,59,60,61] para extraer información de registros de eventos, con el propósito de obtener el proceso de negocio a través de su implementación (ver figura 8).

Fig. 8. Esquema del flujo de la minería de procesos.

Las técnicas de minería de procesos asumen la existencia de registros de sistemas de información que almacenan eventos relevantes del negocio. Estos eventos son una buena fuente de información y hacen que sea posible descubrir procesos de negocios de los datos en los registros. Los registros de eventos en el sistema ProM son representados utilizando minería de lenguaje de marcado extensible (MXML por sus siglas en inglés) [62], que permite compartir un formato de entrada común entre diferentes herramientas de minería.

Como se muestra en la figura 9, el proceso general para detectar las reglas parte por los registros de los sistemas de información, que son llevados al formato de MXML. La técnica utilizada para extraer las reglas puede variar en dependencia del subtipo de reglas a descubrir. Las reglas obtenidas son representadas mediante SBVR [63]. La semántica de vocabulario de negocios y reglas de negocios (SBVR por sus siglas en inglés) es un lenguaje para representar reglas de negocios. Este lenguaje proporciona la interoperabilidad de las reglas y vocabularios entre las herramientas de software que se encargan de las reglas de negocio. Un ejemplo de una regla representada mediante SBVR es:

- "**Es obligatorio que** la orden de pago **tenga** un cliente",

donde las palabras en negrita constituyen palabras claves de SBVR. En la tabla 3 se describe el proceso para obtener reglas de cada subtipo.

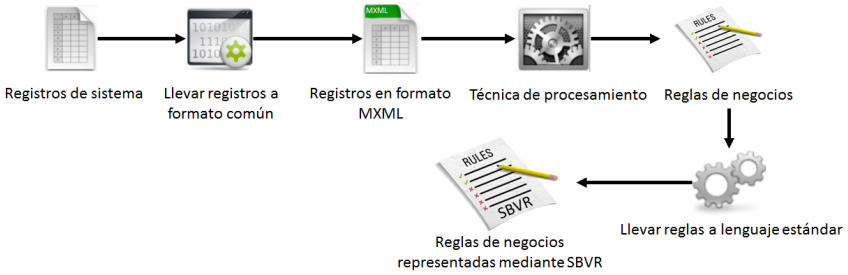

Fig. 9. Esquema de generación de reglas de negocio mediante minería de procesos.

Tabla 3. Descripción de técnicas empleadas para extraer reglas de negocios de distintos subtipos.

Subtipo de regla	Autorización	Condición
Formato común de registros	En ambos casos, los registros del sistema son llevados al formato MXML mediante la herramienta *Mxml ProM Import* [64].	
Técnica de procesamiento	Los registros en formato MXML son procesados y se analizan los datos de cada ejecución. En este proceso son identificadas para cada actividad existente en el registro, cuales categorías de usuarios han ejecutado instancias de dichas actividades. Las reglas se conforman combinando las categorías de usuarios con sus respectivas actividades ejecutadas.	Los registros son llevados al estándar de ARFF [65] para ser procesados por la librería Weka [66] de minería de datos para la ejecución de un algoritmo de clasificación. Los atributos numéricos son descartados, ya que solo son aceptados como atributos de clase para el algoritmo aquellos que son de tipo texto. Las reglas se obtienen mediante el análisis de los árboles de decisión que genera el algoritmo de clasificación.
Lenguaje estándar de las reglas	En ambos casos las reglas son representadas mediante SBVR.	

La técnica utilizada para la extracción de reglas de subtipo autorización, hace posible encontrar reglas como:

- "Solo **el administrador** y **el vendedor** pueden llevar a cabo la actividad **Registrar reclamación**".
- "Solo **el cajero** puede realizar la actividad **Registro de pago**".

La técnica empleada para descubrir reglas de subtipo condición [67] requiere registrar tanta información como sea posible cada vez que se ejecute un evento en el negocio. La información viene dada por los atributos del contexto entre los cuales habrá uno que indicará cuando el evento a sido completado. En la figura 10 se muestra un ejemplo de información contextual de la actividad "Aprobar crédito", donde el atributo "Estado del crédito" indica el estado del evento. Mediante el análisis de los valores de cada atributo en un conjunto de ejecuciones del evento, es posible determinar información como posibles valores de los atributos (ver figura 10).

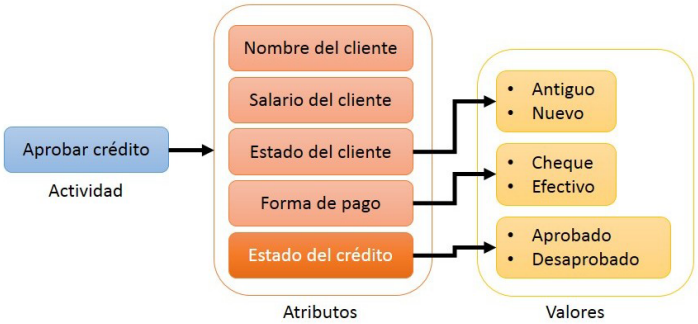

Fig. 10. Ejemplo de atributos de contexto de una actividad y posibles valores.

Cada atributo y su valor representan posibles reglas de negocios que pueden estar restringidas por otros atributos relacionados a su contexto. Analizando los atributos de actividades ejecutadas es posible encontrar reglas como:
"el **Estado de crédito es aprobado** cuando:"

- la **forma de pago es en efectivo** o;
- la **forma de pago es en cheque** y el **estado del cliente es antiguo**.

4.4. Conclusiones parciales

Dado el constante flujo de datos que tiene lugar en los procesos de negocio, constituye una necesidad adaptarse a los cambios en un período de tiempo breve. Es decir, que un algoritmo sea capaz de reaccionar a cambios en los procesos y en consecuencia actualice las reglas. La constante actualización de las reglas contribuye a evitar riesgos de negocios, como pueden ser fraudes, tendencias negativas, entre otros. Además puede contribuir a incrementar la agilidad en la toma de decisiones en situaciones de riesgo.

La idea presentada en la sección 4.1 la cual se centra en la capacidad de auto-adaptación, puede aportar una ventaja adicional en caso de utilizarse en un sistema de detección de fraudes basado en reglas. La auto-adaptación que se evidencia en la modificación de los umbrales definidos en las reglas, proporcionaría robustes al sistema. Por ejemplo, en caso de que la participación de un analista ante un determinado fraude que se este cometiendo no sea posible o no sea lo suficientemente rápida, la modificación automática de los umbrales de las reglas impediría la continuidad del fraude.

Las reglas que se obtienen en la sección 4.3 de subtipo condición, son reglas de negocios que definen el correcto funcionamiento de determinado negocio. Es decir, una actividad del negocio η puede ejecutarse si se cumple una premisa $\overleftarrow{r}$, lo cual puede escribirse en la forma η *si* $\overleftarrow{r}$. Extendiendo esta propuesta sería posible obtener reglas en la forma η *y no* $\overleftarrow{r}$, indicando la ocurrencia

de una actividad del negocio η sin haberse cumplido la premisa $\overleftarrow{r}$. Estas reglas pudieran sugerir la ocurrencia de un evento que se sale del negocio, el cual pudiera representar un fraude o algún tipo de anomalía.

Resulta posible extender una técnica de descubrimiento de reglas de negocios para su posterior aplicación en la detección de fraudes. Pero habría que hacer un análisis más profundo para determinar en que áreas aplicarlo y que modificaciones hacer para que se adecúe a esta. ¿Qué pasaría si aplicamos el enfoque de la sección 4.1 en la detección de fraudes en telecomunicaciones y se pasaran por alto los días festivos o promociones especiales? La respuesta sería una alteración negativa en las reglas, si tenemos en cuenta que en esos días por lo general se sobrepasan los umbrales definidos en las reglas. El sistema asumiría que está en presencia de fraudes y modificaría los umbrales, lo cual pudiera afectar la detección de fraudes reales.

5. Procesamiento de eventos complejos

El procesamiento de eventos complejos (CEP por sus siglas en inglés) consiste en procesar eventos en tiempo real, para detectar situaciones de interés. Un evento es algo que ocurre o cambia el estado actual de los negocios. Por lo cual puede representar un problema, una oportunidad, un umbral, una desviación, una información disponible, etc.

Existen algunos enfoques de CEP con representación basada en reglas [68,69,70]. El lenguaje basado en reglas es más conveniente para representar diversos patrones de eventos complejos. Su uso permite realizar un análisis sobre los eventos, sus relaciones y el posible conocimiento contextual disponible para un dominio particular.

El propósito de un sistema CEP es detectar eventos complejos a partir de la entrada de eventos atómicos. Los eventos atómicos son instantáneos, tienen lugar en un punto específico en el tiempo y su duración es cero. El sistema debe notificar si ha sido detectado algún evento complejo, tan pronto como detecte que un evento atómico completa una secuencia que constituye un evento complejo dada la definición que se tenga del evento complejo. En la mayoría de los lenguajes de CEP, los eventos complejos son definidos por medio de patrones de eventos atómicos. Por tanto, cuando se detecta un patrón el motor CEP infiere que ha ocurrido un evento complejo y lo notifica.

En un sistema CEP, el procesamiento se basa en las reglas definidas por el analista. En dichas reglas se especifica la relación entre los eventos observados y el fenómeno a ser detectado. Los sistemas CEP pueden ser aplicados en varios dominios como redes sensoriales de vigilancia medioambiental [71], análisis de pagos para la detección de fraudes [72], descubrimiento de tendencias en finanzas [73], entre otros. A continuación haremos un análisis de algunas propuestas enfocadas en la generación automática de reglas.

5.1. Generación automática de reglas a partir de datos históricos

En la propuesta de Margara et al. [74] presentan un sistema CEP para generar reglas, mediante el aprendizaje de relaciones causales entre los eventos atómicos y complejos utilizando trazas históricas. Los autores caracterizan cada notificación de evento por un tipo y un conjunto de atributos. El tipo de evento define el número, orden, nombres y tipos de atributos que forman el evento. Como los eventos ocurren instantáneamente, cada notificación incluye una estampilla de tiempo

que representa el momento de la ocurrencia del evento. Por ejemplo, la siguiente notificación:

temp@10 $(apto = 123, valor = 24{,}5)$,

representa el hecho de que la temperatura del aire medida en el apartamento 123, en el instante de tiempo 10, fue de 24.5 grados. Los eventos complejos son definidos a partir de patrones de eventos atómicos. Por ejemplo, el evento complejo $Fuego$ puede ser derivado de la presencia de humo y alta temperatura. Por tanto, el patrón del evento complejo $Fuego$ se puede definir como:

$$W = 5\ \{\text{humo}(area = \wp) \text{ and } \text{temp}(area = \wp \text{ and } valor > 50)\}\ where\ \{temp \rightarrow humo\}.$$

El patrón definido acepta notificaciones de eventos cuyos valores de temperatura excedan los 50 grados; cuenta con una ventana de tiempo W de 5 minutos para encontrar fuego y humo; usa el parámetro $\wp$ como valor común para el atributo $area$ y mediante el operador de secuencia $\rightarrow$ se señala que la notificación de temperatura debe estar precedida por una de humo.

Los autores tienen en cuenta una consideración para llevar a cabo la generación automática de reglas, y es que tanto la regla r_1 como una traza de evento $\overrightarrow{e_1}$ deben estar asociados a un conjunto de restricciones. Es decir, restricciones definidas por las reglas y satisfechas por las trazas. Por ejemplo, consideremos la regla r_1 caracterizada por el patrón:

$$W = 5\ \{e_1(), e_2()\},$$

dicho patrón define el conjunto de restricciones $\Re_{r_1}$:

- e_1: un evento de tipo e_1 debe ocurrir;
- e_2: un evento de tipo e_2 debe ocurrir;

de forma similar la traza de evento $\overrightarrow{e_1} : e_1@0, e_2@2, e_3@3$ satisface el conjunto de restricciones $\Re_{\overrightarrow{e_1}}$:

- e_1: un evento de tipo e_1 debe ocurrir;
- e_2: un evento de tipo e_2 debe ocurrir;
- e_3: un evento de tipo e_3 debe ocurrir;
- $e_1 \rightarrow e_2$: un evento de tipo e_1 debe ocurrir antes de un evento de tipo e_2;
- $e_1 \rightarrow e_3$: un evento de tipo e_1 debe ocurrir antes de un evento de tipo e_3;
- $e_2 \rightarrow e_3$: un evento de tipo e_2 debe ocurrir antes de un evento de tipo e_3.

De esta forma para cada regla r y cada traza de evento $\overrightarrow{e}$, r se dispara si y solo si $\Re_r \subseteq \Re_{\overrightarrow{e}}$. Para realizar el aprendizaje, parten de un conjunto $\overrightarrow{E}$ de trazas de eventos, en el cual están etiquetadas aquellas trazas que representan un evento complejo, llamadas trazas positivas $\overrightarrow{e}^+$. Como vimos anteriormente $\Re_{\overrightarrow{e_1}}$ contiene todas las restricciones definidas por la regla r_1 (e_1, e_2), así como otras cuatro restricciones $(e_3, e_1 \rightarrow e_2, e_1 \rightarrow e_3, e_2 \rightarrow e_3)$ que no satisfacen r_1.

Con el propósito de eliminar aquellas restricciones que no aportan información útil para generar una regla, realizan la intersección entre toda $\overrightarrow{e}^+ \in \overrightarrow{E}$.

Por ejemplo, supongamos que existe una traza $\overrightarrow{e_2} : e_1@0, e_2@3, e_4@4$ que satisface el siguiente conjunto de restricciones $\Re_{\overrightarrow{e_2}}$:

- e_1: un evento de tipo e_1 debe ocurrir;

- e_2: un evento de tipo e_2 debe ocurrir;
- e_4: un evento de tipo e_4 debe ocurrir;
- $e_1 \rightarrow e_2$: un evento de tipo e_1 debe ocurrir antes de un evento de tipo e_2;
- $e_1 \rightarrow e_4$: un evento de tipo e_1 debe ocurrir antes de un evento de tipo e_4;
- $e_2 \rightarrow e_4$: un evento de tipo e_2 debe ocurrir antes de un evento de tipo e_4;

intersectando $\Re_{\overrightarrow{e_1}}$ con $\Re_{\overrightarrow{e_2}}$ se obtiene una aproximación más precisa a $\Re_{r_1}$, la cual solo contiene tres restricciones $(e_1, e_2, e_1 \rightarrow e_2)$.

El sistema está compuesto por siete módulos(ver figura 11), en los cuales se aplica la consideración analizada, al igual que otras como determinar el tamaño de la ventana de tiempo y considerar la presencia de negaciones y parámetros. A continuación se describe cada uno de los módulos.

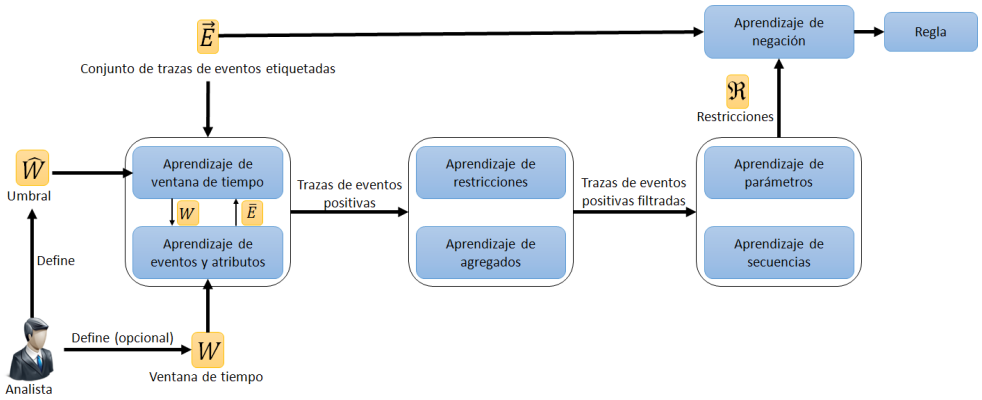

Fig. 11. Esquema de generación de reglas en CEP mediante datos históricos.

El módulo de aprendizaje de eventos y atributos determina cuales tipos de atributos atómicos son requeridos para que ocurra el evento complejo. La ventana de tiempo W es considerada como parámetro opcional de entrada, la cual puede ser definida por un analista o generada por el módulo de aprendizaje de ventana de tiempo. El módulo de aprendizaje de eventos y atributos teniendo en cuenta W descarta los eventos que ocurren fuera del tiempo definido. Para cada traza de evento positiva se extrae el conjunto de tipos de eventos que contiene y se procede a realizar la intersección entre los conjuntos obtenidos como vimos anteriormente.

El módulo de aprendizaje de ventana de tiempo es el responsable del aprendizaje del tamaño de la ventana que incluye todos los eventos atómicos necesarios para que ocurra un evento complejo. Cuando es conocido el conjunto de tipos de eventos atómicos $\bar{E}$, el algoritmo analiza la ocurrencia pasada de eventos complejos. Luego selecciona la menor ventana W, en la cual cada ocurrencia de evento complejo contenga todos los elementos de $\bar{E}$.

En caso de que no sea conocido $\bar{E}$, el algoritmo iterativamente invoca al módulo de aprendizaje de eventos y atributos incrementando el tamaño de W. Cuando W sobrepasa un umbral $\hat{W}$ definido por el analista y no se observe un incremento de los tipos de eventos relevantes se selecciona ese tamaño de ventana.

El módulo de aprendizaje de restricciones, recibe las trazas de eventos positivas después de que son filtradas por los módulos descritos anteriormente. El objetivo de este módulo es identificar el conjunto de restricciones que seleccionan los eventos atómicos relevantes basándose en el valor de sus atributos.

Las restricciones de igualdad representan el caso más elemental, en ellas se impone una igualdad entre un atributo a y una constante. Para detectar este tipo de restricciones, el algoritmo busca cuales valores constantes están asociados al atributo a en todas las trazas positivas. Lo cual se ejecuta extrayendo el conjunto de valores de a de todas las trazas positivas y calculando su intersección.

Para el caso de los atributos numéricos se tienen en cuenta restricciones que soportan relaciones de desigualdad, utilizando signos como $\ominus = \{\leq, \geq, \neq\}$. Como primer paso el algoritmo busca restricciones de igualdad, intersectando los valores del atributo a. El segundo paso consiste en buscar las restricciones desiguales. Para ello extraen el mínimo a^- y el máximo a^+ de los valores de a y se construye una restricción en la forma $a^- \leq a \leq a^+$.

Además el analista puede definir el tipo de relación que desea que aprenda el algoritmo, seleccionando uno de los signos del conjunto $\ominus$. El algoritmo también puede trabajar con múltiples atributos de eventos; basándose en maquinas de soporte vectorial [75] (SVM por sus siglas en inglés), para aprender restricciones que mejor describan los atributos observados en las trazas positivas.

El módulo de aprendizaje de agregados funciona en paralelo con el de aprendizaje de restricciones. Ambos módulos generan restricciones, pero el de aprendizaje de agregados se basa en valores calculado por funciones sobre los eventos de un mismo tipo. Las funciones pueden ser de suma, conteo, mínimo, máximo y promedio. De esta forma se pueden obtener restricciones como $Prom(temp.valor) > 50$, la cual indica que el valor promedio calculado de todos los eventos de tipo *temp* sean mayores que 50.

Luego de calcular el valor de las funciones agregadas sobre todas las trazas positivas, se prosigue con el mismo algoritmo descrito en el módulo de aprendizaje de restricciones. Primero se extraen las posibles restricciones agregadas con relaciones de igualdad. En caso de no detectar ninguna, procede a considerar aquellas restricciones con relaciones de desigualdad.

El módulo de aprendizaje de parámetros recibe como entrada las trazas positivas que coinciden con las restricciones identificadas en el módulo de aprendizaje de restricciones. El objetivo aquí es extraer restricciones de parámetros (p_i) sobre los eventos restantes. El primer paso consiste en extraer de cada traza positiva cualquier relación posible entre los atributos de los distintos eventos presentes en ella. Por ejemplo, supongamos que se tiene la traza:

$$\overrightarrow{e_3} : e_1@10(p_1 = 1, p_2 = 10), e_2@12(p_3 = 1), e_3@15(p_4 = 1)$$

las relaciones que se obtendrían serían: $e_1.p_1 = e_2.p_3$, $e_1.p_1 = e_3.p_4$, $e_2.p_3 = e_3.p_4$, $e_1.p_2 > e_2.p_3$, $e_1.p_2 > e_3.p_4$. En el segundo paso considerando todas las trazas, se calcula la intersección de las relaciones extraídas.

El módulo de aprendizaje de secuencias trabaja en paralelo con el módulo de aprendizaje de parámetros y recibe los mismos datos de entrada. Este módulo produce un conjunto de restricciones ordenadas o secuencias que se deben cumplir para que ocurra un evento complejo.

El primer paso es extraer de cada traza restricciones según el orden en que aparecen los eventos. Por ejemplo, la traza $\overrightarrow{e_3}$ vista en el módulo anterior contiene tres restricciones de secuencia: $e_1 \rightarrow e_2$, $e_2 \rightarrow e_3$, $e_1 \rightarrow e_3$. El segundo paso consiste en intersectar todas las restricciones de secuencias extraídas de cada traza, manteniendo solo aquellas que aparecen en todas las trazas.

El módulo de aprendizaje de negación a diferencia de los demás módulos tiene en cuenta las trazas negativas, que son aquellas que no presentan eventos complejos. La idea de este módulo

es encontrar eventos atómicos que no deben aparecer en una traza para que ocurra un evento complejo.

Para ello el módulo toma como entrada el conjunto $\Re$ de todas las restricciones generadas por los otros módulos anteriormente. Luego selecciona las trazas negativas que satisfacen todas las restricciones en $\Re$ y se aplica el mismo algoritmo que en el módulo de aprendizaje de eventos y en el de aprendizaje de restricciones. Es decir, primero busca los tipos de eventos comunes que aparecen en todas las trazas seleccionadas y después se extraen las restricciones. Las restricciones del conjunto $\Re$ constituyen una regla aprendida. El resultado obtenido en este módulo se agrega a la regla aprendida como una restricción de negación.

5.2. Ajuste de reglas utilizando el paradigma Predicción-Corrección

La propuesta presentada por Turchin et al. [76] consiste en una solución genérica para automatizar la definición de reglas y su actualización a lo largo del tiempo. Esta método combina la información proveída por el analista con técnicas de aprendizaje automático. Dicha propuesta está basada en el Filtro de Kalman Discreto [77]. Los autores definen dos conjuntos de eventos. Un conjunto de eventos históricos h que contiene todos los eventos ocurridos y un conjunto de eventos estimados $\tilde{h}$, el cual contiene todos los eventos que han sido notificados.

Una notificación de evento es representada como una tupla con la estructura $< a_1, a_2, A_1 >$, donde a_1 es un atributo que representa el identificador de cada instancia de eventos, el atributo a_2 indica el tiempo en que un evento fue reportado de ocurrir y A_1 es el conjunto de atributos específicos de un evento. Los eventos son clasificados según sus clases, donde todos los eventos de clase ς_1 comparten el mismo conjunto de atributos. Por ejemplo, un evento e_1 que representa un mensaje escrito en el sistema puede tener la estructura $e_1 =< id, tiempo, long >$, donde $long$ representa la longitud del mensaje.

En este trabajo se hace una diferenciación entre dos tipos de eventos. Son definidos como eventos explícitos aquellos eventos que provienen de una fuente para ser analizados. Por ejemplo, una solicitud de conexión a determinada red.

El otro tipo de evento que se define son los eventos inferidos, los cuales son materializados por el sistema a partir de otros eventos, ya sean explícitos o inferidos. Por ejemplo, un **intento de conexión ilegal** es un evento inferido, materializado por el sistema de seguridad de la red, basándose en el evento explícito de **nueva conexión a la red** y el evento inferido **autorización del usuario denegada**. Ambos tipos de eventos pertenecen al historial de eventos, pero aquellos que ocurren están presentes en h, mientras que los que los estimados están presentes en $\tilde{h}$.

La actual propuesta consta de dos etapas principales repetitivas, una llamada predicción de parámetros de regla y la otra corrección de parámetros de regla. En la etapa de predicción se ejecuta un aprendizaje no supervisado. El procedimiento se basa en el conocimiento pre-existente sobre cómo los parámetros pueden cambiar con el tiempo, así como en la constante actualización de h. Esta etapa permite inferir eventos complejos utilizando los valores de los parámetros de la regla predicha.

En la etapa de corrección los parámetros son ajustados de manera supervisada utilizando la retroalimentación del analista, la cual puede ser dada de forma directa o indirecta. De forma directa implica cambios en el estado del sistema, como actualizar los parámetros de regla para la siguiente etapa de predicción. Mientras que de forma indirecta proporciona una evaluación de la exactitud de $\tilde{h}$, es decir se corrige $\tilde{h}$ para proveer a h de eventos realmente ocurridos. De esta

forma la retroalimentación indirecta puede manifestarse marcando un evento $e_2 \in \tilde{h}$ como no ocurrido o sugiriendo la ocurrencia de un evento e_3 que no se registro en $\tilde{h}$.

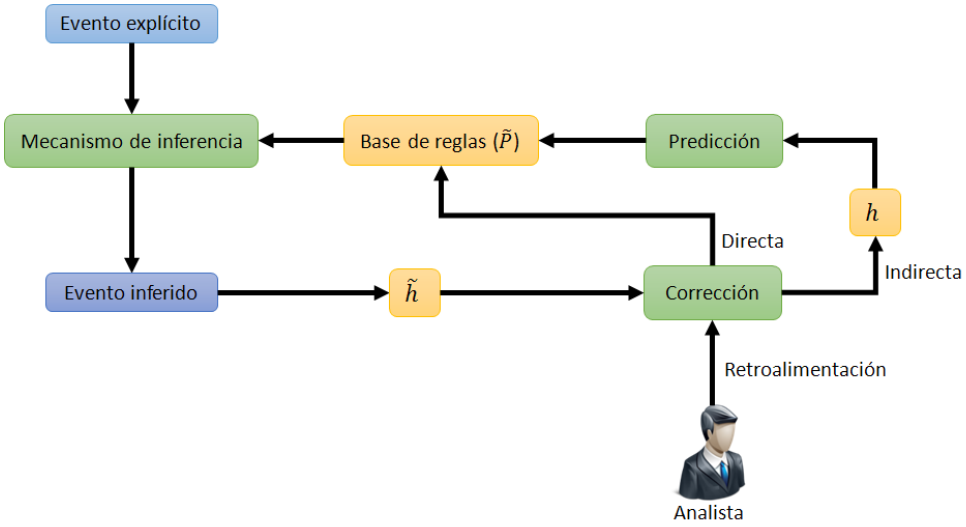

Fig. 12. Esquema de ajuste de reglas utilizando el paradigma Predicción-Corrección.

En esencia, el método de ajuste de reglas consiste en una secuencia repetitiva de acciones que se debe ejecutar para la evaluación y actualización dinámica de los parámetros de la reglas (ver figura 12). A continuación detallamos la secuencia.

1. La etapa de predicción, es en la cual se predice el conjunto estimado de los parámetros $\tilde{P}(I_i)$ de las reglas para el intervalo de tiempo I_i (ver ecuación 1).

$$\tilde{P}(I_i) = \rho(\tilde{P}(I_{i-1}), h(I_{i-1})) + \psi(I_{i-1}). \tag{1}$$

En la ecuación 1, $\rho(\tilde{P}(I_{i-1}), h(I_{i-1}))$ representa la fórmula básica de transición de parámetros, dada por el historial $h(I_{i-1})$ en el intervalo I_{i-1} y $\tilde{P}(I_{i-1})$; a la cual se adiciona la función de ruido predefinida $\psi(I_{i-1})$. La definición inicial del conjunto de parámetros $\tilde{P}(I_0)$ viene dada por el analista.
También se predice el nivel de rendimiento $\varsigma(I_i)$ del mecanismo de inferencia (ver ecuación 3), basándose en el historial de eventos estimados $\tilde{h}(I_{i-1})$ (ver ecuación 2) inferido utilizando los parámetros de las reglas corregidos $\tilde{P}(I_{i-1})$, el conjunto de eventos explícitos $\tilde{E}(I_{i-1})$ y el historial de eventos $h(I_{i-1})$ proveído por el analista.

$$\tilde{h}(I_{i-1}) = F(\tilde{P}(I_{i-1}), \tilde{E}(I_{i-1}), h(I_{i-2})). \tag{2}$$

$$\varsigma(I_i) = Rendimiento(h(I_{i-1}), \tilde{h}(I_{i-1})). \tag{3}$$

F es la función que ejecuta una secuencia de actividades de inferencia en el intervalo de tiempo I_{i-1} y retorna un nuevo historial $\tilde{h}(I_{i-1})$. La evaluación del rendimiento $\varsigma(I_i)$ está motivada por la idea de que la mejora del rendimiento alcanzada tras la corrección del conjunto de parámetros en el intervalo de tiempo anterior, puede servir como una estimación para el rendimiento futuro.

2. La etapa de ejecución del mecanismo de inferencia ocurre durante el intervalo de tiempo I_i, en la cual a partir de los eventos inferidos se obtiene el historial de eventos estimados $\tilde{h}(I_i)$ (ver ecuación 4). Al finalizar el intervalo I_i se obtiene del analista el historial de eventos $h(I_i)$.

$$\tilde{h}(I_i) = F(\tilde{P}(I_i), \tilde{E}(I_{i-1}), h(I_{i-1})). \tag{4}$$

3. En la etapa de corrección de las reglas, se calcula la presición que tubo el mecanismo de inferencia en el intervalo actual I_i, basándose en los eventos inferidos $\tilde{h}(I_i)$ y en el historial actual de eventos $h(I_i)$ (ver ecuación 5).

$$\varsigma(I_i) = Rendimiento(h(I_i), \tilde{h}(I_i)). \tag{5}$$

   Además se corrige el conjunto de parámetros de reglas estimado $\tilde{P}(I_i)$, basándose en la lógica de corrección predefinida (ver ecuación 6).

$$\tilde{P}(I_i) = \xi(\tilde{P}(I_i), h(I_i), \varsigma(I_i)). \tag{6}$$

   La función de corrección $\xi()$ puede ser definida a partir de un método de aprendizaje de datos, como aprendizaje estadístico o técnicas de aprendizaje automático.

4. Se regresa a la primera etapa para el siguiente intervalo de tiempo I_{i+1}.

5.3. Conclusiones parciales

El empleo de técnicas de procesamiento de eventos complejos se ha ido incrementando en los últimos años, por su utilidad en el monitoreo en tiempo real y la detección de eventos de interés.

En la sección 5.2, se analizó un enfoque iterativo para automatizar el proceso de definición de reglas y su actualización durante el transcurso del tiempo. Dicho enfoque requiere de la intervención de analistas en la etapa de ajustes para su correcto funcionamiento. Esta característica lo pone de cierta manera en desventaja con respecto a otros sistemas que no requieren de la intervención de analistas. Por ejemplo, la propuesta analizada en la sección 4.1 no requiere la presencia de un analista para realizar ajustes sobre sus reglas. No obstante, la característica que presenta de ser un proceso iterativo tiene una ventaja, la cual consiste en actualizar las reglas periódicamente.

El trabajo analizado en la sección 5.1 cuenta con una característica que permite incluir en las reglas condiciones de negación. Para ello el algoritmo puede prescribir la ausencia de ciertos eventos y obtener un patrón de negación. Esto hace que el sistema tenga un mejor nivel de expresividad en las reglas, ya que se agrega un operador de negación. Sin embargo en el campo de CEP, el aprendizaje de relaciones causales entre eventos atómicos y complejos, así como la complejidad de la definición de las reglas, continúan siendo retos sobre los cuales trabajar.

6. Descubrimiento de reglas utilizando programación lógica inductiva

La programación lógica inductiva [78] (ILP por sus siglas en inglés) puede ser definida como la intersección del aprendizaje automático y la programación lógica. Es por ello que en un sistema basado en ILP, los ejemplos son representados utilizando un lenguaje de programación lógica para ser procesados por el sistema de aprendizaje. Dicho sistema forma conceptos generalizando

los ejemplos de entrenamiento proveídos por un analista. Los conceptos desarrollados pueden ser utilizados para predecir futuros ejemplos.

Los algoritmos ILP que aplican inferencia inductiva, pueden deducir cláusulas generales $\bar{q}$ a partir de ejemplos ε. Como en el caso de observarse un conjunto de ejemplos ε que indican que uno o mas objetos son blancos, $\bar{q}$ podría contener la cláusula $\bar{r}$ de que todos los objetos son blancos. En la inducción se debe cumplir que $\bar{q} \models \varepsilon$, además $\bar{q}$ y ε deben ser consistentes, lo cual significa que la observación de un objeto negro descarta la cláusula $\bar{r}$. Las cláusulas están formadas por una cabeza y un cuerpo que restringen los predicados que pueden ocurrir. Si analizamos una cláusula como una regla de clasificación (ver sección 2.2), la cabeza representaría a $\overrightarrow{r}$ y el cuerpo a $\overleftarrow{r}$.

En los sistemas basados en ILP es posible separar los elementos de $\bar{q} \models \varepsilon$ en: conocimiento del dominio $\overleftarrow{q}$, hipótesis λ y ejemplos ε, los cuales pueden ser separados en ejemplos positivos ε^+ y ejemplos negativos ε^-. Dichos elementos tienen una relación $\overleftarrow{q} \wedge \lambda \models \varepsilon$ y son programas lógicos. Estos programas lógicos están formados por cláusulas y son representados mediante un lenguaje lógico. Los algoritmos basados en ILP analizados cumplen con el esquema general presentado en esta sección (ver algoritmo 6).

Algoritmo 6: ILP $(\varepsilon, \overleftarrow{q})$

Entrada: ε - conjunto de ejemplos
Entrada: $\overleftarrow{q}$ - conocimiento del dominio
Salida: λ - conjunto de cláusulas generales (hipótesis)

```
1  λ ← ∅
2  ε+, ε− ← Preparar-Entrada (ε);
3  while (Quedan-Instancias(ε+)) do
4  |  ε′ ← Seleccionar-Ejemplos(ε+);
5  |  λ1 ← Obtener-Cláusulas(ε′, ←q);
6  |  r̄ ← Generalizar-Cláusulas(λ1, ε+, ε−);
7  |  λ ← Adicionar-Cláusula(r̄, λ);
8  |  ε+ ← Actualizar-Ejemplos(ε+, r̄);
9  end
10 return λ;
```

Los algoritmos parten de un conjunto de ejemplos ε y un conocimiento previo del dominio $\overleftarrow{q}$. A partir del conjunto ε y con el uso de la función Preparar-Entrada se obtienen dos subconjuntos, ε^+ y ε^- (ver algoritmo 6, línea 2). Luego, mientras no se cumpla la condición de parada definida por la función Quedan-Instancias es ejecutado un proceso iterativo (ver algoritmo 6, línea 3). La condición de parada por lo general es $\varepsilon^+ = \emptyset$ aunque puede tener otra, como en el sistema Golem [79] donde se agrega además la condición de que sino incrementa la cobertura de λ se detenga el ciclo; o como en el algoritmo BACQ [80] donde el analista define el número de iteraciones. La tabla 4 muestra un resumen de los algoritmos ILP analizados en esta sección.

Tabla 4. Estado del arte de los algoritmos voraces dentro de ILP.

	Cigol [81]	Progol [82]	Progolem [83]	Golem [79]	BACQ [80]
Seleccionar-Cláusulas	Selecciona el primer ejemplo positivo de ε^+.			Se selecciona una muestra aleatoria de pares de cláusulas $\ddot{\varepsilon}^+$, donde $\varepsilon^{+\prime} \in \ddot{\varepsilon}^+$, $\varepsilon^{+\prime} : \{\bar{r}_1, \bar{r}_2\}$ tal que $\bar{r}_1, \bar{r}_2 \in \varepsilon^+$, y el tamaño de la muestra $\ddot{\varepsilon}^+$ es definido por el analista.	Se procesa ε en su totalidad.
Obtener-Cláusulas	Se ejecuta un procedimiento de truncación [81], donde se obtiene un conjunto $\overleftarrow{q}'$ tal que $\overleftarrow{q}' \subseteq \bar{q}$. Luego, se tiene que el conjunto de cláusulas $\lambda_1 \leftarrow \overleftarrow{q}' \cup \varepsilon'$.	Construye la cláusula más específica para el ejemplo seleccionado [82]. Luego, se crean nuevas cláusulas combinando los predicados que contiene la cláusula más específica obtenida.		Para cada par en $\ddot{\varepsilon}^+$ se calcula la cláusula que representa la menor generalización general que no cubra ningún ejemplo en ε^- y se adiciona al conjunto λ_1 de cláusulas.	Se ejecuta el algoritmo de búsqueda aleatoria local [80] que genera una cláusula. Este se repite una cantidad de veces k definida por el analista, obteniéndose un conjunto λ_1 de cláusulas.
Generalizar-Cláusulas	Aplicando el algoritmo descrito en [84] se obtiene $\bar{r}$ como la menor generalización general de $\overleftarrow{q}' \cup \varepsilon'$.	La cláusula que más ejemplos positivos cubra y ninguno negativo, es seleccionada como cláusula general.	Se ejecuta el algoritmo ARMG [83] que evalúa las cláusulas del conjunto λ_1 y retorna la cláusula que mayor puntuación obtuvo. La cláusula obtenida se le aplica un procedimiento de reducción basado en los ejemplos negativos [79].	Del conjunto λ_1 se selecciona como $\bar{r}$ la cláusula que mayor cobertura tuvo sobre los ejemplos de ε^+. La cláusula seleccionada puede ser reducida aplicando alguna de las técnicas descritas en [79].	Se selecciona la cláusula λ que maximiza la medida de calidad definida en [80].
Adicionar-Cláusula	Si se determina que $\bar{r}$ puede ser una cláusula negativa se adiciona a $\varepsilon^- \leftarrow \varepsilon^- \cup \bar{r}$, en otro caso se modifica el conjunto de cláusulas $\bar{q} \leftarrow (\bar{q} - \overleftarrow{q}') \cup \{\bar{r}\}$.	Se adiciona a $\lambda \leftarrow \lambda \cup \{\bar{r}\}$.			Se adiciona a $\lambda \leftarrow \lambda \cup \{\bar{r}\}$ y se guarda para $\bar{r}$ su valor de confianza de predicción. Para clasificar una instancia n se suma el valor de confianza de predicción de todas las cláusulas en λ que cubren n. Si la sumatoria es positiva se clasifica n como positivo, de otra manera se clasifica como negativo.
Actualizar-Ejemplos	Se eliminan los ejemplos positivos en ε^+ cubiertos por $\bar{q}$.	Los ejemplos positivos en ε^+ que son cubiertos por la regla general $\bar{r}$ son eliminados.		Los ejemplos positivos en ε^+ que son cubiertos por el conjunto de cláusulas λ son eliminados.	A los ejemplos del conjunto ε clasificados correctamente por $\bar{r}$ se les reduce el peso, mientras que a los clasificados incorrectamente se les aumenta. De esta forma el algoritmo de aprendizaje se centra en los ejemplos que han sido clasificados incorrectamente.

Dentro de cada iteración, se procede a seleccionar un conjunto de ejemplos positivos de forma tal que $\varepsilon' \subseteq \varepsilon^+$, mediante una función Seleccionar-Ejemplos (ver algoritmo 6, línea 4). Seguidamente, con la función Obtener-Cláusulas, las cláusulas que conforman $\overleftarrow{q}$ y los ejemplos seleccionados ε', se procede a obtener un conjunto de cláusulas λ_1 (ver algoritmo 6, línea 5). Del conjunto λ_1 se procede a extraer la cláusula más general, por medio de la función Generalizar-Cláusulas, en la cual se tiene en cuenta ε^+ y ε^- para definir el criterio de selección (ver algoritmo 6, línea 6). En dependencia de la propuesta, la cláusula más general $\bar{r}$ es adicionada al conjunto de cláusulas conocido como hipótesis λ (ver algoritmo 6, línea 7). En el caso del sistema Cigol [81], se realiza una prueba para determinar si $\bar{r}$ puede ser una cláusula negativa y en caso de serlo es adicionado a ε^-. Finalmente, se procede a actualizar los ejemplos cubiertos por $\bar{r}$ (ver algoritmo 6, línea 8).

A continuación se analizará un ejemplo de como se obtiene una cláusula general (regla) utilizando Progol.

6.1. Progol

Progol [82] es un sistema basado en ILP. Para obtener cláusulas generales en este sistema se siguen tres pasos fundamentales.

A partir de los ejemplos ε introducidos por el analista, el primer paso consiste en construir la cláusula más específica del primer ejemplo positivo. Veamos el siguiente ejemplo donde se muestra como queda formada dicha cláusula.

Dado una secuencia de automóviles donde cada uno tiene un modelo de motor y el motor a su vez tiene un número que representa la cantidad de caballos de fuerza. El problema está en encontrar una regla que permita predecir a partir de la propiedad del motor si el automóvil es deportivo o de paseo. Para ello se introduce el primer ejemplo positivo:

$$deportivo(auto1),$$

y se define el tipo de variable *auto* que es a la cual va a pertenecer el término $auto1$:

$$auto(auto1).$$

Luego, se agrega el conocimiento del dominio $\overleftarrow{q}$, la cual consiste en más información sobre el automóvil y su motor:

$$autoMotor(auto1, motor1\_1),$$

también se adiciona el tipo de variable para el motor:

$$motor(motor1\_1).$$

El mismo procedimiento se repite para definir los caballos de fuerza del motor:

$$motorFuerza(motor1\_1, 500)$$
$$motorFuerza(500).$$

Para obtener una cláusula general es necesario definir cuales son los predicados que pueden formar parte de la cabeza y cuales del cuerpo. Para restringir los predicados de la cabeza se utiliza la función *modeh* y para los del cuerpo *modeb*, lo cual quedaría de la siguiente manera:

$modeh(1, deportivo(+auto))$,
$modeb(1, autoMotor(+auto, -motor))$,
$modeb(1, motorFuerza(+motor, \#motorFuerza))$.

El número 1 representa el umbral de la cantidad de instanciaciones alternativas del predicado, el cual puede ser un número entero positivo o $*$ en caso de que no se tenga un limite de soluciones para la instanciación. Una instanciación del predicado es un reemplazo de los tipos de variables o constantes teniendo en cuenta la información suministrada por los símbolos $+$, $-$ y $\#$. Los símbolos $+$ y $-$ indican si el tipo de variable es de entrada o de salida respectivamente. Cuando es usado el símbolo $\#$ indica que se deben tener constantes en lugar de variables del tipo que se defina.

A partir de los ejemplos insertados, Progol aplica un proceso de sustitución, donde cada término $\bar{y}_i$ pasa a ser representado por una variable y_i. Para ello se crea un conjunto finito de pares en la forma $\{y_1/\bar{y}_1, y_2/\bar{y}_2, ..., y_{\hat{\varepsilon}}/\bar{y}_{\hat{\varepsilon}}\}$, donde $\hat{\varepsilon}$ representa la cantidad de términos introducidos. Es decir, a partir de la función *modeh* utilizada en el ejemplo presentado es posible obtener en la cabeza de la cláusula un predicado en la forma $deportivo(y_1)$, donde y_1 es una variable de tipo *auto*. Teniendo en cuenta esto, para el primer ejemplo positivo $deportivo(auto1)$ se obtiene como cláusula más específica:

$deportivo(y_1) : -autoMotor(y_1, y_2), motorFuerza(y_2, 500)$,

la cual define que un auto es deportivo si su motor es de 500 caballos de fuerza.

El segundo paso consiste en crear nuevas cláusulas combinando los predicados que contiene la cláusula más específica obtenida. Las cláusulas obtenidas son comparadas teniendo en cuenta su cobertura. La cláusula que más ejemplos positivos cubra y ninguno negativo, es seleccionada como cláusula general.

Por último, en el tercer paso se adiciona la regla general obtenida al conjunto de conocimiento del dominio. Luego, los ejemplos positivos que son cubiertos por la regla general son eliminados. El procedimiento se repite mientras $\varepsilon^+ \neq \emptyset$.

6.2. Conclusiones parciales

El uso de algoritmos ILP que aplican inferencia inductiva como los analizados en esta sección, permiten aprender cláusulas a partir de un conjunto de ejemplos y un conocimiento del dominio que se tenga. Por lo general, este tipo de técnicas son aplicadas en la bioinformática. Por ejemplo, en la predicción de la acción de fármacos, clasificación biológica del agua, predicción de mutación en células, entre otros. Dada la estructura que presentan las cláusulas es posible representarlas como reglas de clasificación. Teniendo en cuenta la estructura de las cláusulas y el esquema general que siguen las técnicas vistas en esta sección; sería posible extender alguno de los algoritmos existentes o proponer alguna nueva solución basada en este enfoque para su aplicación en la detección de fraudes.

7. Bases de datos usadas en las experimentaciones

Algunos de los métodos analizados en este trabajo reportaron las bases de datos sobre las cuales evaluaron sus propuestas. En la tabla 5 se presenta un resumen de las principales características de las bases de datos reportadas. Las características presentadas son, el nombre, una breve descripción, la cantidad de instancias para el entrenamiento, la cantidad de instancias para la evaluación, el número de clases y su disponibilidad.

Tabla 5. Bases de datos reportadas en las técnicas analizadas.

	Nombre	**Descripción**	**# de instancias de entrenamiento**	**# de instancias de prueba**	**# de clases**	**Disponibilidad**
	Concurso KDD '99 [85]	Registros de conexiones generadas en una simulación de una red militar.	4898431 (23 clases)	311029 (37 clases)	Se agrupan en 5 categotías.	Pública.
	World Factbook [86]	Información geográfica, económica y política de 266 países.	1000	1000	2	
	Mutagenicity [87]	Información de moléculas.	230	-	2	
Repositorio UCI [88]	Mushrooms	Hongos descritos en términos de sus características físicas.	3988	4136	2	
	Iris	Clasificación de flores.	150	-	2	
	Labor Relations	Negociaciones laborales.	57	-	2	
	Promoter Gene Sequences	Secuencias promotoras de ADN.	106	-	2	
	Sonar, Mines vs. Rocks	Clasificación de señales de sonar.	208	-	2	
	Congressional Voting Records	Registros de votos del congreso de Estados Unidos.	300	135	2	
	Image Segmentation	Análisis de imágenes.	1133	1177	7	
	Splice-junction Gene Sequences	Datos de empalmes en las secuencias de ADN.	1614	1561	3	
	Audiology	Diagnostico médico.	226	-	24	
	Thyroid Disease	Diagnostico médico.	2800	972	5	

Las bases de datos presentadas en la tabla 5 se pueden obtener gratuitamente en sus respectivos sitios web en internet. En algunas propuestas analizadas, no se hace público el nombre de la base de datos utilizada, debido a la confidencialidad de sus datos. Es notable que de todas las técnicas analizadas, solo una basada en procesamiento de eventos complejos reportó sus resultados sobre una base de datos pública vinculada a las telecomunicaciones (KDD '99).

Enfocándonos en la cantidad de clases que posee cada base de datos que ha sido procesada por las técnicas analizadas en este trabajo, podemos inferir que los métodos basados en generación automática de reglas pueden atacar problemas de categorización de multiples clases. Además, atendiendo a la cantidad de instancias se evidencia la existencia de variantes para procesar grandes volúmenes de datos.

Existe una gran variedad de bases de datos públicas sobre las cuales se han aplicado técnicas de generación automática de reglas. No obstante, es muy baja la cantidad de bases de datos asociadas al área de las telecomunicaciones que han sido procesadas por los métodos analizados en este trabajo. La principal causa es que en el área de las telecomunicaciones por lo general se

procesan datos reales, por lo que se trabaja con información privada de usuarios. Esto hace que los datos adquieran cierto nivel de confidencialidad que no permita hacerlos públicos.

8. Conclusiones generales

Entre las debilidades detectadas en las técnicas de búsqueda automática de reglas; figura que la ocurrencia de ligeros cambios en los datos pueden cambiar drásticamente la estructura de las reglas. Por tanto la interpretación que se les da a las reglas también se ve afectada, lo cual conduce a una inestabilidad del modelo.

Otra dificultad con que cuentan los métodos tratados es que definen grupos de instancias que contienen los valores más homogéneos. Si las reglas no son adecuadamente definidas mediante la relación entre las condiciones y los resultados, entonces el modelo obtenido tendrá elevados errores de predicción. Lo cual conlleva a que el rendimiento de predicción no sea óptimo.

Dentro de las principales ventajas podemos destacar que los conjuntos de condiciones generados, son altamente interpretables y fáciles de implementar. La lógica que sigue su construcción permite que puedan manejar eficazmente muchos tipos de condiciones sin necesidad de preprocesarlas. Los tipos de condiciones pueden estar asociados a variables continuas, discretas, nominales, entre otras.

También consideramos como una ventaja, el hecho de que no se requiere que el analista especifique la relación entre las condiciones para formar una regla. Además, estos métodos pueden procesar con eficacia los datos faltantes y llevar a cabo implícitamente la selección de atributos. Las ventajas que ofrecen las técnicas basadas en búsqueda automática de reglas hacen que sean deseadas para tratar diversos problemas en el ámbito empresarial, sobre todo en la detección de fraudes.

Obtener bases de datos reales de telecomunicaciones sobre las cuales evaluar el rendimiento de los métodos que se propongan resulta muy difícil. Esto se debe a razones de privacidad y limitaciones legales. Ello impone limitaciones para establecer comparaciónes entre los resultados reportados por distintos autores, ya que no están disponibles de forma pública las bases utilizadas.

En este trabajo analizamos métodos que generan elementos con estructuras similares a las reglas, como son los algoritmos ILP que aplican inferencia inductiva. Estas técnicas permiten formar cláusulas generales que pueden ser vistas como reglas, combinando los ejemplos existentes con el conocimiento del dominio por parte de los analistas. Los datos son introducidos al sistema en un lenguaje de programación lógica. Luego, los ejemplos son generalizados y se producen las cláusulas generales que pueden ser utilizadas para identificar nuevos ejemplos.

Aunque no son utilizados en áreas como la detección de fraudes, dada la forma en que procesan la información y los resultados que se obtienen, es posible extenderlos para su aplicación en nuestra área de interés. Para ello, sería necesario establecer un procedimiento que permita llevar los datos que se procesan a un lenguaje de programación lógica. De forma similar interpretar los resultados obtenidos y convertirlos a reglas de clasificación.

La rápida evolución de los negocios hoy en día hace que las organizaciones empresariales deban tomar decisiones proactivas de manera casi inmediata. Por lo cual es de vital importancia para las empresas tener reglas de negocios válidas actualizadas diariamente. Las técnicas de descubrimiento de reglas de negocio son implementadas para mantener actualizadas las reglas en el menor tiempo posible. Para ello procesan datos que son actualizados con frecuencia, como los

registros que se generan de procesos de negocio. Esta característica permite detectar tendencias negativas en períodos de tiempo cortos y atacarlas por medio de las reglas que genera.

Por lo general las técnicas basadas en descubrimiento de reglas de negocios no se aplican a la detección de fraude, más bien son utilizadas para trazar una estrategia en las empresas que las guíe por un camino que maximice sus ingresos. Desarrollar una propuesta para detectar fraudes basándonos en características propias de las técnicas de descubrimiento de reglas de negocios puede ser una línea a seguir para futuros trabajos.

Referencias bibliográficas

1. CFCA: 2013 global fraud loss survey. [citado 27 de junio de 2014]. Disponible en Internet: http://www.cvidya.com/media/62059/global-fraud_loss_survey2013.pdf
2. Lin, S.J., Jheng, Y.Y., Yu, C.H.: Combining ranking concept and social network analysis to detect collusive groups in online auctions. Expert Systems with Applications, Pergamon Press, Inc., Tarrytown, NY, USA **39**(10) (August 2012) 9079–9086
3. Chaudhary, K., Yadav, J., Mallick, B.: Article: A review of fraud detection techniques: Credit card. International Journal of Computer Applications **45**(1) (May 2012) 39–44
4. Golmohammadi, K., Zaiane, O.R.: Data mining applications for fraud detection in securities market. In: Proceedings of the 2012 European Intelligence and Security Informatics Conference. EISIC '12 (2012) 107–114
5. Jiang, N., Jin, Y., Skudlark, A., Hsu, W.L., Jacobson, G., Prakasam, S., Zhang, Z.L.: Isolating and analyzing fraud activities in a large cellular network via voice call graph analysis. In: Proceedings of the 10th international conference on Mobile systems, applications, and services. MobiSys '12, New York, NY, USA, ACM (2012) 253–266
6. Elmi, A.H., Ibrahim, S., Sallehuddin, R.: Detecting sim box fraud using neural network. Lecture Notes in Electrical Engineering, Springer **215** (2012) 575–582
7. Rajani, S., Padmavathamma, M.: A model for rule based fraud detection in telecommunications. International Journal of Engineering Research & Technology **1**(5) (July, 2012) 1–7
8. Murynets, I., Jover, R.P.: Anomaly detection in cellular machine-to-machine communications. In IEEE International Conference On Communications, Budapest, Hungary (2013)
9. Herrera-Semenets, V., Prado-Romero, M.A., Gago-Alonso, A.: Análisis de los métodos de detección de fraude en servicios de telecomunicaciones. Technical Report RT_023, Serie Gris, Advanced Technologies Application Center (CENATAV), La Habana, Cuba (February 2014)
10. Boding, B.S., Siddens, C.H.: Fraud detection system automatic rule population engine. Patent. US 2012/0278246 A1 (November 2012)
11. Pereira, I.S.A., Dockhorn Costa, P., Almeida, J.P.A.: A rule-based platform for situation management. In: Proceedings of the 2013 IEEE International Multi-Disciplinary Conference on Cognitive Methods in Situation Awareness and Decision Support (CogSIMA), San Diego, CA, USA (2013)
12. Brook, A.A., Kim, N.W., Lingafelt, C.S.: Method of operating an intrusion detection system according to a set of business rules. Patent. US 2002/0169982 A1 (November 2002)
13. Hernández Orallo, J., Ramírez Quintana, M.J., Ferri Ramírez, C.: Introducción a la Minería de Datos. Pearson Educación, Madrid, España (2004) 680 p.
14. Kuhn, M., Johnson, K.: Applied predictive modeling. Springer (2013)
15. Engelbrecht, A.P., Schoeman, L., Rouwhorst, S.: A building block approach to genetic programming for rule discovery. Data Mining: A Heuristic Approach **208** (2002) 174–189
16. Frank, E., Witten, I.H.: Generating accurate rule sets without global optimization. (1998)
17. Ryszard, S.M., Kenneth, A.K., Jaroslaw, P., Janusz, W., Scott, M., Doug, S.: Natural induction and conceptual clustering: A review of applications. Reports of the Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, USA (2006)
18. Liu, H., Tan, S.T.: X2r: A fast rule generator. In: Proceedings of IEEE International Conference on Systems, Man and Cybernetics, Vancouver, Canada (October 1995)
19. William W, C.: Fast effective rule induction. In Proceedings of the 12th international conference on machine learning, Tahoe City (1995) 115–123
20. Parpinelli, R.S., Lopes, H.S., Freitas, A.A.: An ant colony algorithm for classification rule discovery. Data Mining: A Heuristic Approach **208** (2002) 191–208

21. Quinlan, J.R.: Mdl and categorical theories (continued). In: MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-, Citeseer (1995) 464–470
22. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on **26**(1) (1996) 29–41
23. Quinlan, J.R.: C4.5: programs for machine learning. San Francisco, California, Morgan Kaufmann (1993) 302 p.
24. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and regression trees. CRC press (1984)
25. Xiaoliang, Z., Jian, W., Hongcan, Y., Shangzhuo, W.: Research and application of the improved algorithm c4. 5 on decision tree. In: Test and Measurement, 2009. ICTM'09. International Conference on. Volume 2., IEEE (2009) 184–187
26. Lawrence, R.L., Wright, A.: Rule-based classification systems using classification and regression tree (CART) analysis. Photogrammetric engineering and remote sensing **67**(10) (2001) 1137–1142
27. Guerra-Gandón, A., Vega-Pons, S., Ruiz-Shulcloper, J.: Algoritmos de agrupamiento conceptuales: un estado del arte. Technical Report RT_050, Serie Azul, Advanced Technologies Application Center (CENATAV), La Habana, Cuba (May 2012)
28. Pérez Suárez, A., Medina Pagola, J.E.: Algoritmos para el agrupamiento conceptual de objetos. Technical Report RT_024, Serie Gris, Advanced Technologies Application Center (CENATAV), La Habana, Cuba (April 2014)
29. Michalski, R.S.: A theory and methodology of inductive learning. Volume 2., CA, USA (1986) 83–129
30. Michalski, R.S., Diday, E.: A recent advance in data analysis: Clustering objects into classes characterized by conjunctive concepts. (1981) 33–56
31. Michalski, R.S., Diday, E.: Learning from observation: Conceptual clustering. Volume 1., CA, USA (1983) 331–336
32. Michalski, R.S., Stepp, R.E.: Automated construction of classifications: Conceptual clustering versus numerical taxonomy. Volume 5. (1983) 396–410
33. Martínez-Trinidad, J.F., Ruiz-Shulcloper, J., Pons-Porrata, A.: Algoritmo LC-conceptual: Una mejora de la etapa intencional utilizando reglas de generalización., Mexico D.F. (1999) 179–188
34. Pons-Porrata, A., Ruiz-Shulcloper, J., Martínez-Trinidad, J.F.: RGC: a new conceptual clustering algorithm for mixed incomplete data sets. (2002) 1375–1385
35. Hühn, J., Hüllermeier, E.: Furia: an algorithm for unordered fuzzy rule induction. Data Mining and Knowledge Discovery, Springer **19** (2009) 293–319
36. Döhring, M., Klopper, C., Zimmermann, B.: Towards efficiently running workflow variants by automated extraction of business rule conditions. In 23rd GI-Workshop on Foundations of Databases, Obergurgl, Austria (2011)
37. Holmes, G., Hall, M., Prank, E.: Generating rule sets from model trees. Springer (1999)
38. Rosso-Pelayo, D.A., Gonzalez-Mendoza, M., Trejo-Ramírez, R.A., Hernandez-Gress, N.: Business process mining and rules detection for unstructured information. In: Proceedings of the 9th IEEE Mexican International Conference on Artificial Intelligence (MICAI 2010), Pachuca, Mexico (2010)
39. Rosso-Pelayo, D.A., Trejo-Ramírez, R.A.: Detecting associations between business processes and unstructured information. In: Proceedings of the 5th International Congress on Electronics and Biomedical Engineering, Computer Science and Informatics (CONCIBE SCIENCE 2009), Guadalajara, Mexico (2009)
40. Linehan, M.H., Putrycz, E.: Introduction to "rule transformation and extraction" track. Lecture notes in computer science, Springer **5858** (2009) 137–143
41. Alexander-Schutz, P.B.: Relext: A tool for relation extraction from text in ontology extension. Lecture notes in computer science, Springer **3729** (2005) 593–600
42. Martínez-Fernández, J.L., González, J.C., Villena, J., Martínez, P.: A preliminary approach to the automatic extraction of business rules from unrestricted text in the banking industry. Lecture notes in computer science, Springer **5039** (2008) 299–310
43. IBM: Finding and managing your business rules using ILOG and rational asset analyzer (RAA). ftp://ftp.software.ibm.com/software/os/systemz/summit/handouts/Track_3_Session_4_-_Finding_and_managing_your_business_rules_using_ILOG_and_Rational_Asset_Analyzer_28RAA29_2Columbus29.pdf (2011)
44. Schmiedl, R., Ames, M.: Next generation detection engine for fraud and compliance. SAS Global Forum (2013)
45. zur Muehlen, M., Indulska, M., Kamp, G.: Business process and business rule modeling languages for compliance management: a representational analysis. In: Tutorials, posters, panels and industrial contributions at the 26th international conference on Conceptual modeling-Volume 83, Australian Computer Society, Inc. (2007) 127–132
46. Bonais, M., Rahayu, W., Pardede, E.: Integrating information systems business rules into a design model. In 15th International Conference on Network-Based Information Systems (2012)

47. Nazarenko, A., Lévy, F.: Combining acquisition and debugging of business rule models. In: Theory, Practice, and Applications of Rules on the Web. Springer (2013) 234–248
48. Schlosser, S., Baghi, E., Otto, B., Oesterle, H.: Toward a functional reference model for business rules management. In 47th International Conference on System Science, Hawaii (2014)
49. Wang, C., Zhou, Y., Chen, J.: Extracting prime business rules from large legacy system. In: Computer Science and Software Engineering, 2008 International Conference on. Volume 2., IEEE (2008) 19–23
50. Gang, X.: Business rule extraction from legacy system using dependence-cache slicing. In: Information Science and Engineering (ICISE), 2009 1st International Conference on, IEEE (2009) 4214–4218
51. Chaparro, O., Aponte, J., Ortega, F., Marcus, A.: Towards the automatic extraction of structural business rules from legacy databases. In: Reverse Engineering (WCRE), 2012 19th Working Conference on, IEEE (2012) 479–488
52. Francia, L., Moreno, J.J.: Detección automática de tendencias riesgosas en sistemas de gestión de procesos de negocios. In 8th LACCEI Latin American and Caribbean Conference for Engineering and Technology (LACCEI'2010), Arequipa, Perú (June 2010)
53. Minakov, I., Rzevski, G., Skobelev, P., Volman, S.: Automatic extraction of business rules to improve quality in planning and consolidation in transport logistics based on multi-agent clustering. In: Autonomous Intelligent Systems: Multi-Agents and Data Mining. Springer (2007) 124–137
54. Rzevski, G., Skobelev, P., Minakov, I., Volman, S.: Dynamic pattern discovery using multi-agent technology. In: Proceedings of the 6th WSEAS International Conference on Telecommunications and Informatics (TELE_INFO'07), Dallas, Texas, USA. (2007) 75–81
55. Crerie, R., Baião, F.A., Santoro, F.M.: Discovering business rules through process mining. In: Enterprise, Business-Process and Information Systems Modeling. Springer (2009) 136–148
56. Group), B.B.R.: Defining business rules what are they really? rev. 1.3. [citado 23 de abril de 2014]. Disponible en Internet: http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf
57. van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H., Weijters, A., Van Der Aalst, W.M.: The prom framework: A new era in process mining tool support. In: Applications and Theory of Petri Nets 2005. Springer (2005) 444–454
58. Dumas, M., Van der Aalst, W.M., Ter Hofstede, A.H.: Process-aware information systems: bridging people and software through process technology. John Wiley & Sons (2005)
59. van der Aalst, W.M., De Beer, H., van Dongen, B.F.: Process mining and verification of properties: An approach based on temporal logic. Springer (2005)
60. van der Aalst, W.M., Giinther, C.: Finding structure in unstructured processes: The case for process mining. In: Application of Concurrency to System Design, 2007. ACSD 2007. Seventh International Conference on, IEEE (2007) 3–12
61. De Medeiros, A.A., Pedrinaci, C., van der Aalst, W.M., Domingue, J., Song, M., Rozinat, A., Norton, B., Cabral, L.: An outlook on semantic business process mining and monitoring. In: On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops, Springer (2007) 1244–1255
62. de Medeiros, A.K.A., Van der Aalst, W., Pedrinaci, C.: Semantic process mining tools: core building blocks. (2008)
63. SBVR: Semantics of business vocabulary and rules. [citado 24 de abril de 2014]. Disponible en Internet: http://www.omg.org/spec/SBVR
64. ProMimport: Swiss army knife for event logs. [citado 24 de abril de 2014]. Disponible en Internet: http://www.promtools.org/promimport
65. ARFF: Attribute-relation file format. [citado 25 de abril de 2014]. Disponible en Internet: http://www.cs.waikato.ac.nz/ ml/weka/arff.html
66. WEKA: Data mining software. [citado 25 de abril de 2014]. Disponible en Internet: http://www.cs.waikato.ac.nz/ml/weka
67. Crerie, R., Baião, F.A., Santoro, F.M.: Identificacao de regras de negocio utilizando mineracao de processos. In: Companion Proceedings of the XIV Brazilian Symposium on Multimedia and the Web, ACM (2008) 241–246
68. Anicic, D., Fodor, P., Rudolph, S., Stühmer, R., Stojanovic, N., Studer, R.: A rule-based language for complex event processing and reasoning. In: Web Reasoning and Rule Systems. Springer (2010) 42–57
69. Paschke, A., Kozlenkov, A., Boley, H.: A homogeneous reaction rule language for complex event processing. International Workshop on Event Drive Architecture for Complex Event Process. ACM (2007)
70. Bry, F., Eckert, M.: Rule-based composite event queries: The language xchangeeq and its semantics. Springer, Berlin, Heidelberg (2007)
71. Broda, K., Clark, K., Miller, R., Russo, A.: SAGE: a logical agent-based environment monitoring and control system. Springer (2009)

72. Schultz-Møller, N.P., Migliavacca, M., Pietzuch, P.: Distributed complex event processing with query rewriting. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems. Number 4, New York, NY, USA, ACM (2009) 4:1–4:12
73. Demers, A., Gehrke, J., Hong, M., Riedewald, M., White, W.: Towards expressive publish/subscribe systems. In: Advances in Database Technology-EDBT 2006. Springer (2006) 627–644
74. Margara, A., Cugola, G., Tamburrelli, G.: Learning from the past: automated rule generation for complex event processing. In: Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems, ACM (2014) 47–58
75. Cortes, C., Vapnik, V.: Support-vector networks. Machine learning **20**(3) (1995) 273–297
76. Turchin, Y., Gal, A., Wasserkrug, S.: Tuning complex event processing rules using the prediction-correction paradigm. In: Proceedings of the Third ACM International Conference on Distributed Event-Based Systems, New York, NY, USA, ACM (2009) 1–12
77. Kalman, R.E.: A new approach to linear filtering and prediction problems. Journal of Fluids Engineering **82**(1) (1960) 35–45
78. Muggleton, S., De Raedt, L.: Inductive logic programming: Theory and methods. The Journal of Logic Programming **19** (1994) 629–679
79. Muggleton, S., Feng, C.: Efficient induction of logic programs. Inductive logic programming **38** (1992) 281–298
80. Horváth, T., Hoche, S., Wrobel, S.: Effective rule induction from labeled graphs. In: Proceedings of the 2006 ACM symposium on Applied computing, ACM (2006) 611–616
81. Muggleton, S., Buntine, W.: Machine invention of first-order predicates by inverting resolution. Inductive logic programming (1992) 261–280
82. Muggleton, S.: Inverse entailment and progol. New generation computing **13**(3-4) (1995) 245–286
83. Muggleton, S., Santos, J., Tamaddoni-Nezhad, A.: Progolem: a system based on relative minimal generalisation. In: Inductive Logic Programming. Springer (2010) 131–148
84. Plotkin, G.: Automatic methods of inductive inference. PhD thesis, Edinburgh University (1972)
85. 1999, K.C.: Computer network intrusion detection. [citado 10 de noviembre de 2014]. Disponible en Internet: http://sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection
86. CIA: Download the world factbook. [citado 10 de noviembre de 2014]. Disponible en Internet: https://www.cia.gov/library/publications/download/
87. of Oxford, U.: Departament of computer science. [citado 11 de noviembre de 2014]. Disponible en Internet: http://www.cs.ox.ac.uk/activities/machlearn/mutagenesis.html
88. UCI Repository: UC Irvine machine learning repository. [citado 11 de noviembre de 2014]. Disponible en Internet: http://archive.ics.uci.edu/ml/

Anexos

A. Algoritmo de discretización Chi2

Para discretizar los datos numéricos es posible utilizar el algoritmo Chi2 [1] (ver algoritmo 1). Chi2 es un algoritmo de discretización [2] basado en el estadístico χ^2. Cada atributo $a \in A$ está asociado con un nivel de significación $\alpha[a]$, donde A es el conjunto de todos los atributos numéricos de discretización. El nivel de significación $\alpha[a]$ definido al inicio es de $0,5$ para cada atributo (ver algoritmo 1, líneas 2-3).

Algoritmo 1: Algoritmo Chi2 (D, A, $\bar{D}$)

Entrada: D - conjunto de datos, A - conjunto de atributos numéricos de discretización.
Salida: $\bar{D}$ - conjunto de datos discretizados.

```
1  D̄ ← D
2  forall a in A do
3  |  α[a] ← 0,5
4  end
5  while se pueda combinar un atributo de A do
6  |  foreach a combinable do
7  |  |  Ordenar(a, D̄) //ordena los datos para un atributo
8  |  |  chi_sq_ini (a, D̄) //prepara los datos del atributo para calcular χ²
9  |  |  chi_sq_cal (a, D̄) //calcula χ²
10 |  |  while Combinar(D̄) == true do
11 |  |  |  chi_sq_cal (a, D̄)
12 |  |  end
13 |  |  if Inconsistencia(D̄) < δ then
14 |  |  |  DecreNivSig ← decrementa el nivel de significancia
15 |  |  |  DecreNivSig(α[a])
16 |  |  else
17 |  |  |  a no es combinable
18 |  |  end
19 |  end
20 end
21 return D̄
```

Chi2 comienza ordenando los atributos según sus valores (ver algoritmo 1, línea 7). Luego se calcula el valor de χ^2 para cada par de intervalos adyacentes (ver algoritmo 1, líneas 8-9). El par de intervalos adyacentes con menor valor de χ^2 se combinan. En el proceso de combinar es donde se discretizan los atributos numéricos, es decir varios valores numéricos pasan a ser representados por un valor discreto (ver algoritmo 1, líneas 10-12). Se continúa combinando hasta que los valores χ^2 de todos los pares del intervalo excedan $\alpha[a] = 0,5$. El proceso se repite con una disminución de $\alpha[a]$ hasta que se supera un umbral de inconsistencia δ en los datos discretizados (ver algoritmo 1, líneas 6-19). El umbral δ es determinado por la inconsistencia de los datos, normalmente tiene el valor de 0.

La comprobación de inconsistencia tiene lugar después de la combinación de cada atributo (ver algoritmo 1, líneas 13-18). Si no se detecta inconsistencia, se decrementa $\alpha[a]$ para la próxima

combinación del atributo a. En caso de detectarse inconsistencia el atributo a no se tiene en cuenta para la próxima combinación. Este proceso continúa hasta que no exista atributo que sus valores se puedan combinar (ver algoritmo 1, líneas 5-20).

Al final, si un atributo queda combinado en solo un valor, significa que dicho atributo no es necesario para representar el conjunto de datos original. Como resultado al terminar la discretización, la selección de características también se lleva a cabo. La salida de Chi2 contiene los datos discretizados con un contador de frecuencia para cada patrón no duplicado.

B. Intervalo difuso

Un intervalo difuso se especifica mediante cuatro parámetros $\tilde{I} = (\check{\phi}, \check{\varphi}, \hat{\varphi}, \hat{\phi})$ y se define para un número real v como se muestra en la ecuación 7.

$$\tilde{I}(v) = \begin{cases} 1 & \text{si } \check{\varphi} \leq v \leq \hat{\varphi} \\ \frac{v-\check{\phi}}{\check{\varphi}-\check{\phi}} & \text{si } \check{\phi} < v < \check{\varphi} \\ \frac{\hat{\phi}-v}{\hat{\phi}-\hat{\varphi}} & \text{si } \hat{\varphi} < v < \hat{\phi} \\ 0 & \text{en otro caso} \end{cases} \tag{7}$$

Las variables $\check{\varphi}$ y $\hat{\varphi}$ representan el límite central inferior y superior respectivamente del conjunto difuso; mientras $\check{\phi}$ y $\hat{\phi}$ representan el límite del soporte inferior y superior respectivamente (ver figura 1).

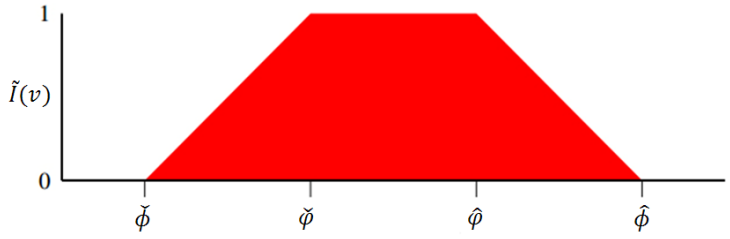

Fig. 1. Intervalo difuso.

C. Ganancia de información

En el contexto de los árboles de decisión, la ganancia de información [3] $G(m)$ para una condición m viene dada por:

$$G(m) = \frac{\left(-\sum_{l=1}^{k} \frac{|D^{c_l}|}{|D|} \cdot log_2\left(\frac{|D^{c_l}|}{|D|}\right)\right) - \left(\sum_{i=1}^{\hat{m}} \frac{|D_i|}{|D|} \cdot \left(-\sum_{l=1}^{k} \frac{|D_i^{c_l}|}{|D_i|} \cdot log_2\left(\frac{|D_i^{c_l}|}{|D_i|}\right)\right)\right)}{-\sum_{i=1}^{\hat{m}} \frac{|D_i|}{|D_i|} \cdot log_2\left(\frac{|D_i|}{|D_i|}\right)}, \tag{8}$$

donde k es un número entero que representa la cantidad de clases, D^{c_l} es el conjunto de datos de entrenamiento etiquetados con la clase c_l, D representa el conjunto de datos de entrenamiento y D_i es el i-ésimo subconjunto de entrenamiento resultante de particionar a D según la cantidad de posibles resultados $\hat{m}$ de la condición m.

El criterio de relación de ganancia expresa la proporción de la información generada como resultado de la división del conjunto de entrenamiento. Dicho criterio evalúa las condiciones y provee un mecanismo para dar un orden al conjunto de condiciones propuesto, dando la posibilidad de que pueda escogerse la condición aparentemente más favorable.

D. Ganancia Gini

La ganancia Gini en el contexto de los árboles de decisión [4], es utilizada como criterio para evaluar cual división en un nodo resulta más óptima para particionar un conjunto de datos D. Dado un nodo, el criterio para particionar D, es definido por la sumatoria de los índices Gini de los nodos sucesores ϱ (ver ecuación 9) y la ganancia Gini $\hat{\varrho}$ (ver ecuación 10) de la división.

$$\varrho = \sum_{l=1}^{d} \frac{\mid D_l \mid}{\mid D \mid} \left(1 - \sum_{i=1}^{k} \left(\frac{\mid D_l^{c_i} \mid}{\mid D_l \mid}\right)^2\right), \tag{9}$$

$$\hat{\varrho} = \left(1 - \sum_{l=1}^{d} \left(\frac{\mid D_l \mid}{\mid D \mid}\right)^2\right) - \varrho, \tag{10}$$

donde la variable $\mid D_l^{c_i} \mid$ representa la cantidad de instancias en el conjunto D_l que pertenecen a la clase c_i, la variable k es la cantidad de clases y la variable d indica la cantidad de divisiones de D. Veamos el siguiente ejemplo para comprender mejor como se utiliza el índice Gini. Supongamos que se está ejecutando el algoritmo sobre el nodo raíz para determinar la división más óptima de D, que está compuesta por nueve instancias de la clase fraude y cinco instancias de la clase no fraude(ver figura 2).

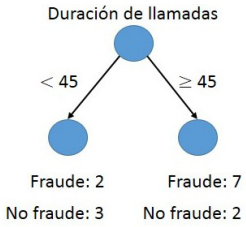

Fig. 2. Ejemplo de posible división de un conjunto de datos.

Teniendo en cuenta la cantidad de instancias que llegan a los nodos sucesores, el índice Gini de la división del conjunto y su ganancia Gini se calculan como:

$$\varrho = \frac{5}{14} \cdot \left(1 - \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2\right) + \frac{9}{14} \cdot \left(1 - \left(\frac{7}{9}\right)^2 - \left(\frac{2}{9}\right)^2\right) = 0{,}3937,$$

$$\hat{\varrho} = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 - \varrho = 0{,}3937.$$

De forma similar se calcula la ganancia Gini para cada posible división en el nodo raíz [5]. Aquella división que maximice el valor de $\hat{\varrho}$ es la seleccionada en el nodo para continuar construyendo el árbol.

Referencias bibliográficas del anexo

1. Liu, H., Setiono, R.: Chi2: Feature selection and discretization of numeric attributes. In: 2012 IEEE 24th International Conference on Tools with Artificial Intelligence, IEEE Computer Society (1995) 388–388
2. Liu, H., Setiono, R.: Discretization of ordinal attributes and feature selection. Technical Report, Department of Informatic Systems and Computer Science, National University of Singapore, Singapore (1995)
3. Xiaoliang, Z., Jian, W., Hongcan, Y., Shangzhuo, W.: Research and application of the improved algorithm c4. 5 on decision tree. In: Test and Measurement, 2009. ICTM'09. International Conference on. Volume 2., IEEE (2009) 184–187
4. Lawrence, R.L., Wright, A.: Rule-based classification systems using classification and regression tree (cart) analysis. Photogrammetric engineering and remote sensing **67**(10) (2001) 1137–1142
5. Shi, H.: Best-first decision tree learning. PhD thesis, Citeseer (2007)

RT_029, enero 2015
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. A No. 21406 e/214 y 216, Rpto. Siboney, Playa;
La Habana. Cuba. C.P. 12200
Impreso en Cuba