Un Estudio del Aprendizaje Inductivo Incremental en Primer Orden

WULFRANO ARTURO LUNA RAMÍREZ

Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Maestría en Inteligencia Artificial

APROBADA:

Dr. Alejandro Guerra Hernández

Dr. Manuel Martínez Morales

Dr. Cora Cora Beatriz Excelente Toledo

Rubén de la Mora Basañez, MIA
Director de la Facultad de Física e Inteligencia Artificial

A mis

padres con afecto, por sus esfuerzos, esperanzas y la enseñanza de vida.

A Cecilia, la flor de sorpresas, por enseñarme a oír su canto de sirena

Un Estudio del Aprendizaje Inductivo Incremental en Primer Orden

por

WULFRANO ARTURO LUNA RAMÍREZ

TESIS

Presentada ante la Facultad Física en Inteligencia Artificial

Universidad Veracruzana

para obtener el grado de

MAESTRO EN INTELIGENCIA ARTIFICIAL

Universidad Veracruzana
Facultad de Física e Inteligencia Artificial
Maestría en Inteligencia Artificial

UNIVERSIDAD VERACRUZANA

20 de agosto de 2007

Índice general

Página

Lista de Tablas

Lista de Figuras

Introducción

El Aprendizaje Automático tiene como tarea central hacer que un sistema mejore su funcionamiento a través de la experiencia. En términos algorítmicos, ésto se concibe como la búsqueda de aquella hipótesis que mejor se ajuste a una experiencia dada. Existen diferentes tipos de aprendizaje: supervisado, no supervisado y por refuerzo, siendo el aprendizaje supervisado el que se estudiará en este trabajo.

Las aplicaciones del Aprendizaje Automático cubren una gran variedad de campos, entre los que se encuentran: la Clasificación, la Minería de Datos, la Robótica, y los Sistemas Multi-Agente. En particular, se menciona que el aprendizaje en Sistemas Multi-Agente (implementados como agentes BDI [36, 23, 35, 28]) representa la motivación práctica de este trabajo y una de sus principales aplicaciones.

En el presente estudio se hace una exploración al aprendizaje inductivo incremental en primer orden desde la perspectiva de la programación lógica inductiva, en él, se tratan dos aspectos del aprendizaje: la *representación* y la *incrementalidad*. El estudio culmina con la implementación y el escalamiento de un algoritmo de aprendizaje inductivo: TILDE [3, 4], en el cual la hipótesis adopta la forma de un árbol lógico, dando lugar al algoritmo incremental ILDT.

En el Capítulo 1 se presenta una introdución al Aprendizaje Automático y a la Programación Lógica Inductiva, en él se aborda la terminología del área necesaria para el subsecuente desarrollo de este trabajo.

Los temas de representación e incrementalidad son expuestos en el Capítulo 2. En cuanto a la primera, se menciona que el aprendizaje supervisado utiliza conjuntos de entrenamiento (la experiencia), los cuales puede estar expresados en términos de pares $atributo - valor$ (*representación proposicional*) o en forma de expresiones de la lógica de primer orden (*representación relacional*). De foma específica, la representación relacional se considera más expresiva que la proposicional, pues permite denotar relaciones o correspondencias entre ob-

jetos, lo que aumenta la capacidad del sistema para aprender otro tipo de características o conceptos, ampliando así su utilidad. Un punto notorio en la representación es la forma en que está expresada la hipótesis, tema que es expuesto también en este capítulo. Por su parte, en el *aprendizaje incremental* se realiza la búsqueda en un ambiente cambiante, donde las evidencias disponibles se están actualizando a lo largo del tiempo. Este flujo de las evidencias hace necesaria la revisión constante de la hipótesis aprendida [32], tarea del algoritmo de aprendizaje. La necesidad de aprender incrementalmente se presenta en situaciones donde no es factible reunir un conjunto de aprendizaje adecuado, por ejemplo: cuando la cantidad de información disponible es muy grande; ésta tiene una disposición geográfica dispersa; o bien, por la escasez de datos. Adicionalmente, en este capítulo se menciona la relevancia que tiene el *sesgo inductivo* para el aprendizaje y la forma en que puede explotarse para conseguir el aprendizaje incremental en primer orden.

En el Capítulo 3 se expone una metodología de escalamiento para obtener algoritmos incrementales en primer orden a partir de algoritmos proposicionales y versiones no incrementales en primer orden. El escalamiento se realiza partiendo de la representación basada en *interpretaciones*. La metodología consiste, de manera breve, en lo siguiente: a) seleccionar el algoritmo que será escalado (según la tarea a realizar); b) cambiar la representación de pares atributo-valor a expresiones en primer orden; c) utilizar los operadores necesarios para realizar la búsqueda de las hipótesis (tomando en cuenta el algoritmo original y la nueva representación); y por último, d) implementar, probar y añadir otras características necesarias.

Las pruebas y sus resultados se muestran en el Capítulo 4, donde también se presenta la aplicación de ILDT como mecanismo de preservación de la consistencia de un sistema de agentes BDI. Finalmente, en el Capítulo 5 son presentados las conclusiones y el trabajo a futuro.

1. Propuesta

En la presente tesis se propone ILDT, un algoritmo de aprendizaje automático incremental

en primer orden, el cual fue obtenido al aplicar la metodología de escalamiento a Tilde, realizando el paso incremental al estilo de ID4. Las principales caractarísticas de ILDT son las siguientes:

- Usa una *representación en primer orden* basada en interpretaciones.
- Usa un sesgo de lenguaje implementado como especificaciones de *rmodes* bajo θ-subsunción.
- Construye *incrementalmente* hipótesis en forma de árboles lógicos, siguiendo una estrategia de expansión y revisión de las estructura del árbol conforme a la implementada en el algoritmo incremental ID4.

2. Justificación

El aprendizaje incremental en primer orden pone al alcance de los algoritmos la capacidad de aprender conceptos complejos como las relaciones que existen entre varios objetos que pueden estar inmersos en un escenario cambiante; además éste es necesario si se pretende atacar problemas donde no existe un conjunto de entrenamiento completo, ya sea porque no es posible recopilarlo en su totalidad, porque no es factible reunirlo debido a su gran tamaño o bien por su diversa distribución geográfica.

Un ejemplo de problemas que requieren un aprendizaje incremental en primer orden se encuentra en los Sistemas Multi-Agente, particularmente, los agentes basados en el modelo BDI: debido a que la información que cada agente puede obtener del ambiente es cambiante e incompleta, ésta se configura como un flujo de datos que son aprovechables para mejorar su comportamiento individual mediante un proceso de aprendizaje. Lo anterior se puede extrapolar al comportamiento colectivo mediante la acción coordinada basada en la actualización de creencias entre los agentes del sistema completo, según lo establecido por el protocolo SMILE [11], el cual especifica un procedimiento de aprendizaje denominado *Mecanismo M*, que se encarga de restablecer la consistencia de las creencias de cada agente entre sí obteniendo con ello la consistencia del sistema completo.

Existe trabajo previo sobre la implementación de este protocolo en agentes BDI el cual utiliza un aprendizaje parcialmente incremental, basado en el uso repetido de Tilde cada vez que se cuenta con datos nuevos [17]. A este respecto se propone ILDT como una forma de implementar el mecanismo M, debido a dos ventajas principales:

1) Emplea la representación en primer orden (utilizada en agentes BDI).

2) Proporciona la incrementalidad demandanda por SMILE en el mecanismo M, para garantizar la *eficiencia local* en la actualización de creencias y con ello la aplicación plena del protocolo [18].

Capítulo 1

Antecedentes

El Aprendizaje se refiere al uso de observaciones del medio en el que se está inmerso para el mejoramiento de la conducta, esto es, no sólo actuar en el momento sino tomar esa experiencia para actuar en el futuro; a esto se le llama aprendizaje inductivo [29].

Para ejemplificar lo anterior, tomemos por caso un agente -robótico o de software- situado en un determinado ambiente en constante cambio. Nuestro agente tiene que decidir, a partir de la información que del medio recoge, cuáles acciones llevará a cabo y aprovechar las acciones pasadas y las nuevas observaciones para que su conducta se adecue a los cambios y por ende, mejore su desempeño en dicho ambiente.

Suponiendo que el agente de nuestro ejemplo está construido como un agente BDI [23], implementado mediante un lenguaje de programación basado en una lógica de primer orden restringida con eventos y acciones, tal como AgentSpeak(L) [27]. Entonces, la conducta del agente se basa en los programas escritos en este lenguaje, de tal manera, el agente cuenta con planes para decidir sus acciones. En este caso, el proceso de aprendizaje puede centrarse en las condiciones bajo las cuales son correctos sus planes [13, 14, 15, 16, 17], de tal manera, el agente utiliza sus percepciones acerca de los contextos de los planes como ejemplos de entrenamiento para realizar el aprendizaje.

El escenario así planteado configura un problema de aprendizaje inductivo que tiene dos características: a) se basa en una *representación de primer orden* (los planes y sus contextos, sus creencias, deseos e intenciones) y b) es de carácter *incremental* (las observaciones de los agentes se presentan como un flujo de evidencias, no como un conjunto completo de ellas).

Partiendo del escenario anteriormente planteado, en el presente trabajo se explora el aprendizaje incremental inductivo en primer orden. De esta forma, en este Capítulo se hace

una revisión del Aprendizaje Automático para tener una idea clara de la terminología y del problema en cuestión, a saber: hacer que un agente aprenda de la experiencia. Además, se exponen algunos conceptos básicos de la Programación Lógica Inductiva, con el fin de entender las características de la representación y del aprendizaje usando representaciones en primer orden. Finalmente se presentan algunas de las aplicaciones del Aprendizaje Automático en otros campos.

1.1. Aprendizaje Automático

El campo del Aprendizaje Automático o Aprendizaje de Máquina tiene como objetivo desarrollar sistemas −programas de computadora− que puedan mejorar su funcionamiento para realizar una tarea de forma automática, basándose en la experiencia, la cual puede consistir en datos de ejemplo de esa tarea en particular (aprendizaje supervisado) [22, 1, 21].

El Aprendizaje Automático se ha utilizado en varias áreas, como el reconocimiento de patrones, la extracción de conocimiento de bases de datos y la clasificación. Sus aplicaciones incluyen medicina, videojuegos, astronomía entre muchos otros campos [22, 1, 21].

Una forma de abordar el aprendizaje es considerarlo un problema de búsqueda. De esta manera, se trata de encontrar dentro de un espacio de hipótesis posibles aquella que mejor corresponda a los datos observados, utilizando un conocimiento *a priori*, lo cual no es una práctica común en los sistemas de aprendizaje.

De forma más precisa, el Aprendizaje Automático se puede definir de la siguiente manera [22]:

Def 1 ***Aprendizaje Automático****. Dados los siguientes elementos: una experiencia* E*, una medida de desempeño* P *y una clase de tareas* T*, se dice que un programa* aprende *si su desempeño mejora con la experiencia* E *al realizar* T *de acuerdo con el valor de* P*.*

Un concepto importante en aprendizaje es el de **función objetivo**, que puede entenderse como la definición del tipo de conocimiento que debe ser aprendido. Este conocimiento puede

referirse por ejemplo, a la evaluación de una tarea representativa dentro de un dominio particular. De modo más formal la función objetivo se define de la siguiente manera:

$$F : E \rightarrow A \tag{1.1}$$

Donde E puede ser el conjunto de estados posibles en el ambiente; A es el conjunto de acciones o valores asignados a las acciones. Por ejemplo, E podrá representar el conjunto de posiciones (estados) en un tablero de ajedrez, y A la evaluación en números reales de los movimientos posibles dadas las posiciones [22].

A partir de lo anterior, el Aprendizaje Automático puede entenderse como la búsqueda de la descripción de una función objetivo con el fin de utilizar algún algoritmo para computarla. Comúnmente la descripción de tal función es *no operacional* −no puede computarse eficientemente. Como consecuencia, los sistema de aprendizaje, buscan una *descripción operacional* de ella, lo que se denomina *aproximación de funciones*.

Al diseñar un sistema de aprendizaje se deben tomar en cuenta varios elementos: el tipo de experiencia de entrenamiento, la medida de desempeño, la función objetivo y su representación así como el algoritmo para aproximarla.

En función de la clase de experiencia utilizada, el Aprendizaje Automático puede dividirse en tres tipos principales [29]:

1. *Supervisado*: Consiste en aprender una función a partir de ejemplos de entradas y salidas, es decir, las clases son definidas previamente y con base en ellas se clasifican los datos.

2. *No supervisado*: Consiste en aprender patrones de entradas cuando no hay valores de salida especificados. Las clases se infieren de los datos, creando grupos diferenciados.

3. *Por refuerzo*: El aprendizaje se basa en la evaluación de un *refuerzo* o *recompensa* para el conjunto de acciones realizadas.

De acuerdo con T. Mitchell [22], un concepto central en el aprendizaje es la *inducción*, que consiste en aprender funciones generales a partir de ejemplos particulares. De esta manera, el Aprendizaje de Conceptos busca una definición de una categoría general basándose en ejemplos positivos y negativos de ella. Cada concepto, también llamado **concepto objetivo**, puede entenderse como una función booleana c definida sobre el conjunto de categorías dado, es decir:

$$c : X \rightarrow \{0, 1\} \tag{1.2}$$

Donde X es el conjunto de ejemplares positivos y negativos de la categoría en cuestión.

Esta clase de aprendizaje parte de la hipótesis siguiente:

Def 2 ***Hipótesis del aprendizaje inductivo***. *Cualquier estimación (hipótesis) que aproxime la función objetivo a partir de un conjunto "suficientemente" grande de ejemplos de entrenamiento, puede aproximar también la función objetivo para ejemplos no observados.*

De esta forma, un algoritmo puede "aprender" a realizar una tarea, por ejemplo clasificar datos, si obtiene una buena aproximación de la función objetivo tomando en cuenta sólo los ejemplos de entrenamiento.

Un elemento importante para efectuar aprendizaje de conceptos como una búsqueda, es el universo de posibles aproximaciones de la función objetivo (hipótesis). A este conjunto se le denomina **espacio de búsqueda**. Si se toma en cuenta que dicho espacio puede ser finitio o infinito [22], surge la necesidad de contar con una manera sistemática de explorar este espacio. Afortunadamente, existe una estructura inherente al problema de aprender conceptos: un *ordenamiento de lo general a lo específico.*

El ordenamiento mencionado se determina de forma intuitiva con la relación $\geq_g$ (**más o tan general que**), que se define partiendo de este hecho[22]:

> para cualquier instancia x en X y cualquier hipótesis h en H, se dice que x satisface h, si y sólo si $h(x) = 1$.

Ahora bien, dadas las hipótesis h_j y h_k, h_j es más_o_tan_general_que h_k, si y sólo si cualquier instancia que satisfaga h_k satisface también h_j.

Formalmente, esto se expresa como sigue[22]:

Def 3 ***Relación más_o_tan_general_que:*** $\geq_g$.

Sean h_j y h_k funciones booleanas definidas sobre X, se dice que $h_j \geq_g h_k$ si y sólo si:

$$(\forall x \in X)\ [(h_k(x) = 1 \rightarrow h_j(x) = 1)]$$

De esta definición, se establecen otras relaciones, como las siguientes [22]:

- generalidad estricta: $h_j >_g h_k$ si y sólo si: $(h_j \geq_g h_k \wedge h_k \not\geq_g h_j)$.
- más_específico_que: $h_j \leq_g h_k$, si $h_k \geq_g h_j$.

Esta característica permite que el espacio sea explorado exhaustivamente evitando la enumeración explícita de cada hipótesis, debido a que establece un *orden parcial* (es una relación reflexiva, antisimétrica y transitiva), con ello pueden descartarse partes del espacio que no conducen al concepto objetivo, reduciendo el tiempo de búsqueda. Partiendo de este ordenamiento, se pueden definir *fronteras* de hipótesis que acoten el espacio de búsqueda. Así se tendrían como límites las hipótesis más específicas y las más generales. Esto induce una estructura de rejilla (*lattice*, en inglés) del espacio de búsqueda, donde cada clase de equivalencia formada por la relación $\geq_g$ representa a un conjunto de hipótesis que pueden tomarse en cuenta o no, así, basta con probar una de ellas para continuar o descartar la búsqueda en esa dirección [22, 3]. En el Capítulo 2 se retomará el concepto del orden en el espacio de búsqueda a partir de la relación θ-subsunción, debido a su relevancia con el trabajo desarrollado en esta tesis.

Como última parte de esta sección, se menciona que uno de los aspectos más importantes en Aprendizaje Automático es la representación, pues ningún algoritmo es capaz de aprender algo que no puede representar[22]. En atención a ello, en el apartado que sigue se exponen algunos conceptos de Programación Lógica Inductiva, mismos que son de utilidad para los

temas de representación y de aprendizaje incremental, los cuales serán abordados en el capítulo siguiente.

1.2. Programación Lógica Inductiva

En la sección anterior, se introdujo la Hipótesis del Aprendizaje Inductivo (Def 2), la cual postula que se puede aprender una función objetivo a partir de datos observados, de manera que un sistema puede contender exitosamente con nuevas evidencias (no observadas), utilizando la función objetivo aproximada en el proceso de aprendizaje. No obstante, conviene resaltar el hecho de que el aprendizaje no parte siempre de cero, sino que con frecuencia se tiene un **conocimiento base** (CB), el cual es relevante para la tarea que se requiere aprender [31]. Retomando esta idea dentro del marco de la lógica, S. Muggletton introdujo en 1991 [24, 31] el nuevo campo denominado **Programación Lógica Inductiva** (PLI), definida como la intersección entre el Aprendizaje Automático y la Programación Lógica (que conjunta la expresión de los datos en lógica de primer orden y métodos de inferencia).

En las Tablas 1.1 y 1.2 se muestra un resumen de la terminología utilizada en Programación Lógica y, por ende, en PLI, de acuerdo con lo expuesto en [22, 8].

En primer lugar, hay que señalar que el uso de lógica de primer orden confiere las siguientes ventajas[31]:

- Disponibilidad de un conjunto de conceptos, técnicas y resultados que han sido bien estudiados y entendidos.
- Establecer una uniformidad en la representación (en un lenguaje clausal), tanto para el CB, como de la hipótesis por aprender (también denominada *teoría inducida*) y del conjunto de entrenamiento.
- Facilidad para interpretar y entender la hipótesis inducida por el sistema de aprendizaje.

Alfabeto Es un conjunto de constantes, functores y símbolos de predicado.

Constante Expresión cuyo valor no cambia, por ejemplo: jaime, monica.

Variable Expresión que puede tomar distintos valores, por ejemplo: X, Y.

Término Es una constante, una variable o un término compuesto aplicado a cualquier término.

Término compuesto Se forma con un símbolo *n-ario* de función f (functor) y n términos t_i: $f(t_i, \ldots, t_n)$. Las funciones toman alguno de los valores de las constantes.

Átomo Expresión lógica formada por un símbolo *n-ario* de predicado y n términos t_i: $p(t_i, \ldots, t_n)$. Los predicados toman valores de Falso o Verdadero.

Literal Es un átomo ($hombre(juan)$) o la negación de un átomo ($\neg hombre(monica)$), llamados átomos positivos o negativos respectivamente.

Cláusula Cualquier disyunción de literales $L_1 \vee \ldots L_n$ cuyas variables están universalmente cuantificadas.

Cláusula de Horn Cláusula que contiene, a lo más, una literal positiva: $H \leftarrow (L_1 \wedge, \ldots, \wedge L_n)$. Donde H (cabeza o consecuente) y $L_1 \wedge, \ldots, \wedge L_n$ (cuerpo o antecedente) son literales positivas. Debido a las equivalencias $(A \leftarrow B) \equiv (A \vee \neg B)$, y $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$ la cláusula de Horn puede tomar esta forma: $H \vee \neg L_1 \vee \ldots \vee \neg L_n$

Tabla 1.1: Terminología básica usada en Programación Lógica Inductiva [22, 8].

El conjunto de entrenamiento se divide en dos partes, ejemplos positivos E^+, o negativos E^-, y corresponden a expresiones fundamentadas (del inglés *grounded*), es decir, sin variables [31]. En el siguiente capítulo se tocará más extensamente este tema.

Una característica de los algoritmos de PLI es la *corrección* de la teoría inducida. De modo informal, se dice que una teoría aprendida es correcta si es *completa* (tanto S como CB validan todos los ejemplos de E^+), y si es *consistente* (ni S ni CB implican ejemplos negativos E^-). Con base en lo anterior, se presenta la configuración normal o explicativa de la tarea de PLI [31], enunciada como se muestra a enseguida:

Def 4 ***Configuración normal del problema de PLI.***

A partir de:

- *Un conjunto finito CB de cláusulas (que puede estar vacío).*
- *Un conjunto de ejemplos positivos E^+ y negativos E^-.*

Substitución Es cualquier función que instancia un término, átomo o cláusula , es decir, reemplaza variables por términos. Por ejemplo, $\theta\{X/yo, Y/Z\}$ reemplazaría la variable X por el término yo y la variable Y por el termino Z. De este modo $L\theta$ denota la aplicación de la substitución θ a la literal L, esto es, el reemplazo simultáneo de todas las variables de L por los términos indicados en θ.

Substitución unificadora Para dos literales L_1 y L_2, es cualquier substitución cuyo resultado sea la equivalencia: $L_1\theta \equiv L_2\theta$.

Interpretación Es una asignación de valores que formaliza la veracidad o falsedad de fórmulas e implicaciones.

Interpretación de Herbrand Instanciación de variables, que hace verdaderos a los elementos de un conjunto de átomos sobre un alfabeto, correspondientemente, los átomos que quedan fuera de tal asignación son considerados falsos (caso booleano). De forma general, es un conjunto de átomos fundamentados (sin variables) pertenecientes a un alfabeto.

Teoría verdadera Una teoría (clausal) T es verdadera en una interpretación de Herbrand I si $T\theta$ es verdadera en I para cada θ para la que $T\theta$ es fundamentada.

Teoría fundamentada Una teoría es fundamentada $T\theta$ es verdadera en una interpretación de Herbrand I is y sólo si cada cláusula de $T\theta$ es verdadera en I.

Modelo Es aquella interpretación que hace verdadera a una teoría. Dicha interpretación es un modelo de la teoría.

Implicación lógica $F \models G$ denota que F implica lógicamente a G cuando todos los modelos de F son también modelos de G. Por otro lado, se dice que $F \models \perp$ (F no se satisface), si no existe una interpretación que sea un modelo de F.

Tabla 1.2: (Continuación) Terminología básica usada en Programación Lógica Inductiva [22, 8].

Encontrar: *Una teoría S, tal que $S \cup CB$ sea correcta con respecto a E^+ y E^-.*

Un aspecto que hay que señalar en PLI es que existen conjuntos E^+ y E^- para los cuales no hay una teoría que sea correcta [31]. Esto se debe a dos razones: a) puede ser que $S \cup CB$ sea inconsistente con respecto a los ejemplos negativos, por ejemplo cuando un ejemplo es positivo y negativo al mismo tiempo. b) que el problema en cuestión tenga un número infinito de ejemplos, lo que ocasiona que existan más ejemplos que teorías, por lo que no habrá una teoría que cubra todos los ejemplos.

Otra posibilidad se presenta cuando la teoría encontrada no tiene poder predictivo, es decir, sólo se ajusta a los ejemplos pertenecientes a E^+ y a nada más, lo cual es opuesto

al cometido del Aprendizaje Automático. Una forma de minimizar este efecto es añadir restricciones a la teoría, lo cual será posible dependiendo de la tarea a resolver, como acotar el número de cláusulas que deba contener la teoría con respecto a la cardinalidad del conjunto de ejemplos [31].

Ahora bien, tomando en cuenta la existencia de una o más teorías correctas para los conjuntos de ejemplos positivos y negativos que se tengan, el aprendizaje consiste en la *búsqueda* de la teoría correcta de entre el universo de cláusulas permitidas (el espacio de búsqueda). Según se indicó en la sección anterior, para realizar esta búsqueda es de gran importancia la existencia de un orden en el dicho espacio, pues permite efectuar una revisión sistemática de las cláusulas [22, 10]. Tal ordenamiento puede establecerse, como ya se dijo, mediante la especificidad de las hipótesis buscadas, esto es, considerando la búsqueda de hipótesis de la más general a la más específica o viceversa. Así, existen dos formas principales de realizar esta búsqueda: Descendente o Ascendente (*Top-down y Bottom-up*, en inglés), según comiencen con una teoría demasiado general -que valida cualquier ejemplo-, o demasiado específica -que no valida ningún ejemplo-, respectivamente.

Dicha búsqueda se realiza con apego a dos operaciones: *especialización* (hacer que S, junto con CB, sea más específica de forma que no implique a ningún ejemplo negativo) y *generalización* (hacer que S y CB validen todos los ejemplos positivos).

En el Capítulo 2 se hablará de un modelo de deducción denominado θ-subsunción [25, 8] que es usado en PLI para conseguir este ordenamiento en el espacio de búsqueda y sirve de base para definir *operadores de refinamiento*, los cuales generan hipótesis candidatas adecuadas para orientar la búsqueda.

Finalmente se menciona que, además de la configuración de PLI introducida antes, existe la configuración basada en *interpretaciones*, de la cual se hablará en el capítulo siguiente por estar estrechamente relacionada con el aprendizaje incremental y con la representaicón del conjunto de entrenamiento.

1.3. Áreas de aplicación

Antes de cerrar este capítulo, se destaca la estrecha relación del Aprendizaje Automático con otras áreas, como la Clasificación, la Minería de Datos, la Robótica y los Sistemas Multi-agente, entre muchas otras. En algunas de ellas, se requiere implementar aprendizaje debido a la necesidad de contar con procedimientos automáticos que puedan identificar objetos (y relaciones entre ellos); o analizar la información en grandes cantidades y obtener conocimiento a partir de ésta.

Por ejemplo, cuando se busca que un agente −robótico o de *software*−se adapte ante cambios y situaciones de un entorno en constante alteración, se debe tener un módulo de aprendizaje; incluso si se desea que un conjunto de ellos se coordinen para efectuar una tarea o compartan conocimiento, esto puede abordarse como un problema de aprendizaje automático, en particular, *aprendizaje incremental*.

Así, el conjunto de entrenamiento se configura a partir de la experiencia de lagente, es decir, de las observaciones que éste recoge del ambiente.

Ahora bien, cuando los Sistemas Multi-Agente realizan tareas en conjunto, se necesita comunicación y coordinación entre ellos. Esta situación representa un problema de aprendizaje incremental, concretamente, de aprendizaje colaborativo, pues cada agente interactúa con los otros para actualizar sus estados internos según los acuerdos a los que llegue con ellos, lo que integra nuevamente un flujo de observaciones susceptibles de utilizarse para aprender y mejorar el desempeño de los agentes en cuestión. Una forma de realizar este aprendizaje lo establece el protocolo SMILE (*Sound Multi-agent Incremental LEarning*) [11], el cual preserva la consistencia de los agentes que colaboran entre sí sin el uso de una memoria central. Así, se dice que el sistema completo es consistente cuando todos sus miembros lo son. El protocolo implementa un procedimiento de aprendizaje incremental para restablecer la consistencia en el sistema.

En la parte de la implementación se mencionará de nuevo este método, como una de las aplicaciones del trabajo presentado en esta tesis.

Capítulo 2

Representación e Incrementalidad

En el presente capítulo se exponen los conceptos en torno a la representación de los datos y al aprendizaje incremental que sirven de base para el desarrollo de esta tesis. De forma concreta, se presentan las siguientes ideas:

1. La representación en primer orden usando interpretaciones.
2. La adopción de árboles lógicos para representar la hipótesis.
3. La estructuración del espacio de búsqueda usando un marco de generalidad.
4. La construcción incremental de hipótesis a partir de un flujo de evidencias (un conjunto de entrenamiento que se va incrementando paulatinamente).

Las dos primeros conceptos están relacionados con el sesgo inductivo de los algoritmos de aprendizaje: el lenguaje para expresar el espacio de búsqueda y la hipótesis. El tema de la representación abarca dos tipos: relacional y proposicional, que son la base para introducir los temas subsecuentes. En cuanto a la forma de la hipótesis se muestran dos, a saber árboles de decisión y árboles lógicos (sección 2.2). Se presenta además, el algoritmo ID3 (pág. 24), por ser característico del aprendizaje de hipótesis en forma de árboles de decisión.

El tercero punto está relacionado tanto con el sesgo inductivo como con la forma en que la hipótesis es buscada. En la subsección 2.1.1 se muestra una discusión al respecto y se señala la importancia para este trabajo.

En tanto que el último punto referido tiene que ver con la estrategia a seguir para resolver el problema de aproximar incrementalmente la función objetivo, en otras palabras: encontrar una hipótesis apropiada a partir de datos incompletos. En primer lugar, se destaca

el algoritmo denominado eliminación de candidatos (pág. 37) como idea base para introducir la incrementalidad. El tema se cierra con con los algoritmos ID4 e ID5r (págs. 39 y 40).

A partir de estas ideas se estructura una propuesta para obtener un algoritmo incremental en primer orden, siguiendo la metodología de escalamiento que se expone en el siguiente capítulo. A continuación se presentan a detalle los conceptos enunciados anteriormente.

2.1. Representaciones del Conjunto de Entrenamiento

Los algoritmos de aprendizaje automático inductivo utilizan un conjunto de entrenamiento. La forma en que está expresado este conjunto de ejemplos se conoce como *representación*. La representación establece aquello que el algoritmo será capaz de aprender, pues determina el tipo de conceptos que pueden derivarse de los datos, como se verá en este apartado.

Una forma de representar los ejemplos de entrenamiento es considerar a cada uno de ellos como un conjunto de atributos con sus valores correspondientes, al cual se le asocia una clase determinada. A esta representación de los ejemplos de aprendizaje se le denomina **proposicional** o **atributo-valor**. En la Tabla 2.1 se muestra un ejemplo de esta representación.

Forma	**Tamaño**	**Color**	**Clase**
Triángulo	Grande	Azul	+
Rectángulo	Grande	Rojo	+
...			
Círculo	Chico	Verde	−

Tabla 2.1: Un ejemplo expresado en representación proposicional.

Uno de los problemas de esta representación es su limitada **expresividad**, ya que existen situaciones donde se tienen varios objetos y relaciones entre ellos, por ejemplo, si se quisiera describir una escena [33], con variaciones en objetos y sus posiciones, como lo ilustra la figura 2.1, la representación proposicional sería inadecuada, por el número de atributos que deben crearse para describir los objetos presentes y sus relaciones, en este caso, las posiciones, el número de ellos y sus movimientos.

La descripción de una escena conlleva tomar en cuenta las características siguientes:

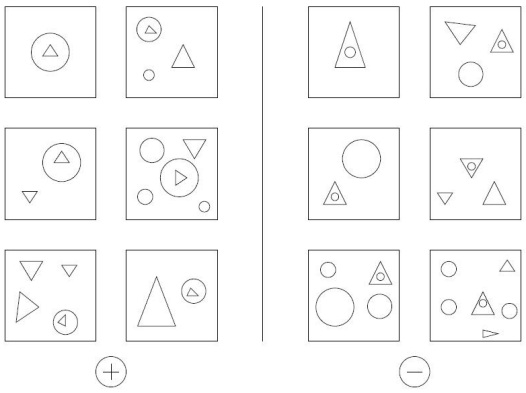


Figura 2.1: Se muestran dos conjuntos de imágenes, correspondientes a escenas de clase + y −, los cuales integran un conjunto de entrenamiento. La descripción de una escena es un problema para el cual la representación en primer orden es adecuada por su posibilidad de expresar relaciones fácilmente, como los cambios de las figuras mostradas en la imagen (triángulo, círculo o cuadrado), la cuales son codificadas eficazmente, facilitándo con ello su clasificación. Adaptado de [33].

- No existe un número fijo de objetos.
- El orden de ellos no es constante, lo que conduce a un número exponencial de representaciones posibles.
- No todas las características son comunes a todos los objetos.
- Si se quisiera denotar relaciones entre objetos, se crearía nuevamente un crecimiento exponencial de acuerdo con el número de éstos.

Para superar esta cuestión, una alternativa es utilizar una **representación relacional**, también denominada **Representación en Primer Orden**,en la cual un ejemplo se considera como un conjunto de hechos fundamentados (del inglés *grounded facts*), que corresponden a tuplas en una Base de Datos relacional, o a una *interpretación de Herbrand* en terminología de la Lógica de Primer Orden.

Cabe señalar además que cada atributo preserva un valor para cada ejemplo y se da por sentado el *supuesto del mundo cerrado*, esto es, si un hecho no está presente en el ejemplo, se supone falso. La Representación en Primer Orden tiene las siguientes ventajas:

- Puede denotar un número ilimitado de objetos.
- No se requiere tener un orden en los objetos.
- Distintos objetos pueden tener distintas propiedades
- El número de relaciones entre los objetos puede ser ilimitado.

Hay que destacar que la Representación en Primer Orden es más general y expresiva que la Reprersentación Atributo-Valor, la cual puede considerarse como un caso especial de ella [33].

Dos formas de traducir una tabla de k pares atributo-valor, mencionadas en la referencia [33], son las que a continuación se muestran:

1. Determinar un hecho que represente a cada ejemplo (tupla o renglón en la Representación Atributo-Valor):

   $\{ejemplo(val_1,\ \ldots,\ val_k\}$

2. Relacionar cada renglón (ejemplo) con un conjunto de k-hechos:

   $\{atrib_1(val_1),\ \ldots,\ atrib_k(val_k)\}$

   Donde: val_i = valor del i-ésimo atributo del ejemplo.

En la Tabla 2.2 se muestran dos ejemplos de la Representación en Primer Orden (ver Tabla 2.1).

1)	{ejemplo(triángulo,grande,azul,+)}
2)	{forma(triángulo),tamaño(grande),color(azul),clase(positivo)}

Tabla 2.2: Dos formas de representación en primer orden de un ejemplo.

De esta manera, la escena superior derecha de la figura 2.1 (un triángulo chico apuntando hacia arriba, dentro de un círculo grande) se puede describir en primer orden usando la forma 2 mostrada en la Tabla 2.2:

{clase(positivo), objeto(o1), objeto(o2), figura(o1,circulo), tamaño(o1,grande), figura(o2,triángulo), tamaño(o2,chico), apunta(o2,arriba),adentro(o2,o1)}

Nuevamente hay que destacar con respecto al Aprendiaje Automático y a las representaciones de los ejemplos, que ningún algoritmo de aprendizaje puede aprender algo si no es capaz de representarlo, es decir, de interpretar una representación dada. A consecuencia de ello, utilizar una representación para los datos sobre los que se debe aprender determina las características del algoritmo de aprendizaje.

2.1.1. Sesgo inductivo

Un concepto central en los algoritmos de aprendizaje, particularmente los desarrollados en Programación Lógica Inductiva es el **sesgo inductivo**, entendido como cualquier aspecto que se tome como base para seleccionar alguna generalización (o especialización) distinto a la consistencia estricta con los datos de entrenamiento [7].

De tal forma, el sesgo puede ser alguno de los factores que se enuncian a continuación:

- El lenguaje en el cual se describen las hipótesis.
- El espacio de hipótesis a considerear.
- Los procedimientos que definen la forma en que se considerarán las hipótesis.
- El criterio de aceptación definido para que un algoritmo se detenga, tomando como resultado la hipótesis obtenida hasta ese momento.

De manera más formal, el sesgo inductivo, se define como aparece enseguida [22]:

Def 5 ***Sesgo inductivo***. *Dados un algoritmo de aprendizaje L, un conjunto de evidencias X, un concepto arbitrario c definido sobre X, un conjunto de ejemplos de entrenamiento correspondientes a c, denotado por $D_c = \{\langle x, c(x)\rangle\}$, y $L(x, D_c)$ una clasificación asignada por L a x_i después del entrenamiento con los datos D_c.*

El sesgo inductivo de L es cualquier conjunto mínimo de supuestos B tal que para cualquier concepto objetivo y sus correspondientes ejemplos D_c, justifican la inferencia inductiva como inferencia deductiva (el concepto aprendido con los datos observados sirve para las nuevas evidencias):

$$(\forall x \in X)[(B \wedge D_c \wedge x) \vdash L(x, D_c)]$$

De acuerdo con T. Mitchell [22], el sesgo inductivo se divide en dos tipos principales: puede tratarse de un **sesgo de búsqueda** o *sesgo preferencial*, o puede ser un *sesgo restrictivo*, mejor conocido como **sesgo de lenguaje**. Este último determina las características sintácticas y de operación de las hipótesis (qué recibe como entrada y cuál es su salida) [20].

Una topología más general del sesgo inductivo se presenta en [7] y es la siguiente:

- *Sesgo de lenguaje*. Define las restricciones del lenguaje en el que se expresarán las descripciones de los conceptos (*lenguaje de descripción del concepto*). Por ejemplo, adoptar cláusulas de Horn sin functores y con un cuerpo reducido a n literales.

- *Sesgo de búsqueda*. Determina cuál parte del espacio de búsqueda debe analizarse y la forma en que ha de hacerse. Un sesgo de lenguaje, de acuerdo al criterio de preferencia, puede ser:

  - De preferencia: indica el orden en que deben considerarse las cláusulas y cuáles predicados deben añadirse o removerse del cuerpo de las cláusulas.
  - De restricción: define qué hipótesis deben ignorarse.

- *Sesgo de validación*. Especifica las circunstancias por las cuales debe detenerse la búsqueda. Es el criterio de paro, por ejemplo el grado de compleción y consistencia de la hipótesis alcanzada.

Partiendo de esta última topología, el sesgo de búsqueda puede a su vez adoptar otras formas [7], de las cuales se mencionan las más relevantes para este trabajo:

- Relación de orden en el espacio de hipótesis: esto es aprovechamiento del orden en el espacio, por ejemplo el que establece la relación de generalidad de la Def 3 (pág. 5).

- Función de cobertura: Determina si una hipótesis obtenida es correcta con respecto a los ejemplos.

Una mención especial merece el hecho de que el conocimiento base actúa como un sesgo de búsqueda basado tanto en la relación de orden del espacio como en la función de cobertura. Es en este último sentido como se utiliza en el presente trabajo (ver la sección 2.3).

De acuerdo con T. Mitchell [22], es preferible un sesgo de búsqueda que uno de lenguaje, porque permite al algoritmo trabajar en un espacio completo de hipótesis con lo que se asegura la contención del concepto objetivo.

En contraparte, se ha observado un mejor rendimiento de los programas de aprendizaje frente a nuevas evidencias -no presentes en el conjunto de entrenamiento-, puesto que el porcentaje de errores cometidos por el programa frente al conjunto de entrenamiento, contrastado con datos de prueba, crece de acuerdo con el tamaño del espacio de hipótesis [7], lo que lleva a pensar que contar con un espacio de hipótesis reducido conducirá a un algoritmo de aprendizaje más preciso. Adicionalmente, en el caso de la representación en primer orden, el espacio posible de hipótesis crece estrepitosamente, lo que hace necesario restringir la búsqueda también. Más específicamente, para el caso de las cláusulas de Horn (utilizadas por muchos algoritmos de Programación Lógica Inductiva) se indica la necesidad de restringir el espacio de hipótesis para obtener un funcionamiento correcto del algoritmo [7].

No obtante lo anterior, dada su expresividad, la Representación en Primer Orden pone al alcance del algoritmo la capacidad de aprender conceptos que quedan fuera de los límites de la Representación Atributo-Valor, por ejemplo:

- Facilita el aprovechamiento de conocimiento base [20].

- Confiere la capacidad de aprender a partir de múltiples relaciones [33, 3, 4, 10].

En este orden de ideas, hay que considerar la existencia de grandes cantidades de información contenida en Bases de Datos Relacionales (BDR), que puede ser analizada mediante

algoritmos de aprendizaje en primer orden [30]. En concordancia con esto, existe una similitud entre las BDR y la Representación en Primer Orden, manifiesta en la posibilidad de convertir una BDR en un programa lógico (conjunto de cláusulas), por lo que existe un paralelismo entre los conceptos que ambas manejan, si bien la terminología cambia [30, 10], por ejemplo: el conocimiento base, cuya importancia ya se ha destacado, puede obtenerse a partir de las *vistas* definidas para una BDR [3, 4], lo que posibilita el uso de algoritmos de Programación Lógica Inductiva para el análisis de los datos presentes en Base de Datos Relacionales, explotando con ello el aprendizaje de relaciones múltiples [10]. Por último, se menciona que los resultados obtenidos por los programas de Programación Lógica Inductiva pueden ser interpretados claramente por los usuarios [20].

Retomando lo que se ha mencionado sobre el sesgo inductivo, es importante señalar que la Representación en Primer Orden, tal como es utilizada en Programación Lógica Inductiva, tiene implícitas dos formas del sesgo: a) el lenguaje en que se expresan los ejemplos y la hipótesis; b) el uso de conocimiento base para realizar la búsqueda.

Resultado de estas dos formas del sesgo inductivo, se presenta en el siguiente punto un marco de generalidad utilizado en algoritmos de Programación Lógica Inductiva para realizar la búsquda: θ-subsunción.

2.1.2. θ-subsunción

En el Capítulo 1 se mencionó la relación $\geq_g$ como una forma de estructurar el espacio de búsqueda, lo que lleva a acotarlo y buscar eficientemente la hipótesis en cuestión. En este sentido, θ-subsunción (definida por Plotkin en 1970 [3, 10, 25]), ha sido utilizada extensamente en PLI [3] por dos características: es fácil de calcular, y establece un *quasi-orden* que induce una relación de equivalencia en el espacio y un orden parcial en las clases de equivalencia [3].

θ-subsunción puede verse como una versión incompleta de la implicación, donde una literal puede implicar a otra, sin θ-subsumirla [3].

Por otro lado, un *quasi-orden* $\leq$ es una relación reflexiva y transitiva, pero no antisimétri-

ca, por lo que se puede dar el caso siguiente: $a \leq b$ y $b \leq a$ donde $a \neq b$.

Recordando el concepto de substitución mostrado en la Tabla 1.2 del Capítulo 1, que establece que una una substitución $\theta = \{V_1/t_1, \ldots, V_n/t_n\}$ es una asignación de los términos t_i a las variables V_i[10], se puede definir la relación θ-subsunción como se expone a continuación [25, 3]:

Def 6 ***θ-subsunción***. *Una cláusula c_1 θ-subsume ($\leq_\theta$) a una cláusula c_2, si y sólo si existe una substitución θ tal que $Lits(c_1\theta) \subseteq Lits(c_2)$.*
Donde: $Lits(c)$ es el conjunto de las literales que ocurren en una cláusula c escrita como una disyunción. Esto es, c_1 es una generalización de c_1 y c_2 es una especialización de c_1 bajo θ-subsunción.

La características de θ-subsunción se muestran a continuación, según lo expuesto en [25]:

- **Implicación**. Si $c_1 \leq_\theta c_2$, entonces $c_1 \models c_2$, pero no lo opuesto. Esto último hace que no se puedan θ-subsumir relaciones recursivas: Si $c_1 = p(f(X)) \leftarrow p(X); c_2 = p(f(f(Y))) \leftarrow p(Y)$, entonces $c_1 \models c_2$, pero $c_1 \not\leq_\theta c_2$. Esto es, la deducción usando θ-subsunción no equivale a la implicación entre cláusulas.

- **Crecimiento infinito**. A partir de una clásula se puede determinar una cadena infinita de cláusulas θ-subsumidas, cuyas fronteras son las máximas especializaciones o generalizaciones posibles en el dominio.

- **Equivalencia**. Existen diferentes cláusulas equivalentes bajo θ-subsunción. Esto permite que el algoritmo necesite a lo más una cláusula de las existentes en las clases de equivalencia.

- **Reducción**. Existe una cláusula representativa de cada clase de equivalencia, la cláusula reducida r. Sea c una cláusula, r es el subconjunto mínimo de literales de c tal que r es equivalente a c.

- **Estructura de rejilla**. El conjunto de cláusulas reducidas forma una estructura de rejilla (del inglés *lattice*), esto es, existe entre dos cláusulas una frontera de generalización y una frontera de especialización.

Si se considera a los miembros de una clase de equivalencia *variantes sintácticas*, se puede considerar en la búsqueda sólo una variante sintáctica. Tal es el fundamento del operador de refinamiento usado para considerar las hipótesis, es decir explorar el espacio de búsqueda. Esto configura un sesgo inductivo para ILP. De este operador se hablará con mayor detalle en el capítulo siguiente.

Como comentario final sobre el tema de la representacíon, se indica que además de la forma en que está expresado el conjunto de entrenamiento, es necesario revisar la representación de la hipótesis que ha de inducirse mediante el algoritmo de aprendizaje automático. Este tema es tratado en la sección que continúa.

2.2. Representación de la hipótesis

Otro aspecto a considerar en los algoritmos de aprendizaje es la representación que adopta la hipótesis. Para el caso de los algoritmos proposicionales, la hipótesis puede expresarse como listas de decisión (conjunción de proposiciones) o árboles de decisión (disyunción de conjunciones proposicionales). En tanto que para el caso relacional, la hipótesis puede ser una conjunción de literales en primer orden, expresadas como listas, o bien, como árboles lógicos.

En este trabajo, la forma de la hipótesis buscada adopta la forma de un árbol lógico de decisión. De forma general, se considera que los árboles de decisión son un método práctico para realizar inferencia inductiva de conceptos. Son empleados en tareas de clasificación y regresión. Tienen como características importantes la siguientes: a) aproximan funciones con valores discretos; b) son robustos ante datos faltantes o ruidosos; c) pueden reescribirse como reglas *si-entonces* para facilitar la comprensión por parte de los usuarios no especializados; y sobre todo, d) representan descripciones disyuntivas de conceptos [22, 1].

Hay que señalar que ésta última característica es relevante para la aplicación presentada en este estudio (ver la sección , pág. 64) por lo enunciado a continuación: debido a que cada rama del árbol (la ruta desde la raíz hasta una hoja) representa un grupo de conjunciones y el árbol completo representa varias expresiones conjuntivas, los árboles lógicos pueden expresar convenientemente los contextos de los planes de cada agente (que están en orimer orden), ya que dichos contextos son elementos centrales en la tarea de aprendizaje en agentes BDI [16, 17].

A continuación se habla de dos formas de representación de la hipótesis: árboles de decisión y árboles lógicos.

2.2.1. Árboles de Decisión

El resultado del algoritmo de aprendizaje es una aproximación de una función objetivo, es decir, la hipótesis inducida del conjunto de entrenamiento. Para abordar la representación de la hipótesis, conviene apoyarse en la tarea de clasificación debido a que en este trabajo se emplearán árboles lógicos para representarla, que son la contraparte en primer orden de los árboles de decisión, los cuales expresan información proposicional y han sido utilizados como clasificadores.

La tarea de **clasificación** consiste en tomar una decisión de pertenencia con respecto a una situación determinada, teniendo como base la información disponible. De forma más acotada, un procedimiento de clasificación consiste en la construcción de un mecanismo aplicable a una secuencia continua de casos, que determina la pertenencia de éstos a una **clase** predefinida, basándose en sus características o **atributos** [21].

Se han utilizado varias técnicas para realizar clasificación, entre ellas están algunos métodos estadísticos y el Aprendizaje Automático (dentro del cual se destacan por su uso las Redes Neuronales Artificiales y el Aprendizaje Inductivo).

Como parte del trabajo realizado en los campos del Aprendizaje Automático y la Clasificación se encuentran los **Árboles de Decisión** los cuales pueden verse como una *hipótesis de clasificación de ejemplos* que puede obtenerse mediante un algoritmo de aprendizaje.

De forma suscinta: un Árbol de Decisión toma como entrada un ejemplo (un conjunto de atributos que describen un objeto o una situación dada), y devuelve una decisión sobre su pertenencia a una clase determinada.

Para poder realizar la clasificación, los Árboles de Decisión cuentan con la siguiente topología:

- Un nodo *hoja* o nodo respuesta. Contiene la salida (el nombre de la clase).
- Un nodo *interno* o nodo de decisión. Contien la prueba para un atributo y un conjunto de ramas para cada valor posible, las cuales conducen a otro árbol de decisión (que toma en cuenta sólo los atributos restantes).

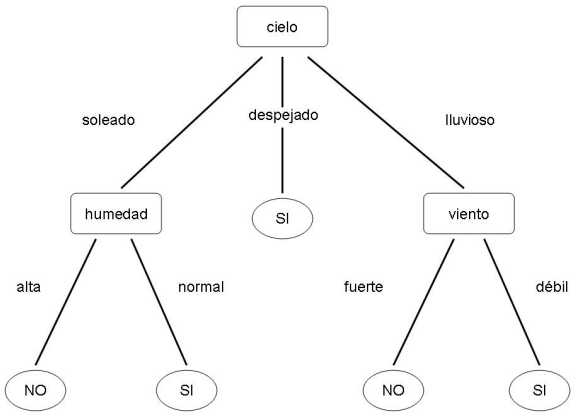


Figura 2.2: Árbol de decisión proposicional (Adaptado de [22]). Cada nodo representa una pregunta sobre el valor del atributo indicado, y dependiendo de tal valor, se toma la rama correspondiente, hasta llegar a una hoja, que contiene la respuesta (la clase), en este caso las etiquetas SI o NO.

En la figura 2.2.1 se muestra gráficamente un árbol de decisión, el cual fue obtenido mediante aprendizaje automático (ejemplo mostrado en el cap. 3 de la obra *Machine Learning* de T. Mitchell [22]).

Los Árboles de Decisión clasifican los ejemplares al realizar una serie de pruebas sobre los valores de sus atributos. Su mecanismo de operación es el siguiente: se toma un atributo del

ejemplar en cada vez, de acuerdo con el valor que ostente el atributo probado, el algoritmo opta por la rama que corresponda a su valor y evalúa el siguiente atributo. La clasificación se realiza repitiendo recursivamente estas pruebas sobre los atributos hasta encontrar un nodo respuesta que indica la clase a la que pertenece el atributo.

Para ilustrar esto, veamos cómo clasificaría el siguiente ejemplo (expresado proposicionalmente) el árbol mostrado en la figura 2.2.1:

Ej 1

$$\begin{array}{rl} \textit{Atributos}: & \textit{cielo} \quad \textit{temperatura} \quad \textit{humedad} \quad \textit{viento} \\ \textit{Valores}: & \langle \textit{soleado} \quad \textit{calor} \quad \textit{alta} \quad \textit{débil} \rangle \end{array}$$

De acuerdo con el árbol referido, en correspondencia con los valores de los atributos del ejemplo, para clasificar el ejemplo se seguiría una rama del árbol, lo que representa una cierta sucesión de pruebas. Ésta se muestra a continuación:

$$\begin{array}{rl} \textit{valor de cielo}: & \textit{soleado} \\ \textit{valor de humedad}: & \textit{alta} \\ \textit{Respuesta}: & \textit{NO} \end{array}$$

Entonces, la respuesta para este conjunto de atributos, es la clase NO. De igual manera, cualquier otra evidencia futura podrá ser clasificada de acuerdo con las pruebas de sus atributos establecidas en este árbol.

Como ya se dijo, la obtención de los Árboles de Decisión es un problema de aprendizaje que consiste en desarrollar un procedimiento que infiera un árbol de un conjunto de datos, para después utilizarlo en la tarea de clasificación [29, 22]. A continuación se muestra un algoritmo para inducir estos árboles.

2.2.2. ID3

Un algoritmo clásico para aprender Árboles de Decisión es **ID3** (propuesto por Ross Quinlan en 1986[22]), el cual utiliza una medida de información para guiar la búsqueda en

el espacio de AD posibles. El Algoritmo 1 muestra una versión de ID3 según lo expuesto en [22].

Algorithm 1 ID3(E:ejemplos, Clase, Atributos).
Recibe un conjunto E de ejemplos, un atributo objetivo o Clase, y una lista de Atributos sin la clase.
La función $mejorAtributo(E, A)$ implementa la función de entropía o alguna otra métrica de evaluación de los atributos.

```
Crear una Raíz (árbol con un solo nodo);
if Si todos los ejemplos son + then
  Regresa Raíz con etiqueta = +;
end if
if todos los ejemplos son − then
  Regresa Raiz con etiqueta = −;
end if
if Si Atributos es vacío then
  Regresa Raiz con etiqueta = valorMasComun(E, Clase);
else
  A ← mejorAtributo(E, A);
  Raiz ← A;
  while ∃v_i de A ∈ E do
    Añadir un nuevo subárbol Raiz (correspondiente a la prueba A = v_i);
    Hacer E_{v_i} (el subconjunto de E con el valor v_i);
    if E_{v_i} es vacío then
      Añadir una nueva hoja con etiqueta ← valorMasComun(E, Clase);
    else
      Añadir nuevoSubarbol ← ID3(E_vi, Clase, Atributos − A);
    end if
  end while
end if
Regresa Raiz
```

ID3 infiere el Árbol de Decisión formándolo desde la raíz hasta las hojas, de manera voraz, seleccionando los atributos que mejor separen los datos para incorporarlos a los nodos de prueba.

De esta forma, el atributo seleccionado parte los datos en subconjuntos, esto es, ejemplos positivos y negativos, creando una rama -subárbol- para cada valor posible del atributo en cuestión. El algoritmo determina cuál es el atributo que aporta mayor información de acuerdo con una medida, por ejemplo la *información esperada* o la *ganancia de información* [3, 22, 32], y lo sitúa en la raíz del árbol, iniciando el procedimiento de manera recursiva en cada uno de los subárboles. La idea es ir encontrando aquellos atributos que dividan mejor los datos con el objeto de generar un árbol de pocos niveles que sitúe en los primeros nodos

aquellos atributos que maximicen (o minimicen) la medida utilizada, tal es el sesgo inductivo de ID3 [22]. Los criterios para decidir el beneficio de un atributo, se determinan mediante una métrica, por ejemplo, **entropía** o la función de **Información Esperada** IE, a su vez definida a partir de la entropía entre dos variables. La entropía para valores positivos y negativos se expresa en la Def 7.

Def 7 ***Entropía***

$$I(x,y) = \begin{cases} si\ x = 0\ \ o\ \ y = 0, & 0 \\ \ de\ otra\ forma, & -\frac{x}{x+y} log \frac{x}{x+y} - \frac{y}{x+y} log \frac{y}{x+y} \end{cases} \tag{2.1}$$

La formula genérica de la entropía (para un mayor número de clases) se expresa como sigue:

$$I(E) = \sum_{i=1}^{k} -p(c_i, E)\, log\, p(c_i, E) \tag{2.2}$$

Donde: k es el número de clases, $p(c_i, E)$ es la proporción de ejemplos en el conjunto de entrenamiento E que pertenecen a la clase c_i.

Por su parte, la Información Esperada se expresa de la siguiente manera [32]:

$\forall a_i \in A$:

$$IE(a_i) \ = \ \sum_{j=1}^{V_i} \frac{p_{ij} + n_{ij}}{p + n} I(p_{ij}, n_{ij}) \tag{2.3}$$

Donde:

A = El conjunto de todos los atributos que describen un ejemplar

a_i = El i-ésimo atributo perteneciente A

V_i = Conjunto de posibles valores para el atributo a_i

p = Número de ejemplos positivos

n = Número de ejemplos negativos

p_{ij} = Número de ejemplos positivos con el valor v_{ij} del atributo a_i

n_{ij} = Número de ejemplos negativos con el valor v_{ij} del atributo a_i

Además de la Información Esperada, se han propuesto otras métricas, como la **Ganancia de Información** y la **Razón de Ganancia** (o *Radio de Ganancia*). La Ganancia de Información representa la reducción esperada en la entropía causada por el particionamiento de los datos de acuerdo con un atributo dado.

Una característica de esta métrica es su tendencia a favorecer atributos de muchos valores, de tal forma que éstos predominan sobre otros con pocos valores, pero cuyo beneficio es mayor. Para evitar esto, se emplea la Razón de Ganancia que evalúa la entropía del conjunto de entrenamiento con respecto a los valores de un atributo. A continuación se presenta la formulación matemática de estas dos métricas.

Def 8 ***Ganancia de información***. *La información ganada al realizar una prueba τ se expresa con la ecuación siguiente (caso binario):*

$$G = I(E^+, E^-) - (\frac{|E^+|}{|E|} + \frac{|E^-|}{|E|}) \tag{2.4}$$

Donde: E^+ y E^- son las particiones de ejemplos positivos y negativos, respectivamente, inducidas por τ en E.

Una forma alternativa de observar la predominancia de atributos de muchos valores se presenta cuando dos pruebas τ y τ' inducen particiones del conjunto de entrenamiento de la misma cardinalidad, pero con distintos beneficios. Como ya se mecionó, en estas circunstancias es útil utilizar el Radio de Ganancia, el cual se calcula como se expresa en la Def 9.

Def 9 ***Razón de Ganancia***. *La Razón de Ganancia o Radio de Ganancia se calcula a partir de la Ganancia Máxima:*

$$GM = \sum_{E_i \in \epsilon} \frac{|E_i|}{|E|} log \frac{|E_i|}{|E|} \tag{2.5}$$

Donde: ϵ es la partición en E inducida por τ.

Entonces, el Radio de Ganancia es el cociente de la Ganancia de información y la Ganancia Máxima:

$$RG = \frac{G}{GM} \tag{2.6}$$

Como se dijo al inicio de este apartado, los Árboles de Decisión expresan información proposicional, para explotar la representación en primer orden se ha propuesto su contraparte [3]: los árboles lógicos, los cuales son explicados a continuación.

2.2.3. Árboles Lógicos

Los árboles de decisión en primer orden o **Árboles Lógicos** son árboles binarios que representan una conjunción de literales. Cada nodo a su vez es una conjunción de literales y cada rama completa (hasta dar con una hoja), es la conjunción completa de pruebas que hay que hacer con la evidencia para encontrar el valor de la etiqueta. Pueden ser utilizados para realizar clasificación y regresión. Formalmente se definen como sigue [3]:

Def 10 ***Árbol Lógico****. Un árbol de decisión en primer orden, o ábol lógico, es un árbol de decisión binario constituido por dos elementos: nodos hoja y nodos de prueba, llamados simplemente* hojas *y* nodos *respectivamente. Éstos mantienen las siguientes particularidades:*

- *Cada nodo contiene una conjunción de literales.*
- *Distintos nodos pueden compartir variables únicamente por la rama izquierda.*

De esta manera, un Árbol Lógico tiene la siguiente morfología:

$$T = hoja(y) \quad Donde\ y\ =\ una\ etiqueta\ de\ respuesta.$$

$$T = nodo(conj, \{(verdadero, izq)\}, \{(falso, der)\})$$

El nodo representa la prueba a realizar sobre los datos. La rama izquierda se sigue cuando la conjunción conj *se hace verdadera. La rama derecha es relevante sólo cuando* conj *falla (por ello no comparte variables con los nodos que la anteceden).*

En la Figura 2.3 se muestra un ejemplo de Árbol Lógico, el cual es la contraparte del Árbol de Decisión presentado en la Figura 2.2. Como se puede observar, el Árbol Lógico es más compacto, pues es binario, y los nodos hoja sólo contienen valores de dos clases: positivo o negativo.

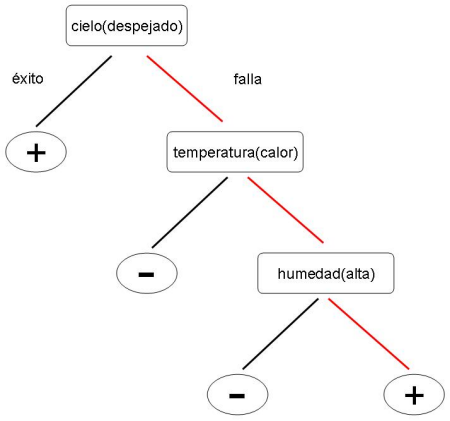


Figura 2.3: Árbol lógico. Cada nodo representa una conjunción de literales con variables existencialmente cuantificadas. Si la conjunción tiene éxito, se sigue la rama izquierda. En caso contrario se sigue la rama derecha sin compartir variables con los nodos de niveles superiores.

Al expresar en primer orden el ejemplo 1 (subsección 2.2.1, pág. 23) utilizando la sintáxis del lenguaje *Prolog*, se tiene lo siguiente:

Ej 2 { *cielo(soleado), temperatura(calor), humedad(alta), viento(débil).* }

Así, al utilizar el Árbol Lógico mostrado en la figura 2.2.3, la sucesión de pruebas indicadas por las literales del árbol sobre el ejemplo sería como se muestra enseguida:

$$cielo(despejado)? : \quad no,\ falla \Rightarrow\ ir\, por\, rama\, derecha$$
$$temperatura(calor)? : \quad si,\ éxito \Rightarrow\ clase : -$$

En este caso, la clase asignada tiene la etiqueta −, y se obtuvo tras probar las literales cielo(despejado) *y* temperatura(calor)*, siendo la primera falsa y la segunda verdadera de acuerdo con la evidencia.*

De esta manera, al utilizar Árbol Lógico, se consigue tener el poder expresivo de la Representación en Primer Orden y la facilidad de interpretación de los árboles de decisión.

Los Árboles Lógicos, pueden derivarse a partir de un conjunto de entrenamiento, siguiendo un procedimiento de aprendizaje semejante al utilizado para inducir árboles proposicionales, aumentándole las características necesarias para el uso de la Representación en Primer Orden. Lo anterior se conoce como un *escalamiento* del algoritmo de inducción (por ejemplo de ID3), tal es el caso del algoritmo TILDE [3], de cual se hablará en el capítulo siguiente.

En concordancia con ello, es importante señalar lo expuesto por Van Laer[33], en el sentido de la factibilidad de escalar algoritmos de aprendizaje proposicionales a primer orden. El atractivo de hacerlo radica en que estos algoritmos han sido trabajados por más tiempo, en comparación con los desarrollados en PLI, y tienen varias características en posibilidad de ser aprovechadas, tales como: la heurística empleada para guiar la búsqueda, los parámetros que requieren, la estrategia de búsqueda, etc. En el Capítulo 3 se hablará más extensamente de esta propuesta.

En la siguiente sección se menciona una configuración de PLI, que sirve de base para el escalamiento de algoritmos proposicionales a primer orden y el paso incremental en el aprendizaje.

2.3. Aprendizaje Inductivo Basado en Interpretaciones

La inducción de hipótesis a partir de evidencias ha sido tratada desde dos paradigmas, a saber: la Representación Proposicional y la Representación en Primer Orden[3]. Estas dos formas de expresar la información determinan las posibilidades del algoritmo de aprendizaje, como ya se dijo en la sección anterior. En este mismo sentido, la representación en primer orden da lugar a dos vertientes dentro de la Programación Lógica Inductiva: el **Aprendizaje Inductivo por Implicación** y el **Aprendizaje Inductivo Basado en Interpretaciones** (*Learning from Entailment* y *Learning from Interpretations*, en inglés).

Como se estableció en el Capítulo 1, el cometido de los sistemas en PLI puede verse como

la obtención de generalizaciones a partir de un Conjunto de Entrenamiento y conocimiento base relevante para el dominio en el que se realice el aprendizaje.

Es precisamente la forma en que se conciben el Conjunto de Entrenamiento y el Conocimiento Base lo que diferencia a las dos configuraciones de aprendizaje mencionadas. En cuanto al Aprendizaje Inductivo por Implicación, que es el paradigma más utilizado en Programación Lógica Inductiva [3], se orienta a la obtención de hipótesis partiendo de E y CB. Tomando como base la configuración normal del problema de Programación Lógica Inductiva mostrado en la Def 4 del capítulo precedente, el Aprendizaje Inductivo por Implicación se expresa formalmente como sigue [3]:

Def 11 ***Aprendizaje Inductivo por Implicación.***
A partir de:

- *Un conjunto de ejemplos positivos* E^+
- *Un conjunto de ejemplos negativos* E^-
- *Conocimiento Base (CB)*
- *Un lenguaje de primer orden:* $L \subseteq$ *Prolog*

Encontrar *una hipótesis* $H \subseteq L$ *tal que:*

- $\forall e \in E^+ : \ H \wedge B \models e$, *y*
- $\forall e \in E^- : \ H \wedge B \not\models e$

Lo importante a resaltar de esta configuración es el uso de la totalidad de los ejemplos E junto con el CB para definir la hipótesis. Sin embargo, no toda la información contenida en E es relevante y no está definida cuál parte si lo es, por lo que es necesario buscarla en todo el universo de datos, lo que origina un alto costo computacional [3]. Desde una perspectiva de implementación, tanto E como CB pueden verse como un sólo programa en Prolog, donde cada ejemplo es un hecho.

Por otro lado, el Aprendizaje Inductivo Basado en Interpretaciones es una configuración alternativa, que contrariamente al Aprendizaje Inductivo por Implicación, toma como base la noción de que la información relevante para cada ejemplo está localizada, sólo en una parte de los datos, por lo que no es necesario considerar todo el conjunto. De esta forma, se asume que que cada ejemplo es *independiente* de los demás, y proporciona la información necesaria para aprender un concepto, teniendo como consecuencia la imposibilidad de aprender definiciones recursivas. Esto se conoce como el *supuesto de localidad*, el cual se enuncia a continuación:

Def 12 ***Supuesto de localidad****. Toda la información relevante para un solo ejemplo está contenida en una pequeña parte de la base de datos.*

En el Aprendizaje Inductivo Basado en Interpretaciones, el Conocimiento Base se representa como un programa en Prolog y cada ejemplo $e \in E$, es representado por un programa en Prolog separado que incluye una etiqueta $c \in Clases\,(+,-)$. Cada ejemplo expresa un conjunto de hechos fundamentados, esto es: un Modelo mínimo de Herbrand, para el cual se cumple $e \wedge CB$, esto se denomina **interpretación** [3].

Formalmente, el Aprendizaje Inductivo Basado en Interpretaciones se define así [3, 12, 4]:

Def 13 ***Aprendizaje Inductivo Basado en Interpretaciones****.*
A partir de*:*

- *Una variable objetivo C*
- *Un conjunto E de ejemplos etiquetados con un valor $c \in C$*
- *Conocimiento Base (CB)*
- *Un lenguaje $L \subseteq$ Prolog*

Encontrar*: una hipótesis $H \in L$ tal que para todo ejemplo $(e, c) \in E$:*

- $H \wedge e \wedge B \models etiqueta(c)$, *y*
- $\forall\, c' \neq c : H \wedge e \wedge B \not\models etiqueta(c)$.

En cuanto a esta configuración de aprendizaje, se señalan las siguientes ventajas [3]:

- La información contenida en los ejemplos es separada del conocimiento base.
- Explota el supuesto de localidad, la información de los ejemplos es independiente entre sí.
- Consecuencia del punto anterior es considerar que los ejemplos provienen de una población (son una muestra) y no agotan la descripción de ella, por lo que debe tomarse en cuenta ruido y valores faltantes. Esto conduce a las siguientes proposiciones:
  - La descripción de los ejemplos es *cerrada*, autocontenida.
  - La descripción de la población es *abierta*, en presencia de más evidencia, se tendrá mayor conocimiento.

En la Tabla 2.3 se sintetizan las diferencias entre los tipos de aprendizaje que se han mencionado hasta el momento: Proposicional, Inductivo por Implicación e Inductivo Basado en Interpretaciones.

Paradigma	**Información**	**Supuesto de localidad**	**Descripción de** $e \in E$
Proposicional	$\{V_i, \ldots, V_n\}$ Donde: $V_x = [valor\_de\_atributo, Etiqueta]$.	SI	CERRADA
AIBI	$\{\{Interpret\ 1\}, \ldots, \{Interpret\ n\}\}$, $\{ConocimientoBase\}\}$	SI	CERRADA
AII	$\{E^+, E^-, ConocimientoBase\}$	NO	ABIERTA

Tabla 2.3: Diferencias en los tipos de aprendizaje, proposicional, por interpretaciones (AIBI) y por implicación (AII) [3]. Los dos primeros comparten el supuesto de localidad y asumen una descripción abierta de los datos.

En lo que respecta al supuesto de localidad y a la apertura en cuanto a la descripción de la población origen de los datos, el Aprendizaje Inductivo Basado en Interpretaciones puede considerarse como una configuración intermedia entre los aprendizajes proposicional y por implicación, aunque con la limitante de no poder aprender definiciones recursivas, sin

embargo, se ha dicho que éstas no tienen una presencia preponderante en las aplicaciones prácticas [3]. En consecuencia, utilizarlo pone al alcance del algoritmo de aprendizaje el poder expresivo de la Representación en Primer Orden y la apertura de la información, que conduce, como se verá en la siguiente sección, a poder implementar algoritmos incrementales, pues se acepta la futura presencia de nuevas evidencias. La Figura 2.4 muestra gráficamente esta situación.

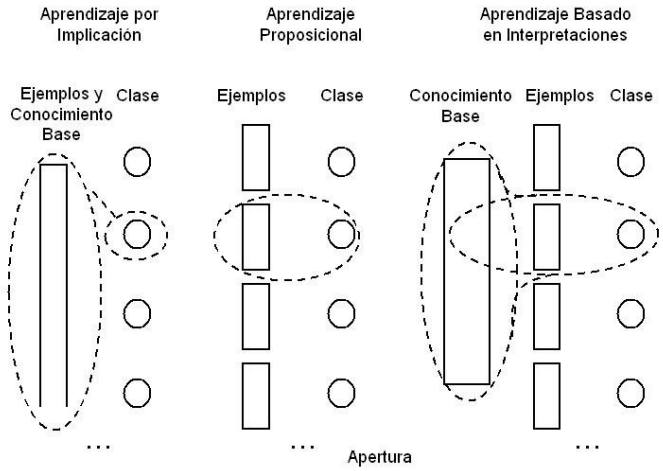


Figura 2.4: Comparación gráfica entre los aprendizajes proposicional (atribuo-valor), por implicación y basado en interpretaciones, con respecto al supuesto de localidad y la apertura de la descripción de la población de los datos. Adaptado de [3].

En la siguiente sección se aborda el tema de la incrementalidad en el aprendizaje automático, la cual se deriva de situaciones en las que no se tiene un conjunto completo de ejemplos de aprendizaje.

2.4. Aprendizaje Incremental

En las secciones precedentes (2.2.1 y 2.2.3) se ha hablado de aprendizaje de conceptos, particularmente mediante Árboles de Decisión, los cuales parten de un conjunto "completo" de datos, es decir, un conjunto estático. Sin embargo, cuando se requiere que un sistema lleve

a cabo tareas que necesitan aprender de forma serial o dinámica −esto es cuando los ejemplos se presentan de manera secuencial, como un flujo, o están distribuidas en varios depósitos y no como un conjunto de tamaño fijo− se necesita revisar la hipótesis aprendida en presencia de nuevos ejemplos, en lugar de rehacerla cada vez que se tienen datos nuevos [32]. Algunos ejemplos de tareas que necesitan de *aprendizaje incremental* son: Bioinformática, Minería de Datos, Sistemas Multi-Agente e Interfases Inteligentes. En las dos primeras la cantidad de datos es tan grande que se hace necesario almacenarla en bases de datos distribuidas en varios sitios y que difícilmente podrían reunisre en un solo punto. En tanto que una interfaz inteligente recibe las información como un flujo discontinuo (a medida que el usuario ocupa la interfaz) y generalmente es escasa. En el caso de los Sistemas Multi-Agente el aprendizaje tiene que ver con la adaptación a un ambiente cambiante, donde las evidencias se presentan como un flujo; un caso interesante se da cuando se intenta la ejecución coordinada de una tarea, donde cada agente debe intercambiar información con los otros y actuar en concordancia, configurando un aprendizaje coordinado de naturaleza incremental [11].

Para tratar el problema de aprendizaje a partir de un conjunto creciente de evidencias, se han desarrollado **algoritmos incrementales**, los cuales revisan el concepto aprendido al recibir nuevos ejemplos de forma eficiente. Un caso intersante es el diseño de algoritmos incrementales en primer orden, lo que puede obtenerse al realizar el escalamiento de algoritmos incrementales proposicionales a sus versiones relacionales, materia de este trabajo. Ejemplos de algoritmos incrementales son Eliminación de Candidatos, ID4 e ID5r [32, 22]. A continuación se revisan brevemente las características principales de los algoritmos mencionados.

2.4.1. El Espacio de Versiones y Eliminación de Candidatos

El algoritmo **Eliminación de Candidatos** [22] utiliza la es importante para el trabajo de esta tesis puesto que presenta dos características principales, a saber:

- Su naturaleza incremental, las hipótesis son revisada a medida que se presentan nue-

vos ejemplos.

- El uso de la estructura del espacio de hipótesis a través de la relación $\geq_g$.

Si bien eliminación de candidatos usa la representación proposicional, las ideas mencionadas pueden extrapolarse para las versiones en primer orden que se presentan en el capítulo siguiente. Antes de mostrar el algoritmo hay que atender a dos definiciones básicas para su entendimiento [22]:

- **Hipótesis consistente**. Una hipótesis es consistente con los ejemplos de entrenamiento E si los clasifica correctamente, es decir, si y sólo si: $h(x) = c(x)$, para cada ejemplo $\langle x, c(x)\rangle$ en E. Formalmente: $consistente(h, E) \equiv (\forall\langle x, c(x)\rangle \in E)h(x) = c(x)$. Nótese la importancia del concepto objetivo $c(x)$ para que un ejemplo sea consistente con una hipótesis.

- **El espacio de versiones**. Es el conjunto H de hipótesis consistentes con E, esto es: $VS_{H,E} \equiv \{h \in H | consistente(h, E)\}$.

- **Frontera en lo general**. Es el conjunto G de hipótesis más generales de H consistentes con E:
  $G \equiv \{g \in H | consistente(g, E) \wedge (\nexists g' \in H)[(g' >_g g) \wedge consistente(g', E)]\}$.

- **Frontera en lo específico**. En contraparte, S es el conjunto de hipótesis más específicas de H consistentes con E:
  $G \equiv \{s \in H | consistente(s, E) \wedge (\nexists s' \in H)[(s >_g s') \wedge consistente(s', E)]\}$.

A partir de estas nociones, particularmente de los conjuntos S y G, se consigue especificar un conjunto parcialmente ordenado cuyos elementos son las hipótesis pertenecientes a las fronteras más las que están entre ellas. Esto se define mediante el siguiente teorema:

Teorema 1 ***Espacio de Versiones***.
Sean X un conjunto arbitrario de ejemplos, H un conjunto de hipótesis booleanas definido sobre X, $c : X \to \{0, 1\}$ un concepto objetivo arbitrario sobre X, y E un conjunto arbitrario

de ejemplos de entrenamiento $\{\langle x, c(x)\rangle\}$. *Para todo* X, H, c *y* E, *tal que las fronteras* S *y* G *están bien definidas:*

$$VS_{H,E} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s)\}$$

El teorema enunciado anteriormente permite entender el algoritmo de eliminación de candidatos. Este algoritmo determina el espacio de versiones que contiene a aquellas hipótesis de H consistentes con una secuencia de evidencias. El pseudocódigo de eliminación de candidatos [22] se muestra en el Algoritmo 2.A continuación se describe brevemente su funcionamiento: Inicializa S y G (con las hipótesis más específica y la más general), y conforme a los ejemplos que van acaeciendo (incrementalmente), realiza operaciones de especialización sobre G y de generalización sobre S, de tal forma que al procesar los ejemplos, se cuenta con un espacio de versiones que contiene todas las hipótesis consistentes con los ejemplos.

Hay que destacar que este algoritmo converge al concepto objetivo cuando se encuentra que $S = G$, es decir, las fronteras en lo general y en lo específico delimitan solamente las hipótesis consistentes con los ejemplos, de tal forma que un concepto bien aprendido contendrá una sola hipótesis, de otra manera se considera que no se ha aprendido completamente. Empero, aún con un concepto aprendido parcialmente es posible clasificar nuevas evidencias de forma acertada de la siguiente manera: probando su consistencia con las hipótesis de S y de G, así, un ejemplo se clasificará como positivo si satisface cada hipótesis en S y G, y será tomado como negativo si no satisface ninguna de G, (puesto que son las más generales) [22]. Los casos donde no todas las hipótesis se satisfacen, se resuelven por voto mayoritario, lo que baja el nivel de certidumbre, pero permite clasificar una evidencia con cierto margen de confiabilidad.

Finalmente, se mencionan las siguientes limitaciones: a) no habrá convergencia cuando exista ruido en los datos, pues se eliminará -indebidamente- la hipótesis correcta al generalizar o especializar S y G; b) cuando el concepto objetivo no sea expresable en el universo de hipótesis considerado (nuevamente, hay que recordar que no se puede aprender si no se está en posibilidad de expresar el objeto del aprendizaje, de ahí la necesidad de tener mejores

Algorithm 2 eliminacionDeCandidatos(E: conjunto de entrenamiento).
Recibe un conjunto de entrenamiento E. Regresa ev, un conjunto de hipótesis consistentes con E.
El conjunto ev se determina sin enumeración explícita. Las hipótesis consideradas son revisadas incrementalmente, conforme se obtienen las nuevas evidencias, aprovechando la relación de generalidad $\geq_g$.

$G \leftarrow hipotesis\_mas\_general$;
$S \leftarrow hipotesis\_mas\_especifica$;
while $\exists\, x \in E$ **do**
  if $c(x) = 1$ **then**
    %El ejemplo es POSITIVO
    Remover de G todas las hipótesis $h(x) = 0$ (inconsistentes con x);
    while $\exists\, s \in S$ tal que $s(x) = 0$ **do**
      $S \leftarrow S \setminus \{s\}$;
      $S \leftarrow S \cup \{generalizaciones\ minimas\ de\ s\ consistentes\ con\ E\}$,
      donde $\exists g \in G \mid g >_g s$;
      $S \leftarrow S \setminus \{s\}$ si $(\exists s' \in S) s >_g s'$;
    end while
  end if
  if $c(x) = 0$ **then**
    %el ejemplo es NEGATIVO
    Remover de S todas las hipótesis $h(x) = 1$ (inconsistentes con x);
    while $\exists g \in G$ tal que $g(x) = 1$ **do**
      $G \leftarrow G \setminus \{g\}$;
      $G \leftarrow G \cup \{especializaciones\ minimas\ de\ g\ consistentes\ con\ E\}$,
      donde $\exists s \in S \mid s <_g g$;
      $G \leftarrow G \setminus \{g\}$ si $(\exists g' \in G) g <_g g'$;
    end while
  end if
end while
Regresa $ev \leftarrow \{G, S\}$;

formas de representar la información).

A continuación se hablará de dos propuestas de algoritmo incrementales para inducir árboles de decisión, por ser fundamentales para el desarrollo de este estudio.

2.4.2. Algoritmos ID4 e ID5r

Los algoritmos ID4 e ID5r representan las versiones incrementales del algoritmo ID3. Para lograr el aprendizaje incremental modifican la estructura de los Árboles de Decisión, incorporándole mayor información a los nodos, con el objetivo de contar con elementos para calcular el beneficio de los atributos presentes en los datos, y con ello mantener actualizada la estructura.

Cada nodo del Árbol de Decisión contiene información sobre el número de ejemplos positivos y negativos para cada atributo de ese nodo. De esta forma se puede calcular la métrica con que se decidirá qué atributo se prueba en cada nodo.

Las estructuras de la hipótesis sufren una modificación para contender con la incrementalidad. En el caso de ID5r, el árbol puede tener dos variantes: A) **Árbol expandido** (isomorfo al que se ha venido manejando hasta ahora, pero con la lista de postivos y negativos) y B) **Árbol contraído**, que es la lista de pares $atributo - valor$ de los atributos restantes (los que no son utilizados para probar el ejemplo). También se menciona que el árbol contraído puede expanderse para regresar la estructura a la forma convencional, con lo que se conserva la hipótesis en la forma que se ha venido planteando.

Así, en un árbol expandido, un nodo puede ser de **decisión** cuando tiene un atributo prueba, o de **respuesta**, cuando tiene la clase a la que pertenecería el ejemplo, a saber: Positiva o Negativa.

A continuación se mencionan brevemente los algoritmos ID4 e ID5r, que como sus nombres lo indican, están basados en el algoritmo no incremental ID3:

- **ID4**. Toma un ejemplo y actualiza el Árbol de Decisión: Actualiza las cuentas (positivas y negativas) de cada atributo; verifica que el atributo de prueba sea el más adecuado (con respecto a criterio de la mínima entropía o máxima ganancia de información) y si es el caso, éste permanece, si no, es substituido y se descartan los subárboles que dependen del nodo. Este procedimiento se sigue recursivamente para cada nodo (y subárbol), expandiendo en su caso las ramas de los atributos que así lo requieran. ID4 se presenta en el Algoritmo 3. Limitación: Este algoritmo no puede aprender algunos conceptos que ID3 aprendería (sólo logra ésto cuando hay un atributo en cada nodo de decisión que es la mejor opción sin disputa sobre los otros [32]), esto se debe a que descarta los subárboles que ya construyó.

- **ID5r**. Está basado en ID4, su principal diferencia está en que introduce un proceso recursivo de *reconstrucción* del árbol para preservar la consistencia, en lugar de descartar

los subárboles. ID5r se muestra en el Algoritmo 4

Algorithm 3 ID4(e: ejemplo de entrenamiento,A: atributos).

Recibe un ejemplo de entrenamiento e y la lista de atributos que contendrán los ejemplos. Regresa un Árbol de Decisión construído incrementalmente.
En cada nodo se mantiene información para calcular la bondad de los atributos: las cuentas + y − para cada atributo, por lo que no es necesario consultar las evidencias pasadas.

```
while ∃ ejemplos nuevos e do
  while ∃ atributos a en nodo actual do
    Actualizar las cuentas + y − para el valor de a en e;
    if todas las cuentas en el nodo actual son + (o −) then
      Convertir nodo actual en nodo de decisión con la etiqueta + (o −);
    else
      if nodo actual es un nodo hoja then
        Convertirlo en nodo de decisión con el atributo de mayor beneficio indicado por
        mejorAtributo(A, nodoActual);
      else
        if el nodo de decisión actual contiene un atributo a tal que
        (∃b ∈ A) b ≠ a | b = mejorAtributo(A, nodoActual) then
          a ← b;
        end if
      end if
      Descartar los subárboles del nodo modificado
    end if
    Actualizar recursivamente los subárboles del nodo modificado pertenecientes a la rama indicada por
    el valor de a en e. Crear subárboles si es necesario;
  end while
end while
Regresar árbol A actualizado
```

El algoritmo ID5r, obtiene los mismos resultados que el ID3, pero a menor costo computacional, ya que no tiene que reconstruir el árbol desde el inicio cada vez que cambian los datos [32], sino que lo reconstruye a medida que recibe nuevas evidencias en función de la métrica de evaluación usada. Lo que lo hace un algoritmo muy adecuado para clasificar datos alojados en bases de datos distribuidas.

Una característica a resaltar de estos algoritmo es el uso de clases binarias (+ y −) por presentar un paralelismo con los algoritmos de ILP como ya se ha comentado anteriormente. Esto es favorable para realizar aprendizaje incremental en primer orden. El capítulo que sigue, dedicado a la implementación se expondrá una propuesta para conseguir un algoritmo incremental en primer orden que hace uso de las ideas expuestas en el presente capítulo: representación relacional, sesgo inductivo, árboles lógicos e incrementalidad.

Algorithm 4 ID5r(e: ejemplo de entrenamiento, A: atributos).

Recibe un ejemplo de entrenamiento e y la lista de atributos posibles de los ejemplos. Regresa un Árbol de Decisión construído incrementalmente. El árbol puede ser expandido o contraído y almacena la información para calcular la bondad de los atributos, esto es: a) las cuentas + y − para cada atributo; b) los ejemplos que cada nodo cubre. De manera que no es necesario consultar las evidencias pasadas.

El procedimiento *reestructa* coloca el mejor atributo en la raíz del árbol, modificando la estructura de los subárboles correspondientes, inviitiéndolos o cambiándolos de nivel.

while ∃ ejemplos nuevos e **do**
  if $vacio(A) = true$ **then**
    $clase \leftarrow tomaClase(e)$;
    Añadir e a A;
  else
    $c \leftarrow claseArbol(A)$;
    $c' \leftarrow tomaClase(e)$;
    if $(arbolContraido(A) = true)$ y $(c = c')$ **then**
      Añadir e a A;
    else
      if $arbolContraido(A) = true$ **then**
        Expandir A un nivel;
        $at \leftarrow sacaAtributo(e)$; /*Toma un atributo aleatoriamente de e */
        $raiz \leftarrow at$;
      end if
      $actualizaEstadisticas(A, e)$;
      $nodoA \leftarrow nodoActual(A)$;
      $atrMax \leftarrow mejorAtributo(A, nodoA)$; /*Devuelve el atributo con la mejor evaluacion de bondad en el nodo actual*/
      $atrPrueba \leftarrow tomaRaiz(nodoA)$;
      if $atrPrueba \neq atrMax$ **then**
        $nodoA \leftarrow reestructura(nodoA)$; /*Pone como atributo del nodo aquel con la mejor evaluacion de bondad*/
        $mejorA \leftarrow tomaRaiz(nodoA)$;
        $subarbolPrueba \leftarrow subarbol(nodoA, mejorA)$;
        $subArbs \leftarrow \{subarboles(nodoA)\} \setminus \{subarbolPrueba\}$; /*Asigna los subarboles cuya raiz no es el atributo de prueba*/
        Aplicar *reestructura* a $subArbs$; /*Reestructura recursivamente los subarboles del nodo actual*/
      end if
      $reestructura(nodoA, subArbolPrueba)$
    end if
  end if
end while
Regresar árbol A actualizado

Capítulo 3

Metodología

Los conceptos mencionados en el capítulo precedente acerca de la representación del conjunto de entrenamiento y la hipótesis, además del aprendizaje incremental, sirven de base para la metodología de escalamiento de algoritmos de aprendizaje inductivo. A continuación se hace una breve recapitulación de algunos de estos conceptos.

En primer término se comentó que el uso de la representación en primer orden (relacional) hace posible que los algoritmos de aprendizaje deriven conceptos que no son accesibles utilizando la representación proposicional, tal es el caso de las relaciones entre múltiples objetos. Existen dos formas en las que se puede explotar la representación relacional:

- Adoptar hipótesis en forma de árboles lógicos, cuya conveniencia se manifiesta en su expresividad (disyunciones de expresiones conjuntivas en primer orden) y, en segundo término, su adecuación a la información con la que los agentes BDI realizan su aprendizaje (ver sección 4.2).

- Emplear interpretaciones para representar el conjunto de entrenamiento, porque posibilitan el aprendizaje incremental debido a que asumen el supuesto de localidad, tomando los datos como un flujo de ejemplos cuyo tamaño no está determinado previamente.

Por otro lado, se destacó el uso de un sesgo inductivo como estrategia de búsqueda que facilita el proceso de aprendizaje, particularmente la definición de especificaciones de *rmodes*, las cuales, tienen la conveniencia de aprovechar la estructuración del espacio de búsqueda mediante θ-subsunción como relación de generalidad. Finalmente, se mencionaron algunos algoritmos de aprendizaje incrementales y no incrementales tales como ID3, ID4 e ID5r, mismos que sirven de base para el algoritmo propuesto en este estudio.

En la primera parte de este capítulo se expone la metodología de escalamiento que permite obtener algoritmos de aprendizaje incremental en primer orden mediante el cambio de representación o el paso incremental, según lo requieran los algoritmos a escalar.

En la segunda parte, se explica la implementación de la metodología y la obtención de ILDT como algoritmo incremental en primer orden, cuyo escalamiento puede sintetizarse en dos puntos principales: a) Tomar como base la representación en primer orden y el sesgo de lenguaje de Tilde y b) Seguir la estrategia incremental de ID4 adecuándola a la nueva representación. Configurando así, la siguiente línea de escalamiento: $Tilde, ID4 \rightarrow ILDT$ (cfr. con la información de la Tabla 3.1, pág. 44).

3.1. Escalamiento

Cuando se realiza el escalamiento de un algoritmo de aprendizaje inductivo proposicional a su versión en primer orden, es necesario optar por una configuración adecuada. En Programación Lógica Inductiva existen dos configuraciones posibles: *Aprendizaje Inductivo por Implicación* y *Aprendizaje Inductivo Basado en Interpretaciones* (AIBI). El segundo es el más adecuado para escalar algoritmos proposicionales a primer orden [3]. Esto se debe principalmente a las similitudes existentes en cuanto al supuesto de localidad y en cuanto a la apertura (véase la Def 12, en la pág. 31 del Capítulo 2), lo que hace factible convertir algoritmos proposicionales a primer orden, sin afectar significativamente la estrategia original.

Los autores Van Laer y De Raedt [33] proponen un esquema para realizar este escalamiento sobre algoritmos de clasificación que tiene como ventaja, además de añadir la expresividad de los algoritmos relacionales, aprovechar la investigación realizada en algoritmos proposicionales e incluir parte de su eficiencia y su efectividad. Aunado a lo anterior, los autores mencionan que se obtiene un vínculo entre los dos algoritmos, llegando incluso a resultados idénticos si se prueban sobre los mismos datos.

La propuesta de Van Laer y De Raedt consiste en los siguientes pasos:

1. Identificar el algoritmo proposicional incremental que se ajuste a la tarea. Existen varios clasificadores proposicionales incrementales y no incrementales susceptibles de ser escalados a primer orden, puesto que algunos de los no incrementales tienen versiones incrementales que permiten realizar el escalamiento. En la Tabla 3.1 se muestran algunos ejemplos.

2. Usar interpretaciones para representar los datos (y aplicar AIBI posteriormente). Como ya se indicó, las representaciones constan de dos componentes: el conocimiento base CB y los ejemplos e. Cada miembro del conjunto e se representa por un programa en Prolog y tiene indicada la clase a la que pertenece la instancia. De igual manera, B es un programa en Prolog. Lo mismo sucede con la teoría que se pretende inducir. Así, se puede aprovechar el potencial de este lenguaje para operar con representaciones en primer orden.

3. Escalar la representación proposicional reemplazando las pruebas $atributo - valor$ por literales en primer orden y modificar la *cobertura* de éstas (cuántos ejemplos representa -define como + o −). En este caso, se pueden usar árboles lógicos para hacer la representación de la hipótesis buscada.

4. Usar el operador θ-subsunción como marco de generalidad, implementando también operadores para guiar la búsqueda, ya sea de *generalización* o de *especialización*, en el espacio de hipótesis que tengan correspondencia con el que utiliza el algoritmo proposicional que se está escalando. Además, se debe emplear un mecanismo de sesgo declarativo para limitar el espacio de búsqueda. Esto es, limitar el número de cláusulas contempladas por el operador θ-subsunción.

5. Implementar el algoritmo escalado. Se deben preservar en la medida de lo posible las características del algoritmo original: estructura de la búsqueda, heurística, manejo de ruido, podas, parámetros, etc.

6. Evaluar el algoritmo escalado. En principio, los resultados deben ser compatibles con

los resultados del algoritmo original para los mismos datos. Evaluarlo también para datos relacionales.

7. Añadir características extras.

Por último, se menciona que esta metodología incluye dos casos:

1. Tomar un clasificador incremental proposicional y escalarlo a primer orden.

2. Tomar un clasificador no incremental en primer orden y escalarlo a su versión incremental.

En la Tabla 3.1 se presentan algunos algoritmos de clasificación con sus escalamientos a primer orden y el paso incremental.

	Algoritmo base	**Algoritmos escalados**	
1	ID3 $\rightarrow_{Incr}$	ID4,ID5r	$\rightarrow_{RPO}$ ID4 e ID5r basado en árboles lógicos
2	AQ,CN2 $\rightarrow_{RPO}$	ICL	$\rightarrow_{Incr}$ FOIL
3	C4.5 $\rightarrow_{RPO}$	Tilde	$\rightarrow_{Incr}$ TildeTG

Tabla 3.1: Una posible sucesión de clasificadores con respecto a los escalamientos incremental y en primer orden.

El primer grupo de algoritmos mostrados en la Tabla 3.1, representa una posible ruta de escalamiento seguida para construcción de hipótesis a partir de datos, tanto en forma de árboles proposicionales como de árboles lógicos. El algoritmo base es ID3, el cual construye árboles proposicionales de forma no incremental. Por su parte el algoritmo ID4 construye árboles proposicionales incrementalmente, pero tiene como inconveniente que la distribución de los datos de ejemplo influye en su posibilidad de construir una hipótesis adecuada [32], de esta manera, si se presentan los mismos datos con un ordenamiento distinto, el algoritmo puede construir distintas hipótesis, dando como consecuencia que algunos conceptos no sean aprendidos, es decir obtiene árboles sin la cobertura adecuada, sin embargo, se conoce como el primer paso para obtener incrementalidad; una mejora a ID4 es ID5r, el cual implementa

un proceso de reestructuración del árbol y además conserva en su estructura los ejemplos que son cubiertos por cada nodo, con esto, ID5r puede determinar cuál nodo debe situarse en la raíz del árbol y en los subárboles, a fin de aumentar la cobertura de los ejemplos. Es así como se consigue la incrementalidad en el caso de aprendizaje proposicional de árboles a partir de datos. El paso a primer orden en este grupo implica una implementación de los algoritmos ID4 e ID5r utilizando árboles lógicos y θ-subsunción.

Por otro lado, el segundo grupo, que construye hipótesis en forma de listas de decisión, sigue una ruta análoga según se expone en [33], partiendo de los algoritmos proposicionales AQ o CN2, para obtener una versión en primer orden: ICL; por último, el escalamiento va hacia el paso incremental al crear una versión incremental como FOIL.

Finalmente, el tercer grupo parte del algoritmo C4.5 (que representa una mejora de ID3), sigue su evolución a primer orden con Tilde, algoritmo capaz de construir árboles lógicos de decisión y de regresión [3, 9], del cual existe una versión incremental para construir éste último tipo de árboles: el algoritmo TG.

Aquí vale la pena señalar que los escalamientos 1 y 3 mostrados en la Tabla 3.1 representan formas opcionales de obtener un algoritmo incremental en primer orden, puesto que las líneas de escalamiento convergen en un algoritmo incremental de inducción de árboles lógicos, sólo que el grupo 1 va de lo incremental al primer orden, mientras que el grupo 2 parte de un algoritmo en primer orden y realiza el paso incremental.

En la siguiente sección se presenta la implementación de la propuesta de esta tesis, la cual parte de las líneas de escalamiento 1 y 2, y tiene como fin la obtención de un algoritmo incremental de inducción de árboles lógicos de decisión a partir de datos, tomando como base los algoritmos Tilde, TG e ID4.

3.2. Implementación

En este apartado se muestra la implementación de la metodología. El lenguaje utilizado para la programación de los algoritmos es Prolog, en la distribución de *SWI-Prolog*. El uso de

Prolog se debe a su capacidad en el manejo de expresiones en primer orden y a que permite determinar la veracidad de una expresión, con lo que se puede verificar si una hipótesis cubre un conjunto de evidencias. Esto lo hace muy atractivo para la construcción de algoritmos de aprendizaje en primer orden, como los tratados en esta tesis. Además el Aprendizaje Inductivo Basado en Interpretaciones por definición actúa con un lenguaje en primer orden que es un subconjunto de Prolog.

En la primera etapa de la implementación se optó por el segundo caso aplicable según la metodología, esto es:

> "tomar un clasificador no incremental en primer orden y escalarlo a su versión incremental"

Para ello se partió del algortimo Tilde (de *Top-Down Induction of First Order Logical Decision Trees*) para efectuar el escalamiento a su versión incremental. Tilde hace una búsqueda en profundidad y utiliza interpretaciones para expresar el conjunto de entrenamiento y árboles lógicos para representar la hipótesis.

Como primera parte, se programó un algoritmo de clasificación en primer orden que usa una hipótesis (árbol lógico) ya construida [3]. La clasificación se consigue al descender en el árbol por las ramas derecha e izquierda según tenga éxito o falle la conjunción asociada (la del nodo raíz junto con la que cada nodo aporta). El procedimiento sigue hasta encontrar las hojas del árbol, las cuales proporcionan finalmente la clasificación del ejemplo. El clasificador implementado se muestra en el Algoritmo 5.

Una vez que se tuvo el algoritmo para utilizar las hipótesis obtenidas, se procedió a implementar el algoritmo para construir tales hipótesis: Tilde.

Tilde construye un árbol lógico a partir de un conjunto de datos, tomando aquellas literales que tengan una mayor ganancia para situarlas en cada nodo. Bifurcando el árbol en dos ramas: una para aquellos ejemplos donde las literales del nodo actual se cumplen (izquierda) y el otro para las que no se cumplen (derecha).

Hay que recordar que el árbol lógico tiene en cada nodo una conjunción de literales y a

Algorithm 5 clasifica(e: ejemplo, BC: conocimiento base).
Recibe e: ejemplo. Regresa c: clase.
Este algoritmo clasifica el ejemplo e basándose en una hipótesis en forma de árbol lógico.

```
Q ← true;
N ← raiz;
while N ≠ leaf(c) do
  N ← inode(conj, left, right);
  if Q ∧ conj se verifica con e ∧ BC then
    Q ← Q ∧ conj;
    N ← left;
  else
    N ← right;
  end if
end while
Regresa c;
```

esta conjunción se añaden las que están en los niveles inferiores hasta llegar a las hojas (conjunción asociada). De esta manera, la tarea del algoritmo es obtener la conjunción asociada, añadiendo una literal -o nodo- en cada nivel.

Algorithm 6 TILDE(T: árbol, E: conjunto de ejemplos) /
construyeArbol(T: árbol, E: conjunto de ejemplos, Q: consulta).
TILDE. Recibe T: true (hipótesis más general), E: conjunto de ejemplos de entrenamiento. B: conocimiento base. Regresa T, el árbol lógico aprendido.
construyeArbol. Recibe T: árbol inicial, E: partición del conjunto de ejemplos. Regresa T, el árbol lógico modificado.
Estos algoritmos construyen árboles lógicos a partir de datos de entrenamiento y conocimiento base (CB).

```
procedimiento tilde(T: árbol, E: conjunto de ejemplos)
    construyeArbol(T, E, true);
procedimiento construyeArbol(T: árbol, E: conjunto de ejemplos, Q: consulta)
← Q_b := elemento de ρ(← Q) con la mayor ganancia (o razón de ganancia);
if ← Q_b no aporta beneficio then
  T ← leaf(claseMayoritaria(E));
else
  conj ← Q_b ← Q;
  E1 ← {e ∈ E | ← Q_b se verifica con e ∧ CB};
  E2 ← {e ∈ E| ← Q_b falla con e ∧ CB};
  construyeArbol(left, E1, Q_b);
  construyeArbol(right, E2, Q);
  T ← inode(conj, left, right);
end if
```

En el Algoritmo 6 se muestran Tilde y el procedimiento de construcción del árbol: construyeArbol.

Para determinar las literales que integrarán la conjunción asociada, se revisan varias candidatas, las cuales son obtenidas mediante un operador de refinamiento de la literal actual,

dicho operador es la función ρ, tal como se muestra en la primera línea del algoritmo 6. En la siguiente sección se expone con mayor claridad la implementación de este operador.

3.2.1. Operador de Refinamiento y RModes

El **Operador de Refinamiento** ρ recibe una conjunción de literales y regresa el conjunto de conjunciones posibles, resultado de la adición de literales bajo θ-subsunción. Es una forma de implementar el sesgo inductivo para orientar la búsqueda en el aprendizaje en primer orden.

Este operador guía la búsqueda de la hipótesis (en forma de árbol lógico), para ello, genera las literales candidatas a formar parte de la conjunción asociada, apoyándose en el marco de generalidad provisto por la relación θ-subsunción. De este modo el algoritmo avanza en el espacio de búsqueda hacia la hipótesis final.

Formalmente el operador de refinamiento se expresa como sigue [3].

Def 14 ***Operador de Refinamiento***. *Un operador ρ de refinamiento bajo θ-subsunción obtiene, a partir de una cláusula c, un conjunto de cláusulas $\rho(c)$ tal que $\forall c' \in \rho(c), c \leq_\theta c'$.*

De esta manera, el algoritmo Tilde explora el espacio de búsqueda mediante el refinamiento de la hipótesis, lo que se consigue al añadir a la conjunción inicial Q la mejor de las literales candidatas (Q_b), mismas que son evaluadas de acuerdo con alguna de las métricas mencionadas en el capítulo precedente: entropía, ganancia de información, o razón de ganancia (véase la sección 2.2.2 en las pág. 23 y ss.). Así, la conjunción de literales que se sitúa en cada nodo está dada por: $Q_b - Q$, es decir, las literales añadidas a Q para producir Q_b [3, 4].

Ahora bien, el operador de refinamiento de la hipótesis se consigue mediante la definición de *modificadores*, mejor conocidos como **rmodes**, los cuales son establecidos previamente por el usuario [3, 4].

Los rmodes son especificaciones de la forma: $rmode(conj)$, donde $conj$ tiene aquellos indicadores posibles (definidos por el usuario) para las para literales que deben añadirse y las variables que deben ocurrir en ellas.

En la Tabla 3.2 se especifican los *rmodes* utilizados en este trabajo.

Tipo	Formato	Descripción
Entrada	rmode(+**A**).	Las variables de A deben ocurrir ya en la conjunción asociada.
Salida	rmode(−**A**).	Las variables de A no deben ocurrir en la conjunción asociada. Se generan variables nuevas.
Entrada/Salida	rmode(+−**A**).	A toma tanto las variables que ya ocurren en la conjunción asociada como variables nuevas.
Constante	rmode(#**A**).	La literal A debe estar fundamentada, es decir, debe instanciarse con los valores constantes presentes en los ejemplos de entrenamiento.

Tabla 3.2: Especificaciones de *rmodes* para la función ρ, usada en la generación de las literales candidatas a formar parte de la conjunción asociada, esto es, el refinamiento de la hipótesis inicial.

Partiendo de las especificaciones definidas por los *rmodes* se puede implementar el operador ρ como una función. El Algoritmo 7 muestra la forma en que se generan las literales con base en las especificaciones de los rmodes. El Algoritmo 7 analiza cada *rmode* para identificar las literales que pueden ser añadidas y su tipo (entrada,salida,constante o entrada/salida). De esta forma, genera literales con las combinaciones de argumentos definidas, introduciendo nuevas variables, buscando aquellas que deban ocurrir ya en la conjunción asociada, o buscando en los ejemplos sus valores constantes correspondientes. A partir de estas variables se integran las literales candidatas para formar parte de la conjunción asociada junto con Q.

En el Ejemplo 3, se muestran dos casos de cómo son obtenidas las literales candidatas, para refinar la hipótesis, mediante el uso de la función ρ, partiendo del conjunto de datos mostrados en la Tabla 3.3 (adaptado de T. Mitchell [22]) y de *rmodes* definidos para generar las literales.

Algorithm 7 Rho(Q: conjunción inicial, RM: rmodes, E: conjunto de ejemplos).
Recibe Q: conjunción inicial, RM: lista de *rmodes*, E: conjunto de ejemplos de entrenamiento. Regresa RMS: conjunto de literales candidatas.
La función Rho implementa el operador de refinamiento ρ, a partir del cual se guía la búsqueda en el espacio de hipótesis posibles de acuerdo con la especificación de *rmodes*.

if existe en los rmodes alguno de tipo # **then**
  Cargar E;
end if
for all $l \in Lits(Q)$ **do**
  for all $rm \in RM | rm \neq l$ **do**
    for all $a \in Args(Lits(rm))$ **do**
      if $a = +$ **then**
        Buscar en $Args(Lits(Q))$ el valor vc (variable/constante) correspondiente;
        $n_l \leftarrow vc$;
      end if
      if $a = -$ **then**
        Generar una variable nueva nv
        $n_l \leftarrow nv$;
      end if
      if $a = +-$ **then**
        Buscar en $Args(Lits(Q))$ el valor vc (variable/constante) correspondiente;
        Generar una variable nueva nv;
        $n_{l1} \leftarrow vc$ /*Entrada*/;
        $n_{l2} \leftarrow nv$ /*Salida*/;
      end if
      if $a = \#$ **then**
        Para cada $e \in E$ tomar el valor $ai = Args(Lits(e))$ correspondiente;
        $n_l \leftarrow ai$
      end if
    end for
    $rm_a \leftarrow \{Todos\ los\ argumentos\ obtenidos\ con\ los\ rmodes\}$
  end for
  $nva_l = \{Todas\ las\ literales\ obtenidas\ a\ partir\ de\ rm_a\}$
end for
Regresa $RMS = \{Todas\ las\ literales\ obtenidas\ con\ cada\ nva_l\}$

Ej 3 *Aplicación del operador* ρ *con base en distintas especificaiones de* $rmodes$

CASO I

Sean los rmodes *siguientes:*

rmode(cielo(#)) *rmode(temp(#))*

rmode(humedad(#)) *rmode(viento(#))*

Partiendo de estas especificaciones, se presentan dos casos de refinamiento de hipótesis iniciales mediante la función ρ*:*

1. *Sea Q = true*

[humedad(alta)], [humedad(normal)],
[temp(calor)], [temp(frio)], [temp(media)],
[viento(debil)], [viento(fuerte)]}

No.	Clase	Atributos
1	si	[viento(debil), humedad(alta), temp(calor), cielo(despejado)]
2	si	[viento(debil), humedad(alta), temp(media), cielo(lluvioso)]
3	si	[viento(debil), humedad(normal), temp(frio), cielo(lluvioso)]
4	si	[viento(fuerte), humedad(normal), temp(frio), cielo(despejado)]
5	si	[viento(debil), humedad(normal), temp(frio), cielo(soleado)]
6	si	[viento(debil), humedad(normal), temp(media), cielo(lluvioso)]
7	si	[viento(fuerte), humedad(normal), temp(media), cielo(soleado)]
8	si	[viento(fuerte), humedad(alta), temp(media), cielo(despejado)]
9	si	[viento(debil), humedad(normal), temp(calor), cielo(despejado)]
10	no	[viento(debil), humedad(alta), temp(calor), cielo(soleado)]
11	no	[viento(fuerte), humedad(alta), temp(calor), cielo(soleado)]
12	no	[viento(fuerte), humedad(normal), temp(frio), cielo(lluvioso)]
13	no	[viento(debil), humedad(alta), temp(media), cielo(soleado)]
14	no	[viento(fuerte), humedad(alta), temp(media), cielo(lluvioso)]

Tabla 3.3: Un conjunto de entrenamiento para realizar aprendizaje en primer orden, los *rmodes* instancian las variables con los valores de los ejemplos. Adaptado de [22].

A partir de la literal inicial Q = true *(el valor de verdad), se obtienen las literales mostradas entre corchetes, las cuales siguen el orden impuesto por θ-subsunción, como se puede apreciar:* true *es más general que cualquiera de las otras hipótesis, puesto que* true *siempre será verdadera, en tanto que una literal fundamentada como* cielo(despejado) *sólo lo será cuando en los datos exista una literal* cielo(X) *tal que X = despejado.*

2. Sea Q = cielo(despejado)

$$\begin{aligned}\rho(Q) = \{&[cielo(despejado),\ humedad(alta)]\\ &[cielo(despejado),\ humedad(normal)]\\ &[cielo(despejado),\ temp(calor)]\\ &[cielo(despejado),\ temp(frio)]\\ &[cielo(despejado),\ temp(media)]\\ &[cielo(despejado),\ viento(debil)]\\ &[cielo(despejado),\ viento(fuerte)]\}\end{aligned}$$

Partiendo de la literal Q = cielo(despejado), se añaden las conjunciones de literales mostradas entre corchetes, las cuales son θ-subsumidas por Q, como se ve para la primera literal candidata c_1*:*

$$cielo(despejado) \leq_\theta cielo(despejado), humedad(alta),$$

puesto que $Lits(Q) \subseteq Lits(c_1)$.

De esta manera, siguiendo el orden $\leq_\theta$ Q *es más general que* c_1*, puesto que* c_1 *requiere que se satisfaga una literal más:* humedad(alta)*, con lo que* Q *cubre más ejemplos que* c_1*, pero ésta es más estricta al clasificar un ejemplo, lo que representa un cambio de clase de equivalencia y por ende, un desplazamiento en el espacio de búsqueda (la rejilla de hipótesis posibles) en dirección a la hipótesis final buscada.*

CASO II

Sean los rmodes *siguientes:*

rmode(cielo(#)). *rmode(cielo(+−V)).*
rmode(temp(+V)). *rmode(humedad(−V)).*
rmode(viento(+−V)).

Partiendo de la literal inicial $Q = cielo(X)$*, se obtienen las siguientes literales, donde se muestra la inclusión de literales con variables que ya ocurren en la conjunción y otras con variables nuevas (p. ej. Y), además de literales instanciadas, de acuerdo con las especificaciones de los rmodes.*

1. Sea Q = cielo(X)

ρ(Q) = {*[cielo(X), cielo(X)]*
[cielo(X), cielo(Y)]
[cielo(X), cielo(despejado)]
[cielo(X), cielo(lluvioso)]
[cielo(X), cielo(soleado)]
[cielo(X), humedad(Y)]
[cielo(X), temp(X)]
[cielo(X), viento(X)]
[cielo(X), viento(Y)]
}

Como se puede apreciar, los rmodes proporcionan la vía para que el algoritmo explore el espacio de hipótesis al determinar las literales que serán incluidas en la conjunción asociada,

tal es el caso de la literal cielo(X), cuyos rmodes indican: a) [#] que debe instanciarse a un valor constante: [cielo(X), cielo(despejado)], [cielo(X), cielo(lluvioso)] y [cielo(X), cielo(soleado)]. b) [+−] incluir una variable que ya ocurre y una nueva variable: [cielo(X), cielo(X)], [cielo(X), cielo(Y)].

Con esto se muestra el uso de las especificaciones de rmodes como sesgo de lenguaje para algoritmos en primer orden.

Como se ilustra en el Ejemplo 3, el operador ρ obtiene las literales candidatas basándose en la conjunción que recibe, los *rmodes* y los datos de los ejemplos (sólo si en la especificación hay indicadores de constantes: #). De este modo, en Tilde la conjunción asociada se va construyendo a partir de la mejor de las literales candidatas (de acuerdo con la métrica que se utilice), hasta que se cumple una condición de paro, momento en el cual se detiene la construcción de la hipótesis y se regresa el árbol lógico obtenido hasta entonces.

En el siguiente apartado se presenta un algoritmo de aprendizaje incremental en primer orden: ILDT, el cual se basa en los algoritmos en primer orden Tilde y TG [3, 9]; y en el algoritmo incremental proposicional ID4 [32].

3.2.2. Aprendizaje Incremental en Primer Orden: ILDT

Los algoritmos inductivos de aprendizaje incremental construyen una hipótesis a partir de un flujo de evidencias. De esta manera, la hipótesis se va ajustando a las fluctuaciones en los datos, esto implica su constante revisión a medida que se obtienen más ejemplos. De acuerdo con lo expuesto en esta tesis, se tienen los siguientes elementos para realizar aprendizaje incremental inductivo en primer orden, particularmente, aprendizaje de árboles lógicos:

1. Sesgo inductivo de búsqueda basado en *rmodes*.

2. Métricas para determinar la conjunción que ha de situarse en cada nodo.

3. Uso de contadores de clase para determinar estás métricas sin almacenar ejemplos.

El punto 3 específicamente, está relacionado con la incrementalidad, ya que tras un arreglo del uso de los contadores, se consigue como en ID4, aprender de los datos prescindiendo de su almacenamiento, es decir, aprender de un flujo de evidencias y no de un conjunto completo de ellas. En concordancia con esto, el agoritmo TG expuesto en [9], añade el uso de una medida que indique si un nodo debe bifurcarse, de esta manera, el algoritmo modifica la hipótesis actual según se presenten nuevas evidencias y en el caso de que éstas sean relevantes numéricamente para hacer crecer el árbol en un nivel. Cada crecimiento del árbol se efectúa situando una literal (la de mayor beneficio según los datos) en un nodo de decisión y agregando sus dos nodos hoja correspondientes. TG se muestra en el Algoritmo 8 como se describe en [9].

Algorithm 8 TG(e: ejemplo de entrenamiento, RM: rmodes).
Recibe e, un ejemplo de entrenamiento y RM las especificaciones de *rmodes*. Regresa T: el árbol lógico de regresión aprendido. TG crea un árbol de regresión en primer orden de forma incremental, utilizando estadístas en cada nodo para determinar su bifurcación.

```
Crear un árbol relacional con estadísticas vacías;
while haya nuevas evidencias e disponibles do
  Recorrer el árbol hasta alcanzar un nodo hoja;
  Actualizar las estadísticas de cada hoja de acuerdo con e;
  if las estadísticas de la hoja indican que es necesario expandir el árbol then
    Generar un nodo interno usando la mejor prueba;
    Generar 2 nuevas hojas al nodo con estadísticas vacías;
  end if
end while
Regresar el árbol aprendido;
```

A partir de los algoritmos TG (basado en Tilde) e ID4 se obtiene ILDT (Incremental Induction of Logical Desicion Trees), el cual induce incrementalmente un árbol lógico. ILDT evalúa las literales candidatas mediante un medida de información, de tal manera que aquella literal con el mayor beneficio se va situando en los nodos de decisión y el árbol se va expandiendo a medida que se tienen disponibles más datos y se encuentran literales que ayuden a discriminarlos. ILDT se muestra en el Algoritmo 9.

Ánalogamente a lo que se hace en Tilde [3], el espacio de búsqueda se explora generando literales candidatas bajo θ-subsunción obtenidas vía la aplicación del operador ρ, el cual se basa en las especificaciones de *rmodes* tal como se explicó en la sección anterior. Inicialmente,

ILDT colecta los ejemplos disponibles si pertenecen a la misma clase, cuando alguno de ellos tiene una clase distinta comienza el proceso de selección de la mejor literal. Las cuentas de cada nodo se actualizan a medida que llega la nueva evidencia, de tal manera, se crean nuevos nodos si es necesario.

El algoritmo toma un ejemplo a la vez para inducir y revisar la hipótesis. La expansión del árbol se efectúa con base en dos procesos principales:

1. *Revisión del beneficio.* Determina la mejor literal generada por ρ, la medida de bondad se calcula con las cuentas de clase $(+, -)$ de cada nodo del árbol, tanto de los nodos de decisión como de las hojas, utilizando una métrica de información (p. ej. entropía, ganancia, o razón de ganancia). La mejor literal es situada en el nodo decisión para expandir el árbol si cumple con un criterio de paro.

2. *Reestructuración de nodos.* Revisa la relevancia de los nodos en el árbol. Esta operación usa un parámetro δ definido por el usuario, el cuál actúa como umbral para determinar si un nodo padre puede intercambiarse por un nodo hijo, en los casos en que el árbol resulta inconsistente, se poda el nodo menos benéfico (al modo de ID4). De esta manera, en el árbol se preservan las literales con mayor relevancia. Este procedimiento puede verse en el Algoritmo 10.

Algorithm 9 ildt(e, RModes, BG, T).

Recibe ejemplo e, RModes, un conjunto de especificaciones de *rmodes*. Regresa T, el árbol aprendido con base en e. Construye un árbol lógico a partir de un flujo de datos de entrenamiento, tomando un ejemplo e a la vez.

collectar(e,E)
if $\forall e \in E | class(e) = c \wedge T = null$ **then**
  $T \leftarrow inode(leaf(c), null, null)$;
  $actualizaEstads(root(T))$;
else
  if $|E| > 1$ **then**
    $litsCandidatas \leftarrow candidatas(E, RModes)$;
    $mejorCandidata \leftarrow mejor(litsCandidatas, BG)$;
    $T \leftarrow inode(mejorCandidata, leaf(c_{left}), leaf(c_{right}))$;
    $actualizaEstads$;
    $borrar(E)$;
  else
    $clase' \leftarrow recorreArbol(e, T)$;
    $G \leftarrow Ganancia(leaf(class'))$;
    $actualiza(leaf(class'))$; {conjuncion asociada y estadisticas}
    if $tomaClase(e) \neq clase'$ **then**
      $G' \leftarrow Ganancia(leaf(class'))$;
      if $particion(G, G') = True$ **then**
        $litsCandidatas \leftarrow candidatas(E, RModes)$;
        $mejorCandidata \leftarrow mejor(litsCandidatas)$;
        $c^+ = tomaEstads(mejorCandidata, +)$;
        $c^- = tomaEstads(mejorCandidata, -)$;
        $c_{left} = mayor(c^+, c^-)$;
        $c_{right} = menor(c^+, c^-)$;
        $nuevoNodo \leftarrow inode(mejorCandidata, leaf(c_{left}), leaf(c_{right}))$;
        $reemplaza(nuevoNodo, leaf(clase'), T)$;
        $T \leftarrow intercambiaNodos(T)$;
      end if
    end if
  end if
end if
$regresa(T)$;

Algorithm 10 intercambiaNodos(T: árbol lógico).

Intercambia los nodos padre por sus nodos hijo en el caso de que el beneficio de éstos sea mayor, o podando si el cambio provoca inconsistencias.

T = inode(Raiz,I,D);
$\Delta_l \leftarrow Raiz - I$
$\Delta_r \leftarrow Raiz - D$
Δ_m := mayor(Δ_l,Δ_r);
if $\Delta_m < \delta$ **then**
    Intercambiar el nodo correpondiente (I o D) con Raiz;
end if
if inconsistency(T) = true **then**
  Podar subarbol;
else
  Intercambiar nodos;
end if
Cambiar las cuentas $(+,-)$ de cada nodo de acuerdo con la nueva estructura; {Seguir revisando nodos inferiores}
intercambiaNodos(subarbol izquierdo);
intercambiaNodos(subarbol derecho);

Capítulo 4

Resultados

4.1. Pruebas

En este capítulo se muestran las pruebas efectuadas con el algoritmo ILDT y sus resultados. De acuerdo con la metodología de escalamiento seguida en este trabajo, concretamente, en el punto 6 (sección 3.1 págs. 43 y 44) se indica que la evaluación del algoritmo escalado comienza por contrastar sus resultados con su contraparte no escalada. Debido a que ILDT conjunta dos tipos de escalamiento, a saber: *Incrementalidad* y *Representación en Primer Orden*, por ello ILDT se contrastó con TILDE, que representa la versión no incremental en primer orden de un algoritmo para obtener árboles lógicos a partir de datos.

A continuación se presenta la evaluación del algoritmo ILDT.

4.1.1. Evaluación

En este caso, ILDT se plantea como el algoritmo TILDE escalado a su versión incremental. Por esta razón el algoritmo de control será TILDE. Para evaluar ILDT con respecto a TILDE, se utilizó una implementación propia del algoritmo mencionado en la sección 3.2 del Capítulo 3 (pág. 45). La comparación se hizo utilizando los conjuntos de entrenamiento A y B mostrados en las Tablas 4.1 y 4.3, los cuales han sido utilizados en las publicaciones sobre los algoritmos ID3, ID4 e ID5r (revisar [22, 32] principalmente) como conjuntos de datos, de manera que realizar las comparaciones con base en estos conjuntos es conveniente para probar los alcances del algoritmo aquí propuesto. En paritcular el Conjunto A es particularmente indicado para realizar la prueba de incrementalidad pues determina una variación en los

datos que requiere que la hipótesis aprendia sea revisada continuamente [32].

Conjunto de entrenamiento A			
ej(#,clase,[lista de literales]).			
ej(1,n,	[estatura(bajo),	cabello(rubio),	ojos(cafe)]).
ej(2,n,	[estatura(alto),	cabello(obscuro),	ojos(cafe)]).
ej(3,p,	[estatura(alto),	cabello(rubio),	ojos(azul)]).
ej(4,n,	[estatura(alto),	cabello(obscuro),	ojos(azul)]).
ej(5,n,	[estatura(bajo),	cabello(obscuro),	ojos(azul)]).
ej(6,p,	[estatura(alto),	cabello(rojo),	ojos(azul)]).
ej(7,n,	[estatura(alto),	cabello(rubio),	ojos(cafe)]).
ej(8,p,	[estatura(bajo),	cabello(rubio),	ojos(azul)]).

Tabla 4.1: Conjunto de entrenamiento A, expresado en Primer Orden para contrastar los algoritmos TILDE e ILDT (Adaptado de [32]).

Arboles obtenidos con el Conjunto A (Tabla 4.1)

Los resultados obtenidos con este conjunto de entrenamiento se muestran en la Tabla 4.2. La métrica utilizada es Ganancia de Información.

La prueba consiste en la ejecución de ILDT con 15 variaciones aleatorias del conjunto de datos A. Tales variaciones se efectuaron con el fin de proporcionarle al algoritmo los mismos datos ordenados de distinta manera, para determinar el impacto de la distribución de los datos en la capacidad de construir una hipótesis adecuada a todo el conjunto.

Arboles obtenidos con el Conjunto B (Tabla 4.3)

El Conjunto B, es el segundo conjunto de entrenamiento con el que se hizo la comparación entre TILDE e ILDT.

En la Tabla 4.4 se muestran los resultados de 15 corridas del algoritmo incremental ILDT. Cada corrida utilizó el conjunto B cambiando aleatoriamente el orden de los ejemplos. La métrica utilizada es Ganancia de Información.

Corrida	Nodos	Cobertura %.
TILDE		
1	2	100.0.
ILDT		
1	3	75.0.
2	3	75.0.
3	3	100.0.
4	**6**	**62.5**.
5	3	75.0.
6	4	100.0.
7	5	87.5.
8	**2**	**100.0**.
9	2	75.0.
10	2	75.0.
11	3	75.0.
12	3	75.0.
13	3	100.0.
14	4	87.0.
15	2	100.0.
μ	**3**	**84.1**.

Tabla 4.2: Cobertura de los árboles obtenidos con TILDE a partir del conjunto de entrenamiento A y con ILDT con 15 variaciones aleatorias de dicho conjunto.

Conjunto de entrenamiento B

ej(#,clase,[lista de literales]).				
ej(1 ,si,	[cielo(despejado),	temp(calor),	humedad(alta),	viento(debil)]).
ej(2 ,si,	[cielo(lluvioso),	temp(media),	humedad(alta),	viento(debil)]).
ej(3 ,si,	[cielo(lluvioso),	temp(frio),	humedad(normal),	viento(debil)]).
ej(4 ,si,	[cielo(despejado),	temp(frio),	humedad(normal),	viento(fuerte)]).
ej(5 ,si,	[cielo(soleado),	temp(frio),	humedad(normal),	viento(debil)]).
ej(6 ,si,	[cielo(lluvioso),	temp(media),	humedad(normal),	viento(debil)]).
ej(7 ,si,	[cielo(soleado),	temp(media),	humedad(normal),	viento(fuerte)]).
ej(8 ,si,	[cielo(despejado),	temp(media),	humedad(alta),	viento(fuerte)]).
ej(9 ,si,	[cielo(despejado),	temp(calor),	humedad(normal),	viento(debil)]).
ej(10,no,	[cielo(soleado),	temp(calor),	humedad(alta),	viento(debil)]).
ej(11,no,	[cielo(soleado),	temp(calor),	humedad(alta),	viento(fuerte)]).
ej(12,no,	[cielo(lluvioso),	temp(frio),	humedad(normal),	viento(fuerte)]).
ej(13,no,	[cielo(soleado),	temp(media),	humedad(alta),	viento(debil)]).
ej(14,no,	[cielo(lluvioso),	temp(media),	humedad(alta),	viento(fuerte)]).

Tabla 4.3: Conjunto de entrenamiento B, expresado en Primer Orden para contrastar los algoritmos TILDE e ILDT (Adaptado de [22]).

4.1.2. Discusión

De acuerdo con los resultados de las pruebas sobre ILDT, puede verse que ILDT es capaz de aprender incrementalmente hipótesis como lo hace TILDE, el algoritmo de control utilizado, sin embargo, adolece de la dependencia de la distribución de los datos.

Para ejemplificar lo anterior, en la Figura 4.1 se muestran el árbol obtenido por TILDE y uno de los obtenidos por ILDT con igual *cobertura*.

Como se puede apreciar en la Figura 4.1, estos árboles difieren en la conjunción asociada, pues son distintas en la segunda literal, pero ambos tienen una extensión equivalente. Estos árboles clasifican el 100 % del conjunto A, es decir, tienen la misma *cobertura*, como se muestra a continuación.

TILDE: $cabello(obscuro) \wedge ojos(azul)$ Extensión: $\{e_1, \ldots, e_8\} \in Conjunto\, A$

ILDT: $cabello(obscuro) \wedge ojos(cafe)$ Extensión: $\{e_1, \ldots, e_8\} \in Conjunto\, A$

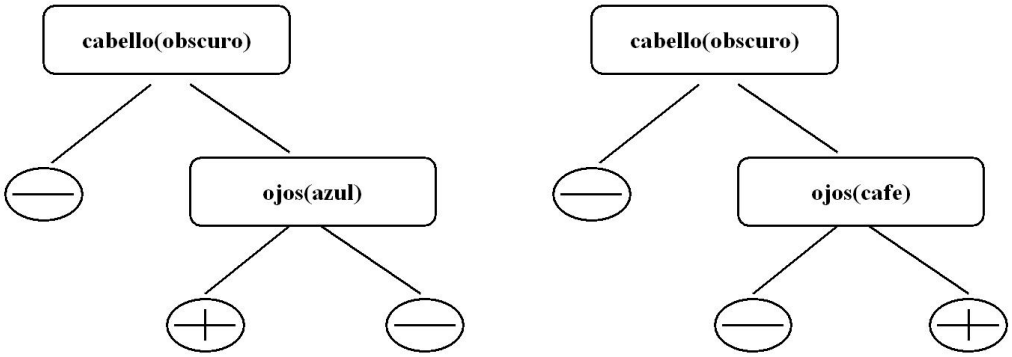
Es de notar, que la distribución de los datos es un factor que influye sobremanera en el algoritmo ILDT, característica que hereda principalmente del algoritmo ID4, puesto que

Corrida	Nodos	Cobertura %.
TILDE		
1	3	85.7143.
ILDT		
1	5	85.7143.
2	4	78.5714.
3	**5**	**57.1429**.
4	4	64.2857.
5	2	71.4286.
6	5	57.1429.
7	7	64.2857.
8	7	92.8571.
9	6	85.7143.
10	5	57.1429.
11	4	85.7143.
12	7	64.2857.
13	7	78.5714.
14	5	78.5714.
15	**5**	**92.8571**.
μ	**5**	**74.2857**.

Tabla 4.4: Cobertura de los árboles obtenidos con TILDE a partir del conjunto de entrenamiento B y con ILDT con diez variaciones aleatorias de dicho conjunto.

los procedimientos de revisión de la estructura no son suficientes para adecuar la hipótesis a los cambios en los datos, dejando conceptos sin aprender, como se ve en los resultados de la Tabla 4.2. A este respecto, es preciso indicar, que el algoritmo ID5r proporciona una manera de reestablecer la estructura del árbol (proposicional) que va construyendo, de tal forma que no importe el orden del conjunto de entrenamiento. Por su parte, ILDT, como primera aproximación al aprendizaje incremental en primer orden funciona como ID4, con miras a reproducir los resultados obtenidos con ID5r pero utilizando árboles lógicos.

Otro punto a destacar, es que ILDT encontró una hipótesis con mayor cobertura a partir de los mismos datos, ésto se ve claramente en la corrida 15 de la Tabla 4.4, donde se alcanza una cobertura del 92,8 %, la cual es superior a la obtenida con el árbol lógico construido por TILDE, el cual, para cualquiera de las 15 ordenaciones aleatorias de los datos obtiene



(a) TILDE, Cobertura 100 %

(b) ILDT, Cobertura 100 %

Figura 4.1: Árboles obtenidos con TILDE e ILDT con igual cobertura.

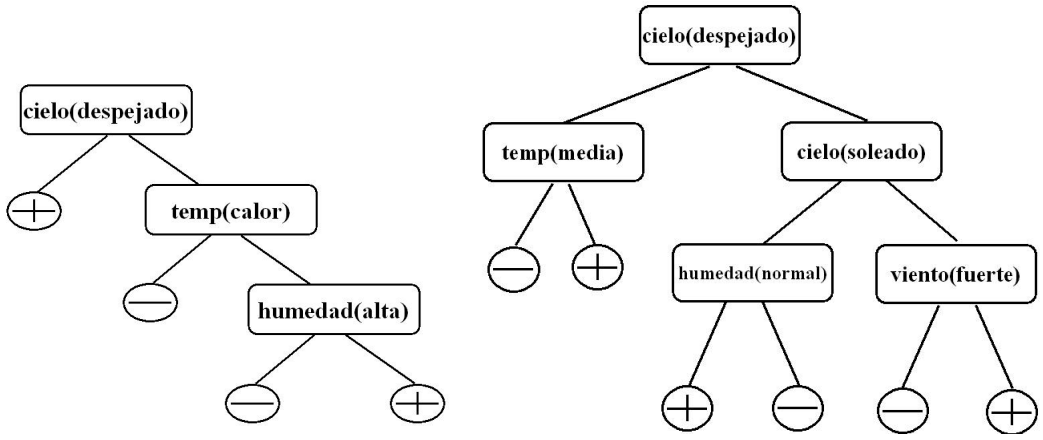
la misma hipótesis. Sin embargo, se trata de una hipótesis que se sobreajusta a los datos, con lo que puede perder capacidad de predicción, además, incluye una rama que clasifica erróneamente uno de los ejemplos de entrenamiento, pues incluye una literal más a la conjunción asociada, producto de la estrategia de expansión del árbol, que no se ve afectada en este caso por la reestructuración del árbol afectándo negativamente a la hipótesis completa, como se muestra a continuación:

$cielo(despejado) \wedge temp(media)$ Extensión: $\{\} \in Conjunto\,A$

$cielo(despejado)$ Extensión: $\{e_1, e_2, e_3, e_4\} \in Conjunto\,A$

En este mismo sentido, de manera generalizada se aprecia que los árboles obtenidos con ILDT constan de más nodos que los que TILDE construye, lo que se ejemplifica en la Figura 4.2, esto se debe, como ya se comentó, a la estrategia de expansión del árbol, la cual a medida que van llegando nuevas evidencias crea subárboles en ciertos nodos para ampliar la cobertura de la hipótesis lograda, y el mecanismo de intercambio de nodos no considera que deba realizarse un cambio ni su posible poda. Lo anterior conduce a revisar la estrategia de expansión y reconstrucción de la hipótesis, con el fin de evitar este crecimiento innecesario de los nodos, tal como se plantea en el trabajo a futuro.

A continuación se muestran los resultados de la aplicación de ILDT como mecanismo de aprendizaje incremental en un Sistem Multi-Agente BDI.



(a) TILDE, Cobertura 85.7143 % (b) ILDT, Cobertura 92.8571 %

Figura 4.2: Árboles obtenidos con TILDE e ILDT donde la cobertura del árbol construido incrementalmente sobrepasa la correspondiente del obtenido por TILDE.

4.2. Aplicaciones

Una de las aplicaciones del Aprendizaje Automático incremental, y del algoritmo ILDT en particular se encuentra en los Agentes Inteligentes y en los Sistemas Multi-Agente (MAS), pues es una parte esencial de éstos. En lo que resta de esta sección se presenta de forma resumida el artículo [18], enviado al congreso *6th Mexican International Conference on Artificial Intelligence* cuyas memorias son editadas en *Leture Notes of Artificial Intelligence*, actualmente en revisión (ver Apéndice A).

Se considera a un agente como un sistema computacional persistente en el tiempo, capaz de actuar *autónomamente* para cumplir sus objetivos o metas, cuando está situado en un ambiente determinado [36]. La característica de autonomía se entiende como la facultad del agente de accionar sin intervención directa ya sea humana o de otra índole, manteniendo control sobre sus estados internos y sus acciones. Además, para que un agente sea considerado *inteligente*, debe mostrar un comportamiento flexible, es decir, reactivo (percibir el ambiente y responder ante cambios en él), proactivo (orientarse a metas y tomar la iniciativa para cumplirlas) y poseer habilidad social (interactuar con otros agentes, inclusive con

humanos) [36, 12].

Tomando en cuenta que el ambiente en el que están inmersos los agentes frecuentemente es cambiante, surge la necesidad de implementar un esquema que le permita contender con los cambios y responder adecuadamente ante ellos. Esta situación puede estructurarse como un caso de aprendizaje, particularmente un *aprendizaje supervisado incremental*.

De esta manera, la experiencia, es decir, los ejemplos que se utilizarán para el aprendizaje son aquellas observaciones que el agente recoge de su ambiente, configurando así su conjunto de entrenamiento.

En el mismo sentido, los SMA realizan tareas en conjunto, lo que implica la existencia de comunicación y coordinación entre ellos. Esto puede verse también como un problema de aprendizaje incremental, pues deben actualizar sus estados internos mediante los acuerdos a los que llega cada agente con sus pares dentro del sistema. Así, se configura de nueva cuenta un flujo de observaciones que eventualmente se pueden considerar coom un conjunto de ejemplares para realizar el aprendizaje, en esta caso, un aprendizaje colaborativo.

Ejemplo de ello es el protocolo SMILE (*Sound Multi-agent Incremental LEarning*) [11] ideado para preservar la consistencia del aprendizaje colaborativo en un SMA sin una memoria central. Cada agente mantiene un conjunto de creencias (que actualiza incrementalmente) y un conjunto de observaciones del entorno. El SMA es consistente cuando todos sus miembros lo son. El mecanismo de actualización de creencias se configura como un aprendizaje incremental de listas de términos. SMILE especifica un algorimo de aprendizaje incremental basado en la cobertura de las hipótesis en ejemplos positivos y negativos.

Debido a que ILDT es un algoritmo de aprendizaje incremental, es posible emplearlo dentro del protocolo SMILE pues permite alcanzar la eficiencia local en términos del protocolo. A continuación se presenta un ejemplo del uso de ILDT como el procedimiento de actualización de creencias de SMILE, donde el protocolo está implementado en Jason, un intérprete extendido de AgentSpeak(L) [27, 6], de esta forma, cada miembro del sistema es un agente BDI, por lo que su conducta está determinada por planes, cuya aplicación depende del contexto, donde se evalúan las creencias, deseos e intenciones del agente, como sus

percepciones del medio ambiente y, por ende, de los otros agentes.

En el caso que nos ocupa, cada ejemplar en este caso representa el contexto de un plan aplicable que tiene un agente en el dominio del *mundo de los cubos*. Cada agente, dentro de SMILE, debe consultar sus creencias con los otros agentes para mantenerse consistente [17]. De este modo, cada gente ejecuta ILDT de acuerdo con sus percepciones. La importancia de ILDT radica en dos puntos principales, a saber:

a) El mecanismo de aprendizaje comienza cuando se encuentra un ejemplo con una clase distinta al resto colectado hasta ese momento, esto es, cuando cada agente encuentra un plan fallido, es decir, hay una inconsistencia en sus creencias y en el ambiente que no le permite actuar. Así, los ejemplos negativos ayudan a que el aprendizaje se realice más rápido, haciendo eficiente también con esto la comunicación entre el SMA completo.

b) El aprendizaje incremental hace que el SMA sea *localmente eficiente* [11], al permitirle a cada agente actualizar sus creencias en concordancia con los demás agentes, obteniendo finalmente la *a-consistencia* del sistema completo.

En la Tabla 4.5 se muestran los contextos de los planes (ejemplos) con los que operó ILDT para realizar el aprendizaje, en el caso concreto de la acción *apilar* un cubo sobre otro.

Contextos de los planes a ser aprendidos

ej(#,clase,[lista de literales]).	
ej(1 ,exito,	[deteniendo(c), libre(b), sobreMesa(a),sobre(b,a)]).
ej(2 ,falla,	[libre(b),libre(c), sobreMesa(a), sobreMesa(c), sobre(b,a), libreBrazo(null)]).
ej(3 ,falla,	[sobre(a,b), deteniendo(c), sobreMesa(b), libre(a)]).
ej(4 ,falla,	[deteniendo(c), libre(a), libre(b), sobreMesa(a), sobreMesa(b)]).
ej(5 ,exito,	[deteniendo(c), libre(a), libre(b), sobreMesa(a), sobreMesa(b)]).

Tabla 4.5: Conjunto de ejemplos consistentes en contextos de planes de agentes para la acción *apilar*, en el dominio del mundo de los cubos.

Al ejecutar ILDT se obtiene el siguiente árbol lógico, que representa un nuevo contexto de un nuevo plan para la acción de apilar que el agente adoptará de acuerdo con el protocolo SMILE:

```
<>deteniendo(X)
    <+>libre(Y)
        <+>libre(Z)
            (+) leaf(exito)
            (-) leaf(falla)
        (-) leaf(exito)
    (-) leaf(falla)
```

De esta forma, el agente tomará como contexto de un nuevo plan para *apilar* cuando sus creencias sean válidas de acuerdo con las dos conjunciones posibles:

- deteniendo(X), libre(Y), libre(Z)
- deteniendo(X), ¬ libre(Y)

Es así como puede aplicarse ILDT como mecanismo de actualización de creencias para un SMA dentro del protocolo SMILE, permietiendo el aprendizaje incremental y reduciendo la comunicación necesaria para que los agentes pertenecientes as SMA puedan coordinarse, pues no es necesario que tengan un conjunto completo de ejemplos para que se ralice el aprendizaje.

Hasta ahora, las implementaciones de aprendizaje en agentes BDI se han efectuado con buenos resultados utilizando procedimientos parcialmente incrementales, esto es, tras haber reunido un número mínimo necesario de ejemplos, se ejecuta Tilde, el cual construye una hipótesis nueva desde cero a medida que aparecen datos no vistos anteriormente; repitiendo el proceso cada vez que se requiera [16, 15, 17]. Sin embargo, SMILE requiere un aprendizaje realmente incremental para garantizar la *a-consistencia* de todo el sistema [11]. Ahí la importancia de ILDT dentro de SMILE, pues permite obtener una consistencia localmente eficiente, lo cual es determinante para que el SMA alcance la consistencia global. De este modo, ILDT como parte de SMILE garantiza que el sistema sea global, reduciendo además la necesidad de comunicación entre los agentes, pues el algoritmo saca provecho de los ejemplos negativos fundamentalmente, sin necesidad de contar con un conjunto completo de ejemplos.

Capítulo 5

Conclusiones y trabajo futuro

En este Capítulo se exponen la conclusión de este estudio y el trabajo que debe realizarse como continuación de la investigación e implementación de algoritmos de aprendizaje incremental en primer orden.

5.1. Conclusiones

En este trabajo se ha visto la manera en que puede obtenerse un algoritmo de aprendizaje incremental en primer orden siguiendo la metodología propuesta en [33].

La línea de escalamiento adoptada consistió en tomar un algoritmo de construcción de árboles lógicos en primer orden y realizar el paso incremental adoptando características de algoritmos incrementales tanto proposicionales como relacionales. El algoritmo aquí propuesto (ILDT) sigue la representación y el sesgo de lenguaje de TILDE, y la estrategia incremental de ID4 fundamentalmente.

Con este estudio se muestra que el aprendizaje incremental en primer orden es factible principalmente a partir de los siguientes elementos:

- El uso de la Representación en Primer Orden basada en *Interpretaciones*: esto permite que el algoritmo considere los datos de manera independiente, haciendo uso del supuesto de localidad. De este modo, no es neceseario tener un conjunto completo de entrenamiento, sino que los ejemplos pueden tomarse uno a uno, esto es: incrementalmente.

- El *sesgo de lenguaje* basado en especificaciones de *rmodes* que siguen el *quasi* orden

impuesto por θ-subsunción: las especificaciones son una forma práctica de implementar el sesgo de lenguaje para guiar la exploración del espacio de hipótesis y al mismo tiempo aprovechar la estructuración del espacio que se consigue con la relación de generalidad θ-subsunción, para moverse en el espacio determinado por las clases de equivalencia definidas, eliminando la necesidad de probar todas las literales pertenecientes a cada clase.

Por las características del algoritmo obtenido, los ejemplos negativos, juegan un papel importante pues son los que desencadenan en primer término el aprendizaje y posteriormente, al cambiar las estadísticas de cada nodo, ayudan a determinar la bondad de las literales candidatas, permitiendo así al algoritmo situarlas en la hipótesis. Esto se refleja en la aplicación mostrada en la sección 4.2, donde los ejemplos negativos aportan mayores beneficios al aprendizaje, reduciéndo con ello la demanda de intercambiar ejemplos entre los agentes de forma intensiva.

De acuerdo con los requerimientos del protocolo SMILE, el mecanismo de actualización de creencias se configura como un procedimiento de aprendizaje incremental, donde las observaciones del agente, junto con las de los otros miembros del sistema, integran el flujo de datos de entrenamiento necesarios para aprender.

En el caso de los agentes BDI, cuya información (planes, creencias, deseos e intenciones) utiliza la representación en primer orden, ILDT resulta conveniente por utilizar expresiones relacionales. Además, este algoritmo, al realizar aprendizaje incremental, garantiza que cada agente pueda actualizar sus creencias en coordinación con los otros, de forma que se tiene lo que en SMILE se denomina *eficiencia local*, permitiendo con ello que todo el SMA sea consistente cada vez que hay una revisión de creencias.

De esta forma, se consigue que el mecanismo M de actualización de creencias establecido en SMILE sea implementado con datos en primer orden, mediante ILDT, de manera que el SMA tenga una consistencia fuerte (*strong MAS-consistency*).

5.2. Trabajo futuro

El algoritmo ILDT, obtenido en este estudio, representa el primer paso en la sucesión del escalamiento incremental en primer orden de los algoritmos de inducción de hipótesis en forma de árboles de desición, en esta caso árboles lógicos.

Debido a que ILDT adopta la estrategia incremental de ID4 principalmente, hereda de éste la dependencia de la distribución de los ejemplos de entrenamiento, y con ello la imposibilidad de aprender conceptos cuando se presenta un flujo de datos que de primera instancia no aportan la suficiente información para discriminar las literales que deben situarse en la conjunción asociada del árbol lógico a construir.

Para contender con ello, es necesario adecuar e implementar las siguientes características a ILDT:

- Un mejor esquema de revisión de la estructura del arbol. Este puede establecerse al menos en dos sentidos:
  - Cuando en un intercambio de nodos resulte un árbol inconsistente, es decir, con dos ramas con la misma etiqueta de clase, en lugar de descartar los subárboles en la reestructuración, duplicarlos. De este modo, en cada intercambio que haya que hacer entre nodos se evitará la pérdida de ramas. Esto implica sin embargo, tener otro mecanismo de revisión y poda de aquellas ramas que se hagan innecesarias en el curso del aprendizaje, pero que evitaría la pérdida de ramas, que eventualmente pudieran ser importantes, como actualmente se hace en ILDT.
  - Diseñar un método de reestructuración del árbol al estilo de ID5r, el cual modifique la conjunción asociada completa, situando en los lugares indicados las literales que aporten más beneficio de acuerdo con las nuevas evidencias.
- Seguir la estrategia de ID5r de representación del árbol de decisión. Esto implica modificar la estructura de los árboles lógicos, sin alterar la semántica, para logar una que permita asociarle a ciertos nodos los ejemplos que éstos cubren, con el fin de disminuir

el impacto de las fluctuaciones de los datos; o bien, implementar un mecanismo que haga factible guardar dicha información en una memoria auxiliar para usarla posteriormente.

- Realizar una implementación que resulte eficiente para tratar con bases de datos de cierta envergadura. Debido a la expansión de posibilidades en el espacio de hipótesis, es necesario reducir la complejidad de la búsqueda. Una opción es el uso de *Query Packs* [5], los cuales son una forma abreviada de representar conjunciones de literales, permitiendo así un manejo eficiente al eliminar literales individuales y substituirlas por expresiones disyuntivas. Así, los *Query packs* estructuran conjuntos de literales similares, esto es, expresan un conjunto de literales en términos de sus componentes sin perder cobertura, evitando que las literales individuales se evalúen múltiples veces.

Finalmente, se menciona que ILDT puede ser aplicado en varias tareas de clasificación, por ejemplo, en las siguientes áreas:

- **Bioinformática**. Si bien en esta área se han aplicado métodos estadísticos como las Redes Bayesianas, existen diversos problemas a los que ILDT puede aplicarse, pues tienen algunas caractarísticas que hacen interesante la aplicación de métodos de aprendizaje incremental en primer orden: a) Parten de modelos (p. ej., modelos evolutivos), que puede representarse como Conocimiento Base en el algoritmo. b) En su gran mayoría, se trabaja con cuantiosas cantidades de datos, distribuidos en varios nodos de internet. c) La información puede necesitar una representación relacional, como en el caso de las estructuras topológicas tridimensionales, las cuale incluyen información espacial. d) Cuentan con métricas de similitud específicas, como las características fisico-químicas, que son adecuadas para la evaluación de la información usada en la construcción de la hipótesis.

  Entre los casos en los cuales ILDT es aplicable se encuentran los siguientes:

  1. Construcción de árboles filogenéticos a partir de secuencias de ADN, en particular, ILDT puede aplicarse para encontrar la *topología óptima* de los árboles

filogenéticos [2], la cual se obtiene de forma incremental, al presentarle al algoritmo secuencia por secuencia. La cantidad de árboles posibles es exponencial, lo que requiere de una heurística de búsqueda, representada por los *rmodes*.

2. Análisis de secuencias de ADN no anotadas e identificación de diferencias entre secuencias con distinta funcionalidad. Esta actividad requiere aprendizaje automático debido a la dificultad del análisis derivada de la gran cantidad de datos.
3. Análisis de *proteomas*. Trata de determinar la secuencias de proteínas de una especie, el análisis de su localización estructural, de su función y precisar el estado bioquímico en que se encuentran las proteínas. ILDT puede aplicarse particularmente en el análisis de la estructura tridimensional, pues sus expresan relaciones espaciales, mismas que pueden representarse como expresiones en primer orden.

- **Minería de Datos**. Debido a que su actividad central es econtrar patrones en los datos de forma automática (o semiautomática), en ésta área se aplican técnicas de aprendizaje automático, además dado que en ella se trabaja con grandes cantidades de datos [26], es necesario contar con un aprendizaje incremental. Aunado a lo anterior, mucha de la información aparece en Bases de Datos Relacionales, lo que hace conveniente el uso de algoritmos cuya representación esté en primer orden, tal como ILDT.

  De esta forma, una aplicación de ILDT en este campo puede ser el minado de información sobre el desempeño y trayectorias académicas de los estudiantes pertenecientes a alguna institución educativa, por ejemplo la Universidad Veracruzana, que conserva información socioeconómica y académica de la cual se pueden obtener aquellas relaciones no explícitas en los datos que podrían servir de base para diversos estudios y decisiones académico-administrativas. Ejemplo de ello es obtener un índice objetivo de los factores que influyen en las trayectorias académicas que pueden considerarse exitosas (bajo ciertos lineamientos). Además de identificar la incidencia de éstos en la eficiencia terminal de las facultades, escuelas e institutos.

Bibliografía

[1] Alpaydin Ethem, *Introduction to Machine Learning*, Cambridge MA (2004).

[2] Brunak Søoren, Baldi Pierre F.*Bioinformatics: The Machine Learning Approach* (2a. Edición). The MIT Press. London England. 2001.

[3] Blockeel Hendrik, Ph.D. Thesis *Top-down induction of first order logical decision trees*, Department of Computer Science, K.U.Leuven, December 1998, Leuven Belgium.

[4] Blockeel H., De Raedt L., Jacobs N., Demoen B. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, Katholieke Universiteit Leuven, 3,59-93, 1999.

[5] Blockeel Hendrik, Dehaspe Luc,Demoen Bart, Janssens Gerda, Ramon Jan, and Vandecasteele Henk. Improving the efficiency of Inductive Logic Programming through the use of query packs. In *Journal of Artificial Intelligence Research*, 16(0):135–166. Enero 2002.

[6] Bordini R.H., Hübner F.J. *Jason. A Java-based agentSpeak interpreter used with saci for multi-agent distribution over the net*. Marzo 2006.

[7] Claire Nédellec, Céline Rouveirol, Francesco Bergadano, Hilde Adé and Birgit Tausend. In L. De Raedt(Ed) *Advances in Inductive Logic Programming*. Vol. 32 of Frontiers in Articial Intelligence and Applications. Amsterdam, The Netherlands: IOS Press. pp. 82-103. 1996.

[8] De Raedt Luc. Logical Settings for concept-learning. In *Artificial Intelligence*. Elsevier. Vol. 95, No. 1, pp. 187-201. 1997.

[9] Driessens Kurt. *Relational Reinforcement Learning*. PhD Thesis. In in Applied Science, Department of Computer Science, K.U.Leuven, Leuven, Belgium. (2004).

[10] Džeroski Sašso. Multi-Relational Data Mining: An Introduction. *SIGKDD Explorations Newsletter*. ACM Press. Vol. 5, No. 1. pp. 1-16, July 1 2003.

[11] Gauvain Bourgne, Amal El Fallah Segrouchni, Henry Soldano. *SMILE: Sound Multi-agent Incremental LEarning*, ACM, AAMAS'07, May 14-18-2007.

[12] Guerra Hernández Alejandro. *Learning in intentional BDI Multi-Agent Systems*. PhD thesis. Universit de Paris 13. Institut Galile. LIPN - Laboratoire d'Informatique de Paris Nord. Villetaneuse, France, (2003).

[13] Guerra Hernández Alejandro, Amal El Fallah-Seghrouchni, Henry Soldano (2001). *BDI Multi-agent Learning Based on First-Order Induction of Logical Decision Trees*. In: Intelligent Agent Technology: Research and Development, Proceedings of the 2nd Asia-Pacific Conference on IAT, World Scientific, p. 160-169, Maebashi, Japan, November 2001.

[14] Guerra Hernández Alejandro, Amal El Fallah-Seghrouchni, Henry Soldano (2004). *Learning on BDI Multiagent Systems*. En: J. Dix, A. Leite (eds). Computational Logic and Multiagent Systems: 4th International Workshop CLIMA IV, Fort Lauderdale, FL., USA, January 6-7, 2004. Revised and Selected Papers. Lecture Notes in Computer Science, Vol. 3259, Springer Verlag, Heidelberg, Germany.

[15] Guerra Hernández Alejandro, Amal El Fallah-Seghrouchni, Henry Soldano (2004). *Distributed Learning in Intentional BDI Multi-Agent Systems* In: Mexican International Conference in Computer Science 2004, ENC-04. IEEE Computer Society Press, Colima, Mxico, September 20-24, 2004.

[16] Guerra Hernández Alejandro, Amal El-Fallah-Seghrouchni, Henry Soldano (2005). *On Learning Intentionally*. Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial. No 25. (2005) p.9-18.

[17] Guerra Hernández Alejandro, Ortiz Hernández Gustavo. *Towards BDI sapient agents: learning intentionally*. Toward Computational Sapience: Principles and Systems. (Eds. Rene V. Mayorga and Dr Leonid I. Perlovsky). Springer-Verlag (In press).

[18] Guerra Hernández Alejandro, Ortiz Hernández Gustavo, Luna Ramírez Wulfrano Arturo. *Jason Smiles*. 6th Mexican International Conference on Artificial Intelligence (MICAI). (en revisión).

[19] Han J., Kamber M., *Data Mining: Concepts and Techniques*, Morgan Kaufmann Publishers, (2000).

[20] Hilde Adé, De Raedt Luc and Bruynooghe Maurice. *Machine Learning*, Kluwer Academic Publishers: Boston. Vol. 20, Nos. 1-2. pp. 119-154, July 1995.

[21] Michie, D., Spiegelhalter D. J., Taylor C. C. (Eds.) *Machine learning, neural and statistical classification*. Gran Bretaa: Ellis Horwood. (1994).

[22] Mitchell Tom, *Machine Learning*, McGraw-Hill, (1997).

[23] Moreira, A. F., and Bordini, R. H. An operational semantics for a BDI agent-oriented programming language. In Meyer, J-J. C. and Wooldrige, M.J. (eds). *Procedings of the Workshop on Logics for Angent-Based Systems* (LABS-02), held in conjunction with the Eighth International Conference on Principles of Knowledge Representation and Reasoning (KR2002), April 22-25, Toulouse, France, 45-59. 2002.

[24] Muggleton S.H. Inductive Logic Programming. In *New Generation Computing*, Vol. 8, No. 4, pp. 295-318. 1991.

[25] Muggleton S.H. and De Raedt L. Inductive Logic Programming: Theory and Methods. *Journal of Logic Programming*. Vol. 19, No. 20. pp. 629-679. 1994.

[26] Piatetsky-Shapiro G. and Frawley W.J. (eds.), *Knowledge Discovery in Databases: An overview*, AAAI Press, Menlo Park, California.

[27] Rao Anand S. . AgentSpeak(L): BDI Agents speak out in a logical coputable languaje. En Walter Van de Velde and John Perram (eds). *Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World* (MAAMAW'96), 22-25 Enero. Eindhoven, Holanda. No. 1038 en Lecture Notes in Artificial Intelligence, pp. 42-55,Springer-Verlag, Londres. 1996.

[28] Rao Anand S. and Georgeff M. P. BDI-agents: from theory to practice. *Proceedings of the First Intl. Conference on Multiagent Systems*. San Francisco, (1995).

[29] Russell Stuart & Norvig Peter.,*Inteligencia artificial: Un enfoque moderno*. 2a. ed. México: Prentice Hall Hispanoamericana. (1996).

[30] Struyf Jan. PhD Thesis. *Techniques for improving the efficiency of inductive logic programming in the context of data mining*. Catholic University Leuven. 2004.

[31] Shan-Hwei Nienhuys-Cheng & Ronald de Wolf. *Foundations of Inductive Logic Programming*. Lecture Notes in Computer Science: 1228, Lecture Notes in Artificial Intelligence. Springer (1997).

[32] Utgoff Paul E., Incremental Induction of Decision, Trees, *Machine Learning*, vol. 4, pp, 161-186. 1989.

[33] Van Laer Wim & De Raedt Luc, How to Upgrade Propositional Learners to First Order Logic: a Case Study, *Lecture Notes in Computer Science*, (2001), vol. 2049, pp. 102-126.

[34] Witten H. Ian & Eibe Frank, *Data Mining. Practical Machine Learning Tools and Techniques* (2nd. Ed.), USA: Elsevier ,(2005).

[35] Wooldridge, M.. *An Introduction to MultiAgent Systems*. John Wiley & Sons Ltd. (2002).

[36] Wooldridge M. and Jennings N.R. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review* 10(2), 1995.