Unidad Iztapalapa
División de Ciencias Básicas e Ingeniería
Posgrado en Ciencias (Ciencias y Tecnologías de la Información)

"Enriquecimiento del conocimiento previo en ILP"
TESIS
Que para obtener el grado de
DOCTOR EN CIENCIAS (CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN)
PRESENTA:
M.C. Orlando Muñoz Texzocotetla
2113802441
magicorlan@gmail.com
ORCID: 0000-0002-3278-4730
DIRECTOR DE TESIS:
Dr. René Mac Kinney Romero
JURADO:
Presidente:
Dr. Eduardo Morales Manzanares
Secretario:
Dr. René Mac Kinney Romero
Vocal:
Dr. Miguel Ángel Gutiérrez Andrade
Vocal:
Dr. Pedro Lara Velázquez
Vocal:
Dr. Mario Graff Guerrero
Iztapalapa, Ciudad de México a 6 de marzo de 2023

Resumen

La *Programación Lógica Inductiva (ILP)* induce conceptos a partir de un conjunto de ejemplos negativos, un conjunto de ejemplos positivos, y un conocimiento previo. La ILP ha sido utilizada en tareas dentro de diversas áreas como procesamiento de lenguaje natural, diseño de malla para elementos finitos, minería de redes, robótica, descubrimiento de nuevas drogas en farmacéutica, etc. Los conjuntos de datos utilizados, usualmente, contienen atributos numéricos y categóricos, sin embargo, son pocos los sistemas de aprendizaje relacional que son capaces de manejar ese tipo de datos de manera eficiente. Esta tesis presenta un método evolutivo, llamado "*Agrupamiento y discretización para el enriquecimiento del conocimiento previo" (en inglés Grouping and Discretization for Enriching the Background Knowledge - GDEBaK)*, el cual, permite manejar atributos numéricos y categóricos. Este método utiliza operadores evolutivos para crear y probar diferentes puntos de división (en el caso de los atributos numéricos) y subconjuntos de valores (en el caso de los atributos categóricos) de acuerdo a una función de aptitud. Después, los mejores puntos de división y subconjuntos de categorías son agregados al conocimiento previo, antes de realizar el proceso de aprendizaje, para que sean utilizados durante la construcción de la teoría final. Implementamos GDEBaK embebido en el sistema *Aleph* y lo comparamos con la discretización perezosa del sistema Aleph y con la discretización llevada a cabo por el sistema *Top-down Induction of Logical Decision Trees (TILDE)* [6]. Aleph es uno de los sistemas de ILP más usados para aprender conceptos que necesitan gran poder de representación [55], y es uno de los más importantes en ILP ya que posee funcionalidades de otros sistemas como Progol, FOIL, FORS o TILDE. Los resultados obtenidos muestran que el método presentado mejora la precisión de las teorías finales y reduce el número de reglas en la mayoría de los casos.

Palabras clave: atributos categóricos, discretización, agrupamiento, ILP, atributos numéricos.

II

Abstract

Inductive Logic Programming (ILP) induces concepts from a set of negative examples, a set of positive examples, and background knowledge. ILP has been applied on tasks in areas such as natural language processing, finite element mesh design, network mining, robotics, drug discovery, and more. These datasets typically contain both numerical and categorical attributes; however, few relational learning systems efficiently handle such data. This thesis introduces an evolutionary method called "Grouping and Discretization for Enriching the Background Knowledge (GDEBaK)," which enables the handling of numerical and categorical attributes. This method employs evolutionary operators to create and test different split points (for numerical attributes) and subsets of values (for categorical attributes) based on a fitness function. Subsequently, the best split points and category subsets are added to the background knowledge before the learning process, to be used during the induction of the final theory. We implemented GDEBaK embedded in the Aleph system and compared it with Aleph's lazy discretization and the discretization performed by the Top-down Induction of Logical Decision Trees (TILDE) system [6]. Aleph is one of the most widely used ILP systems for learning concepts that require high representational power [55]. It is crucial in ILP, incorporating functionalities from other systems such as Progol, FOIL, FORS, or TILDE. The obtained results indicate that the presented method improves the accuracy of final theories and reduces the number of rules in the majority of cases.

keywords: categorical attributes, discretization, grouping, ILP, numerical attributes.

Agradecimientos

Agradezco sinceramente a mi asesor el Dr. René Mac Kinney Romero, por su paciencia, valiosa orientación y apoyo constante en cada uno de los aspectos del presente trabajo.

Extendidos agradecimientos a los distinguidos profesores Dr. Eduardo Morales Manzanares, Dr. Miguel Ángel Gutiérrez Andrade, Dr. Pedro Lara Velázquez y Dr. Mario Graff Guerrero, cuyos comentarios e ideas han enriquecido significativamente mi tesis.

Expreso mi profundo agradecimiento a la Universidad Autónoma Metropolitana por proporcionarme las oportunidades necesarias para completar mi preparación académica desde la licenciatura hasta el doctorado.

Finalmente, deseo expresar mi gratitud al Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCyT) por el respaldo invaluable y los recursos brindados, los cuales han sido fundamentales para llevar a cabo con éxito mi programa de doctorado.

Dedicado a...

A mi madre Cecilia, ya que gracias a su cariño y apoyo incondicional he logrado superar cada reto que me he impuesto.

A mi padre Juan Manuel, por su comprensión y paciencia, y por darme la oportunidad de continuar y concluir mis estudios.

Y a mis hermanos Juan Manuel, Francisco Javier, Alejandro, Manuel, Miriam y Marco Antonio, que de diversas formas me han apoyado, no solo en mi preparación académica sino en otros aspectos de mi vida.

Contenido

Lista de Figuras

Lista de Tablas

Lista de algoritmos

Capítulo 1

Introducción

> "Nunca sabes cuan fuerte eres hasta que ser fuerte es la única elección que tienes. Y cuando esto sucede, eres intocable."
>
> Chuck Palahaniuk

El *aprendizaje maquinal - AM*, junto con el procesamiento del lenguaje natural, la representación del conocimiento y el razonamiento automático, es uno de los campos más importantes de la inteligencia artificial, ya que está considerado dentro de las capacidades que debe tener una computadora para pasar *la prueba de Turing* y poder decir que tiene inteligencia [60]. El AM consiste en el diseño e implementación de algoritmos capaces de adaptarse a un entorno cambiante y de identificar patrones. La definición de AM más aceptada es la que nos proporciona Tom Mitchell: "Se dice que un programa aprende de la experiencia E respecto a una tarea T, y una medida de rendimiento P, si su rendimiento al realizar la tarea T, medido por P, mejora con la experiencia E" [47]. Las aplicaciones de los algoritmos de AM son muy variadas, ya que estas pueden ser diagnóstico médico, filtración de correo no deseado, detección de rostros, detección de fraudes, diseño de nuevos fármacos, robótica, etc.

Dependiendo del tipo de tarea, el AM se divide principalmente en tres tipos: el *aprendizaje predictivo* o *supervisado* consiste en inducir una hipótesis o modelo que mapee un conjunto de ejemplos (conjunto de entrenamiento) a una variable objetivo. Si la variable objetivo es nominal entonces nos referimos a una tarea de clasificación y si la variable objetivo es de tipo real entonces estamos hablando de una tarea de regresión. La hipótesis inducida es usada para predecir la clase o el valor numérico de ejemplos no utilizados durante el aprendizaje. El segundo tipo de aprendizaje es el *descriptivo* o *no supervisado*

para el cual no existe una variable objetivo a la cual mapear los ejemplos, en este caso el propósito es inducir los patrones que describan grupos de ejemplos con características o distribuciones similares. Los algoritmos de agrupamiento son ejemplos de este tipo de aprendizaje. El tercer tipo es el *aprendizaje por reforzamiento* en el cual el aprendizaje no está guiado por una variable objetivo. En este caso se le indica al algoritmo si está realizando bien su tarea o no, de esta forma se busca maximizar un valor numérico que representa una recompensa.

Por otro lado, desde el punto de vista del formato del conjunto de entrenamiento, las técnicas de AM se pueden clasificar en dos grupos:

Aprendizaje proposicional. En este tipo de aprendizaje, también llamado *atributo-valor*, los ejemplos se representan en una sola tabla, y la hipótesis inducida se interpreta por medio de la lógica proposicional. En este cada, ejemplo se representa por un conjunto fijo de características llamadas *atributos*. Con este tipo de representación el conjunto ejemplos se puede ver como una colección de pares atributo-valor. La Tabla 1.1 muestra un conjunto de entrenamiento donde cada ejemplo representa un cliente bancario. A a su vez, cada ejemplo está caracterizado por su nombre, género, edad, balance bancario, y número de tarjetas de crédito [34].

Nombre	**Género**	**Edad**	***Balance***	***Tarjetas de crédito***
Ritchie	Masculino	43	4,000	2
Rendell	Femenino	12	-2,000	0
Gross	Femenino	81	85,000	0
Ferguson	Masculino	62	26,000	1
Smith	F	27	1,000	3

Tabla 1.1: Ejemplo de un conjunto de entrenamiento para aprendizaje proposicional.

Otras características de este tipo de aprendizaje son: cada ejemplo tiene el mismo número de atributos y puede verse como un punto en un espacio de n dimensiones (donde n es el número de atributos). Algunos algoritmos de aprendizaje proposicional son: árboles de decisión, k-vecinos más cercanos, regresión lineal, máquinas de soporte vectorial, etc.

Aprendizaje relacional. Este se caracteriza por la alta complejidad que puede tener cada ejemplo y porque además, los ejemplos pueden estar relacionados entre sí. A diferencia del proposicional, en el aprendizaje relacional los ejemplos son objetos estructurados que se componen de partes que pueden estar conectadas de diferentes formas.

Uno de los ejemplos más conocidos de este tipo de aprendizaje es el de los trenes este-oeste de Michalski [45]. En este la tarea de aprendizaje es clasificar los trenes de acuerdo a la dirección que siguen: este u oeste. El conjunto de entrenamiento se compone de diez

trenes, cinco de los cuales va al este y el resto al oeste, ver Figura 1.1. Por otro lado, cada tren está formado por una serie de carros cuyas características son:

- Longitud: largo o corto.
- Forma: elipse, rectángulo abierto, rectángulo cerrado, etc.
- Cargamento: círculo, triángulo, cuadrado, rectángulo o rombo.
- Número de cargamento: 1, 2 o 3.
- Número de llantas: 2 o 3.

Este problema es de tipo relacional porque cada ejemplo tiene una estructura compleja a diferencia de un ejemplo proposicional. Es decir, cada tren se compone de uno o más vagones, y a su vez cada vagón tiene las características mencionadas anteriormente. En este caso dos o más vagones están relacionados entre sí cuando pertenecen al mismo tren.

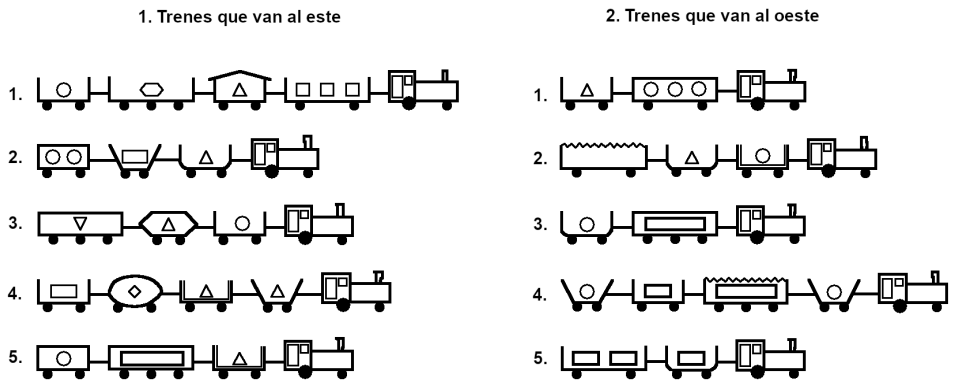

Figura 1.1: Los trenes este-oeste de Michalski es un ejemplo de aprendizaje relacional.

Así que en lugar de poner esta información en una tabla atributo-valor es necesario un lenguaje que permita describir cada ejemplo de manera adecuada sin perder información. Un lenguaje apropiado para esto es Prolog que está basado en la lógica de primer orden. Por ejemplo, si queremos expresar que el tren número 1 que va al este tiene dos vagones cortos y dos largos escribir los hechos mostrados en la Figura1.2:

Después, cada característica de los vagones (si está abierto o cerrado, número de llantas, etc.) también puede ser descrita por medio de hecho o reglas, ver Figura 1.3

```
al_este(tren1).      % Tren 1 que va al este
tiene_vagon(tren1, vagon_10).
tiene_vagon(tren1, vagon_11).
tiene_vagon(tren1, vagon_12).
tiene_vagon(tren1, vagon_13).

corto(vagon_10).
corto(vagon_11).
largo(vagon_12).
largo(vagon_13).
```

Figura 1.2: Estos hechos describen que el tren 1 va al este, y también indican cuáles son los vagones que pertenecen a dicho tren.

```
num_llantas(vagon_10, 2).
num_llantas(vagon_11, 3).
num_llantas(vagon_12, 2).
num_llantas(vagon_13, 2).

abierto(vagon_10).
abierto(vagon_11).
abierto(vagon_13).
cerrado(vabon_12).
...
```

Figura 1.3: Descripción en Prolog de los vagones del tren 1.

1.1. Motivación

La presente tesis está ubicada dentro del marco de la ILP, la cual está definida como la intersección entre el aprendizaje maquinal y la programación lógica [50]. De esta manera los programas o aprendices ILP llevan a cabo la inducción de una teoría T (conjunto de reglas o hipótesis) a partir de un conjunto de ejemplos positivos E^+, un conjunto de ejemplos negativos E^-, y de información del dominio del problema llamada *conocimiento previo* B. Tanto la teoría T como los conjuntos de ejemplos E^+, E^- y el conocimiento previo B están representados en el lenguaje de la lógica de primer orden.

La tarea de la ILP, como en cualquier aprendizaje inductivo, puede ser vista como un proceso de búsqueda, en el cual, el objetivo es encontrar la mejor teoría posible [46]. Esta tarea es llevada a cabo por un algoritmo o aprendiz descrito principalmente por tres características: la *estrategia de búsqueda* define cómo explorar el espacio de búsqueda para

encontrar la hipótesis. La *heurística* es el criterio que permite decidir cuál es el camino más prometedor, dentro del espacio de búsqueda, para encontrar la hipótesis que se busca. La tercer característica es el *espacio de búsqueda* el cual es el conjunto de todas las posibles hipótesis que pueden ser inducidas. En ILP el espacio de búsqueda está determinado por el lenguaje de la lógica de primer orden y por el sesgo de lenguaje. Este último es un mecanismo que restringe el espacio de búsqueda [38], y en el contexto de la ILP está definido a partir de un vocabulario de símbolos de predicados, de funciones, de variables y de constantes.

La motivación del presente trabajo radica en el tercer punto (espacio de búsqueda), del cual reflexionamos lo siguiente: cuando el espacio de búsqueda tiene un lenguaje muy restringido se vuelve más pequeño, por lo tanto es más fácil para el aprendiz realizar la búsqueda, sin embargo es más probable que la hipótesis inducida no explique de manera correcta el concepto objetivo. Por otro lado, si el lenguaje no se restringe el espacio de búsqueda crecerá de manera exponencial haciendo imposible la búsqueda dentro de ese conjunto de hipótesis. La mayoría de los algoritmos ILP definen un sesgo de lenguaje que provoca el trato de cada argumento como un atributo univalor ya que al inducir una hipótesis estos algoritmos prueban por argumento un solo valor constante a la vez. Esta restricción en el lenguaje puede afectar tanto a la calidad de las teorías inducidas (precisión y expresividad) como a la eficiencia de los algoritmos ILP (crecimiento exponencial del espacio de búsqueda y por lo tanto del tiempo de ejecución). Por ejemplo, si el objetivo es inducir con un sesgo de lenguaje convencional una teoría que explique de manera general los casos en que una persona es adulta tomando en cuenta su edad, obtendremos un conjunto de hipótesis como la siguiente[1]:

```
adulto(A) :- edad(A,19).
```

La cláusula anterior es una regla que indica que una persona A es un adulto si tiene 19 años. Evidentemente una teoría que explique de manera general el concepto de "adulto" necesitará muchas reglas como la anterior. De hecho una regla por cada persona mayor a 18. Al final tendremos una teoría poco expresiva ya que tendrá una gran cantidad de reglas y por lo tanto será difícil de interpretar. Además no debemos descartar el caso en que la teoría inducida no tenga una regla para ciertas edades, en este caso la teoría no será precisa.

La problemática anterior también se da con los atributos categóricos multivalor (que tienen muchas categorías). Supongamos que un sistema ILP construyó las siguientes seis hipótesis para determinar si una película le gusta a una persona A tomando en cuenta su ocupación y el género de la película. Como podemos apreciar en la Figura 1.4 se construye

[1] Se utiliza la notación del lenguaje Prolog para representar hechos, cláusulas y literales.

una regla por cada género, y si dicho género le gusta a personas de distintas ocupaciones, entonces se crea una regla por cada ocupación. Esto sucede en el caso de las películas de *terror* ya que se crean dos reglas con este género. Evidentemente entre más posibles valores tenga el segundo argumento de los predicados *ocupacion/2* y *genero/2* más hipótesis serán inducidas en la teoría final, haciéndola más difícil de interpretar.

```
leGusta(A,Peli):- ocupacion(A, estudiante ), genero( Peli , terror ).
leGusta(A,Peli):- ocupacion(A, estudiante ), genero( Peli , cienciaFiccion ).
leGusta(A,Peli):- ocupacion(A, estudiante ), genero( Peli , accion ).

leGusta(A,Peli):- ocupacion(A,empleado),genero( Peli , terror ).
leGusta(A,Peli):- ocupacion(A,empleado),genero( Peli ,comedia).
leGusta(A,Peli):- ocupacion(A,empleado),genero( Peli , gore ).
```

Figura 1.4: Ejemplo de teoría que indica que a una persona A le gustará una película de terror, ciencia ficción o acción si es un estudiante. También describe que a una persona A le gustará una película de terror, comedia o gore si es un empleado.

En este punto podemos preguntarnos lo siguiente:

- ¿Cómo enriquecer el conocimiento previo con la información dada en los argumentos numéricos y categóricos?
- Dicho enriquecimiento ¿Mejorará la precisión y la expresividad de las teorías inducidas?
- ¿Es posible hacer esto en cualquier sistema ILP sin importar si es *top-down* o *bottom-up*?

Para responder a estas preguntas en este trabajo se propone el uso un algoritmo genético para encontrar los mejores puntos de división para cada argumento numérico y los mejores subconjuntos de categorías para cada argumento categórico. Dicha información junto con predicados auxiliares se agrega al conocimiento previo antes de iniciar el proceso de inducción. Al hacer esto el sesgo de lenguaje permite un conjunto de predicados más rico con el cual se pueden inducir hipótesis más precisas y de ser posible más expresivas. Como ejemplo de esto, las reglas mostradas en la Figura 1.5 describen los conceptos mencionados anteriormente pero con un lenguaje menos restringido, el cual incluye el uso de desigualdades (en el problema de la mayoría de edad), de listas de valores categóricos (en el problema de gustos cinematográficos). Sin embargo el uso de estos predicados adicionales y de nuevos valores numéricos en la inducción de hipótesis no es trivial, y de hecho

constituye una problemática en la ILP, ya que como se mencionó anteriormente un lenguaje con pocas restricciones puede provocar un espacio de hipótesis imposible de explorar en un tiempo razonable. Además los valores numéricos (como el punto de división para la mayoría de edad o los coeficientes en el caso de la expresión lineal) y los subconjuntos de valores categóricos (en el caso de gustos cinematográficos) agregados al lenguaje deben ser determinados con diversos métodos que tomen en cuenta la naturaleza relacional de los datos.

```
adulto(A) :- edad(A,Edad), Edad > 18.


leGusta(A,Peli):-
    ocupacion(A, estudiante ), genero( Peli ,[ terror , cienciaFiccion , accion ]).
leGusta(A,Peli):-
    ocupacion(A,empleado),genero( Peli ,[ terror ,comedia,gore ]).


administrar (P) :-
    presion(P, Sist , Diast ), temp(P,Temp),0.12*Sist-.121*Diast+1.2*Temp<0.
```

Figura 1.5: Ejemplos de reglas con manejo de atributos numéricos y categóricos.

1.2. Objetivo principal

El principal objetivo de esta tesis es diseñar e implementar en ILP métodos de análisis univariable sobre atributos numéricos y categóricos que mejoren la calidad de las teorías inducidas, tanto en precisión como en número de hipótesis.

1.2.1. Objetivos particulares

- Realizar una revisión del estado del arte de los métodos de análisis univariable para atributos numéricos y categóricos en aprendizaje proposicional y en ILP.

- Diseñar métodos univariable para el manejo de atributos numéricos y categóricos que potencialmente puedan mejorar de manera significativa las teorías creadas por los sistemas ILP.

- Probar si con el uso de los métodos diseñados e implementados en ILP se obtienen mejores teorías: más precisas y con menor número de reglas (más fáciles de entender).

1.3. Organización del documento

El resto de este documento está organizado de la siguiente manera. En el capítulo 2 presentamos los conceptos más importantes que manejamos en el presente trabajo: aprendizaje proposicional, aprendizaje relacional, programación lógica inductiva, discretización, agrupamiento, proposicionalización, programación lógica de restricciones y algoritmos genéticos. En el capítulo 3 repasamos las principales estrategias usadas en ILP para el manejo de valores numéricos y categóricos. En el capítulo 4 detallamos nuestra propuesta implementada en Aleph. El resultado de los experimentos llevados a cabo para comparar nuestra estrategia con la discretización perezosa en Aleph y con la discretización en TILDE se presentan en el capítulo 5. Finalmente, en el capítulo 6 presentamos nuestras conclusiones y la línea a seguir para trabajo futuro en el manejo de atributos numéricos y categóricos.

Capítulo 2

Conceptos Básicos

> "Quien quiere hacer algo encuentre un medio, quien no quiere hacer nada encuentra una excusa."
>
> Proverbio árabe

En este capítulo se presentan los fundamentos básicos de aprendizaje maquinal, aprendizaje proposicional, aprendizaje relacional, y con más detalle de la programación lógica inductiva. Estos temas nos permitirán de manera general ubicar la tesis dentro del aprendizaje maquinal. Además se revisan los conceptos de discretización, agrupamiento, análisis discriminante, proposicionalización, programación lógica de restricciones y algoritmos genéticos, los cuales son fundamentales en el trabajo realizado para esta tesis. Estos temas no se revisan a profundidad, sin embargo son relevantes para la comprensión de los sistemas ILP y sus estrategias para el manejo de atributos numéricos y categóricos presentados más adelante.

2.1. Aprendizaje Proposicional y Relacional

Como se mencionó anteriormente, dependiendo del formato del conjunto de ejemplos, el aprendizaje maquinal se puede clasificar principalmente en dos tipos: el proposicional y el relacional.

El *aprendizaje proposicional*, también llamado *aprendizaje atributo-valor*, es cualquier tarea de aprendizaje en la que cada ejemplo está descrito como un vector de valores, y cada posición en el vector corresponde a un único atributo. El conjunto total de ejemplos forma una tabla en la que cada renglón corresponde a un único ejemplo, cada columna

representa un atributo (propiedad o variable), y las entradas de esta tabla son los valores para los atributos definidos por cada ejemplo. En la tabla 2.1 se presenta un fragmento del conjunto de ejemplos para el problema de clasificación de plantas [22]. Como podemos observar este conjunto de ejemplos tiene cinco atributos, de los cuales *Especie* es el atributo clase.

Largo Sépalo	**Largo Pétalo**	**Ancho Sépalo**	**Ancho Pétalo**	**Especie**
5.4	3.5	1.4	0.2	Setosa
4.9	3.0	1.4	0.2	Setosa
4.9	3.1	1.5	0.1	Setosa
7.0	3.2	4.7	1.4	Versicolor
6.4	3.2	4.5	1.5	Versicolor
6.9	3.1	4.9	1.5	Versicolor
6.3	3.3	6.0	2.5	Virgínica
5.8	2.7	5.1	1.9	Virgínica
7.1	3.0	5.9	2.1	Virgínica

Tabla 2.1: Fragmento del conjunto de datos del problema de clasificación de Iris.

El objetivo de un algoritmo proposicional es encontrar una hipótesis que clasifique este conjunto de ejemplos en una de las tres clases definidas. Una regla o hipótesis podría ser como la siguiente:

```
Si Ancho de Pétalo <= 0.6 Entonces Especie = Virgínica
```

Otro ejemplo es la hipótesis inducida por un generador de árboles de decisión la cual puede ser interpretada en el lenguaje de la lógica proposicional como una conjunción de disyunciones. En la Figura 2.1 se muestra un árbol de decisión para el mismo conjunto de datos.

El *aprendizaje relacional* es aquella tarea en la cual los aprendices buscan hipótesis a partir de un conjunto de datos donde los ejemplos tienen una estructura compleja (relación interna) o donde los ejemplos están relacionados entre sí (relación externa). Este tipo de aprendizaje es motivado principalmente por las limitaciones del aprendizaje proposicional.

El problema de minería de datos en la *movie internet dataset - imbd (www.imdb.com)* es un ejemplo de un problema de aprendizaje con relaciones externas (relaciones entre ejemplos). En esta base de datos cada película está caracterizada por un número fijo de atributos: año, país, ganancias, director. Sin embargo, los ejemplos están relacionados entre sí a través de otras entidades. Por ejemplo, distintas películas pueden haber sido dirigidas por el mismo director. En este caso la tarea de aprendizaje podría consistir en la

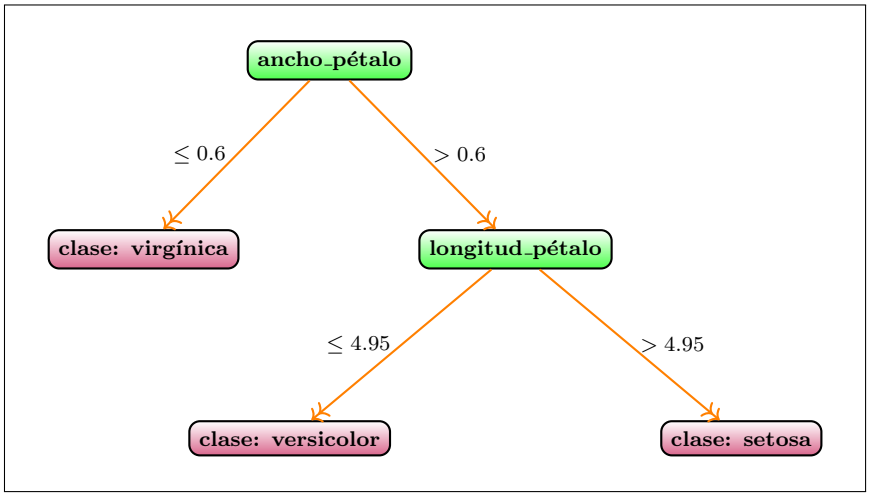

Figura 2.1: Árbol de decisión que clasifica el conjunto de datos Iris.

predicción de ganancias tomando en cuenta el director de la película.

La búsqueda de patrones en compuestos químicos es un ejemplo de aprendizaje relacional donde los ejemplos tienen relaciones internas, ya que la estructura de cada compuesto químico puede variar con respecto a la de otro. En este caso la complejidad interna está data por los componentes del compuesto: átomos y el tipo de enlace que puede haber entre los átomos.

Tratar con tareas de aprendizaje relacional no es trivial, ya que desde la representación del conjunto de ejemplos hasta la hipótesis final se tiene una tarea compleja. Algunos de los métodos desarrollados para tratar con estos tipos de datos relacionales son: el aprendizaje a partir de grafos, la minería de datos multi-relacional, el aprendizaje relacional estadístico y probabilista, el aprendizaje relacional por reforzamiento y la programación lógica inductiva - ILP. Ya que nuestro trabajo está ubicado en la ILP, lo describimos con más detalle a continuación.

2.2. Programación Lógica Inductiva

La programación lógica inductiva o *Inductive Logic Programming - ILP*, es un paradigma dentro del aprendizaje relacional y está definida como la intersección entre el aprendizaje maquinal y la programación lógica [50]. El principal objetivo de la ILP es inducir una teoría T a partir de un conjunto de ejemplos positivos E^+, de un conjunto de ejemplos negativos E^-, y de un conocimiento previo B. La teoría, los ejemplos y el conocimiento previo son representados como programas lógicos, los cuales, están basados

en el lenguaje de la lógica de primer orden. Formalmente, esta tarea se puede describir en términos de lógica de primer orden como sigue:

Dados: un conjunto finito de cláusulas B, un conjunto finito de cláusulas E^+, y un conjunto finito de cláusulas E^-.

Encontrar: una teoría T (conjunto de hipótesis o reglas), tal que $B \wedge T \models E^+$ (completa) y $B \wedge T \not\models E^-$ (consistencia). Si T es completa y consistente entonces T es correcta. Debe aclararse que T puede no ser precisa para ejemplos no usados en la inducción de T.

Como se mencionó en la sección 1.1, el espacio de búsqueda es esencial para describir a un aprendiz o algoritmo ILP. Para definir el espacio de búsqueda necesitamos estructurarlo, y para ello es fundamental determinar una relación de generalidad entre las cláusulas de programa y un lenguaje, basado en el de la lógica de primer orden, para construir cada cláusula de programa.

En la mayoría de los algoritmos ILP, el orden de generalidad viene dado por el concepto de $\theta-$*subsunción*, el cual introduce un orden parcial entre las cláusulas, la Definición 2.1 describe formalmente esta relación.

Definición **2.1.** *Subsunción. Sean c y d dos cláusulas de programa. Decimos que c $\theta-$*subsume *a d si existe una sustitución θ tal que $c\theta \subseteq d$ [52]. Esta relación es denotada como $c \preceq d$.*

De esta manera decimos que si una cláusula c $\theta-$*subsume* a una cláusula d, denotado por $c \preceq d$, entonces c es una cláusula más general que d. Este cuasi-orden dado por la $\theta-$*subsunción* estructura el espacio de búsqueda de tal manera que posible explorarlo para realizar la búsqueda de cláusulas. Gráficamente el espacio de búsqueda ordenado puede ser visto gráficamente como un diagrama de *Hasse* o retícula, en el cual, las cláusulas más generales están arriba y las más específicas están abajo, ver Figura 2.2.

Para buscar las hipótesis, en este caso cláusulas, que formarán parte de la teoría final, algunos algoritmos exploran la retícula a partir de las cláusulas más generales, es decir, de arriba hacia abajo. Estos algoritmos son de tipo *Top-Down*. Otros algoritmos recorren la retícula de abajo hacia arriba. Estos últimos algoritmos son de tipo *Bottom-Up*. También existen algoritmos ILP que combinan ambos tipos de recorrido.

Por otro lado el lenguaje del espacio de búsqueda está determinado por el lenguaje de los programas lógicos, basado a su vez en el de la lógica de primer orden, formados por cláusulas de programa de la forma *Head*$\leftarrow$*Body*, donde *Head* es un átomo de la forma $p(X_1, X_2, \ldots, X_n)$ y *Body* es una conjunción de literales $L_1, L_2, \ldots, L_m$. El vocabulario de los símbolos de predicados L_i está dado por los predicados definidos en el conocimiento previo. El sesgo de lenguaje restringe la forma de cada cláusula de programa a partir del vocabulario de los predicados en el conocimiento previo, de los símbolos de función y de

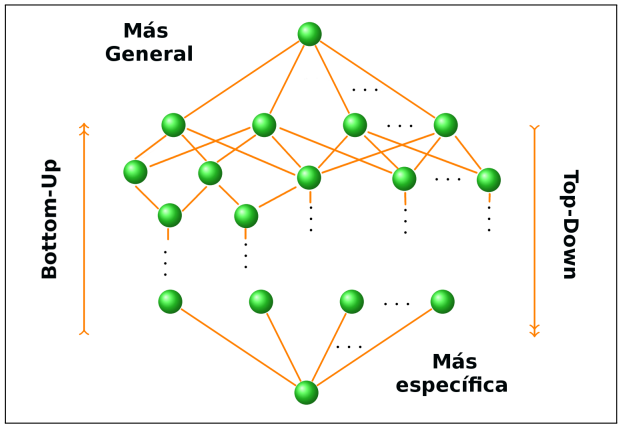

Figura 2.2: Retícula del espacio de búsqueda en ILP.

las constantes. Además si p está dentro de los predicados permitidos en *Body*, entonces la definición de predicados recursivos está permitida.

Independientemente de cómo se realice la exploración del espacio de búsqueda, la mayoría de los algoritmos ILP sigue el proceso general llamado *separar y vencer* o *algoritmo de cobertura* [7]. Esta estrategia consiste en producir un conjunto de reglas especializando repetidamente una regla más general. De manera que en cada iteración se aprende una regla que cubre un subconjunto de ejemplos positivos excluyendo a los ejemplos negativos. Este proceso continua hasta que ya no haya más ejemplos positivos que cubrir. Los argumentos de entrada son un conjunto de ejemplos positivos E^+, un conjunto de ejemplos negativos E^- y un conocimiento previo B. Este proceso es mostrado en el Algoritmo 2.1.

Algoritmo 2.1 Algoritmo ILP Genérico

```
1:  procedure ALGORITMOILPGENERICO(E+, E−, B)
2:      Teoría ← φ
3:      while E+ ≠ φ do
4:          Hipótesis ← aprende_hipótesis(E+, E−, B)
5:          Teoría ← Teoría ∪ {Hipótesis}
6:          B ← B ∪ {Hipótesis}
7:          E+ ← E+ \ {Ejemplos cubiertos por Hipótesis}
8:      end while
9:      return Teoría
10: end procedure
```

La búsqueda de hipótesis del algoritmo ILP genérico (paso 4) es el más importante

para cualquier aprendiz ILP que siga la estrategia de cobertura, ya que la manera de buscar las hipótesis determina la calidad de la teoría final. Para realizar este proceso de búsqueda, los algoritmos ILP definen operadores, los cuales toman una hipótesis H y devuelven otra modificada, que aquí denotamos como H'. Tomando en cuenta el orden de generalidad utilizado, hay dos tipos de operadores. Un *operador de generalización,* toma una hipótesis H y devuelve otra más general. Si la relación de generalizacion está dada por la $\theta-$*subsunción* entonces $H' \preceq H$. Los dos operadores de generalización más comunes son la eliminación de una literal del cuerpo de H (tomemos en cuenta que $H \equiv Head \leftarrow Body$) y la sustitución de una constante por una variable. Estos operadores son utilizados en estrategias de abajo hacia arriba *(Bottom-Up).* El otro operador es el de *refinamiento*, el cual toma una hipótesis H y devuelve otra más específica. Con la relación de $\theta-$*subsunción* tenemos que $H \preceq H'$. En este caso los dos operadores, equivalentes con los de generalización, son la adición de una literal al cuerpo de H y la sustitución de una variable por una constante. Los operadores de refinamiento son usados en estrategias de arriba hacia abajo *(Top-Down).*

2.3. Discretización

Los datos manejados por los sistemas de aprendizaje maquinal o de minería de datos son en gran parte numéricos o continuos. Debido a esto, la mayoría de los sistemas proposicionales y algunos de los aprendices ILP implementan procesos de *discretización* para poder procesar de manera eficiente cada atributo numérico. Un algoritmo de discretización divide el rango de un atributo continuo en intervalos, etiqueta cada intervalo con un valor discreto y mapea los datos originales a los valores discretos [30].

Los principales objetivos de la discretización son: reducir el número de valores de cada atributo, mejorar la calidad del conocimiento descubierto (es decir, representar el conocimiento de manera precisa, concisa y de fácil interpretación), y mejorar el rendimiento del algoritmo de aprendizaje (minimizar el tiempo de aprendizaje). Debemos aclarar que el hecho de obtener hipótesis de mejor calidad no asegura que el proceso de aprendizaje sea más rápido, sobre todo que el proceso de discretización lleva su tiempo.

Todo proceso de discretización tiene dos puntos muy importantes a tomar en cuenta: por un lado es importante determinar el número de intervalos necesarios, este por lo general lo determinan los usuarios, aunque algunos algoritmos son capaces de calcular el mejor número de intervalos posible. Por otro lado, es importante calcular el ancho de cada intervalo, el cual, es determinado por el algoritmo de discretización. En general todo proceso de discretización debe tener tres pasos esenciales:

1. *Ordenamiento.* Ordenar los valores continuos del atributo a ser discretizado. Recor-

demos que no siempre se discretizan valores continuos, p.e. se pueden discretizar valores enteros. La ordenación puede ser ascendente o descendente.

2. *División.* De acuerdo a algún criterio, dividir o unir (*split* o *merge*) los intervalos del atributo (utilizando medidas de entropía, medianas estadísticas, etc.).

3. *Criterio de paro* Finalmente detenerse en algún punto dependiendo del criterio de paro especificado. Este puede estar determinado por el tamaño de los intervalos, el número de intervalos, o por la precisión alcanzada.

Al igual que los algoritmos de aprendizaje, es posible clasificar los de discretización dependiendo de diversos factores. Algunos de estos son los siguientes:

- *Bottom-up y Top-down.* Los métodos *bottom-up* comienzan con los valores del atributo como puntos de corte potenciales, y el proceso va eliminando los puntos de corte uniendo así los intervalos (*mezcla*). Los métodos *top-down* comienzan con todo el rango de valores del atributo y el proceso va encontrando los puntos de corte dividiendo cada vez los nuevos intervalos.

- *Supervisados y no supervisados.* También llamados los métodos no supervisados, son aquellos que no toman en cuenta la información proporcionada por las clases. En cuanto a los supervisados [12] son los métodos que toman en cuenta la información de las clases para determinar los puntos de corte. La mayoría de los métodos son supervisados y estos pueden seleccionar los intervalos (del rango de valores del atributo a discretizar) basándose en el error de los datos de entrenamiento, en la entropía de los intervalos o en medidas estadísticas.

- *Univariable y multivariable.* Los métodos de discretización univariable dividen un sólo atributo a la vez. Por otro lado los multivariables consideran las relaciones existentes entre los atributos durante el proceso de discretización.

- *Globales y locales.* Los métodos globales discretizan todo el espacio del conjunto de entrenamiento una sola vez. Los métodos locales permiten la formación de diferentes conjuntos de intervalos para cada atributo y son aplicados en diferentes contextos de clasificación. En el caso de los árboles de decisión se pueden tener distintas discretizaciones de algún atributo, y cada discretización es aplicada a diferentes nodos en diferentes niveles del árbol inducido.

El algoritmo de discretización más utilizado en ILP es el que está basado en el principio de descripción mínima *(minimum description length principle - MDLP)* [20]. El algoritmo

2.2 muestra este proceso de discretización para un atributo numérico S con M ejemplos. Primero, se ordenan de menor a mayor los valores de S, y se crean n puntos de división candidatos. Cada punto de división candidato es el valor intermedio entre dos valores de S sucesivos. Para cada punto t_i se generan dos subconjuntos S_1, S_2 donde S_1 contiene los valores de S menores a t_i y S_2 los mayores a t_i. Después, se calcula la entropía para cada subconjunto S_j :

$$Ent\left(S_j\right) = -\sum_{i=1}^{k} P\left(C_i, S_j\right) log\left(P\left(C_i, S_j\right)\right)$$

Donde k es el número de clases y (C_i, S_j) es la proporción de ejemplos de S_j pertenecientes a la clase C_i. Con la entropía de cada subconjunto se calcula la información de entropía para t_i:

$$E\left(t_i, S\right) = \frac{|S_1|}{|S|} Ent\left(S_1\right) + \frac{|S_2|}{|S|} Ent\left(S_2\right)$$

Una vez que se tiene la información de entropía para todos los puntos de división candidatos, se elige el de menor entropía, a este punto lo denotamos t_{min}, y se calcula su ganancia de información:

$$Gain\left(t_{min}, S\right) = Ent\left(S\right) - E\left(t_{min}, S\right)$$

Finalmente, se aplica el criterio MDLP, el cual indica que el punto de división t_{min} será rechazado para dividir a S si la ganancia de información de t_{min} es menor a un costo, en caso contrario el punto de división es aceptado y el algoritmo se aplica a S_1, S_2 . El costo con el que se compara la ganancia de información de t_{min} es el siguiente:

$$\frac{log\left(M-1\right)}{M} + \frac{\Delta\left(t_{min}, S\right)}{M}$$

Donde $\Delta\left(t_{min}, S\right) = log\left(3^k - 2\right) - [kEnt\left(S\right) - k_1 Ent\left(S_1\right) - k_2\left(S_2\right)]$, k es el número de clases, y k_1, k_2 es el número de ejemplos para S_1, S_2 respectivamente.

La entropía y el principio de longitud de descripción mínima son dos conceptos importantes para el proceso de discretización explicado anteriormente, por lo que los revisamos a continuación.

Entropía

En el contexto de la mínería de datos y del aprendizaje maquinal, la entropía mide el grado de certidumbre en un sistema. Por ejemplo si tenemos una bolsa con pelotas

Algoritmo 2.2 Algoritmo de discretización MDLP

procedure DISCRETIZACIONMDLP(Atributo: S, Número Ejemplos: M)
  Ordenar valores de S en orden ascendente
  Crear los puntos de división candidatos $t_1, t_2, \ldots, t_n$
  for $i \leftarrow 1, n$ **do**
    Crear subconjuntos S_1 y S_2 con t_i
    Calcular entropía de S_1 y S_2
    Calcular información de entropía de t_i
  end for
  $t_{min} \leftarrow$ Obtener punto de división con menor información de entropía
  Calcular ganancia del punto de división t_{min}, denotada como *gain*(t_{min}, S)
  if *gain*$(t_{min}, S) \leq \frac{log(M-1)}{M} + \frac{\Delta(T,S)}{M}$ **then**
    Regresar nulo
  else
    Crear subconjuntos S_1 y S_2 con t_{min}
    $t_{izq} \leftarrow$ *MDLP(S_1, ejemplos en $S_1 : M_1$)*
    $t_{der} \leftarrow$ *MDLP(S_2, ejemplos en $S_2 : M_2$)*
    Regresar $t_{min}, t_{izq}, t_{der}$
  end if
end procedure

de colores, y debemos sacar una sin ver, y además todos los colores tienen la misma probabilidad de ser elegidos, decimos que la entropía del sistema (la bolsa con pelotas) es máxima, es decir que existe mucha incertidumbre sobre el color que será elegido. En este caso la medida máxima de entropía es 1. Por otro lado si todas las pelotas son del mismo color, entonces no habrá incertidumbre sobre el color que tendrá la pelota seleccionada, en este caso la medida de la entropía mínima es 0. Aplicando esta idea a nuestro entorno, el objetivo es discretizar un conjunto de valores continuos S. Es decir encontrar uno o más puntos de división *(split points)* que divida a S en dos o más subconjuntos $S_1, \ldots, S_n$, cuya entropía sea mínima. Dicho de otro modo se desea tener la certeza (poca incertidumbre) que los elementos de cada uno de estos subconjuntos tengan la misma clase. Entre más elementos pertenezcan a una misma clase, decimos que el conjunto es más puro. Si todos los elementos pertenecen a la misma clase (v.g. pelotas del mismo color) entonces decimos que es puro: $entropia = 0$.

Principio de Longitud de Descripción Mínima

El principio MDLP indica que la mejor teoría es aquella que tiene mayor compresión de datos. Tiene mucha relación con *la navaja de Occam [3]*, la cual indica que la mejor teoría es la más pequeña que pueda explicar todos los ejemplos. De alguna manera MDLP es una generalización de *la navaja de Occam*, pero desde el punto de vista de la teoría de la información [61]. El uso de MDLP como criterio de paro para detener el algoritmo recursivo mostrado en 2.2, tiene como objetivo evitar que se creen demasiados puntos de división que no aporten información adicional innecesariamente. Y en el algoritmo creado por Fayyad e Irani [20], el MDLP permite tener nodos con divisiones múltiples pero más compactos.

2.4. Agrupamiento

Una de las técnicas más importantes para la minería de datos y para el aprendizaje maquinal es el *agrupamiento (clustering)*. Un algoritmo de agrupamiento es aquel que toma un conjunto de objetos, y lo parte en una serie de *grupos (clusters)* de tal manera que los objetos dentro de un mismo grupo son similares entre sí, y los objetos de distintos grupos son diferentes [29]. Un concepto importante dentro de la definición anterior es el de *similitud* ya que en la mayoría de problemas resulta subjetivo afirmar si dos objetos son similares o no. Por ejemplo, en una base de datos de clientes de un súpermercado cuyos atributos son edad, dirección e ingresos, podríamos definir que dos clientes son similares si viven dentro de la misma zona postal, o si su ingreso es mayor a diez salarios mínimos. En el primer caso hablamos de qué tan cerca viven los clientes unos de otros, por lo tanto la distancia es una medida muy importante para la formación de grupos a partir de la dirección. En el segundo caso la similitud entre los clientes se basa en el salario. Para la creación de los grupos es necesario determinar qué tan cercanos son los objetos entre sí. Para ello se tienen medidas de *similitud* y de *distancia*. La similitud es una medida para indicar cuánto se parecen dos objetos, es decir, entre mayor sea la similitud, más cercanos se considerarán entre sí. Por otro lado, si se utiliza la distancia para crear los grupos, dos objetos se considerarán más cercanos cuanto menor sea la distancia entre ellos. Algunas medidas de distancia son: la distancia *Euclidiana [43]*, la distancia *Mahalanobis [43]*, y la distancia *Minkowski [29]*. También es posible determinar medidas de similitud para otro tipo de datos como los categóricos, los binarios o incluso datos mixtos.

Los algoritmos de agrupamiento utilizan principalmente una de las dos siguientes estructuras de datos.

- *Matriz de datos.* Esta matriz, también llamada estructura *objeto por variable*, representa los n objetos del conjunto de entrenamiento y sus p atributos. Al igual que la

estructura utilizada por los algoritmos atributo valor, esta matriz tiene n renglones y p columnas donde cada renglón es un objeto y cada columna un atributo.

- *Matriz de disimilitud.* Esta estructura, llamada también matriz *objeto por objeto o de distancias*, almacena todas las proximidades o distancias que hay entre cada par de objetos. Por lo general esta matriz es una estructura como la siguiente:

$$\begin{bmatrix} 0 & & & & \\ \text{d(2,1)} & 0 & & & \\ \text{d(3,1)} & \text{d(3,2)} & 0 & & \\ \vdots & \vdots & \vdots & & \\ \text{d(n,1)} & \text{d(n,2)} & \dots & \dots & 0 \end{bmatrix}$$

  Donde $d(i, j)$ es la distancia o disimilitud medida entre los objetos i y j. En general, $d(i, j)$ es un número positivo el cual puede ser cercano a cero cuando los objetos i y j son muy cercanos entre sí. Por definición $d(i, j) = 0$ si $i = j$.

Todo proceso de agrupamiento es una técnica de clasificación no supervisada. Cualquier proceso de agrupamiento se compone al menos de los siguientes cuatro pasos básicos.

1. *Selección de atributos.* Este paso consiste en buscar y extraer un subconjunto de atributos lo más pequeño posible pero que permita descubrir grupos que se den de manera natural y que sean interesantes.

2. *Selección o diseño del algoritmo de agrupamiento.* En este paso se define la medida de proximidad o similitud entre los objetos y la función de criterio. La función de criterio proporciona una medida para evaluar la calidad de los grupos generados durante el proceso de agrupamiento. También permite comparar diferentes soluciones de manera que se pueda elegir la mejor. Es importante señalar que tanto la medida de proximidad como la función de criterio determinan los grupos creados, sin embargo, la parte subjetiva sigue siendo inherente al proceso de agrupamiento.

3. *Validación de grupos.* En este caso se determina la calidad del agrupamiento realizado utilizando principalmente dos tipos de medidas: validación de agrupamiento externo, y validación de agrupamiento interno. La diferencia entre estos dos tipos de medidas es que las externas usan información que no está presente en el conjunto de entrenamiento mientras que las internas verifican la calidad del agrupamiento utilizando únicamente la información contenida en el conjunto de entrenamiento.

4. *Interpretación de resultados.* El cuarto paso de todo proceso de agrupamiento es la interpretación de los grupos finales, ya que estos deben proporcionarle al usuario información útil acerca de los datos utilizados en el agrupamiento.

Dentro de los algoritmos de agrupamiento disponibles, destacan los siguientes dos tipos: el *agrupamiento jerárquico* crea una descomposición de jerarquías que puede ser aglomerativa *(bottom-up)* o divisiva *(top-down).* En el primero, cada objeto forma inicialmente un grupo, de tal manera que en cada paso del algoritmo los objetos similares entre sí se van uniendo para formar los grupos finales. Por otro lado, en el modo divisivo, el conjunto total de objetos forma un único grupo el cual se va dividiendo conforme se ejecuta el algoritmo. Al final del proceso, ya sea aglomerativo o divisivo, los grupos finales pueden ser representados por medio de un dendrograma. En la Figura 2.3 podemos ver un agrupamiento jerárquico típico con su correspondiente dendrograma. En este caso los objetos 4 y 5 pertenecen a un grupo y el 1 y 3 a otro, sin embargo estos cuatro objetos pertenecen a un grupo de mayor jerarquía representado en color púrpura. Y este último grupo junto con el objeto 2 pertenecen a un grupo de mayor jerarquía representado en rojo.

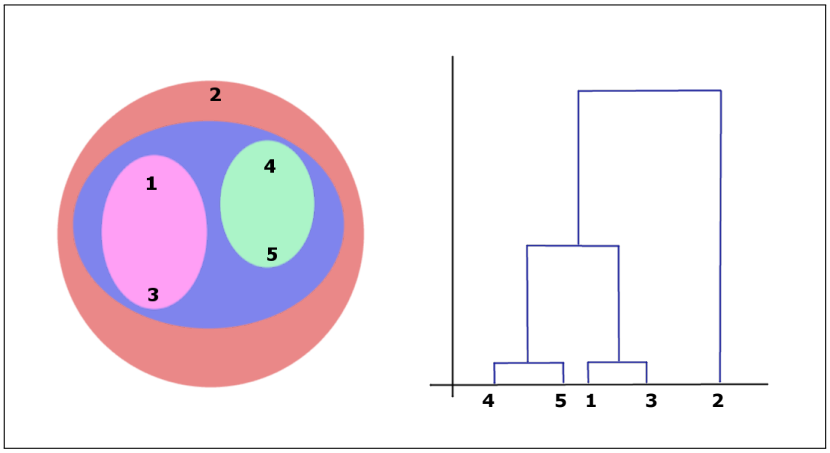

Figura 2.3: El agrupamiento jerárquico crea una serie de grupos que pueden ser representados por medio de un dendrograma.

El otro tipo de agrupamiento es el *particional*, el en cual el objetivo es crear c grupos disjuntos. Para ello el algoritmo comienza con una serie de grupos creados aleatoriamente de tal manera que en cada iteración se reubican los objetos creando una partición diferente a la anterior. Este proceso continúa hasta que la nueva partición ya no mejore a la anterior.

Datos categóricos. Uno de los tipos de datos más comunes en las bases de datos es el categórico ya que frecuentemente nos encontramos con atributos que no tienen un orden

natural como el género o la raza de una persona. Debido a esto es difícil definir una métrica de distancia o de similitud para atributos categóricos. Además, la mayor parte de las funciones de distancia definidas están basadas en métricas que solo pueden ser utilizadas en atributos numéricos, por ejemplo la media o la moda. En [32] proponen un método para el cálculo de la distancia en atributos categóricos basada en el contexto llamado *DIstance Learning for Categorical Attributes - DILCA*, el cual describimos a continuación.

DILCA tiene dos procesos principales para calcular la distancia entre cada par de valores categóricos de un atributo X. Primero se selecciona el conjunto de atributos categóricos más correlacionados a X, a este conjunto se le llama el contexto de X. Después se calcula la distancia entre cada par de valores categóricos de X utilizando los atributos de contexto seleccionados anteriormente. Estos dos pasos se detallan a continuación.

Sean $F = \{X_1, \ldots, X_m\}$ un conjunto de m atributos categóricos. Se denota la cardinalidad de un atributo X como $|X|$. Sea $D = \{d_1, \ldots, d_n\}$ el conjunto de entrenamiento con n ejemplos definido sobre F, y x_i un valor específico del atributo X.

Selección del contexto. En este paso se seleccionan los atributos categóricos en F más relacionados a X. Para ello se calcula la correlación entre cada atributo categórico en F y X. La correlación es medida con una métrica llamada *simetría de incertidumbre* [62], la cual, está inspirada en la teoría de la información y derivada a su vez de la entropía. La Ecuación 2.1 mide la simetría de incertidumbre de X respecto a un atributo categórico Y tal que $Y \in F$.

$$SU(X, Y) = 2 * \frac{\text{IG}(X \mid Y)}{H(X) + H(Y)} \tag{2.1}$$

Donde $H(X)$ es la entropía de una variable aleatoria, $IG(X \mid Y)$ es la ganancia de información de X proporcionada por Y.

Los valores de la simetría de incertidumbre están entre 0 y 1 de tal manera que el valor 1 indica que el valor de Y o de X predice completamente el valor de la otra variable, y el valor de 0 indica que X y Y son totalmente independientes. El objetivo de la selección de contexto es identificar aquellos atributos que tienen un valor de simetría de incertidumbre muy cercano a 1. De esta manera los atributos con mayor simetría de incertidumbre formarán el conjunto llamado contexto de X denotado por *contexto(X)*.

Cálculo de distancias. Una vez que se tiene el contexto de X, es decir los atributos categóricos más correlacionados a X, el siguiente paso es el cálculo de las distancias entre cada par de valores categóricos de X. La distancia entre los valores categóricos x_i y x_j tal que $x_i, x_j \in X$ se presenta en la Ecuación 2.2.

$$d(a_i, a_j) = \sqrt{\sum_{Y \in \textit{context(A)}} \sum_{y_k \in Y} \left(P(a_i \mid y_k) - P(a_j \mid y_k)\right)^2} \tag{2.2}$$

Donde $P(a_i \mid y_k)$ es la probabilidad condicional para a_i dado y_k, y $P(a_j \mid y_k)$ es la probabilidad condicional de a_j dado y_k.

Una vez calculados las distancias para cada par de valores del atributo X es posible crear la matriz de distancias y llevar a cabo el procesamiento de agrupamiento. Debido a que este método fue adaptado a nuestra propuesta para el manejo de valores categóricos en ILP lo presentamos en esta sección junto con los conceptos más importantes del proceso de agrupamiento.

2.5. Proposicionalización

La proposicionalización es el proceso que transforma un problema relacional a un formato proposicional [36]. El problema original puede ser una base de datos relacional o un conjunto de hechos y cláusulas lógicas correspondientes a un problema ILP. El resultado de la proposicionalización es una tabla que puede ser procesada por un aprendiz atributo-valor. El objetivo de la proposicionalización (también llamada *flattening*) es tratar con algoritmos proposicionales problemáticas para las cuales los aprendices relacionales son poco eficientes. Entre estas problemáticas tenemos el procesamiento de valores desconocidos, de ruido o de información numérica. Los métodos de proposicionalización se clasifican en dos grupos.

El primer grupo está compuesto por los *métodos orientados a la lógica.* Estos generan atributos proposicionales a partir de elementos lógicos como las cláusulas, hechos y variables. Tomemos como ejemplo el problema ILP donde el objetivo es determinar la relación de *abuelo* entre dos personas a partir de la relación *padre*. El conocimiento previo y los ejemplos son mostrados en la tabla 2.2.

Conocimiento previo	**Ejemplos**
padre(juan,jose).	*abuelo(juan,fernando).* $\oplus$
padre(jose,fernando).	*abuelo(javier,manuel).* $\oplus$
padre(javier,enrique).	*abuelo(fernando,jose).* $\ominus$
padre(enrique,manuel).	*abuelo(enrique,javier).* $\ominus$

Tabla 2.2: Ejemplo ILP para inducir la relación *abuelo*

La creación de nuevos atributos proposicionales se realiza a partir de las relaciones definidas en el conocimiento previo. LINUS [38], uno de los primeros sistemas de proposicionalización genera un atributo binario por cada combinación de las diferentes variables que pueden ser declaradas en la relación correspondiente. Por ejemplo, la relación *padre*

(Tabla 2.2) tiene dos argumentos, por lo tanto se pueden declarar como máximo dos variables X, Y. Por lo tanto tendremos cuatro posibles atributos nuevos, es decir con las combinaciones (X, X), (X, Y), (Y, X) y (Y, Y). Cada uno de estos atributos es igual a 1 si los valores para las variables correspondientes hacen verdadera la relación y es 0 en caso contrario. La Tabla 2.3 muestra los cuatro nuevos atributos que se crean con la relación *padre*. Las columnas X, Y corresponden a los valores que pueden tomar las variables. Finalmente, solo tenemos dos clases correspondientes a los ejemplos positivos y negativos.

Ejemplo	**X**	**Y**	**padre(X,X)**	**padre(X,Y)**	**padre(Y,X)**	**padre(Y,Y)**	**Clase**
1	juan	fernando	0	0	0	0	$\oplus$
2	javier	manuel	0	0	0	0	$\oplus$
3	fernando	jose	0	0	1	0	$\ominus$
4	enrique	javier	0	0	1	0	$\ominus$

Tabla 2.3: Atributos nuevos generados a partir de la relación *padre*.

El segundo grupo incluye las estrategias *basadas en bases de datos.* La característica principal dentro de este grupo es que la creación de atributos proposicionales se realiza con el uso de funciones de agregación (una función de agregación es aquella que toma un conjunto de registros de la base de datos y devuelve un único valor). Por ejemplo, la función *promedio()* podría usarse para crear un atributo proposicional llamado *promedioCuentasBancarias* el cuál toma como único valor el promedio de todas las cuentas bancarias por cada cliente de un banco.

Debemos tomar en cuenta que la tarea de proposicionalizar información relacional no es trivial, ya puede haber pérdida de información. Por lo tanto muchos sistemas limitan esta transformación a ciertos formatos. Por ejemplo, LINUS [38] solo soporta cláusulas de bases de datos deductivas y jerárquicas *(deductive hierarchical database clauses)* las cuales no permiten definiciones recursivas.

2.6. Programación lógica de restricciones

La *programación lógica de restricciones* (o *CLP*) por sus siglas en inglés extiende las capacidades de la programación lógica para manejar adecuadamente las restricciones [9]. El principal objetivo de la CLP es mejorar la expresividad de los lenguajes de programación de alto nivel, como Prolog, con el uso natural de restricciones. Para observar la diferencia entre la programación lógica y la CLP, ya que ambos lenguajes están basados en el de la lógica de primer orden, tomemos en cuenta el siguiente ejemplo: supongamos que tenemos la regla *suma(X,Y,Z)* $\leftarrow$ *Z is X+Y* y queremos la respuesta a la pregunta *?-suma(X,5,12)*. En este caso CLP puede contestar la pregunta con $X = 7$, sin embargo

Prolog nos devolverá un error de instanciación ya que el mecanismo de unificación usado por Prolog exige que tal valor sea dado de antemano. La CLP puede verse como la fusión entre dos paradigmas declarativos: la programáción lógica y la resolución de restricciones. La resolución de restricciones encuentra soluciones a problemas de satisfacción de restricciones (o *CSP*), los cuales son establecidos de la siguiente manera:

Dados: un conjunto de variables $T = \{t_1, t_2, \ldots, t_n\}$, para cada $t_i \in T$ un conjunto finito de posibles valores D_i (llamado dominio de t_i) y un conjunto de restricciones que limitan los valores que las variables pueden tomar al mismo tiempo.

Encontrar: una asignación de valores a las variables $\{t_1 = v_1, t_2 = v_2, \ldots, t_n = v_n\}$ tal que satisfaga todas las restricciones y donde $v_1 \in D_1, v_2 \in D_2, \ldots, v_n \in D_n$.

Veamos otro ejemplo que nos permitirá observar la ventaja de CLP sobre la programación lógica respecto al manejo de restricciones. Consideremos las reglas de la Figura 2.4 que generan la serie de *Fibonacci* con Prolog y con CLP:

```
% Prolog                          CLP

fib (0,1).                        fib (0,1).
fib (1,1).                        fib (1,1).
fib (N,R):-                       fib (N,R1+R2):-
    N1 is N-1, fib (N1,R1),           N >= 2, fib (N-2,R1),
    N2 is N-2, fib (N2,R1),           fib (N-1,R2).
    R is R1+R2.
```

Figura 2.4: Reglas que generan la serie de Fibonacci en Prolog y CLP.

Los dos lenguajes declarativos pueden resolver la pregunta *?-fib(7,F)* con la respuesta *F=13*, pero solo CLP devuelve una solución a la pregunta *?- fib(N,13)* con la respuesta *N=7*. Como se mencionó anteriormente, el mecanismo de unificación de Prolog requiere que los valores de las variables de entrada, como en el caso anterior donde la variable de entrada es N, sean dados a priori. Sin embargo CLP resuelve este problema de restricción para N en el dominio de los números naturales. Como podemos ver, los sistemas CLP encuentran soluciones tanto para variables de entrada como de salida. Otras diferencias entre CLP y programación lógica son las siguientes:

- Prolog solo declara variables sobre dominios de términos. Las variables en CLP tienen un rango mucho más amplio: $\mathbb{N}, \mathbb{Z}, \mathbb{R}$, etc.
- En CLP las restricciones pueden ser representadas por desigualdades, igualdades, expresiones lineales, expresiones booleanas, etc.

- En Prolog la propagación de restricciones[1] está basado en la unificación. Los sistemas CLP resuelven los problemas de restricciones con más eficiencia con métodos conocidos como *técnicas de consistencia.*

Cabe añadir que CLP no es utilizada solamente para la resolución de problemas de satisfacción de restricciones, como veremos más adelante, la CLP es utilizada por algunos sistemas ILP para el manejo de información numérica.

2.7. Algoritmos genéticos

Los *algoritmos genéticos - AG* [31], son métodos de optimización y búsqueda que imitan la evolución y la genética natural. Una de las definiciones más aceptadas es la que proporciona D. Goldberg: "Los GAs son algoritmos adaptativos de búsqueda heurística basados en ideas evolutivas de selección natural y genética natural" [27]. El proceso básico de los GAs crea un conjunto inicial de soluciones candidatas *(cromosomas)*. Este conjunto es llamado *población*. Después se calcula la aptitud *(fitness)* de cada cromosoma. Una función heurística calcula la aptitud para determinar qué tan buena es cada solución. Esta generación es evolucionada de manera iterativa con al menos tres operadores genéticos: el *operador de selección* elige los cromosomas en la población actual que serán reproducidos, el *operador de cruzamiento* recombina dos cromosomas para generar nuevas, y posiblemente, mejores soluciones y el *operador de mutación* modifica de manera aleatoria los componentes básicos de los cromosomas para introducir nueva información a la población actual. La nueva población se evalua y se seleccionan los cromosomas sobrevivientes. Este proceso se ejecuta hasta que se satisfaga algún criterio de paro. El algoritmo 2.3 describe este proceso.

Para aplicar un AG cada solución debe representarse en un formato adecuado para que los operadores genéticos se puedan aplicar sobre ellas creando nuevas y factibles soluciones. La representación más frecuente para un cromosoma es una cadena binaria en la que cada bit es un gen (unidad básica de un cromosoma). Cada gen es usado por los operadores de cruzamiento y mutación para generar nuevos individuos. El objetivo es que los genes de los mejores cromosomas sean propagados a las siguientes generaciones. Finalmente, si se cumple un criterio de paro se detiene el algoritmo, por ejemplo, si no hay una mejora significativa en las nuevas generaciones.

Ejemplo. Supongamos que deseamos maximizar la función de la Figura 2.5. Para ello, primero debemos encontrar una manera de representar cada cromosoma (solución). Cuando las soluciones son números reales (como en este caso) debemos convertirlas a cadenas

[1] La propagación de restricciones resuelve problemas de restricciones de manera iterativa, reduciendo el dominio de las variables al remover valores que no satisfacen dichas restricciones.

Algoritmo 2.3 Algoritmo genético

```
1:  procedure ALGORITMOGENETICO
2:      generación ← 0
3:      inicializa población(t)
4:      evalua población(t)
5:      while No se satisfaga el criterio de paro do
6:          t ← t + 1
7:          selecciona población(t) de población(t − 1)
8:          cruza pares de cromosomas en población(t)
9:          muta cromosomas en población(t)
10:         evalua población(t)
11:     end while
12: end procedure
```

binarias. Para este ejemplo codificamos cada solución $x \in [0, 50]$ como un cromosoma binario de 8 bits.

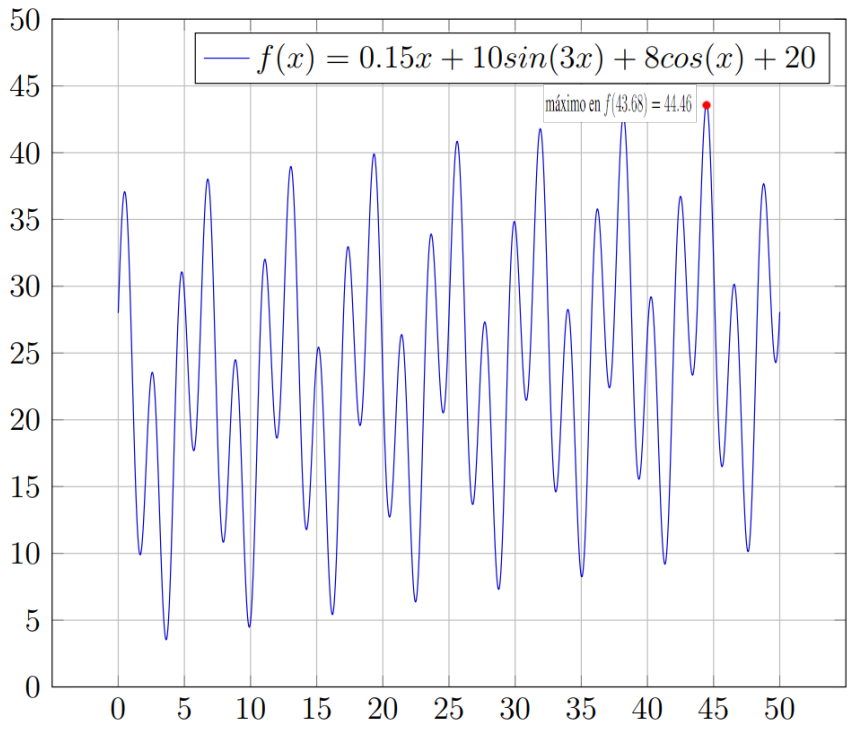

Figura 2.5: Función a maximizar.

El primer paso del algoritmo es generar un conjunto de soluciones aleatorias en el rango $[0, 50]$. Este será la población inicial. Después, cada solución será evaluada con la función dada *(función de aptitud)*. Entre mayor sea la aptitud de cada solución tendrá una mayor probabilidad de ser seleccionada para generar la siguiente generación por medio de la mutación y el cruzamiento.

No. cromosoma	**Valor *c***	**Cromosoma**	**Aptitud *f(c)***	**Probabilidad de selección**
1	27	10001001	15.414	0.05728
2	2	00001010	14.176	0.05268
3	43	11011011	28.956	0.107604
4	32	10100011	41.309	0.15351
5	1	00000101	25.8836	0.09618
6	29	10010011	10.147	0.037708
7	5	00011001	29.522	0.109707
8	38	11000001	41.19	0.15306
9	7	00100011	35.4477	0.1317
10	17	01010110	27.05	0.1005

Tabla 2.4: Población inicial.

Para crear la siguiente generación debemos mutar cromosomas y cruzar parejas de cromosomas. La mutación en las cadenas binarias consiste únicamente en modificar aleatoriamente un número dado de bits en un cromosoma, y de esta manera se genera un nuevo cromosoma. Para el cruzamiento se elige aleatoriamente una posición dentro del par de cromosomas a cruzar, y después se intercambian los bits a partir de dicho punto. Esto produce dos nuevos cromosomas. Las probabilidades de mutación en algoritmos genéticos suelen ser menores al $5\,\%$, sin embargo, para nuestro ejemplo necesitamos generar otra generación de 10 cromosomas así que elegimos 4 para mutación, y 3 parejas para cruzamiento. Para seleccionar cada cromosoma podemos usar el método de la ruleta. En este método simulamos una ruleta dividida en n partes, donde n es el número de cromosomas que hay en la población. El tamaño que le corresponde a cada individuo es proporcional a su probabilidad de ser seleccionado. El proceso de la ruleta es el siguiente:

1. Para cada cromosoma c_i, donde $i = [1, 2, \ldots, n]$, se calcula la aptitud $f(c_i)$. En nuestro ejemplo la función es $f(c_i) = 0.15c_i + 10sen(3c_i) + 8cos(c_i) + 20$ donde $0 \leq c_i \leq 50$.

2. Para cada cromosoma c_i se calcula su probabilidad de ser seleccionado $p(c_i) = \frac{f(c_i)}{\sum_{k=1}^{n} f(c_k)}$.

3. Para cada cromosoma c_i se calcula la probabilidad acumulada $q(c_i) = p(c_1)+p(c_2)+\ldots+p(c_i) = \sum_{j=1}^{i} p(c_j)$

4. Para seleccionar un cromosoma:

   a) Se genera un número aleatorio ρ en el intervalo $[0..1]$.

   b) Se elige el i-ésimo cromosoma c_i tal que $q(c_i) < \rho < q(c_i)$.

Para nuestro ejemplo se seleccionaron para mutación los cromosomas $4, 3, 5$ y 5; y para cruzamiento las parejas: 4 con 1, 4 con 3, y 7 con 8. En la Tabla 2.5 podemos ver reflejados los datos de la nueva generación.

No.	**Cromosoma original**	**Nuevo cromosoma**	***c***	***f(c)***
4	*1 0 100011*	*1 1 100011*	44.509	43.59
3	*1101101 1*	*1101101 0*	42.745	34.42
5	*0 0 000101*	*0 1 000101*	13.529	29.0955
5	*000010 0 1*	*000010 1 1*	2.156	17.76
4	*101000 | 11*	*101000 | 01*	31.5686	37.049
1	*100010 | 01*	*100010 | 11*	27.2549	$20.7294
4	*101 | 00011*	*101 | 11011*	36.666	29.16
3	*110 | 11011*	*110 | 00011*	38.23	42.605
7	*000 | 11001*	*000 | 00001*	0.196	33.42
8	*110 | 00001*	*110 | 11001*	42.54	36.64

Tabla 2.5: Segunda generación. Los 3 primeros cromosomas nuevos se obtuvieron por mutación. El gen *(bit)* que está separado del resto es el que se cambió. Los 6 restantes se generaron por medio del cruzamiento, el caracter $|$ muestra el punto de cruce.

Al comparar el promedio de aptitud por cada generación podemos ver que hay una mejora significativa. La primera generación tiene un promedio de 26.90 mientras que el de la segunda generación es 32.44. También podemos observar una mejora en el mejor cromosoma de cada generación. El mejor cromosoma de la población inicial tiene una aptitud de 41.309 mientras que el de la segunda población es de 43.59. Esta última de hecho se acerca mucho al óptimo de la función como se puede ver en la gráfica de la Figura 2.5.

En este capítulo se detallaron los conceptos más importantes utilizados en el presente trabajo. Primero se hizo énfasis en la diferencia entre aprendizaje proposicional y aprendizaje relacional. Este último es importante ya que la propuesta presentada está basada en la ILP, la cual, realiza su proceso de aprendizaje sobre información relacional. Otros

dos conceptos importantes son la discretización y el agrupamiento. El primero fue utilizado para el manejo de atributos numéricos en la ILP, y el segundo en el agrupamiento de atributos categóricos. Ya que se utilizaron los algoritmos genéticos para el manejo de atributos numéricos y categóricos también se detallaron los elementos más importantes de estos algoritmos. En el siguiente capítulo se describen las estrategias para el manejo de atributos numéricos y categóricos estudiadas en el estado del arte.

Capítulo 3

Estado del arte

> "La adversidad tiene el don de despertar talentos, que en la comodidad hubieran permanecido dormidos."
>
> Horacio

En ILP existen diferentes estrategias para el manejo de atributos numéricos y categóricos. En este capítulo presentamos las principales que están basadas en la discretización, la programación lógica de restricciones, métodos evolutivos y proposicionalización.

3.1. Discretización en ILP

Uno de los métodos de discretización más utilizados es el que está basado en el *principio de longitud de descripción mínima -MLDP [20]*, descrito en la sección 2.3. Sin embargo no es la única técnica utilizada para el manejo de información numérica. Veremos además que uno de estos sistemas implementa un algoritmo para el manejo de argumentos categóricos. A continuación presentamos algunos sistemas ILP que utilizan métodos de discretización para tratar con los atributos numéricos.

3.1.1. Sistema INDUBI/CSL

El sistema INDUBI/CSL [42] es un sistema propuesto para la la minería de datos en el área de comprensión de documentos *(document understanding).* Esta área involucra el diseño e implementación de procesos para la extracción de información comprensible

para los seres humanos a partir de un un conjunto de documentos escaneados, ya sean fotografías, textos, dibujos, etc. [54].

INDUBI/CSL utiliza el lenguaje *variable-valued system* VL_{21} [44], el cual es una extensión de la lógica de primer orden que utiliza *selectores* en lugar de predicados. Los selectores que permite este sistema tienen el formato mostrado en la Ecuación 3.1.

$$f(t_1, \ldots, t_n) = V \text{ y } f(t_1, \ldots, t_n) \in S \tag{3.1}$$

Donde f es una función de aridad n y t_i es una variable o una constante. Además f puede tomar un valor V, o un valor del conjunto S.

A diferencia de la mayoría de los sistemas ILP, el modelo de generalización utilizado por INDUBI/CSL está dado por la θ_{OI}*-subsunción*. Esta relación está basada en la noción de objeto identidad la cual determina que todos los términos denotados con símbolos diferentes deben ser distintos. La siguiente definición formaliza la θ_{OI}*-subsunción* bajo el objeto identidad.

Definición **3.1.** θ-subsunción *Sean C, D dos cláusulas Datalog*[1]*. Decimos que D* θ-subsume *a C bajo Objeto Identidad si y sólo si $\exists\sigma$ substitución tal que $D_{OI^\sigma} \subseteq C_{OI}$ [21].*

Uno de los problemas que surge de la minería de documentos es el manejo de los atributos numéricos. En este contexto, el sistema INDUBI/CSL incluye un operador de especialización que discretiza los atributos numéricos durante el proceso de aprendizaje. Para especializar una hipótesis G, el operador añade un selector L al cuerpo de la hipótesis. En el caso de que L sea numérico entonces el operador discretiza al selector antes de ser añadido. El proceso de discretización de un selector numérico L sigue el proceso siguiente:

1. Ordenamiento de los valores numéricos. Se crea una tabla ordenada de manera ascendente que contiene los pares $\langle$*Valor,Clase*$\rangle$ tal que *Clase* $\in \{+, -\}$ y *Valor* $\in L$. Los valores ordenados podrían ser como los mostrados en la Tabla 3.1.

Value	0.5	0.7	0.9	1.0	1.5	1.5	1.5	1.7	2.5	2.5
Class	+	−	−	−	−	−	+	+	−	+

Tabla 3.1: Valores ordenados de manera ascendente.

2. Creación de puntos de división. Para encontrar los puntos de división se sigue el siguiente criterio: un valor numérico d es un punto de división si cae entre dos intervalos consecutivos y disjuntos $[l_1, l_2]$ y $[r_1, r_2]$ donde l_1 es el valor más pequeño dentro de la

[1]Una cláusula Datalog es una cláusula de programa que no tiene símbolos de función de aridad > 0. Es decir, solo las variables y las constantes pueden ser usadas como argumentos en los predicados.

tabla con clase $+$, l_2 es el valor más grande en la tabla tal que no excede el valor de d, r_1 es el valor más pequeño tal que no excede el valor de d, y r_2 es el valor más grande con clase $+$. Para el ejemplo de la tabla anterior, los puntos de división e intervalos se muestran en la Tabla 3.2.

d	0.60	1.25	1.60	2.10
$[l_1, l_2]$	[0.50..0.50]	[0.50..1.00]	[0.50..1.50]	[0.50..1.70]
$[r_1, r_2]$	[0.70..2.50]	[1.50..2.50]	[1.70..2.50]	[2.50..2.50]

Tabla 3.2: Puntos de división e intervalos encontrados.

3. Selección de intervalos. Los intervalos que cubren el valor numérico que coincide con el ejemplo semilla e^+ son eleccionados. El proceso de aprendizaje de INDUBI/CSL está guiado por un ejemplo semilla que es elegido aleatoriamente al comenzar el aprendizaje de una nueva hipótesis. Por lo tanto los valores numéricos que coincidan o que estén relacionados con e^+ van a variar.

4. Cálculo de la entropía. En este paso se calcula la ganancia de información de cada uno de los intervalos seleccionados de acuerdo a la ecuación, ver Sección 2.3.

5. Selección del mejor intervalo. El intervalo que minimice la entropía (el que tenga mayor ganancia de información) se elige como el mejor intervalo. Dicho intervalo será utilizado por el operador de especialización para ser añadido al cuerpo de la hipótesis.

6. Especialización. El operador de especialización añade el selector L con el mejor intervalo al cuerpo de la hipótesis. En nuestro ejemplo el selector a añadir podría ser como el siguiente:

$$L = f(X_1, \ldots, X_n) \in [1.50..2.50]$$

Como podemos ver este sistema utiliza puntos de división para el manejo de atributos numéricos. Estos son obtenidos por medio de la discretización que se lleva a cabo durante el proceso de especialización (discretización local). Por lo tanto los puntos de división obtenidos dependen en gran medida del operador de refinamiento utilizado. Si se quisiera implementar un operador diferente, por ejemplo uno de generalización, entonces el manejo de atributos numéricos debe ser adaptado a la estrategia que sigue dicho operador para generar nuevas cláusulas. Por otro lado debemos notar también que INDUBI/CSL no maneja de manera explícita los atributos categóricos, sino que los valores probados no cambian durante la inducción.

3.1.2. TILDE e ICL

Los sistemas *Top-Down Induction of Logical DEcision Trees - TILDE*) [6] e *Inductive Constraint Logic - ICL* [37, 14] discretizan los atributos numéricos una sola vez antes del proceso de aprendizaje. Estos sistemas y su proceso de discretización son descritos a continuación.

TILDE

Una de las ventajas que ofrece el aprendizaje proposicional sobre el relacional es el manejo de atributos numéricos. Esta ventaja es adaptada al contexto de la ILP en el sistema de inducción de árboles de decisión lógicos (*Top-Down Induction of Logical DEcision Trees - TILDE*) [6]. En este sistema las teorías son representadas como árboles de decisión lógicos, y el aprendizaje está basado en interpretaciones donde cada ejemplo puede tener varias tuplas. Un árbol de decisión lógico es un árbol de decisión binario que cumple las siguientes restricciones: cada nodo es una conjunción de literales (dentro del contexto de la lógica de primer orden) y una variable que es introducida en algún nodo no puede ocurrir en la rama derecha de ese nodo. La construcción de teorías en TILDE está basado en el *aprendizaje de interpretaciones*) definido de la siguiente manera:

Dados: un conjunto de cláusulas C, un conjunto de ejemplos clasificados E y un conocimiento previo B.

Encontrar: una teoría T tal que $\forall e \in E\ H \wedge e \wedge B \models c$ y $H \wedge e \wedge B \not\models c'$ donde: c es la clase del ejemplo e y $c' \in C - \{c\}$.

En este tipo de aprendizaje cada interpretación es un conjunto de hechos, además permite la inducción de predicados proposicionales que en este caso es el atributo clase.

ICL

Al igual que TILDE, el sistema *(Inductive Constraint Logic - ICL)* [37, 14] induce teorías en el lenguaje de la lógica de primer orden, pero en este caso los ejemplos son vistos como *interpretaciones de Herbrand.* Una *interpretación de Herbrand* es un conjunto de hechos construidos con símbolos de constante, de predicado y de función en un alfabeto de primer orden. La tarea de aprendizaje para ICL se define a continuación:

Dados: un conjunto de interpretaciones P tal que $\forall p \in P, M(B \cup p)$ es una interpretación verdadera de la teoría objetivo (ejemplos positivos), un conjunto de interpretaciones N tal que $\forall n \in N, M(B \cup n)$ es una interpretación falsa de la teoría objetivo (ejemplos negativos), un conjunto de cláusulas definidas *(definite clauses)* como conocimiento previo B y un lenguaje L_T que define el conjunto de las cláusulas aceptadas de antemano en la teoría.

Encontrar: una teoría clausal $T \subset L_H$, tal que $\forall p \in P, M(B \cup p)$ es una interpretación verdadera de T, $\forall n \in N, M(B \cup n)$ es una interpretación falsa de T.

Discretización en TILDE e ICL

Como se mencionó anteriormente, tanto TILDE como ICL siguen el mismo procedimiento de discretización, el cual está basado en el método presentado en la Sección 2.3. Para su uso en TILDE e ICL, este proceso se realiza de la siguiente manera:

Primero, el usuario final debe declarar los atributos que quiere discretizar durante el proceso de aprendizaje. Para esto, el meta-predicado *to_be_discretized* indica tanto la literal como los argumentos a ser discretizados. Por ejemplo, la siguiente declaración indica que el argumento *Edad* de la literal *empleado* será discretizado.

$$\textit{to\_be\_discretized(empleado(Nombre,Direccion,ID,Edad),[Edad])} \tag{3.2}$$

Después los argumentos declarados serán discretizados una sola vez antes del proceso de aprendizaje (discretización global). El proceso de discretización utilzado es el que se basa en el MDLP pero con las siguientes adaptaciones:

- Ya que cada ejemplo puede tener uno o más valores numéricos, para el mismo argumento, se le asigna un peso entre cero y uno que es usado para calcular la ganancia de información. De esta manera, un ejemplo que no tenga valores numéricos para el argumento numérico en cuestión tendrá un peso igual a 0, y un ejemplo que tenga, para ese mismo argumento, el máximo posible de valores tendrá un peso igual a 1.
- El criterio de paro en el algoritmo original de Fayyad e Irani, está basado en el principio MDLP, descrito al inicio de este capítulo. En el caso de TILDE e ICL el usuario final puede definir un número máximo de intervalos M. Por lo tanto el proceso de discretización se detiene cuando se alcanza M.

Para la declaración dada en la ecuación 3.2 un conjunto de intervalos como los siguientes podrían ser obtenidos.

$$discretized(empleado(Nombre, Direccion, ID, Edad), [Edad], [21, 35, 50]) \tag{3.3}$$

Los intervalos encontrados pueden ser usados con desigualdades para identificar si un valor numérico es menor o mayor que algún punto de división o con igualdades para determinar si un valor numérico pertenece a un intervalo delimitado por dos puntos de división consecutivos o no consecutivos. Para esto el meta-predicado *rmode* permite declarar el

uso que el usuario decida para los intervalos calculados. Por ejemplo, si los intervalos encontrados son los mostrados en la ecuación 3.3, entonces el usuario podría decidir utilizar desigualdades para su uso en la teoría final. Esto se muestra a continuación.

$$rmode(\#(C : threshold(empleado(\_, \_, \_, Edad), [Edad], C), +Edad < C) \quad (3.4)$$

En la ecuación 3.4 el meta-predicado *rmode* determina el operador de refinamiento y el sesgo de lenguaje. En este caso se indica que la literal *empleado* debe ser añadida al cuerpo de la cláusula a especializar. La variable *Edad* será comparada a un punto de división C por medio de una desigualdad. Para más detalles del uso de los meta-predicados de TILDE e ICL referimos al manual del sistema ACE que incluye ambos algoritmos en [5].

El uso de meta-predicados en TILDE e ICL les da un lenguaje declarativo superior a INDUBI/CSL, sin embargo el proceso de discretización es global. Solo se obtienen puntos de división al inicio del proceso inductivo y estos no cambian durante la construcción de nuevas hipótesis. Por un lado esto tiene la ventaja de que el tiempo de ejecución no aumenta significativamente, pero ya que el conjunto de ejemplos tiende a ser dinámico (cada hipótesis es generada con un conjunto de ejemplos diferente) entonces no es posible generar y probar potenciales puntos de división con cada nuevo conjunto de ejemplos.

3.1.3. Aprendizaje multivalores en ILP

En [48] se presentó un método para realizar una división binaria en cada atributo numérico y un agrupamiento binario en cada atributo categórico. Esta técnica crea un punto de división d para cada atributo numérico y dos subconjuntos de valores para cada atributo categórico. Esta información junto con predicados adicionales son usados para la creación de cláusulas (multivalores) que permiten el enriquecimiento del conocimiento previo. Una vez que la nueva información se añade al conocimiento previo el sistema FOIL [53] realiza el proceso de inducción. Este método es descrito a continuación:

1. Discretización y agrupamiento. En este paso, cada atributo $A = \{x_1, \ldots, x_n\}$ es discretizado/agrupado de manera binaria.

Si A es numérico, se calcula un punto de división d que divide el rango numérico de A en dos subintervalos. Si A es categórico, entonces dos subconjuntos de valores categóricos S_1 y S_2 son creados. Estos subconjuntos cumplen con los siguientes criterios: $S_1 \cup S_2 = A$ y $S_1 \cap S_2 = \phi$. Este proceso de discretización/agrupamiento es llevado a cabo con el algoritmo para selección de punto de división de dos algoritmos inductores de árboles de decisión: *Quick Unbiased Efficient Statistical Tree - QUEST [39]*, y *Classification Rule With Unbiased Interaction Selection and Estimation - CRUISE [35]*. Estos algoritmos serán descritos más adelante.

2. Construcción de cláusulas multivalores. Despues de discretizar/agrupar los atributos correspondientes, los puntos de división y subconjuntos obtenidos son usados para crear cláusulas multivalores. *Una cláusula multivalores* es aquella que tiene una o más literales con al menos un argumento que hace referencia a un conjunto de valores. Si el argumento es numérico entonces hace referencia a una desigualdad y si es categórico entonces está relacionado a un conjunto de valores. Para cada atributo numérico se tendrá un par de cláusulas multivariables como las siguientes:

```
adulto(P):−edad(P,Edad),Edad<18.

adulto(P):−edad(P,Edad),Edad>=18.
```

En el caso de un atributo categórico, se crea también un par de cláusulas multivalores pero como las que se presentan enseguida:

```
clase(Animal, reptil ):− cobertura(Animal,Cobertura),
                         miembro(Cobertura,[ninguna,escamas]).

clase(Animal, reptil ):− cobertura(Animal,Cobertura),
                         miembro(Cobertura,[plumas,pelo ]).
```

3. Enriquecimiento del conocimiento previo. Las nuevas cláusulas multivalores son agregadas al conocimiento previo para enriquecerlo.

4. Uso. Finalmente, el sistema FOIL realiza el proceso de aprendizaje con el nuevo conocimiento previo. Si los puntos de división y los subgrupos son relevantes, entonces aparecerán en la teoría final.

Como se mencionó anteriormente el algoritmo de selección de punto de división de los inductores QUEST y CRUISE son utilizados para discretizar y agrupar los atributos. Estos algoritmos siguen los siguientes pasos:

Transformación CRIMCOORD. Ya que la discretización binaria se realiza solo con conjuntos de valores numéricos, tanto QUEST como CRUISE transforman cada atributo categórico a un formato numérico. Si A es un atributo categórico con valores $A = \{x_1, \ldots, x_n\}$, entonces:

- A cada valor $x_i \in A$ se le asigna un vector binario *dummy*. Por ejemplo si se tiene el atributo $A = \{rojo, verde, azul\}]$ entonces se tendrían los siguientes tres vectores dummy: *rojo* $= 100$, *verde* $= 010$, *azul* $= 001$,
- Después cada vector dummy es proyectado a su coordenada discriminante más grande (CRIMCOORD). Esta técnica permite transformar un vector multidimensional, como los vectores dummy, a un valor numérico [26].

Selección del punto de división. Para determinar el mejor punto de división sobre los datos numéricos, QUEST y CRUISE realizan lo siguiente:

- *QUEST.* El algoritmo *k-means* [41] con $k = 2$ es llevado a cabo sobre los valores de A para formar dos conjuntos de valores numéricos con clases A_1 y A_2. Este proceso asegura una partición binaria. Por último se aplica un análisis discriminante cuadrático [25] para calcular el mejor punto de división d entre los conjuntos con clases A_1 y A_2.

- *CRUIISE.* En este caso se lleva a cabo un análisis discriminante lineal para encontrar el mejor punto de división d. Ya que este análisis es más eficiente con distribuciones normales, entonces se lleva a cabo una transformación *BOX-COX* sobre el conjunto A.

Como hemos visto a diferencia de los sistemas TILDE, ICL e INDUBI/CSL, el método multivalores adaptado al sistema FOIL es capaz de manejar tanto atributos númericos como categóricos. Sin embargo el método utilizado solo genera un punto de división para los atributos numéricos, y en el caso de los categóricos solo genera dos subconjuntos, lo cual puede ser una limitante si el problema a tratar necesita de más de un punto de división o más subonjuntos de valores categóricos.

3.2. CLP y evaluación perezosa en ILP

Algunos sistemas ILP se auxilian con la programación lógica de restricciones o CLP (Sección 2.6) ya que permite el manejo eficiente de restricciones sobre atributos numéricos. A continuación presentamos dos de estos sistemas.

3.2.1. Algoritmo NUM

En [2] Anthony y Frisch proponen un algoritmo llamado *NUM* el cual genera literales numéricas durante un proceso de aprendizaje de tipo top-down. Al igual que FOIL, el algoritmo presentado comienza con la cláusula más general c. A partir de esta un operador de refinamiento genera un conjunto de cláusulas más específicas. El operador de refinamiento es la adición de una literal al cuerpo de c y se lleva a cabo utilizando los predicados y valores definidos en el conocimiento previo. Cada hipótesis se construye de acuerdo al meta-predicado *usage/1* de la siguiente manera:

Sea p un símbolo de predicado en el conocimiendo previo B. Una *declaración de uso* para p es el meta-predicado *usage/1* cuyo único argumento es un átomo o una restricción,

construida sobre p. Además un *término de uso* es un término que reemplaza los símbolos de variable de p junto con uno de los modos siguientes: *positivo (denotado $+$)* indica que esta posición puede contener un símbolo de variable existente en la presente cláusula, *negativo (denotado $-$)* indica que esta posición puede contener una variable inexistente en la cláusula actual, finalmente *constante (denotado $\#$)* indica que esta posición puede contener un valor de constante. Por ejemplo, la declaración *usage(suma(+int,+int,-int))* indica que el predicado *suma/3* puede ser usado para refinamiento de una hipótesis c, y que sus dos primeros argumentos son variables que deben existir en c y el tercero es una variable nueva.

Si el predicado p de la literal a añadir es no numérica, entonces el operador de refinamiento genera el conjunto de literales candidatas. Si p es numérico entonces la tarea de generar el conjunto de las literales candidatas es delegado al algoritmo NUM. Este proceso es explicado a continuación. Supongamos que tenemos la relación objetivo $t(X, Y) \leftarrow$, el siguiente conjunto de entrenamiento, y la declaración de uso dada por el usuario final. Estos dos últimos elementos dados en la Tabla 3.3.

E^+	E^-	Declaración de uso
$t(-2,-2)$	$t(-1,1)$	
$t(-1,6)$	$t(-1,-2)$	$usage((+real) =$
$t(2,0)$	$t(2,4)$	$(\#real) \times (+real) + (\#real))$
$t(3,-2)$	$t(3,0)$	
$t(4,1)$	$t(4,-1)$	

Tabla 3.3: Ejemplos positivos y negativos junto con la declaración de uso.

La declaración de uso indica que NUM creará literales numéricas con la forma $Y = C_1X + C_2$, donde C_1, C_2 llamadas *c-variables* son declaradas con el símbolo $\#$. En este caso el operador de refinamiento le delega al algoritmo NUM la creación de literales numéricas, las cuales serán añadidas a la hipótesis actual para refinarla. El proceso seguido por NUM es el siguiente:

1. Sistema de ecuaciones. En este paso NUM crea un conjunto de sistemas de ecuaciones. Cada sistema de ecuaciones contiene n ecuaciones con n c-variables. En nuestro ejemplo, la declaración de uso contiene sólo dos c-variables, por lo tanto tendremos sistemas de dos ecuaciones. Cada ecuación es creada reemplazando las variables por los valores existentes en el conocimiento previo. Por ejemplo, los dos primeros ejemplos positivos del conjunto de entrenamiento producen el siguiente sistema:

$$\begin{aligned} -2 &= -C_1 2 + C_2 \\ 6 &= -C_1 + C_2 \end{aligned}$$

Donde los valores correspondientes a las variables de la relación objetivo son $X = -2, Y = -2$ en la primera ecuación, y $X = -1, Y = 6$ en la segunda ecuación.

2. Resolución. Un sistema CLP encuentra la solución a cada sistema de ecuaciones. Cada solución representa una literal numérica que será usada por el operador de refinamiento. La solución para el sistema del ejemplo anterior es $C_1 = 8$, $C_2 = 14$. Por lo tanto la correspondiente literal es $Y = 8X + 14$.

3. Especialización. Todas las literales numéricas generadas en el paso anterior son usadas para especializar la cláusula c y generar las hipótesis candidatas. Para la relación objetivo $t(X, Y) \leftarrow$ se tendrían hipótesis como la siguiente: $t(X, Y) \leftarrow Y = 8X + 14$.

4. Aprendizaje. Finalmente el proceso de aprendizaje continúa de manera normal para generar la siguiente hipótesis que formará parte de la teoría final.

El algoritmo NUM permite generar literales numéricas donde dos o más atributos se relacionan. Estas relaciones pueden tener diversas formas: cuadráticas, lineales, exponenciales, etc. Estas características permiten mejorar significativamente la eficiencia de los sistemas ILP en el manejo de información numérica. Sin embargo, el algoritmo es difícil de utilizar ya que los usuarios deben saber a priori la forma de las relaciones numéricas que serán utilizadas en la construcción de hipótesis. Por otro lado, el usuario debe tener al menos conocimientos básicos sobre programación lógica para poder codificar estas relaciones.

3.2.2. Sistema BPU-CILP

En general un sistema CLP permite el manejo de atributos numéricos en ILP sin necesidad de conocimiento previo adicional, aunque se asume la existencia de un sistema que revuelve los problemas de restricciones para la creación de cláusulas o literales numéricas. En [63] se propone un sistema llamado BPU-CILP, el cual reemplaza el uso de un algoritmo para resolver las restricciones. La estrategia de búsqueda está basada en una versión optimizada del algoritmo *beam search* [40], el cual utiliza una función heurística basada en la ganancia de información (ecuación 3.5).

$$H(L, p_0, n_0, p_1, n_1) = p_1(log_2 \frac{p_1}{p_1 + n_1} - log_2 \frac{p_0}{p_0 + n_0}) - \alpha CP(L) \tag{3.5}$$

Donde: L es una literal candidata, p_o, n_o es el número de ejemplos positivos y negativos respectivamente, p_1, n_1 es el número de ejemplos de ejemplos positivos y negativos

respectivamente cubiertos por L. El coeficiente de penalización es α con valor de 0.2. Finalmente, *CP(L)* es la complejidad de la literal L definida como sigue:

$$CP(L) = \begin{cases} \text{aridad de } L, & \text{si } L \text{ es una literal normal} \\ \text{la suma de los grados de los términos en } L, & \text{si } L \text{ es una restricción} \end{cases}$$

BPU-CILP construye cláusulas numéricas con las siguientes dos formas:

1. Ecuaciones lineales. En este caso, BPU-CILP determina los valores de las constantes en las siguientes desigualdades:

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_0 > 0 \tag{3.6}$$
$$w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_0 < 0 \tag{3.7}$$

Donde: $w_0, w_1, \ldots, w_n$ son constantes y $x_1, x_2, \ldots, x_n$ son los atributos en la cláusula a ser especializada.

Los valores constantes en las desigualdades, Ecuaciones 3.7 y 3.7, deben ser óptimos respecto a la función heurística de la ecuación 3.5. Para encontrar dichos valores, BPU-CILP lleva a cabo un análisis discriminante lineal. Este análisis puede proyectar datos multidimensionales a una representación con menor número de dimensiones. Dicha proyección maximiza la media entre dos o más clases y minimiza la covarianza dentro de cada una de las clases [16].

Por ejemplo si se debe especializar la cláusula: $goal(A, B, C) \leftarrow$, donde cada par (B, C) se encuentra en el conjunto de los ejemplos negativos *Neg* $= \{(10, 8), (7, 6), (9, 6)\}$ y positivos *Pos* $= \{(8, 6), (11, 8), (4, 6), (10, 6), (5, 5), (7, 5), (12, 9)\}$. Entonces el análisis discriminante lineal devolverá las siguientes desigualdades como resultado del proceso de clasificación:

$$-0.8425A + 0.5287B + 4.7098 < 0$$

$$-0.8425A + 0.5287B + 3.8604 > 0$$

Por lo tanto, las cláusulas candidatas son:

$$goal(A, B, C) \leftarrow -0.8425B + 0.5287C + 4.7098 < 0$$

$$goal(A, B, C) \leftarrow 0.8425B + 0.5287C + 3.8604 > 0$$

2. Ecuaciones y desigualdades lineales inexactas. En este caso BPU-CILP induce ecuaciones inexactas de la forma $p \approx 0$ y desigualdades de la forma $p \neq 0$, donde $p = w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_0$ es un polinomio que representa todos los puntos $t = (x_1, x_2, \ldots, x_n)$ tal que, para un error especificado ε, la distancia Euclidiana entre t y p es menor que ε. Para encontrar p, BPU-CILP utiliza el algoritmo de agrupamiento mostrado en la figura 3.1, el cual tiene una medida de proximidad definida como la distancia cuadrada de t a p. Dicho algoritmo es similar a *k-means*.

Algoritmo 3.1 Algoritmo de agrupamiento de BPU-CILP

procedure AGRUPAMIENTO BPU-CILP(E^+, E^-, B)
  Puntos ← EjemplosPositivos $(x_1(i), \ldots, x_n(i))$, $i = 1, \ldots, k$
  for all (j_1, j_2) con $1 \leq j_1 < j_2 \leq n$ **do**
    SubPuntos ← $(x_{j1}(i), x_{j2}(i)), i = 1, \ldots, k$
    for $i \leftarrow 1$ **to** k_1 (pre-especificado) **do**
      Clusters$[i]$ ← Resultado de detectar i SubGrupos lineales en SubPuntos
    end for
    Determinar el Clusters$[j]$ más sensible (de acuerdo al criterio de agrupamiento
    Construir soluciones inexactas de acuerdo a Clusters$[j]$ y salvarlos en *Búfer*
  end for
  \\ Una solución es una ecuación o desigualdad lineal inexacta
  SolucionesCandidatas ← k elementos óptimos en *Búfer* de acuerdo a Ec. 3.5
end procedure

Como podemos ver el sistema BPU-CILP utiliza ecuaciones y desigualdades lineales para crear nuevas cláusulas. El artículo correspondiente a este trabajo muestra algunos resultados sobre pequeños ejemplos, sin embargo no se discute sobre la precisión que puede tener el método propuesto con distintas bases de datos ILP.

3.2.3. Evaluación perezosa en Aleph

A Learning Engine for Proposing Hypotheses - Aleph, es un sistema ILP que está implementado en el lenguaje lógico Prolog [55]. Este sistema es capaz de emular algunas funcionalidades de los sistemas ILP más conocidos: Progol [49], FOIL [53], TILDE [6], etc. El algoritmo básico de Aleph selecciona de manera aleatoria un ejemplo e. Este ejemplo guía la construcción de la cláusula más específica (*bottom clause*) la cual se caracteriza por cubrir a e y por tener una gran cantidad de literales en su cuerpo. Después con el uso de esta cláusula específica se busca una que sea más general. Para este proceso Aleph incluye distintos algoritmos de búsqueda (primero en profundidad, primero en anchura, primero

el mejor, etc.) así como diversas funciones heurísticas para evaluar las cláusulas (entropía, el índice de Gini, Laplace, Bayes, etc.). Una vez que se elige la mejor cláusula entonces se añade a la teoría final y se eliminan los ejemplos cubiertos por la teoría. Este proceso se repite hasta que no haya más ejemplos por cubrir. Estos cuatro pasos se presentan en el Algoritmo 3.2.

Algoritmo 3.2 Algoritmo básico Aleph

1: Seleccionar un ejemplo e. Si no hay ejemplos entonces para.
2: Etapa de saturación. Construir la cláusula más específica que cubra a e.
3: Buscar la cláusula más general que la cláusula más específica.
4: Añadir la mejor cláusula a la teoría actual y remover los ejemplos redundantes. Ir al paso 1.

Este algoritmo básico no implementa procesos que manejen de manera eficiente los atributos numéricos y categóricos con muchos valores ya que, como se mencionó en la Sección 1.1, Aleph al igual que la mayoría de los algoritmos ILP prueban un solo valor por literal al construir las hipótesis candidatas. En [56] se propone un proceso de evaluación perezosa de funciones para la creación y evaluación de predicados numéricos. Esta estrategia se propone para mejorar las capacidades numéricas de los sistemas ILP. Implementado en Aleph, esta estrategia comprende los dos siguientes pasos:

Definición de predicados numéricos. El usuario del sistema ILP debe definir en el conocimiento previo una serie de predicados capaces de realizar cálculos numéricos, así como sus respectivas *declaraciones*. Las declaraciones le indican a Aleph como serán construidas las reglas: un símbolo $+$ indica un atributo de entrada que debe ser instanciado, $-$ es un atributo o argumento de salida que no es necesario instanciar, y $\#$ indica que el argumento es un valor constante. El meta-predicado *lazy_evaluation* está definido en Aleph para indicar los predicados numéricos que serán evaluados de manera perezosa. Por ejemplo, si deseamos que un argumento A sea discretizado, podemos definir el predicado *menor o igual a* con la declaración lteq(+A,#Split). Si este predicado será evaluado de forma perezosa entonces debemos definir el meta-predicado lazy_evaluate(lteq(+A,#Split)). Donde A es el argumento a ser discretizado y *Split* es un valor numérico que representa el punto de división que será calculado por el predicado *lteq/2*. El usuario puede definir diferentes tipos de predicados numéricos: funciones trigonométricas, de discretización o de regresión.

Evaluación perezosa de los predicados numéricos. La definición de predicados numéricos en el conocimiento previo no es suficiente para darle a Aleph mayor eficiencia en el manejo de información numérica, ya que para evaluar cada función se necesitaría un conjunto de valores relacionados a más de un ejemplo, sin embargo la búsqueda en Aleph está

guiada por un solo ejemplo. Debido a ello se define la evaluación perezosa que consiste en evaluar los predicados numéricos solo cuando sus resultados sean requeridos, por ejemplo antes de comenzar el proceso de búsqueda.

La eficiencia de la evalución perezosa en Aleph depende en gran medida de los predicados definidos por el usuario. Por lo tanto, es evidente que el usuario debe tener conocimientos de programación lógica, lo cual no es conveniente sobre todo si el usuario es investigador en otras áreas, ya que en este caso no tendrá idea de cómo llevar a cabo la definición de los predicados numéricos. Por otro lado, los autores de este trabajo no mencionan cómo definir predicados que permitan el manejo más eficiente de atributos categóricos.

3.3. Algoritmos genéticos en ILP

Los algoritmos genéticos son un conjunto de métodos de optimización y búsqueda, Sección 2.7. El principal objetivo de estos algoritmos en ILP es encontrar mejores teorías, ya que a diferencia de los métodos tradicionales, los sistemas ILP basados en algoritmos genéticos pueden crear y probar conjuntos de hipótesis a la vez [17]. En el caso del manejo de atributos numéricos y categóricos la idea es la misma. Crear y probar diferentes valores para un mismo atributo. En esta sección presentamos diferentes sistemas ILP que basan el procesamiento de atributos numéricos y categóricos en los algoritmos genéticos.

3.3.1. SIA01

El sistema *Supervised Inductive Algorithm version 01 - SIA01* [4] es un sistema de tipo bottom-up cuyo proceso de generalización es realizado por un algoritmo genético. El proceso principal está guiado por un ejemplo *Semilla* a partir del cual se encuentra la mejor generalización de acuerdo a la función de aptitud definida. Cada nueva generación es creada con dos operadores genéticos: mutación y cruzamiento. El proceso general del algoritmo de SIA01 es presentado en el algoritmo 3.3, en donde las variables n, *max* son proporcionadas por el usuario. La función de aptitud para evaluar cada cromosoma φ se basa en la consistencia y la completud de la cláusula actual. Es decir, que una cláusula que cubre más ejemplos negativos será penalizada, mientras que una cláusula que cubre más ejemplo positivos recibirá una calificación de aptitud más elevada. SIA01 representa cada cromosoma φ en un formato lógico, donde cada cláusula es un cromosoma y sus predicados y argumentos son sus genes. Por ejemplo, la cláusula $p(X, Y) \leftarrow r(X, 12, a)$ es representada con un cromosoma cuyos genes son: $p, X, Y, r, X, 12, a$.

Los operadores de mutación siguen el proceso de generalización del sistema SIA01. Si el gen del cromosoma a mutar es numérico, entonces se creará un intervalo numérico que

Algoritmo 3.3 Algoritmo SIA01

```
1:  procedure SIA01
2:      Poblacion ← Semilla
3:      repeat
4:          for ∀φ ∈ Poblacion do
5:              if φ no ha producido descendencia then
6:                  Crea 1 descendiente al mutar al cromosoma φ
7:                  Crea 2 descendientes al cruzar al cromosoma φ con φ'
8:                  Agregar a los descendientes a Poblacion'
9:              end if
10:         end for
11:         for ∀φ ∈ Poblacion' do
12:             if φ ∉ Poblacion then
13:                 if |Poblacion| < max ó la aptitud de φ es mejor que la peor en Poblacion then
14:                     Agrega φ en Poblacion
15:                 end if
16:             end if
17:         end for
18:     until La mejor aptitud de φ no cambie en las últimas n generaciones
19: end procedure
```

contendrá a ese gen. Si el gen es ya un intervalo numérico entonces lo crecerá. Por ejemplo, a la izquierda de la Figura 3.1 se muestra el caso de un gen numérico. Si el gen a mutar es categórico, entonces se añade un valor para crear un conjunto de valores categóricos. Si el gen es ya un conjunto entonces se agrega uno más. La Figura 3.1 muestra a la derecha este proceso para el caso categórico.

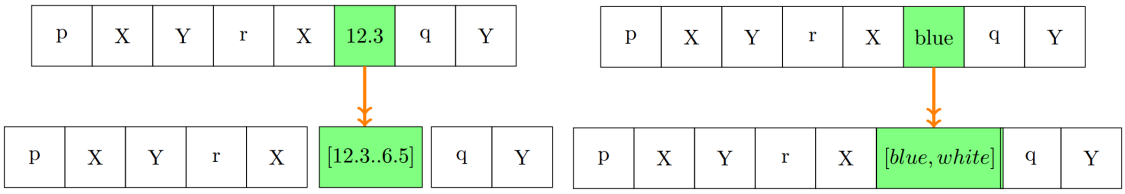

Figura 3.1: Mutación con un gen numérico y con un gen categórico.

El operador de cruzamiento de SIA01 intercambia los genes de dos cromosomas solo cuando dichos genes son del mismo tipo, es decir, si ambos genes son numéricos o

categóricos. Esto implica la verificación del tipo de los genes cada que se realiza el cruzamiento. La Figura 3.2 muestra este proceso para dos cromosomas que intercambian genes numéricos. Cabe añadir que los operadores genéticos solo manipulan genes que representan argumentos dentro de las literales de la hipótesis a generalizar.

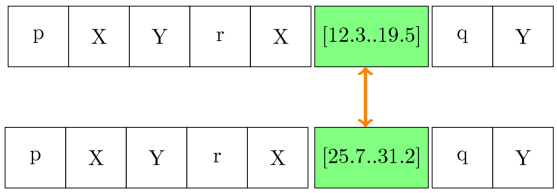

Figura 3.2: El operador de cruzamiento intercambia dos genes del mismo tipo.

Como podemos ver el manejo de atributos numéricos y categóricos es no supervisado ya que en el caso de la mutación los nuevos intervalos o subconjuntos son elegidos de manera aleatoria. Por otro lado este proceso es de tipo *bottom-up*, lo cual implica que potenciales intervalos o subconjuntos más específicos no sean creados y probados, ya que una vez que estos crecen ningún operador los especializa.

3.3.2. SMART+

En [8] se presenta *SMART+* un sistema ILP cuya estrategia de aprendizaje es de tipo *top-down.* Este sistema cuenta con dos estrategias para el manejo de atributos numéricos. La primera es local y se ejecuta al especializar una hipótesis. La segunda está basada en una estrategia evolutiva y se lleva a cabo después del proceso de especialización. A continuación describimos estos dos métodos.

Local. Para especializar una cláusula o hipótesis φ, SMART+ añade una literal l al cuerpo de φ. Si l tiene un conjunto de argumentos numéricos $k_1, k_2, \ldots, k_n$ entonces define para cada k_i una tupla de tres entradas *(mín,máx,st)* donde $[$*mín..máx*$]$ es el intervalo de k_i y st su rango de variabilidad. Después SMART+ inicializa cada argumento k_i con su valor mínimo: $k_1 = mín_1, k_2 = mín_2, \ldots, k_n = mín_n$. Estos valores representan la combinación inicial de los argumentos de l y lo denotamos como t_1. El siguiente paso es calcular la ganancia de información de t_1: $ganancia(t_1) = g_1$. Si $g_1 > 0$ entonces la correspondiente combinación se añade a una lista *BEST.* Esta lista contendrá las mejores combinaciones numéricas para l. Después se generan nuevas combinaciones al variar el valor de un argumento k_i a la vez. Un nuevo valor para algún k_i se obtiene al sumar su rango de variabilidad st. Cuando $k_i = máx_i$ entonces se varía el argumento siguiente. Para cada combinación se calcula su ganancia de información y si ésta es mayor que la de la última combinación añadida a BEST, entonces también se agrega a la lista. Cuando el último argumento llega

a su máximo valor, entonces se detiene el proceso. La tabla 3.4 muestra de manera general la construcción de la lista BEST. Finalmente, las combinaciones en BEST son usadas para crear las hipótesis candidatas al especializar φ.

Combinación	Ganancia de información
$k_1 = min_1, k_2 = min_2, \ldots, k_n = min_n$	g_1
$k_1 + step_1, k_2 = min_2, \ldots, k_n = min_n$	g_2
$k_1 + (2 * step_1), k_2 = min_2, \ldots, k_n = min_n$	g_3
$\vdots$	$\vdots$
$k_1 = max, k_2 = min_2, \ldots, k_n = min_n$	g_r
$k_1 = max, k_2 + step_2, \ldots, k_n = min_n$	g_{r+1}
$\vdots$	$\vdots$
$k_1 = max, k_2 = max_2, \ldots, k_n = max_n$	g_{final}

Tabla 3.4: La lista BEST contiene las mejores combinaciones de valores numéricos para un conjunto de argumentos en una literal. Por definición de esta lista, cada combinación tiene una ganancia de información mayor que la anterior.

Global. En este caso SMART+ usa un algoritmo genético basado en el que describe Goldberg [27] para optimizar los valores de todos los argumentos numéricos de una cláusula φ. Este GA se ejecuta después del proceso de especialización. El GA toma todos los valores numéricos de φ como una cadena de valores reales $rs = k_1 k_2 \ldots k_n$ la cual es transformada a una binaria para representar un cromosoma. Para que la transformación de una cadena de reales *rs* a una binaria *bs* no genere valores fuera de rango para algún k_i se siguen los dos siguientes pasos.

a. Dado un argumento k_i y su rango $[min_i, max_i]$, entonces k_i es representado de la siguiente manera:

$$k_i = min_i + \delta_i / \Delta \tag{3.8}$$

Donde $0 \leq \delta_i \leq 2^{N_i}$, N_i es el número de bits elegidos por el usuario para representar el incremento δ_i y $\Delta_i = \frac{2^{N_i}}{max_i - min_i}$.

b. Dada una cadena $rs = k_1 k_2 \ldots k_n$, para cada argumento k_i en *rs*, un conjunto de N_i bits será definido en *bs*, donde el valor del correspondiente δ_i está representado.

Una vez definido el procedimiento para convertir cadenas numéricas a cadenas binarias, el algoritmo genético utilizado por SMART+ genera una población $A(\varphi)$ donde se calcula la ganancia de información de cada cromosoma. Para la creación de la nueva población se definen dos operadores de cruzamiento, el primero sigue el proceso estándar propuesto por Goldberg, y el segundo consta de los siguientes tres pasos:

Dados dos cromosomas (padres) bs_1 y bs_2 y un parámetro k_c seleccionado aleatoriamente, los descendientes bs'_1 y bs'_2 son generados en tres pasos:

1. Los parámetros a la izquierda de k_c en bs'_1 y bs'_2 son obtenidos a partir de bs_1 y bs_2 respectivamente.

2. El parámetro k_c en ambos descendientes se obtiene promediando los dos correspondientes valores en ambos padres.

3. Los parámetros del lado derecho de k_c son obtenidos en bs'_1 al copiar los correspondientes valores en bs_2 y viceversa para bs'_2.

Para seleccionar entre los dos operadores de cruzamiento c_1 y c_2, se definen dos probabilidades p_1 y p_2 respectivamente, con la restricción $p_1 + p_2 \leq 1$. El operador de mutación es el estándar, es decir, se eligen un conjunto de genes de manera aleatoria y sus valores son cambiados. Este operador es aplicado con una probabilidad $p_m = 0.001$.

El mecanismo de SMART+ para el manejo de atributos numéricos es doble ya que ejecuta un proceso local y uno global, esto puede ser ventajoso respecto a otros sistemas que llevan a cabo solo un tipo de procesamiento numérico. Sin embargo, el algoritmo local de SMART+ genera y prueba demasiados valores cuando las literales tienen muchos argumentos numéricos. Por otro lado, el proceso global aprovecha las ventajas de los algoritmos genéticos, pero a diferencia de otros sistemas ILP que utilizan restricciones, las hipótesis finales solo declaran un solo valor numérico por argumento. Lo cual representa uno de los problemas que describimos en la Sección 1.

3.3.3. Manejo de atributos numéricos en el sistema ECL

El sistema *Evolutionary Concept Learning - ECL system [18]* es un aprendiz basado en algoritmos genéticos que permite la inducción de teorías en lenguaje de lógica de primer orden. Este sistema utiliza cuatro métodos de discretización para el manejo de atributos numéricos: discretización global supervisada - GSD, discretización local no supervisada - LUD, discretización local supervisada con inicialización de *grano fino* - LSDf y discretización local supervisada con inicialización de *grano grueso* - LSDc. Este sistema y sus métodos de discretización son descritos a continuación.

Sistema ECL

ECL es un sistema ILP basado en algoritmos genéticos con un alto nivel de codificación en el cual una cláusula representa a un cromosoma y cada literal es un gen. Por ejemplo, la cláusula $p(X, Y) \leftarrow r(X, 12, a)$ es un cromosoma de 2 genes. Este tipo de

codificación permite aplicar operadores de refinamiento y de generalización a una cláusula dada. El algoritmo de ECL tiene un ciclo *repeat* principal que crea de manera iterativa un conjunto de cromosomas llamado *PoblacionFinal*. De este conjunto de cromosomas se crea la teoría final. En cada iteración de este ciclo se toma de manera aleatoria un subconjunto de hechos del conocimiento previo. Este conocimiento previo parcial junto con un conjunto de ejemplos *Sel* son usados por un algoritmo genético, en este caso representado dentro de un ciclo *while*, para evolucionar una población. La función de aptitud de cada cromosoma es la inversa de su precisión, es decir, el número total de ejemplos dividido por el número de ejemplos correctamente clasificados por el cromosoma. Los operadores genéticos usados por ECL son selección, mutación y optimización. El operador de selección está basado en el *operador de selección sufragio universal* en el cual los ejemplos positivos pueden ser vistos como votantes y los cromosomas como candidatos a ser elegidos. Además todos los cromosomas tienen la misma probabilidad de ser elegidos. El operador de mutación es dual y puede borrar o añadir una literal del cuerpo de una cláusula. El operador de optimización es la aplicación repetida de operadores de especialización y generalización mientras la aptitud del cromosoma se mantenga o mejore y hasta que se alcance el número máximo de aplicaciones de este operador. El algoritmo 3.4 muestra el proceso general del sistema ECL.

Algoritmo 3.4 Algoritmo ECL

```
1:  procedure ECL
2:      Sel ← EjemplosPositivos
3:      repeat
4:          Seleccionar conocimiento previo parcial
5:          Poblacion ← φ
6:          while No termina do
7:              Cromosomas ⊆ Sel
8:              for all Crom ∈ Cromosomas do
9:                  Mutar y Optimizar Crom
10:                 Insertar Crom en Poblacion
11:             end for
12:         end while
13:         Almacenar Poblacion en PoblacionFinal
14:         Sel ← Sel - EjemplosPositivosCubiertos
15:     until Máximo número de iteraciones
16:     Extraer Teoría de PoblacionFinal
17: end procedure
```

Cuando las literales tienen argumentos numéricos se ejecuta alguna de las siguientes estrategias para el manejo de información numérica.

ECL-GSD

La estategia ECL-GSD es la misma estrategia utilizada por los sistemas TILDE e ICL, ver Sección 3.1.2. Como se vio anteriormente, este método consiste en la discretización de los atributos numéricos antes del proceso de aprendizaje. Este proceso es conocido como discretización global y los puntos de división obtenidos son utilizados durante todo el proceso de aprendizaje junto con las relaciones de desigualdad *mayor o igual que* y *menor o igual que*. Al igual que en los sistemas TILDE e ICL, el algoritmo de discretización es el que está basado en el principio de longitud mínima o MDLP.

ECL-LUD

Los operadores de mutación o de optimización del algoritmo básico de ECL generalizan o especializan una cláusula. Cuando esta última operación se ejecuta, el operador correspondiente añade una literal C al cuerpo de la cláusula. Si C tiene argumentos numéricos entonces se añade también una desigualdad por cada uno de estos argumentos. Dicha desigualdad tiene la forma $a \leq X \leq b$ donde a, b son constantes contenidas en el conocimiento previo y además $a \neq b$. Es decir, que esta inicialización es un punto dentro del rango de X. Esta estrategia para tratar con los argumentos numéricos en ECL es llamada *discretización local no supervisada - LUD* y define cinco operadores de mutación, los cuales utilizan la información de la distribución de cada atributo. Esta información es obtenida antes del proceso de aprendizaje con el algoritmo *Expectation Maximization - EM [15].* El algoritmo EM devuelve n grupos o *clusters* descritos por una media μ_i y una desviación estándar σ_i, donde $1 \leq i \leq n$. A continuación describimos los operadores de mutación que actúan sobre los atributos numéricos. Consideremos la desigualdad $I = a \leq X \leq b$, entonces los siguientes operadores de mutación modifican los valores de I para modificar un cromosoma.

- *Enlarge.* Este operador aplicado a I devuelve el intervalo $I' = a' \leq X \leq b'$, donde $a' \leq a$ y $b \leq b'$.
- *Shrink.* Aplicado a I este operador devuelve el intervalo $I' = a' \leq X \leq b'$ donde $a' \geq a$ y $b \geq b'$.
- *Shift.* Este operador aplicado a I devuelve el intervalo $I = a' \leq X \leq b'$ donde a' y b' son puntos del cluster que contiene a a, b tal que $P(a' \leq X \leq b') = P(a \leq X \leq b)$.

- *Cluster change.* Este operador aplicado a I devuelve el intervalo $I' = a' \leq X \leq b'$ donde a', b' pertenecen a un grupo o cluster diferente.

- *Ground.* Este devuelve un intervalo $I' = a' \leq X \leq b'$ donde sólo a' se encuentra en el mismo grupo que contiene a a, b.

ECL-LSDf y ECL-LSDc

En la estrategia ECL-LUC los valores iniciales para cada desigualdad $a \leq X \leq b$, donde X es un argumento de una literal utilizada para especializar una cláusula, equivale a un punto de X, es decir $a = b$. En [18] los autores presentan dos variantes para inicializar los valores de a, b. La primera es llamada *discretización local supervisada con inicialización de grano fino - LSDf* y en esta los valores iniciales de a, b pertenecen al conjunto de puntos frontera de X denotado como $BP(X)$. Un *punto frontera* es el punto medio entre dos valores consecutivos de X. La segunda inicialización es llamada *discretización local supervisada con inicialización de grano grueso - LSDc* y en este caso los valores iniciales de a, b pertenecen al conjunto de puntos de discretización de X, denotado como $DP(X)$, obtenidos con el algoritmo MDLP mencionado anteriormente.

A diferencia de los sistemas SIA01 y SMART+, la estrategias evolutivas del sistema ECL para procesamiento de los atributos numéricos permite tanto la especialización como la generalización de las cláusulas ya sea reduciendo o aumentando el intervalo de cada atributo numérico. Si bien esta característica dual le permite explorar un espacio más amplio de hipótesis no permite expresar relaciones entre atributos numéricos en caso de que existan. Además ECL tampoco implementa estrategias para el manejo de atributos categóricos de tal manera que las hipótesis puedan expresar cada uno de estos atributos con más de un valor.

3.4. Proposicionalización

La proposicionalización de manera general es una técnica para transformar un problema relacional a un formato atributo-valor (Sección 2.5). El objetivo principal es utilizar aprendices proposicionales para tratar problemas difíciles de tratar con algoritmos relacionales. A continuación presentamos dos técnicas de proposicionalización capaces de tratar atributos numéricos.

3.4.1. Sistema LINUS

LINUS [38] es un sistema ILP que a partir de un conjunto de hechos E y un conjunto de cláusulas de base de datos deductiva o DDB[2] como conocimiento previo B induce un conjunto de cláusulas de bases de datos jerárquico deductiva y restringida o DHDB[3]. Los tres principales pasos para realizar este aprendizaje inductivo son: transformar el problema relacional a una forma atributo-valor, inducir una hipótesis usando un aprendiz proposicional y transformar la hipótesis inducida a una forma relacional. Estos procesos son presentados a continuación.

La transformación del problema relacional a uno proposicional genera nuevos atributos a partir de las relaciones del conocimiento previo. Cada nuevo atributo es la aplicación de un predicado del conocimiento previo sobre los argumentos de la relación objetivo, tomando en cuenta el tipo de argumento. Por ejemplo, si la tarea consiste en inducir una teoría que explique la relación *hija/2* a partir de las relaciones *progenitor/2*, *hombre/2* y *mujer/2*, como se muestra en la tabla 3.5, una aplicación del predicado *progenitor* sobre la relación *hija* es la siguiente *hija(X,X)←progenitor(X,X)* donde los dos argumentos son iguales en las dos relaciones. En este caso *progenitor(X,X)* representa un nuevo atributo binario el cual toma valor 1 si la aplicación es verdadera al tomar X los valores constantes declarados en el conocimiento previo y toma el valor 0 en caso contrario. Otra aplicación se da cuando los argumentos son diferentes: *hija(X,Y)←progenitor(X,Y)*. Esta última aplicación también genera un atributo binario cuyos valores dependen de las constantes declaradas en el conocimiento previo y en los ejemplos. La tabla 3.6 muestra todos los atributos que resultan de la tarea de aprendizaje definida. En esta tabla $h \equiv$ *hombre*, $m \equiv$ *mujer* y $p \equiv$ *progenitor*.

Ejemplos	Conocimiento previo		
hija(sue,eve). $\oplus$	progenitor(eve,sue)	mujer(ann).	hombre(pat).
hija(and,pat). $\oplus$	progenitor(ann,tom).	mujer(sue).	hombre(tom).
hija(tom,ann). $\ominus$	progenitor(pat,ann).	mujer(eve).	
hija(eve,ann). $\ominus$	progenitor(tom,sue).		

Tabla 3.5: Ejemplos y conocimiento previo para aprender la relación *hija/2*.

Una vez que se ha creado el nuevo conjunto de entrenamiento, un aprendiz atributo-valor genera la hipótesis correspondiente. El sistema LINUS cuenta con tres aprendices

[2]Una base de datos deductiva es un conjunto de cláusulas de base de datos. Una cláusula de base de datos es una cláusula de programa de la forma. $T \leftarrow L_1, L_2, \ldots, L_m$ donde T es un átomo y $L_1, L_2, \ldots, L_m$ son literales.

[3]Una base de datos jerárquica deductiva es una base de datos deductiva que está restringida a definiciones de predicados no recursivos y tipos de datos no recursivos.

	Variables		Atributos proposicionales							
Clase	X	Y	m(X)	m(Y)	h(X)	h(Y)	p(X,X)	p(X,Y)	p(Y,X)	p(Y,Y)
$\oplus$	sue	eve	1	1	0	0	0	0	1	0
$\oplus$	ann	pat	1	0	0	1	0	0	1	0
$\ominus$	tom	ann	0	1	1	0	0	0	1	0
$\ominus$	eve	ann	1	1	0	0	0	0	0	0

Tabla 3.6: Tabla atributo-valor resultante para el problema de aprendizaje de la relación *hija/2*.

de este tipo. *ASSISTANT.* es un algoritmo que es una extensión del algoritmo ID3 y que genera reglas en forma de árboles de decisión. Los árboles generados por ASSISTANT son modelos de clasificación entre dos clases: ejemplos positivos y negativos en este caso. Una de las mejoras de ASSISTANT respecto a ID3 es el manejo de atributos numéricos, ya que hace uso de una estrategia de discretización para los atributos numéricos y al igual que ID3 la clasificación de los ejemplos está basada en la ganancia de información.

El segundo sistema es *NEWGEM* el cual induce un conjunto de reglas (una por cada clase) llamado hipótesis. Al construir una regla, NEWGEM considera los objetos de la clase correspondiente como ejemplos positivos y el resto como negativos. Las reglas inducidas son de tipo *si-entonces* y están descritas con el lenguaje VL_1 la cual es una variante de VL_{21} usado por el sistema INDUBI/CSL (Sección 3.1.1) en la cual los selectores relacionan un atributo con un valor p.e. $[color = \textit{'rojo'}]$ o con una disyunción de valores como el siguiente $[color = \textit{'rojo'} \vee \textit{'azul'}]$. Además una conjunción de selectores es llamada *compleja* A diferencia de la mayoría de los aprendices atributo-valor, NEWGEM sigue una estrategia de cobertura en la cual se siguen de manera iterativa los tres pasos siguientes: construir una regla, añadir la regla a la hipótesis actual y eliminar los ejemplos positivos cubiertos por la regla. Si no hay más ejemplos positivos entonces se detiene el proceso y se devuelve la hipótesis. La construcción de cada regla se lleva a cabo con un algoritmo *beam search* cuyos criterios (de mejor a peor) para seleccionar la mejor regla son: mayor número de ejemplos positivos cubiertos, menor número de condiciones en el cuerpo de cada regla y menor número de valores en cada selector.

Por último, el tercer sistema proposicional es el de inducción de reglas llamado *CN2*. Este sistema trata de manera eficiente el ruido en los datos y utiliza también un algoritmo beam search para la construcción de reglas, aunque CN2 no depende de un ejemplo positivo usado como semilla para guiar el proceso de búsqueda. El algoritmo principal de CN2 consta de un ciclo principal el cual toma el conjunto de entrenamiento actual (al inicio es el conjunto de entrenamiento completo) y la hipótesis actual H (que al principio es un conjunto vacío de reglas) y genera una regla. En cada iteración se crea una regla por cada

clase y para ello toma los ejemplos de la clase correspondiente como positivos y el resto como negativos. Las cláusulas son evaluadas con la precisión de clasificación de *Laplace*, la cual estima la probabilidad de que un ejemplo cubierto por la regla actual sea positivo.

El tercer paso que ejecuta LINUS es la transformación de la reglas proposicionales del tipo *si-entonces* a una forma relacional. Por ejemplo para el problema de la relación *hija/2* al obtener la regla *si progenitor(Y,X)*$= 1 \wedge$ *mujer(X)* $= 1$ *entonces clase* $= \oplus$ su equivalente a formato relacional es *hija(Y,X)* $\leftarrow$ *progenitor(Y,X)* $\wedge$ *mujer(X).*

El manejo de información numérica en LINUS depende del aprendiz proposicional utilizado y del lenguaje permitido para las literales del cuerpo de las cláusulas. En primer lugar el aprendiz proporciona el algoritmo de discretización obtener los puntos de división del correspondiente atributo y el lenguaje de las literales permite expresar esos puntos de división dentro de las cláusulas finales. En el caso de LINUS las literales permitidas son de los siguientes tipos: una variable igual a un valor constante $X = a$, una igualdad de variables $X = Y$, un predicado con argumentos de entrada de tipo variable las cuales deben ocurrir en la cabeza de la cláusula $p(X, Y)$ y un átomo con un símbolo de predicado con argumentos de entrada de tipo variable las cuales deben ocurrir en la cabeza de la cláusula y argumentos de salida los cuales deben ser instanciados $p(X, Y, Z), Z = 4$. De estas literales permitidas en LINUS la primera y la cuarta pueden tomar la forma de desigualdades, por ejemplo $X > s$ y $X < s$ donde s es un punto de división obtenido al discretizar la variable X.

Para el manejo de atributos numéricos, LINUS nos permite tomar ventaja de casi cualquier aprendiz proposicional (en esencia de aquellos cuyas hipótesis son de tipo *si-entonces*) que pueda procesar este tipo de información eficientemente. Sin embargo, al transformar un problema relacional los atributos suelen ser binarios, por lo que gran parte de la información numérica se pierde en este proceso. Por otro lado una desventaja es la limitación en cuanto al lenguaje permitido por LINUS, ya que éste no permite la definición de predicados recursivos.

3.4.2. Cardinalización y cuantiles

En el trabajo presentado en [1] se proponen dos técnicas para proposicionalizar información relacional con atributos numéricos. La primera variante es llamada *cardinalización* y para crear nuevos atributos proposicionales utiliza la cardinalidad entre la tabla primaria (donde se encuentran los ejemplos) y una tabla secundaria, la cual contiene al menos un atributo numérico. La segunda técnica crea un número fijo de atributos proposicionales de acuerdo al número de cuantiles declarado por el usuario. El número de cuantiles representa el número de nuevos atributos proposicionales que serán creados y en los cuales se distribuirán los valores numéricos del atributo numérico en turno relacionados con cada

ejemplo. Estos dos métodos son presentados a continuación.

La *cardinalización* es una técnica de proposicionalización basada en la cardinalidad entre los ejemplos y los atributos numéricos. Por ejemplo, en la Figura 3.3 se muestra una tabla primaria con un conjunto de ejemplos etiquetados con su respectiva clase y una tabla secundaria con un atributo numérico A. La cardinalidad en este ejemplo es $1:4$, es decir, cada ejemplo puede estar relacionado hasta con cuatro valores numéricos de A. En lenguaje lógico esto equivale a tener los hechos *primaria(a,neg)*, *primaria(c,pos)*, *primaria(b,pos)*, *secundaria(a,13.5)*, *secundaria(a,12.2)*, etc. En [19] se presenta una tabla de equivalencia entre la terminología de las base de datos relacionales y de programación lógica.

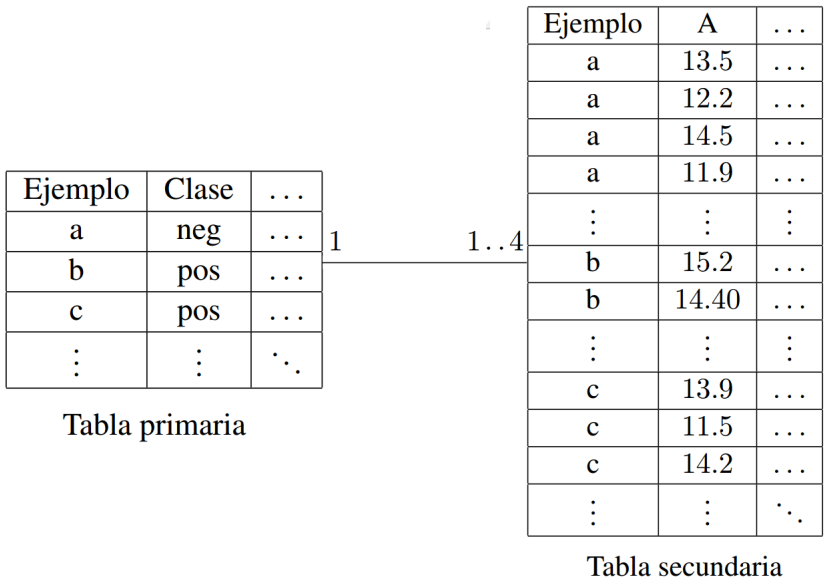

Figura 3.3: Cardinalidad entre una tabla primaria y una secundaria que contiene al menos un atributo numérico A.

Para crear atributos proposicionales con información como la mostrada en la Figura 3.3, la cardinalización ordena en forma creciente los valores del argumento numérico A relacionados con cada ejemplo. En nuestro caso tenemos para el ejemplo a los valores ordenados: $11.9, 12.2, 13.5, 14.5$. Una vez ordenados los valores de cada ejemplo, se crean M atributos proposicionales, donde M es la cardinalidad entre la tabla primaria y la secundaria. Cada nuevo atributo es nombrado *min_k_Atr*, donde $1 \leq k \leq M$ y *Atr* es el correspondiente atributo numérico. En el ejemplo presentado $M = 4$ y *Atr*=*A*. El valor del atributo *min_k_Atr* es el $k-$*ésimo* valor del correspondiente ejemplo que está relacionado

con *Atr*. Siguiendo con el problema planteado anteriormente, tenemos que *min_2_A*$= 12.2$ para el segundo valor del ejemplo a, *min_2_A*$= 14.4$ para el segundo ejemplo del ejemplo b y así sucesivamente. Hay algunos ejemplos cuya cardinalidad con el atributo numérico es menor a M. En este caso el valor del atributo proposicional para dicho ejemplo es ∞. La Tabla 3.7 muestra los atributos proposicionales creador a partir de los datos mostrados en la Figura 3.3.

Ejemplo	*min_1_A*	*min_2_A*	*min_3_A*	*min_4_A*	*Clase*
a	11.9	12.2	13.5	14.5	neg
b	14.4	15.2	∞	∞	pos
c	11.5	13.9	14.2	∞	pos
⋮	⋮	⋮	⋮	⋮	⋮

Tabla 3.7: Ejemplo de atributos creados con cardinalización.

El prefijo *min* en el nombre de cada atributo proposicional se debe a la definición de cardinalización la cual indica que el valor del nuevo atributo *min_k_Atr* es el valor mínimo del umbral de *Atr* tal que al menos k tuplas t tienen un valor $V_A(t)$, para el atributo, menor o igual a dicho umbral. El Algoritmo 3.5 presenta el proceso de cardinalización para todos los atributos numéricos de una tabla secundaria.

El segundo método de proposicionalización está basado en los cuantiles. Este método distribuye los valores de un atributo numérico A, relacionados a un ejemplo, en n cuantiles. En este caso n es un parámetro que el usuario debe asignar. Cada nuevo atributo proposicional es denotado *min_k_n_A* donde los valores de A relacionados a un solo ejemplo de la tabla primaria con aquellos que caen en el *k-ésimo* cuantil, tal que la proporción de dichos valores son al menos $\frac{k}{n}$. Tomando el mismo ejemplo presentado para la cardinalización, si tenemos $n = 4$ entonces se crearán cuatro atributos proposicionales. La Tabla 3.8 muestra los atributos creados al partir la distribución de los valores de A en cuantiles. De los tres ejemplos que tenemos en la tabla primaria es claro que los dos únicos valores para b se deben duplicar para poderse distribuir en los cuantiles. En el caso del ejemplo c solo uno fue duplicado. Notemos que a diferencia de la cardinalización no se tienen valores ∞. El proceso para la creación de atributos proposicionales con cuantiles está dado en el Algoritmo 3.6.

La cardinalización como la técnica de los cuantiles permiten crear un conjunto de datos atributo-valor, es decir, en forma de tabla, de tal manera que le dejan al aprendiz proposicional elegir los puntos de división más adecuados. Sin embargo, la cardinalización podría crear demasiados atributos nuevos sobre todo si la cardinalidad entre la tabla primaria y la secundaria es muy grande. La técnica de los cuantiles resuelve esta problemática al fijar

Algoritmo 3.5 Algoritmo cardinalización

```
1: procedure CARDINALIZACIÓN(Tabla primaria P, tablas secundarias TS)
2:     Sea M la cardinalidad entre P y TS
3:     Inicializar la tabla T con las columnas de P
4:     for all Atributo numérico A en TS do
5:         for k ← 1, M do
6:             Agregar una columna en T con nombre min_k_A
7:         end for
8:         for all Ejemplo i en P do
9:             Sea E el conjunto de valores en A relacionados con i
10:            num = |E|
11:            Ordena E en forma ascendente de acuerdo a V_A (conjunto de valores de A)
12:            for all j ← 1, num do
13:                Llena columna min_j_A con V_A(j)
14:            end for
15:            for all j ← num + 1, M do
16:                Llena columna min_j_A con ∞
17:            end for
18:        end for
19:    end for
20: end procedure
```

Ejemplo	*min_1_4_A*	*min_2_4_A*	*min_3_4_A*	*min_4_4_A*	*Clase*
a	11.9	12.2	13.5	14.5	neg
b	14.4	14.4	15.2	15.2	pos
c	13.9	11.5	14.2	14.2	pos
⋮	⋮	⋮	⋮	⋮	⋮

Tabla 3.8: Ejemplo de atributos creados con cuantiles.

un número de cuantiles, sin embargo esto crea exceso de información ya que muchos de los valores deben duplicarse para poder ser distribuidos en los cuantiles correspondientes.

Algoritmo 3.6 Algoritmo cuantiles

procedure CUANTILES(Tabla primaria P, tablas secundarias *TS*, número de cuantiles n)
  Inicializar la tabla T con las columnas de P
  for all Atributo numérico A en *TS* **do**
    for $k \leftarrow 1, n$ **do**
      Agregar columna en T con nombre *min_k_n_A*
    end for
    for all Ejemplo i en la tabla P **do**
      Sea E el conjunto de valores en A relacionados con i
      $num = |E|$
      Ordena E en forma ascendente de acuerdo a V_A (conjunto de valores de A
      $k \leftarrow 1$
      for $j \leftarrow 1, num$ **do**
        while $j \leq ceiling(\frac{k*num}{n}) < j + 1$ **do**
          Llena columna *min_k_n_A*
          $k \leftarrow k + 1$
        end while
      end for
    end for
  end for
end procedure

3.5. Resumen

En este capítulo hemos revisado diferentes estrategias para el procesamiento de atributos numéricos y categóricos en ILP. Dentro de estas identificamos aquellas que están basadas en la discretización, en la programación lógica de restricciones, en el razonamiento numérico con evaluación perezosa, en los algoritmos genéticos y en la proposicionalización. Para cada estrategia tomamos en cuenta también el número de atributos analizados (univariable o multivariable), las características del algoritmo ILP y de la estrategia de procesamiento. En la Tabla 3.9 mostramos el tipo y número de atributos que maneja la estrategia de cada algoritmo ILP revisado.

Como podemos ver en la Tabla 3.9 todos los métodos revisados procesan atributos numéricos y solo tres de ellos lo hacen también con atributos numéricos: aprendizaje multivalores, SIA01 y LINUS, aunque este último procesa información en formato atributo-valor. TILDE, ICL y aprendizaje multivalores utilizan métodos globales de discretización,

No.	Sistema ILP	Atributos numéricos	Atributos categóricos	Univariable	Multivariable
1	INDUBI/CSL	✓	χ	✓	χ
2	TILDE e ICL	✓	χ	✓	χ
3	Aprendizaje multivalores	✓	✓	✓	χ
4	NUM	✓	χ	✓	✓
5	BPU-CILP	✓	χ	χ	✓
6	Evaluación perezosa	✓	χ	✓	✓
7	SIA01	✓	✓	✓	χ
8	SMART+	✓	χ	✓	χ
9	ECL	✓	χ	✓	χ
10	LINUS	✓	✓	✓	✓
11	Cardinalización y Cuantiles	✓	χ	✓	✓

Tabla 3.9: Resumen del tipo y número de atributos que maneja cada sistema ILP revisado y el número.

pero una desventaja de estos es que generan los puntos de división usando todo el conjunto de ejemplos y éste suele cambiar durante el proceso de aprendizaje. Por lo tanto es posible que potenciales puntos de división no sean creados ni probados. Los sistemas que utilizan algoritmos genéticos (SIA01 y SMART+ y ECL) superan esta dificultad ya que generan y prueban un mayor número de valores cada vez que se aprende una nueva regla. Aunque en el caso de las estrategias presentadas, el manejo de atributos está dado dentro del mismo operador de refinamiento o especialización. Esto dificulta la implementación de dichas estrategias en otros sistemas ILP cuyos algoritmos de inducción (top-down o bottom-up) y representación de cromosomas sean diferentes (cadenas binarias o en lógica de primer orden).

El algoritmo NUM y la evaluación perezosa en ILP mejoran significativamente las capacidades numéricas de los sistemas ILP, ya que se pueden definir y usar distintos tipos de funciones numéricas (lineales, cuadráticas, trigonométicas o más especializadas). En el caso de NUM el usuario define la forma de la literal numérica y NUM utiliza el conocimiento previo para crear sistemas de ecuaciones con la forma definida, finalmente un sistema de resolución de restricciones en formato lógico resuelve estos sistemas para crear literales numéricas. Por otro lado, la evaluación perezosa permite definir dentro del conocimiento previo las relaciones numéricas necesarias para el manejo de atributos numéricos y éstas son evaluadas durante el proceso de aprendizaje. Sin embargo es difícil el uso tanto de NUM como de la evaluación perezosa, ya que el usuario debe definir y codificar los

predicados numéricos. Por lo tanto su eficiencia depende en gran medida de que el usuario tenga al menos conocimientos básicos de programación lógica.

La proposicionalización es otra técnica utilizada no solo para el manejo de atributos numéricos y categóricos sino además para tratar, con aprendices proposicionales, problemas difíciles de resolver en ILP como el ruido en los datos o los valores desconocidos en las bases de datos *(missing values)*. Pero la eficiencia que pueden tener los aprendices proposicionales utilizados se puede ver limitada por ciertos factores, por ejemplo LINUS crea un conjunto de atributos binarios a partir de relaciones en el conocimiento previo por lo cual los aprendices no pueden crear particiones para los atributos numéricos. La cardinalización y los cuantiles sí crean atributos proposicionales numéricos a partir de los originales, sin embargo en algunos casos generan una gran cantidad de ellos o un exceso de información lo cual puede afectar la rapidez de los aprendices atributo-valor. Además, sin importar la técnica cualquier algoritmo de proposicionalización puede provocar pérdida de información al transformar datos relacionales a proposicionales.

Tomando en cuenta lo anterior, en este trabajo proponemos una estrategia univariable para el manejo de atributos numéricos y categóricos en ILP. Este método utiliza algoritmos genéticos para crear y probar puntos numéricos de división y subconjuntos de valores categóricos, los cuales serán evaluados con una función de aptitud. Después los mejores calificados serán añadidos junto con nuevas relaciones al conocimiento previo. Este proceso se lleva a cabo antes de construir una nueva regla, por lo tanto no depende ni del proceso de búsqueda ni de la estrategia de inducción (ya sea *bottom up* o *top down*) ni tampoco de la implementación de los operadores de refinamiento. Nuestra conjetura como se mencionó en la Sección 1.1 es que la nueva información añadida al conocimiento previo mejorará las teorías aprendidas sin tener un aumento exponencial en el tiempo de ejecución.

Decidimos implementar nuestra propuesta, presentada en el capítulo siguiente, en el sistema Aleph [55] ya que tiene un eficiente sistema declarativo que facilita la adición de nuevo conocimiento previo, además este sistema está implementado en el lenguaje Prolog el cual es *software* libre.

Capítulo 4

Método evolutivo para el manejo de atributos numéricos y categóricos en ILP

> "Los malos tiempos tienen un valor científico. Son ocasiones que un buen alumno no se perdería."
>
> Ralph Waldo Emerson

En este capítulo presentamos un método para el manejo de atributos numéricos y categóricos en ILP llamado *Grouping and Discretization for Enriching the Background Knowledge- GDEBaK*, el cual, fue presentado en el artículo *"An evolutionary-based approach for dealing with numerical and categorical attributes in ILP" [51].* GDEBaK utiliza algoritmos genéticos para encontrar puntos de división en el caso de los atributos numéricos, y subconjuntos de valores en el caso de los atributos categóricos, que al agregarse al conocimiento previo mejoran la calidad de las teorías construidas por un algoritmo ILP. En este contexto una teoría de mejor calidad tendrá menos reglas y mayor precisión.

4.1. Método GDEBaK

Como se mencionó anteriormente el método GDEBaK permite un mejor manejo de los atributos numéricos y categóricos en ILP para obtener teorías con mayor precisión y menor número de reglas. Para ello incrustamos a GDEBaK en el proceso general de aprendizaje ILP como se muestra en el algoritmo 4.1. Como en cualquier sistema ILP

los argumentos de entrada son un conjunto de ejemplos positivos E^+, un conjunto de ejemplos negativos E^- y un conocimiento previo B, pero además, incluimos el parámetro *declaracionAtributos*. Este último es escrito en el formato declarativo del sistema ILP correspondiente e indica cuáles atributos serán procesados.

El proceso general es el siguiente. En cada iteración del ciclo *while*, antes de aprender la siguiente regla, se ejecuta el método GDEBaK (Paso 4). Este crea, por medio de un algoritmo genético, un conjunto de puntos de división por cada atributo numérico, y subconjuntos de categorías por cada atributo categórico. Estos son añadidos al conocimiento previo junto con nuevas relaciones que permiten al sistema ILP usar dicha información, esto lo denotamos en el paso 4 como una asignación a B. El aprendizaje de la siguiente hipótesis (Paso 5) se lleva a cabo con la nueva información añadida al conocimiento previo. El resto del proceso inductivo se ejecuta de acuerdo al sistema ILP utilizado pero con el conocimiento previo enriquecido. Cabe añadir que la diferencia más importante entre los distintos sistemas ILP radica en los algoritmos con los que se aprende una hipótesis (Paso 5). Estos sistemas utilizan diversos operadores de refinamiento o generalización así como diferentes algoritmos de búsqueda. Ya que que GDEBaK se ejecuta antes de aprender una hipótesis entonces no depende del tipo de búsqueda o del operador de refinamiento utilizado. Esta característica le permite ser usado de manera global o local. En el primer caso, usa todo el conjunto de ejemplos para generar puntos de división (o subconjuntos de categorías), los cuales son usados durante todo el proceso de aprendizaje: discretización/agrupamiento global. En el segundo caso, GDEBaK es usado solamente con los ejemplos disponibles en cada iteración: método local. GDEBaK proporciona una variable Booleana que determina el tipo de proceso que se llevará a cabo.

Algoritmo 4.1 Algoritmo ILP Genérico

1: **procedure** ALGORITMOILPGENERICO($E^+, E^-, B \cup$ *declaracionAtributos*)
2: *Teoría* $= \phi$
3: **while** $E^+ \neq \phi$ **do**
4: $\mathbf{B} \leftarrow$ **GDEBaK(**$\mathbf{E^+}, \mathbf{E^-}, \mathbf{B} \cup$ **declaracionAtributos)**
5: *Hipótesis* $\leftarrow$ *aprende_hipótesis*(E^+, E^-, B)
6: *Teoría* $\leftarrow$ *Teoría* $\cup \{$*Hipótesis*$\}$
7: $B \leftarrow B \cup \{$*Hipótesis*$\}$
8: $E^+ \leftarrow E^+ \setminus \{$*Ejemplos cubiertos por Hipótesis*$\}$
9: **end while**
10: **return** *Teoría*
11: **end procedure**

En el Algoritmo 4.2 se presenta el método GDEBaK, el cual está basado en un es-

quema evolutivo. Los argumentos de entrada son los mismos del algoritmo ILP general. GDEBaK consta de un ciclo *for* principal que genera para cada argumento o atributo declarado en *declaracionAtributos* sus correspondientes puntos de división o conjuntos de valores categóricos de acuerdo a un algoritmo genético. Los principales elementos del algoritmo GDEBaK son presentados a continuación.

Algoritmo 4.2 Algoritmo GDEBaK

```
1:  procedure ALGORITMOGDEBAK(E+, E−, B ∪ declaracionAtributos) (§4.1.1.)
2:      for each Atributo en declaracionAtributos do
3:          crea Tabla a partir de E+, E−, B ∪ Atributo (§4.1.1.)
4:          t ← 0
5:          inicializa Poblacion(t) (§4.1.2.)
6:          evalúa Poblacion(t) (§4.1.2.)
7:          while no se cumpla condición de paro do
8:              t ← t + 1
9:              selecciona Poblacion(t) de Poblacion(t − 1) (§4.1.3.)
10:             cruza Poblacion(t) (§4.1.3.)
11:             muta Poblacion(t) (§4.1.3.)
12:             evalúa Poblacion(t)
13:         end while
14:         selecciona Mejores_Individuos de Poblacion(t) (§4.1.4.)
15:         agrega Mejores_Individuos a B (§4.1.4.)
16:     end for
17: end procedure
```

4.1.1. Declaración de atributos a procesar

Como se mencionó anteriormente, además de los ejemplos y el conocimiento previo que utiliza un sistema ILP, GDEBaK permite que el usuario pueda definir un conjunto de declaraciones, denotado como *declaracionAtributos*, para indicarle al sistema cuáles son los atributos que serán procesados. Si el atributo es numérico se utiliza el meta-predicado *dicretization/2*, y si el atributo es categórico entonces se usa el meta-predicado *grouping/2*.

Cada declaración en *declaracionAtributos* tiene el formato siguiente: *metapredicate(goal(E,C), lit(E,...,Attr))*, donde *metapredicate* $\in \{$*discretization, grouping*$\}$, donde *goal* es la relación objetivo, *C* es el argumento clase de la relación objetivo y solo aparece cuando el problema tiene más de dos clases, *Attr* es el atributo numérico (o categórico), y *E*

es el argumento de la relación objetivo que está ligado al correspondiente argumento en la relación *lit*.

Estas declaraciones le permiten a GDEBaK hacer las consultas necesarias, dentro del conocimiento previo y ejemplos, para obtener la información del atributo a procesar y crear una tabla de tres columnas: *id_ejemplo, valor_atributo, valor_clase*, la cual, será usada por el algoritmo genético de GDEBaK. Por ejemplo, si la relación objetivo es *calidad_pelicula/1* cuyo argumento es el identificador de una película, y *genero/2* es una relación en el conocimiento previo donde el primer argumento es el identificador de una película y el segundo es su correspondiente género, entonces la siguiente declaración indicará que el segundo argumento de *genero/2* deberá ser agrupado por GDEBaK.

```
grouping( calidad_pelicula (E), genero(E, Attr )).
```

Notemos en la declaración anterior que el argumento *E* permite relacionar el predicado objetivo con el predicado *genero/2*, el cual, contiene el argumento a procesar. De esta forma GDEBaK puede obtener la información que coincida en las dos relaciones y crear una tabla como la que se muestra en Tabla 4.1. Dicha tabla tiene tantos renglones como valores contiene el correspondiente atributo dentro del conocimiento previo.

ejemplo	**attr**	**clase**
e1	ciencia_ficcion	positivo
e2	romance	negativo
e3	accion	negativo
⋮	⋮	⋮

Tabla 4.1: Tabla creada por GDEBaK para el atributo *genero*.

4.1.2. Representación y evaluación de los cromosomas

Para crear, evaluar y evolucionar una población de soluciones con algoritmos genéticos, primero es necesario definir una representación adecuada para cada solución (cromosoma), así como una función de aptitud que permita asignarle un valor numérico a cada cromosoma. Para los atributos numéricos y categóricos, GDEBaK usa las siguientes representaciones y funciones de aptitud.

Cromosomas categóricos. Sea A un atributo categórico con valores en el conjunto $A_c = \{a_1, \ldots, a_m\}$, entonces un cromosoma categórico es el subconjunto $A_s \subseteq A_c$, el cual, es

representado por el vector binario $C_c = (c_1, \ldots, c_m)$ donde el gen $c_i = 1$ si el correspondiente valor $a_i \in A_s$ y $c_i = 0$ en otro caso. Así, si $A_c = \{accion, suspenso, romance, fantasia, comedia\}$ entonces el cromosoma $C_c = (0, 1, 0, 1, 0)$ representa al subconjunto $A_c = \{suspenso, fantasia\}$.

Para evaluar cada cromosoma, GDEBaK usa una medida de similitud basada en el contexto, llamada *DIstance Learning of Categorical Attributes (DILCA) [32]*, la cual se explica con detalle en la sección 2.4. Esta medida calcula la distancia entre cada par de atributos de un subconjunto de valores categóricos. A diferencia de otras medidas similares DILCA toma en cuenta la relación del atributo A y los otros atributos (contexto de A). El método DILCA se lleva a cabo en dos fases. Primero, se seleccionan los atributos categóricos mejor relacionados con A. A este conjunto se le llama *contexto(A)*. La correlación entre los atributos en *contexto(A)* y A se mide por medio de la *métrica de incertidumbre simétrica*, la cual, está basada en la entropía. En la segunda fase, DILCA usa el conjunto *contexto(A)* para calcular la distancia entre cada par de categorías $a_i, a_j \in A_c$ con la siguiente fórmula:

$$d(a_i, a_j) = \sqrt{\sum_{Y \in \textit{contexto(A)}} \sum_{y_k \in Y} \left(P(a_i \mid y_k) - P(a_j \mid y_k)\right)^2}$$

donde, para cada atributo Y, se calculan las siguientes probabilidades: $P(a_i \mid y_k)$ es la probabilidad condicional del valor a_i dado el valor $y_k \in Y$, y $P(a_j \mid y_k)$ es la probabilidad del valor a_j dado $y_k \in Y$. Recordemos que la fórmula para calcular la probabilidad condicional es $P(a_i \mid y_k) = P(a_i \cap y_k)/P(a_i)$.

Notemos que para cada atributo categórico se creará un conjunto de datos como el que se muestra en la Tabla 4.1. Por lo tanto, solo contamos con el atributo A y no es posible crear el conjunto *contexto(A)*, así que para adaptar DILCA a GDEBaK usamos el atributo clase Y (que sí está en la tabla mostrada anteriormente) como contexto, de esta manera tenemos que $Y = contexto(A)$. La idea es encontrar cromosomas con los valores categóricos más similares entre sí (más cercanos) que estén relacionados a una clase $y_k \in Y$. En ILP los valores de estos cromosomas estarán relacionados con los ejemplos positivos. Intuitivamente, podemos observar que si tenemos el conjunto de categorías A_c ={*drama, cienciaFiccion, aventura, familia, acción, comedia, horror, música, romance*}, entonces los valores en $C_{c1} = (1, 0, 0, 1, 0, 0, 0, 1, 1) \equiv$ {*drama, familia, música, romance*} son más similares entre sí que las categorías en $C_{c2} = (1, 0, 0, 0, 1, 0, 1, 0, 1) \equiv$ {*drama, acción, horror, romance*}.

Por lo tanto, la aptitud de un cromosoma categórico C_c está definido como la suma negativa de las distancias de cada par de categorías $a_i, a_j \in A_c$ tal que $c_i = c_j = 1$ en C_c más el valor $\frac{1}{N}$, donde N es el número de unos en C_c (Ecuación 4.1). Este último

valor permite que se beneficie el cromosoma que represente un conjunto de más valores categóricos cuando dos o más tengan la misma suma total de distancias. De esta forma serán usados subconjuntos con más categorías para crear teorías con el menor número posible de reglas. Por ejemplo, podemos ver que el subconjunto {*drama, familia, música, romance*} es una mejor generalización que {*música, romance*} si ambas tienen la misma suma de distancias. Observemos que la suma de distancias más el valor $1/N$ se multiplica por -1. Esto se realiza ya que al usar un algoritmo genético lo que se desea es maximizar la función objetivo, y si no se multiplica por -1 la función tendría que minimizarse, ya que lo que deseamos es que las distancias entre las categorías sean lo más pequeñas posible.

$$aptitud(C_c) = -\left[\sum_{a_i,a_j \in C_c} \sqrt{\sum_{y_k \in Y} \left(P(a_i \mid y_k) - P(a_j \mid y_k)\right)^2} + \frac{1}{N}\right] \tag{4.1}$$

Cromosomas numéricos. Sea A un atributo numérico con valores en $I_n = [r_{min}, r_{max}]$ donde $r_{min}, r_{max} \in \mathbb{R}$, entonces un cromosoma numérico C_n es una cadena binaria de longitud l, la cual, es equivalente a un número real r_n tal que $r_{min} < r_n < r_{max}$. Notemos que r_n divide al intervalo I en dos subintervalos: $I_a = [r_{min}, r_n)$ y $I_b = [r_n, r_{max}]$.

Para transformar un cromosoma binario a un número real, primero se convierte la cadena binaria C_n a un entero z_n. Después, z_n se transforma a un número real con la Ecuación 4.2.

$$r_n = \frac{r_{max} - r_{min}}{2^l - 1} z_n + r_{min} \tag{4.2}$$

donde r_{min} y r_{max} representan los valores mínimo y máximo en el intervalo I, y l es el número de *bits* en la el cromosoma C_n. Para GDEBaK se utilizó un valor de $l = 25$, ya que con este número de bits podemos representar números reales con una precisión de 8 decimales. Tomando en cuenta los valores numéricos que contienen los conjuntos de datos, usados en los experimentos, dicha precisión es suficiente.

Para evaluar cada cromosoma, GDEBaK usa una función basada en la entropía. El uso de esta función en un método evolutivo permite que se escape de los óptimos locales y por lo tanto la discretización es más eficiente que con métodos más simples [17]. El objetivo es encontrar un cromosoma C_{ni} cuyo equivalente número real r_{ni} minimice la entropía en los subintervalos I_a, I_b. Una baja entropía implica una alta ganancia de información y una reducción en la incertidumbre de clasificación. Así, la aptitud de un cromosoma C_n está dada por la ganancia de información del número real r_n dado por la Ecuación 4.3.

$$IG(r_n) = \textit{Entropía}(I_n) - \left[\frac{\textit{count}(I_a)}{\textit{count}(I_n)} * \textit{Entropía}(I_a) + \frac{\textit{count}(I_b)}{\textit{count}(I_n)} * \textit{Entropía}(I_b)\right] \quad (4.3)$$

Además, la entropía en el intervalo I_x está dada por

$$\textit{Entropía}(I_x) = -\sum_{j \in clases} p(I_x, j) * log_2(p(I_x, j)) \quad (4.4)$$

donde $count(I_x)$ es el número de valores en un intervalo I_x, y $p(I_x, j)$ es la proporción de valores en I_x que pertenecen a la clase j. Por lo tanto, GDEBaK busca puntos de división con la mayor ganancia de información posible.

4.1.3. Operadores genéticos

GDEBaK utiliza los tres operadores genéticos usuales para evolucionar la población inicial en cada iteración del algoritmo genético, estos son: selección, mutación y cruzamiento, estos son detallados a continuación para cada tipo de cromosoma.

Selección Para los dos tipos de cromosomas (numéricos y categóricos) utilizamos el esquema de truncación. Este consiste en seleccionar un porcentaje de los mejores cromosomas, de acuerdo a la función de aptitud. Por ejemplo, si $T = 25\,\%$ entonces el 25 por ciento de los mejores cromosomas son seleccionados como padres, y el resto no producirá descendientes. Se probaron los siguientes valores: $T = \{10\,\%, 20\,\%, 30\,\%40, 50\,\%\}$ y observamos que para valores más pequeños que $40\,\%$ la convergencia es más lenta mientras que no hubo diferencia entre $40\,\%$ y $50\,\%$. Así que se decidió utilizar $40\,\%$ como parámetro para los dos tipos de cromosomas.

Cruzamiento. Tanto los cromosomas categóricos como numéricos se representan como una cadena binaria, por lo que se decidió utilizar operadores de cruzamiento de uno y dos puntos. En el primero los dos cromosomas seleccionados (padres) se cortan en un punto seleccionado aleatoriamente. Este punto es llamado *punto de pivote o de corte*. Después, se intercambian, entre los dos cromosomas, los genes que están a la derecha de ese punto de corte. Este proceso genera otros dos cromosomas (hijos), ver Figura 4.1.

Por otro lado, en el cruzamiento de dos puntos se generan aleatoriamente dos puntos de corte, con los cuales, se crean tres segmentos por cada cromosoma padre. Después, se intercambian alternadamente los segmentos en los cromosomas padre, ver Figura 4.2.

En cuanto a la probabilidad de cruzamiento se probaron diferentes valores, sin embargo no se observaron diferencias en cuando a la precisión de las teorías finales, por lo que se decidió usar una probabilidad estándar de $p_c = 0.6$.

Figura 4.1: Cruzamiento de un punto para cromosomas binarios.

Figura 4.2: Cruzamiento de dos puntos para cromosomas binarios.

Mutación. Al igual que en el cruzamiento se utilizó el mismo operador de mutación para cromosomas numéricos y categóricos. Este operador cambia el valor de cada uno de los genes de acuerdo a una probabilidad de mutación. Se probaron diferentes valores para la probabilidad de mutación sin encontrar una diferencia en la precisión de las teorías finales por lo que se eligió el valor $p_m = 0.01$. Al mutar un cromosoma padre se genera un sólo hijo como se muestra en la Figura 4.3.

Figura 4.3: Mutación binaria. Los genes del cromosoma seleccionado cambian su valor aleatoriamente en base a una probabilidad de mutación.

4.1.4. Enriquecimiento del conocimiento previo

Cuando el algoritmo genético de GDEBaK termina su ejecución, los mejores N cromosomas son agregados al conocimiento previo. Si la población tiene al menos 50 el valor por defecto es del $5\,\%$, cuando la población es más pequeña se agrega sólo un cromosoma, aunque usuario puede modificar dicho valor. Además de agregar los mejores cromosomas al conocimiento previo, también se agregan otras relaciones que le permiten al sistema

ILP utilizar la información dada en los cromosomas, y dependiendo del tipo de atributo se agregan los siguientes:

Cromosomas categóricos. Recordemos que cada cromosoma categórico representado por una cadena binaria representa a su vez un subconjunto de datos de un atributo categórico B. Por lo tanto, no se agregarán las cadenas binarias sino los conjuntos a los que representan. Para que el conocimiento previo pueda utilizar estos subconjuntos en el proceso de aprendizaje también se añade la relación de membresía *member/2*, la cual es verdadera si un elemento pertenece a un conjunto de datos, y falso en caso contrario. La manera en que esta información, que enriquece al conocimiento previo, es usada en las reglas candidatas es la siguiente.

Sea B un argumento categórico, $C_{ci} = (c_1, \ldots, c_m)$ un cromosoma binario inducido por GDEBaK, y $B_{ci} = \{b_1, \ldots, b_m\}$ el conjunto de valores categóricos representados en C_{ci}, donde cada valor $b_j \in B$, entonces una regla candidata puede tener la siguiente forma:

```
goal (...) :- ..., lit (..., B), member(B, [c1, ..., cm])
```

donde *goal* es la relación objetivo, y *lit* una relación en el conocimiento previo con el atributo categórico B. La relación *member/2* se agregará cuando se procese al menos un argumento categórico. Por ejemplo, la siguiente regla podría generarse si el género de una película es usada para indicar si una película tiene calidad o no.

```
calidad_pelicula (A):-enero(A,B), member(B, [ ciencia_ficcion , accion , suspenso ]),
```

Cromosomas numéricos. Al igual que en el caso de los cromosomas categóricos, los numéricos también son cadenas binarias, sin embargo, lo que enriquecerá al conocimiento previo serán los correspondientes valores reales que representan. Para que el sistema pueda usar esta nueva información se agregan a su vez las dos relaciones *lteq/2* y *gteq/2* que corresponden a $\leq$ y $\geq$ respectivamente. La forma en que el sistema ILP utilizará esta información será la siguiente.

Sea A un atributo numérico en $I = [min, max]$, C_{ni} un cromosoma binario inducido por GDEBaK, y $min \leq x \leq max$ el valor real representado por C_{ni}, entonces una regla candidata podrá tener las siguientes estructuras:

```
goal (...) :- ..., lit (..., A), lteq (A,x)
goal (...) :- ..., lit (..., A), gteq (A,x)
```

donde *goal* es la relación objetivo, y *lit* es una relación en el conocimiento previo con un argumento (atributo) numérico A. Las relaciones *lteq/2* y *gteq/2* se agregarán al conocimiento previo si al menos un atributo es procesado.

Es posible también agrupar y discretizar en la creación de una teoría. Por ejemplo, si GDEBaK agrupa el segundo argumento de la relación *genero/2*, que tiene como argumentos el identificador de una película y su correspondiente género, y discretiza el segundo argumento de la relación *presupuesto/2*, que tiene como argumentos el identificador de una película y su correspondiente presupuesto, entonces es posible que el sistema genere una regla, dentro de la teoría final, como la siguiente:

```
calidad_pelicula (A):-
  genero(A,B), member(B, [ ciencia_ficcion , accion , suspenso ]),
  presupuesto(A,C), gteq(C, 100).
```

donde el valor 100 está en millones de dólares.

4.1.5. Ejemplo en Aleph

Como se mencionó anteriormente, el algoritmo GDEBaK puede implementarse en cualquier sistema ILP. Para este trabajo se implementó en el sistema Aleph [55] ya que es un sistema de código abierto *(open source)*, y su código está disponible. Además, es uno de los sistemas más utilizados en gran parte de lo trabajos relacionados al área de ILP. Por otro lado, Aleph proporciona un lenguaje declarativo que nos permitió agregar, el código necesario para que la declaración de los atributos a procesar sea más fácil. En el Algoritmo 4.3 se muestra el algoritmo básico de Aleph con el método GDEBaK incrustado. Aleph es un sistema de tipo *bottom-up* por lo que para generar la siguiente regla primero crea una cláusula e muy específica, con muchas literales en el cuerpo, llamada *bottom clause*. Después, busca una regla más general que e.

Algoritmo 4.3 Algoritmo Básico de Aleph con GDEBaK

1: **procedure** ALEPH($E^+, E^-, B \cup$ *declaracionAtributos*)
2: Selecciona un ejemplo e
3: **GDEBaK($\mathbf{E}^+, \mathbf{E}^-, \mathbf{B} \cup$ declaracionAtributos)**
4: Construir la cláusula más específica
5: Buscar siguiente hipótesis
6: Remover ejemplos redundantes. Ir al paso 2.
7: **end procedure**

El siguiente ejemplo muestra el método GDEBaK para el agrupamiento de un atributo

categórico en Aleph. El problema de aprendizaje utilizado fue tomado del conjunto de datos *IMDB's Top 250 [33]* que consiste en inducir una teoría que describa las características de lo que se considera una buena película. La relación objetivo es

```
calidad_pelicula ( id_pelicula ) :-
```

donde un ejemplo positivo indica que la película es buena, y un ejemplo negativo indica que NO es una película buena. El conocimiento previo contiene las relaciones siguientes:

```
genero_director ( id_director , genero, _, id_pelicula )
genero_pelicula ( id_pelicula , genero, _)
rol_actor ( id_actor , id_pelicula , _)
actor ( id_actor , genero)
director ( id_director )
```

Los argumentos en guión bajo los omitimos porque no son relevantes para el ejemplo. El atributo categórico que se desea agrupar es el tercer argumento de la relación *genero_director* que corresponde al género en el que se encasilla a un director. Los principales elementos de GDEBaK se presentan a continuación:

Declaración de atributos a procesar. El sistema Aleph requiere de tres archivos: *file.b* contiene el conocimiento previo y los datos de configuración del algoritmo, *file.f* contiene los ejemplos positivos, y *file.n* contiene los ejemplos negativos. Ya que el archivo de conocimiento previo también contiene las declaraciones de configuración, es ahí donde se indica cuáles son los atributos que se van a procesar por medio de GDEBaK. Para el ejemplo que se presenta la siguiente declaración indica que se desea agrupar el tercer atributo de la relación *genero_director*:

```
:- grouping( calidad_pelicula (e), genero_director (e, no, attr , no )).
```

La declaración anterior está dada en formato de Aleph. Los argumentos con valor *no* indican que no serán procesados. El argumento *e* es el que liga a las dos relaciones en la declaración, como se mencionó en la Sección 4.1.1, ese argumento permite a crear la tabla con la que se agrupará el atributo correspondiente. Una muestra del conjunto de datos para este ejemplo se muestra en la Tabla 4.2 muestra algunos ejemplos de estos datos que se requieren.

Representación de cromosomas y aptitud. En este ejemplo, GDEBaK crea cromosomas con 6 bits o genes ya que tenemos las categorías {*acción, suspenso, fantasía, musical,*

e (example)	attr (attribute)	clase
2136	acción	negativo
10830	terror	positivo
11440	guerra	positivo
54726	comedia	negativo
31615	suspenso	positivo
⋮	⋮	⋮

Tabla 4.2: Esta tabla muestra algunos valores del atributo categórico que corresponden a los géneros de cada director en el conjunto de datos.

comedia, romance$\}$. Cabe añadir que en el conjunto de datos IMDB original hay 18 géneros, pero por simplicidad presentamos solo 6. La Tabla 4.3 muestra algunos cromosomas con su valor de aptitud calculado con la ecuación 4.1.

No.	Cromosoma binario	Valores categóricos	Aptitud
1	$(1, 1, 1, 0, 0, 0)$	$\{action, thriller, fantasy\}$	-26.933
2	$(1, 0, 0, 0, 1, 1)$	$\{action, comedy, romance\}$	-124.03

Tabla 4.3: Ejemplos de cromosomas categóricos para el atributo *género* con su correspondiente valor de aptitud.

Operadores genéticos. Como hemos observado cada cromosoma binario tiene la misma longitud sin embargo el número de categorías que representan puede variar. A continuación, se muestran unos ejemplos de cómo se llevan a cabo las operaciones de cruzamiento y mutación para este tipo de cromosomas.

El operador de mutación cambia el valor de los genes. Si un gen cambia su valor de uno a cero, entonces se eliminará un valor del cromosoma, y si cambia de cero a uno se añadirá el valor correspondiente al subconjunto representado por ese cromosoma. El ejemplo siguiente muestra precisamente este último caso.

$(1, 1, 1, 0, 0, 0) \equiv$ {action, thriller, fantasy}
$\downarrow$
$(1, 1, 1, 1, 0, 0) \equiv$ {action, thriller, fantasy, musical}

Al aplicar el operador de cruzamiento el tamaño de los subconjuntos que representa cada cromosoma también puede cambiar. En el ejemplo siguiente uno de ellos quedó con solo dos categorías mientras que el otro con cuatro.

$(1,1,\mid 1,0,0,0) \equiv$ {action, thriller, fantasy}
$(1,0,\mid 0,0,1,1) \equiv$ {action, comedy, romance}
$\Downarrow$
$(1,0 \mid 1,0,0,0) \equiv$ {action, fantasy}
$(1,1,\mid 0,0,1,1) \equiv$ {action, thriller, comedy, romance}

Enriquecimiento del conocimiento previo El último paso de GDEBaK es enriquecer el conocimiento previo con los mejores cromosomas encontrados. Recordemos que el usuario debe indicar el porcentaje de cromosomas que se agregarán al conocimiento previo. Con esta nueva información Aleph genera la siguiente regla que será añadida a la teoría final. A continuación tenemos un ejemplo de una regla que es creada por Aleph con la información proporcionada por GDEBaK.

```
movie_quality(A) :-
   director_genre(A,B,C,D), member(C,[action, thriller , romance]).
```

Un ejemplo más completo es presentado en el Apéndice A, el cual, describe el proceso GDEBaK con más detalle para el conjunto de datos mutagenesis.

En el siguiente capítulo se describen los experimentos realizados con GDEBaK, discretización perezosa en Aleph, y la discretización en TILDE. También se presentan los resultados obtenidos con dichos experimentos, con los cuales, comparamos la propuesta presentada con la discretización en Aleph y TILDE.

Capítulo 5

Experimentos y resultados

> "Cuando todo parezca estar en tu contra, recuerda que los aviones despegan con el aire en contra, no a favor."
>
> Henry Ford

En este capítulo presentamos los experimentos realizados para evaluar el método GDEBaK. Para ello realizamos varios experimentos con diferentes conjuntos de datos usando como medida de evaluación la precisión y el número de reglas por teoría. La precisión de una teoría se calcula con la fórmula $\frac{p}{p+n}$, donde p es es el número de ejemplos positivos cubiertos (verdaderos positivos), y n es el número de ejemplos negativos cubiertos, estos últimos corresponden a los falsos positivos. Ya que GDEBaK puede procesar atributos numéricos y categóricos, los experimentos fueron llevados a cabo de la siguiente manera:

1. Primero, usamos GDEBaK para procesar únicamente los atributos numéricos, y los resultados obtenidos fueron comparados con la discretización perezosa de Aleph *(lazy discretization)*, con la discretización voraz *(greedy)* en Aleph[1], y la discretización del sistema TILDE.

2. Después, se ejecutó GDEBaK sobre conjuntos de datos con atributos categóricos para agruparlos. En este caso, comparamos los resultados obtenidos con agrupamiento

[1]Para comparar GDEBaK con un método no evolutivo, implementamos en Aleph un algoritmo voraz que busca el punto de división. En este se elije la solución más prometedora, sin embargo al llegar a un óptimo local ya no puede salir de él.

voraz en Aleph[2], así como con Aleph y TILDE sin agrupar ningún atributo, ya que estos sistemas no cuentan con algún método para agrupar valores categóricos.

3. Otro conjunto de experimentos se consideró procesando con GDEBaK atributos numéricos y categóricos con el mismo conjunto de datos, y los resultados fueron comparados con Aleph y TILDE.

4. Finalmente, ejecutamos GDEBaK sobre algunos conjuntos de datos para comparar la discretización/agrupamiento local y global.

El objetivo de estos experimentos es demostrar que el método propuesto puede generar puntos de división y subconjuntos de valores categóricos que mejoran la precisión de las teorías finales y/o disminuir el número de reglas de cada teoría.

5.1. Materiales

Para los experimentos se utilizaron los siguientes conjuntos de datos, los cuales, contienen atributos numéricos y/o categóricos.

- Del *UCI Machine Learning Repository [23]*, usamos algunos de los conjuntos de datos más conocidos: *Australian Credit Approval, Pittsburgh Bridges, Japanese Credit Screening, German Credit, Iris Dataset, Ecoli y Student Loan.*

- Otros conjuntos de datos que son muy utilizados en ILP: Carcinogenesis,[57] KRK illegal chess position (KRKi),[10] Mutagenesis data,[59] MovieLens,[28] IMDB Top 250 movies,[33] Metabolism,[11], y Hepatitis.[24]

Los sistemas usados para los experimentos son Aleph, implementado en Yap Prolog versión 6.2.2.,[13] y TILDE que está incluido en el sistema de minería de datos ACE.[5]. La comparación con otros sistemas ILP como ECL, SMART+ o SIA-01 no fue posible ya que no están disponibles. Todos los experimentos fueron llevados a cabo en una computadora multi-núcleo.

[2]Se implementó el agrupamiento voraz en Aleph. Este algoritmo genera y prueba nuevos subconjuntos de valores categóricos únicamente cuando mejoren al actual.

5.2. Metodología

Al dividir los experimentos en cuatro categorías (solo discretización, solo agrupamiento, discretización más agrupamiento, y discretización más agrupamiento tanto local como global) podemos evaluar de mejor manera el rendimiento de GDEBaK. A continuación detallamos la manera en que se llevaron a cabo estos experimentos.

Discretización. En este caso, cada conjunto de datos fue procesado tomando en cuenta únicamente los atributos numéricos con: método voraz en Aleph, GDEBaK en Aleph, discretización perezosa en Aleph, y discretización en TILDE. La Tabla 5.1 muestra las características de los conjuntos de datos procesados: número de ejemplos, número de clases, número de argumentos discretizados, tamaño de la población, número de generaciones, y el número de cromosomas que fueron añadidos al conocimiento previo. Los últimos tres parámetros son exclusivos de GDEBaK.

Conjunto de datos	**Número de ejemplos**	**Número de clases**	**Argumentos discretizados**	**Tamaño de la población †**	**Número de generaciones †**	**Cromosomas agregados †**
Australian credit	690	2	3	50	30	5
Pittsburgh bridges	108	6	2	50	30	10
Japanese credit	125	2	5	50	30	10
Ecoli	336	8	5	50	30	10
Student loan	1000	2	2	30	20	5
Carcinogenesis	298	2	3	30	40	10
Iris	150	3	4	30	50	10
German credit	1000	2	4	40	50	5
Mutagenesis	188	2	2	30	50	5
IMDB Top 250	166	2	1	30	50	5

Tabla 5.1: Esta tabla muestra las características de los conjuntos de datos con atributos numéricos. Los parámetros marcados con † indican que para cada conjunto de datos el valor mostrado fue aquel con el que se obtuvo mejor precisión.

Agrupamiento. Para este grupo de experimentos, los conjuntos de datos utilizados fueron aquellos con atributos categóricos. Debido a que no siempre se tienen atributos categóricos y numéricos, los conjuntos de datos utilizados en cada grupo de experimentos es diferente. Para el agrupamiento se compararon los resultados de GDEBaK con los obtenidos en agrupamiento voraz en Aleph, Aleph sin agrupamiento, y TILDE sin agrupamiento. La tabla 5.2 muestra los datos más importantes de cada conjunto de datos.

Agrupamiento y discretización. En este grupo de experimentos se tomaron en cuenta tanto los atributos o argumentos numéricos como los categóricos de manera simultánea. El

Conjunto de datos	Número de ejemplos	Número de clases	Argumentos agrupados	Tamaño de la población †	Número de generaciones †	Cromosomas agregados †
Pittsburgh bridges	108	6	6	20	30	5
MovieLens	100,000	2	2	20	30	5
Student Loan	1000	2	2	20	30	5
KRKi	1000	2	3	15	20	5
German Credit	1000	2	11	15	30	5
Hepatitis	500	2	3	50	25	5
IMDB Top 250	166	2	2	50	100	5
Metabolism	230	2	2	100	100	5

Tabla 5.2: Esta tabla muestra los parámetros más importantes de los conjuntos de datos cuyos atributos categóricos fueron agrupados. El símbolo † indica que los valores usados en esos parámetros fueron aquellos con los que se obtuvo mejor precisión.

número de conjuntos de datos utilizados fue menor que en los casos anteriores ya que no todos cuentan con atributos numéricos y categóricos. Los resultados de GDEBaK fueron comparados con los obtenidos con discretización perezosa en Aleph, y TILDE. Debemos aclarar que GDEBaK es el único de estos sistemas en poder agrupar atributos categóricos, por lo que en el caso de Aleph y TILDE solo se discretizaron los atributos numéricos. La tabla 5.3 muestra los principales parámetros de cada conjunto de datos utilizados en este grupo de experimentos.

Conjunto de datos	Número de ejemplos	Número de clases	Tamaño de la población †	Número de generaciones †	Cromosomas agregados †
Pittsburgh bridges	108	6	20	30	5
Student Loan	1000	2	20	30	5
IMDB Top 250	166	2	50	100	5
Mutagenesis	188	2	50	30	5

Tabla 5.3: Esta tabla muestra los parámetros de los conjuntos de datos que contienen parámetros numéricos y categóricos. El símbolo El símbolo † nos indica que los valores usados en esos parámetros fueron aquellos con los que se obtuvo una mejor precisión.

Local vs Global Ya que GDEBaK puede usarse de manera global y local, comparamos los resultados obtenidos al procesar los diferentes conjuntos de datos con estas dos opciones. La Tabla 5.4 muestra los valores de los parámetros más importantes. Es necesario aclarar que los anteriores experimentos los atributos fueron procesados de manera local, ya que es la opción por defecto de GDEBaK.

Para calcular la precisión y el número de reglas por teoría llevamos a cabo, en todos los experimentos planteados anteriormente, una validación cruzada de 10 iteraciones. To-

Conjunto de datos	**Número ejemplos**	**Número de clases**	**Tamaño de la población †**	**Número de generaciones †**	**Cromosomas agregados †**
Australian credit	690	2	50	20	10
Pittsburgh bridges	108	6	50	30	10
Japanese credit	125	2	25	30	10
Ecoli	336	8	50	50	10
German credit	1000	2	30	50	10
Metabolism	230	2	50	100	10
Mutagenesis	188	2	20	50	10
Student Loan	1000	2	20	50	10

Tabla 5.4: Esta tabla muestra los principales parámetros para conjuntos de datos con atributos categóricos y numéricos. Cada conjunto de datos fue procesado de manera local y global. El símbolo † nos indica que los valores usados en esos parámetros fueron aquellos con los que se obtuvo una mejor precisión.

mando en cuenta los diferentes parámetros de Aleph y TILDE ajustamos los siguientes valores para evitar que un sistema tuviera una ventaja sobre el otro. El primero de ellos es *minacc* (en Aleph) y *accuracy* (en TILDE). Este fija un valor mínimo para la precisión que debe tener una regla aceptable en el caso de Aleph o de un nodo en el caso de TILDE (tomemos en cuenta que TILDE genera un árbol de decisión lógico que es equivalente a una teoría expresada en cláusulas o reglas [6]).

El otro parámetro es *minpos* y *minimal_cases(2)* de Aleph y TILDE respectivamente. El valor de este parámetro indica el número mínimo de ejemplos positivos que debe cubrir una regla en Aleph o un nodo en TILDE, ver Figura 5.1.

Aleph	TILDE
set (minacc, 0.80).	accuracy (0.80).
set (minpos, 2).	minimal_cases (2).

Figura 5.1: Parámetros principales de Aleph y TILDE.

5.3. Resultados

En esta sección se presentan los resultados de los cuatros grupos de experimentos que se llevaron a cabo para evaluar GDEBaK. Se hace énfasis en que esta evaluación está basada en la calidad de las teorías: precisión y número de reglas.

Discretización Los resultados obtenidos al procesar únicamente argumentos numéricos se muestran en la Tabla 5.5. En esta se puede ver que GDEBaK mejora la precisión significativamente en siete conjuntos de datos. Se marcan con † los conjuntos de datos en los que GDEBaK mejoró considerablemente la precisión respecto a discretización perezosa en Aleph, y con ‡ cuando la mejora en precisión es respecto a TILDE. Estas diferencias son significativas de acuerdo a la prueba *t pareada* con un nivel de significancia de 0.05 sobre las validaciones cruzadas de 10 iteraciones. En estos resultados también podemos ver que en algunos casos, GDEBaK tiene casi la misma precisión que TILDE: en los conjuntos de datos *Australian, German credit* y *Mutagenesis*. Con *Carcinogenesis*, TILDE mejoró la precisión del método propuesto, sin embargo GDEBAK mejora considerablemente el número de reglas. La decisión de tomar el número de reglas por teoría como una medida de evaluación se debe a que una teoría con menos reglas es más fácil de interpretar. También, podemos observar que GDEBaK es mejor que la evaluación perezosa en Aleph en todos los casos, aunque no disminuye el número de reglas. Esto se debe principalmente a que ambos métodos encuentran diferentes puntos de división.

Conjunto de datos	**GDEBaK**		**Voraz**		**Aleph perezoso**		**TILDE**	
	Prec. (%)	**No. Reglas**	**Prec. (%)**	**No. Reglas**	**Prec. (%)**	**No. Reglas**	**Prec. (%)**	**No. Rules**
Australian †	**70.28**	4.5	56.77	**1.0**	61.31	6.4	69.7	7.8
Pittsburgh bridges †‡	**98.3**	7.5	96.0	8.58	88.62	**6.0**	52.77	8.4
Japanese credit †‡	**74.42**	3.0	52.1	**1.2**	54.8	3.7	66.4	5.9
Ecoli ‡	97.1	9.7	**98.8**	8.0	94.4	**7.7**	87.0	11.8
Student loan †‡	**90.6**	18.2	77.7	10.9	82.5	34.1	80.2	**6.0**
Carcinogenesis	56.6	**13.5**	53.6	14.0	51.9	19.2	**63.0**	34.3
Iris ‡	99.3	5.4	**100.0**	3.5	97.3	3.0	93.1	**3.0**
German credit	**72.2**	53.6	71.2	**51.8**	71.1	57.1	71.4	69.8
Mutagenesis †	**87.7**	4.4	66.4	4.4	76.7	5.0	84.1	**3.7**
IMDB Top 250	**84.53**	84.4	69.88	7.9	80.7	13.1	80.1	**5.5**
Mean	**83.1**	20.42	74.2	**11.1**	75.9	15.5	74.7	15.62

Tabla 5.5: Resultados obtenidos con GDEBaK, algoritmo voraz en Alpeh, discretización perezosa en Aleph, y TILDE. Los mejores valores se muestran en *negrita*. El símbolo † indica que la precisión de GDEBaK es significativamente mejor que la discretización perezosa en Aleph, y el símbolo ‡ indica que GDEBaK fue significativamente mejor en precisión que TILDE.

En la Figura 5.2 resumimos los resultados obtenidos con estos cuatro métodos: GDEBaK, algoritmo voraz en Aleph, discretización perezosa en Aleph y discretización en TILDE. En dicha imagen podemos ver que GDEBaK tiene mejor precisión en la mayoría de los casos con excepción de *Carcinogenesis*, ya que en este caso TILDE tiene mejor precisión aunque como vimos anteriormente el método propuesto disminuye considerablemente el número de reglas.

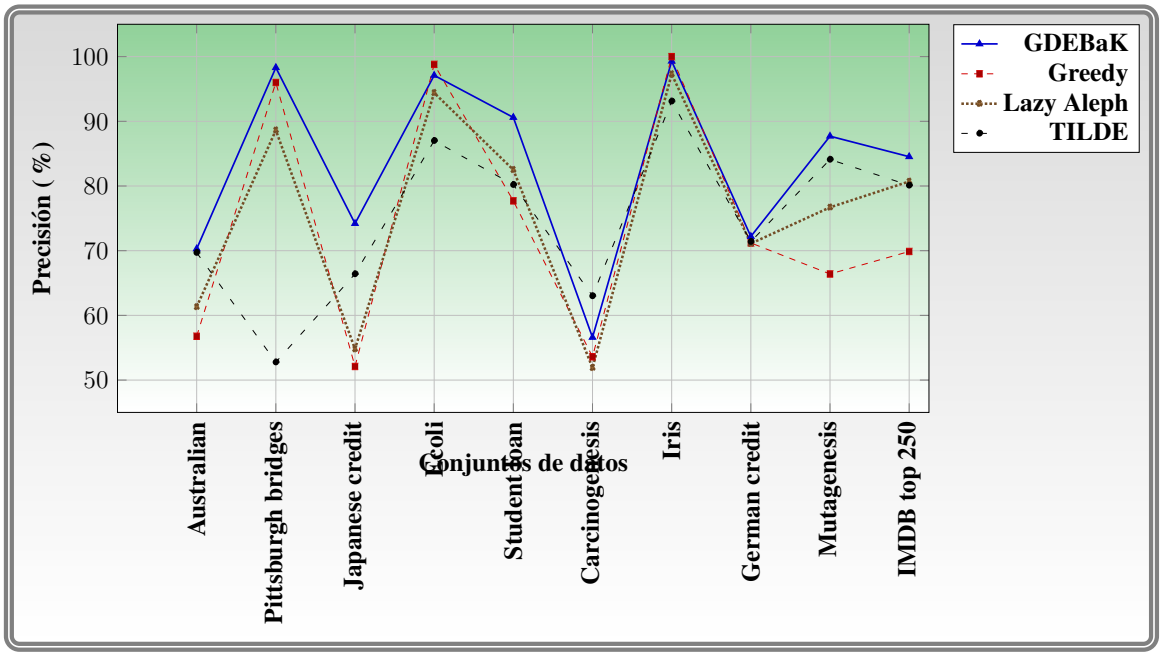

Figura 5.2: Esta gráfica muestra las precisiones obtenidas al discretizar los atributos numéricos para cada conjunto de datos. Se puede apreciar que GDEBaK es mejor que discretización perezosa de Aleph y que TILDE en la mayoría de los casos.

Agrupamiento La tabla 5.6 presenta las precisiones y el número de regla obtenidos al agrupar los atributos categóricos en los diferentes conjuntos de datos. Como podemos observar, GDEBaK tiene una mejor precisión con cinco conjuntos de datos: student loan, german credit, hepatitis, metabolism e IMDB Top 250. Por otro lado, en el caso de MovieLens y KRKi, GEBaK no mejora la precisión (en ambos casos el mejor resultado está dado con TILDE), sin embargo, la propuesta presentada disminuye considerablemente el número de reglas. Para MovieLens, GDEBaK tiene en promedio 7.5 reglas por teoría contra 37.8 del algoritmo Voraz, 108.6 para discretización perezosa y 22.1 de TILDE. Con KRKi, GDEBaK tiene en promedio 72.6 reglas por teoría, pero el algoritmo Voraz pro-

media 81.3, mientras que el algoritmo perezoso 89.8. Con este mismo conjunto de datos TILDE incluso casi duplica el número de reglas al promediar 143.9. Creemos que esta diferencia entre TILDE y Aleph se debe a que no son estrictamente análogos: el algoritmo de Aleph es bottom-up mientras que el de TILDE es top-down, y por otro lado, las opciones de configuración también son diferentes. La gráfica 5.3 resume los resultados obtenidos con agrupamiento.

Conjunto de datos	**GDEBaK**		**Voraz**		**Aleph perezoso**		**TILDE**	
	Prec. (%)	**No. Reglas**	**Prec. (%)**	**No. Reglas**	**Prec. (%)**	**No. Reglas**	**Prec. (%)**	**No. Rules**
Pittsburgh bridges ‡	95.3	**7.7**	**95.4**	7.8	90.21	8.5	64.8	11.7
MovieLens	53.0	**7.5**	47.96	37.8	57.74	108.6	**56.1**	22.1
Student loan ‡	**95.29**	41.0	91.7	48.6	91.60	46.6	79.0	**10.1**
KRKi	49.7	**72.6**	40.1	81.3	49.0	89.8	**50.0**	134.9
German credit	**72.0**	**37.8**	70.78	48.1	**72.0**	51.4	70.2	66.5
Hepatitis	**83.0**	**3.8**	82.6	13.5	81.7	14.5	81.8	9.9
Metabolism	**64.4**	**11.5**	58.63	12.9	60.3	13.5	63.3	21.8
IMDB top 250 †‡	**83.7**	3.9	54.8	**2.25**	68.1	6.4	77.1	3.6
Mean	**74.5**	**23.22**	67.64	31.53	71.3	42.41	67.78	35

Tabla 5.6: Esta tabla muestra la precisión y el número de reglas obtenidos al aplicar agrupamiento a los atributos categóricos de cada conjunto de datos. Los datos presentados corresponden a GDEBAK, algoritmo voraz con Aleph, discretización perezosa con Aleph y discretización en TILDE. El símbolo † indica que la precisión de GDEBaK es significativamente mejor que TILDE.

Agrupamiento y discretización El tercer grupo de experimentos fue realizado con el objetivo de analizar el agrupamiento y la discretización en un mismo conjunto de datos. Ya que no todos los conjuntos de datos tienen tanto atributos categóricos como numéricos solo lo aplicamos en cuatro de ellos. En tres de ellos GDEBaK tiene una clara mejoría respecto a la discretización perezosa de Aleph y respecto a TILDE. Por otro lado, GDEBaK disminuye el número de reglas en los cuatro casos, ver Tabla 5.7.

En la Figura 5.4 se puede apreciar con mayor claridad la diferencia de precisión entre GDEBaK y la discretización con Aleph y TILDE.

Local vs Global El cuarto grupo de experimentos realizados nos permitieron comparar el proceso local con el global que como se mencionó anteriormente el proceso local se lleva a cabo sólo con el conjunto de datos que se tiene al momento de aprender una nueva regla mientras que el global utiliza todo el conjunto de datos para generar los puntos de división (o los subconjuntos de categorías) y solo es llevado una sola vez antes de iniciar

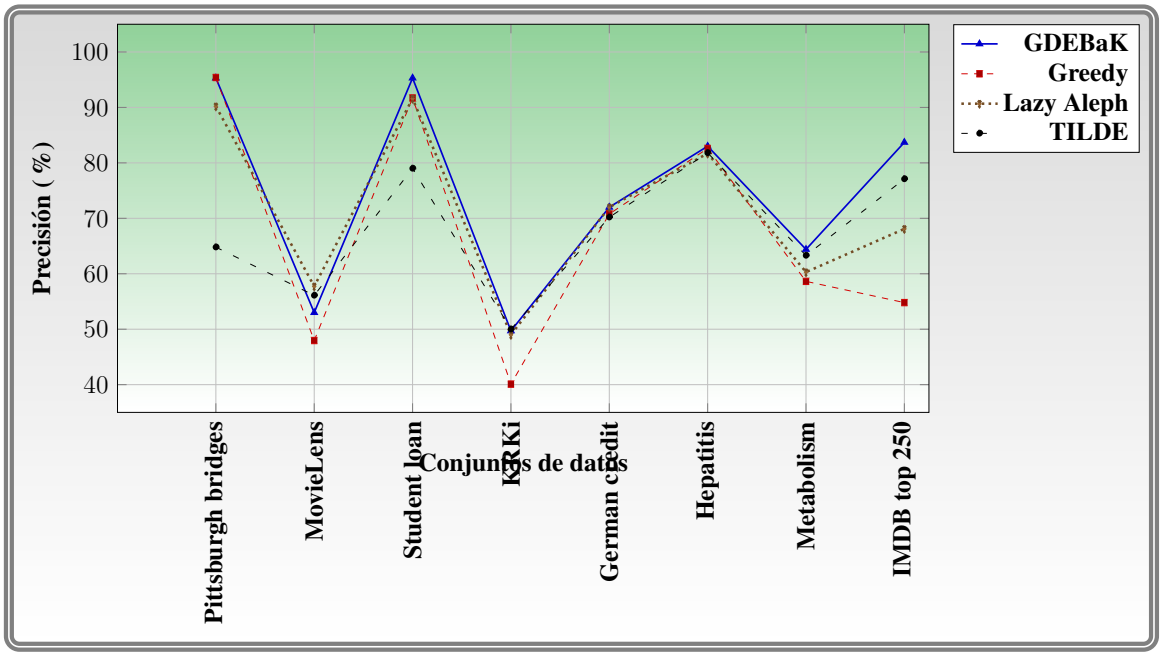

Figura 5.3: Esta gráfica muestra las precisiones obtenidas al agrupar los atributos categóricos para cada conjunto de datos.

Conjunto de datos	GDEBaK		Lazy Aleph		TILDE	
	Precisión (%)	No. Reglas	Precisión (%)	No. Reglas	Precisión (%)	No. Reglas
Pittsburgh bridges	**92.3**	**7.9**	90.21	8.5	64.8	11.7
Student Loan ‡	**97.1**	**8.8**	91.6	46.6	79.0	10.1
IMDB Top 250	**83.6**	**2.3**	68.1	6.4	77.1	3.6
Mutagenesis	**98.0**	**1.8**	97.2	2.7	84.1	3.7

Tabla 5.7: Esta tabla muestra la precisión y el número de reglas para cada conjunto de datos. Estos fueron obtenidos al aplicar discretización y agrupamiento con GDEBaK, y sólo discretización con Aleph y TILDE. El símbolo ‡ indica que la precisión obtenida con GDEBaK es significativamente mejor que TILDE.

la creación de la teoría. La tabla 5.8 muestra los resultados obtenidos y como se puede ver en algunos conjuntos de datos se agruparon los atributos categóricos y en otros se discretizaron los numéricos. En la mayoría los resultados son muy similares, sin embargo, el agrupamiento local mejora la precisión en dos casos, aunque con un incremento en el número de reglas. Por otro lado, la discretización global mejoró la precisión en tres casos, aumentando también el número de reglas.

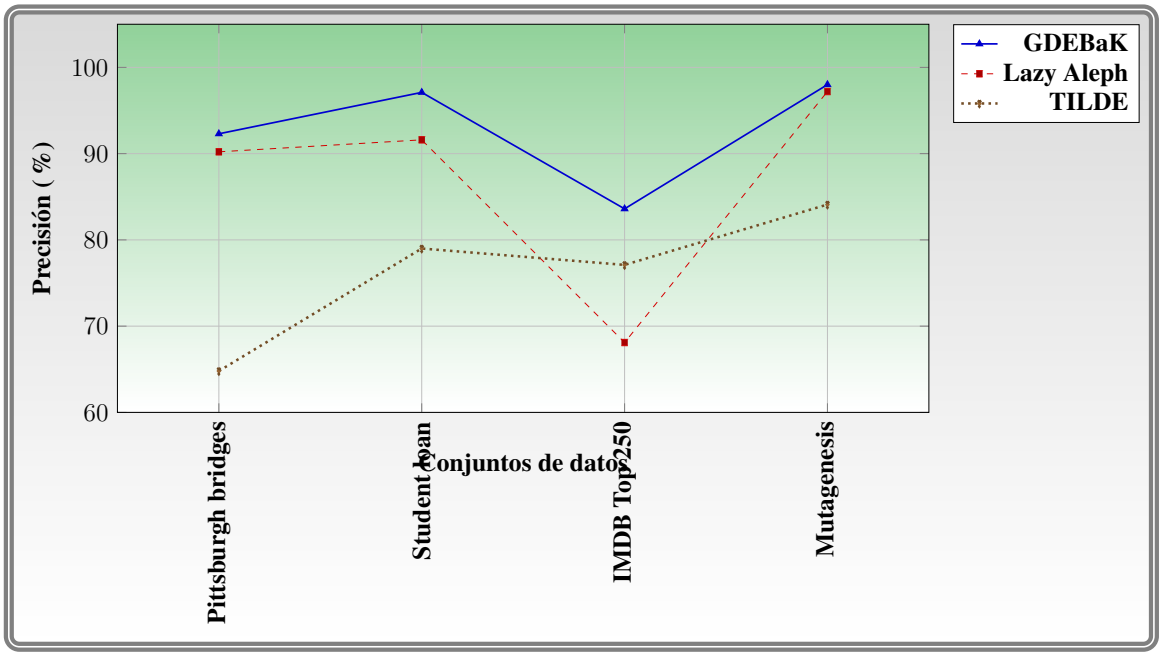

Figura 5.4: Esta gráfica muestra las precisiones obtenidas al aplicar agrupamiento y discretización con cada conjunto de datos (solo con GDEBaK). Ya que Aleph y TILDE no tienen método de agrupamiento solo se discretizaron los atributos numéricos en cada caso.

Conjunto de datos	**Precisión**		**No. Reglas**		**Método**
	Local	**Global**	**Local**	**Global**	
Australian	71.4	**72.1**	**5.9**	9.3	Discretization
Pittsburgh bridges	89.8	89.8	8.9	8.6	Grouping
Japanese Credit	73.5	**74.3**	**3.0**	3.5	Discretization
Ecoli	96.1	96.7	10.2	9.7	Discretization
German Credit	69.8	70.1	42.5	43.1	Grouping
Metabolism	**63.9**	61.8	**11.72**	12.73	Grouping
Mutagenesis	95.78	**98.24**	1.6	1.7	Discretization
Student Loan	**95.29**	90.0	41	47.7	Grouping

Tabla 5.8: Comparación entre procesamiento local y global.

Como podemos ver en estos experimentos GDEBaK es competitivo en cuanto a la mejora de precisión y la disminución del número de reglas. En el siguiente capítulo presentamos nuestras conclusiones de acuerdo a los resultados obtenidos, y planteamos las líneas de investigación a seguir a partir de este trabajo.

Capítulo 6

Conclusiones y trabajo futuro

> "No son los más fuertes de la especie los que sobreviven, ni los más inteligentes. Sobreviven los más flexibles y adaptables a los cambios."
>
> Charles Darwin

En este trabajo se revisaron diferentes métodos para el manejo de atributos numéricos y categóricos en ILP. Se encontró que para los atributos numéricos las estrategias principales estuvieron basadas en la discretización perezosa, en la programación lógica de restricciones, en la proposicionalización y en algoritmos genéticos. Sin embargo, estos métodos dependen principalmente de sus operadores por lo que si se desean utilizar en otros sistemas ILP deberá modificarse cada operador o incluso su algoritmo de búsqueda. Por otro lado, solo tres de los sistemas ILP revisados toman en cuenta a los atributos categóricos.

Tomando en cuenta lo anterior se propuso una estrategia univariable para el manejo de atributos numéricos y categóricos en ILP. Esta estrategia utiliza un algoritmo genético para encontrar los mejores puntos de división y los mejores subconjuntos de categorías (para atributos numéricos y categóricos respectivamente). Esta información junto con nuevas relaciones se añaden al conocimiento previo justo antes de inducir la siguiente regla de la teoría correspondiente. Por lo tanto, el método propuesto es independiente de los operadores de refinamiento y del proceso de búsqueda, lo cual permite que pueda ser implementado en cualquier sistema ILP. De esta manera los puntos de división y los subconjuntos de categorías son utilizados también en la creación de las reglas de la teoría final.

Debido a que no todos los sistemas ILP tienen el *software* disponibles solo pudimos comparar nuestro método, llamado GDEBaK (Grouping and Discretization for Enriching

the Backgound Knowledge), con discretización perezosa en Aleph y discretización en TILDE. Tomando en cuenta los resultados obtenidos podemos concluir los siguientes puntos:

- A diferencia de los métodos revisados en el estado del arte nuestra propuesta permite trabajar tanto con atributos numéricos como con atributos categóricos. Además, permite agrupar y discretizar conjuntos de datos con más de dos clases, y no solo con ejemplos positivos y negativos.
- La calidad de las teorías finales no depende solamente del algoritmo ILP o del método de agrupamiento y discretización sino de los atributos seleccionados y usados para esos análisis. Si al discretizar o agrupar un atributo la precisión o el número de reglar no cambia entonces es atributo no es relevante dentro del contexto del conjunto de datos.
- Aquellos problemas cuyas teorías finales no necesitan valores constantes para ser inducidas no pueden ser beneficiadas al discretizar o agrupar atributos. En dichos casos, las variables son más útiles para explicar mejor el concepto objetivo. Un ejemplo de esto, ver Figura 6.1, es la teoría que explica la relación de abuelo entre dos personas:

```
abuelo_de(A,B):-
  padre_de(A,C), padre_de(C,B).

abuelo_de(A,B):-
  padre_de(A,C), madre_de(C,B).
```

Figura 6.1: Teoría de la relación abuelo entre dos personas.

- El método voraz es bueno cuando los conjuntos de datos son simples, como en el problema *Iris dataset*, en donde la mayoría de los sistemas tienen una precisión arriba del $90\,\%$ pero su rendimiento es bajo con conjuntos de datos más complejos como IMDB o Mutagenesis.
- En promedio el método GDEBaK fue mejor que los demás métodos, por lo que podemos decir que el método propuesto en este trabajo puede ser útil para mejorar las teorías inducidas. Por otro lado, como se esperaba el método voraz tiene un rendimiento en promedio más bajo que los otros.
- Finalmente, ya que los usuarios no saben de antemano cuáles atributos serán los más útiles, para obtener las mejores teorías, la intuición será un elemento importante al

momento de elegir dichos atributos. El método propuesto le permite al usuario elegir fácilmente los atributos que serán analizados.

El presente trabajo sirve como punto de partida para las siguientes líneas de investigación:

- Ya que no en todos los casos se pudo mejorar la calidad de las teorías inducidas es necesario probar con nuevas funciones de aptitud, tanto para atributos numéricos como para los categóricos. De esta manera será posible identificar los factores que influyen en la precisión y el número de reglas de cada teoría.
- También es importante considerar las posibles correlaciones entre dos o más atributos. Para ello métodos como el análisis discriminante (lineal y cuadrático) podría ser de mucha utilidad.
- Por último, podría ser necesario y útil diseñar e implementar métodos para identificar los atributos más relevantes. De esta manera, para la creación de cada teoría, solo se tomarán en cuenta los atributos identificados removiendo aquellos que no influyen en la generación de mejores teorías.

Referencias

[1] C. F. AHMED, N. LACHICHE, C. CHARNAY, S. EL JELALI, AND A. BRAUD, *Flexible propositionalization of continuous attributes in relational data mining*, Expert Systems with Applications, 42 (2015), pp. 7698–7709.

[2] S. ANTHONY AND A. M. FRISCH, *Generating numerical literals during refinement*, in Inductive Logic Programming: 7th International Workshop, ILP-97, Springer Verlag, 1997, pp. 61–76.

[3] R. AUDI AND P. AUDI, *The Cambridge dictionary of philosophy*, vol. 584, Cambridge university press Cambridge, 1999.

[4] S. AUGIER, G. VENTURINI, AND Y. KODRATOFF, *Learning first order logic rules with a genetic algorithm*, in in Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, AAAI Press, 1995, pp. 21–26.

[5] H. BLOCKEEL AND L. DEHASPE, *The ACE Data Mining System, User's Manual*, 2009.

[6] H. BLOCKEEL AND L. D. RAEDT, *Lookahead and discretization in ilp*, in In Proceedings of the 7th International Workshop on Inductive Logic Programming, Springer-Verlag, 1997, pp. 77–85.

[7] H. BOSTRÖM AND L. ASKER, *Combining divide-and-conquer and separate-and-conquer for efficient and effective rule induction*, in International Conference on Inductive Logic Programming, Springer, 1999, pp. 33–43.

[8] M. BOTTA AND A. GIORDANA, *Smart+: A multi-strategy learning tool*, in IJCAI, 1993, pp. 937–945.

[9] I. BRATKO, *Prolog programming for Artificial Intelligence*, Addison Wesley, 1989.

[10] R. CAMACHO, *An experimental comparison of human and machine learning formalisms*, in In The Eighth Scandinavian Conference on Artificial Intelligence, 2003, pp. 47–58.

[11] J. CHENG, C. HATZIS, H. HAYASHI, M.-A. KROGEL, S. MORISHITA, D. PAGE, AND J. SESE, *Kdd cup 2001 report*, ACM SIGKDD Explorations Newsletter, 3 (2002), pp. 47–64.

[12] K. J. CIOS, W. PEDRYCZ, R. W. SWINIARSKI, K. J. CIOS, W. PEDRYCZ, AND R. W. SWINIARSKI, *Clustering*, Data mining methods for knowledge discovery, (1998), pp. 375–429.

[13] V. S. COSTA, R. ROCHA, AND L. DAMAS, *The yap prolog system.*, TPLP, 12 (2012), pp. 5–34.

[14] L. DE RAEDT AND W. VAN LAER, *Inductive constraint logic*, in Algorithmic Learning Theory: 6th International Workshop, ALT'95 Fukuoka, Japan, October 18–20, 1995 Proceedings 6, Springer, 1995, pp. 80–94.

[15] A. P. DEMPSTER, N. M. LAIRD, AND D. B. RUBIN, *Maximum likelihood from incomplete data via the em algorithm*, JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B, 39 (1977), pp. 1–38.

[16] P. DEVYVER AND J. KITTLER, *Pattern Recognition: A Statistical Approach*, Prentice-Hall, 1982.

[17] F. DIVINA AND E. MARCHIORI, *Evolutionary concept learning.*, in GECCO, 2002, pp. 343–350.

[18] F. DIVINA AND E. MARCHIORI, *Handling continuous attributes in an evolutionary inductive learner*, IEEE Trans. Evolutionary Computation, 9 (2005), pp. 31–43.

[19] S. DŽEROSKI, *Multi-relational data mining: an introduction*, SIGKDD Explor. Newsl., 5 (2003), pp. 1–16.

[20] U. M. FAYYAD AND K. B. IRANI, *Multi-interval discretization of continuous-valued attributes for classification learning*, in IJCAI, 1993, pp. 1022–1029.

[21] S. FERILLI, N. FANIZZI, N. D. MAURO, AND T. M. A. BASILE, *Efficient theta-subsumption under object identity*, in AI * IA Workshop su Apprendimiento Automatico: Metodi e Applicazioni dell'Ottavo Convegno della Associazione Italiana per L'Intelligenza Artificiale (Siena, Italy, 2002), 2002.

[22] R. FISHER, *Linear discriminant analysis*, Ann. Eugenics, 7 (1936), p. 179.

[23] A. FRANK AND A. ASUNCION, *UCI machine learning repository*, 2010. http://archive.ics.uci.edu/ml.

[24] R. FRANK, F. MOSER, AND M. ESTER, *A method for multi-relational classification using single and multi-feature aggregation functions*, in European Conference on Principles of Data Mining and Knowledge Discovery, Springer, 2007, pp. 430–437.

[25] J. H. FRIEDMAN, *Regularized discriminant analysis*, Journal of American Statistical Association, (1989), pp. 165–175.

[26] R. GNANADESIKAN, *Methods for Statistical Data Analysis of Multivariate Observations*, A Wiley-Interscience publication, Wiley, 1997.

[27] D. E. GOLDBERG, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st ed., 1989.

[28] GROUPLENSRESEARCH, *Movielens ml-100k*. http://www.grouplens.org, 1998.

[29] J. HAN, *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[30] B. HEMADA AND K. LAKSHMI, *A study on discretization techniques*, Int. J. of Engineering Research & Technology, 2 (2013), pp. 1887–1892.

[31] J. H. HOLLAND, *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, 1975.

[32] D. IENCO, R. G. PENSA, AND R. MEO, *Context-based distance learning for categorical data clustering*, in Advances in Intelligent Data Analysis VIII, 2009, pp. 83–94.

[33] IMDB, *Imdb top 250*. http://www.imdb.com/chart/top, 2010.

[34] B. KACZMAREK AND A. KNOBBE, *Multi-relational data mining*, vol. 145, Ios Press, 2006.

[35] H. KIM AND W.-Y. LOH, *Classification trees with unbiased multiway splits*, Journal of the American Statistical Association, (2001), pp. 589–604.

[36] S. KRAMER, N. LAVRAČ, AND P. FLACH, *Propositionalization approaches to relational data mining*, Springer, 2001.

[37] W. V. LAER, S. DZEROSKI, AND L. D. RAEDT, *Multi-class problems and discretization in icl (extended abstract)*, in In Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming (ILP for KDD, 1996, pp. 53–60.

[38] N. LAVRAC AND S. DZEROSKI, *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, New York, 1994.

[39] W.-Y. LOH AND Y.-S. SHIH, *Split selection methods for classification trees*, Statistica sinica, (1997), pp. 815–840.

[40] B. T. LOWERRE, *The harpy speech recognition system.*, Carnegie Mellon University, 1976.

[41] J. MACQUEEN, *Some methods for classification and analysis of multivariate observations*, in In 5-th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 281–297.

[42] D. MALERBA, F. ESPOSITO, G. SEMERARO, AND S. CAGGESE, *Handling continuous data in top-down induction of first-order rules*, in AI*IA, 1997, pp. 24–35.

[43] G. J. MCLACHLAN, *Mahalanobis distance*, Resonance, 4 (1999), pp. 20–26.

[44] R. S. MICHALSKI, *Pattern recognition as rule-guided inductive inference*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2 (1980), pp. 349–361.

[45] D. MICHIE, S. MUGGLETON, D. PAGE, AND A. SRINIVASAN, *To the international computing community: A new east-west challenge*, Distributed email document available from http://www. doc. ic. ac. uk/˜ shm/Papers/ml-chall. pdf, (1994).

[46] T. M. MITCHELL, *Generalization as search*, Artificial intelligence, 18 (1982), pp. 203–226.

[47] T. M. MITCHELL, *Machine Learning*, McGraw-Hill, Inc., New York, NY, USA, 1 ed., 1997.

[48] O. MUÑOZ AND R. MACKINNEY-ROMERO, *Multivalue in ILP*, in Latest Advances in Inductive Logic Programming, World Scientific, 2015, pp. 143–150.

[49] S. MUGGLETON, *Inverse entailment and progol*, New generation computing, 13 (1995), pp. 245–286.

[50] S. MUGGLETON AND L. D. RAEDT, *Inductive logic programming: Theory and methods*, Journal of Logic Programming, 19/20 (1994), pp. 629–679.

[51] O. MUÑOZ-TEXZOCOTETLA AND R. MACKINNEY-ROMERO, *An evolutionary-based approach for dealing with numerical and categorical attributes in ilp*, Computational Intelligence, (2019).

[52] G. D. PLOTKIN, *A note on inductive generalization*, Machine Intelligence, 5 (1970), pp. 153–163.

[53] R. QUINLAN, *Learning logical definitions from relations*, Machine Learning, (1990), pp. 239–266.

[54] S. N. SRIHARI, *Document image understanding*, in Proceedings of 1986 ACM Fall joint computer conference, ACM '86, Los Alamitos, CA, USA, 1986, IEEE Computer Society Press, pp. 87–96.

[55] A. SRINIVASAN, *The Aleph Manual*, 2004.

[56] A. SRINIVASAN AND R. CAMACHO, *Experiments in numerical reasoning with inductive logic programming*, Journal of Logic Programming, (1997).

[57] A. SRINIVASAN, R. KING, S. H. MUGGLETON, AND M. STERNBERG, *Carcinogenesis predictions using ILP*, in Inductive Logic Programming, Springer, 1997, pp. 273–287.

[58] A. SRINIVASAN, S. MUGGLETON, AND R. KING, *Comparing the use of background knowledge by inductive logic programming systems*, in Proceedings of the 5th International Workshop on Inductive Logic Programming, 1995, pp. 199–230.

[59] A. SRINIVASAN, S. MUGGLETON, R. D. KING, AND M. J. STERNBERG, *Mutagenesis: Ilp experiments in a non-determinate biological domain*, in Proceedings of the 4th international workshop on inductive logic programming, vol. 237, Citeseer, 1994, pp. 217–232.

[60] A. M. TURING, *Computing machinery and intelligence*, 1950. One of the most influential papers in the history of the cognitive sciences: http://cogsci.umn.edu/millennium/final.html.

[61] I. WITTEN, E. FRANK, AND M. HALL, *Data Mining: Practical Machine Learning Tools and Techniques*, Morgan Kaufmann series in data management systems, Elsevier Science & Technology, 2011.

[62] L. YU AND H. LIU, *Feature selection for high-dimensional data: A fast correlation-based filter solution*, in Proceedings of the 20th international conference on machine learning (ICML-03), 2003, pp. 856–863.

[63] L. ZHENG, C. LIU, D. JIA, AND N. ZHONG, *A new approach to constraint inductive logic programming.*, in ISMIS, N. Zhong, Z. W. Ras, S. Tsumoto, and E. Suzuki, eds., vol. 2871 of Lecture Notes in Computer Science, Springer, 2003, pp. 357–364.

Apéndice A

Ejemplo completo con mutagenesis

Para este ejemplo, presentado en [51], utilizamos el conjunto de datos *mutagenesis* [59], el cual, contiene 230 moléculas con actividad mutagénica y está dividido en dos subconjuntos de 188 y 44 moléculas. Cada molécula tiene un valor mutagénico obtenido con la prueba *Ames* [58].

La tarea de aprendizaje para este conjunto de datos consiste en discriminar las moléculas con mutagenicidad logarítmica positiva de aquellas que tienen mutagenicidad logarítmica negativa o de valor cero. Para este ejemplo usamos el subconjunto de 188 moléculas. De este subconjunto 125 compuestos tienen mutagenicidad logarítmica positiva y están etiquetados con la clase *activa.* Estos son los ejemplos positivos. Las otras 63 moléculas están etiquetadas con la clase *inactiva* y son los ejemplos negativos. Como conocimiento previo, utilizamos información de estás moléculas disponible en tres formas: descripciones estructurales primitivas de las moléculas, propiedades químicas específicas, y conocimiento químico genérico de plantillas estructurales. Esta información está explicada detalladamente en el trabajo de *Srinivasan* [58].

Analizamos el rendimiento de GDEBaK al discretizar y agrupar los atributos numéricos y categóricos respectivamente. Los resultados obtenidos los comparamos con la discretización perezosa del sistema Aleph y la discretización llevada a cabo en el sistema TILDE. Para determinar si nuestro método mejora significativamente el rendimiento de las otras dos técnicas aplicamos la prueba *t apareada* sobre la validación cruzada de 10 iteraciones.

A.1. Descripción del conocimiento previo.

El conjunto de datos *mutagenesis* proporciona información de cada molécula en los siguientes predicados:

- `bond/4` y `atm/5`. Estos predicados proporcionan los átomos, enlaces, tipos de enlaces, tipos de átomos y cargas parciales de cada molécula. Esta información está contenida en $10,136$ hechos.

- `logp/2` y `lumo/2`. El primero proporciona una medida de la hidrofobicidad (repelen el agua), y el segundo es la energía del orbital desocupado más bajo del compuesto correspondiente.

- *Otros.* El resto de predicados proporciona definiciones de grupos metilo, grupos nitro, anillos aromáticos, anillos heteroaromáticos, anillos conectados, longitud de anillos, etc. Estos predicados son los siguientes: `act/2, benzene/2, carbon_5_aromatic_ring/2, carbon_6_ring/2, hetero_aromatic_6_ring/2, hetero_aromatic_5_ring/2, ring_size_6/2, ring_size_5/2, nito/2, methyl/2, anthracene/2, phenanthrene/2, ball3/2`

- *Predicados auxiliares.* los predicados `member/2` y `connected/2` facilitan la representación de cada molécula.

A continuación se presentan las declaraciones usadas en TILDE y en Aleph (para discretización perezosa y GDEBaK).

A.2. Definición de lenguaje en Aleph

Para definir el lenguaje en Aleph, utilizamos las declaraciones de modo: `modeh/2` para la relación objetivo y `modeb/2` para el resto de las relaciones. Estas establecen cómo se va a construir cada hipótesis. Los símbolos $+$ (entrada) and $-$ (salida) indican los lugares donde se permiten las variables. El símbolo $\#$ indica los lugares donde se permiten valores constantes. El primer argumento de cada declaración de modo es un número $n \geq 1$, llamado *recall*. Este argumento es usado para acotar el número de instancias que se pueden generar. Un asterisco, en lugar de un número entero, indica que no hay límite en el número de instancias que pueden generarse. Los siguientes modos definen el lenguaje con el que se crearon las hipótesis para este ejemplo.

```
% Declaraciones para Aleph (discretización perezosa y GDEBaK)
:- modeh(1,active(+drug)). % relación objetivo

:- modeb(1,lumo(+drug,-energy)).
:- modeb(1,logp(+drug,-hydrophob)).
:- modeb(1,bond(+drug,+atomid,-atomid,-numberBond)).
:- modeb(1,atm(+drug,-atomid,-element,-floatAtm1,-charge)).
```

```
:- modeb(1,act(+drug, -floatAct )).
:- modeb(1,benzene(+drug, -ring )).
:- modeb(1,carbon_5_aromatic_ring(+drug, -ring )).
:- modeb(1,carbon_6_ring(+drug, -ring )).
:- modeb(1, hetero_aromatic_6_ring (+drug, -ring )).
:- modeb(1, hetero_aromatic_5_ring (+drug, -ring )).
:- modeb(1,ring_size_6 (+drug, -ring )).
:- modeb(1,ring_size_5 (+drug, -ring )).
:- modeb(1,nitro(+drug, -ring )).
:- modeb(1,methyl(+drug, -ring )).
:- modeb(1,anthracene(+drug, - ringlist )).
:- modeb(1,phenanthrene(+drug, - ringlist )).
:- modeb(1,ball3(+drug, - ringlist )).
:- modeb(1,member(-ring, + ringlist )).
:- modeb(1,member(+ring, + ringlist )).
:- modeb(1,connected(+ring, +ring )).
```

Las siguientes declaraciones, usadas sólo para GDEBaK, indican todos los atributos que serán procesados.

```
:- discretization ( active ( link ), lumo(link , attribute )).
:- discretization ( active ( link ), logp( link , attribute )).
:- discretization ( active ( link ), bond(link , no, no, attribute )).
:- discretization ( active ( link ), atm(link , no, no, no, attribute )).
:- discretization ( active ( link ), act ( link , attribute )).

:- grouping( active ( link ), atm(link , no, attribute , no, no )).
```

Las siguientes declaraciones, sólo usadas para Aleph con discretización perezosa, indican los atributos que serán discretizados.

```
% El segundo argumento en la relación gteq/2 y en lteq/2 es
% un argumento numérico
:- modeb(1,gteq(+energy, #energy )).
:- modeb(1,lteq(+energy, #energy )).
:- modeb(1,geq(+hydrophob, #hydrophob)).
:- modeb(1,leq(+hydrophob, #hydrophob)).
:- modeb(1,geq(+numberBond, #numberBond)).
:- modeb(1,leq(+numberBond, #numberBond)).
:- modeb(1,geq(+floatAtm1, #floatAtm1 )).
:- modeb(1,leq(+floatAtm1, #floatAtm1 )).
:- modeb(1,geq(+floatAct, # floatAct )).
:- modeb(1,leq(+ floatAct , # floatAct )).
:- modeb(1,geq(+charge, #charge )).
:- modeb(1,leq(+charge, #charge )).
```

```
% Declaraciones para indicar que se ejecute
% discretización perezosa
:- lazy_evaluate ( gteq /2).
:- lazy_evaluate ( lteq /2).
```

A.3. Definición de lenguaje en TILDE

En el sistema TILDE, la declaración equivalente a `mode` es `rmode`, y sus argumentos (variables y constantes permitidas) son los mismos que en Aleph. Las siguientes declaraciones indican el lenguaje para la creación de las hipótesis en TILDE.

```
% Relación objetivo
predict ( active (+drug,- class )).

rmode(1: lumo(+Drug, -Number)).
rmode(1: logp(+Drug, -Number)).
rmode(1: bond(+Drug, +Atomid, -Atomid, -Number)).
rmode(1: atm(+Drug, -Atomid, -Element, -Number, -Number)).
rmode(1: act(+Drug, -Number)).
rmode(1: benzene(+Drug, -Ring)).
rmode(1: carbon_5_aromatic_ring (+Drug, -Ring)).
rmode(1: carbon_6_ring (+Drug, -Ring)).
rmode(1: hetero_aromatic_6_ring (+Drug, -Ring)).
rmode(1: hetero_aromatic_5_ring (+Drug, -Ring)).
rmode(1: ring_size_6 (+Drug, -Ring)).
rmode(1: ring_size_5 (+Drug, -Ring)).
rmode(1: nitro (+Drug, -Ring)).
rmode(1: methyl(+Drug, -Ring)).
rmode(1: anthracene(+Drug, - Ringlist )).
rmode(1: phenanthrene(+Drug, - Ringlist )).
rmode(1: ball3 (+Drug, - Ringlist )).
rmode(1: member(-Ring, +Ringlist )).
rmode(1: member(+Ring, +Ringlist )).
rmode(1: connected(+Ring, +Ring)).
```

A.4. Teorías aprendidas

A continuación presentamos las teorías obtenidas con los tres métodos mencionados anteriormente. Debemos aclarar que para método sólo presentamos una teoría para una

sola iteración de la validación cruzada. Como podemos apreciar en el caso de nuestra propuesta la teoría obtenida contiene únicamente sólo una regla y con muy buena precisión. No mucho mayor que la obtenida con Aleph. Por otro lado, la teoría obtenida con TILDE no sólo es menos precisa sino que necesita de cuatro reglas para poder describirla.

```
% Discretización y agrupamiento con GDEBaK
[theory]

[Rule 1] [Pos cover = 112 Neg cover = 2]
active(A) :-
   act(A,B), aGteq(B,-0.02).

[Training set performance]
            Actual
         +          -
    + 112           2         114
Pred
    -   0          54          54

        112         56         168

Accuracy = 0.988

[Test set performance]
            Actual
         +          -
    +   13           0          13
Pred
    -    0           7           7

         13          7          20

Accuracy = 1.0
```

```
% Discretización perezosa en Aleph
[theory]

[Rule 1] [Pos cover = 79 Neg cover = 0]
active(A) :-
   act(A,B), geq(B,1.51).

[Rule 3] [Pos cover = 108 Neg cover = 0]
active(A) :-
   act(A,B), geq(B,0.15).

[Training set performance]
            Actual
         +          -
    + 108           0         108
Pred
    -   5          56          61

[Test set performance]
            Actual
         +          -
    +   11           0          11
Pred
    -    1           7           8
```

```
        113          56          169                12          7          19

Accuracy = 0.970                          Accuracy = 0.947
```

```
% Discretización en TILDE. Programa equivalente
active (A,[neg]) :- lumo(A,B),B=< -1.145,logp(A,C),C=<6.57,C=<2.03, !.
% 17.0/22.0=0.772727272727273
active (A,[pos]) :- lumo(A,B),B=< -1.145,logp(A,C),C=<6.57, !.
% 109.0/124.0=0.879032258064516
active (A,[neg]) :- lumo(A,B),B=< -1.145, !.
% 3.0/3.0=1.0
active (A,[neg]).
% 20.0/20.0=1.0

Training :                                Testing :
Number of examples: 169                   Number of examples: 19

REAL\PRED | pos neg                       REAL\PRED | pos neg
------------------------------            ------------------------------
pos       | 109   5   | 114               pos       | 8     3   | 11
neg       | 15    40  | 55                neg       | 2     6   | 8
------------------------------            ------------------------------
          | 124   45  | 169                         | 10    9   | 19

    Accuracy: 0.88165                     Accuracy: 0.7368
```

Como hemos podido ver en las teorías presentadas anteriormente, los métodos usados con Aleph fueron más precisos que TILDE. Dicho comportamiento ocurre con varios conjuntos de datos, ver Sección 5.3.

Casa abierta al tiempo
UNIVERSIDAD AUTÓNOMA METROPOLITANA

ACTA DE DISERTACIÓN PÚBLICA

No. 00021
Matrícula: 2113802441

Enriquecimiento del conocimiento previo en ILP

ORLANDO MUÑOZ TEXZOCOTETLA
ALUMNO

REVISÓ

MTRA. ROSALÍA SERRANO DE LA PAZ
DIRECTORA DE SISTEMAS ESCOLARES

En la Ciudad de México, se presentaron a las 16:00 horas del día 6 del mes de marzo del año 2023 en la Unidad Iztapalapa de la Universidad Autónoma Metropolitana, los suscritos miembros del jurado:

DR. EDUARDO MORALES MANZANARES
DR. MIGUEL ANGEL GUTIERREZ ANDRADE
DR. PEDRO LARA VELAZQUEZ
DR. MARIO GRAFF GUERRERO
DR. RENE MAC KINNEY ROMERO

Bajo la Presidencia del primero y con carácter de Secretario el último, se reunieron a la presentación de la Disertación Pública cuya denominación aparece al margen, para la obtención del grado de:

DOCTOR EN CIENCIAS (CIENCIAS Y TECNOLOGIAS DE LA INFORMACION)

DE: ORLANDO MUÑOZ TEXZOCOTETLA

y de acuerdo con el artículo 78 fracción IV del Reglamento de Estudios Superiores de la Universidad Autónoma Metropolitana, los miembros del jurado resolvieron:

Aprobar

Acto continuo, el presidente del jurado comunicó al interesado el resultado de la evaluación y, en caso aprobatorio, le fue tomada la protesta.

DIRECTOR DE LA DIVISIÓN DE CBI

Roman Linares Romero
DR. ROMAN LINARES ROMERO

PRESIDENTE

DR. EDUARDO MORALES MANZANARES

VOCAL

DR. MIGUEL ANGEL GUTIERREZ ANDRADE

VOCAL

DR. PEDRO LARA VELAZQUEZ

VOCAL

DR. MARIO GRAFF GUERRERO

SECRETARIO

DR. RENE MAC KINNEY ROMERO