UNIVERSIDAD AUTÓNOMA METROPOLITANA

Aprendizaje Maquinal Multivalores

Para obtener el grado de

MAESTRO EN CIENCIAS
(CIENCIAS Y TECNOLOGÍAS DE LA INFORMACIÓN)
presenta:
Lic. Orlando Muñoz Texzocotetla

Asesor:
Dr. René Mac Kinney Romero

Sinodales:
Presidente: Dr. Miguel Ángel Gutiérrez Andrade
Secretario: Dr. René Mac Kinney Romero
Vocal: Dr. Eduardo Morales Manzanares
Vocal: Dr. John Henry Goddard Close

Abril 2011

Resumen

El aprendizaje puede ser visto como la búsqueda de aquella hipótesis que satisfaga ciertos criterios de calidad (p.e. que sea consistente) [12]. Dicha búsqueda es realizada por un *aprendiz*, el cual puede describirse a partir de tres elementos muy importantes: el espacio de búsqueda, la estrategia de búsqueda y la heurística de búsqueda.

De los tres elementos anteriores el espacio de búsqueda es el conjunto de hipótesis donde el aprendiz realiza su búsqueda. En programación lógica inductiva (ILP) está determinado por el lenguaje de los programas lógicos, es decir por la lógica de primer orden. En el caso de los árboles de decisión el lenguaje es el de la lógica proposicional.

Por otro lado el prejuicio de lenguaje en ILP es el conjunto de restricciones que determinan la forma sintáctica de las hipótesis que pueden formar parte del espacio de búsqueda. Si el prejuicio es muy fuerte entonces el espacio de búsqueda se hace más pequeño y la búsqueda más eficiente, sin embargo entre más restricciones se usen para formar las hipótesis es más probable que la hipótesis final no represente una solución adecuada para la relación objetivo. Por ejemplo si el prejuicio de lenguaje restringe el uso de la literal de la cabeza en el cuerpo de las cláusulas, entonces la hipótesis final no podrá generalizar de manera adecuada una relación recursiva.

Cada algoritmo de ILP define su propio prejuicio de lenguaje con el objetivo de que las hipótesis construidas generalicen eficientemente cada relación objetivo. Sin embargo, para construir las hipótesis, los algoritmos ILP actuales prueban un solo valor por cada aparición del atributo que representa, por lo que es probable que las hipótesis se construyan con gran cantidad de cláusulas. Entre más grande sea la base de conocimientos, es más probable que la hipótesis final esté construida con más cláusulas, por lo tanto es más difícil de interpretar.

De este inconveniente nace la necesidad de crear nuevas cláusulas que permitan crear hipótesis con menos cláusulas, que además aporten información, que a pesar de estar contenida en el conocimiento previo, no es utilizada en el proceso de aprendizaje de los algoritmos de ILP. Para ello proponemos el siguiente método de cuatro pasos:

- Extracción de información contenida en el conocimiento previo de cada problema ILP. Para ello se utilizarán algoritmos inductores de árboles de decisión.

- Con la información obtenida en el paso anterior se crearán nuevas cláusulas (cuyos atributos presenten más de un valor a la vez).

- Las nuevas cláusulas se añadirán al conocimiento previo del problema ILP correspondiente, de manera que estas cláusulas serán utilizadas, por cada algoritmo ILP, en el proceso de aprendizaje.

- Aprendizaje y análisis. Después de añadir las nuevas cláusulas al conocimiento previo de cada problema ILP, se realizará la ejecución de algoritmos de ILP de manera que al final se comprobará si efectivamente con estas nuevas cláusulas es posible crear hipótesis con menos cláusulas.

De los tres elementos con los que podemos describir a un aprendiz, en este documento trabajamos sobre el espacio de búsqueda, ya que lo enriquecemos con información obtenida con algoritmos inductores de árboles de decisión. Esto permite realizar aprendizaje multivalores en ILP, ya que las cláusulas añadidas al conocimiento previo permiten utilizar un conjunto de valores por cada atributo elegido.

Dedicado a...

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

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

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

Agradecimientos

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

También quiero dar las gracias a los profesores Dr. Eduardo Morales Manzanares, Dr. John Goddard Close y Dr. Miguel Ángel Gutiérrez Andrade por sus comentarios e ideas aportados para el mejoramiento de mi tesis.

Quiero agradecer también a la Universidad Autónoma Metropolitana por las oportunidades brindadas para la realización, no solo de esta tesis sino de mi licenciatura y mi maestría.

Por último doy gracias al Consejo Nacional de Ciencia y Tecnología (CONACyT) por el apoyo y recursos brindados para la realización de la maestría y por supuesto de esta tesis.

Índice

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

La inteligencia artificial (IA), es un área de las ciencias y tecnologías de la información (CyTI) que tiene como objetivo crear sistemas computacionales inteligentes. Existen principalmente, entre otras, cuatro definiciones para IA que presentamos a continuación [29]:

1. Algunos definen la IA como la rama de las CyTI que crean sistemas que piensan como humanos.
2. Otros autores indican que la IA es la ciencia que crea máquinas que actúan como humanos.
3. También se define como la ciencia que crea sistemas que actúan racionalmente.
4. Por último otros expertos la definen como la ciencia que crea sistemas que piensan racionalmente.

Desde la perspectiva de la definición 1, Alan Turing diseñó una prueba con la intención de proporcionar una definición operacional y satisfactoria de inteligencia artificial [34]. Esta prueba se compone de dos personas A y B respectivamente, y una computadora. La persona A será un interrogador que conversará tanto con B como con la computadora. A no sabe quien le responde sus preguntas, ya que B y la computadora están en habitaciones cubiertas. Si al término del interrogatorio, el individuo A no logra identificar la computadora de la persona B, entonces se dirá que la computadora piensa, es decir, tiene inteligencia artificial. Las características principales que debe tener la máquina son: procesamiento del lenguaje natural, representación del conocimiento, razonamiento automático y aprendizaje maquinal (o automático).

De las características anteriores, el aprendizaje automático es el que nos interesa, pues el trabajo desarrollado en esta tesis concierne a dos técnicas de aprendizaje maquinal:

- **Árboles de decisión.** En este proceso los algoritmos inducen, a partir de un conjunto de ejemplos, un árbol de decisiones que representa la hipótesis que explicará el concepto objetivo [24]. En el capítulo 3 profundizaremos en esta técnica de aprendizaje maquinal.
- **Programación lógica inductiva (ILP).** Este método realiza el aprendizaje utilizando programación lógica. Stephen Muggleton define este método como la intersección entre la programación lógica y el aprendizaje maquinal [22]. En el capítulo 4 expondremos este aprendizaje con más detalle.

En las siguientes secciones definimos el problema central de este trabajo, así como el objetivo principal y la estructura de esta tesis.

1.1 Descripción y definición del problema

El concepto de aprendizaje puede ser visto como la búsqueda de una hipótesis que satisfaga ciertos criterios de calidad [19]. Dicha búsqueda es realizada por un sistema computacional llamado *aprendiz*, el cual puede ser descrito a partir de la estructura de su espacio de búsqueda, de su heurística de búsqueda y de su estrategia de búsqueda [12].

El espacio de búsqueda es el conjunto de hipótesis donde el aprendiz realiza la búsqueda, en ILP está determinado por el lenguaje de los programas lógicos, los cuales, están formados por cláusulas de programa de la forma $T \leftarrow Q$[1], donde T es un átomo $p(X_1, \ldots, X_n)$ y Q es una conjunción de literales $L_1, \ldots, L_m$ (en el apéndice A se describen con más detalle las principales definiciones de la lógica de primer orden). Por otro lado la forma de las hipótesis del espacio de búsqueda, está restringida sintácticamente por el prejuicio de lenguaje. Este prejuicio determina las cláusulas, que pueden formar parte de cada hipótesis, a partir del vocabulario de los predicados, de los símbolos de función y de las constantes declaradas en el conocimiento previo[2] [12].

Cuando el prejuicio de lenguaje es muy fuerte, lenguaje poco expresivo, el espacio de búsqueda se vuelve más pequeño, y aunque la búsqueda es más eficiente, es más probable que la hipótesis encontrada no represente una solución adecuada para el problema que se intenta resolver. Por ejemplo si el prejuicio de lenguaje impide el uso de la literal de la cabeza en el cuerpo de las cláusulas, entonces ninguna hipótesis del espacio de búsqueda representará la solución a un problema cuya naturaleza es recursiva.

En cada algoritmo ILP se define un prejuicio de lenguaje, con el objetivo de que las hipótesis se construyan con la mayor expresividad posible, y de que la hipótesis encontrada represente una mejor generalización. Sin embargo, para construir las hipótesis, los algoritmos ILP actuales prueban un solo valor por cada aparición del atributo que representa, por lo que es probable que las hipótesis se construyan con una gran cantidad de cláusulas. Entre más grande sea la base de conocimientos, es más probable que la hipótesis final esté construida con más cláusulas, por lo tanto es más difícil de interpretar. Para describir este comportamiento presentamos un problema de clasificación planteado por Stephen Muggleton [21]. Este problema consiste en clasificar un conjunto de animales en las siguientes clases: ave, reptil, mamífero y pez. La literal objetivo es:

$$clase\,(Animal, Clase) \leftarrow$$

El aprendiz creará cada hipótesis (conjunto de reglas) con la información de cada animal acerca de su habitat, su cubierta, su número de patas, si toma leche, si es homeotérmico, si pone huevos o si tiene branquias. Las literales que representan la información mencionada anteriormente, conocimiento previo, se muestran en la lista siguiente:

[1] Cada hipótesis en ILP es un programa lógico el cual consiste de un conjunto de cláusulas de programa.

[2] El conocimiento previo, como veremos en el capítulo 4, es la información que de antemano debe *saber* el aprendiz para realizar su proceso de aprendizaje.

- $habitat\,(Animal, Habitat)$. Hay cuatro: $\{tierra, agua, aire, cueva\}$.
- $tiene\_cubierta\,(Animal, Cubierta)$. Hay cuatro $\{pelo, plumas, escamas, ninguna\}$.
- $tiene\_patas\,(Animal, Patas)$. Los números de patas son: $\{0, 2, 4\}$.
- $toma\_leche\,(Animal)$.
- $homeotermico\,(Animal)$.
- $pone\_huevos\,(Animal)$.
- $tiene\_branquias\,(Animal)$.

En este caso el algoritmo *Progol*, uno de los algoritmos ILP creado e implementado por Stephen Muggleton [22], devuelve la siguiente hipótesis con cinco reglas:

1. $clase\,(serpiente, reptil)$.
2. $clase\,(A, mamifero) \leftarrow toma\_leche\,(A)$.
3. $clase\,(A, pez) \leftarrow tiene\_branquias\,(A)$.
4. $clase\,(A, ave) \leftarrow tiene\_cubierta\,(A, plumas)$.
5. $clase\,(A, reptil) \leftarrow tiene\_cubierta\,(A, escamas)\,, tiene\_patas\,(A, 4)$.

Para crear una hipótesis como la anterior, los algoritmos ILP utilizan cláusulas cuyas literales declaran un valor por cada atributo. Por ejemplo, en la cláusula cuatro de la hipótesis anterior, el segundo argumento de la literal *tiene_cubierta*, que llamaremos *Cubierta*, es un atributo categórico con cuatro posibles valores: escamas, pelo, plumas y ninguna. Cada vez que aparece la literal *tiene_cubierta* en alguna de las cláusulas que forman las hipótesis del espacio de búsqueda, el atributo *Cubierta* presenta sólo uno de sus valores. A este tipo de cláusulas las llamaremos *univalor*. Es decir, si todos los argumentos de cada literal presente en el cuerpo de una cláusula, declaran un sólo valor, llamaremos a esta *cláusula univalor*. Si algún argumento presenta dos o más valores, entonces la llamaremos *cláusula multivalores*.

Notemos que la cláusula univalor cuatro de la hipótesis mostrada anteriormente, nos indica que un animal A es un ave si su cubierta es *plumas*. Sin embargo es evidente que si las aves tuvieran dos o más tipos de cubierta, como característica propia, la hipótesis final tendría que declarar una cláusula por cada tipo de cubierta como en las reglas de la ecuación 1.1.

$$\begin{array}{l} \vdots \\ clase\,(A, ave) \leftarrow tiene\_cubierta\,(A, tipo_1)\,. \\ \vdots \\ clase\,(A, ave) \leftarrow tiene\_cubierta\,(A, tipo_n)\,. \\ \vdots \end{array} \tag{1.1}$$

En el caso de los atributos numéricos el resultado es el mismo: una cláusula por cada valor del atributo; lo cual, no parece muy problemático cuando hay pocos atributos con pocos valores.

Sin embargo cuando el número de atributos y sus posibles valores se incrementa considerablemente, las hipótesis creadas pueden constar de una gran cantidad de reglas, lo que las vuelve más difíciles de interpretar.

De este inconveniente nace la necesidad de crear cláusulas multivalores (debilitar el prejuicio de lenguaje en ILP o mejorar su expresividad) que permitan a los algoritmos ILP crear hipótesis más pequeñas. Por lo tanto debemos plantear las siguientes cuestiones:

1. ¿Cómo crear los conjuntos de valores para cada atributo seleccionado?
2. ¿Cuántos conjuntos de valores crear por cada atributo?
3. Una vez creados los conjuntos, ¿cómo crear con ellos las cláusulas multivalores?
4. ¿Se deben analizar todos los atributos existentes en el conocimiento previo para crear las cláusulas multivalores?
5. ¿Las cláusulas multivalores ayudarán a crear hipótesis con menos reglas?

1.2 Objetivo principal

Las respuestas a las preguntas formuladas en la sección 1.1 forman la base de la presente tesis. Nuestro objetivo principal es crear cláusulas multivalores que mejoren la expresividad del lenguaje en ILP. Con ello se espera que los algoritmos ILP construyan hipótesis con menos cláusulas y por lo tanto más fáciles de interpretar.

1.3 Estructura del documento

Para llevar a cabo nuestro objetivo primero es necesario definir formalmente qué es el aprendizaje maquinal, esto lo presentamos en el capítulo 2. En el 3 presentamos el algoritmo principal de árboles de decisión, esta técnica de aprendizaje maquinal nos permitirá obtener información del conocimiento previo de cada problema de ILP, con la cual haremos más expresivo su lenguaje. Ya que el objetivo principal concierne a la ILP, presentamos esta metodología en el capítulo 4.

Teniendo las bases necesarias acerca del aprendizaje maquinal y los métodos que utilizaremos, en el capítulo 5 formalizaremos nuestra propuesta para permitir a los algoritmos ILP crear cláusulas multivalores. Los algoritmos utilizados para ello los presentamos en el capítulo 6.

En el capítulo 7 presentamos los experimentos llevados a cabo para probar si las nuevas hipótesis creadas en ILP efectivamente contienen menos reglas o cláusulas. Por último en el capítulo 8 exponemos nuestras conclusiones y las propuestas a trabajos futuros que se puedan llevar a cabo para mejorar todavía más el nivel expresivo de la ILP.

Capítulo 2

Aprendizaje Maquinal

Normalmente vemos el aprendizaje como algo exclusivo de los seres vivos, en particular de los humanos. Sin embargo una de las tareas de la inteligencia artificial es buscar maneras de hacer que las computadoras también aprendan. Falta mucho por hacer para que un sistema aprenda por sí mismo, al nivel en que lo hacemos nosotros, pero las bases están sentadas y se trabaja continuamente para mejorar las técnicas y algoritmos de aprendizaje.

Antes de presentar algunos de los métodos de aprendizaje, sus algoritmos y la presente propuesta, es necesario definir formalmente qué es el aprendizaje, qué características tiene un sistema que aprende, en que áreas es útil automatizar el aprendizaje, qué técnicas existen para ello y cómo es que un sistema lo lleva a cabo. Respecto a lo anterior en este capítulo exponemos:

- Definición formal de *aprendizaje maquinal (AM)*.
- Diseño de un sistema de AM.
- Aplicaciones de AM.
- *Aprendizaje de Conceptos,* y algoritmos de aprendizaje de conceptos..
- Prejuicio de lenguaje, de búsqueda y de prevención de sobreentrenamiento.

2.1 Definición de Aprendizaje Maquinal

Si preguntamos a varias personas qué se puede hacer con una computadora, la mayoría seguramente responderá: almacenar información, comunicarse con otras personas, realizar operaciones matemáticas, escribir textos, leer libros, ver sitios web, escuchar música, etc. Sin embargo uno de los objetivos de las CyTI, y más específicamente de la *inteligencia artificial* (disciplina de las CyTI), es ir más allá de estas *simples* tareas.

La inteligencia artificial es una especialidad que busca crear sistemas inteligentes, es decir, darle a las computadoras la capacidad de adaptarse a un entorno versátil, y para poder adaptarse deben tener la capacidad de aprender [1].

De la definición anterior podemos notar dos cosas importantes:

1. **El sistema se encuentra en un ambiente cambiante.** Por supuesto cualquier sistema que se desarrolla hoy en día se encuentra en un entorno versátil con el que tiene que lidiar. Pensemos en un sistema que está dedicado a detectar usos ilegales de las tarjetas de crédito (p.e. clonación), para ello se basa en la información obtenida de retiros y compras realizadas con dichas tarjetas. El sistema puede obtener de toda esta información ciertos patrones que indican que alguien ha clonado una tarjeta (demasiadas compras en lugares muy distintos por ejemplo). Cuando el sistema identifica estos patrones entonces ha aprendido a identificar el uso ilegal de las tarjetas. Sin embargo las personas que clonan tarjetas siempre buscan maneras más sofisticadas para realizar la actividad ilícita sin ser detectados, y cada vez el sistema deberá aprender a identificar los nuevos patrones que indiquen la falsificación de tarjetas de crédito.

2. **Aprendizaje.** El sistema debe tener la capacidad de aprender de ese entorno para poder mejorar su desempeño en la tarea para la que fue creado. En el ejemplo del párrafo anterior el sistema debió identificar los patrones que indican la clonación de una tarjeta de crédito.

Podemos decir entonces que si queremos darle inteligencia a un sistema, debemos proporcionarle la capacidad de *aprender*. A continuación ampliamos este concepto, que es parte central de esta tesis.

Una de las definiciones de la "Real Academia española (RAE)" nos dice que aprender es: "Adquirir el conocimiento de algo por medio del estudio o de la experiencia". Aunque no sólo el conocimiento se adquiere de la experiencia, también obtenemos y mejoramos habilidades o destrezas. Para ello existe más de un tipo de aprendizaje, y dos de los más importantes son:

- **Aprendizaje deductivo.** Llamado también inferencia deductiva, este tipo de aprendizaje toma un conjunto de hechos (premisas) y a partir de estos infiere una regla particular (conclusión). Este tipo de aprendizaje va de lo general a lo particular. En lógica este tipo de aprendizaje se lleva a cabo utilizando reglas de inferencia como el *modus ponens, modus tollens, etc.*

- **Aprendizaje inductivo.** Conocido como inferencia inductiva, a diferencia del anterior este tipo de aprendizaje va de lo particular a lo general, es decir, que a partir de un conjunto dado de ejemplos se infiere un conjunto de reglas que generalizan algún concepto, estas reglas son llamadas *hipótesis*. Este es el tipo de aprendizaje que realizamos más a menudo puesto que desde niños obtenemos todo tipo de ejemplos para poder generalizar, a partir de estos, una regla o una serie de reglas. Por ejemplo, cuando los niños aprenden a hablar reciben una extraordinaria cantidad de ejemplos, con sus familiares, en la televisión, con otros niños, etc. Un ejemplo de regla que generalizan los niños pequeños (aunque no de manera conciente) a partir de toda esta información recibida es que va primero el artículo y después el sustantivo: el gato, el perro, el árbol; y no al revés.

Hay otros tipos de aprendizaje que están fuera del alcance de este trabajo, sin embargo no dejan de ser importantes: aprendizaje por reforzamiento, aprendizaje por observación, aprendizaje abductivo, aprendizaje por analogías, etc.

Ya que tenemos una visión más clara de lo que significa *aprender*, definimos formalmente tal concepto en el entorno de la inteligencia artificial.

DEFINICIÓN 2.1 *"Se dice que un programa de computadora aprende de la experiencia **E**, respecto a una tarea **T** y una medida de rendimiento **P**, si su rendimiento en la tarea **T**, medida por **P** mejora con la experiencia **E**", [19].*

De lo anterior se tiene que existe un área dentro de la inteligencia artificial, llamada *aprendizaje maquinal*, que diseña métodos y algoritmos que le dan a los sistemas la capacidad de aprender.

Los conceptos anteriores, incluyendo la definición 2.1, nos permiten realizarnos las siguientes interrogantes:

- ¿Cómo definir y representar la experiencia que adquiere un sistema?
- ¿Cómo medir el rendimiento que tiene el sistema al realizar dicha tarea?
- ¿Cómo saber si el sistema de aprendizaje ha mejorado su rendimiento al realizar alguna tarea?
- ¿Cómo representar el conocimiento que el sistema adquiere?

En el resto del capítulo se pretende contestar a éstas preguntas.

Antes de continuar es necesario aclarar que el tipo de aprendizaje que cubriremos en esta tesis es el inductivo. De manera que los métodos expuestos aquí inducen hipótesis a partir de ejemplos.

2.2 Diseño de un sistema de Aprendizaje Maquinal

Como cualquier sistema, los de AM no se pueden crear de la nada. Primero es necesario realizar un diseño que cubra el requerimiento principal: *aprender*.

En [19], Tom Mitchell presenta las bases para diseñar e implementar un sistema de AM. A continuación presentamos el desarrollo de cada uno de estos elementos, en el diseño de un sistema que aprende a jugar ajedrez.

Tarea *T*

El primer paso es definir explícitamente qué es lo que el sistema va a aprender. En nuesto caso el requerimiento principal es aprender a jugar ajedrez. La tarea la denotamos *T*.

Medida de rendimiento *P*

Este punto se refiere a la manera de saber si el sistema está aprendiendo, y con qué precisión. Para ello se debe definir una métrica que nos indique el nivel de aprendizaje que ha logrado el sistema. En nuestro ejemplo, esta métrica se basa en el porcentaje de juegos ganados. Sin embargo no es la única manera de medir qué tanto está aprendiendo el sistema. La medida de rendimiento, denotada por *P*, depende del creador del sistema y del problema de aprendizaje.

Experiencia de entrenamiento E

Este elemento define el origen de la experiencia. Para nuestro ejemplo, el sistema aprenderá más al jugar contra sí mismo. Por supuesto, dependiendo del problema a tratar, para cada sistema se debe definir el proceso de entrenamiento.

Ya que el tipo de aprendizaje que tratamos en esta tesis es inductivo, al determinar la experiencia de entrenamiento *E*, necesitamos tomar en cuenta los siguientes aspectos, respecto a los ejemplos utilizados, para inducir las hipótesis: tipo de retroalimentación, control de los ejemplos y distribución de ejemplos.

Retroalimentación directa e indirecta. En la retroalimentación directa le indicamos al sistema cada una de las reglas que debe utilizar para llevar a cabo su tarea. En nuestro caso consiste en indicarle cada uno de los movimientos permitidos para cada una de las piezas del ajedrez. Por ejemplo en la figura 2.1 se muestran los posibles movimientos de la reina en un tablero de ajedrez.

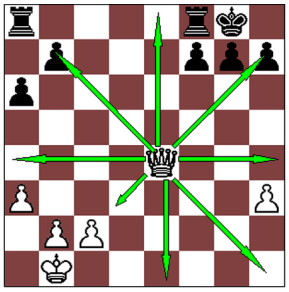


Figura 2.1: Retroalimentación directa

La retroalimentación indirecta consiste en proporcionarle al sistema un conjunto de estados iniciales y finales, de manera que el mismo sistema deba inferir las reglas válidas para realizar su tarea. En nuestro ejemplo, los estados iniciales se refieren al estado del tablero antes de que comience el juego. Por otro lado los estados finales son tableros donde el juego ha terminado, sin importar si el sistema ganó. En la figura 2.2 podemos ver un estado inicial (un juego nuevo) y un estado final (un partido ganado por el sistema representado por las piezas blancas).

Cada estado intermedio, entre el inicial y el final, forma parte de un conjunto de movimientos válidos para cada pieza del ajedrez. Estos movimientos, son los que el sistema deberá inferir a partir del conjunto de estados iniciales y finales. De manera general, el estado de un tablero, consiste en las posiciones en que se encuentran las piezas, blancas y negras, en un determinado momento del juego.

En este último caso, es necesario asignar un crédito por cada movimiento. A un movimiento correcto se le asigna un crédito positivo, y a uno incorrecto se le asigna un crédito negativo. De esta manera se determina si el sistema está jugando bien. Esta asignación de valores hace de la retroalimentación indirecta un método más complejo que la directa.

Grado de control de los ejemplos. Al conjunto de ejemplos que utilizará el sistema para aprender, se le denomina *conjunto de entrenamiento*. Cabe aclarar que al proceso de aprendizaje o de inducción de la hipótesis buscada, le llamaremos *entrenamiento*. De acuerdo al grado de control que tiene el sistema sobre este conjunto, podemos distinguir entre tres tipos:

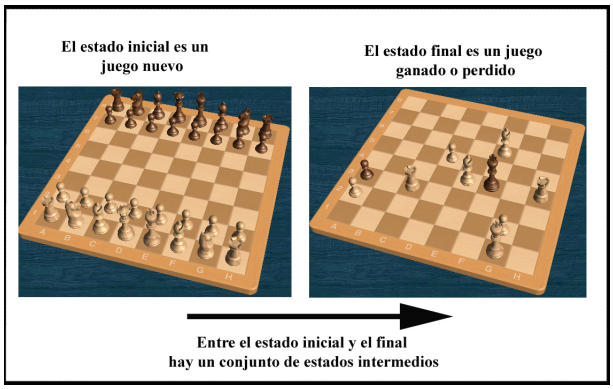


Figura 2.2: Retroalimentación indirecta

- **Sin Control.** Un sistema no tiene control sobre los ejemplos cuando éstos son proporcionados por un maestro, el cuál, puede ser un usuario u otro sistema.

- **Control medio.** Este tipo de control lo tiene aquel sistema que crea ejemplos, pero un maestro es el que indica si estos son apropiados para que el aprendizaje del sistema sea exitoso.

- **Control total.** En este caso el sistema tiene el control total de los ejemplos, de manera que el sistema los crea, e infiere si son adecuados para un buen aprendizaje. Cuando un sistema tiene este grado de control decimos que está realizando un aprendizaje *no supervisado*, ya que ningún maestro lo guía durante el proceso de aprendizaje. Los dos tipos de control anteriores corresponden a un aprendizaje *supervisado*.

Distribución de ejemplos de entrenamiento. Además del conjunto de entrenamiento, se debe contar con otro que es para prueba (conjunto de prueba). Éste último se utiliza para probar si el sistema aprendió y en qué medida. Se dice además que si el conjunto de entrenamiento tiene la misma distribución que la del conjunto de prueba, entonces el aprendizaje realizado por el sistema es más confiable.

Respecto a las distribuciones de los conjuntos de entrenamiento y de prueba, es importante mencionar un problema que puede surgir durante el proceso de aprendizaje: *sobreentrenamiento (en inglés overfitting)*. En su libro Tom Mitchell [19] lo define de la siguiente manera: "Dado un espacio de hipótesis H, $h \in H$ sobreajusta los datos de entrenamiento si $\exists h' \in H$ tal que h tiene un error pequeño sobre los ejemplos de entrenamiento, pero h' tiene un error más pequeño que h sobre la distribución completa de instancias." [19].

En la figura 2.3 el sobreentrenamiento se muestra como el crecimiento del error que hay entre las distribuciones del conjunto de ejemplos y el conjunto de prueba. En algún momento durante la ejecución del algoritmo de aprendizaje, la hipótesis es tal, que el error (diferencia entre la precisión de la hipótesis sobre el conjunto de entrenamiento y el conjunto de prueba) crece, en ese instante se está presentando el sobreentrenamiento, y la hipótesis obtenida generalizará los elementos del conjunto de ejemplos, pero no el conjunto de prueba.

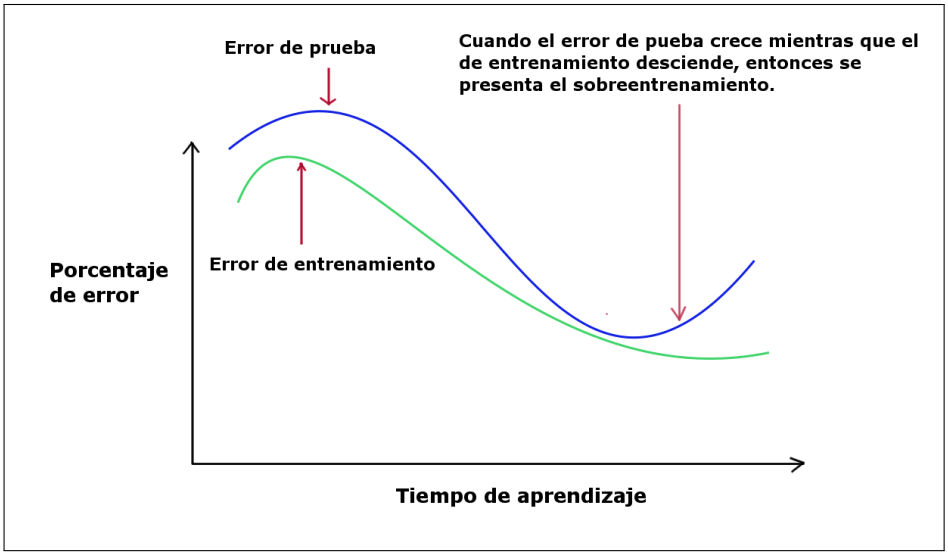


Figura 2.3: Desarrollo del aprendizaje y surgimiento del sobreentrenamiento

Función objetivo y su representación

Hasta ahora hemos especificado tres elementos importantes dentro de un sistema de aprendizaje: La tarea *T*, la medida de rendimiento *P* y la experiencia *E*. Sin embargo es en este este punto donde surgen las siguientes cuestiones: ¿Cómo se implementa? ¿Cómo se representa la información? ¿Cómo aprende el sistema? ¿Cómo representamos los datos de entrenamiento y los de prueba?

Para responder a lo anterior necesitamos encontrar un modelo formal que nos permita representar el proceso de aprendizaje, el conjunto de entrenamiento y el conjunto de prueba.

Tomemos en cuenta que el entrenamiento es la búsqueda de aquella hipótesis que explica aquello que se quiere aprender. En nuestro ejemplo, buscamos el conjunto de valores que permita al sistema ganar el mayor número posible de partidas de ajedrez. Este conjunto de valores representa la hipótesis buscada, la cual permitirá, en cada juego, realizar los movimientos necesarios para ganar.

Con esta última observación, logramos ver que el problema de aprendizaje podemos modelarlo como un problema de búsqueda, cuyo objetivo es encontrar el conjunto de valores que nos permitan representar una hipótesis. Este tipo de búsqueda se puede realizar con algoritmos de optimización que devuelven la mejor hipótesis posible.

Muchas veces el número de posibles combinaciones de valores, o de hipótesis, es demasiado grande. En ocasiones infinito. Para un juego de ajedrez se calcula que hay 10^{40} tableros. Esto trae como consecuencia que la búsqueda no sea trivial, por lo que se hace necesario el uso de algoritmos cada vez más eficientes y efectivos. Estos algoritmos encuentran la hipótesis en tiempo polinomial o incluso menor.

Ya que hemos llegado a la conclusión de que podemos modelar el proceso de aprendizaje como un problema de optimización, podemos definir sus elementos:

DEFINICIÓN 2.2 *Una **función objetivo** (ec. 2.1) es aquella que está sujeta a optimización. Esta función mapea del conjunto de los posibles estados del problema al conjunto de los números reales. El conjunto V puede contener cualquier tipo de elementos dados, p.e. números, listas, o en nuestro caso los estados posibles del tablero de ajedrez, dependiendo del problema de optimización. Las funciones objetivo no necesariamente son expresiones matemáticas [35].*

$$V : B \rightarrow \mathbb{R} \tag{2.1}$$

Donde:

- *B* es el conjunto de todos los estados posibles del problema a optimizar (posibles configuraciones del tablero de ajedrez en nuestro ejemplo).
- $\mathbb{R}$ es el conjunto de los reales.

El objetivo es encontrar $t_0 \in B$, en nuestro ejemplo t_0 representa los valores que definen un estado del tablero de ajedrez, tal que:

- $V(t_0) \leq V(t) \forall t \in B$, en caso de minimización
- $V(t_0) \geq V(t) \forall t \in B$, en caso de maximización

El problema a tratar determina si hay que maximizar o minimizar una función.

El dominio *V* de la función se le llama espacio de búsqueda (en AM este conjunto es llamado *espacio de hipótesis*) y cada elemento de *V* es una solución factible.

Hemos definido que debemos tener una función objetivo, pero aún no contestamos a la cuestión sobre cómo representamos los datos. Para ello debemos elegir una función que represente el problema de aprendizaje planteado. Algunas funciones que podemos utilizar son las lineales, las utilizadas en redes neuronales, polinomiales, etc. Para nuestro sistema podemos utilizar la ecuación 2.2:

$$V(t) = W_0 + W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 \tag{2.2}$$

Donde:

- Sea B el conjunto de todos los estados posibles del tablero de ajedrez.
- $t \in B$ y representa una configuración del tablero de ajedrez en un momento que está determinado por las variables de la función.
- X_1 representa el número de piezas blancas sobre el tablero.
- X_2 representa el número de piezas negras sobre el tablero.
- X_3 representa el número de piezas negras amenazadas por las blancas.
- X_4 representa el número de piezas blancas amenazadas por las negras.

- Cada W_i representa un peso que permitirá determinar el ganador. Si los mayores pesos son los coeficientes de las variables que representan a las piezas blancas entonces la ecuación 2.2 indica que dicho jugador es el que tiene ventaja. De manera análoga si los coeficientes de las piezas negras tienen los mayores pesos estaremos hablando de que este jugador estará mas cerca de ganar la partida. Por lo tanto, el sistema podrá determinar de manera formal si un movimiento es bueno o no.

La función 2.2 nos permite representar el conocimiento o la información dentro del contexto de nuestro problema. Con ella podemos indicar cualquiera de los estados posibles del tablero de ajedrez, y además nos permite asignarle a cada uno de ellos un valor numérico con el que sabemos si es un buen elemento para el aprendizaje. Por ejemplo podríamos asignar $V(t_1) = 100$, si la configuración del tablero es un estado final, en el cuál, el sistema ganó el juego, entonces 100 sería la puntuación máxima. Por otro lado si asignamos $V(t_2) = -100$ como el menor valor posible para una configuración del tablero, significará entonces que cuando el sistema pierda, el valor de ese tablero será -100. Cuando el sistema queda tablas con su competidor (empate) podemos asignar $V(t_3) = 0$. Para saber si un estado del tablero es bueno o no para el aprendizaje, podríamos tomar como criterio si en el movimiento anterior perdimos o ganamos una pieza, más aún podríamos especificar el tipo de pieza que perdimos o ganamos, ya que en el ajedrez no todas tienen el mismo valor.

Hemos definido cómo vamos a representar el conocimiento por medio de una ecuación objetivo. Específicamente para nuestro problema con la ecuación 2.2. Ahora es necesario definir la representación del conjunto de entrenamiento.

DEFINICIÓN 2.3 *Sea B el conjunto de todos los estados posibles del problema de aprendizaje. Un **conjunto de entrenamiento** es una colección de parejas ordenadas $(t, V(t))$, donde $t \in B$. La primera entrada corresponde a un estado del problema, y la segunda entrada al valor asignado a ese estado.*

En la tabla 2.1 mostramos un conjunto de entrenamiento para el sistema que aprende a jugar ajedrez. Cada fila corresponde a un elemento del conjunto de entrenamiento. La primera columna, por cada fila, es un estado del tablero de ajedrez en un momento del juego. La segunda columna es el valor que se le asigna a ese ejemplo, en este caso ese valor se asigna según la ecuación 2.2; para este ejemplo 100 será el valor máximo, es decir cuando se ha ganado una partida, y -100 cuando se ha perdido. En caso de empate el valor será 0.

t	V(t)
5 blancas, 2 negras, 0 blancas amenazadas, 0 negras amenazadas	100
3 blancas, 2 negras, 0 blancas amenazadas, 0 negras amenazadas	0
2 blancas, 5 negras, 0 blancas amenazadas, 0 negras amenazadas	-100
8 blancas, 7 negras, 1 blanca amenazadas, 2 negras amenazadas	25

Tabla 2.1: Representación de un conjunto de entrenamiento

El conjunto de prueba tiene exactamente el mismo formato que el conjunto de entrenamiento.

Elección del algoritmo de entrenamiento

Ahora ¿cómo es que el sistema se entrena para aprender? La respuesta está en el algoritmo de búsqueda que usamos, en adelante le llamaremos algoritmo de entrenamiento. Debemos elegir el algoritmo de entrenamiento adecuado, para ello:

- Debe ser capaz de utilizar los datos de entrenamiento y de prueba tal y como fueron especificados.
- Debe ser capaz de encontrar la mejor solución posible, llamada hipótesis, es decir los valores más adecuados de cada W_i, de manera que el sistema pueda generalizar a partir de los ejemplos dados. Una manera de encontrar estos valores, es utilizando un algoritmo que encuentre los valores de cada W_i, tal que minimicen el error cuadrático (ver ecuación 2.3), es decir, $E \approx 0$. Este error es un estimador que permitirá determinar con que eficacia ha aprendido el sistema. Entre más pequeño sea el valor absoluto de E, mayor rendimiento del sistema de aprendizaje.

$$E = \sum \left(V_k(t) - V'(t)\right)^2 \tag{2.3}$$

  Donde:

  - $V_k(t)$ son los valores de cada elemento del conjunto de entrenamiento, evaluados con la función de entrenamiento.
  - $V'(t)$ son los valores encontrados por la hipótesis final (solución encontrada).

Para nuestro ejemplo proponemos el algoritmo de minimos cuadrados (LMS - Least Mean Square), éste permite hacer el ajuste de pesos necesarios para minimizar el error cuadrático, ver algoritmo (2.1:)

Algoritmo 2.1 *Algoritmo de Minimos Cuadrados*

```
begin
  Usa los pesos actuales W_i para calcular V(t)
  for i := 1 to No. de pesos W_i do
      W_i := W_i + (V_k(t) − V'(t))
  od
end
```

Por último el proceso de diseño e implementación de un sistema de aprendizaje maquinal lo podemos resumir como sigue:

1. **Tarea *T*:** Jugar ajedrez.
2. **Medida del rendimiento *P*:** Porcentaje de juegos ganados.

3. **Experiencia de entrenamiento *E*:** Jugar contra sí mismo.
4. **Función Objetivo:** $V : B \rightarrow \mathbb{R}$
5. **Representación de la función objetivo:** $V(t) = W_0 + W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4$
6. **Elección del algoritmo de entrenamiento:** LMS (Least Mean Square).

Los puntos 1-3 representan la especificación del problema de aprendizaje, y los puntos 4-6 representan la implementación del sistema.

Con este pequeño ejemplo ejemplificamos cómo se puede diseñar un sistema de aprendizaje maquinal, sin embargo el alcance de este trabajo no nos permite desarrollar toda su implementación, aunque en la siguiente sección presentamos los puntos más importantes que un sistema de aprendizaje realiza para llevar a cabo el proceso de búsqueda de hipótesis.

Esquema de sistema de AM

En párrafos anteriores hemos presentado los puntos principales que se siguen en el diseño e implementación de un sistema de aprendizaje maquinal; y en la figura 2.4 presentamos un esquema que representa los puntos principales que estos sistemas siguen durante su aprendizaje. Este esquema, propuesto por Mitchell [19], consta de 4 módulos donde:

- **Generalizador.** Crea una hipótesis inicial, y la envía al Generador de Experimentos.
- **Generador de experimentos.** Crea un estado inicial del problema y lo proporciona como dato de entrada del módulo Sistema de Rendimiento.
- **Sistema de rendimiento.** Resuelve el problema con el estado inicial presente y obtiene un conjunto de estados que representan el historial del problema desde el estado inicial al final, y envía este historial al módulo crítico.
- **Crítico.** A partir del historial crea un conjunto de datos de entrenamiento, que le sirven al Generalizador para crear una nueva hipótesis.

Figura 2.4: Estructura de un sistema de AM

Hemos definido hasta ahora qué es el aprendizaje maquinal y los elementos importantes de un sistema que aprende automáticamente, sin embargo de nada sirve si no tenemos idea de la clase de problemas que podemos resolver con estos algoritmos. En la sección 2.3 presentamos algunas aplicaciones del AM para darle más sentido a la construcción y diseño de técnicas de aprendizaje.

2.3 Aplicaciones del Aprendizaje Maquinal

Las aplicaciones del aprendizaje maquinal son muy variadas, a continuación mencionamos algunas de las más comunes:

- **Reconocimiento del habla.** Actualmente existen muchas aplicaciones de reconocimiento de voz que utilizan el AM para mejorar de manera continua la exactitud con la que reconocen las palabras pronunciadas por las personas.

- **Visión artificial.** Al igual que el reconocimiento del habla, el reconocimiento de imágenes utiliza el aprendizaje maquinal para reconocer cierto tipo de imágenes y el entrenamiento de estos sistemas permite mejorar la precisión con la que reconoce cada tipo de imagen (p.e. un rostro).

- **Control de robots.** Uno de los objetivos más importantes de la construcción de robots, es que sean autónomos, es decir, que tomen decisiones por sí mismos sobre qué operación realizar tomando en cuenta ciertos elementos en el entorno del robot. Para tomar estas decisiones es necesario que el robot aprenda del mismo entorno para poder determinar cada uno de sus maniobras, el AM le permite ir mejorando sus decisiones cada vez que obtenga más experiencia.

- **Ambiente bancario.** En el caso de las tarjetas de crédito, los bancos pueden utilizar sistemas de aprendizaje maquinal que aprenden a detectar ciertos patrones en el uso de las tarjetas, de manera que el sistema detecte clonaciones de tarjetas o usos ilegales.

En general cuando se necesita clasificar datos o tomar decisiones basadas en la experiencia, sin importar la disciplina que lo requiera, se puede utilizar técnicas de aprendizaje maquinal. Sin embargo tenemos hasta ahora una idea muy general de cómo se representa el conocimiento y cómo se realiza la búsqueda. En la siguiente sección y en los siguientes dos capítulos presentamos dos métodos de AM de manera más detallada.

2.4 Aprendizaje de Conceptos y Orden General a Específico

El *aprendizaje de conceptos* es una de las capacidades que tenemos como seres humanos. Desde que nacemos aprendemos desde los conceptos más sencillos (ave, silla, manzana, etc.) hasta los más complejos (función, conjunto o grupo en matemáticas).

El aprendizaje de conceptos también se puede lleva a cabo en el ámbito del AM, y para ello deben formalizarse antes ciertas definiciones.

En el resto de la presente sección presentaremos la definición formal de aprendizaje de conceptos, así como un ejemplo. Después describimos cómo se define un orden para las hipótesis, de tal manera que la búsqueda de la hipótesis óptima tenga sentido. Presentamos también dos algoritmos de AM que nos permiten ejemplificar la búsqueda de la mejor hipótesis. Por último exponemos el concepto de *prejuicio* que es referente a las decisiones que tomamos al crear un sistema de AM.

2.4.1 Definición de Aprendizaje de Conceptos

El aprendizaje de conceptos requiere que se definan los siguientes términos:

DEFINICIÓN 2.4 *"Un **concepto** es una función que mapea los elementos de un conjunto X a valores booleanos* $\{0, 1\}$*, la ecuación 2.4 representa un concepto", ver [19].*

$$c : X \rightarrow \{0, 1\} \tag{2.4}$$

Por ejemplo, una función que represente el concepto *ave* podría estar definida sobre el conjunto X =$\{$Todos los animales$\}$, cuyos valores booleanos podrían ser 1 para cualquier $x \in X$ tal que *x* sea ave, y 0 para cualquier $x \in X$ tal que *x* sea cualquier otro animal.

DEFINICIÓN 2.5 *"El **Aprendizaje de conceptos** es la inferencia de una función booleana a partir de un conjunto de ejemplos de entrenamiento", ver [19].*

Ejemplo: concepto *ave*

En el caso del aprendizaje del concepto ave, un sistema de aprendizaje de conceptos debe encontrar una hipótesis tal que infiera la función booleana (ec. 2.5).

$$c : X \rightarrow \{0, 1\} \tag{2.5}$$

Donde:

- X es el conjunto de todos los animales.
- $x = 1$ si $\forall x \in X$ es un ave.
- $x = 0$ si $\forall x \in X$ no es un ave.

Como vimos en la sección 2.2 es necesario crear un conjunto de entrenamiento para que el sistema pueda aprender a partir de éste. En el aprendizaje de conceptos un conjunto de entrenamiento se representa como en la tabla 2.2.

En la tabla 2.2 cada renglón representa un elemento del conjunto de entrenamiento, aquí lo denotamos con la tupla $(x, c(x))$. De esta tupla:

- *x* es una instancia o estado del problema, en este caso *x* representa los valores de los atributos (Alas, Pico,...), para los cuáles $x \in X$ es o no un ave.
- $c(x)$ es el valor asignado a *x*: $\{0, 1\}$ ó $\{No, Si\}$ respectivamente.

Alas	Pico	Plumas	Vuela	Pone huevos	EsAve
Sí	Sí	Sí	No	Sí	Sí
No	No	No	No	Sí	No
Sí	Sí	Sí	Sí	Sí	Sí
Sí	No	No	Sí	No	No

Tabla 2.2: Conjunto de entrenamiento para concepto *ave*

Además cada columna (excepto la última) representa un posible valor para el atributo correspondiente, y la última columna representa el valor de la función para cada renglón (elemento del conjunto de entrenamiento). En la sección 2.4.1 se muestra un ejemplo completo de este tipo de aprendizaje.

Ejemplo: *disfruta deporte*

A continuación presentamos un ejemplo, dentro del aprendizaje de conceptos, del planteamiento de un problema.

Problema: Plantear el siguiente ejercicio como uno de aprendizaje de conceptos.

Identificar los días en que una persona Q disfruta hacer su deporte acuático favorito. Se tomarán en cuenta los siguientes atributos que caracterizan un día cualquiera: Cielo, Temperatura, Humedad, Viento, Agua y el Pronóstico.

Planteamiento del problema: Las ideas principales para el planteamiento de este problema fueron tomadas del libro de Mitchell [19].

Como el aprendizaje de conceptos es un tipo de aprendizaje maquinal, entonces debemos definir cada uno de los puntos vistos en la sección 2.2.

1. Tarea T a realizar.
2. Medida de rendimiento P.
3. Experiencia de entrenamiento E.
4. Función Objetivo.
5. Representación de la función objetivo.
6. Elección del algoritmo de entrenamiento.

Tarea *T*: Aprender el concepto "Días en los que disfruta la persona Q de hacer su deporte acuático favorito", abreviaremos el concepto como *DisfrutaDeporte*.

Medida de rendimiento *P*: En este caso para saber si el sistema aprendió el concepto se debe utilizar un conjunto de prueba, el cuál, se le proporcionará al sistema una vez que se haya ejecutado el algoritmo de entrenamiento. El conjunto de prueba al igual que el de entrenamiento se puede representar como en la tabla 2.3. Entonces la medida de rendimiento será el porcentaje de elementos del conjunto de prueba que la hipótesis pueda representar. Más adelante veremos

cómo es que una hipótesis puede representar cada instancia del conjunto de entrenamiento o de prueba.

Experiencia de entrenamiento *E*: La experiencia la obtendrá el sistema al utilizar cada elemento del conjunto de entrenamiento en la ejecución del algoritmo de búsqueda, ver tabla 2.3. Recordemos que cada renglón de la tabla 2.3 representa un elemento del conjunto de entrenamiento definido anteriormente como la pareja ordenada $(x, c(x))$ donde x es una instancia de todos los posibles días que se pueden representar con los atributos dados y $c(x)$ es uno o cero.

No.	Cielo	Temperatura	Humedad	Viento	Agua	Pronóstico	DisfrutaDep.
1	Soleado	Calor	Normal	Fuerte	Caliente	Estable	Sí
2	Soleado	Calor	Alta	Fuerte	Caliente	Estable	Sí
3	Lluvioso	Frío	Alta	Fuerte	Caliente	Inestable	No
4	Soleado	Calor	Alta	Fuerte	Fresca	Inestable	Sí

Tabla 2.3: Conjunto de entrenamiento para concepto *DisfrutaDeporte*

Cada atributo puede tomar un valor de los siguientes:

- Cielo $\in$ {Soleado, Lluvioso}
- Temperatura $\in$ {Calor,Frío}
- Humedad $\in$ {Alta, Normal}
- Viento $\in$ {Fuerte,Débil}
- Agua $\in$ {Caliente, Fresca}
- Pronóstico $\in$ {Estable, Inestable}

Función objetivo y su representación: Esta función es la función booleana de la ecuación 2.6 que mapea desde cada instancia del conjunto de todos los posibles días, que podemos representar con los atributos mencionados anteriormente, a los valores $\{0, 1\}$.

$$c : X \rightarrow \{0, 1\} \tag{2.6}$$

Donde:

- X es el conjunto de todos los días representados por los atributos.
- $x = 1$ si $\forall x \in X$ si Q disfruta de su deporte acuático favorito.
- $x = 0$ si $\forall x \in X$ no Q no disfruta de su deporte acuático favorito.

Sólo falta por definir el algoritmo de entrenamiento para que el sistema busque la mejor hipótesis posible. Sin embargo hasta ahora no nos hemos ocupado en definir una representación de las hipótesis, por lo que es imposible realizar la búsqueda. En la sección 2.4.2 definimos una representación (no es la única) de las hipótesis para problemas de aprendizaje de conceptos.

2.4.2 Orden General a Específico

Para elegir o diseñar el algoritmo que buscará la hipótesis óptima, debemos definir cómo representar las hipótesis.

En aprendizaje de conceptos, podemos ver cada hipótesis como una conjunción de instancias de cada atributo del concepto objetivo (el que se va a aprender). Esta conjunción la denotaremos como una tupla, donde cada entrada representa a cada instancia, y cada coma equivale al operador de conjunción lógica $\wedge$. Por ejemplo, supongamos (hipótesis) que un día en que la persona Q disfruta de hacer su deporte acuático favorito es aquel en que el cielo está soleado, hay viento fuerte, el pronóstico será estable y los demás atributos como temperatura, humedad y agua no afectan, por lo que pueden tomar cualquier valor (representado por el símbolo "?") , entonces esta hipótesis la podemos representar como en la ecuación 2.7.

$$h_1 = (soleado, ?, ?, fuerte, ?, estable) \tag{2.7}$$

Otras hipótesis posibles están planteadas en las ecuaciones 2.8 y 2.9 respectivamente, donde la primer hipótesis indica que cada atributo puede tomar cualquier valor, y la segunda indica que ningún atributo puede tomar algún valor, esto lo denotamos con el símbolo "ϕ".

$$h_2 = (?, ?, ?, ?, ?, ?) \tag{2.8}$$

$$h_3 = (\phi, \phi, \phi, \phi, \phi, \phi) \tag{2.9}$$

Observaciones: Cada hipótesis tiene tantas entradas como atributos, en este caso seis. De las soluciones o hipótesis anteriores la número 2.8 indica que todos los atributos pueden tomar cualquier valor, y la 2.9 restringe todos los atributos, pues nos indica que para cualquier atributo no es factible ningún valor.

Así cada hipótesis se forma siguiendo los siguientes puntos:

- Cada entrada de la hipótesis corresponde a la misma columna en la tabla de datos de entrenamiento, así la primera entrada corresponde al atributo en la columna uno de la tabla de entrenamiento, la segunda entrada al atributo de la segunda columna, y así sucesivamente.
- El símbolo *?*, indica que en esa entrada el atributo puede tomar cualquier valor, lo que de alguna manera generaliza la hipótesis puesto que se pueden tomar más valores.
- El símbolo ϕ, indica que en esa entrada el atributo no puede tener ningún valor.
- Un valor del atributo en la entrada correspondiente significa que ese atributo sólo puede tener ese valor.

El conjunto de todas las posibles hipótesis que se pueden crear es llamado *espacio de hipótesis*, y lo denotaremos como *H*, además corresponde al dominio de la función objetivo (ver definición 2.2).

El objetivo del aprendizaje de conceptos es encontrar una hipótesis $h \in H$ tal que:

$$h(x) = c(x), \forall x \in X \tag{2.10}$$

Donde: *X* es el conjunto de todos los estados posibles del problema dado (p.e. todos los posibles días que se pueden formar con los valores de los atributos dados).

La ecuación 2.10 indica que la hipótesis óptima debe ser *consistente*, es decir, que represente a aquellos elementos del conjunto de entrenamiento cuyos valores $c(x) = 1$ (recordemos que $si = 1$ y $no = 0$). Además la hipótesis no debe representar a ningún elemento del conjunto de entrenamiento cuyos valores $c(x) = 0$. Veamos algunos ejemplos.

Si tenemos la instancia de la tabla 2.4:

Cielo	Temperatura	Humedad	Viento	Agua	Pronóstico	DisfrutaDeporte
Soleado	Calor	Normal	Fuerte	Caliente	Estable	Sí

Tabla 2.4: Instancia de un conjunto de entrenamiento

y la hipótesis: $h_2 = (soleado, ?, ?, ?, ?, ?)$

Podemos ver que $Temperatura = Soleado$ tanto en la hipótesis como en el ejemplo de entrenamiento. Para las restantes entradas en la hipótesis h_2 está indicando que pueden tomar cualquier valor, así que no importa qué valores tenga en el ejemplo. Por lo tanto h_2 *cubre* al ejemplo de la tabla 2.4.

Si la hipótesis es la siguiente:

$h_3 = (Soleado, Calor, Normal, Fuerte, Caliente, Inestable)$

Entonces no se cubre al ejemplo en 2.4, ya que a pesar de que los primeros cinco valores tanto en el ejemplo como en la instancia son los mismos, el último es diferente (en h_3 $Pronostico = Inestable$ y en el ejemplo $Pronostico = Estable$), y por lo tanto $h(x) \neq c(x)$.

DEFINICIÓN 2.6 *Una hipótesis $h \in H$ es **consistente** si:*

- $h(x) = c(x) \forall x \in X$*, tal que* $c(x) = 1$*. Es decir, h cubre los ejemplos positivos, y no a los negativos.*

Observación: Los ejemplos positivos son aquellos que tienen $c(x) = 1$ y los negativos son aquellos que $c(x) = 0$.

Al terminar su ejecución el algoritmo de entrenamiento, debe encontrar una hipótesis *h* que sea consistente. Es decir, que sólo cubra a los ejemplos positivos del conjunto de entrenamiento.

Por otro lado el conjunto de entrenamiento, de algunos problemas de aprendizaje, es tan grande que es imposible representarlo todo. En este caso sólo se toma un subconjunto para el entrenamiento, y entonces debemos suponer que la hipótesis obtenida será lo suficientemente buena, como para ser consistente con aquellos ejemplos que no fueron incluidos en el conjunto de entrenamiento. De esta manera tenemos la definición 2.7.

DEFINICIÓN 2.7 ***Hipótesis de aprendizaje inductivo:*** *"Cualquier hipótesis encontrada para aproximar la función objetivo sobre un sistema suficientemente grande de ejemplos de entrenamiento, también aproximará la función objetivo sobre los ejemplos no observados", [19].*

Como ejemplo de conjuntos de entrenamiento muy grandes, tenemos a aquellos problemas cuyos atributos pueden tomar valores en el conjunto de los números reales, por lo que el número total de posibles ejemplos es infinito. Evidentemente no se puede representar todo el conjunto de los reales como conjunto de entrenamiento, así que debemos:

- Tomar un conjunto de entrenamiento lo suficientemente grande para que el aprendizaje sea lo mejor posible; esto depende, entre otros elementos, del problema de aprendizaje que se va a resolver.
- Escoger el mejor conjunto de hipótesis iniciales posible para que la búsqueda sea más eficiente, y la hipótesis propuesta por el algoritmo de aprendizaje sea capaz de representar al conjunto de entrenamiento. Asumiendo además de que lo hará para todo el conjunto de entrenamiento (hipótesis de aprendizaje inductivo). Cuando suponemos esto último, esperamos además que la distribución del conjunto de ejemplos sea parecido a la distribución del conjunto de pruebas.

Ya que hemos definido una representación para las hipótesis, debemos ordenarlas de manera que el algoritmo de aprendizaje pueda realizar la búsqueda eficiente de aquella que es consistente con el conjunto de entrenamiento. El ordenamiento de las hipótesis tiene el mismo objetivo que el ordenamiento de un directorio telefónico: si no se ordenan los datos de las personas, la búsqueda de una persona es poco eficiente. En el caso de las hipótesis si no las ordenamos entonces no podremos encontrar la hipótesis consistente.

Para el ordenamiento de las hipótesis definimos los siguientes conceptos:

DEFINICIÓN 2.8 *Sea X el conjunto de ejemplos y H el conjunto de las hipótesis. Para cualquier instancia $x \in X$ y $h \in H$, decimos que x* ***satisface*** *a h si y sólo si $h(x) = 1$.*

La definición 2.8 nos indica que si una hipótesis puede representar a una instancia de *X*, sin importar si $c(x) = 1$ o $c(x) = 0$, entonces decimos que *x* satisface a la hipótesis. Por ejemplo, si $h = (?, ?, ?, ?, ?, ?)$, entonces cualquier $x \in X$ satisface a la hipótesis ya que no hay restricción en los valores que puede tomar x. Si $h = (\phi, \phi, \phi, \phi, \phi, \phi)$ entonces ninguna $x \in X$ va a satisfacer a *h*, ya que la hipótesis indica que ningún valor está permitido para ningún atributo.

Se dice que una hipótesis cubre a un ejemplo, si éste satisface dicha hipótesis.

Se dice que la hipótesis que tiene todas sus entradas con signo *?* es la más general.

Se dice que la hipótesis que tiene todas sus entradas con signo ϕ es la más específica.

DEFINICIÓN 2.9 ***Más general o igual que:*** *Sea h_j y h_k funciones booleanas definidas sobre X, entonces h_j es mas_general_o_igual_a h_k (denotado como $h_j \geq h_k$) si:* $(\forall x \in X)(h_k = 1 \rightarrow h_j = 1)$

En otras palabras si tenemos dos hipótesis h_j y h_k, decimos que la primera es más general si cubre más ejemplos que la segunda. Por ejemplo, si tenemos las hipótesis siguientes: $h_j = (?, ?, ?, ?, ?, ?)$, $h_k = (Soleado, Calor, Normal, Fuerte, Caliente, Estable)$, y el conjunto de entrenamiento de la tabla 2.3. Entonces h_j es más general que h_k, ya que cubre cuatro ejemplos del conjunto de entrenamiento, mientras que h_k sólo cubre el primer ejemplo.

DEFINICIÓN 2.10 ***Más general:*** *Diremos que h_j es (estrictamente) más_general_que h_k (denotado como $h_j > h_k$) si y sólo si ($h_j \geq h_k$) y ($h_k \not\geq h_j$)*

DEFINICIÓN 2.11 ***Más específica:*** *Si tenemos dos hipótesis: h_j y h_k, entonces diremos que h_j es mas_especifica_que h_k cuando $h_k > h_j$*

Las relaciones definidas en 2.9, 2.10 y 2.11 nos van a permitir realizar la búsqueda en todo el espacio de hipótesis, ya que podremos ir de la hipótesis más general a la más específica y viceversa, y explorar así este espacio. Al crear la relación de orden de general a específica se crea una retícula, como se muestra en la figura figura 2.5, en la cual, podemos apreciar cómo es que una hipótesis entre más general más ejemplos cubre del conjunto de entrenamiento, y entre más específica menos ejemplos cubre.

Por ejemplo en la figura (2.5) podemos ver que h_0 es la hipótesis más específica por lo que no cubre ningún elemento del espacio de ejemplos. En contraste la hipótesis h_4 es la más general pues cubre cada ejemplo cubierto por las demás hipótesis: h_0, h_1, h_2, y h_3. Por lo tanto el orden de estas hipótesis es el siguiente: $h_4 \geq h_3 \geq h_2 \geq h_1 \geq h_0$.

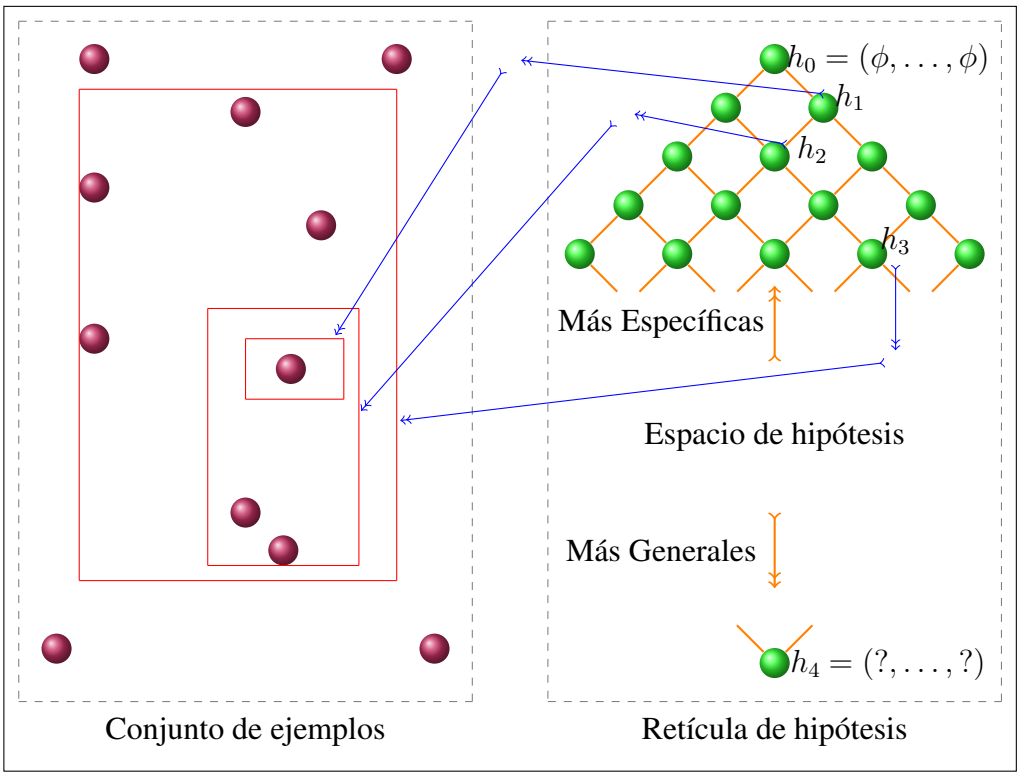


Figura 2.5: Representación del espacio de hipótesis: Retícula

Como podemos ver en la figura 2.5 (diagrama de Hasse)[1], h_4 que es la hipótesis más general cubre más ejemplos, y h_1 que está muy cerca de la hipótesis más específica apenas cubre un

[1]Los diagramas de Hasse permiten representar un conjunto ordenado que está determinado por una relación de orden. En estos diagramas se ponen los elementos superiores en escalones también superiores, los cuales, están unidos por una sucesión ascendente de arcos [28].

elemento del conjunto de ejemplos. Este diagrama nos permite tener una idea más clara de cómo debe ser la búsqueda de la mejor hipótesis, y podemos observar que:

- Si tenemos la hipótesis más general ($h = (?, ?, ?, ?, ?, ?)$) seguramente va a cubrir todos los ejemplos positivos, y aunque es lo que queremos esta hipótesis también va a cubrir todos los ejemplos negativos, por lo que la hipótesis será inconsistente.

- Si tenemos la hipótesis más específica ($h = (\phi, \phi, \phi, \phi, \phi, \phi)$) seguramente no va a cubrir ningún ejemplo negativo, uno de nuestros objetivos, sin embargo esta hipótesis no cubrirá ningún ejemplo positivo, por lo que esta hipótesis es inconsistente.

- En algún punto intermedio entre la hipótesis más general y la más específica se encuentra una hipótesis que es consistente con el conjunto de entrenamiento.

- Como hipótesis inicial para el algoritmo de entrenamiento podemos proponer la más específica (técnicas *bottom-up*), la más general (técnicas top-down) o ambas (técnicas híbridas).

- Si nuestra hipótesis es muy específica es claro que debemos hacerla más general.

- Por otro lado si nuestra hipótesis es muy general, debemos hacerla más específica.

- De lo anterior, concluimos que para recorrer el espacio de búsqueda (esquematizado en la retícula de la figura 2.5) debemos generalizar o especificar. Si la hipótesis cubre ejemplos negativos entonces tendremos que especificar la hipótesis. Por otro lado si la hipótesis no cubre ejemplos positivos, debemos generalizarla.

De esta manera ya tenemos el último punto cubierto de nuestro diseño del sistema de aprendizaje de conceptos, el cuál, debe aprender a inferir cuáles son las condiciones propicias para que la persona *Q* disfrute de su deporte acuático favorito. Presentamos enseguida el diseño conforme a los elementos mostrados en [19].

1. **Tarea *T*:** Deducir las condiciones que debe tener un día para que la persona *Q* pueda disfrutar de su deporte acuático favorito.

2. **Medida de rendimiento *P*:** Porcentaje de ejemplos positivos y negativos cubiertos.

3. **Experiencia de entrenamiento *E*:** Conjunto de entrenamiento.

4. **Función Objetivo y su representación:** Función booleana del concepto DisfrutaDia.

5. **Elección del algoritmo de entrenamiento:** Algoritmo que generalice o especifique las hipótesis iniciales dependiendo de las características de estas.

En las secciones 2.4.3 y 2.4.4 mostramos dos algoritmos que utilizan la notación para las hipótesis mostradas en esta sección.

2.4.3 Algoritmo FIND-S

El algoritmo, diseñado por Mitchell [19], FIND-S inicializa o propone la hipótesis más específica posible, y a partir de esta va comparando cada elemento del conjunto de entrenamiento que tiene valor *1*, de manera que en cada una de estas comparaciones empareja la hipótesis con el elemento de entrenamiento actual, para ello generaliza en cada iteración la hipótesis. El algoritmo 2.2 presenta los pasos a seguir para encontrar la hipótesis óptima.

Algoritmo 2.2 *Algoritmo FIND-S*

```
begin
  Inicializar con la hipótesis más específica h = (φ, φ, φ, ..., φ)
  for Cada restricción del atributo a_i ∈ h
      if a_i es satisfecha por x, donde x es el elemento de entrenamiento
        then No hacer nada
        else Reemplazar a_i en h por la siguiente restricción más general que es satisfecha por x
      fi
  od
end
```

En la figura 2.6 esquematizamos este proceso. Comenzamos con la hipótesis más espécifica que es $h_1 = (\phi, \phi, \phi, ..., \phi)$ y después la generalizamos para cubrir cada elemento del conjunto de entrenamiento, en este ejemplo se hicieron cinco generalizaciones, por lo que el conjunto de entrenamiento tiene sólo cinco elementos positivos.

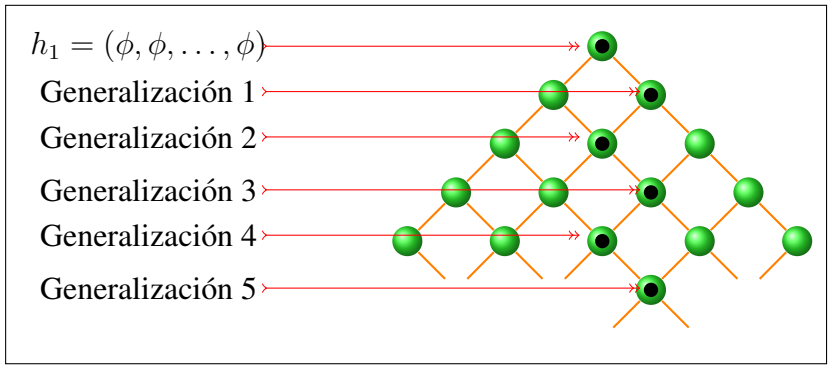


Figura 2.6: Búsqueda de hipótesis: algoritmo FIND-S

2.4.4 Algoritmo Candidato-Eliminación

Este algoritmo también propuesto por Tom Mitchell [19], inicializa dos conjuntos de hipótesis: el de las más generales H_g y el de las más específicas H_e. Después recorre la retícula, formada por el orden de las hipótesis, a partir de la más general y de la más específica, como se muestra en la figura 2.7. Al final devuelve un conjunto de hipótesis consistentes con el conjunto de entrenamiento llamado *espacio de versiones*. El algoritmo 2.3 describe el proceso seguido para encontrar el espacio de versiones.

Algoritmo 2.3 *Algoritmo candidato-eliminación*

Inicializar el conjunto H_g de las hipótesis más generales del espacio de hipótesis
Inicializar el conjunto H_e de las hipótesis más específicas del espacio de hipótesis
Para cada elemento del conjunto de entrenamiento d, hacer:
if d es un ejemplo positivo
  then Eliminar de H_g las hipótesis inconsistentes con d
       Generalizar las hipótesis en H_e inconsistentes con d, asegurándose
       que sean más específicas que algún elemento de H_g, y que no sean más
       generales que algún elemento de H_e.
  else
       Eliminar de H_e las hipótesis consistentes con d
       Especificar las hipótesis en H_g consistentes con d, hasta que sean
       inconsistes, manteniendo que sean más generales que algún elemento en
       H_e, y que no sean más específicas que algún elemento de H_g
fi

Debemos recalcar que este algoritmo devuelve un conjunto de hipótesis donde *todos* sus elementos con consistentes con el conjunto de entrenamiento. En la definición 2.12 describimos de manera formal el espacio de versiones.

DEFINICIÓN 2.12 *El **Espacio de versiones** denotado como $VS_{H,D}$, con respecto al espacio de hipótesis H y al conjunto de entrenamiento D, es el subconjunto de hipótesis de H consistentes con los ejemplos de entrenamiento D, ver ec 2.11.*

$$VS_{H,D} \equiv \{h \in H : Consistente\,(h, D)\} \tag{2.11}$$

Para un propósito pedagógico los algoritmos *FIND-S* y *Candidato-Eliminacion* son buenos, sin embargo, cuando se quieren resolver problemas más complejos, están limitados debido al lenguaje poco expresivo con que se representan los problemas y las hipótesis. Por otro lado existen técnicas de aprendizaje maquinal que representan las hipótesis y los problemas con un lenguaje más completo, dos de estas técnicas son: inducción de árboles de decisión y programación lógica inductiva, que se verán en los capítulos 3 y 4 respectivamente.

2.4.5 Prejuicio

La creación de métodos de AM, implica tomar decisiones acerca de:

- El lenguaje apropiado para representar las hipótesis.
- Cómo recorrer el espacio de búsqueda.
- Cómo evitar el sobreentrenamiento.

Estas decisiones son conocidas como prejuicio o *bias* [36], y tomando en cuenta la lista anterior tenemos los siguientes tipos.

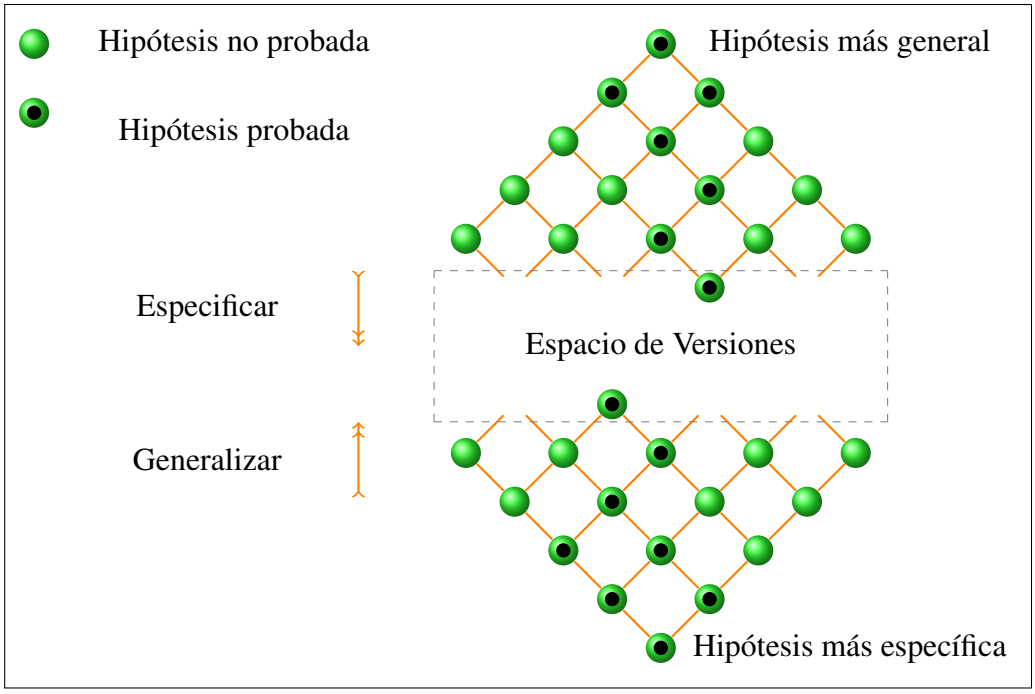


Figura 2.7: Representación del espacio de versiones en una retícula

Prejuicio de lenguaje

Se refiere a la especificación del lenguaje con el que se representan las hipótesis. Dicho lenguaje determina si existen restricciones. Por ejemplo, en los conceptos vistos hasta ahora, utilizamos una conjunción de valores para representar las hipótesis. Sin embargo, de manera implícita impusimos una restricción: no permitir disyunciones y/o negaciones entre los valores de los atributos. Por lo tanto si el problema exige una hipótesis con disyunciones o negaciones, entonces sería imposible crear un espacio de búsqueda que contenga la hipótesis buscada. Cada problema y método de aprendizaje implica tomar decisiones respecto al lenguaje que se utilizará.

Prejuicio de búsqueda

Cuando se determina el lenguaje con el que se representarán las hipótesis, de manera inmediata se define el espacio de búsqueda. Por ejemplo si hemos decidido que cada hipótesis será la conjunción de valores de cada atributo (uno por cada atributo), entonces conceptualmente tendremos una noción del espacio de búsqueda. Si hay 4 atributos posibles y por cada uno de ellos hay 3 posibles valores, sabremos de antemano que hay 2^{3^4} posibles hipótesis. Por lo tanto la siguiente decisión a tomar es cómo recorrer ese espacio de búsqueda. Esta decisión es el prejuicio de búsqueda (o *search bias*). En el caso de las hipótesis expuestas hasta ahora, este prejuicio se refiere a la generalización y especificación de las hipótesis, las cuales nos permiten recorrer el espacio de búsqueda. También esta decisión se toma de acuerdo al método y problema de aprendizaje.

Prejuicio de prevención de sobreentrenamiento

Este tipo de prejuicio, se refiere a la elección del método que se utilizará para evitar el sobreentrenamiento. Un ejemplo de este prejuicio, que veremos más adelante, se define cuando se inducen árboles de decisión. Para evitar el sobreentrenamiento, se determinan criterios de paro para detener el crecimiento del árbol antes que surja el sobreentrenamiento.

El término *prevención* nos remite a evitar algo antes que surja. Sin embargo, hay ocasiones en que el sobreentrenamiento no se evita, sino que se elimina después de que sucede. En la inducción de los árboles de decisión, el sobreentrenamiento se elimina utilizando la poda.

2.5 Resumen

En este capítulo hemos partido de la definición de *aprendizaje*, para definir formalmente qué es el *aprendizaje maquinal*. También hemos presentado los elementos básicos que deben ser definidos para diseñar e implementar un programa que aprende. Estos elementos son:

- Tarea T.
- Medida de rendimiento P.
- Experiencia de entrenamiento E.
- Función objetivo.
- Representación de la función objetivo.
- Elección del algoritmo de entrenamiento.

Asimismo indicamos que existen distintos tipos de aprendizaje, pero en particular el aprendizaje que estudiamos en esta tesis es el inductivo: aprendizaje que induce una hipótesis a partir de un conjunto de ejemplos.

Es importante recalcar que muchos problemas de aprendizaje se pueden modelar como problemas de optimización, como en nuestro ejemplo del ajedrez, donde se tiene que optimizar la función objetivo.

Por otro lado, mencionamos que cuando se requiere clasificar un conjunto de datos, o tomar decisiones basadas en la experiencia, el aprendizaje maquinal se puede tomar como herramienta para llevar a cabo tales tareas.

El aprendizaje de conceptos, nos ha permitido ejemplificar y ampliar la idea de AM al definir:

- Un lenguaje para representar las hipótesis, el espacio de búsqueda y el conjunto de entrenamiento.
- Una relación de orden entre las hipótesis de manera que se pueda recorrer el espacio de búsqueda.

Los algoritmos *FIND-S* y *Candidato-Eliminación*, nos permitieron ejemplificar y resumir el proceso de aprendizaje presentado en párrafos anteriores. Sin embargo, existen técnicas con las que podemos representar las hipótesis y el planteamiento de los problemas con más expresividad. En los siguientes dos capítulos presentaremos dos de estos métodos (*inducción de árboles de decisión* y *programación lógica inductiva*), que serán el punto de partida para la propuesta principal de esta tesis.

Capítulo 3

Árboles de Decisión

Una de las técnicas más utilizadas para encontrar hipótesis inducidas a partir de un conjunto de ejemplos son los árboles de decisión, este tipo de estructuras, también conocidas como TDIDT (*Top Down Induction of Decision Trees*) [24], permiten superar las limitaciones impuestas por el uso de tuplas para realizar el proceso de aprendizaje. Pero quizá el inconveniente más evidente que se mejora, se da en relación a la representación de las hipótesis. Un árbol de decisión aporta más información que una tupla.

En este capítulo definimos los árboles de decisión y presentamos el algoritmo general que los induce. También describimos cada elemento involucrado en el diseño de un algoritmo que crea árboles de decisión:

- Elección de atributos para asignarlos a cada nodo.
- División de los nodos en subárboles u hojas.
- Criterios para detener el crecimiento del árbol.
- Poda.

Asimismo listamos las ventajas y desventajas de utilizar los árboles de decisión para aprendizaje y clasificación. Como algoritmo para ejemplificar esta técnica, describimos *ID3* [24].

3.1 Definición de árbol de decisión

La inducción de árboles de decisión, es una de las técnicas de AM más utilizadas. Pero antes de describir el proceso seguido por los algoritmos utilizados para crear árboles, definimos qué son.

DEFINICIÓN 3.1 *Un árbol de decisión, en AM, representa la hipótesis que el algoritmo de aprendizaje encuentra al final de su ejecución. Como en cualquier estructura de datos que representa un árbol, el de decisiones tiene un nodo raíz, nodos internos que se dividen en dos o más hojas y/o subárboles. Pero en éstos, cada nodo interno así como la raíz representa uno o más atributos del concepto de aprendizaje. Cada división (rama) representa un subconjunto de valores que pueden asignarse al atributo en cuestión. Y cada hoja es uno de los valores que pueden ser asignados a la función objetivo.*

En la figura 3.1, mostramos un árbol de decisión que representa la regla que generaliza el clima propicio, que debe tener un día, para que se pueda jugar al tenis. Los atributos que se consideran son: pronóstico, humedad y viento.

Los nodos internos, incluyendo la raíz, representan sólo uno de los atributos mencionados en el párrafo anterior. Aunque, dependiendo del algoritmo de aprendizaje, cada nodo podría tener más de un atributo.

Cada nodo se divide en dos o más subárboles u hojas. Para este ejemplo cada división o rama, representa sólo un valor del atributo del nodo padre. Sin embargo, debemos aclarar que algunos algoritmos, permiten que cada rama sea representada por más de un valor.

En el caso de los atributos con valores continuos, el nodo se puede dividir con una relación de desigualdad ($\leq$ ó $\geq$). Donde los valores que cumplan con esta relación crean un subárbol y los que no, crean otro subárbol.

Por último, los valores que puede tomar la función objetivo para este problema, son: $\{si,no\}$, por lo que las hojas del árbol representan uno de los dos posibles valores.

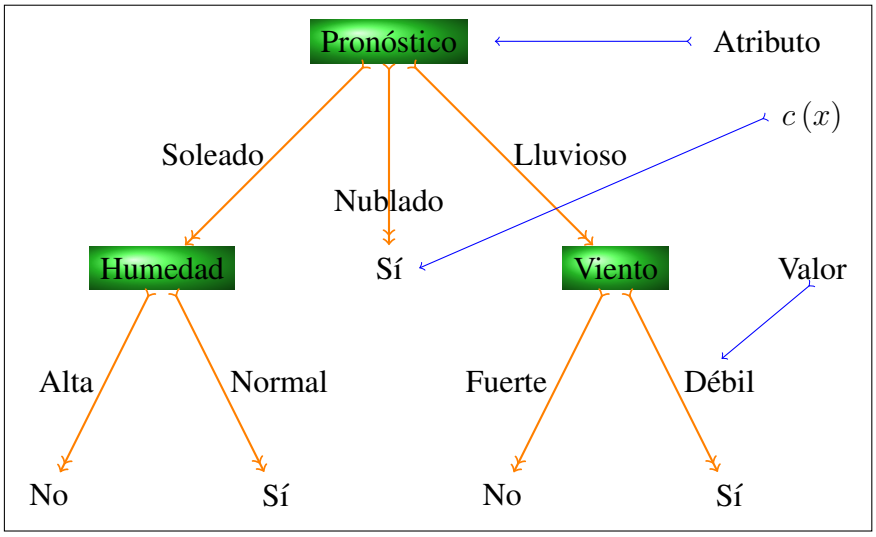


Figura 3.1: Árbol de decisión: días propicios para jugar tenis

Para interpretar la hipótesis obtenida, podemos utilizar una regla del tipo *si-entonces*. En nuestro ejemplo de la figura 3.1 la regla de decisión es:

Hipótesis: "**Si** (Pronóstico = Soleado y Humedad = Normal) ó (Pronóstico = Nublado) ó (Pronóstico = Lluvioso y Viento = Débil) **Entonces** (JugarTenis = Si)".

Estas reglas de decisión obtenidas son en realidad una disyunción de conjunciones, donde cada conjunción es una ruta de la raíz a una de las hojas.

Los árboles de decisión, son de hecho clasificadores [27], desde esta perspectiva, nuestro ejemplo tiene sólo dos clases: *si,no*. Cuando utilizamos un árbol como clasificador, entonces obtenemos un número de reglas equivalente al número de clases definidas (dos o más).

3.2 Algoritmo básico de inducción de árboles de decisión

Todos los algoritmos existentes que inducen árboles de decisión tienen una característica en común: comienzan con un nodo raíz, y a partir de éste el algoritmo crece o poda el árbol hasta que se cumple algún criterio de paro. En otras palabras el algoritmo comienza con la hipótesis más general posible y la especifica hasta obtener la hipótesis consistente.

El algoritmo 3.1, presentado en [15], muestra de manera general la inducción de los árboles de decisión.

Algoritmo 3.1 *Algoritmo general de árboles de decisión*

begin
arbolDecision $(S, A, y, CriterioDivision, CriterioParo)$
Donde:
S: Conjunto de entrenamiento
A: Conjunto de elementos de entrada
y: Función objetivo
$CriterioDivision$: Método para evaluar cada división
$CriterioParo$: Criterio para detener el crecimiento del árbol
Crear un nuevo árbol T con un nodo raíz
if $CriterioParo\,(S)$
then
Marcar a T como una hoja, con el valor más común de y con la etiqueta S
Regresar
else
$\forall a_i \in A$ encontrar a tal que se obtenga el mejor $CriterioDivision\,(a_i, S)$
Etiquetar t con a
for cada resultado v_i de a do
Asignar $SubArbol_i = arbolDecision\,(\sigma_{a=v_i} S, A, y)$
Conectar el nodo raíz de t_T al $SubArbol_i$ con una arista etiquetada como v_i
od
fi
end
Regresa $PodaArbol\,(S, T, y)$

. . . . . .

$PodaArbol\,(S, T, y)$
Donde:
S: Conjunto de entrenamiento
y: Función objetivo
T: Árbol que será podado
do
Selecciona un nodo t en T tal que podarlo mejore algún criterio de evaluación
if $t \neq 0$
then

$T = poda\,(T, t)$
fi
until $(t = 0)$
Regresa T

El algoritmo 3.1 es aparentemente simple, sin embargo requiere la implementación de los siguientes componentes:

- Elección de los mejores atributos para cada nodo interno.
- Criterios de paro para detener el crecimiento del árbol.
- División de cada nodo.
- Técnicas de poda.

En el resto de esta sección presentamos estos elementos.

3.2.1 Elección de atributos

Hemos visto que en un árbol de decisión cada nodo interno representa un atributo del concepto de aprendizaje (aunque podría representar más de uno), y el algoritmo general para inducir árboles de decisión nos indica que debemos crear un nodo raíz, a partir del cual, se comienza el crecimiento del árbol.

Es evidente que en cada nodo interno del árbol tendremos la necesidad de elegir los atributos que lo representarán. La elección del atributo o atributos para cada nodo tiene el objetivo de seleccionar, para cada nodo, aquellos atributos que mejor clasifiquen el conjunto de entrenamiento.

Existen muchas funciones heurísticas para elegir los mejores atributos para cada nodo. Estas funciones se clasifican en dos categorías bajo el criterio de división del nodo [27].

- **Criterio de división univariable.** Se elige solamente un atributo por cada uno de los nodos. Este criterio puede ser caracterizado de varias maneras, de acuerdo al origen de la medición (teoría de la información, dependencia y distancia); y de acuerdo a la estructura de la medición, como el criterio basado en impuridad o criterio binario. La ganancia de la información, *Gini Index* y *Gain Ratio* son ejemplos de este tipo de funciones heurísticas.
- **Criterio de división multivariable.** Se eligen dos o más atributos por cada uno de los nodos. Para este tipo de criterio es necesario encontrar el mejor subconjunto de atributos que serán representados en cada uno de los nodos. Este criterio permite obtener árboles más compactos y muchas veces más exactos que los obtenidos con criterio univariable. Las técnicas más utilizadas encuentran la mejor combinación lineal de un subconjunto de atributos. Ejemplos de estas técnicas son: búsqueda ávida, programación lineal y análisis discriminante lineal.

3.2.2 División de nodo

Una vez que se han elegido los atributos para un determinado nodo, es necesario definir cómo se dividirá. Para los objetivos de esta tesis, sólo tomamos en cuenta cuando un nodo es representado por un sólo atributo. En este caso contamos con los siguientes criterios para dividir el nodo:

1. **Univalor.** Los algoritmos con este criterio, solamente toman un valor del atributo por cada rama del árbol. Un ejemplo lo mostramos en la figura 3.2, donde podemos ver que el nodo raíz representa al atributo X con valores en $\{V_1, V_2, \ldots, V_9\}$, y cada rama representa un sólo valor de la variable X.

2. **Multivalor.** Como ejemplo de este criterio presentamos el árbol de la figura 3.3, en el cual apreciamos que la raíz representa al mismo atributo que en el punto anterior, sin embargo sólo tiene dos subárboles, esto es debido a que cada rama representa más de un valor del atributo X. Este criterio es llamado también *aprendizaje multivalores*, y tiene como ventaja la inducción de árboles más compactos y más precisos que los generados con el aprendizaje univalor.

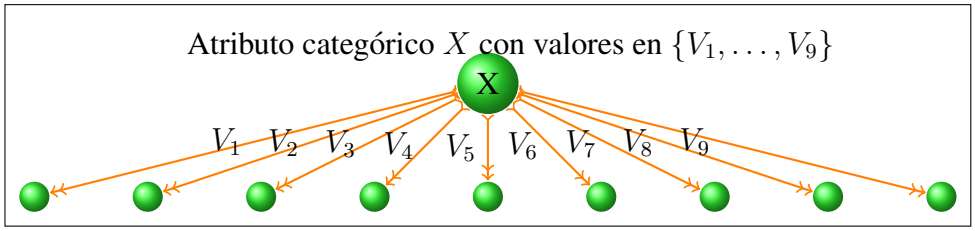


Figura 3.2: División del nodo con aprendizaje univalor

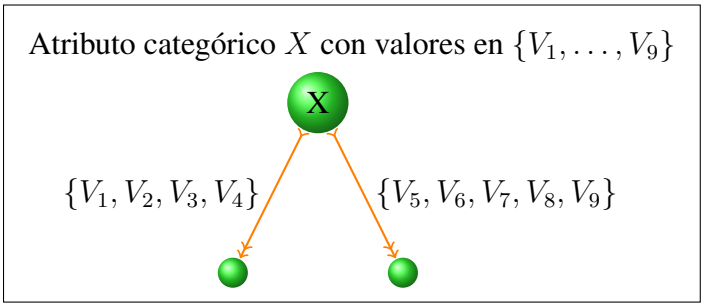


Figura 3.3: División del nodo con aprendizaje multivalores

3.2.3 Criterio de paro

Conforme crece el árbol la hipótesis obtenida será más específica, sin embargo debe haber un criterio que nos indique en donde detener el crecimiento del árbol, de manera que se obtenga la mejor hipótesis posible. Por lo general el crecimiento del árbol se detiene cuando:

- **Se ha alcanzado el máximo nivel de profundidad.** Muchas veces se puede especificar el nivel máximo que puede alcanzar el árbol de decisión, y cuando el crecimiento del árbol ha alcanzado dicho nivel el crecimiento se detiene.

- **El conjunto de entrenamiento restante pertenece a la misma clase**. Cada nodo hijo representa un subconjunto de entrenamiento del nodo padre, cuando todos los elementos o un porcentaje (preespecificado) de estos pertenecen a la misma clase, entonces ese nodo se convierte en hoja.

- **Si el número de elementos del nodo hijo tiene menos elementos que el especificado**, entonces ese nodo se convierte en hoja y se detiene el crecimiento del árbol.

- **Cuando todos los elementos restantes tienen el mismo valor**, el nodo se convierte en hoja y se detiene el crecimiento del árbol.

3.2.4 Poda

En la sección 2.2 vimos que durante la ejecución de los algoritmos de aprendizaje maquinal (entrenamiento), hay un momento donde el aprendizaje es máximo, después de este punto el error crece y se da un fenómeno llamado sobreentrenamiento (overfitting). Los algoritmos de inducción de árboles de decisión no son la excepción, y el sobreentrenamiento se presenta en un punto del crecimiento del árbol, por lo que tenemos dos opciones: evitarlo o eliminarlo.

1. **Pre-poda.** El objetivo es detener el crecimiento del árbol antes de que se presente el sobreentrenamiento. Los criterios los presentamos en la sección 3.2.3.

2. **Post-poda.** Después del entrenamiento el árbol se poda (se generaliza la hipótesis) hasta eliminar el sobreentrenamiento.

Algunas técnicas para llevar a cabo la poda son:

- **Cost-complexity pruning.** Se lleva a cabo en dos etapas. En la primera etapa se realiza el crecimiento del árbol de decisión y en la segunda uno de los subárboles del nodo raíz es eliminado. La elección del nodo eliminado se basa en la estimación de generalización del error.

- **Reduced error pruning.** Realiza un barrido de los nodos, desde las hojas hasta la raíz, en forma transversal. Por cada nodo analizado se verifica si no hay reducción en la precisión de la hipótesis obtenida al reemplazar el nodo, por la clase más frecuente, de ser así el nodo se poda.

- **Minimum Error Pruning (MEP).** Esta técnica también realiza un barrido de los nodos de forma transversal, desde las hojas hasta la raíz, pero a diferencia de la anterior por cada nodo analizado se compara el error de probabilidad de cada nodo antes y después de podarlo.

3.3 Algoritmo ID3

Para ejemplificar cómo se induce un árbol de decisión, describimos a continuación el primer algoritmo desarrollado: *ID3* (Iterative Dichotomiser 3) [24].

Este algoritmo fue creado por Ross Quinlan y es de tipo *Top-Down*. Aunque más específicamente pertenece a la clase *TDIDT (Top Down Induction of Decision Trees)*. El algoritmo 3.2 muestra su proceso de inducción.

Algoritmo 3.2 *Algoritmo ID3*

```
begin
  ID3(Ejemplos, atributo_objetivo, atributos)
  Crear un nodo raíz para el árbol
  if Todos los ejemplos son positivos then Regresar el nodo raíz con la etiqueta de + fi
  if Todos los ejemplos son negativos then Regresar el nodo raíz con la etiqueta de - fi
  if Ejemplos está vacío
    then Regresar el nodo raíz con la etiqueta del valor más común de atributo_objetivo
    else begin
           A = Atributo con mayor Ganancia de Información
           El atributo de decisión para la raíz es A
           for v_i ∈ A
             Añadir una rama a la raíz con la prueba A = v_i
             Ejemplos_vi es el subconjunto de Ejemplos con valor v_i para A
             if Ejemplos_vi esta vacío
               then Bajo nueva rama añadir una hoja con etiqueta del valor más común del
                    atributo_objetivo en Ejemplos
               else
                    Añadir otro sub-árbol en la rama nueva
                    ID3(Ejemplos, atributo_objetivo, atributos − A)
             fi
           od
         end
  fi
  Regresar raíz
end
```

Para elegir el atributo para cada nodo, ID3 utiliza una heurística basada en la medición llamada *ganancia de información* [31]. Ésta se obtiene a partir de otra medida llamada *entropía*; por lo que primero definiremos ésta. Dado un conjunto S de ejemplos, su entropía está definida por la ecuación 3.1.

$$Entropia(S) \equiv -p_{\oplus} log_2 p_{\oplus} - p_{\ominus} log_2 p_{\ominus} \tag{3.1}$$

Donde: $p_{\oplus}$ es la proporción de ejemplos positivos en el conjunto S. $p_{\ominus}$ es la proporción de ejemplos negativos en S. Si sucede que: $0\ log\ 0$, entonces la entropía es 0.

La entropía se puede ver como el grado de incertidumbre. Si la entropía es cero (su valor mínimo) no existe incertidumbre, por lo que sabremos de antemano la clase a la que pertenece un ejemplo. Esto sucede cuando todos los ejemplos corresponden a la misma clase.

Cuando la entropía es uno (su valor máximo), entonces hay incertidumbre. No sabemos a qué clase pertenecerá un ejemplo si lo tomamos al azar. En la figura 3.4 se muestra la gráfica de la entropía.

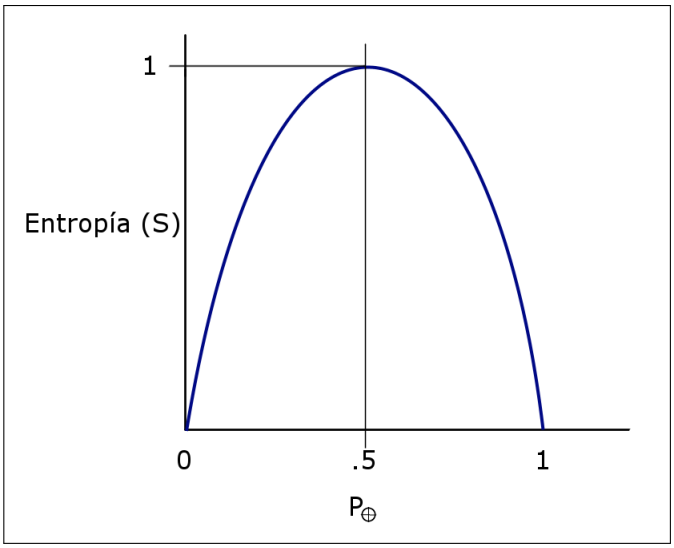


Figura 3.4: Gráfica de entropía

Ya que medimos la entropía del conjunto de ejemplos S, el siguiente paso es calcular la ganancia de información para cada atributo A. El atributo que posea mayor ganancia de información, será el elegido. Este atributo es el que reduce la entropia (la incertidumbre), y por lo tanto el que mejor clasifica el conjunto de entrenamiento. La ecuación 3.2 define la ganancia de información para un atributo A relativo al conjunto de entrenamiento S [19].

$$Ganancia(G, A) \equiv Entropia(S) - \sum_{Val(A)} \frac{S_v}{S} Entropia(S_v) \tag{3.2}$$

Donde: $Val\,(A)$ es el conjunto de todos los posibles valores del atributo A. S_v es el subconjunto de S, para el cual, el atributo A tiene el valor v.

El primer término de la ecuación 3.2 es la entropía del conjunto de entrenamiento S, y el segundo término es la entropía esperada después de dividir el conjunto S utilizando el atributo A.

3.3.1 Ejemplo: *días propicios para jugar tenis*

Para ejemplificar el funcionamiento de este algoritmo, veamos cómo se genera el árbol para el problema planteado en la sección 3.1. El objetivo es obtener una regla que nos indique los valores que debe tener cada característica de un día en particular, de manera que sea propicio para jugar tenis. Como conjunto de entrenamiento tomamos el pronóstico del tiempo que predominó

en días anteriores, tomando en cuenta también si se consideró jugar al tenis. Los datos que se recopilaron se muestran en la tabla 3.1.

Día	Pronóstico	Temperatura	Humedad	Viento	Jugar Tenis
1	Soleado	Caliente	Alta	Débil	No
2	Soleado	Caliente	Alta	Fuerte	No
3	Nublado	Caliente	Alta	Débil	Sí
4	Lluvioso	Media	Alta	Débil	Sí
5	Lluvioso	Fresco	Normal	Débil	Sí
6	Lluvioso	Fresco	Normal	Fuerte	No
7	Nublado	Fresco	Normal	Fuerte	Sí
8	Soleado	Medio	Alta	Débil	No
9	Soleado	Fresco	Normal	Débil	Sí
10	Lluvioso	Medio	Normal	Débil	Sí
11	Soleado	Medio	Normal	Fuerte	Sí
12	Nublado	Medio	Alta	Fuerte	Sí
13	Nublado	Caliente	Normal	Débil	Sí
14	Lluvioso	Medio	Alta	Fuerte	No

Tabla 3.1: Conjunto de entrenamiento para ID3

Como primer paso debemos elegir el primer atributo para el nodo raíz. Los atributos correspondientes al clima son: pronóstico, temperatura, humedad y viento. Si calculamos la ganancia de información utilizando las ecuaciones 3.1 y 3.2, obtendremos los siguientes resultados:

- Ganancia(S, pronóstico) = 0.246
- Ganancia(S, humedad) = 0.151
- Ganancia(S, viento) = 0.048
- Ganancia(S, temperatura) = 0.029

El atributo elegido, con mayor ganancia de información, es *pronóstico*. Ya que se definió el atributo para el primer nodo (la raíz del árbol), éste se divide en los tres posibles valores que tiene ese atributo: soleado, nublado y lluvioso.

Por cada subconjunto obtenido al dividir el nodo raíz, se realiza el mismo proceso (llamada recursiva a ID3). El proceso termina cuando cada elemento del subconjunto pertenece a la misma clase.

En la figura 3.5 y en la regla 3.3, se muestran el árbol inducido y la hipótesis final respectivamente. La interpretación del árbol de decisión y de la hipótesis que éste representa fueron aclarados en la sección 3.1.

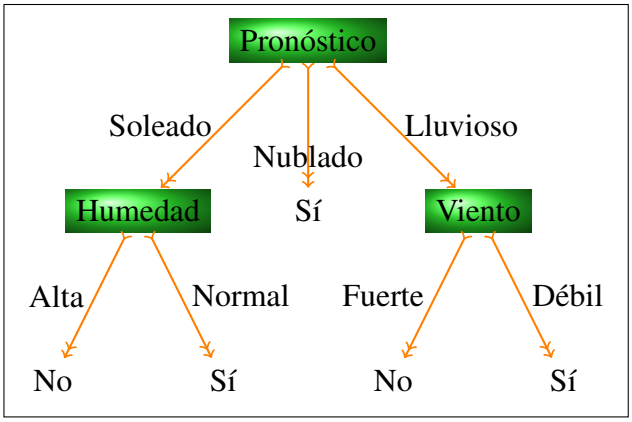


Figura 3.5: Árbol inducido con algoritmo ID3

$$\begin{array}{ll} Si & (Pronostico = Nublado) \vee \\ & ((Pronostico = Soleado) \wedge (Humedad = Normal)) \vee \\ & ((Pronostico = Lluvioso) \wedge (Viento = Debil)) \\ Entonces & (Si\ Jugar\ Tenis) \end{array} \tag{3.3}$$

No es difícil traducir este árbol a una hipótesis, sin embargo cuando se tienen más atributos y cada uno de ellos tiene a su vez muchos valores, el árbol obtenido seguramente crecerá demasiado y la regla será difícil de interpretar.

3.4 Resumen

Como pudimos apreciar los árboles de decisión son una buena opción para poder encontrar reglas que clasifiquen datos. También permiten identificar ciertos patrones, sobre todo cuando hay una gran cantidad de datos, que permiten tomar decisiones dependiendo del contexto en el que se utilicen. El diagnóstico de enfermedades, tomando en cuenta el historial médico de muchos pacientes, es un buen ejemplo donde se puede utilizar este tipo de algoritmos.

Frecuentemente los árboles obtenidos son demasiado grandes, por lo que las técnicas de poda, los criterios de paro, la elección de atributos para cada nodo y el aprendizaje multivalores, son factores importantes para obtener el árbol más pequeño posible, y que éste represente la hipótesis consistente con el conjunto de entrenamiento.

En AM, los árboles de decisión no son la única opción, en el capítulo 4 describimos otra técnica de aprendizaje maquinal cuyo lenguaje para representar los problemas y las hipótesis es más expresivo.

Capítulo 4

Programación Lógica Inductiva

La *Programación Lógica Inductiva (ILP)* es otra técnica de aprendizaje maquinal, cuyas características principales son el uso de la lógica de primer orden para representar e inducir las hipótesis, y el uso de información adicional, además de los ejemplos, para obtener hipótesis más precisas [22].

Los algoritmos de ILP al igual que los árboles de decisión utilizan un conjunto de ejemplos para inducir la hipótesis que explicará el concepto que se quiere aprender.

En este capítulo presentamos los principales elementos de la ILP, los cuales listamos a continuación:

- Definición de ILP.
- Orden general a específico en ILP.
- Clasificación de algoritmos de ILP.
- Ejemplo de algoritmo en ILP: *FOIL*.

4.1 Definición de Programación Lógica Inductiva

Como hemos visto, para inducir una regla o un conjunto de reglas (hipótesis), que generalicen algún concepto, los sistemas basados en AM utilizan un conjunto de ejemplos. Esta manera de aprender, la conocemos desde que somos muy pequeños. Tomemos como ejemplo el caso de un niño de cinco años al que queremos enseñarle a sumar. Lo más normal en este caso, es que le proporcionemos un conjunto de ejemplos, además de ciertas explicaciones. Podemos indicarle que:

Es correcto:

- $5 + 5 = 10$
- $1 + 1 = 2$
- $\vdots$

Es incorrecto:

- $5 + 5 = 15$
- $1 + 1 = 20$
- $\vdots$

Por otro lado, podemos notar que para que el niño aprenda, a partir de estos ejemplos, es necesario que sepa *a priori*:

- La definición, al menos intuitiva, de qué es un número natural.
- Que hay números *más grandes* que otros (por lo general se utiliza didácticamente una recta numérica).
- El simbolismo utilizado para cada número (un niño que lo tenga presente notará que $@+@$ no tiene ningún sentido).
- etc.

Es evidente que sin los conocimientos necesarios, un niño no podría aprender a sumar, restar, etc. Y más adelante si la persona no tiene tales conocimientos, no podrá aprender matemáticas más avanzadas. En general, siempre debemos tener un conocimiento básico que nos ayude a aprender nuevas cosas.

La *Programación Lógica Inductiva (ILP)*, es una disciplina de aprendizaje maquinal que, además de ejemplos, utiliza este conocimiento base para poder generar hipótesis que generalicen con mayor precisión. Este conocimiento es llamado *conocimiento previo*.

Otra característica de la ILP es que utiliza el lenguaje de la lógica de primer orden para representar las hipótesis, los ejemplos y el conocimiento previo.

Stephen Muggleton [22] define formalmente la ILP como la intersección entre el aprendizaje maquinal y la programación lógica. En esta metodología se realiza un aprendizaje inductivo como en AM, pero con las ventajas del lenguaje de la programación lógica, que permite plantear los problemas y las soluciones con un lenguaje más natural, ver ec. 4.1.

$$ILP = \text{Aprendizaje Maquinal} \cap \textit{Programación Lógica} \tag{4.1}$$

En ILP la hipótesis *H*, el conjunto de ejemplos *E* y el conocimiento previo *B* son programas lógicos (ver apéndice B para una breve introducción a Prolog). Sin embargo la esencia de cualquier sistema de AM se mantiene, es decir, se busca una hipótesis *H* que junto con el conocimiento previo *B*, describa al conjunto de ejemplos *E*. En otras palabras lo que se busca en ILP es encontrar un programa lógico (hipótesis) que junto con el conocimiento previo, también programa lógico, sirvan como base para poder deducir sólo los ejemplos positivos, ver ec. 4.2.

$$B \wedge H \models E \tag{4.2}$$

4.2 Orden general a específico en ILP

Recordemos que el prejuicio de búsqueda (sección 2.4.5), implica la toma de decisiones respecto a cómo se va a realizar la búsqueda de las hipótesis. En el caso de las tuplas utilizamos los símbolos *?* y ϕ para generalizar y especificar las hipótesis respectivamente. En el caso de los árboles de decisión, la especificación y la generalización se realizan con el crecimiento y la poda del árbol.

En ILP también se generalizan y se específican las hipótesis. Para ello, se define una relación de orden entre las hipótesis utilizando una propiedad llamada *subsunción.* Esta propiedad, sirve como punto de partida para poder explorar el espacio de hipótesis y buscar la hipótesis consistente. Formalizamos la subsunción en la definición 4.1. Ver apéndice A para los términos y definiciones de la lógica de primer orden.

DEFINICIÓN 4.1 *Sean dos cláusulas C y D, decimos que C subsume a D si existe una substitución* $\ominus$ *tal que* $C\ominus \subseteq D$*. Denotado por* $C \succeq D$ *[30].*

Utilizando la *subsunción* podemos determinar la relación de orden entre las hipótesis, def. 4.2.

DEFINICIÓN 4.2 *Sean dos cláusulas C y D decimos que C es más general que D si la subsume.*

De esta manera si queremos especificar una cláusula *C* que se encuentre dentro de un programa *h*, aplicamos una substitución $\ominus$ a *C* o le agregamos una literal, ver figura 4.1. Por otro lado si queremos generalizar una hipótesis C' le aplicamos una substitución inversa $\ominus^{-1}$ o le quitamos una literal.

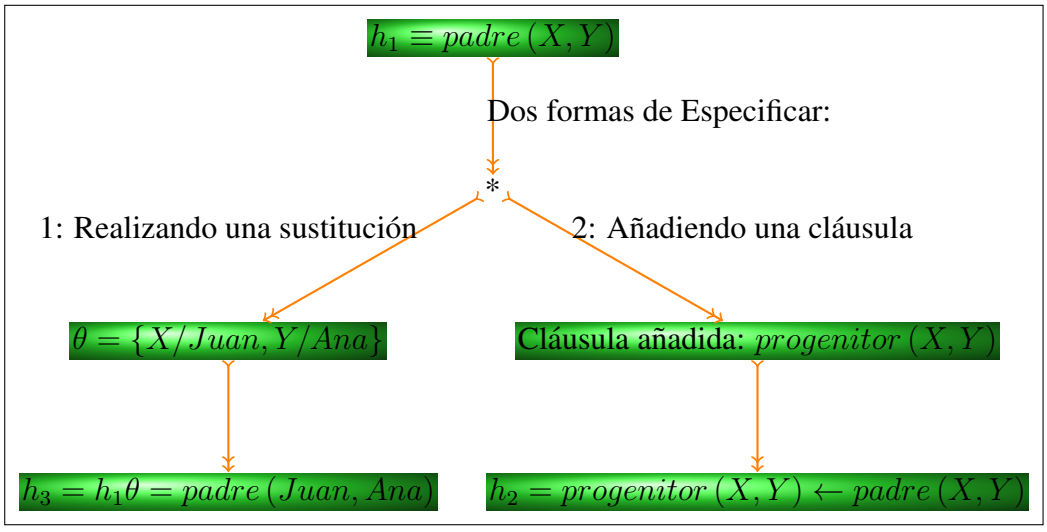


Figura 4.1: Especificación de una hipótesis en ILP

En los métodos vistos hasta ahora, las operaciones para recorrer el espacio de hipótesis son: generalizar y especificar, en la tabla 4.1 podemos apreciar la equivalencia entre ILP y árboles de decisión.

Operación	Equivalencia en ILP	Equivalencia en árboles de decisión
Especificación	Adición de literales a cláusulas	Crecimiento del árbol de decisión
Generalización	Eliminación de literales a cláusulas	Poda del árbol de decisión

Tabla 4.1: Equivalencia de generalización y especificación en ILP y árboles de decisión

En la sección 4.3 veremos la clasificación típica de los algoritmos de ILP basada en la forma de inicializar la hipótesis y de recorrer el espacio de búsqueda.

4.3 Clasificación de algoritmos de ILP

La relación de orden entre hipótesis, permite realizar la búsqueda de la hipótesis consistente. Para ver la necesidad de poner orden veamos un ejemplo sencillo. Supongamos que debemos buscar una persona en el directorio telefónico, en este caso el orden definido para realizar la búsqueda se da de manera natural utilizando el alfabeto. Basta imaginar lo complicado que sería buscar una persona en un directorio desordenado, para darse cuenta de lo esencial que es poner en orden un conjunto de datos donde se va a realizar una búsqueda.

La importancia de ordenar las hipótesis es evidente con el ejemplo anterior, sin embargo en nuestro contexto (inteligencia artificial), al igual que en otras disciplinas, es necesario formalizar cada definición que se realice, y el orden de las hipótesis no es la excepción. Por lo tanto en la definición 4.3 formalizamos la relación de orden.

DEFINICIÓN 4.3 *Una relación R sobre un conjunto S es llamada orden pacial si es reflexiva, antisimétrica y transitiva. Un conjunto S junto con la relación de orden pacial R es llamado conjunto de orden parcial y se denota como (S,R) [28].*

En aprendizaje maquinal la relación *R*, es aquella con la que indicamos que una hipótesis es mayor o menor que otra: $\leq$ (mayor o menor que) ó $\succeq$ (subsunción). Y el conjunto *S* sobre el que definimos la relación es el espacio de hipótesis.

Para introducir de manera formal la visualización gráfica de una relación de orden, veamos un pequeño ejemplo. Definimos una relación de orden parcial *R1* sobre el conjunto $S = \{A, B, C, D\}$, donde: $D \leq C \leq B \leq A$. Gráficamente podemos ver la relación *R1* como en la figura 4.2(a). Sin embargo, debido a que la relación es reflexiva no es necesario mostrar las flechas de un nodo a sí mismo, en la figura 4.2(b) eliminamos estas flechas. Ya que es transitiva, tampoco es necesario mostrar las flechas que van de un nodo a otro no consecutivo. Por último las flechas de un nodo a su *vecino* no se deben mostrar debido a que también se puede dar la igualdad (la propiedad antisimétrica permite *ir de un nodo a otro y viceversa*). El resultado es una retícula como la que mostramos en la figura 4.2(c).

La figura 4.2(c) representa una retícula o diagrama de Hasse. Pero ¿Qué relación tiene el orden parcial y el diagrama de Hasse con la clasificación de algoritmos de ILP?

La respuesta a la pregunta anterior tiene que ver con la elección de la hipótesis inicial, y con la dirección de la búsqueda a través de la retícula formada por la relación de orden. Los algoritmos que realizan el proceso de aprendizaje en ILP se clasifican en:

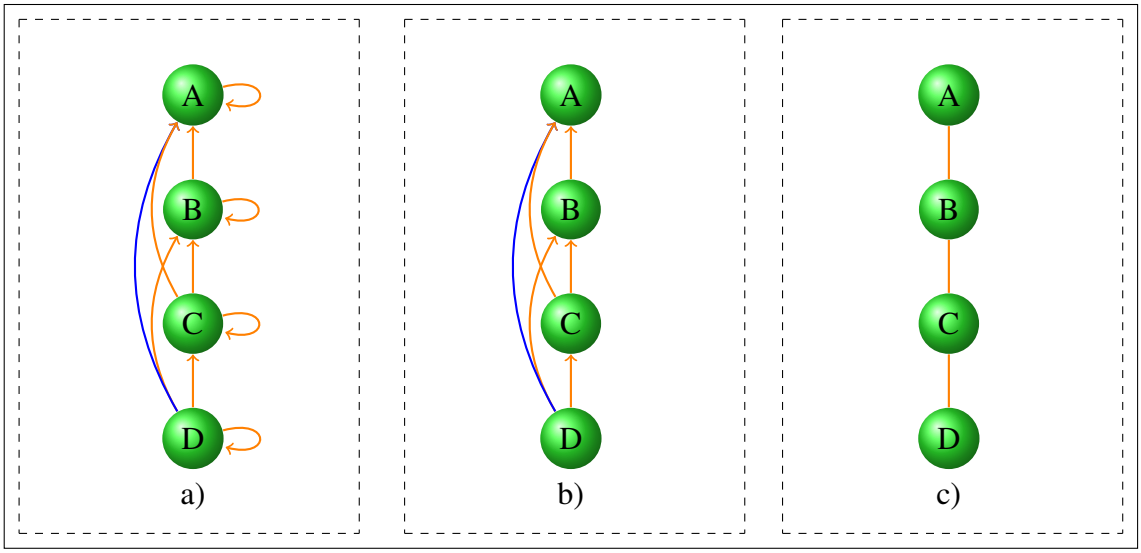


Figura 4.2: Relación de orden y formación de una retícula

- **Métodos Top-Down.** Son aquellos métodos que comienzan con la hipótesis más general posible y la especifican durante el proceso de aprendizaje, ver figura 4.3.
- **Métodos Bottom-Up.** Comienzan con una hipótesis muy especifica y la generalizan hasta encontrar la hipótesis consistente, ver figura 4.3.

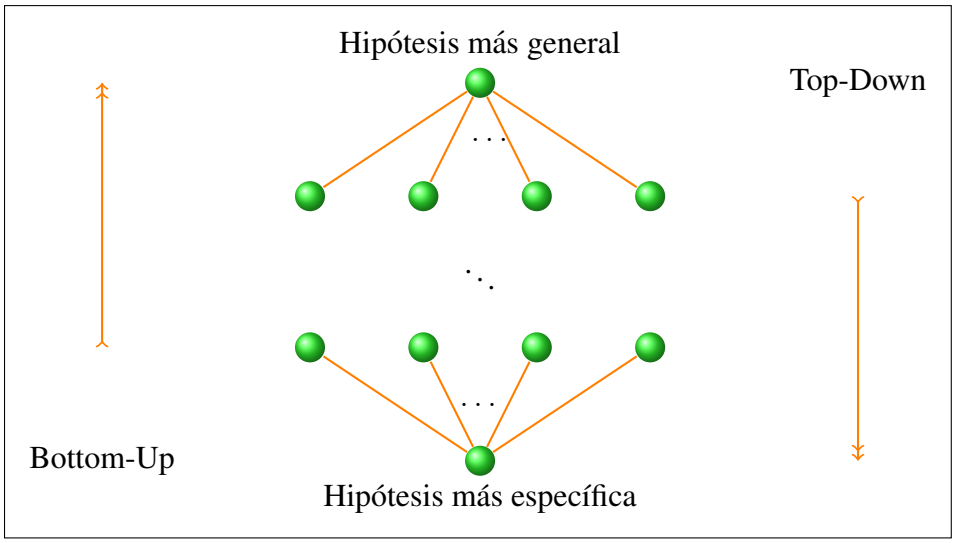


Figura 4.3: Tipos de algoritmos en ILP

Una hipótesis consistente es aquella que cubre sólo los ejemplos positivos. Pero cuando una hipótesis cubre uno o más ejemplos negativos decimos que es *fuerte* y entonces debemos especificarla; cuando una hipótesis no cubre uno o más ejemplos positivos decimos que es *débil* y entonces debemos generalizarla. Este proceso de especificación y generalización de hipótesis se resume en el algoritmo 4.1.

Algoritmo 4.1 *Esquema de ILP*

begin
  Entradas: B, E^-, E^+
  Salida: Un programa H tal que $H \cup B$ sea consistente
  while ($H \cup B$ no sea consistente) do
    if ($H \cup B$ es fuerte) then generalizar H fi
    if ($H \cup B$ es débil) then especificar H fi
  od
end

En la sección 4.5 presentamos un algoritmo de ILP que se encuentra dentro del grupo de los algoritmos de tipo top-down.

4.4 Prejuicio en ILP

Anteriormente mencionamos que el prejuicio es la toma de decisiones en la creación de los sistemas de AM [36]. También explicamos que tres de los principales prejuicios son: prejuicio de lenguaje, de búsqueda y de prevención de sobreentrenamiento. Sin embargo en ILP es común el prejuicio *declarativo*, el cual definimos a continuación.

El *prejuicio declarativo* define las restricciones en que se basará la formación de cláusulas e hipótesis [22]. Este tipo de prejuicio se divide en dos:

- **Prejuicio semántico.** Consiste en el uso de modos y tipos para guiar al algoritmo ILP a la formación de hipótesis bien formadas.
- **Prejuicio sintáctico.** Define la sintáxis que debe seguir cada cláusula para crear un conjunto de hipótesis bien formadas. Éste está muy relacionado con el prejuicio de lenguaje, que define el lenguaje que deben seguir las hipótesis para formar el espacio de búsqueda.

4.5 Algoritmo FOIL

FOIL (First Order Inductive Learner) es un algoritmo de ILP de tipo top-down creado por Ross Quinlan [23]. Este algoritmo comienza con la regla más general posible, y a partir de ésta realiza un proceso de especificación para encontrar una hipótesis consistente. En el algoritmo 4.2 mostramos los pasos necesarios para conseguir esto.

Algoritmo 4.2 *Algoritmo FOIL*

begin FOIL(E^+, E^-, Literales de B, $Cabeza \equiv Regla\_Objetivo$)
  Conjunto de reglas $\equiv \phi$
  while ($Ejemplos^+ \neq \phi$) **Generador de cláusulas**
    Regla más general $\equiv Cabeza :-$
    $Regla\_Actual \equiv Cabeza :-$

```
        while (Ejemplos⁻ ≠ φ) Especificador
            Crea espacio de búsqueda al añadir las literales en B y Cabeza a Regla_Actual
            Regla_Actual ≡Literal con mayor Ganancia de Información
            Elimina ejemplos en E⁻ que sean NO cubiertos por Regla_Actual
        od
        Añade Regla_Actual al Conjunto de Reglas
        Elimina ejemplos en E⁺ que SI sean cubiertos por Regla_Actual
    od
end
```

Los puntos importantes del algoritmo 4.2 son:

- Generador de cláusulas.
- Especificador.
- Creación del espacio de búsqueda.
- Elección de la mejor cláusula.

El *generador de cláusulas*, nos permite inicializar la hipótesis más general e invocar al especificador, del cuál obtendrá una regla que se pueda añadir al conjunto que formará la hipótesis final. En ILP, una hipótesis está formada por varias reglas. En el apéndice B se pueden consultar los términos de Prolog utilizados en esta sección.

El *especificador*, recibe del generador de cláusulas la regla más general, y la especifica hasta encontrar la que sea más *débil*, esto se logra cuando la regla no cubra ningún elemento negativo.

La *creación del espacio de búsqueda* se logra al añadir, al cuerpo de la regla actual, literales con ciertas restricciones [23], las cuáles indican que las literales que se añadan al cuerpo de la regla actual deben tener una de las siguientes cuatro formas: $X_i = X_j$, $X_i \neq X_j$, $Q(V_1, V_2, \ldots, V_r)$ o $\neg Q(V_1, V_2, \ldots, V_r)$, donde cada X_i es una variable existente, cada V_i es una variable existente o nueva en la regla actual y Q es alguna relación; además debemos resaltar lo siguiente:

- La literal añadida debe tener al menos una variable ya existente. Esto permite que haya una relación directa con la cabeza de la regla objetivo, y con cada una de las literales añadidas en iteraciones anteriores.
- Si Q es la misma relación que la regla objetivo, se deben añadir además otras restricciones para impedir ciclos infinitos. Las restricciones para evitar recursiones infinitas, se basan en definir una relación de orden entre las constantes, variables y literales de la regla actual [4], de manera que la literal que conforma la cabeza de la regla objetivo, sea estrictamente menor o mayor que la literal recursiva que se añade.
- La *Ganancia de información* utilizada como heurística para elegir a la mejor literal, permite una poda parecida al *alfa-beta* [1].

[1] Muchas veces el espacio de búsqueda puede crecer de manera exponencial, gráficamente lo podemos ver como un árbol que crece demasiado rápido. La poda alfa-beta evita que se genere todo el árbol, podando aquellos nodos o subárboles que no son necesarios [6]

Para elegir la mejor regla o cláusula, en FOIL se utiliza la misma heurística que en el algoritmo ID3 (algoritmo de inducción de árboles de decisión también desarrollado por Ross Quinlan): se elige la cláusula que proporcione la mayor ganancia de información [23]. Se define entonces la ganancia de información *G* al añadir la literal *L* en la ecuación 4.3.

$$Ganancia(L,G) = \left(log_2 * \frac{P_i^+}{P_i^- P_i^+} - log_2 * \frac{P_j^+}{P_j^- P_j^+} \right) \tag{4.3}$$

4.5.1 Ejemplo: *grafos*

Para aclarar el funcionamiento de FOIL veamos un ejemplo, supongamos que queremos aprender un programa que identifique bajo qué condiciones un nodo alcanza a otro en un grafo dado, figura 4.4. Llamemos a la regla objetivo (la que queremos aprender) como: *alcanza_a*.

En este punto, es necesario notar que el prejuicio de búsqueda en ILP, nos permite definir un espacio de búsqueda que contiene hipótesis recursivas, nuestro problema del grafo, es la búsqueda de una de éstas. La decisión aquí tomada, prejuicio de búsqueda, es la libertad de poder añadir la literal objetivo al cuerpo de las cláusulas.

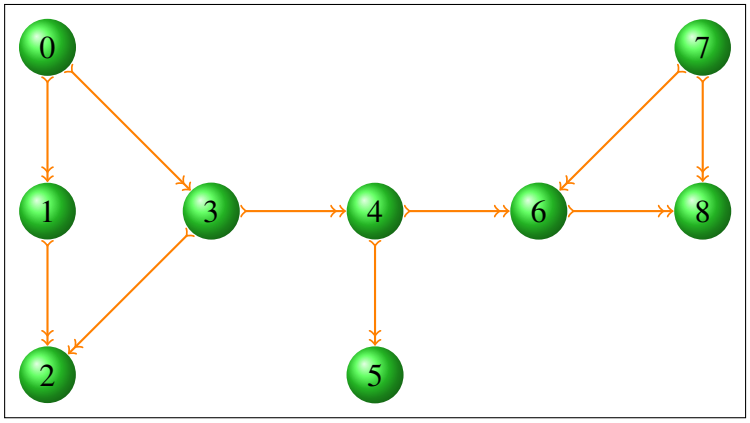


Figura 4.4: Ejemplo de problema ILP: grafo dirigido

El conocimiento previo *B*, es lo que debemos saber de antemano para poder inducir la regla que indique si un nodo del grafo puede alcanzar a otro. En este caso, lo único que conocemos es la relación entre dos nodos que indican cuándo están unidos o ligados. En la figura 4.4, podemos ver todas las relaciones de ligadura entre cada par de nodos, y las representamos en ILP como una serie de hechos (para este ejemplo utilizamos la notación de Prolog):

conectado(0,1).	conectado(4,5).
conectado(0,3).	conectado(4,6).
conectado(1,2).	conectado(6,8).
conectado(3,2).	conectado(7,6).
conectado(3,4).	conectado(7,8).

El conjunto de ejemplos positivos E^+, son aquellos hechos donde sabemos de antemano que un nodo alcanza a otro:

alcanza_a(0,1).	alcanza_a(3,5).
alcanza_a(0,2).	alcanza_a(3,6).
alcanza_a(0,3).	alcanza_a(3,8).
alcanza_a(0,4).	alcanza_a(4,5).
alcanza_a(0,5).	alcanza_a(4,6).
alcanza_a(0,6).	alcanza_a(4,8).
alcanza_a(0,8).	alcanza_a(6,8).
alcanza_a(1,2).	alcanza_a(7,6).
alcanza_a(3,2).	alcanza_a(7,8).
alcanza_a(3,4).	

El conjunto de ejemplos negativos E^-, es el conjunto de hechos *alcanza_a*, que no forman parte del conjunto de ejemplos positivos utilizando las constantes existentes (1,2,3,4,5,6,7,8,9,0); a esta forma de construir el conjunto de ejemplos negativos se le conoce como *mundo cerrado*.

Ya que hemos construido el conocimiento previo, el conjunto de ejemplos positivos y el conjunto de ejemplos negativos, el algoritmo comienza con la regla o hipótesis más general, que en este caso definimos como: $alcanza\_a(X, Y) : -$. Para que sea la más general posible utilizamos variables en lugar de constantes.

El siguiente paso se da dentro del ciclo interior del algoritmo, donde se añaden todas las posibles literales existentes que en este caso son: $ligado(X, Y)$ y $alcanza\_a(X, Y)$, por lo tanto las posibles reglas resultantes son:

- alcanza_a(X,Y) :- ligado(X,Y).
- alcanza_a(X,Y) :- ligado(Y,X).
- alcanza_a(X,Y) :- ligado(X,Z).
- alcanza_a(X,Y) :- ligado(Z,X).
- alcanza_a(X,Y) :- ligado(X,Z).
- alcanza_a(X,Y) :- ligado(Y,Z).
- alcanza_a(X,Y) :- ligado(Z,Y).
- alcanza_a(X,Y) :- alcanza_a(Z,X).
- alcanza_a(X,Y) :- alcanza_a(Y,Z).
- alcanza_a(X,Y) :- alcanza_a(Z,Y).
- alcanza_a(X,Y) :- X=Y.
- La negación de todas las anteriores.

Usando la ecuación 4.3, notamos que la regla $alcanza\_a(X, Y) : -ligado(X, Y)$ tiene la ganancia de información más alta. Como además esa regla resultante no cubre ningún ejemplo negativo, se eliminan todos, y el ciclo interior devuelve la regla encontrada para añadirla al programa lógico buscado. De manera contraria el ciclo exterior no termina, ya que la regla obtenida solo cubre 10 ejemplos positivos de los 19 existentes, entonces se eliminan los 10 cubiertos, y se continua con la ejecución del ciclo interior para buscar una nueva regla que se añada al programa lógico.

En la segunda iteración, se comienza con una nueva regla, que es la más general posible creada en el ciclo exterior, y con el conjunto de ejemplos negativos inicial, entonces el espacio de búsqueda es el mismo que se encontró en la iteración anterior. Pero en este caso la ganancia de información más alta la tiene la regla: $alcanza\_a(X, Y) : -ligado(X, Z)$, la cuál cubre sólo algunos ejemplos negativos, por lo que se eliminan del conjunto aquellos que no han sido cubiertos. En una segunda iteración del ciclo interior se tiene el siguiente espacio de búsqueda:

- alcanza_a(X,Y) :- ligado(X,Z), ligado(Z,X).
- alcanza_a(X,Y) :- ligado(X,Z), ligado(Y,Z).
- alcanza_a(X,Y) :- ligado(X,Z), ligado(Z,Y).
- alcanza_a(X,Y) :- ligado(X,Z), alcanza_a(Z,X).
- alcanza_a(X,Y) :- ligado(X,Z), alcanza_a(Y,Z).
- alcanza_a(X,Y) :- ligado(X,Z), alcanza_a(Z,Y).
- alcanza_a(X,Y) :- X=Y.
- La negación de todas las anteriores.

Donde la regla que tiene la información de ganancia más alta es:

$$alcanza\_a(X, Y) : -ligado(X, Z), alcanza\_a(Z, Y).$$

Con esta regla, no se cubre ningún ejemplo negativo, por lo que el ciclo interior devuelve esta regla para añadirla a la hipótesis.

En el ciclo exterior, se verifica que tomando en cuenta las dos reglas obtenidas, todos los ejemplos positivos restantes son cubiertos, por lo que se termina la ejecución del algoritmo, y se devuelve el conjunto de reglas creado:

- alcanza_a(X,Y) :- ligado(X,Y).
- alcanza_a(X,Y) :- ligado(X,Z), alcanza_a(Y,Z).

Estas dos reglas componen la hipótesis buscada. Debemos recalcar que en ILP las hipótesis son programas lógicos, por lo que pueden estar formadas por una, dos o más cláusulas.

4.6 Resumen

Además de las heurísticas utilizadas, los algoritmos de ILP, tienen como ventaja el uso de la lógica de primer orden para representar las hipótesis. El conocimiento previo, también es esencial para encontrar hipótesis más precisas. Es decir, que generalicen mejor los conceptos de aprendizaje.

Por otro lado, a veces el espacio de búsqueda crece exponencialmente. Por lo que es necesario buscar mejores heurísticas, que permitan identificar aquellos elementos del espacio de hipótesis que no son necesarios, y no tomarlos en cuenta (o podarlos).

En el siguiente capítulo, identificaremos las ventajas y desventajas de los dos métodos vistos hasta ahora (árboles de decisión y programación lógica inductiva), lo cual nos permitirá definir la problemática que abordaremos en el resto del presente trabajo.

Capítulo 5

Propuesta de ILP Multivalores

En los capítulos 3 y 4 hemos visto a grandes rasgos las características principales de dos técnicas de aprendizaje maquinal: árboles de decisión y programación lógica inductiva. Existen otros métodos (redes neuronales, aprendizaje bayesiano, aprendizaje por reforzamiento, algoritmos genéticos, etc.), sin embargo sería imposible, para el alcance de este trabajo, abarcarlos todos.

La elección de estas dos metodologías se debió principalmente a:

- En el caso de los árboles de decisión, a su capacidad de aprendizaje multivalores.
- Y al uso de las propiedades de la lógica de primer orden para representar e inducir las hipótesis en el caso de ILP.

En las siguientes secciones presentamos las ventajas y desventajas de estas dos metodologías. También exponemos una breve discusión acerca de la problemática identificada y la solución propuesta.

5.1 Ventajas y desventajas de los árboles de decisión

A continuación listamos las ventajas y desventajas de los árboles de decisión, las cuales nos servirán de base para la discusión de la problemática encontrada, y posteriormente para la propuesta planteada.

Ventajas

- **Clasificación:** Los árboles de decisión son en realidad clasificadores. Si tenemos un conjunto de ejemplos pertenecientes a N clases, un árbol de decisión se interpreta como un conjunto de N reglas que indican los valores que debe tener un elemento para pertenecer a cada una de esas clases.
- **Datos desconocidos:** Se pueden obtener árboles de decisión muy exactos a pesar de que el conjunto de entrenamiento tenga datos desconocidos.
- **Fácil preparación de los datos:** Preparar el conjunto de ejemplos para un algoritmo inductor de árboles de decisión es relativamente fácil, otros métodos realizan normalización

de datos u otros análisis para preparar los datos. En el caso de los árboles de decisión por lo general no hay que hacer absolutamente nada a los datos de entrenamiento.

- **Tipos de datos:** Para la inducción de árboles de decisión es posible utilizar datos numéricos, categóricos o ambos.

- **Multivalores:** Durante el crecimiento de los árboles de decisión muchos algoritmos utilizan un sólo valor por cada ramificación o subárbol, sin embargo hay técnicas que permiten utilizar más de un valor para inducir los árboles (aprendizaje multivalores). Este tipo de aprendizaje permite obtener árboles más compactos y en muchos casos más precisos que los univalores.

- **Multivariables:** Algunos algoritmos inductores de árboles de decisión, utilizan más de un atributo por cada nodo. Esto permite, al igual que con el aprendizaje multivalores, obtener árboles más compactos y más precisos.

- **Reducción de variables independientes**. Las reglas obtenidas de los árboles de decisión, no necesariamente están construidas con todos los atributos del conjunto de entrenamiento. Esto se debe a que este tipo de algoritmos, discriminan aquellas variables que no son adecuadas para una buena clasificación o un buen aprendizaje.

Desventajas

- **Árboles demasiado grandes:** Muchas veces cuando el concepto de aprendizaje tiene una gran cantidad de atributos es posible inducir árboles de decisión demasiado profundos y con muchas ramificaciones, esto tiene como consecuencia que el árbol sea difícil de interpretar y que haya sobreaprendizaje.

- **Prejuicio de selección:** La mayoría de algoritmos inductores de árboles de decisión eligen, durante su ejecución, cierto tipo de atributos sin importar que sean los mejores para clasificar los datos (prejuicio de selección). Algunos algoritmos eligen atributos categóricos, sobre todo cuando éstos tienen un gran número de categorías; otros algoritmos seleccionan atributos numéricos, [11].

- **Lenguaje poco expresivo:** Los árboles de decisión pueden interpretarse con el lenguaje de la lógica de proposiciones. Este lenguaje es más limitado que el usado en la lógica de primer orden, por lo que es evidente la desventaja respecto a la programación lógica inductiva.

5.2 Ventajas y desventajas de la ILP

Ya que hemos expuesto las ventajas y desventajas de los árboles de decisión, a continuación enlistamos las de ILP

Ventajas

- **Lenguaje:** El lenguaje utilizado por la ILP es el de la lógica de primer orden, éste es mucho más expresivo que la mayoría de los lenguajes utilizados por otros métodos de aprendizaje maquinal.
- **Conocimiento previo:** ILP introduce información extra llamada conocimiento previo que sirve para obtener mejores reglas (más precisas).
- **Recursividad:** El lenguaje utilizado por la ILP permite crear reglas recursivas, esto es muy útil al hacer aprendizaje donde la solución es naturalmente recursiva (p.e. factorial de un número).
- **Tipo de datos:** Combina cálculos numéricos con datos relacionales. Por lo general para implementar algoritmos de ILP se utiliza como lenguaje principal Prolog, lo que permite utilizar análisis matemático (no muy robusto) a datos relacionales.
- **Variedad en el lenguaje:** El lenguaje utilizado de manera general es el mismo que la lógica de primer orden, sin embargo el mismo usuario puede determinar su propio lenguaje, es decir, los predicados y los hechos definidos pertenecen al lenguaje del creador del sistema de ILP.

Desventajas

- **Espacio de búsqueda:** El espacio de búsqueda crece demasiado rápido, por lo que se vuelve más complejo buscar la hipótesis óptima.
- **Recursividad:** Parece contradictorio que la recursividad forme parte tanto de las ventajas como de las desventajas, pero no es así, de hecho es un *arma de doble filo*. Si bien la ventaja es que hace más robusto el lenguaje utilizado para crear hipótesis más completas, siempre existirá la posibilidad de crear reglas recursivas que tengan ciclos infinitos. El manejo de ciclos infinitos en definiciones recursivas no es trivial y se han publicado estudios al respecto, uno de ellos de Ross Quinlan [4].
- **Aprendizaje multivalores:** Los sistemas existentes sólo realizan aprendizaje univalor, esto quiere decir que cada vez que realizan la operación de generalización o especificación de reglas, se utiliza sólo un valor por cada aparición del atributo correspondiente.

5.3 Identificación de problemática y propuesta

En la tabla 5.1 se resumen los principales *pros y contras* mencionados anteriormente. De estos resaltamos en dicha tabla la falta de aprendizaje multivalores en ILP, problema principal tratado en esta tesis.

Al no realizar aprendizaje multivalores, los algoritmos ILP prueban cláusulas con literales cuyos atributos presentan un valor: *aprendizaje univalor*. Como consecuencia las hipótesis obtenidas pueden contener una gran cantidad de reglas o cláusulas, lo cual las hace más difíciles de interpretar. A continuación ejemplificamos este comportamiento.

Atributo	Árboles de decisión	ILP
Alta expresividad de lenguaje	No	Sí
Multivalores	Sí	No
Multivariable	Sí	No
Conocimiento Previo	No	Sí

Tabla 5.1: Ventajas y desventajas de árboles de decisión e ILP

5.3.1 Aprendizaje Univalor

Supongamos que la hipótesis devuelta por algún algoritmo ILP contiene, entre otras, las cláusulas mostradas en el conjunto de reglas 5.1, donde cada Q_i representa una conjunción de literales. En este conjunto de cláusulas, podemos notar que en cada una de ellas la literal $p(A, B)$ declara un sólo valor del atributo categórico B (con valores en $v_1, v_2, \ldots, v_{15}$). De hecho hay una cláusula por cada valor de B, por lo tanto si la hipótesis final presentara más valores de B entonces la hipótesis estaría compuesta por un mayor número de cláusulas. Por otro lado si en dicha hipótesis estuvieran declarados más atributos de manera análoga a B, entonces es posible que la hipótesis final se construya con una gran cantidad de cláusulas[1].

$$\begin{aligned} T &\leftarrow Q_1, p(A, v_1), Q_2. \\ T &\leftarrow Q_3, p(A, v_3), Q_4. \\ T &\leftarrow Q_5, p(A, v_4), Q_6. \\ T &\leftarrow Q_7, p(A, v_{14}), Q_8. \\ T &\leftarrow Q_9, p(A, v_{15}), Q_{10}. \end{aligned} \tag{5.1}$$

Esta relación entre el número de atributos y valores, y el número de cláusulas en las hipótesis, tiene implicaciones que vale la pena mencionar. En primer lugar entre más atributos y valores se declaren en el conocimiento previo, es más probable que las hipótesis contengan más cláusulas por lo que será más difícil interpretarlas. Por otro lado cuando se prueba un valor a la vez, no se garantiza que el conjunto de valores declarados en la hipótesis obtenida sea el mejor. Por lo tanto consideramos necesario crear un método que elija el mejor conjunto de valores para cada atributo, y lo declare en una cláusula, de manera que las hipótesis creadas contengan menos cláusulas. A partir de lo mencionado anteriormente podemos formularnos las siguientes preguntas:

1. ¿Cómo crear cada subconjunto de valores (para cada atributo) de manera que cada uno sea el mejor? En nuestro ejemplo los valores elegidos por el algoritmo ILP (hipotético) son $v_1, v_3, v_4, v_{14}, v_{15}$, sin embargo los algoritmos actuales no crean el subconjunto, sino que prueban a la vez un solo valor, por lo tanto es necesario determinar una manera general de elegir cada subconjunto de valores. Hablamos de subconjuntos de valores ya que la hipótesis no presenta todos los valores posibles para cada atributo.

[1] Es difícil afirmar a partir de cuantos atributos y cuantos valores podemos decir que son muchos, también resulta impreciso decir que una hipótesis contiene demasiadas cláusulas ya que es subjetivo. Sin embargo lo importante para nosotros es que entre más atributos y valores se presenten en el conocimiento previo, las hipótesis resultantes pueden tener más reglas y por lo tanto se vuelven más difíciles de interpretar.

2. ¿Cuántos subconjuntos crear para cada atributo?

3. ¿Cómo representar cada conjunto de valores en una sola cláusula?

A continuación presentamos la propuesta con la cual contestamos las preguntas anteriores

5.3.2 Propuesta

Antes de presentar nuestra propuesta es necesario definir los dos tipos de cláusulas que mencionaremos en el resto del documento:

DEFINICIÓN 5.1 ***Cláusulas univalor.*** *Son aquellas que presentan un valor por cada atributo declarado en cada una de las literales del cuerpo de la cláusula, por ejemplo:*

$$T \leftarrow p_1\left(A, 5.4\right), p_2\left(B, valor1\right), p_3\left(A, B, 12\right)$$

DEFINICIÓN 5.2 ***Cláusulas multivalores.*** *Si al menos una literal del cuerpo de la cláusula presenta más de un valor o hace referencia a un conjunto de valores de algún atributo, entonces la llamaremos cláusula multivalores, por ejemplo:*

$$T \leftarrow p_1\left(A, 5.4\right), p_2\left(B, [valor1, valor2, valor3]\right), p_3\left(A, B, 12\right)$$

$$T \leftarrow p_1\left(A, 5.4\right), p_2\left(B, valor1\right), B > 45.6$$

En la sección anterior formulamos tres preguntas, y para contestar a ellas proponemos un método que consta de los siguientes cuatro pasos:

1. **Creación de subconjuntos de valores.** Implementar algoritmos inductores de árboles de decisión que realicen una partición sobre cada conjunto de valores correspondientes a cada atributo. Los datos analizados serán aquellos declarados en el conocimiento previo. Específicamente implementaremos aquella parte que realiza la división del nodo con criterio de división multivalor (visto en la sección 3.2.2). Para no crecer demasiado el espacio de búsqueda la implementación realizará una división binaria sobre cada conjunto de valores. De esta manera:

   - Para cada atributo categórico con valores en el conjunto $C = \{v_1, \ldots, v_m\}$ se obtendrán dos subconjuntos C_1 y C_2, de tal manera que $C_1 \cap C_2 = \phi$.
   - Para cada atributo numérico se obtendrá un punto de división d, el cual permitirá dividir el conjunto de valores en dos subconjuntos. El primero de ellos con elementos menores o iguales a d, y el segundo con valores estrictamente mayores.

2. **Creación de cláusulas multivalores.** Cada subconjunto de valores se utilizará para crear las cláusulas multivalores, y al igual que en el punto anterior se tratará de manera distinta a cada tipo de atributo.

- Atributo categórico. Cada subconjunto de valores se declarará en la literal correspondiente en lugar de un único valor. De esta manera se crearán dos nuevas cláusulas multivalores.
- Atributo numérico. Para este tipo de atributos también se crearán dos cláusulas multivalores, en la primera de ellas se utilizará el punto de división d, calculado en el paso anterior, para determinar aquellos elementos que son menores o iguales a d. En la segunda cláusula el punto d determinará aquellos valores del atributo numérico que son estrictamente mayores.

3. **Modificación del conocimiento previo.** El siguiente paso, una vez que se crearon las cláusulas multivalores, es añadirlas al conocimiento previo. De esta manera enriqueceremos la información que proporciona el conocimiento previo, y habremos debilitado el prejuicio de lenguaje. Con esto se permite a los algoritmos ILP agregar a su espacio de búsqueda cláusulas multivalores.

4. **Aprendizaje y análisis.** Después de añadir las cláusulas multivalores al conocimiento previo, esta información será analizada con algoritmos ILP para obtener hipótesis construidas con cláusulas multivalor, con lo cual se espera que estas hipótesis contengan menos reglas.

Al añadir las cláusulas multivalores al conocimiento previo, no sólo estamos modificando el prejuicio de lenguaje, también se modifican los siguientes prejuicios:

- **Prejuicio de búsqueda.** En ILP, este prejuicio también es conocido como *prejuicio sintáctico (syntactic bias)* [22] y se refiere al tipo de hipótesis que formarán el espacio de búsqueda. En nuestro caso al permitir la adición de cláusulas multivalores, se estará cambiando el formato de las hipótesis del espacio de búsqueda.

- **Prejuicio declarativo.** No todos los problemas de ILP requerirán el uso del aprendizaje multivalores, por lo que el usuario del algoritmo implementado tendrá la posibilidad de indicar si se deben añadir las nuevas cláusulas. Esta decisión por parte del usuario, es otro tipo de prejuicio llamado *declarativo (declarative bias)* [12].

5.4 Resumen

Cada técnica de aprendizaje maquinal tiene sus ventajas y desventajas, y la necesidad de enriquecerlas nos lleva no sólo a mejorar cada aspecto de tales técnicas, sino a tomar lo mejor de cada una y unirlos en un sólo método. En la propuesta que presentamos tomamos esencialmente una de las ventajas de los árboles de decisión (aprendizaje multivalores) y lo añadimos a la ILP para enriquecer su lenguaje. En el capítulo 6 explicamos con más detalle el método propuesto y los algoritmos utilizados.

Capítulo 6

Aprendizaje Multivalores en ILP

El método propuesto en el capítulo 5 permite pasar del aprendizaje ILP univalor al multivalores, por lo que en la siguiente sección ampliamos el concepto de *aprendizaje multivalores*.

Posteriormente presentamos con mayor detalle nuestro método, el cual se resume en los siguientes pasos:

1. Creación de subconjuntos de valores para cada atributo.
2. Creación de cláusulas multivalores.
3. Adición de cláusulas multivalores al conocimiento previo.
4. Análisis de conjunto de ejemplos y conocimiento previo (obtenido del paso anterior) con algoritmos ILP.

Para el paso 1 implementamos el criterio de división multivalores (visto en la sección 3.2.2) de dos algoritmos inductores de árboles de decisión: *QUEST (Quick Unbiased Efficient Statistical Tree)* ver [16] y *CRUISE (Classification Rule with Unbiased Interaction Selection and Estimation)* ver [11]. Estos algoritmos los presentamos en las dos últimas secciones de este capítulo. Cabe añadir que QUEST y CRUISE fueron elegidos tomando en cuenta:

- La exactitud con la que clasifican los datos.
- No tienen una complejidad espacial ni temporal muy grande.
- Utilizan métodos que eliminan el prejuicio de selección.
- Mantienen las características anteriores incluso con datos desconocidos.

En cuanto al cuarto paso, para analizar cada problema, se utilizaron dos algoritmos:

- *FOIL.* Este algoritmo presentado en la sección 4.5 fue elegido debido a su simplicidad para formar las hipótesis, lo cual permite que la implementación haya sido relativamente fácil. Otra razón para su elección es que es de tipo *top-down* al igual que los algoritmos inductores de árboles de decision.

- *Progol.* Utilizamos el sistema *Aleph* creado por Ashwin Srinivasan [33]. Dicho sistema implementa el algoritmo *Progol* desarrollado por Stephen Muggleton [20]. Este algoritmo fue elegido ya que es uno de los más utilizados para resolver problemas de ILP. Describimos este algoritmo en el apéndice E.

6.1 Aprendizaje Multivalores

Antes de entrar de lleno a los elementos principales de la propuesta presentada, en esta sección retomamos el concepto de aprendizaje multivalores, introducida junto con el método de inducción de árboles de decisión.

En el capítulo 3, mencionamos que en el aprendizaje univalor cada rama representa un sólo valor del atributo, en la figura 6.1 se muestra un nodo dividido de esta forma. Sin embargo, el principal inconveniente de este tipo de aprendizaje, es que cuando existen atributos con muchos valores, entonces los árboles obtenidos son demasiado grandes, en la figura 6.2 se muestra un árbol con este problema. Esto puede causar, en algunos casos, no solamente que sean más difíciles de interpretar, sino que la clasificación sea ineficiente.

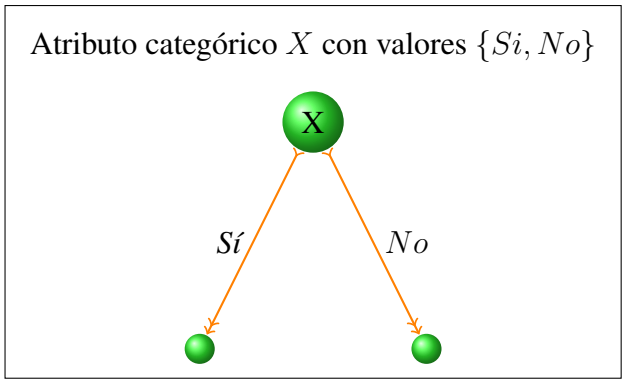


Figura 6.1: Árbol compacto con aprendizaje univalor

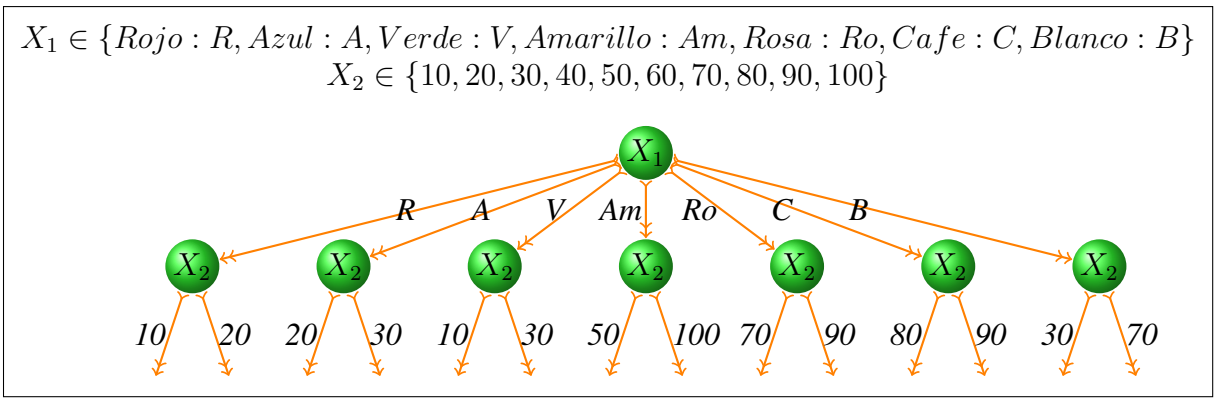


Figura 6.2: Árbol complejo con aprendizaje univalor

La alternativa a este tipo de aprendizaje, es el uso de un subconjunto de valores por cada subárbol (aprendizaje multivalores). El árbol mostrado en la figura 6.3, es obtenido con este

tipo de aprendizaje. Como podemos apreciar, dicho árbol es más pequeño que el obtenido con aprendizaje univalor (mostrado en la figura 6.2).

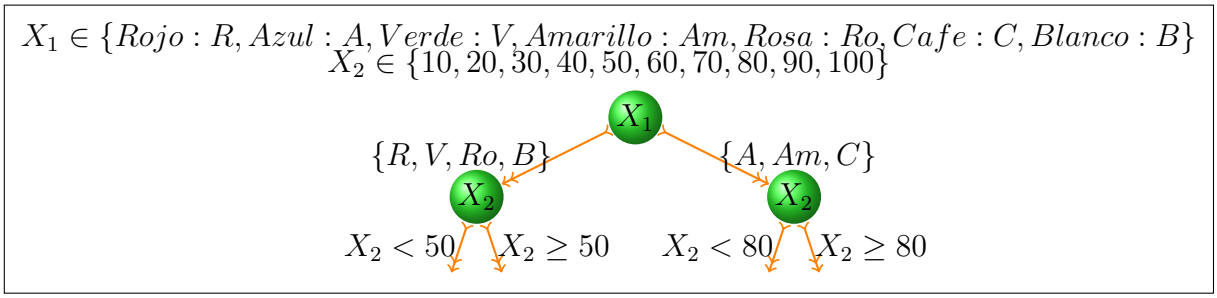


Figura 6.3: Árbol con aprendizaje multivalores

Para su implementación, los algoritmos inductores de árboles multivalores, no son sencillos sin embargo el resultado final tiene mayores ventajas, algunos no tienen prejuicio de selección (ver CRUISE en [11]), la clasificación es más precisa y son más rápidos de inducir. Estas ventajas del aprendizaje multivalores, son las que queremos proporcionarle a la programación lógica inductiva. En seguida exponemos el método propuesto en el capítulo anterior.

6.2 Espacio de búsqueda multivalores en ILP

El método que hemos planteado afecta únicamente al espacio de búsqueda de los algoritmo ILP, ya que no modificamos ni la heurística ni la estrategia de búsqueda de dichos algoritmos. A continuación describimos cada uno de los cuatro pasos que componen nuestro método.

6.2.1 Creación de subconjuntos

Este paso es el más importante del método ya que permitirá el aprendizaje multivalores en ILP. Este consiste en la partición de los valores, de cada atributo. Esta división se lleva a cabo con la implementación del criterio de división multivalores de QUEST y CRUISE, de manera que cada algoritmo realizará una partición binaria por cada atributo. Ya que cada atributo puede ser categórico o numérico, debemos describir la división binaria para cada tipo.

Subconjuntos con valores categóricos

Sea el atributo categórico X con valores en el conjunto $C = \{v_1, \ldots, v_m\}$, entonces QUEST o CRUISE realizan una partición binaria sobre el conjunto de datos contenido en el conocimiento previo correspondiente a X, de manera que se obtienen dos subconjuntos de valores categóricos C_1 y C_2 tal que $C_1 \cap C_2 = \phi$ y $C_1 \cup C_2 = C$.

Veamos ahora un ejemplo que permitirá aclarar este paso para atributos categóricos. Supongamos que tenemos la literal $numeroLados(Figura, Numero)$, donde el atributo *Numero* es categórico con valores en $C = \{3, 4, 5, 6, 7, 8\}$. Si el objetivo del algoritmo ILP está relacionado con la clasificación de aquellas figuras que son triángulos y aquellas que no lo son, de acuerdo al número de lados, entonces QUEST o CRUISE hacen la partición del conjunto de datos, correspondientes a *Numero*, devolviendo los conjuntos $C_1 = \{3\}$ y $C_2 = \{4, 5, 6, 7, 8, 9\}$.

Subconjuntos con valores numéricos

Sea el atributo numérico X con valores en el conjunto de los reales $\mathbb{R}$, entonces QUEST o CRUISE realizan una partición binaria sobre el conjunto de datos contenido en el conocimiento previo correspondiente a X, de manera que se obtiene el punto de división d, que permite dividir el conjunto de valores que puede tomar X. El punto de división d permitirá identificar aquellos valores que son menores o iguales a d y los que son estrictamente mayores.

Para ejemplificar este paso supongamos que tenemos la literal

$$edad\,(NombrePersona, Edad)$$

donde el atributo $Edad$ es de tipo numérico con valores en el conjunto de los reales $\mathbb{R}$. Si el objetivo del algoritmo ILP está relacionado con la clasificación de personas tomando en cuenta si son mayores de edad (tomamos como 18 años la mayoría de edad), entonces QUEST o CRUISE realizan la partición del conjunto de datos, correspondientes a *Edad* , devolviendo $d = 18$.

6.2.2 Creación de cláusulas multivalores

Una vez que se tiene los subconjuntos de valores o el punto de división, calculados por QUEST o CRUISE, el siguiente paso es crear las cláusulas multivalores. Al igual que en el paso anterior es necesario realizar una descripción para cada tipo de atributo.

Cláusulas multivalores con datos categóricos

Sea la literal $p\,(\ldots, AtributoC, \ldots)$, donde $AtributoC$ es un atributo de tipo categórico con valores en el conjunto $C = \{v_1, \ldots, v_m\}$ y sean los subconjuntos C_1 y C_2 creados por QUEST o CRUISE, entonces creamos las cláusulas multivalores:

$$pA\,(\ldots) \leftarrow p\,(\ldots, AtributoC, \ldots)\,, member\,(AtributoC, C_1)$$

$$pB\,(\ldots) \leftarrow p\,(\ldots, AtributoC, \ldots)\,, member\,(AtributoC, C_2)$$

Debemos señalar que las literales pA y pB contienen los mismos argumentos que la literal p a excepción del atributo categórico $AtributoC$. El predicado *member* nos permite declarar un conjunto de valores de manera que en el proceso de aprendizaje se pueda probar cada subconjunto en una sola cláusula.

Continuando con el ejemplo de la sección 6.2.1 del caso categórico, las literales creadas serían las siguientes.

$$numeroLadosA\,(Figura) \leftarrow numeroLados\,(Figura, Numero)\,, member\,(Numero, [3])$$

$$numeroLadosB\,(Figura) \leftarrow numeroLados\,(Figura, Numero)\,, member\,(Numero, [4, 5, 6, 7, 8])$$

Cláusulas multivalores con datos numéricos

Sea la literal $q(\ldots, AtributoN, \ldots)$, donde $AtributoN$ es un atributo de tipo numérico con valores en el conjunto de los reales $\mathbb{R}$ y sea el punto de división d calculado por QUEST o CRUISE, entonces creamos las cláusulas multivalores:

$$qA(\ldots) \leftarrow p(\ldots, AtributoN, \ldots), AtributoN \leq d$$

$$qB(\ldots) \leftarrow p(\ldots, AtributoN, \ldots), AtributoN > d$$

En este caso el punto de división d permite identificar los dos subconjuntos de valores del atributo numérico, separándolos en aquellos valores que son menores o iguales a d y aquellos que son estrictamente mayores.

Las cláusulas para el ejemplo de la sección 6.2.1 en el caso numérico son:

$$edadA(NombrePersona) \leftarrow edad(NombrePersona, Edad), Edad \leq 18$$

$$edadB(NombrePersona) \leftarrow edad(NombrePersona, Edad), Edad > 18$$

6.2.3 Adición de cláusulas al conocimiento previo y análisis

Una vez creado cada par de cláusulas multivalores, estas se añaden al conocimiento previo del problema analizado. Esto permite enriquecer, con nuevas cláusulas con información multivalores, el lenguaje que utilizará cada algoritmo ILP al crear el espacio de búsqueda.

Como último paso del método tenemos el análisis del problema con algoritmos ILP. Como mencionamos al inicio de este capítulo, para realizar el análisis utilizamos los sistemas FOIL y Aleph.

Al aplicar este método a problemas ILP, debemos reflexionar sobre los siguientes puntos:

- ¿Se obtendrán hipótesis con menos reglas?
- ¿Deben analizarse todos los atributos declarados en el conocimiento previo?
- ¿Que características debe tener un atributo para ser analizado con los algoritmos QUEST y CRUISE?

En el capítulo siguiente analizamos algunos problemas ILP para determinar si efectivamente se obtienen hipótesis con menos reglas, y en el capítulo posterior determinaremos las características que debe tener cada atributo para ser analizado con aprendizaje multivalores. Para terminar este capítulo presentamos en las siguientes dos secciones los algoritmos QUEST y CRUISE, los cuales serán descritos completamente aún cuando se utiliza solamente el criterio de división de nodo en el método descrito anteriormente.

6.3 Algoritmo QUEST

Este algoritmo inductor de árboles de decisión implementa un criterio de división univariable y multivariable, pero en el desarrollo de nuestra propuesta sólo utilizamos el univariable.

Como cualquier algoritmo que induce árboles de decisión, QUEST implementa: selección de la variable o atributo, división del atributo, criterio de paro, manejo de valores desconocidos y poda. Estos puntos los detallamos a continuación.

6.3.1 Selección de variable

Un algoritmo que divide cada nodo en un número de subárboles equivalente al número de valores que tiene el atributo a dividir, sólo puede resolver problemas con variables que tengan un número finito de valores. Por ejemplo, si el atributo elegido para cierto nodo tiene los valores: blanco, negro, azul, rojo y verde, entonces habrá un subárbol por cada uno de estos colores. Por otro lado si un atributo tiene valores que pueden caer dentro de un conjunto infinito, como el de los reales, entonces es imposible crear un subárbol por cada uno de estos valores. En consecuencia estos algoritmos no pueden resolver problemas con este tipo de valores, a menos claro, que se discretice dicho conjunto, sin embargo en muchas ocasiones esto no es práctico ya que los árboles inducidos devolverán reglas imprecisas.

Los algoritmos que hemos elegido superan este problema al tratar problemas con variables que pueden ser de dos tipos:

- **Variable numérica.** Este tipo de variables toman sus valores del conjunto de los reales.
- **Variable categórica.** Son variables que pueden ser medidas usando un valor finito de valores o categorías.

El que mencionemos de manera introductoria los tipos de variable que pueden procesar los algoritmos que hemos implementado, tiene como origen el hecho de que cada variable se analiza y se procesa de manera diferente dependiendo del tipo al que pertenece.

En la elección del mejor atributo, se realiza un análisis de varianza (ANOVA presentado en el apéndice C.1) por cada una de las variables numéricas. Cuando el problema de aprendizaje o clasificación tiene dos o más clases, podemos estimar la media por cada una de estas clases utilizando los valores del atributo que estamos analizando. Al realizar el ANOVA podemos determinar, para cada atributo numérico, si todas las medias son iguales o no, y el atributo cuyas medias difieran más entre sí entonces es el atributo que mejor clasifica los datos, de entre todos los numéricos, y automáticamente se convierte en uno de los atributos que puede ser elegido para representar al nodo que se va a dividir.

Pero ¿por qué es mejor el atributo cuyas medias son distintas que aquel con medias iguales?. Comparemos dos atributos, el primero es una variable cuyas medias (de dos clases) son iguales. Cómo podemos ver en la figura 6.4, el punto de división entre las dos medias no se puede determinar, por lo que si ese atributo fue el elegido para representar al nodo, no será posible decidir dónde debe dividirse dicho nodo. El resultado claramente es la obtención de una regla que no clasifica el conjunto de ejemplos de manera correcta, en otras palabras, es como adivinar los valores que debe tener cada atributo para crear una regla de clasificación.

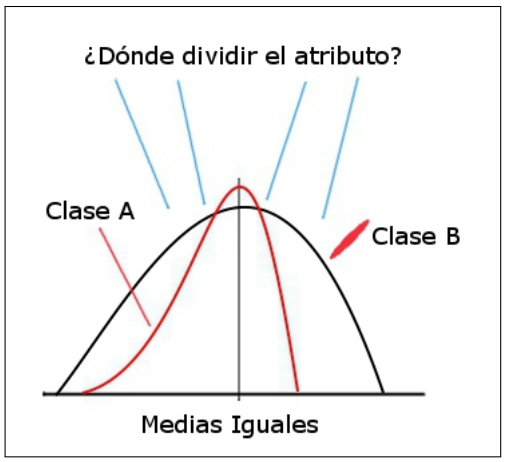


Figura 6.4: División sobre atributo con medias iguales

Cuando las medias son distintas, como en el atributo esquematizado en la figura 6.5, es relativamente sencillo determinar el punto de división. En consecuencia al realizar el análisis ANOVA, para cada atributo numérico, aquel con el *p-valor*[1] más pequeño será el atributo con las medias más separadas o distintas entre sí, por lo que será uno de los atributos más idóneos para representar el nodo en cuestión.

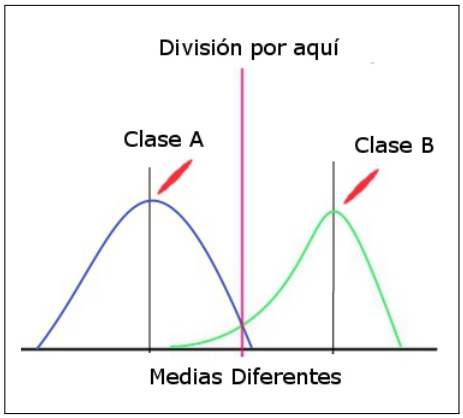


Figura 6.5: División sobre atributo con medias diferentes

En cuanto a las variables categóricas si el conjunto de entrenamiento contiene este tipo de atributos, entonces es necesario definir cuál de ellos es superior, de manera que podamos compararla con la mejor variable numérica y determinar enseguida si la que representará al nodo siguiente, en el crecimiento del árbol, será la numérica o la categórica.

El análisis correspondiente para cada variable categórica es el de la *Ji-cuadrada* (en el apéndice C.2 se detalla este análisis). Si convirtiéramos los valores categóricos a numéricos y calculáramos el ANOVA para cada una de estas variables, entonces las variables categóricas

[1]En los análisis estadísticos de los algoritmos multivalor que implementamos, calculamos el *p-valor*, el cuál, es el nivel de significancia más pequeño que conduce al rechazo de una hipótesis (denotada por H_0) [5]. En nuestro proyecto, este valor es un parámetro que nos permite identificar aquella variable que mejor clasifica el conjunto de entrenamiento.

tendrían más probabilidad de ser elegidas (prejuicio de selección). Para evitar esto y asegurar que todas las variables, sin importar su tipo, tengan la misma probabilidad de ser elegidas, se realiza para cada atributo categórico el análisis de la Ji-cuadrada de manera que podamos identificar aquellas variables que son independientes de las clases. Identificamos la mejor variable categórica, como aquella con el p-valor más pequeño.

Ya que tenemos la mejor variable de cada tipo, la categórica y la numérica, entonces debemos determinar si la mejor de las dos es lo suficientemente buena para clasificar el conjunto de ejemplos, la denotamos $X_{k'}$. De ser así, entonces elegimos esa variable para el nodo siguiente, en caso contrario se lleva a cabo un análisis de *Levene* (ver apéndice C.3) sobre cada variable numérica, y se elige aquella con el p-valor más pequeño, la denotamos como $X_{k''}$. Si esta última es mejor que $X_{k'}$, entonces es elegida, en caso contrario elegimos $X_{k''}$.

Este proceso que describimos en los párrafos anteriores lo resumimos a continuación (pero también se puede consultar en [11] con el nombre de algoritmo *1D*).

Algoritmo 6.1 *Algoritmo 1D*

Sea α el nivel de significancia elegido (por default es 0.05). Suponemos además que las variables $X_1, \ldots, X_{K_1}$ y $X_{K_1+1}, \ldots, X_k$ son numéricas y categóricas respectivamente.

1. *Realizar un análisis ANOVA a cada variable numérica y calcular su p-valor. Supongamos que X_{k_1} tiene el p-valor más pequeño $\hat{\alpha_1}$.*

2. *Para cada variable categórica, formar una tabla de contingencia con los valores categóricos como filas y los valores de las clases como columnas y encontrar su p-valor al realizar el análisis χ^2, sea $\hat{\alpha_2}$ el p-valor más pequeño y su variable asociada X_{k_2}.*

3. *Definir*
$$k' = \begin{cases} k_1, & \hat{\alpha_1} \leq \hat{\alpha_2} \\ k_2, & \hat{\alpha_1} > \hat{\alpha_2} \end{cases}$$

4. *Si $min\left(\hat{\alpha_1}, \hat{\alpha_2}\right) < \alpha/K$ (primera corrección Bonferroni[2]), entonces elegir $X_{k'}$ como la variable a dividir.*

5. *En otro caso, encontrar el p-valor del análisis de Levene llevado a cabo para cada variable numérica. Supongamos que $X_{k''}$ tiene el p-valor más pequeño $\tilde{\alpha}$.*

   a Si $\tilde{\alpha} < \alpha/\left(K + K_1\right)$, elegir $X_{k''}$ (segunda correción Bonferroni).

   b En otro caso, escoger $X_{k'}$.

Después de elegir el mejor atributo para dividir el nodo, el siguiente paso es precisamente encontrar el punto donde debe dividirse dicho nodo, en la siguiente sección profundizamos más en este punto.

[2]Cuando los análisis estadísticos no son significativos, esta corrección nos permite identificar el atributo con el p-valor más pequeño, es decir, el atributo que divide mejor el conjunto de ejemplos.

6.3.2 Punto de división

Una vez que hemos elegido el atributo, que corresponde a un nodo interno de un árbol de decisión, debemos definir el número de subárboles que debe tener dicho nodo, suponiendo que ningún criterio de división de paro se cumple, ya que de ser así entonces el nodo no se dividirá y se considerará como una hoja. Para ello hay que tomar en cuenta el tipo de atributo y el número de divisiones en que se partirá.

En la implementación que realizamos, cada nodo se dividirá en dos subárboles, por lo que al elegir un atributo X debemos encontrar un solo punto de división.

Si el atributo X es numérico entonces el punto de división P nos indicará que un subárbol estará formado por aquellos elementos del conjunto de ejemplos cuyos valores correspondientes a X son menores o iguales a P. Y el otro subárbol estará compuesto por los elementos del conjunto de ejemplos con valores correspondientes a X mayores a P.

Para un atributo X que es categórico, y cuyos valores forman el conjunto $C = \{v_1, v_2, \ldots, v_n\}$, el punto de división P nos permite dividir el conjunto C en dos subconjuntos C_1 y C_2 de tal manera que un subárbol estará formado por aquellos elementos del conjunto de ejemplos cuyos valores correspondientes a X pertenecen al conjunto C_1 y el otro subárbol será compuesto por los elementos del conjunto de ejemplos cuyos valores correspondientes a X están en C_2.

Punto de división para variable numérica

Uno de los objetivos de QUEST es crear árboles más sencillos de manera que puedan interpretarse fácilmente. Sin embargo, si cada nodo se divide en tantos subárboles como clases tenga el problema de clasificación, entonces el árbol resultante podría ser demasiado complejo para poder interpretarse. La solución es dividir cada nodo en sólo dos subárboles.

Para asegurar que cada nodo se divida únicamente en dos, a pesar de que el problema de clasificación tenga más de dos clases, QUEST procesa los valores del conjunto de ejemplos correspondientes a la variable elegida con el algoritmo *k-means*, con $k = 2$. Esto crea dos clases (llamadas superclases) a partir del conjunto analizado.

En general este algoritmo crea K grupos o *clusters* a partir de un conjunto de N datos con I dimensiones. Un elemento $i \in N$, pertenecerá a un cluster k_j si la distancia de i al centroide de k_j es menor que la distancia de i al centroide de los demás clusters (por lo general se utiliza la distancia euclidiana). En [17] podemos encontrar la descripción detallada de este algoritmo. A continuación presentamos el algoritmo general:

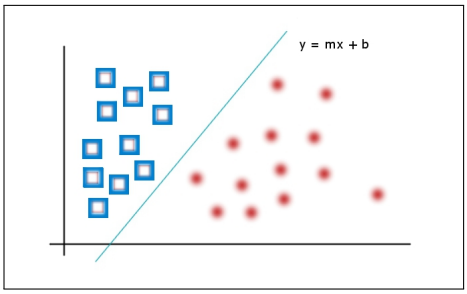
Algoritmo 6.2 (h) *Algoritmo k-means*

Sea G el conjunto de N elementos I-dimensionales que se agruparán
Especificar de manera aleatoria los k centroides
while Ningún elemento de G cambia de cluster do
    Se determina la distancia de los N elementos a cada uno de los k centroides
    Se agrupan los N elementos a los k centroides: de acuerdo al centroide más cercano
    Se recalculan los nuevos k centroides
od

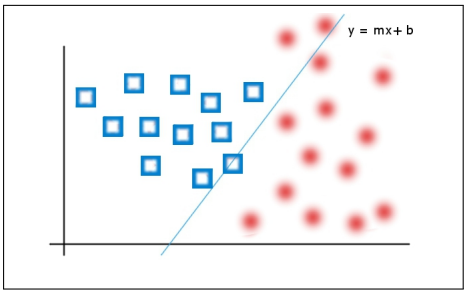
Una vez que se haya reestructurado el conjunto objetivo en dos superclases, debemos encontrar el punto que divide a las superclases, para ello se lleva a cabo un análisis discriminante

cuadrático también llamado *QDA* por las siglas en inglés de *Quadratic Discriminant Analysis*). Este análisis permite obtener una función cuadrática que permita identificar si un elemento pertenece a una de las superclases o a la otra.

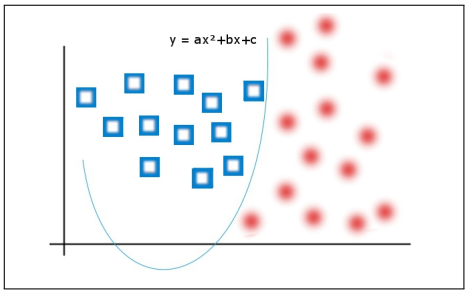
El *análisis discriminante lineal* (LDA - Lineal Discriminant Analysis) también permite obtener una ecuación para clasificar datos (ver figura 6.6(a)), sin embargo en algunos casos no es posible dividir un conjunto de elementos con una línea como se muestra en la figura 6.6(b). En este último caso lo ideal es clasificar los elementos con una curva. En la figura 6.6(c) podemos ver que efectivamente una línea curva dada por una ecuación cuadrática en ese tipo de casos es mejor para dividir o clasificar los elementos de dos distintos grupos.



(a) División con LDA



(b) Limitación de LDA



(c) División con QDA

Figura 6.6: Diferencia entre LDA y QDA

El cálculo del punto de división de un nodo, como podemos ver se da en dos pasos. Primero se debe asegurar con el algoritmo k-means que se tengan dos superclases. A continuación se debe realizar el análisis discriminante cuadrático para determinar la ecuación que permita identificar si un elemento pertenece a una super clase o a otra. Los pasos precisos para llevar a cabo lo anterior lo presentamos a continuación, (se puede consultar [16] para mayor detalle al respecto del método).

Algoritmo 6.3 *Algoritmo para búsqueda del punto de división*

Sea X la variable seleccionada para dividir el nodo t.

1. *Aplicar el algoritmo de agrupamiento 2-medias para dividir las J_t clases en dos super clases A y B, usando las dos medias más opuestas como centroides iniciales. Si las medias de las muestras son iguales, entonces la clase A será aquella clase que contenga más elementos y B el resto de las otras clases.*

2. *Sean $\overline{x}_A$ y ${s^2}_A$ que denotan la media y la varianza de la super clase A. De manera similar, sea $\overline{x}_B$ y ${s^2}_B$ que denota las correspondientes cantidades para B. Sean $p(A \mid t) = \sum_{j \in A} p(j \mid t)$ y $p(B \mid t) = 1 - p(A \mid t)$ que denotan las probabilidades de cada superclase.*

3. *Tomamos el logaritmo a ambos lados de la siguiente ecuación ecuación*

   $$p(A \mid t)\, {s_A}^{-1} \phi\{(x - \overline{x}_A)/s_A\} = p(B \mid t)\, {s_B}^{-1} \phi\{(x - \overline{x}_B/\, s_B\}$$

   para obtener la ecuación cuadrática $ax^2 + bx + c = 0$, donde:

   $a = {s^2}_A - {s^2}_B$

   $b = 2(\overline{x}_A {s^2}_B - \overline{x}_B {s^2}_A)$

   $c = (\overline{x}_B s_A)^2 - (\overline{x}_A s_B)^2 + 2{s^2}_A {s^2}_B log[\{p(A \mid t)\, s_B\} / \{p(B \mid t)\, s_A\}]$

   Si $a = 0$ y $\overline{x}_a \neq \overline{x}_B$ hay sólo una raíz dada por:

   $x = (\overline{x}_A + \overline{x}_B)/2 - (\overline{x}_A - \overline{x}_B)^{-1} {s^2}_A log\{p(A \mid t)/p(B \mid t)\}$

   La ecuación no tiene raíces si $a = 0$ y $\overline{x}_A = \overline{x}_B$

4. *El nodo se divide en $X = d$ donde d está definida como sigue:*

   a Si $a = 0$, entonces

   $$d = \begin{cases} (\overline{x}_A + \overline{x}_B)/2 - (\overline{x}_A - \overline{x}_B)^{-1} {s^2}_A log\{p(A \mid t)/p(B \mid t)\}, & \overline{x}_A \neq \overline{x}_B \\ \overline{x}_A, & \overline{x}_A = \overline{x}_B \end{cases}$$

   b En otro caso si $a \neq 0$, entonces:

   i Si $b^2 - 4ac < 0$, definimos $d = (\overline{x}_A + \overline{x}_B)/2$

   ii En otro caso si $b^2 - 4ac \geq 0$ entonces:

   A Definimos d como la raíz $(2a)^{-1}\left\{-b \pm \sqrt{b^2 - 4ac}\right\}$ más cercana a $\overline{x}_A$.

   B En otro caso, definimos $d = (\overline{x}_A + \overline{x}_B)/2$.

Una vez que se ha calculalo el valor del punto de división d, entonces un subárbol estará formado por aquellos elementos del conjunto de ejemplos, tales que sus valores correspondientes a la variable elegida X son menores o iguales a d. Aquellos elementos que su valor en X sean mayores exclusivamente que d, formarán el otro subárbol.

En el caso de los atributos categóricos el proceso es básicamente el mismo, aunque el conjunto de valores se procesa *a priori* de manera que se puedan analizar con las técnicas vistas en esta sección

Punto de división para variable categórica

El algoritmo k-means y el análisis discriminante cuadrático procesan datos de tipo numérico y no categórico. La razón es porque los categóricos no necesariamente están ordenados, es decir, no existe una relación de orden que nos permita identificar si un subconjunto de datos es menor, igual o mayor que un punto de división dado.

Si tenemos un conjunto de datos cuyos valores se encuentran en el conjunto $C = \{$México, EU, Canadá, Colombia, Francia, Ecuador$\}$, entonces, ¿cómo indicar si un dato es menor, mayor o igual que otro? Una acción inmediata pero muy ingenua sería asignar valores reales, de manera aleatoria, a cada uno de los valores categóricos. Así por ejemplo podríamos asignar: México$= 0.005$, EU$= 2.54$, etc. Sin embargo el llevar a cabo esta acción tendría como consecuencia que gran parte de la información que proporciona el conjunto de ejemplos se distorsione.

Por lo tanto si la variable a dividir es categórica, antes de calcular el punto de división, debemos transformar los valores categóricos a valores numéricos de manera que no se pierda información ni se tergiverse la esencia del problema que se quiere resolver.

El método que se utiliza tanto en QUEST como en CRUISE es el mismo. Primero se asigna a cada valor categórico un vector de ceros y unos, este tipo de vectores es llamado *dummy vectores* ó vectores ficticios. Por ejemplo si tenemos una variable categórica, llamada *color*, con los valores pertenecientes al conjunto $C = \{$rojo, verde, azul, amarillo, gris, negro, blanco, morado$\}$, le asignamos a cada uno de los elementos de C un vector ficticio diferente, donde cada vector tiene tantas entradas como elementos tiene el conjunto C. En la tabla 6.1 podemos ver los correspondientes vectores ficticios.

Valor categórico	Vector ficticio asociado
rojo	1,0,0,0,0,0,0,0
verde	0,1,0,0,0,0,0,0
azul	0,0,1,0,0,0,0,0
amarillo	0,0,0,1,0,0,0,0
gris	0,0,0,0,1,0,0,0
negro	0,0,0,0,0,1,0,0
blanco	0,0,0,0,0,0,1,0
morado	0,0,0,0,0,0,0,1

Tabla 6.1: Vectores ficticios variable categórica: *Color*

La razón de que casi todas las entradas sean cero, excepto aquella con valor uno, se basa en el hecho de que las variables categóricas tienen un comportamiento binario, es decir, un elemento tiene, uno y sólo uno, de los valores del conjunto de categorías.

Ya que los vectores ficticios por lo general tendrán más de una dimensión (más de una entrada), debemos reducir ésta, en otras palabras debemos proyectar cada vector a un número real. La técnica de reducción de dimensión utilizada, en QUEST y CRUISE, es la de coordenadas discriminantes o para más breve *crimcoords*. Esta técnica permite proyectar cada vector ficticio a una de sus coordenadas discriminantes, es decir a un valor real, de manera que éste es asignado a cada vector ficticio, y a su vez a su respectivo valor categórico.

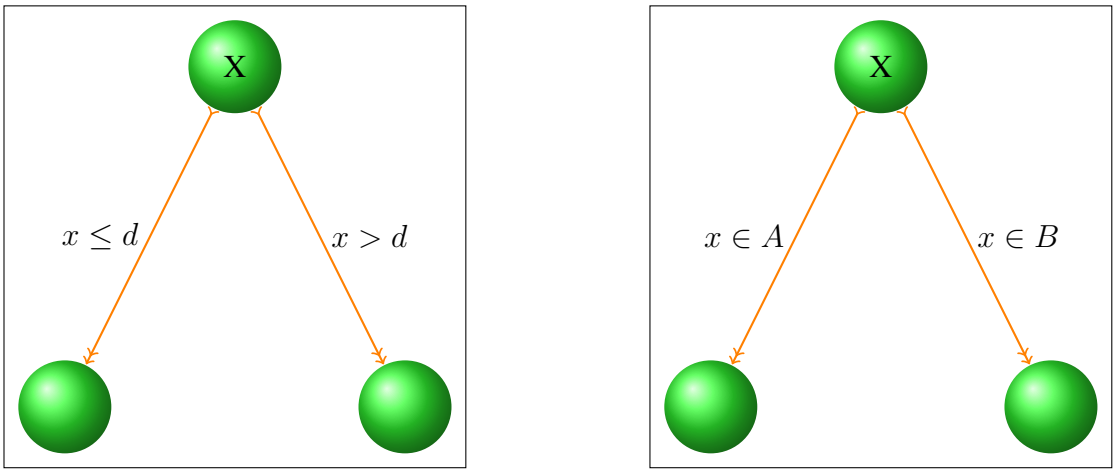
Cuando los datos categóricos son convertidos a numéricos, entonces ya se pueden analizar para encontrar el punto de división. En los siguientes pasos se resume el proceso de búsqueda del punto de división cuando el atributo es categórico.

- Conversión de datos categóricos a numéricos. El método crimcoords se describe en el apéndice D. Cabe añadir que muchas de las técnicas que se describen en la sección de apéndices, incluyendo crimcoords, no se exponen en las secciones donde se mencionan, ya que lo importante en dichas secciones es la descripción a un alto nivel del proceso de aprendizaje (ya sea crecimiento del árbol de decisión o creación de reglas de ILP).
- Creación de dos superclases con algoritmo k-means.
- Cálculo del punto de división d con QDA.

Una vez que se encontró el punto de división d, ya sea de un atributo numérico o categórico, entonces se reexpresa la partición.

Si el atributo dividido X es numérico, y d es el punto de división encontrado. Entonces un subárbol estará formado por aquellos elementos del conjunto de entrenamiento, cuyo valor correspondiente a X es menor o igual que d. El otro subárbol estará formado por aquellos elementos del conjunto de entrenamiento cuyo valor correspondiente a X es mayor a d. En la figura 6.7(a) podemos ver este tipo de partición.

Si el atributo dividido X es categórico, y d es el punto de división encontrado. Entonces un subárbol estará formado por aquellos elementos del conjunto de entrenamiento cuyo valor de X se encuentre en el conjunto A. El otro subárbol estará formado por aquellos elementos del conjunto de entrenamiento cuyo valor de X se encuentre en B. Y donde A está formado por los elementos cuyos valores categóricos al ser convertidos a numéricos son menores o iguales a d, y B estará formado por aquellos elementos cuyos valores categóricos al ser convertidos a numéricos son mayores que d. En la figura 6.7(b) se muestra este tipo de de división.



(a) División binaria con datos numéricos

(b) División binaria con datos categóricos

Figura 6.7: División binaria en árboles de decisión

La operación de especificación, en el contexto de los árboles de decisión, básicamente es elegir un atributo y dividirlo para crear dos subárboles. Esta operación se repite por cada nodo

del árbol de decisión, sin embargo en algún momento debe detenerse este proceso de especificación. En la sección siguiente veremos las condiciones que se deben tomar en cuenta para detener el proceso.

6.3.3 Criterio de paro

QUEST detiene el proceso de crecimiento del árbol de decisión en las siguientes situaciones.

- **Si un nodo es puro.** Se dice que un nodo es puro cuando todos los elementos del conjunto de ejemplos que restan por clasificar pertenecen a una misma clase. Es indiscutible que si todos los elementos de un conjunto de entrenamiento forman parte de una misma clase, entonces no hay necesidad de clasificar, por lo que en este caso el nodo se convierte en hoja, y se le asigna la clase representada por esos elementos. En algunos casos se puede especificar un porcentaje de pureza, es decir, que basta con que un determinado porcentaje de elementos estén relacionados con una clase para asignársdela al nodo y convertirlo en hoja.

- **Si todos los datos son iguales.** Cuando esto sucede, se asigna aquella clase que esté representada por más ejemplos del conjunto restante.

- **Si el árbol alcanza cierta profundidad.** Se puede especificar un nivel máximo que puede alcanzar el árbol de decisión. Si durante el entrenamiento el árbol ha alcanzado su profundidad máxima entonces el nodo no será dividido, y se asignará la clase que sea representada por más elementos.

- **Si no alcanza el tamaño mínimo del nodo.** También puede especificarse el número mínimo de elementos que puede tener cada nodo, es decir, si el conjunto de elementos restantes es menor que el valor especificado entonces no se divide el nodo y se asigna la clase con más elementos.

- **Si uno de los nodos hijos tiene pocos elementos.** Si al dividir el nodo uno de los subárboles tiene un número de elementos menor que el especificado, entonces no se divide el nodo y se asigna la clase mayoritaria.

6.3.4 Valores desconocidos

Cuando se crean bases de datos en la vida real es común que muchos de estos elementos de información se desconozcan, ya sea por que en algunos casos no fue posible realizar la captura u obtención del dato, o porque se haya cometido errores al recolectar la información. El manejo de datos desconocidos no es trivial y muchos autores han sugerido diferentes procedimientos para tratar con estos. En QUEST el manejo de valores desconocidos se realiza como sigue:

Si la variable dependiente de un caso es desconocida, entonces el caso será ignorado en el análisis, en este caso la variable dependiente es la clase. Si todas las variables predictoras de un caso son desconocidas, entonces el caso será ignorado.

De esta manera podemos apreciar que QUEST utiliza solamente aquellos ejemplos que no tienen valores desconocidos, esta estrategia es conocida como la *estrategia de los casos disponibles*.

En este trabajo no tomamos mucha importancia al manejo de los valores desconocidos, ya que analizaremos ejemplos de ILP sin valores desconocidos.

6.3.5 Poda

Hemos mencionado que en el caso de los árboles de decisión, la operación de especificación corresponde al crecimiento del árbol y la de generalización a la poda. En este contexto el criterio de paro es una especie de prepoda ya que, idealmente, detiene el crecimiento del árbol antes de que surja el sobreentrenamiento. Desgraciadamente muchas veces los criterios de paro no son suficientemente buenos para evitar el sobreentrenamiento, y el árbol termina su crecimiento después de manifestarse este sobreajuste. Como consecuencia los árboles obtenidos terminan con subárboles y hojas de más. La solución más obvia es eliminar (podar) aquellos subárboles u hojas que están de más.

Existen varias técnicas de poda para eliminar el sobreajuste, y tanto CRUISE como QUEST, implementan una de éstas llamada *validación cruzada*, sin embargo ya que nuestro objetivo no es crecer los árboles de decisión, sino encontrar predicados que aporten información nueva sobre los problemas de ILP, no fue necesario implementarlo. No obstante para que la información sobre estos algoritmos sea completa, presentamos a continuación una breve descripción de la técnica de poda llevada a cabo por éstos algoritmos.

La validación cruzada consiste en dividir al conjunto de ejemplos en k subconjuntos, aproximadamente del mismo tamaño ó número de elementos, llamados *folds*. En una primera instancia se toman $k-1$ conjuntos para entrenamiento, y el subconjunto restante se toma como el de prueba. Con esta división inicial se realiza el entrenamiento, se obtiene la regla correspondiente (en nuestro caso el árbol decisión) y se calcula, con el conjunto de prueba, la tasa de error. En un segundo paso se toma otro subconjunto de prueba, que no se haya usado en la ejecución anterior, y los restantes $k-1$ como conjunto de entrenamiento. De esta manera se obtiene un segundo árbol de decisión y se obtiene su tasa de error igual que en el anterior paso. Este proceso se lleva a cabo k veces.

Una vez que se han obtenido los k árboles de decisión y calculado sus respectivas tasas de error, se realiza un promedio total de las tazas de los error, de manera que se pueda determinar qué árboles tienen subárboles u hojas de más y se puedan borrar. Al final se obtendrá un árbol de decisión que de alguna manera es el promedio de los k obtenidos.

En la figura 6.8 representamos la validación cruzada con $k = 3$. El conjunto total se divide en tres partes del mismo tamaño. En la primera ejecución del algoritmo de entrenamiento se toma un subconjunto para prueba (el conjunto con color diferente de los otros dos) y los otros dos se utilizan para entrenamiento (los dos conjuntos con igual color). Al llevar a cabo el entrenamiento se obtiene un modelo de aprendizaje, ya sea un árbol de decisión o un programa lógico en ILP, que se utiliza para analizar el conjunto de prueba y obtener así una tasa de error. Ésto se realiza tres veces para obtener el promedio total y un modelo final.

Para algoritmos de aprendizaje maquinal lo usual es dividir el conjunto de entrenamiento en 10 subconjuntos (10-folds). Aunque en el software disponible de QUEST y CRUISE el usuario puede especificar el número de folds con el que desea realizar la validación cruzada.

Hasta aquí hemos cubierto los puntos más importantes en la inducción de un árbol de decisión concernientes a QUEST. En la siguiente sección veremos las principales características de CRUISE.

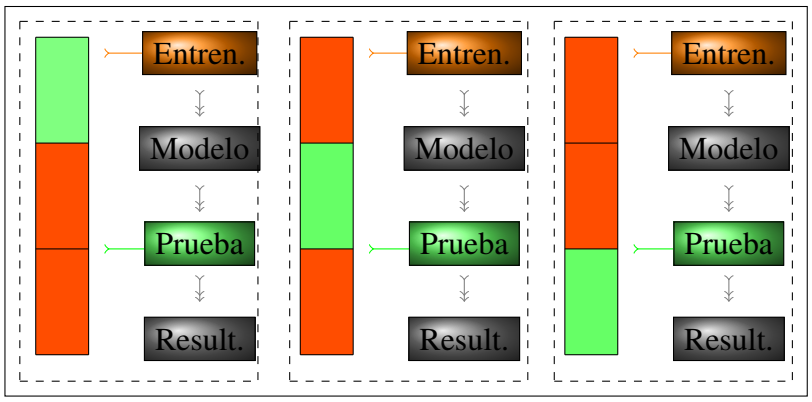


Figura 6.8: Validación cruzada para tres folds

6.4 Algoritmo CRUISE

Para la elección del mejor atributo, CRUISE utiliza un algoritmo distinto a QUEST llamado *2D*. En el caso del cálculo del punto de división es básicamente el mismo procedimiento que QUEST, pero a diferencia de éste se realiza previamente una transformación a los datos. Para no ser redundantes, presentaremos en este capítulo solamente aquellos métodos y algoritmos que difieran de los implementados en QUEST.

6.4.1 Selección de variable

Para seleccionar el mejor atributo para cada nodo, CRUISE al igual que QUEST, realiza una prueba de independencia con cada par de variables categóricas, pero además realiza esta prueba también con cada par de variables numéricas, y cada par de variables donde una es categórica y la otra numérica. Esto permite que el método sea sensible a las interacciones entre cada par de variables.

Para evitar el prejuicio de selección[3] hacia los atributos categóricos, algo que se dá cuando estos atributos tienen demasiadas categorías, se realiza una corrección *bootstrap*[4]. Si el prejuicio de selección es muy bajo y además se lleva a cabo la corrección bootstrap, es posible *sobrecorregirlo*. Para evitar esto, se incrementa deliberadamente el prejuicio de selección antes de llevar a cabo la corrección bootstrap, seleccionando la variable categórica si el p-valor más significativo proviene del análisis de interacción llevado a cabo entre atributos diferentes (categórico-numérico).

En los dos siguientes algoritmos se presenta el método de elección de atributo, aunque también pueden consultarse en [16].

[3]El prejuicio o sesgo de selección, es el error debido a diferencias sistemáticas entre los elementos seleccionados para ser analizados en un estudio estadístico, y aquellos elementos que no fueron seleccionados. En la selección de variables para cada nodo, este sesgo provoca que las variables de un tipo, categórico o numérico, tengan más probabilidades de ser elegidas.

[4]Este método de corrección, reduce el prejuicio de selección existente debido a las diferencias entre variables categóricas y numéricas. Además es muy importante ya que el sesgo depende de muchos aspectos de los datos, tales como el tamaño de la muestra, el número y tipo de variables, los valores desconocidos, etc. [11]

Algoritmo 6.4 *Algoritmo 2D*

Supongamos que $X_1, \ldots, X_{K_1}$ y $X_{K_1+1}, \ldots, X_k$ son variables numéricas y categóricas respectivamente. Sea J_t el número de clases representadas en el nodo t.

1. *Se realiza un análisis marginal para cada variable numérica X:*

   (a) *Dividir los datos en cuatro grupos utilizando los cuartiles de la muestra de X.*

   (b) *Construir una tabla de contingencia de $J_t X4$ con las clases como renglones y los grupos como columnas.*

   (c) *Calcular el estadístico Pearson χ^2 con $\nu = 3\,(J_t - 1)$ grados de libertad.*

   (d) *Convertir χ^2 a un valor aproximadamente normal estándar con la transformación Peizer-Pratt*

$$z = \begin{cases} \mid W \mid^{-1} (W - 1/3) \sqrt{(\nu - 1)\, log\, \{(\nu - 1)/\chi^2\} + W}, & \nu > 1 \\ \sqrt{\chi^2}, & \nu = 1 \end{cases} \tag{6.1}$$

   donde $W = \chi^2 - \nu + 1$.

   Denotamos como z_n al z-valor más grande de los K_1 calculados.

2. *Se realiza un análisis marginal para cada variable categórica X: Sea C que denota el número de categorías de X.*

   (a) *Construir una tabla de contingencia de $J_t XC$ con las clases como filas y las C categorías como columnas.*

   (b) *Calcular el estadístico Pearson χ^2 con $(J_t - 1)\,(C - 1)$ grados de libertad.*

   (c) *Usar la transformación Peizer-Pratt de la ecuación 6.1 para convertirlos en z-valores.*

   Sea z_c que denota al z-valor más grande de los $(K - K_1)$ calculados.

3. *Realizar la prueba de interacción por cada par de variables numéricas $(X_k, X_{k'})$:*

   (a) *Dividir cada espacio $(X_k, X_{k'})$ en cuatro cuadrantes en las medias de las muestras.*

   (b) *Construir una tabla de contingencia de $J_t X4$ con las clases como filas y los cuadrantes como columnas.*

   (c) *Calcular el estadístico Pearson χ^2 con $3\,(J_t - 1)$ grados de libertad.*

   (d) *Usar la transformación Peizer-Pratt de la ecuación 6.1 para convertirlos en z-valores.*

   Sea z_{nn} el z-valor más grande de los $K_1\,(K_1 - 1)/2$ calculados.

4. *Se realiza la prueba de interacción por cada par de variables categóricas: Usar pares de categorías para formar los grupos en la tabla. Si el par de variables C_1 y C_2 valores categóricos, entonces se obtiene una tabla de $J_t X C_1 C_2$. Sea Z_{cc} que denota el z-valor más grande de los $(K - K_1)\,(K - K_1 - 1)/2$ calculados.*

5. *Se realiza una prueba de interacción por cada par de variables $(X_k, X_{k'})$ donde X_k es numérica y $X_{k'}$ es categórica. Si $X_{k'}$ toma C valores, crear una tabla de $J_t X 2C$. Sea z_{nc} que denota el z-valor más grande de los K_1 $(K - K_1)$ calculados.*

Sea f^* el valor bootstrap obtenido en el algoritmo 6.5 y definido como:

$$z^* = max\left\{f^* z_n, z_c, f^* z_{nn}, z_{cc}, z_{nc}\right\}$$

1. *Si $f^* z_n = z^*$, seleccionar la variable numérica con el z-valor más grande.*

2. *Si $z_c = z^*$, seleccionar la variable categórica con el z-valor más grande.*

3. *Si $f^* z_{nn} = z^*$, seleccionar la variable numérica del par con el z-valor más grande.*

4. *Si $z_{cc} = z^*$, seleccionar la variable categórica del par con el z-valor más grande.*

5. *Si $z_{nc} = z^*$, seleccionar la variable categórica del par.*

Algoritmo 6.5 *Corrección de prejuicio bootstrap*

1. *Crear una muestra de aprendizaje bootstrap. Esto se llevó a cabo realizando un remuestreo aleatorio con reemplazo*[5] *a partir de los valores originales del conjunto de entrenamiento. Esto permite crear un nuevo conjunto de entrenamiento.*

2. *Aplicar los pasos $1 - 5$ del algoritmo 6.4 al conjunto de entrenamiento creado en el paso 1 y obtener los correspondientes cinco z-valores.*

3. *Dado $f > 1$, seleccionar una variable numérica si $f max\left\{z_n, z_{nn}\right\} \geq max\left\{z_c, z_{cc}, z_{nc}\right\}$. En otro caso seleccionar una variable categórica.*

4. *Repetir los pasos 1-3 muchas veces con varios valores de f. Sea $\pi\left(f\right)$ la proporción de veces que se seleccionó una variable numérica.*

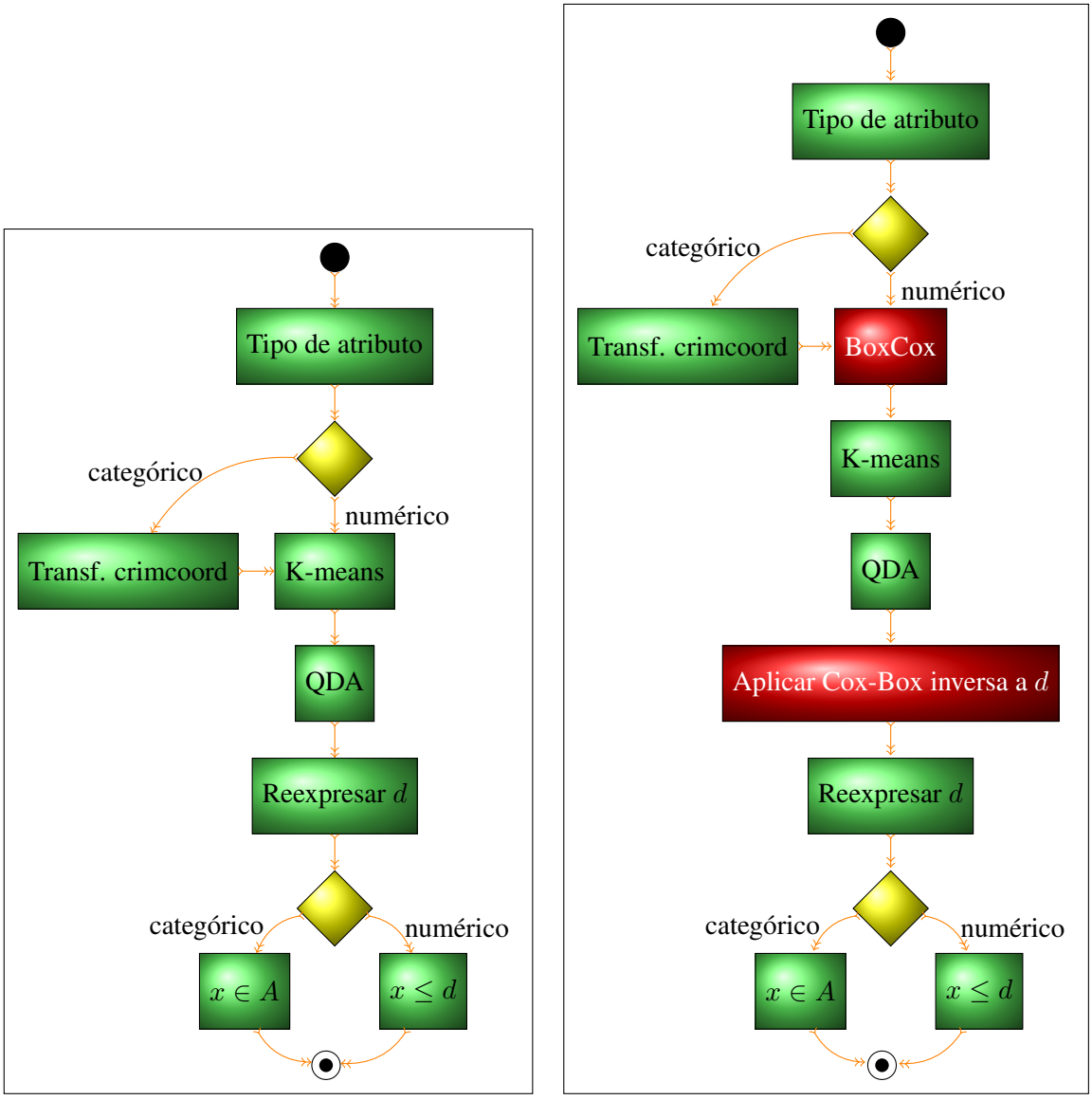
5. *De ser necesario interpolar linealmente para encontrar f^* de manera que $\pi\left(f^*\right)$ sea igual a la proporción de variables numéricas en los datos.*

De las partes más importantes de un algoritmo de inducción de árboles de decisión, la elección del atributo es la que cambia totalmente entre CRUISE y QUEST. Como veremos más adelante las demás sólo difieren un poco de un algoritmo a otro.

[5]El remuestreo aleatorio con reeemplazo, es aquel donde cada elemento de un conjunto N_1 puede ser seleccionado, más de una vez, para formar un nuevo conjunto N_2 con el mismo número de elementos que N_1. En el algoritmo de corrección de prejuicio bootstrap, los nuevos conjuntos de entrenamiento formados son utilizados para corregir el sesgo o prejuicio de selección. En la referencia [7] se detalla ampliamente este método creado por Bradley Efron.

6.4.2 Punto de división

La diferencia esencial entre el método de QUEST y el de CRUISE, para encontrar el punto de división, es la *transformación Box-Cox*. El análisis discriminante cuadrático funciona mejor cuando los datos sobre los que trabaja tienen una distribución normal, por lo cual, la transformación Box-Cox se utiliza precisamente para normalizar los datos antes de poder clasificarlos con QDA. En la figura 6.9 resumimos ambos métodos de manera que la diferencia queda más clara que si los presentamos como una lista de pasos.



(a) QUEST (b) CRUISE

Figura 6.9: Diferencia QUEST-CRUISE: punto de división

A continuación detallamos la transformación Box-Cox, que como mencionamos anteriormente normaliza un conjunto de datos.

Algoritmo 6.6 *Transformación Box-Cox*

Supongamos que X es la variable seleccionada. Si X es catégorica, entonces sus valores deben ser transformados, a valores numéricos, antes de procesar el conjunto de datos con este algoritmo.

1. *Si existen valores negativos, entonces a cada elemento del conjunto de valores de la variable correspondiente, se le suma el valor absoluto más grande de los elementos negativos. Con esto aseguramos que no haya valores negativos. Si imaginamos una gráfica en un plano cartesiano, y parte de ésta es negativa en el eje de las ordenadas, entonces, este paso equivale a "subir" la gráfica hasta que no haya un sólo punto por debajo de 0 en el eje y.*

2. *Dada λ Definimos*

$$x^{(\lambda)} = \begin{cases} \left[(x-\theta)^{\lambda} - 1\right]/\lambda, & Si \;\; \lambda \neq 0 \\ log\,(x-\theta), & Si \;\; \lambda = 0 \end{cases} \tag{6.2}$$

3. *Sea $\hat{\lambda}$ el valor que minimiza la siguiente expresión.*

$$\sum_j \sum_i \left[{x_{ji}}^{(\lambda)} - \bar{x}_j^{(\lambda)}\right]^2 exp\left\{-2n^{-1}\lambda\left[\sum_j \sum_i log\,(x_{ji})\right]\right\} \tag{6.3}$$

   donde X_{ji} es el i-ésimo valor de X en la clase j y $\bar{x}_j^{(\lambda)}$ es la media de la clase de la muestra de sus valores transformados.

4. *Transformar cada valor x a $x^{(\lambda)}$.*

El algoritmo 6.6 se implementó de la siguiente manera. El primer paso es asegurarnos que no haya elementos negativos en el conjunto de valores correspondientes a la variable elegida X. Para ello primero verificamos si existen valores negativos, de ser así entonces se obtiene el valor n negativo con el valor absoluto más grande. Después se suma $n+1$ a cada uno de los elementos de X. Con esto aseguramos que todos los elementos sean positivos.

El siguiente paso es encontrar el valor óptimo de λ, en este caso queremos que minimice la ecuación 6.3. Ya que esto es un problema de optimización, decidimos utilizar un algoritmo evolutivo, éste es detallado a continuación. En [10] y [13] podemos encontrar más información de éste y otros algoritmos evolutivos.

Algoritmo 6.7 *Algoritmo $\mu + \lambda$*

```
begin
  Crear una población inicial de μ elementos (llamados padres)
  while (Condición de terminación no se cumpla) do
        Crear una población de λ elementos (llamados hijos)
```

```
        Evaluar padres e hijos: μ + λ
        Elegir los μ mejores elementos de μ + λ (son los nuevos padres)
  od
  Elegir el mejor elemento
end
```

Cuando tenemos el valor óptimo de λ, entonces éste es usado para transformar cada uno de los valores del conjunto de X de acuerdo con la expresion de la ecuación 6.2. Cabe añadir que en dicha expresión el valor $x - \theta$ corresponde al valor de $x \in X$ tal que es positivo, es decir, que la transformación se aplica a los valores exclusivamente positivos y no a los originales.

Al tener todos los datos transformados se procede a calcular el punto de división. Debemos aclarar que el punto de división d calculado, corresponde al punto que divide al conjunto transformado con Cox-Box, por lo que después de haberlo encontrado se aplicó la transformación inversa de la ecuación 6.2 a este punto de división, para encontrar el que corresponde con el conjunto original de X. Al terminar esto se procede a expresar el punto de división como en el caso del algoritmo QUEST.

Tanto el criterio de paro, como la poda y el manejo de los valores desconocidos se tratan igual que en el algoritmo QUEST. Además estos no se implementaron, ya que como mencionamos anteriormente el interés principal lo focalizamos sobre el cálculo del punto de división, por lo tanto consideramos innecesario volver a presentar dichos métodos. En el siguiente capítulo presentamos los análisis realizados sobre problemas ILP, y posteriormente las conclusiones sobre los resultados obtenidos.

Capítulo 7

Experimentos y resultados

En este capítulo presentamos los experimentos llevados a cabo, así como los resultados alcanzados al comparar las hipótesis obtenidas con nuestra propuesta, y las obtenidas con ILP sin aprendizaje multivalores.

7.1 Análisis de bases de datos

Recordemos que nuestro objetivo es crear cláusulas multivalores para obtener hipótesis con menos reglas. Para determinar si se cumplió el objetivo, se analizaron las siguientes bases de datos:

1. **Taxonomía de animales.** Este problema consiste en encontrar una hipótesis que explique las características que debe tener un animal para pertenecer a una de las siguientes clases: ave, reptil, pez ó mamífero.

2. **Problema Bongard.** Problema típico en el reconocimiento de patrones donde el objetivo es encontrar las características que definen a cada elemento del conjunto de ejemplos positivos.

3. **Figuras geométricas.** Para este ejemplo se debe encontrar una hipótesis que indique el número de lados que debe tener una figura geométrica para ser cuadrilátero.

4. **Student Loan Relational Data Set.** Base de datos obtenida del repositorio de aprendizaje maquinal [8], consiste en encontrar una hipótesis que explique qué personas no requieren pagar un préstamo de estudiante.

5. **Japanese Credit Screening Data Set.** Esta base de datos también se obtuvo del repositorio citado en la referencia [8] y el problema es encontrar una hipótesis que indique si debe otorgarse un crédito a una persona.

Cada base de datos se analizó primero con un algoritmo sin aprendizaje multivalores: Progol; y después con dos algoritmos ILP que fueron adaptados con nuestro método para agregar cláusulas multivalores: FOIL y Progol. A continuación listamos estos algoritmos:

- **Aleph.** Este es un sistema ILP creado por Ashwin Srinivasan [33] que implementa el algoritmo *Progol* desarrollado por Muggleton [20]. Ya que este algoritmo no fue implementado no lo presentamos en el capítulo 6, sin embargo es necesario describirlo, por lo que lo exponemos en el apéndice E.

- **FOIL multivalores.** Adaptación de FOIL al método que hemos propuesto.

- **Aleph multivalores.** Adaptación de Progol, implementado por Aleph, al método que hemos propuesto.

Debemos aclarar que no se analizaron las bases de datos con FOIL sin aprendizaje multivalores, ya que las hipótesis creadas por este algoritmo no permiten representar una solución adecuada para cada problema analizado en este trabajo.

Por otro lado cada base de datos fue analizada utilizando validación cruzada [14]; debido a que los conjuntos de entrenamiento no son muy grandes las bases de datos 1, 2 y 3 fueron divididas en 3 subconjuntos (*folds*), y la 4 y 5 fueron divididas en 10 subconjuntos.

Para cada problema se analizó la siguiente información:

- Número de reglas que conforman la hipótesis obtenida.

- Precisión: porcentaje de ejemplos de prueba cubiertos.

- Tiempo. En este caso sólo comparamos los resultados obtenidos con Aleph y Aleph multivalores. No calculamos el tiempo para FOIL y FOIL multivalores ya que los problemas analizados no pueden ser resueltos con FOIL, por lo tanto no pueden ser comparados. Todos los ejemplos fueron ejecutados en una máquina con dos procesadores *Pentium 4 de doble núcleo a 3.00GHz de velocidad*.

A continuación presentamos los resultados obtenidos para cada base de datos.

7.1.1 Taxonomía de animales

El primer ejemplo analizado fue el de taxonomía de animales planteado por Muggleton [21], el cual consiste en clasificar un conjunto de animales en cuatro posibles clases. El átomo objetivo es:

$$clase\,(Animal, Clase) \leftarrow$$

Donde: $Clase \in \{mamifero, pez, ave, reptil\}$ y
$Animal \in \{perro, delfin, ornitorrinco, murcielago, trucha, arenque, tiburon, anguila, lagarto, cocodrilo, t\_rex, serpiente, tortuga, aguila, avestruz, pinguino, gato\}$

El objetivo es encontrar un programa lógico que indique las características que debe tener un animal para pertenecer a cada una de las clases mencionadas anteriormente.

Para la clasificación, el conocimiento previo contiene información acerca de varios atributos que caracterizan a cada animal: número de patas, tipo de cubierta, tipo de habitat, etc. A

continuación presentamos los predicados que forman parte del conocimiento previo y que proporcionan la información necesaria para encontrar el programa lógico que determine cada clase.

- $cubierta\,(Animal, TipoCubierta)$. Donde $TipoCubierta \in \{plumas, pelo, escamas, ninguna\}$. Esta literal declara cada tipo de cubierta que puede tener un animal, así por ejemplo una trucha que tiene escamas la denotamos: $cubierta\,(trucha, escamas)$.
- $habitat\,(Animal, TipoHabitat)$. Donde $TipoHabitat \in \{aire, cueva, agua, tierra\}$. Esta literal indica el tipo de habitat para cada animal, por ejemplo para indicar que una trucha vive en agua declaramos: $habitat\,(trucha, agua)$.
- $homeotermico\,(Animal)$. Indica aquellos animales que regulan por sí mismos su temperatura sin depender del medio ambiente. En este caso solo se declaran aquellos animales que son homeotérmicos.
- $mama\_leche\,(Animal)$. Con esta literal presentamos aquellos animales que maman leche, esta literal es esencial para determinar la clase de los mamíferos.
- $pone\_huevos\,(Animal)$. Presenta aquellos animales que producen huevos, y al igual que las dos literales anteriores solo declaramos los animales que cumplen con esta característica.
- $tiene\_branquias\,(Animal)$. Los animales que poseen branquias son declarados con esta literal, exluyendo aquellos que no las tienen.
- $tiene\_patas\,(Animal, NumeroPatas)$. Donde $NumeroPatas \in \{0, 2, 4\}$. Con esta literal se determina el número de patas de cada animal.

Como podemos ver en la lista de literales anterior los atributos *TipoHabitat, TipoCubierta y NumeroPatas* son categóricos, por lo que se realizó aprendizaje multivalores con estas variables para crear nuevas cláusulas con ellas.

El conjunto de ejemplos positivos consta de 17 animales cuyas características se encuentran contenidas en el conocimiento previo.

A continuación presentamos los resultados obtenidos.

Resultados

En la tabla 7.1, mostramos que tanto el porcentaje de ejemplos cubiertos como el número de reglas mejoró, aunque debemos mencionar que el número de reglas no es muy diferente si comparamos las hipótesis multivalores con las no multivalores. Por otro lado el promedio de precisión aumentó notablemente: aproximadamente 14%.

En cuanto al tiempo de ejecución tampoco vemos mucha diferencia, de hecho con *Aleph multivalores* aumentó el tiempo 0.004 segundos en promedio. Lo ideal en cuanto al tiempo, es que disminuya o en el peor de los casos que el aumento no afecte demasiado. Aunque debemos tomar en cuenta que nuestro método crece el espacio de búsqueda por lo que se espera que el tiempo de ejecución también siga dicho comportamiento.

Para apreciar mejor la diferencia entre los dos tipos de aprendizaje, veamos las hipótesis obtenidas con el primer *fold*. Sin aprendizaje multivalores la hipótesis es la siguiente:

- $clase\,(A, mamifero) \leftarrow mama\_leche\,(A)\,.$

	Fold 1	Fold 2	Fold 3	Promedio
Número de Reglas	===	===	===	===
Aleph	5 reglas	4 reglas	4 reglas	4.33 reglas
Aleph multivalores	4 reglas	4 reglas	4 reglas	4 reglas
FOIL multivalores	4 reglas	4 reglas	4 reglas	4 reglas
Porcentaje de ejemplos cubiertos	===	===	===	===
Aleph	100%	83.33%	75%	86.11%
Aleph multivalores	100%	100%	100%	100%
FOIL multivalores	100%	100%	100%	100%
Tiempo de ejecución	===	===	===	===
Aleph	0.012 seg.	0.012 seg.	0.012 seg.	0.012 seg.
Aleph multivalores	0.02 seg.	0.012 seg.	0.016 seg.	0.016 seg.

Tabla 7.1: Resultados obtenidos en el problema de clasificación de animales.

- $clase\,(A, pez) \leftarrow tiene\_branquias\,(A)\,.$
- $clase\,(A, reptil) \leftarrow cubierta\,(A, escamas)\,, tiene\_patas\,(A, 4)\,.$
- $clase\,(serpiente, reptil)\,.$
- $clase\,(A, ave) \leftarrow cubierta\,(A, plumas)\,.$

La hipótesis multivalores, para el primer *fold*, obtenida con FOIL multivalores es:

- $clase\,(A, mamifero) \leftarrow mama\_leche\,(A)\,.$
- $clase\,(A, reptil) \leftarrow cubierta\,(A, B)\,, member\,(B, [escamas])\,, not\,(tiene\_branquias\,(A))$.
- $clase\,(A, pez) \leftarrow tiene\_branquias\,(A)\,.$
- $clase\,(A, ave) \leftarrow homeotermico\,(A)\,, not\,(mama\_leche\,(A))\,.$

La cláusula clave en las dos hipótesis anteriores es aquella que define la clase reptil, sin embargo aún con aprendizaje multivalores virtualmente no hay diferencia, ya que ambas indican lo mismo: La cubierta de los reptiles es escamas.

La diferencia viene implícita en la cláusula multivalores, ya que nuestro método divide los tipos de cubierta en dos sunconjuntos creando las dos cláusulas multivalores siguientes:

- $clase\,(A, reptil) \leftarrow cubierta\,(A, B)\,, member\,(B, [escamas])\,, \ldots.$
- $clase\,(A, reptil) \leftarrow cubierta\,(A, B)\,, member\,(B, [pelo, plumas, ninguna])\,, \ldots.$

Sin aprendizaje multivalores, los algoritmos ILP probarían cuatro cláusulas, una por cada cubierta.

Además de la cubierta, como habíamos mencionado, el análisis multivalor también se realiza para los habitats y el número de patas que tiene cada animal, ya que son variables categóricas.

7.1.2 Problema Bongard

Los problemas *Bongard* [2] son ejemplos clásicos dentro del área de reconocimiento de patrones. Estos problemas consisten en dos grupos de figuras, de los cuales el grupo de elementos positivos comparten una o más características en común (patrones), de manera que los algoritmos de clasificación deben encontrar esos patrones.

El problema que abordamos en nuestro trabajo contiene grupos de figuras como las que se muestran en la figura 7.1. Cada ejemplo, negativo o positivo, puede contener alguna de las siguientes figuras: círculo, cuadrado o triángulo. Cada triángulo puede apuntar en alguna de las siguientes direcciones: oeste(o), noroeste(no), norte(n), noreste(ne), este(e), sureste(se), sur(s) o suroeste(so).

El objetivo del problema es encontrar un programa lógico que indique los patrones que sigue cada ejemplo positivo. Para ello el átomo objetivo es:

$$bongard\,(Ejemplo) \leftarrow$$

Donde: $Ejemplo$ representa algún conjunto de figuras.

El proceso de aprendizaje se realiza utilizando las siguientes literales contenidas en el conocimiento previo.

- $circulo\,(Ejemplo, Elemento)$. En este caso *Elemento* representa alguno de los 10 círculos existentes en el conocimiento previo.
- $cuadrado\,(Ejemplo, Elemento)$. *Elemento* representa uno de los 8 cuadrados existentes en el conocimiento previo.
- $triangulo\,(Ejemplo, Elemento)$. *Elemento* representa uno de los 47 triángulos existentes en el conocimiento previo.
- $direccion\,(Elemento, Direccion)$. *Dirección* puede ser alguna de las direcciones mencionadas anteriormente, como podemos ver la variable *Direccion* es un atributo categórico, por lo tanto lo utilizamos para analizarlo con aprendizaje multivalores.

La base de datos consta de 18 ejemplos positivos y 14 negativos, de estos mostramos algunos en la figura 7.1, y como podemos observar los patrones de cada ejemplo positivo son:

- Cada ejemplo positivo tiene al menos un triángulo.
- Dicho triángulo apunta a una de las siguientes direcciones: oeste(o), noroeste(no), norte(n) o noreste(ne). Las otras direcciones son: este(e), sureste(se), sur(s) y suroeste(so).

Tomando en cuenta el tipo de figura y la dirección de los triángulos contenidos en cada ejemplo, los resultados para este problema se presentan a continuación:

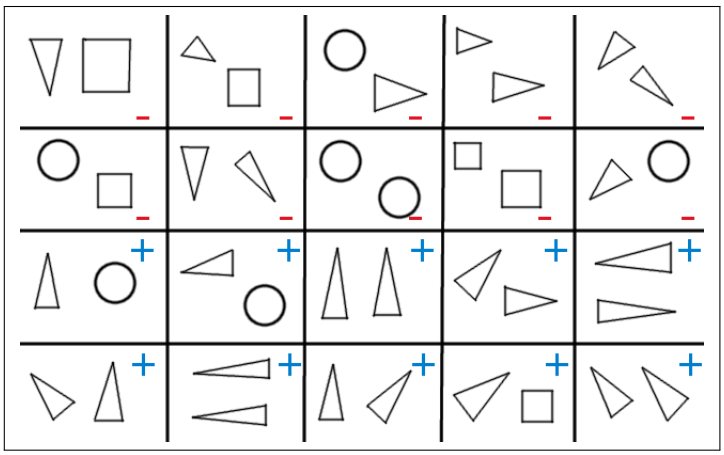


Figura 7.1: Problema Bongard

Resultados

Los resultados mostrados en la tabla 7.2 muestran que el número de reglas mejoró notablemente, ya que con los dos métodos multivalores se obtuvo una regla: en promedio 2.66 reglas menos que con la técnica no multivalor. Por otro lado el porcentaje de ejemplos cubiertos mejoró 11.11% con los métodos multivalores. Al igual que en el ejemplo de clasificación de animales, el tiempo de ejecución no varió significativamente.

	Fold 1	Fold 2	Fold 3	Promedio
Número de Reglas	===	===	===	===
Aleph	3 reglas	4 reglas	4 reglas	3.66 reglas
Aleph multivalores	1 reglas	1 reglas	1 reglas	1 reglas
FOIL multivalores	1 reglas	1 reglas	1 reglas	1 reglas
Porcentaje de ejemplos cubiertos	===	===	===	===
Aleph	66.66%	100%	100%	88.88%
Aleph multivalores	100%	100%	100%	100%
FOIL multivalores	100%	100%	100%	100%
Tiempo de ejecución	===	===	===	===
Aleph	0.007 seg.	0.0079 seg.	0.012 seg.	0.00896 seg.
Aleph multivalores	0.004 seg.	0.004 seg.	0.004 seg.	0.004 seg.

Tabla 7.2: Resultados obtenidos en el problema Bongard

Veamos la diferencia en las hipótesis obtenidas. Las reglas creadas por Aleph son:

- $bongard\,(A) \leftarrow triangulo\,(A, B)\,, direccion\,(B, n)\,.$
- $bongard\,(A) \leftarrow triangulo\,(A, B)\,, direccion\,(B, o)\,.$
- $bongard\,(A) \leftarrow triangulo\,(A, B)\,, direccion\,(B, no)\,.$
- $bongard\,(A) \leftarrow triangulo\,(A, B)\,, direccion\,(B, ne)\,.$

Con aprendizaje multivalor obtenemos solamente una regla:

- $bongard\,(A) \leftarrow triangulo\,(A, B)\,, direccion\,(B, C)\,, member\,(C, [n, o, no, ne])\,.$

Para este ejemplo el uso del aprendizaje multivalores se hace más evidente debido al decremento en el número de cláusulas que forman la hipótesis. Con las reglas multivalores representamos todas las direcciones (patrón que forma parte de los ejemplos positivos) que sigue al menos uno de los triangulos de cada ejemplo positivo.

Podemos notar también que si otras direcciones formaran parte del patrón que siguen los ejemplos positivos, también podrían representarse en una sola cláusula. Esto es debido a que se hace una partición de todos los ángulos hacia los cuales apuntan los triángulos, y aquellos que caracterizan a los triángulos de cada elemento positivo forman parte de la cláusula multivalores. En contraste la hipótesis que obtendríamos con Aleph contendría cada vez más cláusulas. A continuación presentamos las cláusulas multivalores creadas por nuestro método y probadas por los algoritmos ILP.

- $bongard\,(A) \leftarrow triangulo\,(A, B)\,, direccion\,(B, C)\,, member\,(C, [n, o, no, ne])\,.$
- $bongard\,(A) \leftarrow triangulo\,(A, B)\,, direccion\,(B, C)\,, member\,(C, [e, se, s, so])\,.$

La partición realizada por nuestro método, permite probar solo dos cláusulas multivalores, por otro lado sin este tipo de aprendizaje, los algoritmos ILP tendrían que probar ocho cláusulas. En el siguiente ejemplo este comportamiento también será evidente.

7.1.3 Figuras geométricas

Otro ejemplo que utilizamos para probar nuestra propuesta es la creación de una hipótesis que indique la característica que tiene una figura geométrica para ser un cuadrilátero o cualquier otra figura. El programa obtenido debe indicar el número de lados que debe tener una figura para ser un cuadrilátero, y el número de lados que debe tener para no ser un cuadrilátero. El átomo objetivo es el siguiente:

$$clase\,(Figura, Clase) \leftarrow$$

Donde: $Clase \in \{cuadrilatero, no\_cuadrilatero\}$ y
$Figura \in \{cuadrado, rectangulo, trapecio, rombo, romboide, trapezoide, triangulo, circulo, pentagono, hexagono, elipse\}$

El conocimiento previo contiene una literal que indica el número de lados, esta información la presentamos a continuación:

$lados\,(cuadrado, 4)\,.\ lados\,(rectangulo, 4)\,.$
$lados\,(triangulo, 3)\,.\ lados\,(circulo, 0)\,.$
$lados\,(pentagono, 5)\,.\ lados\,(hexagono, 6)\,.$
$lados\,(trapecio, 4)\,.\ lados\,(rombo, 4)\,.$
$lados\,(elipse, 0)\,.\ lados\,(romboide, 4)\,.$
$lados\,(trapezoide, 4)\,.$

El método propuesto fue aplicado al segundo argumento de la literal *lados*, por lo que se realizó una partición del número de lados de los cuadriláteros y los que no lo son. A continuación presentamos los resultados obtenidos.

Resultados

En la tabla 7.3, se muestran los resultados obtenidos al clasificar un conjunto de figuras tomando en cuenta el número de lados que tiene cada una de ellas. En este ejemplo el número de cláusulas que componen cada hipótesis es la mitad, en promedio, con aprendizaje multivalores. En cuanto al tiempo de ejecución es el mismo en promedio por lo que no es necesario mencionar más al respecto, sin embargo en los dos siguientes ejemplos debemos tomarlo en cuenta, ya que el conjunto de ejemplos que se debió procesar con cada método es mucho más grande.

Debemos resaltar también que el porcentaje de ejemplos cubiertos no se mejoró con aprendizaje multivalores, se necesitarían muchos ejemplos para determinar la razón de por qué en los dos ejemplos anteriores se mejoró la precisión y en este ejemplo en particular no fue así; tomando en cuenta además que los tres problemas son similares, es decir, en los tres ejemplos vistos hasta ahora la partición de un conjunto de valores categóricos permitió construir hipótesis con menos reglas.

	Fold 1	Fold 2	Fold 3	Promedio
Número de Reglas	===	===	===	===
Aleph	4 reglas	3 reglas	5 reglas	4 reglas
Aleph multivalores	2 reglas	2 reglas	2 reglas	2 reglas
FOIL multivalores	2 reglas	2 reglas	2 reglas	2 reglas
Porcentaje de ejemplos cubiertos	===	===	===	===
Aleph	75%	50%	100%	75%
Aleph multivalores	75%	50%	100%	75%
FOIL multivalores	75%	50%	100%	75%
Tiempo de ejecución	===	===	===	===
Aleph	0.004 seg.	0.004 seg.	0.0079 seg.	0.0053 seg.
Aleph multivalores	0.004 seg.	0.004 seg.	0.008 seg.	0.005333 seg.

Tabla 7.3: Resultados obtenidos en el problema de figuras geométricas.

A continuación presentamos una de las hipótesis obtenidas para este ejemplo. Sin aprendizaje multivalores obtenemos las siguientes reglas:

- $clase\,(A, cuadrilatero) \leftarrow lados\,(A, 4)$.
- $clase\,(triangulo, no\_cuadrilatero)$.
- $clase\,(A, no\_cuadrilatero) \leftarrow lados\,(A, 0)$.

Con aprendizaje multivalores obtenemos únicamente dos cláusulas:

- $clase\,(A, cuadrilatero) \leftarrow lados\,(A, B)\,, member\,(B, [4])$.
- $clase\,(A, no\_cuadrilatero) \leftarrow lados\,(A, B)\,, member\,(B, [0, 3])$.

Este ejemplo y los dos anteriores, son pequeños y sin embargo hemos podido ver que hay ventaja al aplicar el aprendizaje multivalores a la ILP. Observemos además que aún cuando se aumente el número de figuras geométricas que se utilicen en el conocimiento previo, el número de reglas multivalores seguirán siendo dos, esto debido a que realizamos una partición binaria, y la naturaleza del problema y su solución también es binaria: cualquier figura es o no un cuadrilátero.

7.2 Problema: Préstamo de estudiante

Los datos utilizados en este problema y el siguiente son dos bases de datos obtenidas del repositorio de aprendizaje maquinal [8]. Para este problema el objetivo es encontrar un programa lógico que indique cuándo debe un estudiante devolver un préstamo. La relación objetivo para este problema es:

$$no\_payment\_due\,(Estudiante) \leftarrow$$

La relación anterior es verdadera solo para aquellos estudiantes que no están obligados a pagar algún préstamo estudiantil.

Los predicados declarados en el conocimiento previo, proporcionan información acerca del género del estudiante, del tiempo máximo que se ha ausentado de la escuela, la universidad a la que está inscrito, si está desempleado, si está en bancarrota y si tiene alguna discapacidad; y dependiendo de la información anterior, el estudiante no estará obligado a pagar su préstamo de estudiante. A continuación presentamos los respectivos predicados que forman el conocimiento previo:

- $male\,(Estudiante)$.
- $longest\_absense\_from\_school\,(Estudiante, Numero\_de\_meses)$. Donde $Numero\_de\_meses$ es numérico.
- $enrolled\,(Estudiante, Escuela, Unidades)$. Donde $Escuela \in \{ucsd, ucb, ucla, uci, occ, smc\}$ y $Unidades$ es numérico.
- $enlist\,(Estudiante, Organizacion)$. Donde: $Organizacion \in (army, navy, airforce, marines, peacecorps, firedepartment, foreignlegion\}$.
- $unemployed\,(Estudiante)$.
- $filed\_for\_bankrupcy\,(Estudiante)$.
- $disabled\,(Estudiante)$.

Este problema consta de dos atributos numéricos: *Numero_de_meses* y *Unidades*; y dos atributos categóricos: *Escuela* y *Organizacion*. Por otro lado la base de datos consta de 100 ejemplos positivos que representan los estudiantes que no están obligados a pagar un préstamo estudiantil. A continuación presentamos los resultados obtenidos.

Resultados

Tomando en cuenta que este problema y el siguiente se dividieron en diez subconjuntos para la validación cruzada, y que mostrar los resultados de todos ellos sería cargar con demasiada información las tablas de resultados, exponemos únicamente el promedio del número de reglas, del porcentaje de ejemplos cubiertos y del tiempo de ejecución de cada método.

En la tabla 7.4 mostramos los resultados para esta base de datos, y se hace evidente el menor número de reglas por hipótesis creadas por los métodos ILP multivalores. También el

	Promedio
Número de Reglas	===
Aleph	9 reglas
Aleph multivalores	6.2 reglas
FOIL multivalores	5 reglas
Porcentaje de ejemplos cubiertos	===
Aleph	71%
Aleph multivalores	89%
FOIL multivalores	87%
Tiempo de ejecución	===
Aleph	1.658 seg.
Aleph multivalores	2.8868 seg.

Tabla 7.4: Resultados obtenidos para el problema: préstamo de estudiante.

porcentaje de ejemplos cubiertos se mejora notablemente. Sin embargo el promedio del tiempo de ejecución aumenta, y tomando en cuenta que existen bases de datos mucho mayores, será necesario realizar a futuro un análisis con una gran cantidad de bases de datos para saber la complejidad temporal que implica nuestro método al aplicarlo a algoritmos ILP.

Ahora comparemos una de las hipótesis obtenidas sin aprendizaje multivalores con una creada utilizando el método que hemos propuesto. La hipótesis sin cláusulas multivalores es:

- $no\_payment\_due\,(A) \leftarrow filedforbank\,(A).$
- $no\_payment\_due\,(A) \leftarrow longestabsense\,(A, 0)\,, enrolled\,(A, B, C).$
- $no\_payment\_due\,(A) \leftarrow disabled\,(A).$
- $no\_payment\_due\,(A) \leftarrow unemployed\,(A).$
- $no\_payment\_due\,(A) \leftarrow longestabsense\,(A, 1)\,, enrolled\,(A, B, C).$
- $no\_payment\_due\,(A) \leftarrow enrolled\,(A, B, 12).$
- $no\_payment\_due\,(A) \leftarrow longestabsense\,(A, 9)\,, enlist\,(A, B).$
- $no\_payment\_due\,(A) \leftarrow enrolled\,(A, smc, B)\,, enlist\,(A, C).$
- $no\_payment\_due\,(A) \leftarrow enrolled\,(A, uci, 8).$
- $no\_payment\_due\,(A) \leftarrow enlist\,(A, army).$

La hipótesis con cláusulas multivalores es:

- $no\_payment\_due\,(A) \leftarrow enrolled\,(A, B, C)\,, C > 6.30860223, longestabsense\,(A, D)\,, D \leq 4.53507734.$
- $no\_payment\_due\,(A) \leftarrow enlist\,(A, B)\,, member\,(B, [army, airforce, navy, peacecorps, marines]).$
- $no\_payment\_due\,(A) \leftarrow disabled\,(A).$
- $no\_payment\_due\,(A) \leftarrow filedforbank\,(A).$

- $no\_payment\_due\,(A) \leftarrow unemployed\,(A)$.

De las hipótesis anteriores podemos observar que las literales que tienen una mayor importancia son aquellas que contienen los atributos elegidos para realizar aprendizaje multivalores: *enrrolled, longestabsense y enlist*. De estas literales la hipótesis sin cláusulas multivalores solamente presenta un dato categórico para *enlist*: *army*; y la hipótesis con cláusulas multivalores presenta los siguientes valores: *army, airforce, navy, peacecorps, marines*. Por lo tanto podemos decir que para que un estudiante no esté obligado a realizar el pago de su préstamo, es importante estar enlistado en una de las anteriores corporaciones y no solamente en el ejército (*army*).

De manera general podemos concluir que muchos valores que son importantes para determinar las hipótesis, no son tomados en cuenta en aquellos programas lógicos que no contienen cláusulas multivalores. Este mismo comportamiento lo vemos en el siguiente ejemplo.

7.3 Problema: Concesión de créditos

La segunda base de datos del repositorio de aprendizaje maquinal que analizamos (*Japanese Credit Screening Data Set*) consta de ejemplos de personas a las que se les concedió o no créditos (ejemplos positivos y negativos respectivamente). El objetivo de este problema es encontrar un programa lógico que determine si es factible concederle crédito a una persona. Esta información fue generada a partir de empresas japonesas que concedieron créditos. La relación objetivo es:

$$creditscreening\,(Persona) \leftarrow$$

Esta relación es verdadera para aquellas personas a quienes se les concedió un crédito.

La decisión de otorgarle crédito a una persona depende de varios parámetros como: desempleo, el tipo de compras que realiza, su género, su estado civil, si vive en una región problemática, su edad, el monto de sus depósitos, si su pago es mensual y el número de años que tiene de cliente. Las literales que representan la información anterior se declaran con las literales siguientes:

- $jobless\,(Persona)$.
- $purchase\_item\,(Persona, Opcion)$.
  Donde $Opcion \in \{pc, car, stereo, jewel, medinstru, bike, furniture\}$.
- $male\,(Persona)$.
- $female\,(Persona)$.
- $unmarried\,(Persona)$.
- $problematic\_region\,(Persona)$.
- $age\,(Persona, Opcion)$. Donde “Opcion” es de tipo numérico.
- $deposit\,(Persona, Opcion)$. Donde “Opcion” es de tipo numérico.

- $monthly\_payment\,(Persona, Opcion)$. Donde "Opcion" es de tipo numérico.
- $numb\_of\_months\,(Persona, Opcion)$. Donde "Opcion" es de tipo numérico.
- $numb\_of\_years\_in\_company\,(Persona, Opcion)$. Donde "Opcion" es de tipo numérico.

De las literales anteriores podemos ver que tenemos cinco atributos de tipo numérico y uno de tipo categórico, por lo tanto analizamos estos seis con aprendizaje multivalores. El conjunto consta de 85 ejemplos positivos y 40 negativos. A continuación presentamos los resultados obtenidos.

Resultados

La tabla 7.5 muestra los resultados para esta base de datos, en ella podemos notar que el número de reglas que componen una hipótesis decrece con los métodos ILP multivalor. Por otro lado vemos también que la precisión o porcentaje de ejemplos cubiertos mejora, por lo que también es necesario a futuro realizar un análisis al respecto para determinar de manera formal este incremento en la precisión.

	Promedio
Número de Reglas	===
Aleph	17.8 reglas
Aleph multivalores	14.7 reglas
FOIL multivalores	10.1 reglas
Porcentaje de ejemplos cubiertos	===
Aleph	79.87%
Aleph multivalores	82.23%
FOIL multivalores	96.66%
Tiempo de ejecución	===
Aleph	1.39 seg.
Aleph multivalores	1.97 seg.

Tabla 7.5: Resultados obtenidos para el problema: concesión de créditos.

También es evidente el incremento del tiempo de ejecución con el método ILP multivalores, y como mencionamos anteriormente un análisis de este aspecto podrá servirnos para saber si este aumento del tiempo de ejecución será un inconveniente para el proceso de bases de datos mucho más grandes.

A continuación describimos la diferencia que hay en los atributos mostrados por las hipótesis obtenidas con los métodos evaluados:

1. **Número de meses.** Este es un atributo importante puesto que la hipótesis obtenida con nuestro método indica que este número debe ser menor a 12.89, en cambio sin aprendizaje multivalores se muestran los valores: 5,10,12,15 y 20. Estos últimos valores se presentan en siete cláusulas univalor, mientras que con nuestro método el valor obtenido se muestra en cuatro cláusulas multivalores.

2. **Número de años en la compañía.** Sin aprendizaje multivalores, el número de años en la compañía determinado por el algoritmo ILP es de 5 y 7, mientras que con nuestro método obtenemos que el cliente debe tener al menos 3.2 años como cliente en la compañía, lo cual parece ser más razonable.

3. **Tipo de artículos comprados.** Sin aprendizaje multivalores, el tipo de artículos comprados por los clientes determinado por el algoritmo ILP son: *stereo, jewel, medinstru o medical instruments y furniture*, los cuales son declarados en cinco cláusulas. Por otro lado con nuestro método tenemos los valores: *jewel,medinstru,furniture*. Estos últimos valores declarados en tres cláusulas.

Como podemos ver en la lista anterior, los valores de cada atributo son declarados en menos cláusulas, por lo que una de nuestras conclusiones es que efectivamente se obtienen hipótesis con menos reglas. En el capítulo 8 presentamos esta y otras conclusiones que hemos determinado al finalizar los análisis presentados.

Capítulo 8

Conclusiones y trabajo futuro

En el capítulo 1 presentamos como objetivo principal, la adición del aprendizaje multivalores a algoritmos ILP, con lo cual esperamos obtener hipótesis con menos reglas, y los resultados del trabajo y análisis realizado los presentamos en el capítulo 7. Sin duda alguna al ver los resultados pudimos comprobar que efectivamente se redujo el número de reglas por hipótesis en cada uno de los ejemplos que analizamos. Sin embargo es necesario resumir tales resultados y concluir sobre ellos, además de plantear a futuro lo que se puede mejorar en la ILP, en este capítulo abarcamos estos puntos.

8.1 Conclusiones

Como hemos visto en el capítulo 7, nuestra propuesta ha logrado reducir el número de reglas con que representamos cada hipótesis. Además en la mayoría de los ejemplos analizados la precisión se mejora.

Esta reducción de cláusulas es consecuencia de la partición que realizamos sobre el conjunto de reglas sobre las cuales realizamos el aprendizaje multivalores. Esta partición es binaria debido al uso del algoritmo k-means con $k = 2$, por lo que por cada predicado del conocimiento previo analizado, sólo podemos obtener dos cláusulas que pueden formar parte de la hipótesis.

A diferencia del análisis multivalor, los algoritmos ILP sin este tipo de aprendizaje comparan un mayor número de cláusulas, ya que utilizan uno por uno los valores, ya sean numéricos o categóricos. Como consecuencia el conjunto de cláusulas crece y de este conjunto una, dos o más pueden formar parte de la hipótesis final.

Podemos añadir que nuestra propuesta no mejora la eficacia con la que se buscan las hipótesis en el espacio de estas, sino que amplía este conjunto con mejores hipótesis. Es decir, los algoritmos ILP a los que se les permitió realizar aprendizaje multivalores siguen realizando la búsqueda de las hipótesis de la misma manera, pero el espacio de búsqueda contiene además hipótesis multivalores.

Vale la pena resumir los aspectos de los algoritmos de ILP sobre los cuales se trabajó en esta tesis:

- Prejuicio de lenguaje: Se permitió la inclusión de nuevas literales para formar nuevas reglas.

- Prejuicio de búsqueda: Las nuevas reglas (multivalores) hacen crecer el espacio de búsqueda. En este aspecto debemos mencionar que a pesar de haber aumentado el espacio de búsqueda, el tiempo de ejecución no aumentó significativamente por lo que es necesario estudiar este fenómeno a futuro.

- Prejuicio declarativo: El usuario determina las cláusulas que son apropiadas para realizar el aprendizaje multivalores.

Para hacer uso de las ventajas del aprendizaje multivalores, es necesario identificar dos tipos de predicados.

- El primero de ellos es aquel predicado que tiene al menos una entrada cuyos valores son categóricos. Ejemplo de estos predicados los vimos en los primeros ejemplos: número de lados de figuras geométricas, número de patas en animales, tipo de pelaje en animales, dirección hacia donde apuntan los triángulos, etc.

- El otro tipo de predicado que debemos identificar es aquel con al menos una entrada cuyos valores son numéricos. La base de datos *japanese credit* tiene cinco predicados con este tipo de entradas.

Además de identificar el tipo de atributo, también es necesario determinar si dicha entrada puede generalizar mejor con un valor categórico, numérico o con una variable.

Si la entrada puede generalizar el conjunto de ejemplos positivos con una variable, entonces no debe ser tomada para realizar aprendizaje multivalores. Por ejemplo si queremos generalizar una relación familiar con una hipótesis, como la relación *tío*, entonces en el conocimiento previo podemos tener la literal de la relación 8.1. Si realizamos aprendizaje multivalores con este predicado obtendríamos una cláusula como la de la ecuación 8.2, ya que los valores que toma la segunda entrada de la relación *hermano_de* es tipo categórica, sin embargo no generaliza de manera adecuada, ya que no todo el mundo es hermano de paco o de manuel.

$$hermano\_de\,(A, B)\,. \tag{8.1}$$

$$tio\_de\,(A, B) \leftarrow hermano\_de\,(A, C)\,, member\,(C, [paco, manuel])\,, progenitor\_de\,(C, B)\,. \tag{8.2}$$

Si la entrada de un predicado, ya sea numérico o categórico, puede generalizar a todo el universo de ejemplos posible, entonces podemos utilizar aprendizaje multivalores. Por ejemplo, si tenemos el predicado de la ecuación 8.3, podemos observar que su segunda entrada es numérica. Por otro lado si dicho predicado es utilizado como conocimiento previo para generalizar el concepto *mayor_de_edad*, entonces, es posible utilizar aprendizaje multivalores, ya que el split encontrado es 18, y es aplicable a cualquier persona, es decir, a cualquier elemento del universo de ejemplos que se puede construir. La hipótesis correspondiente la mostramos en la ecuación 8.4

$$edad\,(Persona, Edad)\,. \tag{8.3}$$

$$mayor\_de\_edad\,(Persona) \leftarrow edad\,(Persona, Edad)\,, Edad \geq 18. \tag{8.4}$$

Resumiento y contestando a las preguntas planteadas en la sección 6.2.3 concluimos:

- El método efectivamente permite reducir el número de reglas que componen las hipótesis. Esto sucede cuando los problemas son de clasificación, y los atributos elegidos para crear las cláusulas multivalores son importantes para realizar la clasificación.
- No todos los atributos pueden elegirse para crear las cláusulas multivalores. Además de ser categórico o numérico, un atributo puede elegirse únicamente si la información que aporta dicho atributo puede generalizar a todo el universo de ejemplos posible.

8.2 Trabajo futuro

Además de los resultados mostrados y comentados anteriormente, el presente trabajo permitió advertir diversas mejoras que se pueden realizar en trabajos futuros, y que sin duda alguna son oportunidades de desarrollo en el área del aprendizaje maquinal, a continuación listamos todas las que planteamos.

- Para realizar un mejor análisis de nuestra propuesta, es necesario implementar el aprendizaje multivalores con más algoritmos ILP, además es necesario utilizar más bases de datos y con un número mucho mayor de ejemplos.
- Es posible ampliar aún más el espacio de búsqueda implementando los algoritmos multivalores pero con divisiones múltiples, es decir, con más de dos particiones. Es necesario también identificar el tipo de problemas de ILP a los que se les puede aplicar este tipo de particiones.
- Para esta tesis hemos elegido dos de los mejores algoritmos inductores de árboles multivalores, sin embargo no son los únicos, y es necesario implementar más de los ya existentes, de manera que puedan ser comparados utilizando bases de datos mucho mayores y para problemas ILP más complejos.
- En esta tesis calculamos el tiempo de ejecucion de Aleph y Aleph multivalores y compararlos, sin embargo esto sólo nos da una idea del incremento que resulta de realizar aprendizaje multivalor en ILP. Como consecuencia es necesario realizar un estudio exhaustivo sobre este incremento de la complejidad temporal.
- En todos los ejemplos expuestos en el capítulo 7 calculamos la precisión y pudimos apreciar que el porcentaje de ejemplos cubiertos aumentó, por lo que al igual que la complejidad temporal, también es necesario realizar un análisis más a fondo sobre este punto.
- Como hemos visto, en el presente trabajo tomamos una entrada de algún predicado y realizamos el aprendizaje multivalor, sin embargo ¿qué pasaría si tenemos un predicado con dos o más entradas, ya sean categóricas o numéricas, y realizamos aprendizaje multivariable? En este caso ¿se reducirá aún más el número de reglas por cada hipótesis y

se mejorará además la precisión de estas? ¿Qué tipo de problemas de ILP son adecuados para este tipo de aprendizaje? El uso del aprendizaje multivariable es una oportunidad para mejorar aún más las técnicas ILP.

- Con todos los puntos anteriores es necesario crear un software que nos permita probar cada una de las propuestas. Weka [25] es un ejemplo de este tipo de software para probar diversos algoritmos de aprendizaje maquinal.

8.3 Balance

Resumiendo los logros que alcanzamos durante el desarrollo de esta tesis podemos mencionar:

- Al estudiar e implementar los algoritmos de árboles de decisión mencionados en capítulos anteriores, hemos identificado un área de oportunidad muy importante al plantear para trabajos futuros la implementación de algoritmos multivariables.
- El resultado más importante se dió al comprobar la reducción de reglas que componen las hipótesis generadas por los algoritmos ILP multivalores.
- La implementación de los algoritmos, tanto de ILP como de árboles de decisión, nos ha permitido identificar la importancia de tomar en cuenta que el espacio de búsqueda puede crecer demasiado. Como consecuencia es necesario que para futuras implementaciones el diseño de estos algoritmos tome en cuenta el uso de memoria.
- El presente trabajo sirve como plataforma para mejorar aún más los algoritmos ILP.

Sin embargo ¿qué no se hizo durante el progreso de esta tesis?

- Faltó realizar un análisis más profundo sobre la complejidad temporal y la precisión de los algoritmos ILP multivalores.
- La implementación de FOIL multivalores, no permitió utilizar bases de datos suficientemente grandes como para realizar análisis temporal y de precisión (p.e. bases de datos con 600 o más ejemplos de entrenamiento).
- No se realizó una arquitectura apropiada que sirva de base para futuras mejoras que se puedan realizar a esta clase de algoritmos, y que además incluya a más de un algoritmo.
- Una interface gráfica de usuario tampoco fue implementada, aunque esto debe ser tomado en cuenta en posteriores implementaciones. Esto es necesario para que este tipo de sistemas sea más amigable con usuarios que no son expertos en el área de aprendizaje maquinal, pero que necesitan realizar este tipo de análisis.

Es necesario recalcar también que no sólo los puntos enlistados en la sección de trabajo futuro servirán de base para realizar mejoras en los algoritmos ILP, sino que también el trabajo que no se realizó es un área de oportunidad muy importante que debe tomarse en cuenta para continuar desarrollando algoritmos de ILP más eficientes.

Otro punto importante a tomar en cuenta es referente a la decisión tomada en este trabajo para resolver problemas de clasificación con algoritmos ILP en lugar de utilizar inductores de árboles de decisión. Sobre todo tomando en cuenta que los árboles de decisión ya realizan aprendizaje multivalores. Nuestra decisión se basó principalmente en la mayor capacidad expresiva de la lógica de primer orden, utilizada en ILP, para plantear los problemas y para representar las soluciones a dichos problemas. Es decir, los árboles de decisión utilizan el lenguaje proposicional para interpretar cada árbol de decisión, sin embargo el lenguaje de la lógica de primer orden es mucho más expresivo ya que permite representar objetos y relaciones entre ellos, también añade el uso de variables para denotar diferentes objetos.

Apéndice A

Definiciones de Lógica de Primer Orden

La *Programación Lógica Inductiva* hace suyo el lenguaje de la *lógica de primer orden* para representar con mayor naturalidad los elementos que componen un sistema de *aprendizaje inductivo*, a continuación presento las definiciones de la *lógica de primer orden* utilizadas en el capítulo 4.

- Las *variables* están representadas como una cadena de caracteres, en la cuál, el primer carácter es una letra mayúscula: $A, B, ..., X, Y$.
- Una *función* está representada por una cadena de caracteres, donde el primer carácter es una letra minúscula, por ejemplo, $f(g(X), h)$ es una *función*.
- Una *variable* y una *función* son *términos*.
- Un *predicado* está representado por una cadena de caracteres, donde el primer carácter es una letra minúscula, por ejemplo, con *padre(X,Y)* estamos indicando que *X* es el padre de *Y*, donde *X* y *Y* son variables.
- El símbolo de negación está representado por el carácter ¬, por ejemplo, $\neg padre\,(X, Y)$ indica que *X* ***no*** es el padre de *Y*.
- Un *átomo* o *fórmula atómica* es cualquier predicado de aridad *n*, por ejemplo, *suma(X,Y,Z)* es el átomo con el que indicamos que $X + Y = Z$ y la aridad de este *predicado* es 3.
- Una *literal* es un átomo o su negación. Tanto *padre(X,Y)* como $\neg padre\,(X, Y)$ son *literales*.
- Una *cláusula* es una disyunción de literales, así:

$$(L_1 \vee L_2 \vee .... \vee \neg L_1 \vee \neg L_2 \vee ...) \equiv$$
$$(L_1, L_2, ..., \neg L_1, \neg L_2) \equiv$$
$$L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow L_1 \vee L_2 \vee ... \vee L_n$$

  son *literales*.

- El conjunto de *literales* positivas es llamado la *cabeza* de la *cláusula* y el conjunto de las negativas es llamado el *cuerpo* de la cláusula, por ejemplo:

  Sea la cláusula $T = (\neg L_1 \vee \neg L_2 \vee L_3 \vee L_4)$ entonces
  podemos re-escribirla como: $T = (L_1 \wedge L_2 \rightarrow L_3 \vee L_4)$, y:
  $\{L_1, L_2\}$ es la *cabeza* y $\{L_3, L_4\}$ es el *cuerpo* de la *cláusula*.

- La *cláusula de Horn* o *Definite Clause* [18] es aquella cláusula que tiene como máximo una *literal* positiva, por ejemplo:

  $$T_1 = (\neg L_1 \vee \neg L_2 \vee ... \vee \neg L_n) \equiv L_1 \wedge L_2 \wedge ... \wedge L_n \rightarrow$$
  $$T_2 = (\neg L_1 \vee \neg L_2 \vee ... \vee L_n) \equiv L_1 \wedge L_2 \wedge ... \wedge L_{n-1} \rightarrow L_n$$

  son *cláusulas de Horn*.

- Un *Programa Lógico* es un conjunto de cláusulas de Horn.

- Sea un $\theta = \{v_1/t_1, ..., v_n/t_n\}$. Decimos que θ es una substitución cuando cada v_i es una *variable* y cada t_i es un *término*.

- Una substitución que hace que dos átomos sean iguales se conoce como *unificador*, por ejemplo:

  La substitución $\theta = \{x/f(A), y/g(u), z/A\}$ es un unificador para las literales: $L_1 \equiv R(x, g(u))$ y $L_2 \equiv R(f(z), y)$ ya que $L_1\theta \equiv L_2\theta$

Observación: Las implicaciones $P \rightarrow Q$ suelen representarse como: $Q \leftarrow P$ en el contexto de la *Programación Lógica* y en el lenguaje de *Programación Lógica* llamado *Prolog* suele representarse como *Q :- P* y se lee como *P implica Q*, *Q si P* o *Si P entonces Q*.

Apéndice B

Introducción a Prolog

Prolog es un lenguaje de programación desarrollado para resolver problemas que involucran objetos y las relaciones entre estos. La principal característica de este lenguaje es que para representar los datos y sus relaciones utiliza el lenguaje de la lógica de primer orden. Un ejemplo típico que nos ayuda a familiarizarnos con *Prolog* es el siguiente: supongamos que tenemos el árbol genealógico de la figura B.1 y queremos definir dos relaciones, la primera indica quien es progenitor de otra persona, y la segunda indica quién es el antepasado de otra persona, (ejemplo tomado de [3]).

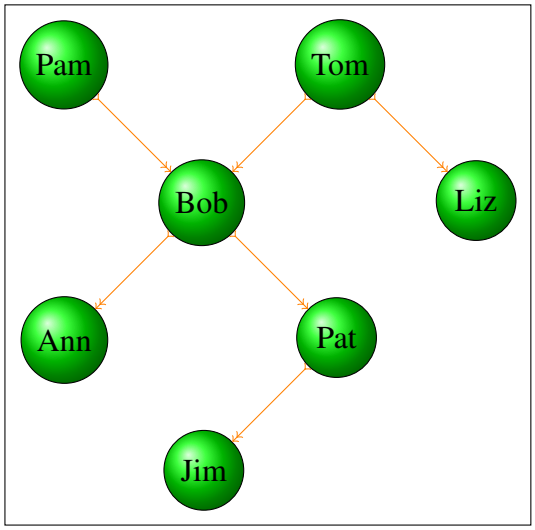


Figura B.1: Ejemplo de relación: Árbol genealógico

En *Prolog* podemos definir las relaciones entre objetos de dos maneras: con *hechos* y con *reglas*.

Los *hechos* son afirmaciones que se pueden hacer directamente sobre uno, dos o más objetos; la sintáxis básica de un hecho es la siguiente: $p(X_1, X_2, ..., X_m)$, donde m es la aridad del hecho, y p es el nombre de la relación llamado predicado. Para la primera relación del ejemplo del árbol genealógico tenemos los siguientes hechos:

1. progenitor(pam,bob).

2. progenitor(tom,bob).
3. progenitor(tom,liz).
4. progenitor(bob,ann).
5. progenitor(bob,pat).
6. progenitor(pat,jim).

De las relaciones anteriores podemos notar que los nombres no se escribieron en mayúscula pues son constantes y su primer caracter es minúscula. Cada línea de código debe terminar con un punto.

Una *regla* en *Prolog* es una cláusula del tipo $B \leftarrow A$, que se denota como B:-A, y donde B es una conjunción de literales; el símbolo de la conjunción es la coma. Regresando al problema del árbol genealógico, si queremos relacionar a cada objeto o persona de manera que especifiquemos si una persona es antepasado de otra, podemos hacerlo de dos formas:

1. Con hechos. Al igual que la relación *progenitor* podemos listar todas las relaciones posibles: antepasado(pam, bob), antepasado(pam, pat), etc. Esta manera resultaría muy engorrosa, además sólo se limitaría a este árbol genealógico, ya que si se agregarán más personas (objetos) al árbol genealógico se tendrían que agregar sus correspondientes relaciones de descendencia.

2. Otra manera es creando *reglas* que generalicen la relación *antepasado* a partir de la relación progenitor. Para la segunda relación del ejemplo tenemos dos reglas que podemos crear, la primera es la inmediata: si X es progenitor de Y entonces X es antepasado de Y, la segunda es recursiva: si X es progenitor de Z y Z es antepasado de Y entonces X es antepasado de Y, en sintáxis de *Prolog* las reglas serían las siguientes:

   - 7. antepasado(X,Y) :- progenitor(X,Y).
   - 8. antepasado(X,Y) :- progenitor(X,Z), antepasado(Z,Y).

En el caso de la regla 8, podemos ver que la recursiva tiene dos literales del lado derecho, en este caso la coma es equivalente a la conjunción en la lógica de predicados. También podemos ver que podemos definir una relación utilizando una o más reglas, en este caso la primer regla es el caso base, y la segunda es la recursiva.

Tanto los hechos como las reglas son cláusulas y un programa *Prolog* es un conjunto de ellas, por lo que el programa que representa las relaciones progenitor y antepasado de nuestro ejemplo es el conjunto de las 8 cláusulas planteadas anteriormente.

Además de hechos y reglas podemos tener metas que nos permiten hacer preguntas a *Prolog* respecto al programa que tenemos (en este caso programa es el conjunto de los hechos donde definimos la relación *progenitor* y las dos reglas donde definimos la relación *antepasado*). Respecto a nuestro programa podemos hacer las siguientes preguntas:

1. antepasado(X,jim) $\equiv$ ¿Quienés son los antepasados de Jim?
2. antepasado(pam, X) $\equiv$ ¿Quienés descienden de Pam?

3. progenitor(pam, bob) $\equiv$ ¿Pam es progenitor de Bob?

Cabe añadir que las variables son las cadenas de caracteres que comienzan con un guión bajo o con una letra mayúscula, y son importantes en el caso de las cuestiones que le hacemos a *Prolog* respecto a algún programa puesto que nos devolverá todas las posibles soluciones, en el caso de la segunda pregunta nos devolverá: X = bob, X=ann, X=pat, X=jim.

Unificación

En Prolog existen tres tipos de términos: constantes, variables y términos complejos; teniendo esto en cuenta definimos la unificación como sigue:

DEFINICIÓN B.1 *Dos términos se unifican, si son iguales o si contienen variables que pueden ser instanciadas de tal manera que los dos términos se vuelvan iguales.*

Ejemplos:

- $term_1 \equiv ama\,(X, maria)$ unifica a $term_2 \equiv ama\,(juan, Y)$ ya que si $X \equiv juan$ y $Y \equiv maria$ entonces $term_1 \equiv term_2$.
- $term_1 \equiv 5$ unifica a $term_2 \equiv 5$.
- $term_1 \equiv 5$ no unifica a $term_2 \equiv 55$.
- $term_1 \equiv ama\,(jose, maria)$ no unifica a $term_2 \equiv ama\,(juan, Y)$.

Principio de resolución

El principio de resolución de Robinson [26], es una regla de inferencia que en la lógica de proposiciones puede verse como una generalización de la regla de inferencia *Modus Tollens* $((\neg p \vee q) \wedge \neg q \Longrightarrow \neg p)$:

$$\frac{C_1, C_2}{(C_1 \setminus \{A\}) \vee (C_2 \setminus \{\neg A\})} si A \in C_1, \neg A \in C_2 \tag{B.1}$$

Para todo par de cláusulas C_1 y C_2, si existe una literal $L_1 \equiv A$ en C_1, que es complemento de otra literal $L_1 \equiv \neg A$ en C_2, entonces se eliminan L_1 y L_2 de C_1 y C_2 respectivamente, y se construye la disyunción de las cláusulas restantes. A la cláusula obtenida se le llama *resolvente de C_1 y C_2*.

Un ejemplo lo mostramos en la ecuación B.2.

$$\frac{a \vee b,\ \neg a \vee c}{b \vee c} \tag{B.2}$$

La resolución anterior la leemos: Si (a ó b es verdad) y (a es falsa ó c es verdad) Entonces b ó c es falsa.

Apéndice C

Análisis Estadísticos

En este capítulo presentamos los métodos estadísticos utilizados: análisis de varianza, Pearson Ji-cuadrada, y análisis de Levene; los cuales pueden consultarse también en [32]. A continuación presentamos la notación utilizada en las siguientes secciones.

Notación	Significado
Y	Variable dependiente o concepto objetivo.
$X_m, m = 1, \ldots, M$	Conjunto de atributos del concepto objetivo.
$\hbar = \{x_n, y_n\}_{n=1}^{N}$	Conjunto de entrenamiento
$\hbar(t)$	Elementos del conjunto de entrenamiento que caen en el nodo t
f_n	Frecuencia asociada al caso n.
N_f	Número total de ejemplos.
$N_{f,j}$	Número total de ejemplos en la clase j.
$N_{f(t)}$	Número total de ejemplos en el nodo t.
$N_{f,j}(t)$	Número total de ejemplos de la clase j en el nodo t.
$n_{i\bullet}$	Suma de todos los valores correspondientes a la categoría i.
$n_{\bullet j}$	Suma de los valores correspondientes a la clase j.
$n_{\bullet\bullet}$	Suma de todos los valores correspondientes a cada una de las clases.

Tabla C.1: Notación utilizada en los análisis estadísticos

C.1 Análisis de Varianza

Suponemos, para cada nodo t, que hay J_t clases de la variable dependiente Y. El estadístico F para una variable numérica X está dado por

$$F_x = \frac{\sum_{j=1}^{J_t} N_{f,j}(t) \left(\bar{x}^{(j)}(t) - \bar{x}(t)\right)^2 / (J_t - 1)}{\sum_{n \in \hbar(t)} f_n \left(x_n - \bar{x}^{y_n}(t)\right)^2 / (N_f(t) - J_t)}$$

donde

$$\bar{x}^{(j)} = \frac{\sum_{n \in \hbar(t)} f_n x_n I(y_n = j)}{N_{f,j}(t)}, \bar{x}(t) = \frac{\sum_{n \in \hbar(t)} f_n x_n}{N_f(t)}.$$

Su correspondiente p-valor está dado por

$$p_x = Pr\left(F\left(J_t - 1, N_f\left(t\right) - J_t\right) > F_x\right)$$

donde $F\left(J_t - 1, N_f\left(t\right) - J_t\right)$ sigue una F-distribución con $J_t - 1$ y $N_f\left(t\right) - J_t$ grados de libertad.

C.2 Análisis Pearson de Ji-cuadrada

Supongamos, para un nodo t, que hay J_t clases de la variable dependiente Y. El estadístico *Pearson de Ji-cuadrada*, para una variable categórica X con I_t categorías, está dado por

$$X^2 = \sum_{j=1}^{J_t} \sum_{i=1}^{I_t} \frac{\left(n_{ij} - \hat{m}_{ij}\right)^2}{\hat{m}_{ij}}$$

donde

$$n_{ij} = \sum_{n \in \hbar(t)} f_n I\left(y_n = j \wedge x_n = i\right), \hat{m}_{ij} = \frac{n_{i\bullet} n_{\bullet j}}{n_{\bullet\bullet}}$$

con

$$n_{i\bullet} = \sum_{j=1}^{J_t} n_{ij}, n_{\bullet j} = \sum_{i=1}^{I_t} n_{ij}, n_{\bullet\bullet} = \sum_{j=1}^{J_t} \sum_{i=1}^{I_t} n_{ij}.$$

donde $I\left(y_n = j \wedge x_n = i\right) = 1$ si para n sucede que $y_n = j$ y $x_n = i$; 0 en otro caso.

El correspondiente p-valor está dado por $p_x = Pr\left(\chi_d^2 > X^2\right)$ donde χ_d^2 sigue una distribución Ji-cuadrada con $d = \left(J_t - 1\right)\left(I_t - 1\right)$ grados de libertad.

C.3 Análisis de Levene

Para una variable numérica X, calcular $z_n = \mid x_n - \hat{x}^{(y_n)} \mid$. El estadístico F de Levene para la variable X es el estadístico F de ANOVA para z_n.

Apéndice D

Transformación CRIMCOORDS

La transformación de datos categóricos a numéricos (presentada en [16]), consta de dos pasos:

1. Los elementos del conjunto de ejemplos que toman valores de una variable categórica, se mapean a *vectores ficticios 0-1*.
2. Los vectores ficticios son proyectados a su coordenada discriminante más grande (este método de reducción de dimensión es llamado ***CRIMCOORDS***).

A continuación detallamos el algoritmo.

Algoritmo D.1 *Transformación CRIMCOORDS*
Supongamos que X es una variable categórica con valores en el conjunto $C = \{c_1, \ldots, c_M\}$.

- *Transformar cada valor de X a un vector ficticio columna: $v = (v_1, \ldots, v_M)^T$, donde*

$$v_l = \begin{cases} 1, & si \;\; X = c_l \\ 0, & en\ otro\ caso \end{cases}$$

  Sea V la matriz de datos de $N \times M$ que contiene los valores del conjunto de datos de entrenamiento correspondientes a la variable categórica X. N corresponde al número de elementos que contiene el conjunto de entrenamiento.

- *Creamos la matriz identidad I de $N \times N$.*
- *Creamos la matriz P de $N \times 1$, con el valor 1 para cada uno de sus elementos.*
- *Calculamos $H = I - \left(N^{-1}\ P\ P^T\right)$.*
- *Calculamos HV.*
- *Realizamos una descomposición de valores singulares a $HV = P\ D\ Q^T$. Donde $D = diag\,(d_1, \ldots, d_M)$ y $d_1 \geq \ldots \geq d_M \geq 0$.*

- *Sea r el número de elementos positivos de la matriz diagonal D. Si algún elemento de D es muy pequeño, entonces se descarta aunque sea positivo.*
- *Creamos la matriz F de $M \times r$, tal que F es una submatriz de Q.*
- *Creamos la matriz diagonal U de $r \times r$ donde $U = diag\left({d_1}^{-1}, \ldots, {d_r}^{-1}\right)$.*
- *Reducimos la dimensión de cada vector ficticio v_i (Donde $i \in \{1, \ldots, M\}$): $y_i = F^T\, v_i$.*
- *Creamos la matriz G.*

  Sea ${v_i}^{(j)}$ que denota el i-ésimo valor observado del vector ficticio v en la j-ésima clase, definimos los vectores M-dimensionales.

$$\bar{v}^{(j)} = {N_j}^{-1} \sum_{i=1}^{N_j} {v_i}^{(j)}, \qquad \bar{v} = N^{-1} \sum_{j=1}^{J} \sum_{i=1}^{N_j} {v_i}^{(j)}.$$

  Observaciones:

  - *$\bar{v}^{(j)}$: Es una matriz de $M \times 1$, con los promedios de aparición de cada uno de los valores categóricos de X, por cada una de las clases.*
  - *$\bar{v}$: Es una matriz de $M \times 1$, con los promedios de aparición de cada uno de los valores categóricos de X, en todo el conjunto de ejemplos.*

  Creamos la matriz L_j de $M \times N_j$, donde j es la j-ésima clase, $L_j = \left(\bar{v}^{(j)} - \bar{v}, \ldots, \bar{v}^{(j)} - \bar{v}\right)$.

  Por último creamos la matriz G de $N \times M$. $G = (L_1, \ldots, L_J)^T$, donde J es el número de clases.
- *Creamos la matriz $GFU = G\ F\ U$.*
- *Realizamos la descomposición de valores singulares de GFU, entonces $GFU = P_2\ D_2\ {Q_2}^T$*
- *Elegimos el vector **a** como el eigenvector asociado al eigenvalor más grande de ${Q_2}^T$, es decir, la columna de ${Q_2}^T$ donde se encuentre el valor más grande de esa matriz.*
- *El valor crimcoord correspondiente a cada vector ficticio v_i es $\xi_i = a^T\, U\, y_i$.*

Apéndice E

Algoritmo Progol

Progol, desarrollado por Stephen Muggleton [20], es un algoritmo ILP que de manera general sigue los pasos que presentamos a continuación:

1. **Selección de ejemplo.** Si no hay ejemplos en el conjunto de entrenamiento termina, en otro caso continua en el paso 2.

2. **Saturación.** Se toma un ejemplo positivo del conjunto de entrenamiento y se construye la cláusula más específica. Para ello se aplica repetidamente la resolución inversa [20] sobre el ejemplo seleccionado, hasta que todos sus términos han sido reemplazados por variables. Este reemplazo, debe ser tal que las variables del cuerpo de la cláusula, estén debidamente ligadas a la literal de la cabeza. La cláusula creada debe cubrir al ejemplo positivo que se seleccionó. Esta cláusula es llamada *bottom clause*.

3. **Reducción.** Se lleva a cabo la búsqueda de la mejor cláusula posible. Esta búsqueda se lleva a cabo entre todas aquellas cláusulas que se encuentren entre la *bottom clause* y la regla más general (aquella con el cuerpo vacío). Debemos notar que el espacio entre la cláusula más general y la *bottom clause* está parcialmente ordenado (es parte de la rejilla que representa a todo el espacio de hipótesis) por las propiedades descritas en las definiciones E.1 y E.2, las cuales fueron presentadas anteriormente en el capítulo 4.

   DEFINICIÓN E.1 *Sean dos cláusulas C y D, decimos que C subsume a D si existe una substitución* $\ominus$ *tal que* $C\ominus \subseteq D$. *Denotado por* $C \succeq D$ *[30].*

   DEFINICIÓN E.2 *Sean dos cláusulas C y D decimos que C es más general que D si la subsume.*

   El algoritmo utilizado en la búsqueda es A^* [9] y la función utilizada para evaluar la mejor cláusula es la siguiente:

   $$f(C) = P - N - L + 1$$

   donde P, N es el número de ejemplos positivos y negativos que la cláusula C cubre respectivamente, y L es el número de literales en la cláusula. Esta función calcula la compresión de cada cláusula evaluada, y la elegida es aquella que tiene mayor compresión (Compresión de Occam [20]).

4. **Remover redundancia.** Los elementos cubiertos por la cláusula seleccionada se eliminan del conjunto de ejemplos y se regresa al paso 1.

A continuación presentamos un problema resuelto con este algoritmo para ejemplificar cada uno de los puntos anteriores.

El objetivo es encontrar una regla que generalice la relación familiar de *abuelo*. Para este problema el único predicado que tenemos es el que indica la relación de *progenitor* entre dos personas, y a continuación presentamos todas las relaciones que forman el conocimiento previo.

- $progenitor\,(juan\_m, paco)$.
- $progenitor\,(paco, jasel)$.
- $progenitor\,(paco, jael)$.
- $progenitor\,(paco, poncho)$.
- $progenitor\,(jasel, leonardo)$.
- $progenitor\,(juan\_m, juan)$.
- $progenitor\,(juan, axel)$.
- $progenitor\,(juan, uriel)$.
- $progenitor\,(juan, miguel)$.
- $progenitor\,(juan, berenice)$.
- $progenitor\,(juan\_m, miriam)$.
- $progenitor\,(miriam, elizabeth)$.
- $progenitor\,(miriam, daniel)$.

Como ejemplos positivos tomamos, de la lista anterior, aquellos que cumplen con la relación de *abuelo*.

- $abuelo\,(juan\_m, jasel)$.
- $abuelo\,(juan\_m, jael)$.
- $abuelo\,(juan\_m, poncho)$ . ...

Y los ejemplos negativos son todas aquellas relaciones *abuelo* que no aparecen en la lista anterior.

A continuación presentamos los pasos seguidos por Progol al ejecutarlo para resolver el problema planteado.

1. Selección de ejemplo

Como es la primera iteración se selecciona el primer ejemplo del conjunto de entrenamiento.

2. Saturación

Se crea la *bottom clause*, la cual mostramos en la ecuación E.1.

$$abuelo\,(A,B) : -progenitor\,(A,C)\,, progenitor\,(A,D)\,, progenitor\,(C,B)\,. \tag{E.1}$$

3. Reducción

La búsqueda de la cláusula con mayor compresión se realiza entre la cláusula de la ecuación E.1 y la cláusula con cuerpo vació (la más general) que presentamos en la ecuación E.2:

$$abuelo\,(A,B)\,. \tag{E.2}$$

Las cláusulas probadas deben ser más específicas que la de la ecuación E.2 y más general que la *bottom clause*, ecuación E.1. A continuación presentamos aquellas que Progol prueba.

1. $abuelo\,(A,B) : -progenitor\,(A,C)\,.$
2. $abuelo\,(A,B) : -progenitor\,(A,C)\,, progenitor\,(C,B)\,.$

De las dos anteriores la número 2 es la que tiene mayor compresión, por lo que se elige y se continua al siguiente paso.

4. Remover redundancia

La cláusula elegida cubre todos los ejemplos positivos, por lo que se eliminan y se regresa al paso 1 del algoritmo.

Como ya no hay ejemplos positivos, en nuestro problema, entonces se termina la ejecución del algoritmo y se devuelve la siguiente hipótesis construida con una sóla cláusula:

$$abuelo\,(A,B) : -progenitor\,(A,C)\,, progenitor\,(C,B)\,.$$

Bibliografía

[1] E. ALPAYDIN, Introduction to Machine Learning, Massachusetts Institute of Technology, 2004.

[2] M. M. BONGARD, Pattern Recognition, Spartan Books, 1970.

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

[4] R. M. CAMERON-JONES AND R. QUINLAN, Avoid pitfalls when learning recursive theories, Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, (1993), pp. 1050–1057.

[5] W. J. DECOURSEY, Statistics and Probability for Engeenering Applications, Newnes, 2003.

[6] D. EDWARDS AND T. HART, The tree prune. memo 30 (revised), Artificial Intelligence Project, RLE and MIT Computation Center, (1963).

[7] B. EFRON, An introduction to the bootstrap, Chapman and Hall, Inc., 1993.

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

[9] P. E. HART, N. J. NILSSON, AND B. RAPHAEL, A formal basis for the heuristic determination of minimum cost paths, SIGART Bull., (1972), pp. 28–29.

[10] R. L. HAUPT AND S. E. HAUPT, Practical Genetic Algorithms, Wiley-Interscience, 2004.

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

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

[13] K. Y. LEE AND M. A. EL-SHARKAWI, Modern Heuristic Optimization Techniques, Mohamed E. El-Hawary, Series Editor, 2008.

[14] M. T. O. LING LIU, Encyclopedia of Database Systems, Springer, 2009.

[15] O. M. LIOR ROKACH, Data Mining with Decision Trees, World Scientific Publishing, 2008.

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

[17] J. MACQUEEN, Some methods for classification and analysis of multivariate observations, in Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability, 1967, pp. 1:281–297.

[18] J. MALUSZYNSKI, Logic, Programming and Prolog, John Wiley and Sons, 2000.

[19] T. M. MITCHELL, Machine Learning, McGraw-Hill, 1997.

[20] S. MUGGLETON, Inverse entailment and progol, New Generation Comput., (1995), pp. 245–286.

[21] S. MUGGLETON, Learning from positive data, in Inductive Logic Programming Workshop, 1996, pp. 358–376.

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

[23] J. R. QUINLAN, Learning logical definitions from relations, Machine Learning, 5 (1990), pp. 239–266.

[24] R. QUINLAN, Induction of decision trees, Kluwer Academic Publishers, (1986).

[25] E. F. REMCO R. BOUCKAERT, WEKA Manual for Version 3-6-0, University of Waikato, Hamilton, New Zealand, 2008.

[26] J. A. ROBINSON, A machine-oriented logic based on the resolution principle, Journal of the Association for Computer Machinery, Vol. 12, (1965), pp. 23–41.

[27] L. ROKACH AND O. MAIMON, Data mining and knowledge discovery handbook, Springer, 2005.

[28] K. H. ROSEN, Discrete Mathematics and its Applications, McGraw-Hill, 2007.

[29] S. J. RUSSELL AND P. NORVIG, Inteligencia Artificial, un enfoque moderno, Pearson, Prentice-Hall, 2004.

[30] N.-C. SHAN-HWEI AND R. DE WOLF, Foundations of Inductive Logic Programming, Springer, 1997.

[31] C. E. SHANNON, A mathematical theory of communication, Bell System Technical Journal, (1948), pp. 379–423 y 623–656.

[32] Y.-S. SHIH, QUEST User Manual, Department of Mathematics National Chung Cheng University, 2008.

[33] A. SRINIVASAN, The Aleph Manual, 2004. http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/.

[34] A. TURING, Computing machinery and intelligence, Mind, (1950), pp. 433–460.

[35] T. WEISE, Global Optimization Algorithms: Theory and Application, Thomas Weise, 2007.

[36] H. I. WITTEN AND E. FRANK, Data Mining, Practical Machine Learning Tools and Techniques, Morgan Kaufmann Publishers, 2005.