UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

DEPARTAMENTO DE LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

**Tesis Doctoral**

# NUEVOS ENFOQUES EN APRENDIZAJE INCREMENTAL

AUTOR:
JOSÉ DEL CAMPO ÁVILA

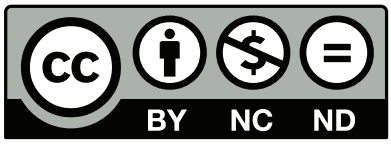
DIRECTORES:
DR. RAFAEL MORALES BUENO
DR. GONZALO RAMOS JIMÉNEZ

UNIVERSIDAD DE MÁLAGA

JUNIO, 2007

UNIVERSIDAD DE MÁLAGA
SPICUM
servicio de publicaciones

AUTOR: José del Campo Ávila
EDITA: Servicio de Publicaciones de la Universidad de Málaga



*Dedicado a mis padres,*
*a mi hermana y a Araceli*

# Agradecimientos

Muchas han sido las personas que han influido en el trabajo que se presenta en esta tesis. No podría enumerarlas por ser tantas y tan diversas las formas en las que han contribuido. A todas ellas mi agradecimiento. Sin embargo, me gustaría mostrar mi gratitud a ciertas personas en particular.

En primer lugar, mi más sincero agradecimiento a mis directores de tesis: Rafael Morales y Gonzalo Ramos. No sólo por su inestimable ayuda, sin la que no hubiese podido completar esta tesis, sino también por su constante cercanía y sus acertados consejos. Gracias por ayudarme a iniciar mi carrera investigadora.

Diversas han sido las Universidades y Centros de Investigación por los que he pasado durante estos últimos años y en los que he encontrado grandes investigadores y mejores compañeros. Mi gratitud a las distintas personas con las que he coincidido en las Universidades de Valladolid, Zaragoza o en la Politécnica de Cataluña. No puedo olvidar los tres estupendos meses que pasé en Oporto compartiendo trabajo y ocio con los miembros del LIACC, y debo agradecérselo a todos ellos.

Si mi paso por otras instituciones ha sido provechoso, el conocimiento que he podido adquirir gracias a los integrantes del grupo de investigación al que pertenezco lo ha sido aún más. Muchas gracias a profesores y compañeros por haber compartido conmigo tantas experiencias y buenos ratos.

Debo reconocer el apoyo económico recibido por parte del Ministerio de Educación y Ciencia, al concederme la posibilidad de disfrutar de una beca FPI con la que he podido desarrollar mi actividad investigadora.

Por último, en el plano personal, quiero agradecerle a mis amigos y a mi familia, todo el apoyo que me han dado. En especial a mis padres, a mi hermana Mercedes y a Araceli, porque su paciencia y el continuo ánimo que me han infundido han sido las herramientas más útiles que hubiese podido encontrar.

# Índice general

# Introducción

# Introducción

El aprendizaje incremental será el tema principal de esta tesis y definirá el ámbito en el que hemos desarrollado las aportaciones que presentamos. En este capítulo introductorio, describimos los conceptos necesarios para contextualizar nuestro trabajo y advertimos la necesidad de incorporar nuevos métodos y enfoques que permitan avanzar en el área del aprendizaje automático. En primer lugar definimos el concepto de aprendizaje desde una perspectiva computacional y a continuación mostramos las características propias del aprendizaje incremental. Para terminar enumeramos las aportaciones que se proponen en esta tesis y describimos la estructura planteada para presentarlas.

## 0.1. Concepto de aprendizaje

Las *ciencias cognitivas* tienen un carácter interdisciplinar y se dedican al estudio de la cognición [Rap-2000], que es la acción o efecto de *conocer* [RAE-2001]. Dependiendo de la consideración que se le quiera dar a dicho estudio, éste se puede ver restringido al ámbito humano o puede extenderse más allá y alcanzar a cualquier sistema (sea natural o artificial) [Nor-1981]. En nuestro caso consideraremos la acepción más amplia, puesto que nuestro objeto de estudio no es la mente humana, sino la forma en que determinados sistemas artificiales pueden adquirir conocimiento. Existen múltiples aspectos que son estudiados por las ciencias cognitivas, siendo también múltiples las disciplinas que participan de dicho estudio: Psicología, Filosofía, Biología, Informática, etc.

El nexo común que existe entre las ciencias cognitivas y la Informática está definido por el área de la *Inteligencia Artificial*. Este área es muy extensa y comprende, entre otros objetos de estudio, al *aprendizaje automático*, que será nuestro marco de trabajo. A continuación daremos una descripción de tres conceptos muy relacionados con la adquisición de conocimiento y cuyo significado puede confundirse dependiendo el ámbito en que nos encontremos: *extracción de conocimiento*, *minería de datos* y *aprendizaje automático*.

Estos conceptos cobran cada vez más importancia y deben enfrentarse continuamente con nuevas dificultades debido al creciente ritmo de adquisición de información. En un principio, los datos constituían en sí mismo un producto que se obtenía de distintos procesos (industriales, médicos, etc.), pero han pasado a convertirse en una materia prima, en una fuente de conocimiento. Distintos agentes que actúan en la sociedad (empresas, científicos, etc.) han tomado conciencia de que, en los datos con los que trabajan, pueden encontrar conocimiento que antes no habían considerado y que puede resultarles muy útil para incrementar su rendimiento.

### 0.1.1. Extracción de conocimiento

La *extracción de conocimiento* es el proceso por el cual se identifican, de forma no trivial, patrones válidos, novedosos, potencialmente útiles y comprensibles que se encuentren en los datos. Esta definición es la que propusieron Fayyad, Piatetsky-Shapiro y Smith en 1996 [Fay-1996] y es una de las más extendidas porque recoge las principales características que definen al proceso de extracción de conocimiento. Su nombre en inglés es "*Knowledge Discovery in Databases*" y las siglas por las que se le puede identificar normalmente en la literatura es *KDD*. Aunque este nombre haga referencia a grandes bases de datos, su ámbito no se restringe únicamente a ellas pudiéndose considerar cualquier fuente de datos, ya esté contenida

en una base de datos o en un fichero, ya sea grande o pequeña, ya sea estática o dinámica ("*flujo de datos*").

El concepto de extracción de conocimiento (o descubrimiento) suele confundirse con el de *minería de datos* dependiendo del ámbito en que nos encontremos, pero no se refieren a lo mismo puesto que éste último es, en realidad, parte del primero. Si queremos hablar del concepto de extracción de conocimiento en términos de minería de datos debemos usar el término de *proceso de minería de datos*. Esta última terminología está más extendida en el ámbito industrial, mientras que el concepto de extracción de conocimiento se utiliza más habitualmente en los entornos académicos [Han-2000], pero ambos describen lo mismo: el proceso que hay que seguir para conseguir conocimiento útil, novedoso, válido y comprensible.

En el ámbito industrial tenemos distintos ejemplos de metodologías que han sido diseñadas para realizar la extracción de conocimiento (concepto que ellos denominan como proceso de minería de datos). Así, tenemos desde enfoques más específicos diseñados por empresas individualmente, como puede ser la metodología SEMMA [Cer-2006], hasta enfoques más consensuados entre distintas empresas, como puede ser la metodología CRISP-DM [Cha-1999]. A pesar de las diferencias que puedan existir, tanto en el ámbito académico como en el industrial, las fases son similares y pueden diferenciarse tres grandes etapas [Fre-2002]: el tratamiento previo antes de intentar extraer modelos de los datos (pre-procesamiento), la propia extracción de modelos y la fase posterior a dicha extracción (post-procesamiento). En la figura 1 mostramos el esquema básico con las tres etapas más relevantes y en la figura 2 presentamos dos enfoques más detallados: uno académico [Her-2004] y otro industrial [Cha-1999]. Se puede observar cómo son similares, haciendo que los términos "extracción de conocimiento" y "proceso de minería de datos" se refieran al mismo concepto, pero destacando que la "minería de datos" propiamente (o "modelado") aparece como una etapa concreta y cuyo concepto trataremos en el siguiente apartado.

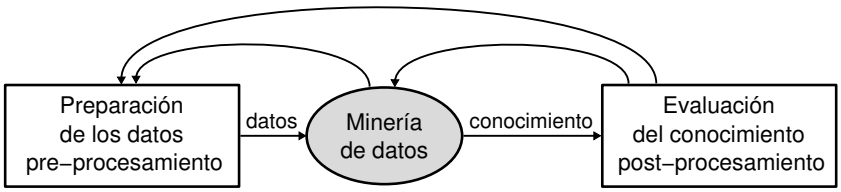



Figura 1: Esquema general de las etapas del proceso de extracción de conocimiento.

Lo que es estrictamente compartido por todas las metodologías es que este proceso es *iterativo* e *interactivo*. Es iterativo porque la salida de una etapa no tiene que ir siempre a la siguiente etapa, sino que puede realimentar alguna etapa anterior para modificarla. La característica de ser iterativo se aprecia aún más en el modelo CRISP-DM, donde es todo el proceso el que se repetirá iterativamente mejorando progresivamente los resultados alcanzados. La forma de pasar de unas etapas a otras no está retringida únicamente a las indicadas en las figuras, las transiciones que están presentes son las más habituales pero cualquier otra también sería factible. Se dice que el proceso de extracción de conocimiento también es interactivo porque el usuario es quien decide cómo actuar en cada una de las etapas. Dicha intervención es necesaria para evaluar la validez de los modelos generados, para estudiar su utilidad y para orientar futuras iteraciones hacia los objetivos deseados. Esta es la razón por la que se dice que el proceso no es trivial: necesita de la participación de un experto (o equipo de expertos) que controlen en todo momento el proceso para que sea posible alcanzar resultados de calidad.

### 0.1.2. Minería de datos

La *minería de datos* es el proceso de descubrimiento de patrones a partir de los datos [Wit-2005]. Ese descubrimiento (extracción) de patrones (modelos) debe ser lo más automático posible porque el volumen de datos que es usado en el proceso es potencialmente elevado. Ahí reside el interés de este proceso: teniendo tan grandes cantidades de información, a los humanos nos resultaría imposible trabajar con toda ella para extraer alguna conclusión. Además, somos conscientes del aumento progresivo que se está observando en el

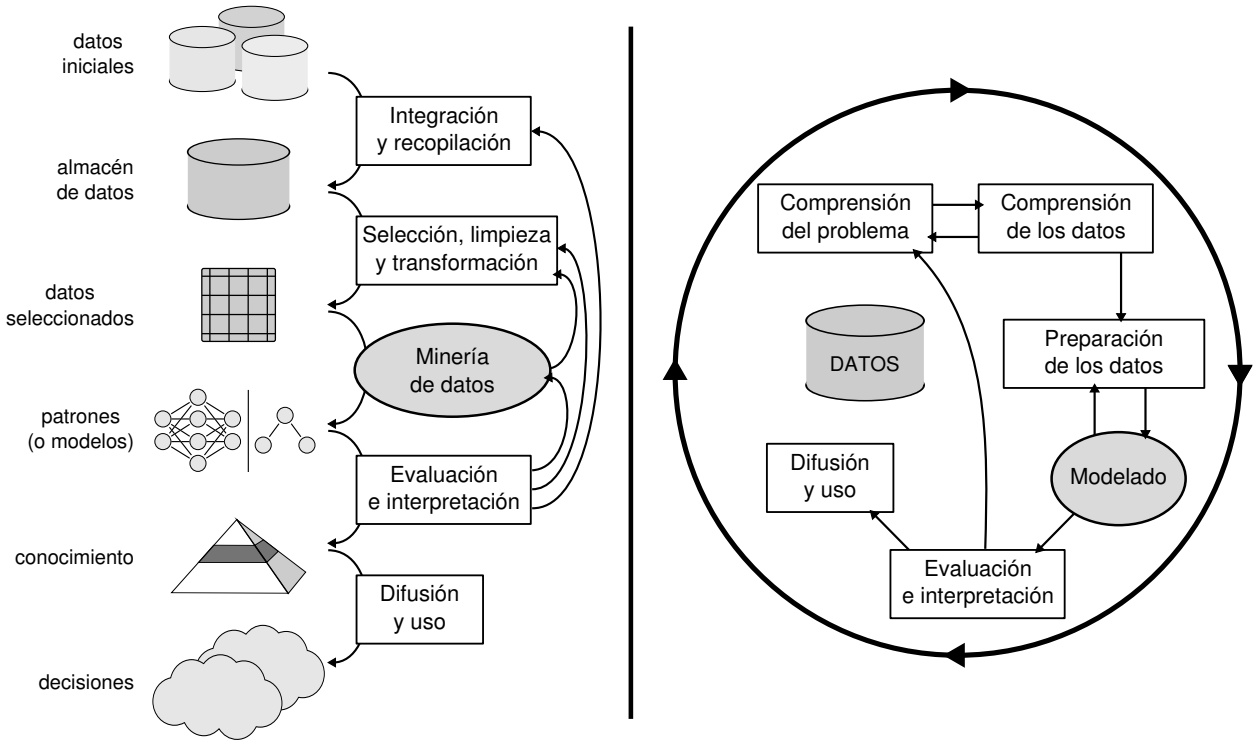



Figura 2: Etapas del proceso de extracción de conocimiento. *Izquierda*: esquema académico según Hernández y otros [Her-2004]. *Derecha*: esquema industrial CRISP-DM [Cha-1999].

volumen de datos (información) que utilizan las organizaciones (públicas o privadas) debido a la reducción del coste que supone almacenarlos. La minería de datos se convierte así, cada vez en mayor medida, en una herramienta de gran utilidad para extraer patrones que serían inalcanzables de otra forma. Su nombre en inglés es "*Data Mining*" y las siglas por las que se le puede identificar normalmente en la literatura es *DM*.

El objetivo de la minería de datos es inferir conocimiento, que suele expresarse en forma de patrones (o modelos), que sea útil para alcanzar alguna mejora (comprensión de nuevos conceptos, aumento de precisión en la predicción, etc.). Se sitúa como etapa clave en el proceso de extracción de conocimiento y es la responsable de inferir el conocimiento que se espera obtener al final de dicho proceso. La entrada a esta etapa, como se puede observar en las figuras 1 ó 2, son datos que han sido previamente procesados para adecuarlos a la técnica concreta que se vaya a utilizar. Posteriormente, los patrones descubiertos serán evaluados para estudiar su calidad y es posible que se vuelva a iterar la misma etapa de minería de datos, variando las técnicas utilizadas o modificando los datos usados como entrada.

Existen multitud de disciplinas que toman parte en la minería de datos: aprendizaje automático, estadística, computación paralela, recuperación de información, procesamiento de imágenes, procesamiento de señales, etc. Pero, para realizar su tarea, la minería de datos suele aplicar, al menos, un *análisis inteligente de datos* [Ber-2003] (término que a veces se usa como sinónimo). Dos disciplinas destacan a la hora de definir las técnicas de análisis inteligente de datos: la *estadística* y el *aprendizaje automático*. Definir una división clara entre ambas disciplinas es algo complejo, puesto que ambas están relacionadas. La estadística, usada como técnica de análisis de datos, está más relacionada con la comprobación de hipótesis, mientras que el aprendizaje automático está más enfocado a buscar las mejores hipótesis de una forma automática. No obstante, la estadística no sólo sirve para comprobar hipótesis, sino que también es incorporada frecuentemente por las técnicas de aprendizaje automático para controlar su funcionamiento. Como ejemplo de esta interrelación podemos ver que los modelos basados en árboles de decisión (que expondremos en el siguiente apartado) han sido inducidos, siguiendo esquemas similares, desde las dos perspectivas: la estadística (en el trabajo de Breiman, Friedman, Olshen y Stone [Bre-1984]) y la de aprendizaje automático (en el trabajo de Quinlan [Qui-1986]).

Las tareas que determinarán el tipo de conocimiento que se desea extraer y que se verá reflejado en la inducción de un tipo de modelo u otro, se pueden clasificar en dos grandes grupos [Her-2004]: tareas *descriptivas* y tareas *predictivas*. Las tareas descriptivas están más enfocadas a explicar o resumir los datos, es decir, a identificar propiedades que se puedan extraer de ellos. Entre ellas podemos citar las tareas de agrupamiento o de correlación. Por su parte, las tareas predictivas están dirigidas a estimar valores futuros o desconocidos de variables relevantes y entre ellas, como representantes destacadas, están las tareas de clasificación y regresión. Para la consecución de todas estas tareas existen multitud de técnicas en el ámbito del aprendizaje automático.

### 0.1.3. Aprendizaje automático

El *aprendizaje automático* es la disciplina encargada del estudio de los métodos usados para programar los ordenadores de forma que aprendan [Die-2003]. Su nombre en inglés es "*Machine Learning*" y las siglas por las que se le puede identificar normalmente en la literatura es *ML*. Existe gran variedad de definiciones para el término aprendizaje, dependiendo del ámbito donde se aplique y del uso que se le vaya a dar (Psicología, Filosofía, Inteligencia Artificial, etc.). No entraremos a discutir las diferentes acepciones que se le pueden dar y nos centraremos en la que tiene desde la perspectiva de la minería de datos: el *aprendizaje* nos permite identificar regularidades en un conjunto de experiencias (datos seleccionados) [Her-2004]. Dichas regularidades pueden expresarse en forma de patrones (o modelos) que describan diversos conocimientos que subyacen en los datos.

El uso de los patrones para describir el conocimiento puede interpretarse como una forma de *compresión* y puede ser utilizado con dicha finalidad, puesto que se consigue representar el conjunto de datos (o la mayor parte de estos) empleando mucho menos espacio. Se pasa de la necesidad de disponer del espacio que ocupa el conjunto de datos, a únicamente requerir el espacio que es necesario para representar su modelo (donde se comprimen dichos datos). Aunque no profundizaremos en esta aplicación del aprendizaje automático, debemos resaltar que existen múltiples trabajos que relacionan la compresión y el aprendizaje automático [Lop-2005; May-2007], y que las conexiones teóricas y empíricas que existen entre ellos pueden ayudar a conseguir avances en ambos campos [Scu-2006].

Una consideración que hay que tener presente cuando se habla de aprendizaje automático es la *complejidad* de los propios algoritmos. Cualquier técnica que se use estará formalizada, en último término, en forma de algoritmo y, como tal, su ejecución llevará asociada una complejidad (temporal y espacial). Puesto que los recursos siempre serán limitados (aunque continuamente estén creciendo), las técnicas de aprendizaje automático deberán extraer conceptos en un número de pasos razonable [Val-1984] y la calidad del modelo que infieran dependerá, en gran medida, de los recursos disponibles. Con la intención de estudiar las limitaciones que puedan surgir en función de los modelos que se intenten inferir, de los conceptos que puedan ser aprendidos o de las técnicas utilizadas, surge una importante línea de investigación: la "Teoría del aprendizaje computacional" [Val-1984; Kea-1994; Ant-1997]. Debemos señalar que el término *aprendizaje computacional* es sinónimo de *aprendizaje automático*, por lo que la anterior línea de investigación se sigue refiriendo al aprendizaje automático, aunque lo denomine como computacional.

Dependiendo de la tarea que se quiera realizar, descriptiva o predictiva, y del tipo de datos de que se disponga, las técnicas más apropiadas para inferir dichos patrones serán unas u otras. Las técnicas utilizadas en aprendizaje automático han sido aprovechadas en multitud de ámbitos reales, lo que hace de esta disciplina una herramienta claramente útil. En el libro "Machine Learning and Data Mining: Methods and Applications" [Mic-1998] se pueden encontrar ejemplos en diferentes ámbitos entre los que podemos citar aplicaciones en el área de diseño [Man-1998], en el área de control [Bra-1998], o en el área de medicina [Kon-1998].

Como comentamos en el apartado anterior, las tareas a abordar desde el aprendizaje automático se pueden dividir en dos grandes grupos: descriptivas y predictivas. Entre las primeras están las tareas de detección de agrupamientos y correlaciones, y entre las segundas están las tareas de clasificación y regresión. Existen distintas técnicas para abordar las tareas citadas y, aunque algunas están más enfocadas para resolver una tarea u otra (por ejemplo, las que generan reglas de asociación [Man-1994; Agr-1996] son más des-

criptivas y las que entrenan redes neuronales [Wid-1960; Rum-1986] son más predictivas), la separación no siempre es tan clara. Como ejemplo podemos poner a los algoritmos que inducen árboles de decisión [Bre-1984; Qui-1986]. Estos modelos pueden ser fácilmente comprensibles y, por tanto, pueden ofrecer descripciones sobre distintos comportamientos en forma de reglas; y también tiene capacidad para ser utilizado como un sistema predictor. Existen multitud de técnicas empleadas para inferir gran diversidad de modelos y éstas pueden usarse de forma conjunta o aislada. En esta tesis nos centraremos en las tareas de clasificación y predicción mediante la inducción de árboles de decisión.

**Clasificación mediante la inducción de árboles de decisión**

El objetivo de la tarea de *clasificación* es la construcción de modelos concisos que representen la distribución del atributo dependiente (clase) en función de los atributos predictores (atributos) [Ye-2003]. El modelo resultante se usará, principalmente, para determinar la clase a la que pertenecen las observaciones de las que se conocen todos los valores de sus atributos a excepción del valor de clase, es decir, su tarea será preferentemente predictiva. Dependiendo del modelo generado, éste además puede presentar características descriptivas, lo que lo hará aún más deseable desde el punto de vista de la extracción de conocimiento. Según Michie, Spiegelhalter y Taylor existen tres enfoques tradicionales usados en la tarea de clasificación [Mic-1994]: aproximación estadística, aprendizaje automático y redes neuronales. Posteriormente, Kononenko, Bratko y Kukar matizaron el enfoque de aprendizaje automático y lo relajaron sustituyéndolo por aprendizaje inductivo [Kon-1998]. De esta forma no entran en conflicto el aprendizaje automático, que es un concepto más general, con el resto de conceptos, que son más particulares y pueden considerarse como pertenecientes al más general.

El *aprendizaje inductivo* es aquel aprendizaje que parte de casos particulares (experiencias) y obtiene casos generales (modelos o reglas) [Her-2004] y es el tipo de aprendizaje más comúnmente usado en el ámbito del aprendizaje automático. Se contrapone al *aprendizaje deductivo* en el cuál el experto "deduce" que una regla (modelo o hipótesis) puede servir para describir el conocimiento y lo comprueba consultando en el conjunto de datos. Aquí se puede apreciar cómo el aprendizaje deductivo debe hacerse manualmente y el aprendizaje inductivo tiene la ventaja de poder automatizarse. De los tres enfoques más utilizados para la tarea de clasificación, el más apropiado para construir el modelo de árbol de decisión es el aprendizaje inductivo [Hun-1993].

Un *árbol de decisión* es un árbol (grafo dirigido acíclico donde todos los nodos salvo la raíz tienen un solo padre y pueden tener cero, uno o más hijos) etiquetado [Ram-2001]. Los nodos que no tienen hijos se llaman hojas. Las hojas se etiquetan con alguna de las clases definidas. Los nodos que no son hojas se etiquetan con un atributo y cada uno de los arcos que parten de ellos hacia sus hijos se etiquetan con cada uno de los valores del atributo correspondiente. Las ramas son los caminos que van desde la raíz hasta una hoja.

La inducción de árboles de decisión presenta una serie de características que la hacen bastante interesante en el ámbito de la minería de datos:

- Representación intuitiva: permite que el modelo sea comprensible por expertos [Bre-1984; Ye-2003].
- Método no paramétrico: apropiado para descubrimiento de conocimiento puesto que los árboles de decisión pueden modelar un amplio rango de distribuciones [Mur-1998; Ye-2003].
- Construcción rápida: su inducción puede concluir bastante antes que la de otros modelos [Lim-2000].
- Precisión adecuada: comparable a la de otros modelos de clasificación [Han-1997; Lim-2000].
- Válido ante problemas con alta dimensionalidad: la presencia de atributos irrelevantes no afecta considerablemente a la inducción del modelo [Her-2004].
- Válido ante problemas con atributos categóricos y continuos: puede trabajar con conjuntos de datos que presenten ambos tipos de atributos [Her-2004].

Los algoritmos para inducir árboles de decisión fueron inicialmente propuestos por Hunt, Marin y Stone [Hun-1966] y, posteriormente, empezaron a tomar relevancia a partir de los trabajos de Quinlan [Qui-1979; Qui-1983] (desde la perspectiva del aprendizaje inductivo) y del trabajo de Breiman, Friedman, Olshen y Stone [Bre-1984] (desde una perspectiva estadística). Las bases de la inducción de árboles de decisión quedó entonces bastante bien definida y daría como resultado una línea de investigación que se conoce por sus siglas en inglés con el nombre *TDIDT* ("*Top Down Induction of Decision Trees*"). El procedimiento básico para inducir un árbol de decisión es expandir recursivamente los nodos, hasta que ninguno sea expansible. Surgen así dos aspectos fundamentales a la hora de definir cualquier algoritmo TDIDT:

- Problema de la elección de atributo para expandir: hay que decidir el criterio que definirá la forma de elegir qué atributo debe ser usado para expandir un nodo en concreto (atributo que servirá para etiquetar dicho nodo). El problema no es trivial puesto que inducir el árbol óptimo (entendido como el árbol más pequeño que sea capaz de clasificar correctamente cualquier experiencia) es un problema de tipo NP-completo [Hya-1971; Nau-1991]. La solución adoptada es utilizar heurísticos que, aunque no aseguren alcanzar el árbol óptimo, permitan inducir modelos próximos a él. Estos heurísticos, basándose en el concepto de medida de desorden (como la propuesta por Shannon [Sha-1948]), miden cuánto desorden se elimina al elegir uno u otro atributo y eligen aquél que consiga dejar el árbol lo más ordenado posible.

- Problema de la poda: hay que determinar qué condición debe cumplirse para detener la expansión por una rama concreta. Existen aproximaciones diferentes según se haga una prepoda (el árbol no llega a expandirse completamente) o una postpoda (el árbol se expande completamente y posteriormente se poda), y hay numerosos trabajos que abordan este aspecto [Min-1989; Esp-1997; Her-2004].

Existen otros problemas que afectan a la inducción de árboles de decisión, como la aparición de ruido en los conjuntos de datos [Kod-1987], la consideración de conocimiento de base para evaluar los costes de la clasificación [Nun-1991] o la presencia de atributos con valores desconocidos [For-2006], pero no son exclusivos de esta técnica y afectan también a otros métodos.

Puede encontrarse un estudio bastante completo acerca de los métodos clásicos para la inducción de árboles de decisión en los trabajos de Murthy [Mur-1998] o de Rokach y Maimon [Rok-2005].

## 0.2. Aprendizaje incremental

El proceso de extracción de conocimiento puede aplicarse sobre diversas fuentes de datos, pero, cuantos más datos estén disponibles, más dificultades suelen aparecer en la aplicación de algoritmos de aprendizaje automático. Además, cada vez son más las situaciones en las que se dispone de un gran volumen de información de la que intentar extraer conocimiento. Si se intenta usar un enfoque clásico para extraer conocimiento (como pueda ser el que realizan los algoritmos ID3 [Qui-1986] o C4.5 [Qui-1993]), es necesario que los datos cumplan una serie de condiciones [Fer-2005]: deben estar disponibles todas las experiencias (datos) que van a ser usadas por el algoritmo y todas estas experiencias deben ser almacenadas en la memoria principal del ordenador (RAM). Como resultado de la aplicación de un algoritmo de aprendizaje automático a un conjunto de datos con las características anteriores, se obtendrá un modelo y el proceso de aprendizaje se dará por finalizado.

La primera limitación que puede surgir es la de que todas las experiencias deben estar disponibles en la memoria principal para que puedan ser utilizadas por el algoritmo de aprendizaje. Para superar dicha limitación se propusieron determinados enfoques que no precisaban tener las experiencias en memoria principal, sino que éstas podían estar en memoria secundaria (dispositivo de almacenamiento, típicamente un disco duro). Entre ellos podemos destacar los algoritmos SLIQ [Meh-1996] o SPRINT [Sha-1996]. Pero, aunque estos algoritmos basados en disco hacen un uso mucho más eficiente de la memoria, siguen presentando limitaciones. A la que hemos presentado anteriormente que exige disponer de todas las experiencias antes de iniciar la inducción de los modelos, hay que añadir la posibilidad de que el conjunto de datos tampoco quepa

en ningún dispositivo de almacenamiento. Para resolver estas y otras limitaciones surgen los algoritmos de aprendizaje incremental.

El *aprendizaje incremental* se caracteriza principalmente por dos aspectos: es un aprendizaje capaz de incorporar la información que aporten nuevas experiencias (que antes no estaban disponibles en el conjunto de datos) al modelo que se está induciendo [Sch-1986] y capaz de hacerlo evolucionar para que cada vez represente conceptos más complejos [Qui-1993]. Esta definición concuerda con el concepto propuesto por Giraud [Gir-2000] y por Ferrer y Aguilar [Fer-2005] en el que el modelo inducido por el algoritmo en cada momento es obtenido mediante la utilización de nuevas experiencias y el modelo anterior. Surge así el concepto de aprendizaje cuyo modelo está disponible en cualquier momento (en inglés "*any-time learning*"). Aunque el proceso de aprendizaje no haya concluido, como éste usa el modelo anterior, siempre habrá un modelo disponible que podrá ser usado para clasificar o para visualizar el conocimiento extraído hasta ese momento.

El concepto de aprendizaje incremental (conocido a veces como aprendizaje *en línea* o aprendizaje *secuencial*, aunque con diversos matices) ha recibido un tratamiento bastante diferente, pasando de acepciones muy relajadas (como la propuesta por Michalski [Mic-1985]) a otras excesivamente restrictivas (como la que presenta Polikar [Pol-2001]). Michalski propone dos aproximaciones para abordar el problema, una *evolucionaria*, que sería la que concuerda con la definición que hemos dado, y otra *revolucionaria*, que entraría en conflicto con dicha definición. El enfoque revolucionario (también llamado *temporal por lotes* por Maloof y Michalski [Mal-2000]) se basa en la destrucción del modelo inducido cada vez que llega un lote con nuevas experiencias y la re-inducción de un nuevo modelo usando esas últimas experiencias junto con todas las previas. De esta forma no hay una evolución incremental del modelo en el que se acomoda la información de las nuevas experiencias. Por otro lado, Polikar considera, como condición necesaria, la capacidad de considerar nuevas clases que pudiesen aparecer en nuevas experiencias. Sin duda, esa característica es deseable, pero no consideramos que sea imprescindible para hablar de aprendizaje incremental. De igual forma tampoco consideramos como un requisito indispensable la capacidad de tratar conjuntos de datos en los que pueda aparecer cambio de concepto, como sí se sugiere en otros trabajos [Gir-2000; Fer-2005]. Por supuesto, disponer de esa capacidad aumenta considerablemente el tipo de problemas que pueden ser abordados por el algoritmo y es una característica deseable para realizar aprendizaje incremental, pero no consideramos que sea esencial según la definición dada.

La *representación probabilística* [Smi-1981] proporciona la oportunidad de explorar los datos, manteniendo la información más relevante extraída de las experiencias [Fis-1998]. De esta forma, los requisitos de memoria no dependen del número de experiencias en el conjunto de datos, sino de la estructura que se está induciendo y de los contadores que lleva asociados. Algunos algoritmos usan este enfoque de una u otra forma: ID4 [Sch-1986], ID5 [Utg-1988], ITI [Utg-1997] o VFDT [Dom-2000]. Aún así, el beneficio que se pueda alcanzar usando dicha representación probabilística se encuentra limitado según el tipo de *modelo de memoria de experiencias* que se use [Rei-1988; Mal-2000]. Existen tres tipos de modelo de memoria de experiencias: la que almacena todas las experiencias (total), la que conserva únicamente las experiencias que necesite el algoritmo (parcial) y la que olvida todas las experiencias conforme son procesadas (ninguna). Como ejemplos de algoritmos que usan un modelo de memoria que almacena todas las experiencias tenemos ID5 o ITI. Al guardar todas las experiencias, estos algoritmos ofrecen la característica adicional de asegurar que el modelo que induzcan será idéntico al que se obtendría si se tomasen todas las experiencias de una vez, pero tiene el inconveniente de necesitar bastante memoria para almacenar esas experiencias. En el otro extremo se encuentran los algoritmos ID4, VFDT o IADEM-0 [Ram-2004] que no utilizan la memoria de experiencias. Como algoritmos que usan modelos de memoria parcial podemos citar el algoritmo AQ-PM [Mal-2000], los algoritmos FLORA [Wid-1996] o el algoritmo Online Tree [Nun-2005], que incluyen, adicionalmente, mecanismos de *olvido* para almacenar sólo las experiencias relevantes en cada momento. Considerando las limitaciones que ofrece un algoritmo de aprendizaje incremental que utilice un modelo de memoria total, debemos concluir que las opciones factibles para realizar un aprendizaje de estas características deben usar modelos de memoria parcial con olvido o no usar ningún modelo de memoria de experiencias, es decir, olvidarlas todas según son usadas para modificar el modelo.

Al igual que ocurría a la hora de diseñar algoritmos de aprendizaje tradicional, con el aprendizaje incremental también se diseñan técnicas que inducen multitud de modelos diferentes que representen el conocimiento. Añadir la característica incremental al aprendizaje no limita la diversidad de modelos que pueden ser inducidos, lo único que varía serán las técnicas para hacerlo. Así, tenemos algoritmos que inducen redes neuronales (Learn++ [Pol-2001]), árboles de decisión (VFDT [Dom-2000]), reglas de decisión (FACIL [Fer-2005a]), etc. En la figura 3 hemos presentado una clasificación de los algoritmos más comunes en el campo del aprendizaje incremental atendiendo al tipo de memoria de experiencias que usan, al enfoque que utilizan para evolucionar (o revolucionar) los modelos, si es que representan el conocimiento usando alguno. En el caso de los algoritmos IB1, IB2 o IB3 [Aha-1991] no se usa ningún modelo para representar el conocimiento, puesto que usan el vecino más cercano y, por tanto, el modelo lo compondrían las propias experiencias almacenadas. En la figura se puede apreciar que los algoritmos ID3 y C4.5 aparecen repetidos en dos ramas. La razón es que, como propone Quinlan [Qui-1993], se pueden utilizar dos enfoques diferentes para el aprendizaje temporal por lotes (o revolucionario): reconstruir el modelo si se detecta algún error en las nuevas experiencias (memoria total) o dejarlo fijo aunque lleguen nuevas experiencias que sean incorrectamente clasificadas (sin memoria).

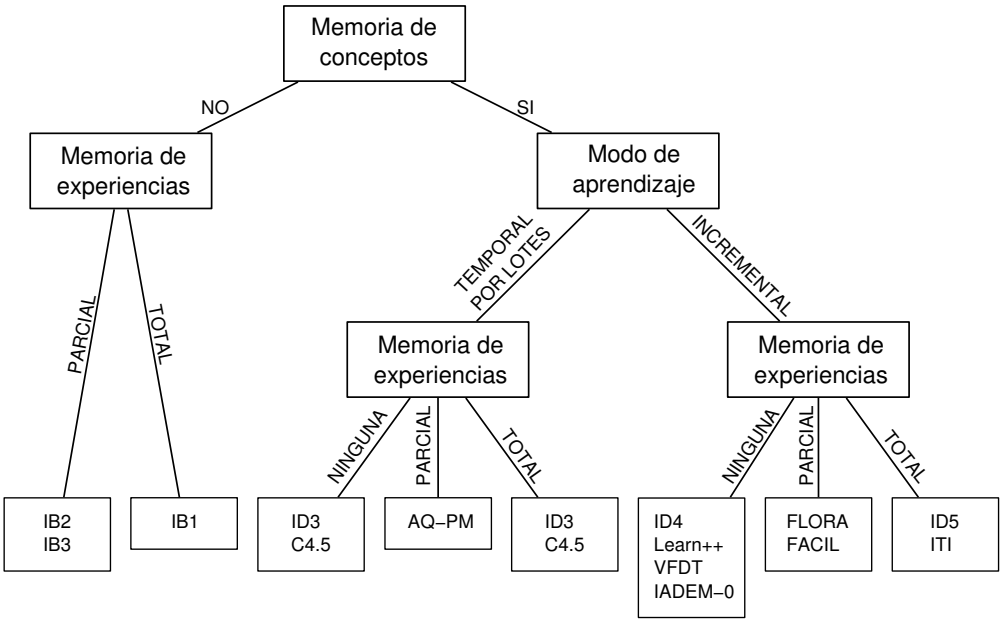



Figura 3: Clasificación de sistemas de aprendizaje incremental.

Como es de esperar, las estrategias para abordar las problemáticas que surgen al añadir la característica incremental a la tarea de aprendizaje automático, son múltiples. En esta tesis nos centraremos en dos de ellas, dedicando una parte a cada una. Utilizaremos técnicas basadas en muestreo secuencial [Dom-1999] y basadas en sistemas multiclasificadores [Str-2001].

## 0.3. Aportaciones y estructura de la tesis

La tesis ha sido dividida en cuatro partes, además de esta introducción. En las dos primeras partes se realiza la presentación de las aportaciones que proponemos (incluyendo los estudios experimentales), en la tercera parte se describen las conclusiones y en la última parte se encuentran los apéndices, un índice alfabético y la bibliografía. A continuación enumeramos el contenido de los diferentes capítulos, agrupándolos por la parte en la que están incluidos, e indicamos las aportaciones que proponemos:

- **Parte I: Algoritmos Incrementales basados en Cotas de Concentración**

  La primera parte de la tesis está dedicada a los algoritmos de aprendizaje incremental basados en cotas de concentración. En el capítulo 1 presentamos el concepto de cota de concentración, describimos los usos que se les pueden dar y, como punto destacado, detallamos un procedimiento para combinar dos de las principales cotas de concentración utilizadas en el campo del aprendizaje automático (cotas de Chernoff [Che-1952] y Hoeffding [Hoe-1963]). Gracias a este uso combinado se pueden estimar cotas más ajustadas que serán de gran utilidad en los desarrollos que proponemos.

  A continuación (capítulo 2), describimos formalmente el algoritmo IADEM-0. Este algoritmo, que fue propuesto por Ramos y Morales [Ram-2000; Ram-2001] y que induce árboles de decisión para representar el conocimiento que extrae, será la base de todas las aportaciones que realicemos en esta parte de la tesis. Como primeras aportaciones, aunque no incluyan ninguna funcionalidad adicional, tenemos que: se ha sustituido la definición original del algoritmo por una formalización mucho más general [Cam-2002] que permite abstraer su funcionamiento, ofreciendo así total libertad a la hora de implementar cualquier versión del mismo; se ha mejorado el algoritmo en algunos aspectos al considerar casos extremos previamente no contemplados y al ajustar los cálculos que se realizan; se ha realizado un extenso estudio de la influencia de los parámetros internos que lo configuran para explicar su comportamiento y para establecer unos valores por defecto válidos en la mayoría de situaciones; por último, se ha introducido un análisis básico sobre la complejidad temporal y espacial del algoritmo.

  En el capítulo 3 se describen todas las aportaciones realizadas para mejorar las características del algoritmo IADEM-0, lo que dará lugar al algoritmo que proponemos y al que llamamos IADEM-2 [Cam-2007]. Las características añadidas han sido clasificadas según el ámbito en el que se produce la mejora: convergencia del algoritmo, calidad del modelo inducido o precisión alcanzada. De esta forma, un nuevo mecanismo para detectar la estabilización o empeoramiento del aprendizaje supone, junto con otras modificaciones, una mejora en la convergencia del algoritmo; la posibilidad de trabajar con atributos continuos de forma directa (sin discretización previa) y la reformulación del procedimiento que detecta la idoneidad del mejor atributo para expandir el árbol de decisión se encuadran dentro de las aportaciones encaminadas a mejorar la calidad del modelo inducido; por último, la utilización de hojas funcionales supone un incremento de la precisión alcanzada cuando se utiliza el modelo generado como sistema de predicción. Cada una de estas extensiones se presenta de forma rigurosa, y es acompañada de un estudio de la influencia de sus parámetros internos y de la complejidad que añaden al algoritmo.

  Para comprobar las ventajas que aporta la utilización del algoritmo propuesto (IADEM-2) presentamos, en el capítulo 4, un detallado estudio experimental. En él se evalúa el comportamiento del algoritmo en distintas situaciones (variando el tamaño del conjunto de datos o el nivel de ruido que hay presente en él) y se compara con los resultados alcanzados por otros algoritmos. Se muestran las capacidades más importantes y se comprueba su buen funcionamiento en multitud de situaciones.

  Esta primera parte de la tesis termina con el capítulo 5 en el que se realiza un resumen de los conceptos presentados y de las aportaciones propuestas. También se enumeran las principales ventajas y limitaciones del algoritmo y se presentan las futuras líneas de investigación que se plantean a partir de los trabajos realizados.

- **Parte II: Algoritmos Incrementales basados en Sistemas Multiclasificadores**

  La segunda parte de la tesis está enfocada al aprendizaje incremental utilizando sistemas multiclasificadores. En el primer capítulo de esta parte (capítulo 6) se muestran las características más destacadas que ofrecen los sistemas multiclasificadores y se describe la facilidad con que estos sistemas pueden ser adaptados para realizar un aprendizaje incremental. En este mismo capítulo se describen los algoritmos básicos que serán usados como base de los que se proponen y se presenta un primer algoritmo incremental basado en sistemas multiclasificadores: MultiCIDIM-DS.

En el capítulo 7 se introduce un nuevo concepto que permite aumentar la calidad de la clasificación realizada por los sistemas multiclasificadores con un enfoque incremental. Hemos llamado a dicho concepto *filtro corrector* debido a que su función es filtrar el conjunto de clasificadores básicos que son combinados para realizar una predicción, utilizando únicamente los que dispongan de información relevante. Como resultado de la incorporación de este método al algoritmo MultiCIDIM-DS hemos creado un nuevo algoritmo: MultiCIDIM-DS-CFC.

Al igual que se ha realizado un estudio experimental en la primera parte de la tesis, en esta segunda parte también se ha incluido otro (capítulo 8) con el mismo objetivo: evaluar el comportamiento de los algoritmos propuestos y, sobre todo, comprobar la mejora introducida al usar el nuevo método definido. Además, se comparan los resultados con los obtenidos por otros algoritmos y se muestran las capacidades más importantes comprobando su buen funcionamiento en las situaciones para las que ha sido diseñado.

Esta segunda parte de la tesis termina con el capítulo 9 en el que se realiza un resumen de los conceptos presentados y de las aportaciones propuestas. También se enumeran las principales ventajas y limitaciones de los algoritmos y métodos propuestos y se presentan las futuras líneas de investigación que se plantean a partir de los trabajos realizados.

- **Parte III: Conclusiones**

  Esta tercera parte de la tesis está compuesta únicamente por el capítulo 10 en el que se recapitulan las conclusiones a las que se pueden llegar después de estudiar las aportaciones presentadas en las dos primeras partes. También incluye un apartado en el que se comentan las principales líneas de investigación que nos planteamos seguir en el futuro.

- **Parte IV: Apéndices, índice alfabético y bibliografía**

  Para terminar la tesis, en esta última parte se presentan dos apéndices. El primero (apéndice A) describe la formulación explícita de la cota de Chernoff utilizada en el capítulo 1 y el segundo (apéndice B) detalla la relación que existe entre las distintas publicaciones que hemos realizado y las aportaciones presentadas en esta tesis. Después de estos dos apéndices hemos incluido un índice alfabético, que ayude a localizar más cómodamente los conceptos más relevantes de la tesis, y la bibliografía utilizada.

Debemos señalar que los resúmenes de las dos primeras partes (capítulos 5 y 9), las conclusiones y futuras líneas de trabajo (capítulo 10) y el apéndice donde relacionamos las publicaciones que hemos realizado con las aportaciones presentadas en esta tesis (apéndice B), están escritos tanto en español como en inglés. Para diferenciarlos, hemos utilizado el estilo de fuente *cursiva* para el texto escrito en inglés.

# Parte I

# Algoritmos Incrementales basados en Cotas de Concentración

# Capítulo 1

# Cotas de Concentración

El objetivo principal de la estimación estadística [Sie-1988; Per-2001] es determinar el valor aproximado de una variable a partir de resultados muestrales. Para ello se dispone de multitud de técnicas (método de los mínimos cuadrados, de máxima verosimilitud, etc.), cada una de ellas dirigida a resolver problemas diferentes. Los estimadores de máxima verosimilitud tienen una característica fundamental que los hace muy apropiados para el aprendizaje incremental: el sesgo, en caso de existir, tiende a cero conforme aumenta el tamaño de la muestra, siendo además asintóticamente eficiente.

Considerando los estimadores de máxima verosimilitud, éstos se pueden dividir en dos grupos: los estimadores puntuales y los estimadores por intervalo. Los estimadores puntuales únicamente calculan un estadístico muestral (el más extendido es la *media*), que es el que se usa para estimar el verdadero valor de la variable que se estudia. Pero la correspondencia entre el valor estimado de la media ($\overline{X}$) y el real ($\overline{\mu}$) no siempre está garantizada. Para solucionar este problema se usan los estimadores por intervalo que, en vez de dar un valor puntual, calculan el intervalo en el que debe encontrarse el valor real con una determinada *confianza* $(1-\delta)$. Podemos hablar de dos tipos de intervalo:

- por aproximación absoluta: $\overline{X} - \varepsilon \leq \overline{\mu} \leq \overline{X} + \varepsilon$, donde $\varepsilon$ es el error absoluto cometido; y
- por aproximación relativa: $(1-\beta)\overline{X} \leq \overline{\mu} \leq (1+\beta)\overline{X}$, donde $\beta$ es el error relativo cometido.

Las *cotas de concentración* constituyen uno de los enfoques que existen para determinar dichos intervalos y han recibido gran atención [Dia-2001; Jan-2002; Chu-2006] debido a su extendida utilidad. Sea $X$ una variable aleatoria y sea $x$ un valor real, las probabilidades $Pr[X < x]$ y $Pr[X > x]$ son conocidas como las colas izquierda y derecha respectivamente. Las cotas de concentración ofrecen cotas superiores para las colas (izquierda y derecha) de la variable aleatoria $X - \mathrm{E}[X]$. En general, las cotas de concentración sirven para probar que la diferencia entre un valor estimado y el valor real de una variable se encuentra acotada en un intervalo.

Existe una gran variedad de cotas de concentración y, desde finales del siglo XIX y principios del siglo XX en que se propusieron las clásicas cotas de Chebyshev o Markov, han sido numerosos los matemáticos que han contribuido a aumentar el conocimiento y variedad de las cotas de concentración. De hecho, sigue siendo una tarea relevante como atestigua el continuo interés que existe por estudiar y establecer nuevas cotas [The-2007]. De entre la gran variedad de cotas existentes comentaremos a continuación las cotas de Chernoff [Che-1952] y Hoeffding [Hoe-1963] por ser las más relevantes en nuestro trabajo. Podemos adelantar que en el algoritmo que presentamos en esta parte de la tesis, y cuya descripción comienza en el capítulo 2, necesitamos estimar los valores para una serie de variables (que en general serán frecuencias relativas) y, por tanto, haremos uso de la estimación estadística considerando dichas cotas de concentración.

## 1.1. Cotas de concentración de Chernoff y Hoeffding

Las cotas propuestas por Chebyshev y Markov son bastante útiles para determinar otras cotas más ajustadas. Tal es el caso de la cota propuesta por Chernoff [Che-1952] en 1952. El objetivo de usar esta cota es

estimar la suma de variables aleatorias independientes y binarias. Así, la cota de Chernoff es una extensión de la cota de Markov en la que las cotas de las dos colas de la distribución serán funciones exponenciales negativas que dependen del número de observaciones registradas. Como se puede intuir, esta propiedad será muy adecuada para ser usada en algoritmos incrementales, puesto que la fiabilidad de la variable calculada será mayor cuanto mayor sea el número de experiencias observadas. Aunque parece [McD-1989] que las ideas previas a la cota de Chernoff y sus demostraciones ya estaban presentes en los trabajos de Bernstein [Ber-1924], esta cota se conoce como cota de Chernoff y se ha convertido en una herramienta ampliamente usada en distintas áreas de investigación en Informática. A partir de la formulación implícita dada por Chernoff en su trabajo [Che-1952], otros autores han reformulado su cota de forma explícita y la han hecho más fácil de interpretar [Oka-1958; Hag-1990; Mot-1995].

La formulación explícita más sencilla la hemos encontrado en el material usado por Zhang para impartir un curso en la Universidad de Berkeley [Zha-2002]. Un desarrollo que permite alcanzar dicha formulación puede encontrarse en el apéndice A. Seguidamente presentamos el teorema de la cota general de Chernoff y el corolario que usaremos para determinar el error de estimación absoluto.

TEOREMA 1 (COTA GENERAL DE CHERNOFF )
Sean $X_1, X_2, \ldots, X_n$ variables aleatorias independientes que se corresponden con experimentos de Bernouilli. Cada variable $X_i$ tiene una probabilidad de éxito igual a $p_i$ y una probabilidad de fracaso igual a $q_i = (1 - p_i)$. Sea $X_s = \Sigma_{i=1}^{n} X_i$ una variable aleatoria cuyo valor esperado es $\mu_s = \Sigma_{i=1}^{n} p_i$.
Entonces, para cualquier $\beta > 0$,

$$\Pr[X_s > (1+\beta)\mu_s] \leq \left( \frac{e^{\beta}}{(1+\beta)^{1+\beta}} \right)^{\mu_s} \tag{1.1}$$

y para el caso en que $0 < \beta < 1$,

$$\Pr[X_s < (1-\beta)\mu_s] \leq \left( \frac{e^{-\beta}}{(1-\beta)^{1-\beta}} \right)^{\mu_s} \tag{1.2}$$

Este teorema es el punto de partida necesario para desarrollar variantes de la cota de Chernoff para problemas más concretos. Estas variantes pueden no ser tan ajustadas, pero son más sencillas de usar. Es el caso del corolario que presentamos a continuación y cuya demostración presentamos en el apéndice A.

COROLARIO 1
Sean $X_1, X_2, \ldots, X_n$ variables aleatorias independientes que se corresponden con experimentos de Bernouilli. Cada variable $X_i$ tiene una probabilidad de éxito igual a $p_i$ y una probabilidad de fracaso igual a $q_i = (1 - p_i)$. Sea $X_s = \Sigma_{i=1}^{n} X_i$ una variable aleatoria cuyo valor esperado es $\mu_s = \Sigma_{i=1}^{n} p_i$.

$$\Pr[X_s > (1+\beta)\mu_s] \leq \begin{cases} e^{-\frac{\beta^2 \mu_s}{2+\beta}} & \text{si } \beta \geq 1 \\ e^{-\frac{\beta^2 \mu_s}{3}} & \text{si } 0 < \beta < 1 \end{cases} \tag{1.3}$$

$$\Pr[X_s < (1-\beta)\mu_s] \leq e^{-\frac{\beta^2 \mu_s}{2}} \quad \text{si } 0 < \beta < 1 \tag{1.4}$$

Hoeffing extendió, en 1963, la cota de Chernoff y propuso una nueva cota que recibió su nombre, la cota de Hoeffding [Hoe-1963]. Presentamos dicha cota en el siguiente teorema:

TEOREMA 2 (COTA GENERAL DE HOEFFDING )
Sean $X_1, X_2, \ldots, X_n$ variables aleatorias independientes tales que para cada $X_i$ existen valores $a_i$ y $b_i$ y se cumple que $a_i \leq X_i \leq b_i$ donde $1 \leq i \leq n$ y $\mathrm{E}(X_i) = p_i$. Sea $\overline{X} = \Sigma_{i=1}^{n} X_i / n$ una variable aleatoria cuyo

valor esperado es $\overline{\mu} = \Sigma_{i=1}^{n} p_i/n$.
Entonces, para cualquier $\varepsilon > 0$,

$$\Pr[\overline{X} - \overline{\mu} > \varepsilon] \leq e^{-2n^2\varepsilon^2/\sum_{i=1}^{n}(b_i-a_i)^2} \tag{1.5}$$

$$\Pr[\overline{\mu} - \overline{X} > \varepsilon] \leq e^{-2n^2\varepsilon^2/\sum_{i=1}^{n}(b_i-a_i)^2} \tag{1.6}$$

Una de las principales ventajas de usar las cotas de Chernoff y Hoeffding es que no dependen de la distribución de probabilidad que genera las observaciones [Dom-2000] y las condiciones que deben darse para usarlas son mínimas [Dom-1999]: únicamente se exige que las observaciones sean independientes y que no cambie la distribución que las genera. Este aspecto las hace muy útiles en cualquier situación, puesto que no hay que presuponer ningún tipo de distribución. Como contrapartida, estas cotas son bastante conservativas, lo que significa que el número de observaciones para alcanzar cotas ajustadas es mayor que usando cotas dependientes de la distribución. A pesar de esto último, puesto que nuestro principal objetivo es abordar grandes conjuntos de datos, el número de observaciones no es un problema determinante.

Las cotas de Chernoff y Hoeffding han sido formuladas de distintas formas [Dia-2001; Jan-2002; Chu-2006] dependiendo del uso que se les pretende dar. Puesto que el objetivo de este apartado es justificar la importancia de las cotas para nuestro trabajo, sin profundizar en toda la gama de variantes disponibles, las fórmulas que hemos presentado son las más adecuadas para nuestro enfoque.

## 1.2. Determinación del error de estimación absoluto para las cotas de concentración de Chernoff y Hoeffding

Existen ciertas diferencias entre las cotas de Chernoff y Hoeffding que nos harán usar enfoques diferentes para determinar el error de estimación calculado por ellas. Como puede apreciarse (ver ecuaciones 1.3 y 1.4), la cota de Chernoff está expresada en forma multiplicativa y el error cometido se presenta mediante una aproximación relativa. Por su parte, la cota de Hoeffding (ver ecuaciones 1.5 y 1.6) se expresa en forma aditiva y el error cometido se presenta mediante una aproximación absoluta. Otra diferencia a destacar es que, mientras que la cota de Chernoff hace los cálculos usando la suma de los eventos, la cota de Hoeffding usa directamente el valor medio. Por último y no menos importante, hay que considerar que para calcular la cota de Chernoff es preciso conocer el valor esperado para la suma de las variables, mientras que para la cota de Hoeffding tan sólo es preciso conocer el número de observaciones que se han registrado.

En primer lugar, y antes de presentar el desarrollo necesario para calcular el error de estimación en términos absolutos, vamos a unificar el tipo de estimación del error convirtiendo la estimación relativa usada por la cota de Chernoff en una estimación en términos absolutos (presentamos el razonamiento para la cota superior, pero el razonamiento de la cota inferior sigue un proceso similar):

$$\Pr[X_s > (1+\beta)\mu_s] = \Pr[X_s/n > (1+\beta)\mu_s/n] = \Pr[\overline{X} > (1+\beta)\overline{\mu}] \tag{1.7}$$

y considerando $\varepsilon = \beta\,\overline{\mu}$

$$\Pr[\overline{X} - \overline{\mu} > \beta\,\overline{\mu}] = \Pr[\overline{X} - \overline{\mu} > \varepsilon] \tag{1.8}$$

Como se puede apreciar, la fórmula resultante es equivalente a la estimación del error mediante aproximación absoluta presente en la cota de Hoeffding, con la única salvedad de que el error realmente sigue dependiendo del valor esperado de la media.

Además de calcular la estimación del error absoluto, nos interesa que ese cálculo sea soportado con una confianza mínima $(1-\delta)$. Para el peor caso, esta condición se expresa de la siguiente forma:

$$\Pr[\overline{\mu} - \overline{X} > \varepsilon] + \Pr[\overline{X} - \overline{\mu} > \varepsilon] = \Pr[|\overline{X} - \overline{\mu}| > \varepsilon] = \delta \tag{1.9}$$

y, si tenemos en cuenta que las colas sean simétricas, tenemos:

$$\Pr[\overline{\mu} - \overline{X} > \varepsilon] = \Pr[\overline{X} - \overline{\mu} > \varepsilon] = \frac{\delta}{2} \tag{1.10}$$

### 1.2.1. Error de estimación usando la cota de Chernoff

Para conseguir que la cota de Chernoff tenga la misma expresión para ambas colas, y coincida en esa característica con la cota de Hoeffding, vamos a relajar la cota inferior de la siguiente forma:

$$\Pr[X_s < (1-\beta)\mu_s] \leq e^{-\frac{\beta^2\mu_s}{2}} \leq e^{-\frac{\beta^2\mu_s}{3}}$$

consiguiendo simplificar la cota de Chernoff a las siguientes expresiones:

$$\Pr[X_s > (1+\beta)\mu_s] \leq \begin{cases} e^{-\frac{\beta^2\mu_s}{3}} & \text{si } 0 < \beta < 1 \\ e^{-\frac{\beta^2\mu_s}{2+\beta}} & \text{si } \beta \geq 1 \end{cases}$$

$$\Pr[X_s < (1-\beta)\mu_s] \leq \begin{cases} e^{-\frac{\beta^2\mu_s}{3}} & \text{si } 0 < \beta < 1 \\ e^{-\frac{\beta^2\mu_s}{2+\beta}} & \text{si } \beta \geq 1 \end{cases}$$

que, usando las ecuaciones 1.7 y 1.8 y las igualdades $\varepsilon = \beta\,\overline{\mu}$ y $\mu_s = \overline{\mu}\,n$, podemos escribir como:

$$\Pr[\overline{X} - \overline{\mu} > \varepsilon] \leq \begin{cases} e^{-\frac{n\varepsilon^2}{3\overline{\mu}}} & \text{si } 0 < \frac{\varepsilon}{\overline{\mu}} < 1 \\ e^{-\frac{n\varepsilon^2}{2\overline{\mu}+\varepsilon}} & \text{si } \frac{\varepsilon}{\overline{\mu}} \geq 1 \end{cases} \tag{1.11}$$

$$\Pr[\overline{\mu} - \overline{X} > \varepsilon] \leq \begin{cases} e^{-\frac{n\varepsilon^2}{3\overline{\mu}}} & \text{si } 0 < \frac{\varepsilon}{\overline{\mu}} < 1 \\ e^{-\frac{n\varepsilon^2}{2\overline{\mu}+\varepsilon}} & \text{si } \frac{\varepsilon}{\overline{\mu}} \geq 1 \end{cases} \tag{1.12}$$

De esta forma, haciendo el razonamiento para la cola de la ecuación 1.11 (que será similar para la otra cola – ecuación 1.12–), sabemos que para $0 < \frac{\varepsilon}{\overline{\mu}} < 1$:

$$\begin{aligned} \Pr[\overline{X} - \overline{\mu} > \varepsilon] &\leq e^{-\frac{n\varepsilon^2}{3\overline{\mu}}} \\ \frac{\delta}{2} &\leq e^{-\frac{n\varepsilon^2}{3\overline{\mu}}} \\ \frac{n\varepsilon^2}{3\overline{\mu}} &\leq \ln\left(\frac{2}{\delta}\right) \\ \varepsilon &\leq \sqrt{\frac{3\,\overline{\mu}}{n}\ln\left(\frac{2}{\delta}\right)} \end{aligned} \tag{1.13}$$

y, para el caso en que $\frac{\varepsilon}{\overline{\mu}} \geq 1$:

$$\begin{aligned} \Pr[\overline{X} - \overline{\mu} > \varepsilon] &\leq e^{-\frac{n\varepsilon^2}{2\overline{\mu}+\varepsilon}} \\ \frac{\delta}{2} &\leq e^{-\frac{n\varepsilon^2}{2\overline{\mu}+\varepsilon}} \\ \frac{n\varepsilon^2}{2\overline{\mu}+\varepsilon} &\leq \ln\left(\frac{2}{\delta}\right) \\ n\varepsilon^2 - \varepsilon\ln\left(\frac{2}{\delta}\right) - 2\overline{\mu}\ln\left(\frac{2}{\delta}\right) &\leq 0 \end{aligned}$$

con lo que, al tener una parábola en función de $\varepsilon$ con un mínimo y dos raíces de distinto signo, la cota para $\varepsilon$ en el peor caso estará definida por la raíz positiva:

$$\varepsilon \leq \frac{\ln\left(\frac{2}{\delta}\right) + \sqrt{\ln^2\left(\frac{2}{\delta}\right) + 8n\overline{\mu}\ln\left(\frac{2}{\delta}\right)}}{2n} \tag{1.14}$$

Si combinamos las dos ecuaciones anteriores (1.13 y 1.14) podemos expresar el error absoluto en una cola usando la cota de Chernoff como:

$$\varepsilon \leq \begin{cases} \sqrt{\frac{3\,\overline{\mu}}{n}\ln\left(\frac{2}{\delta}\right)} & \text{si } 0 < \frac{\varepsilon}{\overline{\mu}} < 1 \\ \frac{\ln\left(\frac{2}{\delta}\right) + \sqrt{\ln^2\left(\frac{2}{\delta}\right) + 8n\overline{\mu}\ln\left(\frac{2}{\delta}\right)}}{2n} & \text{si } \frac{\varepsilon}{\overline{\mu}} \geq 1 \end{cases} \tag{1.15}$$

### 1.2.2. Error de estimación usando la cota de Hoeffding

La cota de Hoeffding está definida para variables acotadas entre dos valores, pero si consideramos, como Chernoff, que las variables sólo podrán tomar valores en 0 y 1, es decir $X_i \in \{0, 1\}$, tendremos que $a_i = 0$, $b_i = 1$ y $\sum_{i=1}^{n}(b_i - a_i)^2 = n$.

Para calcular el error absoluto en una cola a partir de la cota de Hoeffding seguimos un razonamiento similar al empleado con la cota de Chernoff, como mostramos a continuación (el razonamiento para la otra cola es simétrico):

$$\begin{aligned} \Pr[\overline{X} - \overline{\mu} > \varepsilon] &\leq e^{-2n^2\varepsilon^2/\sum_{i=1}^{n}(b_i-a_i)^2} \\ \frac{\delta}{2} &\leq e^{-2n^2\varepsilon^2/n} \\ 2n\varepsilon^2 &\leq \ln\left(\frac{2}{\delta}\right) \\ \varepsilon &\leq \sqrt{\frac{1}{2n}\ln\left(\frac{2}{\delta}\right)} \end{aligned} \tag{1.16}$$

## 1.3. Uso de cotas de concentración en aprendizaje incremental

Las cotas de concentración, como ya hemos comentado, han tenido un uso bastante extendido en el área del aprendizaje automático y de la minería de datos. En esta sección pretendemos reflejar cómo se han usado las cotas de concentración, y más concretamente las cotas de Chernoff y Hoeffding, para resolver diversos problemas.

En primer lugar, y como era de esperar, las cotas de Chernoff y Hoeffding se han usado para estimar la suma de variables aleatorias independientes. Muchas aplicaciones derivadas del uso directo de esta estimación no pertenecen al área del aprendizaje automático, sino a tareas de optimización (empaquetado [Rag-1988], asignación de recursos [Hes-2002], etc.). En el artículo de Schdmit, Siegel y Srinivasan de 1995 [Sch-1995] se siguen presentando algunas de estas tareas de optimización (enrutamiento de paquetes o planificación de tareas) pero también empiezan a incluirse tareas englobadas claramente en el ámbito del aprendizaje automático. Se comienza a poner de manifiesto que el uso de las cotas de concentración se consolida como herramienta útil para afrontar ciertos problemas.

A principios de los años 90 empiezan a usarse las cotas para calcular la proximidad entre la media estimada y el valor real con una cierta confianza. Greiner y Jurisica [Gre-1992] utilizan la cota de Hoeffding para determinar si el resultado de una modificación concreta a un modelo aprendido previamente se convertirá, con alta probabilidad, en una mejora real. Por su parte, Maron y Moore [Mar-1994] usan la misma

cota de Hoeffding para distinguir los modelos de aprendizaje más apropiados para resolver un determinado problema. Estos cálculos tienen en cuenta el tamaño de la muestra utilizada y, aunque no están enfocadas para trabajar con grandes volúmenes de datos, establecen un primer paso hacia los usos actuales.

Otro uso bastante extendido de las cotas de concentración ha sido determinar qué tamaño de muestra mínimo era el más apropiado para tomar alguna decisión. Dicha decisión debería satisfacer al mismo tiempo unos niveles mínimos de confianza que aseguraran que tal decisión era lo suficientemente buena. La tarea de determinar ese tamaño mínimo para la muestra se conoce como el problema de estimar la "*complejidad de la muestra*" para la tarea de aprendizaje [Kea-1994]. Así, el objetivo de Haussler [Hau-1992] era determinar cuándo una regla de decisión podía ser considerada como suficientemente buena para ser usada. Kivinen y Mannila [Kiv-1994] querían calcular, con cierta confianza, qué tamaño de muestra era necesario para asegurar que una fórmula lógica con cuantificadores universales (o existenciales) no se satisfacía en una base de datos relacional. Con la misma finalidad Srikant y Agrawal [Sri-1995; Sri-1997], y posteriormente Toivonen [Toi-1996], usan las cotas en el campo de las reglas de asociación para, respectivamente, determinar el soporte de las reglas o encontrar los conjuntos frecuentes. Otra tarea relevante, propia de la extracción de conocimiento en bases de datos, es la estimación de la frecuencia relativa de diferentes subgrupos y Wrobel [Wro-1997] usó las cotas para acotar dicha estimación, limitando la probabilidad de no considerar alguna de las mejores hipótesis.

Más recientemente, a partir del trabajo de Domingos y Hulten [Dom-2000], esta técnica se ha usado en el ámbito de la inducción de árboles de decisión para determinar cuántas experiencias deben ser muestreadas antes de poder elegir, con determinada confianza, qué atributo debe ser usado para la expansión de cada nodo. Una vez comprobadas las ventajas de este enfoque para inducir árboles de decisión de forma rápida y fiable, otros autores han mejorado y extendido el trabajo de Domingos y Hulten. Es el caso de Gama, Rocha y Medas que han trabajado en la adaptación del algoritmo propuesto por Domingos y Hulten para poder tratar atributos numéricos y mejorar la calidad en la predicción de observaciones [Gam-2003]. Estos mismos autores también se han basado en el mismo criterio de selección de atributos para la expansión de nodos durante la inducción de bosques de árboles [Gam-2004].

Otro enfoque bastante interesante, y que será una de las bases del trabajo que presentamos en esta parte de la tesis, es usar las cotas de concentración para el muestreo secuencial (o adaptativo). En este caso, ni se usa el tamaño de la muestra completa para determinar los intervalos de confianza, ni se fija el tamaño de la muestra necesario para hacer afirmaciones con determinada confianza. Con el enfoque de muestreo secuencial lo que se pretende es muestrear experiencias de forma constante hasta que el nivel de confianza exigido sea alcanzado. Ésta es la razón para que el muestreo secuencial también se conozca como adaptativo: el tamaño necesario de la muestra dependerá de las experiencias que se hayan muestreado y variará según el problema. De entre los distintos enfoques que hay para hacer muestreo adaptativo, en esta sección nos centramos en aquellos que usan las cotas de concentración. Domingo, Gavaldà y Watanabe [Dom-1999; Dom-2002] definen el algoritmo AdaSelect que usa un enfoque basado en muestreo secuencial que procesa experiencias de forma continua y calcula los valores estimados de ciertas variables usando la cota de Hoeffding. Ramos, Morales y del Campo [Ram-2000; Ram-2001; Ram-2004] también plantean este enfoque aplicándolo a la tarea de inducir árboles de decisión en el algoritmo IADEM-0, pero a la novedad del enfoque también hay que añadir el hecho de que los intervalos definidos para las variables son calculados combinando las cotas de Chernoff y Hoeffding. La razón para usar ambas cotas es que los intervalos son más ajustados para una cota u otra dependiendo de las condiciones del problema. Este uso combinado se ha mostrado bastante útil y su utilización está empezando a extenderse [Bif-2006].

## 1.4. Combinando las cotas de Chernoff y Hoeffding

Como acabamos de comentar, las cotas de Chernoff y Hoeffding ofrecen valores diferentes para el margen de error dependiendo de la información disponible, siendo en algunas circunstancias más ajustada la cota de Chernoff que la de Hoeffding y viceversa. Esto tiene la ventaja de que podemos seleccionar la más adecuada para reducir el margen de error lo más rápidamente posible sin disminuir la confianza en el

resultado obtenido.

Esta característica permite combinar ambas cotas y usarlas para calcular, de forma más ajustada, el intervalo donde se encontrarán determinadas variables. Como ya hemos adelantado, en el algoritmo que presentaremos como núcleo de esta parte de la tesis (que comienza en el capítulo 2), necesitaremos estimar los valores para una serie de variables (frecuencias relativas) con una determinada confianza $(1-\delta)$. Para calcularlos dispondremos de su valor estimado ($\overline{\mu}_n$), obtenido después de haber procesado un conjunto de experiencias (cuyo tamaño será $n$). En esta sección definiremos la función cuyo objetivo será calcular el menor margen de error posible para una variable y mostraremos que bajo diferentes condiciones se usa una cota de concentración u otra.

### 1.4.1. Estimación del margen de error en el cálculo de una variable

La estimación del intervalo en el que deben encontrarse ciertas variables se determina usando el valor estimado de su media y considerando el margen de error absoluto . Para calcular dicho error absoluto ($\varepsilon$) usaremos las cotas de concentración que tienen en cuenta el total de experiencias que se han observado ($n \in \mathbb{N}$), la frecuencia absoluta calculada usando esas experiencias ($\overline{\mu}_n \in [0,1]$), y el valor que determina la confianza requerida ($\delta \in (0,1)$). Esa es la razón de que hayamos definido el margen de error usando tres variables: $margen\_\varepsilon(n, \overline{\mu}_n, \delta)$. A continuación estudiaremos cómo combinar las dos cotas de concentración que hemos presentado, la de Chernoff y la de Hoeffding, para obtener la estimación del menor margen de error, pero manteniendo la confianza exigida por el usuario.

Puesto que lo que se pretende es saber si la probabilidad de que el valor estimado ($\overline{\mu}_n$) esté cerca del valor real ($\overline{\mu}$) con una confianza mayor que la exigida por el usuario ($\Pr[|\overline{\mu}_n - \overline{\mu}| \leq \varepsilon] \geq 1-\delta$) y dado que $\overline{\mu}_n \in [0,1]$, estaremos considerando el intervalo $\overline{\mu} - \varepsilon \leq \overline{\mu}_n \leq \overline{\mu} + \varepsilon$. Esto requiere que $\varepsilon < \overline{\mu}$ y, en consecuencia, que sólo consideremos la expansión de la cota de Chernoff en la rama que contempla el caso $\frac{\varepsilon}{\overline{\mu}} < 1$. De esta forma, el error absoluto que buscamos vendrá determinado por las fórmulas:

$$\Pr[\overline{\mu}_n - \overline{\mu} > \varepsilon] \leq e^{-\frac{n\varepsilon^2}{3\overline{\mu}}} \tag{1.17}$$

$$\Pr[\overline{\mu} - \overline{\mu}_n > \varepsilon] \leq e^{-\frac{n\varepsilon^2}{3\overline{\mu}}} \tag{1.18}$$

Si observamos las ecuaciones anteriores (1.17 y 1.18) podemos ver que el error ($\varepsilon$) depende del valor real de la media ($\overline{\mu}$), pero nosotros tan solo conocemos el valor estimado después de haber observado $n$ experiencias ($\overline{\mu}_n$). La forma que tenemos para aproximar dicho valor real será usando el valor estimado ($\overline{\mu}_n$) en combinación con el propio error que deseamos calcular ($\varepsilon$): $\overline{\mu}_n - \varepsilon \leq \overline{\mu} \leq \overline{\mu}_n + \varepsilon$. Consideraremos el valor de $\overline{\mu}$ que nos sitúe en el peor caso, es decir, aquél en el que el error sea el mayor posible. Así, sustituiremos $\overline{\mu}$ por $\overline{\mu}_n + \varepsilon$ en la ecuación anterior y obtendremos la siguiente expresión (que será similar para la otra cola):

$$\begin{aligned}
\Pr[\overline{\mu}_n - \overline{\mu} > \varepsilon] &\leq e^{-\frac{n\varepsilon^2}{3(\overline{\mu}_n+\varepsilon)}} \\
\frac{\delta}{2} &\leq e^{-\frac{n\varepsilon^2}{3(\overline{\mu}_n+\varepsilon)}} \\
\frac{n\varepsilon^2}{3(\overline{\mu}_n+\varepsilon)} &\leq \ln\left(\frac{2}{\delta}\right) \\
n\varepsilon^2 - 3\ln\left(\frac{2}{\delta}\right)\varepsilon - 3\overline{\mu}_n \ln\left(\frac{2}{\delta}\right) &\leq 0
\end{aligned}$$

Tenemos una parábola en función de $\varepsilon$, con un mínimo y dos raíces de distinto signo, haciendo que la cota

para $\varepsilon$ en el peor caso sea la raíz positiva:

$$\varepsilon \leq \frac{3\ln\left(\frac{2}{\delta}\right) + \sqrt{9\ln^2\left(\frac{2}{\delta}\right) + 12n\overline{\mu}_n \ln\left(\frac{2}{\delta}\right)}}{2n} \tag{1.19}$$

Notaremos el margen de error calculado con la cota de Chernoff con $\varepsilon_c$ y será el mayor valor alcanzado por la ecuación anterior (1.19):

$$\varepsilon_c = \frac{3\ln\left(\frac{2}{\delta}\right) + \sqrt{9\ln^2\left(\frac{2}{\delta}\right) + 12n\overline{\mu}_n \ln\left(\frac{2}{\delta}\right)}}{2n} \tag{1.20}$$

El uso de la cota de Hoeffding para determinar el margen de error es bastante más directa, dado que en su fórmula sólo se considera el número de experiencias observadas ($n$). Notaremos el margen de error calculado con la cota de Hoeffding con $\varepsilon_h$ y será el mayor valor alcanzado por la ecuación 1.16:

$$\varepsilon_h = \sqrt{\frac{1}{2n}\ln\left(\frac{2}{\delta}\right)} \tag{1.21}$$

Una vez presentado todo el razonamiento previo y recordando que las dos colas son simétricas, podemos resumir, en la siguiente ecuación, la fórmula final empleada para calcular el error cometido en una cola para la estimación de una variable combinando las cotas de Chernoff y Hoeffding:

$margen\_\varepsilon : \mathbb{N} \times [0,1] \times (0,1) \to (0,1]$

$$margen\_\varepsilon(n, \overline{\mu}_n, \delta) = \begin{cases} 1 & \text{si } n = 0 \\ \\ \text{mín}\{\varepsilon_c, \varepsilon_h, 1\} & \text{si } n \neq 0 \end{cases} \tag{1.22}$$

donde

$$\varepsilon_c = \frac{3\ln\left(\frac{2}{\delta}\right) + \sqrt{9\ln^2\left(\frac{2}{\delta}\right) + 12n\overline{\mu}_n \ln\left(\frac{2}{\delta}\right)}}{2n}$$

$$\varepsilon_h = \sqrt{\frac{1}{2n}\ln\left(\frac{2}{\delta}\right)}$$

Debemos hacer notar que el valor para $margen\_\varepsilon(n, \overline{\mu}_n, \delta)$ es 1 (el máximo error posible) cuando el divisor para ambas cotas ($n$) es 0. Este caso se da cuando no se dispone de ninguna información y, por lo tanto, el margen de error será el máximo posible.

### 1.4.2. Representación gráfica del margen de error en el cálculo de una variable

Una vez hemos determinado la ecuación que calcula el error cometido en una cola para la estimación de una variable considerando el número de experiencias observadas, vamos a comprobar que, dependiendo de la información disponible, es mejor usar la cota de Chernoff o la cota de Hoeffding.

En la figura 1.1 hemos representado las curvas del error calculado con las cotas de Chernoff (ecuación 1.20) y Hoeffding (ecuación 1.21) cuando la confianza exigida en la estimación es del 95 % ($\delta = 0,05$). Como puede apreciarse, la cota de Chernoff para una confianza previamente establecida depende tanto de $n$ (número de experiencias observadas), como de $\overline{\mu}_n$ (valor estimado usando $n$ experiencias). Por su parte, la cota de Hoeffding tan sólo depende de $n$.

Como es lógico, pretendemos usar la cota más ajustada ofrecida por ambas cotas. Para determinar cuándo se usaría la cota de Chernoff ($\varepsilon_c$) y cuándo la de Hoeffding ($\varepsilon_h$) debemos comparar las ecuaciones 1.20 y

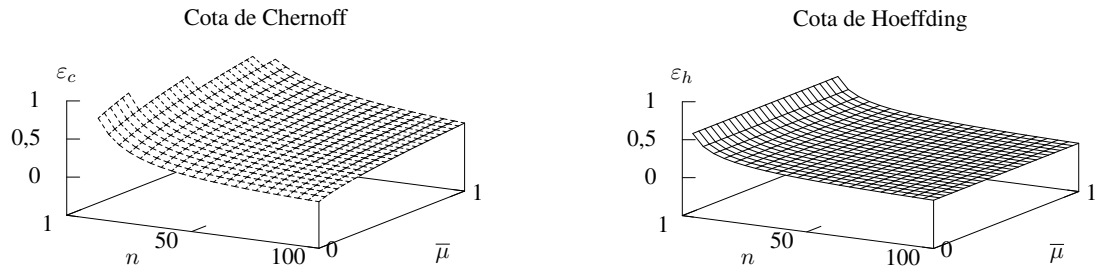



Figura 1.1: Márgenes de error calculados con las cotas de Chernoff y Hoeffding.

1.21 para saber en qué caso tenemos la menor cota. De comparar ambas cotas resulta la siguiente desigualdad:

$$\begin{aligned}
\varepsilon_h &\leq \varepsilon_c \\
\sqrt{\frac{1}{2n}\ln\left(\frac{2}{\delta}\right)} &\leq \frac{3\ln\left(\frac{2}{\delta}\right)+\sqrt{9\ln^2\left(\frac{2}{\delta}\right)+12n\overline{\mu}_n\ln\left(\frac{2}{\delta}\right)}}{2n} \\
\sqrt{2n\ln\left(\frac{2}{\delta}\right)}-3\ln\left(\frac{2}{\delta}\right) &\leq \sqrt{9\ln^2\left(\frac{2}{\delta}\right)+12n\overline{\mu}_n\ln\left(\frac{2}{\delta}\right)} \qquad (1.23)
\end{aligned}$$

A partir de cierto valor para $n$ ($n > \frac{9}{2}\ln\left(\frac{2}{\delta}\right)$) podemos asegurar que el término de la izquierda en la inecuación anterior (1.23) es positivo. Pero, si queremos elevar al cuadrado ambos términos en dicha inecuación y asegurar que la desigualdad se mantiene incluso cuando el término de la izquierda es negativo, debemos comprobar que para $1 \leq n \leq \frac{9}{2}\ln\left(\frac{2}{\delta}\right)$ se cumple lo siguiente:

$$\begin{aligned}
\left|\sqrt{2n\ln\left(\frac{2}{\delta}\right)}-3\ln\left(\frac{2}{\delta}\right)\right| &\leq \sqrt{9\ln^2\left(\frac{2}{\delta}\right)+12n\overline{\mu}_n\ln\left(\frac{2}{\delta}\right)} \\
\left|\sqrt{2n\ln\left(\frac{2}{\delta}\right)}-3\ln\left(\frac{2}{\delta}\right)\right| &\leq 3\ln\left(\tfrac{2}{\delta}\right) \leq \sqrt{9\ln^2\left(\frac{2}{\delta}\right)+12n\overline{\mu}_n\ln\left(\frac{2}{\delta}\right)}
\end{aligned}$$

Habiendo demostrado lo anterior, podemos elevar al cuadrado la ecuación 1.23 y completar su desarrollo:

$$\begin{aligned}
2n\ln\left(\frac{2}{\delta}\right)-6\ln\left(\frac{2}{\delta}\right)\sqrt{2n\ln\left(\frac{2}{\delta}\right)}+9\ln^2\left(\frac{2}{\delta}\right) &\leq 9\ln^2\left(\frac{2}{\delta}\right)+12n\overline{\mu}_n\ln\left(\frac{2}{\delta}\right) \\
n-3\sqrt{2n\ln\left(\frac{2}{\delta}\right)} &\leq 6n\overline{\mu}_n \\
\overline{\mu}_n &\geq \frac{1}{6}-\sqrt{\frac{1}{2n}\ln\left(\frac{2}{\delta}\right)}
\end{aligned}$$

Por lo tanto:

$$\varepsilon_h \leq \varepsilon_c \quad \text{cuando} \quad \overline{\mu}_n \geq \frac{1}{6}-\sqrt{\frac{1}{2n}\ln\left(\frac{2}{\delta}\right)}$$

$$\varepsilon_h > \varepsilon_c \quad \text{cuando} \quad \overline{\mu}_n \; < \; \frac{1}{6} - \sqrt{\frac{1}{2n} \ln\left(\frac{2}{\delta}\right)}$$

Podemos observar que la cota de Hoeffding se usará siempre para valores de $\overline{\mu}_n$ mayores que $1/6$, mientras que, para valores menores que $1/6$ se usará mayoritariamente la cota de Chernoff. La cota de Hoeffding se seguirá usando para valores cercanos y menores que $1/6$ cuando $n$ tome valores pequeños. En la figura 1.2 hemos representado conjuntamente ambas superficies y puede apreciarse gráficamente las conclusiones que acabamos de describir.

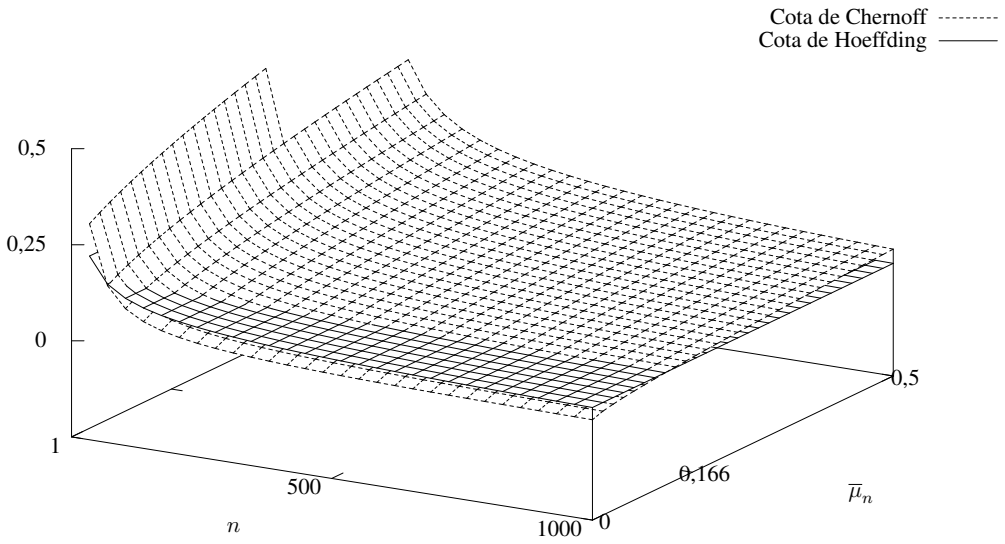



Figura 1.2: Combinando los márgenes de error de las cotas de Chernoff y Hoeffding.

En esta sección hemos mostrado, analítica y gráficamente, la utilidad de usar de forma combinada las cotas de Chernoff y Hoeffding, puesto que, dependiendo de la información disponible, la cota más ajustada será una u otra. La ecuación 1.22, resultado de la combinación de ambas cotas, será usada en uno de los algoritmos que proponemos y cuya descripción comienza en el siguiente capítulo.

# Capítulo 2

# Inducción de Árboles de Decisión por Muestreo: IADEM-0

La facilidad que existe para recopilar y almacenar información de diversas fuentes sobre diferentes escenarios hace que el volumen de datos disponible en determinados ámbitos sea muy elevado. Extraer conocimiento a partir de tal cantidad de información se convierte en una tarea inabordable con los procedimientos tradicionales que intentan estudiar todos los casos al mismo tiempo. Surge, por tanto, la necesidad de desarrollar métodos que permitan inducir conocimiento de forma escalable [Pro-1999]. Existen diversos enfoques para afrontar esta problemática, pero el "*muestreo*" sienta las bases de uno de los más empleados. Las técnicas de muestreo han sido ampliamente estudiadas desde hace muchos años [Fel-1968] y múltiples resultados han sido publicados en el campo de la estadística (teorema central del límite, cotas de concentración, etc.).

Una de las ventajas de este enfoque es su versatilidad, lo que permite que la práctica totalidad de los algoritmos de aprendizaje puedan usar el muestreo para conseguir resultados que de otra forma no serían alcanzables. Además de lo beneficioso que puede resultar usar esta técnica, tenemos que destacar la riqueza de variantes que existen para alcanzar diversos objetivos. Catlett [Cat-1991] estudió diferentes técnicas y comparó empíricamente sus resultados, centrándose en el "*muestreo aleatorio*" y en el "*muestreo estratificado*". El primero de ellos es la versión más sencilla del muestreo y consiste simplemente en tomar una muestra del conjunto de datos seleccionando los elementos de forma aleatoria. Por su parte, el muestreo estratificado está diseñado de forma que las clases minoritarias sean incorporadas a la muestra con mayor frecuencia que el resto. Surgiendo del mismo ámbito de la estadística tenemos el "*doble muestreo*" propuesto por Cox [Cox-1952] en el que una primera muestra es tomada para capturar las características del problema y estimar cuál es el mejor tamaño necesario para realizar la tarea requerida [Hou-1991].

Existen distintas formas de usar las técnicas de muestreo en el ámbito del aprendizaje computacional. Un enfoque es procesar una muestra de tamaño fijo y determinar la calidad que se puede esperar basándose en las características de los datos. Como ejemplo de este uso tenemos algoritmos que estiman el error descomponiéndolo en distintas componentes (una por cada hipótesis) [Hau-1992], pero tiene el inconveniente de que están diseñados suponiendo que se conoce la distribución. Otros algoritmos que usan el mismo enfoque de extraer una muestra de tamaño fijo y evaluar la calidad del modelo son los propuestos por Freund y conocidos como algoritmos de aprendizaje "*auto-acotados*" [Fre-1998], pero en este caso no se necesita conocer la distribución. Este enfoque puede resultar útil, pero nos encontramos con el problema de tener que determinar cuál será el tamaño de la muestra. Para solucionar este aspecto, se puede abordar el problema desde la perspectiva contraria: establecer cuál es la calidad mínima que se le exige al modelo y calcular que tamaño de muestra es necesario para asegurar (con cierta confianza) que se alcanza el objetivo. Para este segundo enfoque son especialmente útiles las cotas de concentración , puesto que nos permiten calcular el tamaño de la muestra para el peor caso. Diversos algoritmos han aprovechado esta característica de una u otra forma [Kiv-1994; Toi-1996; Wro-1997].

El inconveniente que tiene estimar el tamaño de la muestra a priori ("*muestreo estático*") es que podemos

sobrestimar, es decir, las cotas de concentración calculan el tamaño para el peor caso y puede que ese no sea el caso actual. En tal situación, el tamaño será mayor que el necesario, con el consecuente coste. Dada esta circunstancia, y apoyándose en el análisis secuencial [Dod-1929; Wal-1947], surge el "*muestreo secuencial*". Sin usar explícitamente tal terminología, Greiner propuso el algoritmo PALO [Gre-1996], con el que se pone de manifiesto que el número de experiencias necesarias en la muestra no tiene que ser siempre tan elevado como el que sea calculado por las cotas de concentración para el peor caso.

El muestreo secuencial [Gav-2001; Sch-2002] no fija el tamaño de la muestra a priori, sino que iterativamente toma muestras del conjunto de datos. Con cada nueva muestra se vuelven a evaluar los criterios establecidos previamente. Cuando sean alcanzados, la ejecución se detiene habiendo logrado el objetivo de satisfacer los requisitos fijados. Dado que el número total de experiencias que se tomarán del conjunto de datos no es fijado a priori, sino que se determina para cada problema teniendo en cuenta el problema y el modelo inducido hasta el momento, al muestreo secuencial también se le conoce como "*dinámico*" [Joh-1996] o "*adaptativo*" [Lip-1990; Dom-1999; Dom-2002]. Típicamente el número de experiencias que se toma en cada muestra es fijo, pero, en el caso de que el tamaño de la muestra varíe, hablaremos de "*muestreo progresivo*" [Pro-1999a; Par-2002]. Otra variante, algo más alejada del muestreo secuencial, pero que tampoco establece el tamaño de la muestra a priori es el "*muestreo activo*" [Saa-2004] en el que la principal característica es que incluye criterios para seleccionar el tipo de elementos que deben pertenecer a la muestra.

El algoritmo que presentamos en este capítulo y que será la base del que propondremos en el siguiente (capítulo 3) usa el enfoque de muestreo secuencial (adaptativo o dinámico) aprovechando sus beneficios y tomando (por muestreo) la cantidad de experiencias necesarias hasta alcanzar un modelo con la calidad mínima requerida por el usuario. No es el objetivo de este trabajo estudiar la forma más eficiente de extraer las muestras del conjunto de datos (aspecto sobre el que se está investigando recientemente [Jer-2004; Wan-2005]) sino, simplemente, usar el enfoque basado en muestreo.

El algoritmo que exponemos a continuación, IADEM-0, fue desarrollado, en su versión inicial, por Ramos y Morales [Ram-2000; Ram-2001]. En la primera sección (2.1) se introducen sus ideas más básicas de manera informal para establecer un primer contacto con las características del mismo. A continuación, en la sección 2.2, se presenta con todo detalle cómo opera este algoritmo. La implementación inicial propuesta [Ram-2001] ha sido sustituida por una definición general del funcionamiento del algoritmo [Cam-2002]. De esta forma hemos abstraído los procedimientos explícitos previos y se dota de total libertad al diseñador a la hora de implementar cualquier versión del algoritmo. El proceso de abstracción llevado a cabo ha permitido mejorar el algoritmo contemplando casos extremos anteriormente no considerados y refinando los cálculos bajo diversas condiciones. A continuación hemos incluido un estudio sobre la importancia de los parámetros internos utilizados en el algoritmo (ver sección 2.3) y de sus entradas (ver sección 2.4) para analizar su influencia en el comportamiento del mismo. Por último, antes de terminar este capítulo, desarrollaremos un estudio sobre la complejidad del algoritmo en la sección 2.5.

## 2.1. Descripción informal del algoritmo IADEM-0

El algoritmo IADEM-0 [Ram-2000; Ram-2001; Ram-2004] induce árboles de decisión (concepto presentado en la sección 0.1.3) a partir de las experiencias muestreadas del conjunto de datos del que se quiere extraer conocimiento. Como se ha comentado anteriormente, el muestreo ofrece al campo del aprendizaje automático unas ventajas considerables, que se acrecientan cuando el aprendizaje que se intenta hacer es incremental. En esta sección expondremos las ideas principales que confieren al algoritmo IADEM-0 sus características.

El nombre del algoritmo, IADEM-0, viene del acrónimo de "*Inducción de Árboles de DEcisión por Muestreo*" y el número cero hace referencia a que el nivel de ruido en el conjunto de datos debe ser cero para que funcione correctamente (esta limitación y otras han sido superadas y se comentan en el capítulo 3).

IADEM-0 es un algoritmo sin memoria para almacenar las experiencias que procesa, por lo tanto, estamos hablando de un algoritmo que olvida cualquier experiencia que le llega. Pero, aunque olvida las ex-

periencias, no olvida la información relevante que de ellas puede extraer. Para conservar dicha información usa una representación probabilística sustentada en un modelo basado en árboles de decisión.

En el árbol de decisión que induce IADEM-0, cada nodo de decisión se etiqueta con un atributo categórico (o nominal) y de cada nodo padre salen tantos arcos como valores hayan sido definidos para el atributo que lo etiqueta, pero incluye una novedad: más allá del borde formado por las hojas del árbol se define una frontera de hojas virtuales. En la figura 2.1 mostramos la representación de un posible árbol de decisión generado por el algoritmo. El problema que aborda es sencillo, sólo presenta cuatro atributos, pero creemos que es lo suficientemente ilustrativo para explicar la estructura que proponemos sin considerar aún los detalles. A las hojas que forman la frontera real del árbol las hemos llamado "*hojas reales*" y a todas las posibles futuras hojas las hemos llamado "*hojas virtuales*". En las hojas (tanto reales como virtuales) se incluyen contadores que permitirán al algoritmo estimar un conjunto variables con las que inducir el árbol de decisión. Una vez se ha determinado la hoja real que va a ser expandida y se ha seleccionado el atributo que se va a usar para realizar la expansión, el proceso de expansión consiste en eliminar la hoja real y sustituirla por un nodo interno (o nodo de decisión) etiquetado con el atributo seleccionado. Las hojas virtuales que contenían información para ese atributo se convierten en nuevas hojas reales y descendientes del nuevo nodo de decisión. Al realizar el proceso de expansión de esta forma, el nuevo modelo resultante dispone de información previa (la almacenada en lo que antes eran hojas virtuales y ahora son reales) y no existe ninguna rama (camino desde la raíz hasta una hoja) sin información, ni siquiera las nuevas.

Así pues, si existen $a$ atributos y para alcanzar una hoja se han usado $x$ atributos ($x < a$), esa hoja tiene $a - x$ "*hijos virtuales*" (uno por cada atributo aún no usado) y de cada hijo virtual dependerán tantas hojas virtuales como valores tenga el atributo correspondiente. Al expandir la hoja usando uno de los atributos libres en la rama, se crea un nodo de decisión etiquetado con el atributo seleccionado y se sustituye la hoja que se está expandiendo por el nuevo nodo de decisión. Las hojas reales que dependerán del nuevo nodo interno que se acaba de definir serán las hojas virtuales que dependían del hijo virtual que llevaba la información del atributo usado. El resto de hojas virtuales, que estaban asociadas con los atributos que no han sido usados, son eliminadas y para cada nueva hoja real se crean $a - (x + 1)$ nuevos hijos virtuales.

La ventaja de utilizar un árbol de decisión con una estructura como la descrita es doble: siempre se dispone de información en el árbol para todas las ramas y se puede decidir qué expansión es la más adecuada para una hoja teniendo en cuenta cómo quedaría el árbol de decisión, tratando de inducir el árbol que más se parezca al óptimo. Sabemos que inducir un árbol de decisión óptimo es un problema NP-completo [Hya-1971; Nau-1991], y por eso se usan heurísticos que intentan aproximarse lo más posible a él. El árbol óptimo será aquél capaz de clasificar correctamente las observaciones usando el menor número de decisiones, por lo tanto, el tamaño y la precisión alcanzada por el árbol serán determinantes para evaluar la calidad del resultado obtenido. Hacia esa mayor calidad del árbol inducido está enfocada la estructura utilizada y la forma de inducir el árbol.

En la figura 2.2 presentamos el pseudocódigo del algoritmo IADEM-0 cuyos componentes describiremos a continuación y cuya definición formal se presentará en la siguiente sección. La primera acción a realizar es la inicialización de la estructura (INICIALIZAR), cuyo objetivo es generar un árbol de decisión con un único nodo, que será una hoja real de la que dependerán tantas hojas virtuales como atributos y valores hayan sido definidos en la descripción del problema. Todos los contadores asociados a las hojas (reales o virtuales) serán inicializados a cero.

El núcleo del algoritmo viene caracterizado por una acción iterativa que será la que inducirá el árbol de decisión y que se repetirá mientras no se alcance la calidad exigida. Dicha acción iterativa se define con dos acciones fundamentales: la observación de nuevas experiencias (MUESTREAR_Y_RECALCULAR) y la expansión del árbol (EXPANDIR_ÁRBOL) si se considera necesaria. Como ya hemos adelantado, la información relevante de las experiencias observadas se conserva usando una representación probabilística soportada por la estructura del árbol de decisión. Usando esa información, el algoritmo puede calcular una estimación (en el mejor y en el peor caso) del error de clasificación cometido por el árbol. Esta estimación nos ofrece una interesante ventaja: conocemos en todo momento cuál es el error cometido por el árbol que estamos induciendo. Con esa estimación se le puede ofrecer al usuario la posibilidad de que él establezca

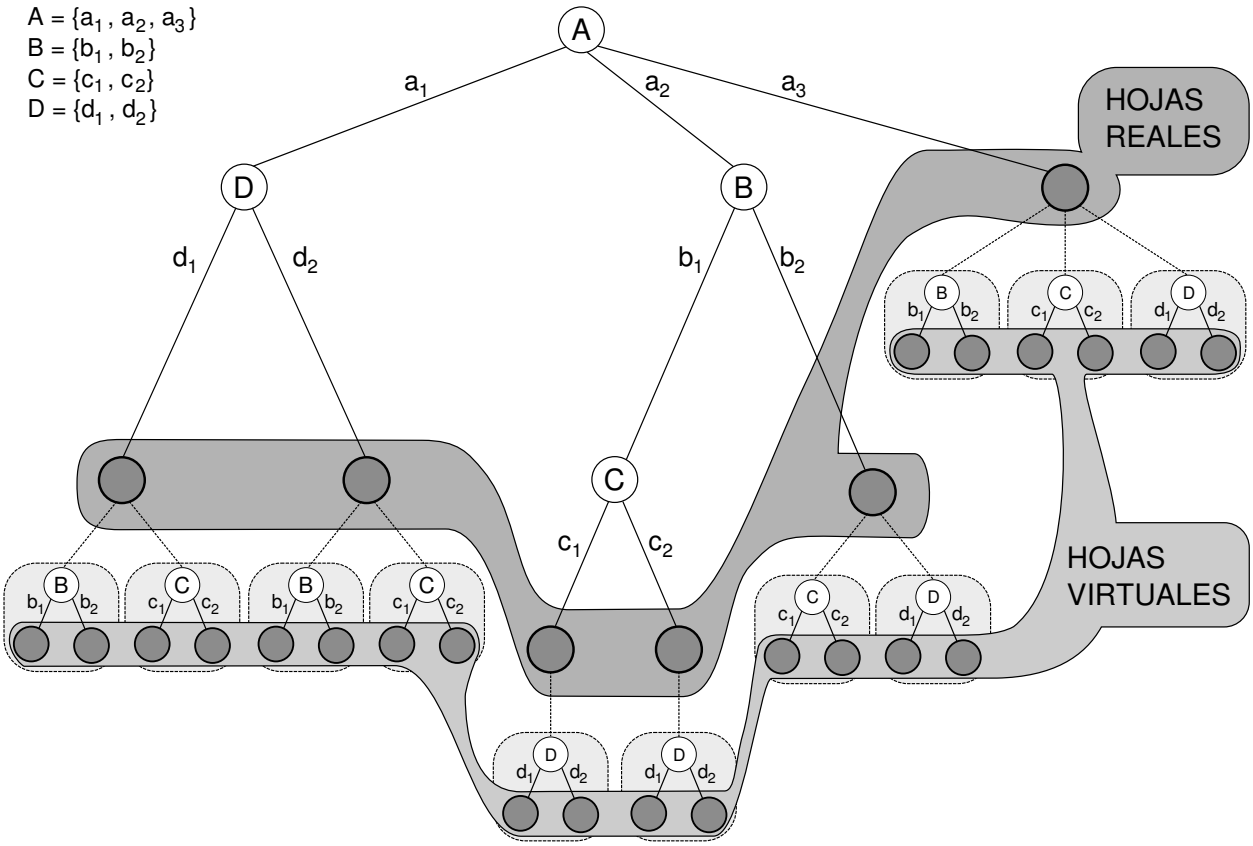



Figura 2.1: Representación de un árbol de decisión inducido por IADEM-0.

el error máximo tolerado ($\varepsilon$) para el problema que está afrontando, indicando también la confianza deseada para la estimación ($1-\delta$). Por tanto, las únicas entradas necesarias para que el algoritmo IADEM-0 induzca un árbol de decisión son el error máximo tolerado ($\varepsilon$), la confianza exigida ($1-\delta$) y el conjunto de datos ($E$) del que se desea extraer conocimiento.

Además de las entradas que sirven para determinar la calidad deseada para la salida, el algoritmo usa internamente una serie de parámetros, que configuran diversos aspectos de la inducción del árbol de decisión y cuya función iremos detallando en la presentación formal del algoritmo. Debemos hacer notar que los valores para dichos parámetros pueden modificarse, pero los valores que tienen por defecto han sido seleccionados tras un intenso estudio cuya finalidad ha sido ofrecer un comportamiento adecuado del algoritmo para el mayor número posible de situaciones. Los aspectos más interesantes de dicho estudio se presentan en la sección 2.3.

Las acciones propias del algoritmo están controladas por una serie de predicados, de los cuales, el más determinante es el que controla que la precisión estimada del árbol sea la deseada (*Condición_Parada*). Para ello considera el error máximo tolerado por el usuario ($\varepsilon$) y la estimación del error cometido por el árbol en el peor caso. El segundo motivo para detener la inducción del árbol es que el árbol no pueda seguir expandiéndose (*Condición_Completamente_Expandido*), es decir, que no queden más atributos libres por los que expandir. En ese caso el árbol estaría completo y no tendría sentido seguir muestreando puesto que el árbol inducido no cambiaría. Debemos indicar que no es habitual que se produzca esta situación y los casos más susceptibles de provocarla son aquellos en los que hay un nivel de ruido en el conjunto de datos mayor que el error máximo tolerado por el usuario. Los otros dos predicados que aún no hemos presentado son *Condición_Expansión* y *Condición_Es_Expansible* y son usados para determinar la idoneidad de expandir el árbol en un determinado momento. La primera condición está diseñada para no expandir bajo cualquier circunstancia, retrasar la expansión a momentos en los que sea más útil y conseguir así árboles de decisión de menor tamaño. La segunda condición afecta únicamente a la hoja seleccionada para expandir el árbol

**Entrada:** $\varepsilon$ (error máximo tolerado), $\delta$ (confianza), $E$ (conjunto de datos)

1. INICIALIZAR
2. **mientras** $\neg$*Condición_Parada* $\wedge$
   $\neg$*Condición_Completamente_Expandido* **hacer:**
2.1. MUESTREAR_Y_RECALCULAR
2.2. **si** $\neg$*Condición_Parada* $\wedge$
   *Condición_Expansión* $\wedge$
   *Condición_Es_Expansible*(*peor_hoja*) **entonces:**
2.2.1. EXPANDIR_ÁRBOL

**Salida:** $HOJAS$ (árbol de decisión)

Figura 2.2: Pseudocódigo del algoritmo IADEM-0.

($peor\_hoja$) y comprueba que exista suficiente evidencia de que el mejor atributo para expandir la hoja ha sido claramente identificado. De esta forma garantizamos que la expansión será de calidad y, aunque no nos garantice que se produzca un árbol óptimo, normalmente se inducirá un árbol cercano al óptimo.

## 2.2. Descripción formal del algoritmo IADEM-0

En esta sección presentamos en detalle la estructura usada para almacenar la representación probabilística en el árbol de decisión, y las definiciones de los procedimientos y predicados que componen el algoritmo IADEM-0. Sin pérdida de generalidad, y para facilitar la definición de los distintos componentes del algoritmo, la representación del árbol de decisión se ha realizado manteniendo únicamente sus hojas (reales y virtuales). En ellas se almacena tanto la información estructural para representar el árbol como los contadores necesarios para realizar las tareas propias del algoritmo.

El conjunto de todas las hojas reales que definen el árbol se llama $HOJAS$, siempre contendrá algún elemento ($|HOJAS| \in \mathbb{N}^+$), asumiremos que las hojas están identificadas por un índice y que éstos forman una secuencia consecutiva de naturales, es decir, $HOJAS \subset \mathbb{N}^+$. Recordemos que las hojas reales son las que forman la frontera real del árbol que se ha inducido hasta el momento. En adelante, cuando usemos el término "*hojas*" nos referiremos a "*hojas reales*".

Asociada a cada hoja existirá una serie de hojas virtuales y al conjunto de todas las hojas virtuales lo denominaremos $VIRTUALES$. Cada hoja virtual está definida por tres elementos: el índice de la hoja real de la que depende ($i \in HOJAS$), el índice de un atributo no usado en la hoja de la que depende ($l$) y uno de los valores de ese atributo ($v$). Usando las hojas virtuales conseguimos almacenar todas las posibles opciones que resultarían de expandir cualquier hoja por cualquier atributo aún no usado (sólo consideramos un nivel de profundidad adicional).

La información almacenada en las hojas reales y virtuales ("*componentes básicos*" en la subsección 2.2.2) será usada para calcular los intervalos estimados para diferentes variables ("*componentes calculados*" en la subsección 2.2.3). Estas variables serán usadas para evaluar los predicados del algoritmo (subsección 2.2.4) y ejecutar los procedimientos (subsección 2.2.5). Finalmente, una vez se disponga del modelo, además de poder ser usado para extraer conocimiento, podrá ser usado como un sistema de predicción (subsección 2.2.6).

Los procesos para calcular los diferentes componentes y predicados o ejecutar los procedimientos pueden realizarse mediante programación lineal [Win-1994]. Por simplificar la presentación de las definiciones hemos usado fórmulas matemáticas que serían fácilmente transformables para implementar un programa usando programación lineal (por ejemplo, el algoritmo SIMPLEX [Dan-1997]). Nuestra intención no es tender hacia ningún tipo de implementación en concreto, sino facilitar la comprensión del algoritmo. De

hecho, IADEM-0 ha sido implementado utilizando una aproximación específica computacionalmente más eficiente.

### 2.2.1. Definiciones previas: problema, universo de experiencias y de observaciones

Antes de describir los componentes que definen el algoritmo daremos algunas definiciones previas que serán necesarias y donde fijaremos algunos aspectos de notación. El primer concepto es la definición de $problema$ que, en la versión inicial de IADEM-0, sólo considera atributos categóricos multivaluados (este aspecto será ampliado en una de las extensiones presentadas en el capítulo 3). A continuación, basándonos en la definición de $problema$, definiremos formalmente los conceptos de $experiencia$ y $observación$, que serán los elementos usados para desarrollar las tareas de "*clasificación*" y "*predicción*". Debemos hacer notar que el objetivo de la "*clasificación*" (aprendizaje o extracción de conocimiento) es construir el árbol de decisión con menor tamaño posible que mejor represente al conjunto de experiencias dado, mientras que el objetivo de la "*predicción*" es asignar una clase a cada nueva observación, intentando reducir al mínimo las equivocaciones.

#### Problema

Un $problema$ con $a$ atributos puede definirse como un par $(\vec{m}, k) \in \mathbb{N}^a \times \mathbb{N}$, donde $\vec{m} = (m_1, \ldots, m_a)$. Cada $atributo\ x_i$ se define en un dominio $X_i = \{1, 2, \ldots, m_i\}$, donde $i = 1, \ldots, a$. Además de estos atributos, definimos otro llamado $x_0$, cuyo dominio es $X_0 = \{0\}$, y se usará para evitar funciones parciales. Existe un atributo especial llamado $clase$ que notamos como $c$ y cuyo dominio es $C = \{1, 2, \ldots, k\}$, donde $k \geq 2$.

Como ejemplo, podemos considerar un problema con tres atributos donde $x_1$ es la cantidad de lluvia registrada, $x_2$ es la humedad del suelo y $x_3$ describe la orografía. El atributo de clase $c$ determina el tipo de riesgo. De este modo, el problema es del tipo $(\vec{m}, 3)$ con $\vec{m} = (2, 3, 3)$:

$x_1 \in X_1$ donde $X_1 = \{intensa, escasa\}$
$x_2 \in X_2$ donde $X_2 = \{empapado, húmedo, seco\}$
$x_3 \in X_3$ donde $X_3 = \{escarpado, pendiente, plano\}$
$c \in C$ donde $C = \{serio, menor, nulo\}$

#### Universo de experiencias

Dado un problema, podemos definir el universo de experiencias ($U_E$) como $U_E = X_1 \times \ldots \times X_a \times C$ y una $experiencia$ ($e$) como un elemento de $U_E$. De esta forma $e = (x_1(e), x_2(e), \ldots, x_a(e), c(e)) \in U_E$, donde $x_i(e)$ representa el valor del atributo $i$ en la experiencia $e$ y $c(e)$ es la clase que le corresponde a la experiencia. Para evitar funciones parciales definimos $x_0(e) = 0\ \forall e \in U_E$.

Trabajaremos con secuencias finitas de experiencias ($E$), así $E = \{e_1, e_2, \ldots, e_N\}$, donde los elementos pueden repetirse. $\mathbf{E}$ representa el conjunto de todas las posibles secuencias de experiencias.

Continuando con el ejemplo anterior, podemos considerar las experiencias $e_1 = (intensa, empapado, escarpado, serio)$ o $e_2 = (escasa, húmedo, escarpado, nulo)$ como dos posibles experiencias para el problema.

#### Universo de observaciones

Dado un problema, podemos definir el universo de observaciones ($U_O$) como $U_O = X_1 \times \ldots \times X_a$. Por lo tanto, el universo de experiencias puede obtenerse del universo de observaciones ($U_E = U_O \times C$). Una $observación$ ($o$) es un elemento de $U_O$. De esta forma $o = (x_1(o), x_2(o), \ldots, x_a(o)) \in U_O$, donde $x_i(o)$ representa el valor del atributo $i$ en la observación $o$ donde $i = 1, \ldots, a$. Como hicimos anteriormente, para evitar funciones parciales definimos $x_0(o) = 0\ \forall o \in U_O$.

Continuando con el ejemplo anterior, podemos considerar las observaciones $o_1 = (escasa, seco, plano)$ o $o_2 = (intensa, empapado, pendiente)$ como dos posibles observaciones para el problema.

### 2.2.2. Componentes básicos

Los elementos que presentamos a continuación se usan para diferentes tareas: representar el árbol a partir de las hojas y contabilizar diferentes cantidades en hojas reales y virtuales. Debido a esa separación en las tareas hemos dividido los componentes básicos en dos grupos: elementos estructurales y elementos contadores. Recordemos que el conjunto de hojas reales que forman el árbol ha sido notado como $HOJAS \subset \mathbb{N}^+$ y el conjunto de hojas virtuales como $VIRTUALES = \{(i, l, v) \in \mathbb{N}^3 \mid i \in HOJAS,\, l \in atributos\_libres_i,\, v \in X_l\}$.

#### Elementos estructurales

Antes de comenzar la descripción de los elementos estructurales definidos en una hoja debemos explicar el uso que le hemos dado al número 0 para evitar funciones parciales. Para alcanzar el nodo raíz del árbol no se usa ningún atributo y, considerando ese particularidad, hemos usado el atributo $x_0$ para representar al atributo usado en el nivel 0 del árbol, que se corresponde con la raíz.

A continuación detallamos los elementos básicos almacenados en las hojas necesarios para definir la estructura del árbol:

- Conjunto de índices de los atributos usados desde el nodo raíz hasta la hoja $i$. Hemos incluido el 0 para evitar funciones parciales.

  $atributos\_usados_i \subseteq \{0, 1, \ldots, a\}$ donde $0 \in atributos\_usados_i$

- Conjunto de índices de los atributos libres en la hoja $i$.

  $atributos\_libres_i = \{0, 1, \ldots, a\} - atributos\_usados_i$

- Profundidad (nivel) de la hoja $i$.

  $N_i \in \{0, 1, \ldots, a\}$

- Función que ordena los atributos usados en la rama que llega a la hoja $i$ dependiendo de la profundidad

  $A_i : \{0, 1, \ldots, N_i\} \rightarrow atributos\_usados_i$ donde $A_i(0) = 0$

- Valor usado del atributo $u$ en la rama que lleva a la hoja $i$. Esta es la etiqueta que corresponde al arco que sale del nodo interno en el que se usó el atributo $u$ y que pertenece a la rama cuyo extremo es la hoja $i$.

  $V_{i,u} \in X_u$ donde $u \in atributos\_usados_i$

#### Elementos contadores

Los elementos que presentamos a continuación llevan la cuenta de las ocurrencias de diversos eventos desde que una hoja virtual es creada o desde que una hoja virtual es redefinida como real. Como ya se comentó anteriormente, la idea de contabilizar los eventos ocurridos desde que una hoja virtual se crea es transferir dicha información a las nuevas hojas reales que resultan de redefinir hojas virtuales en el proceso de expansión. Los contadores definidos son:

- Número de experiencias muestreadas desde que la hoja $i$ existe como virtual.

  $t_i \in \mathbb{N}$

- Número de experiencias muestreadas desde que la hoja $i$ existe como virtual y que, además, se corresponden con la rama que alcanza esa hoja ($x_{A_i(d)}(e) = V_{i,x_{A_i(d)}}\ \forall d \in \{0, \ldots, N_i\}$).

  $r_i \in \mathbb{N}$

- Número de experiencias muestreadas desde que la hoja $i$ existe como virtual, que se corresponden con la rama que alcanza esa hoja y que, además, están etiquetadas con la etiqueta de clase $z$ ($c(e) = z$).

  $r_{i,z} \in \mathbb{N}$ donde $z \in C$

  Por lo tanto, $r_i = r_{i,1} + \ldots + r_{i,k}$.

- Número de experiencias muestreadas desde que la hoja $i$ se redefine como real (número que será menor o igual que $t_i$). Este contador también puede definirse como el número de experiencias muestreadas desde que las hojas virtuales de la hoja $i$ son creadas.

  $total_i \in \mathbb{N}$

- Número de experiencias muestreadas desde que la hoja $i$ se redefine como real y que, además, se corresponden con la rama que alcanza esa hoja (número que será menor o igual que $\leq r_i$). Este contador también puede definirse como el número de experiencias muestreadas desde el momento en que las hojas virtuales de la hoja $i$ son creadas y que, además, se corresponden con la rama que alcanza dicha hoja $i$.

  $rama_i \in \mathbb{N}$

- Número de experiencias muestreadas desde que la hoja $i$ se redefine como real, que se corresponden con la rama que alcanza esa hoja ($rama_i \leq r_i$) y que, además, están etiquetadas con la etiqueta de clase $z$ ($c(e) = z$).

  $rama_{i,z} \in \mathbb{N}$

  Por lo tanto, $rama_i = rama_{i,1} + \ldots + rama_{i,k}$

- Número de experiencias muestreadas que se corresponden con la rama que llega a la hoja virtual $(i, l, v)$ desde que ésta es creada.

  $r'_{(i,l,v)} \in \mathbb{N}$ donde $(i, l, v) \in VIRTUALES,\ l \in atributos\_libres_i$ y $v \in X_l$

  Por lo tanto, $rama_i = r'_{(i,l,1)} + \ldots + r'_{(i,l,m_l)}$ para cualquier $l \in atributos\_libres_i$.

- Número de experiencias muestreadas que se corresponden con la rama que llega a la hoja virtual $(i, l, v)$ desde que ésta es creada y que, además, están etiquetadas con la etiqueta de clase $z$.

  $r'_{(i,l,v),z} \in \mathbb{N}$ donde $(i, l, v) \in VIRTUALES$ y $z \in C$

  Por lo tanto, $r'_{(i,l,v)} = r'_{(i,l,v),1} + \ldots + r'_{(i,l,v),k}$ y $rama_{i,z} = r'_{(i,l,1),z} + \ldots + r'_{(i,l,m_l),z}$ para cualquier $l \in atributos\_libres_i$.

Como se puede comprobar, alguno de los contadores anteriormente expuestos ($r_i$, $rama_i$, $rama_{i,z}$ y $r'_{(i,l,v)}$) son redundantes, pero han sido definidos para hacer más simple la descripción del algoritmo.

### 2.2.3. Componentes calculados

Los componentes que presentamos a continuación se calculan a partir de los que acabamos de definir. Los cálculos se realizan cada vez que se muestrean nuevas experiencias o cada vez que se produce una expansión y son empleados para evaluar los predicados. En la sección 1.4 presentamos el uso combinado de las cotas de concentración de Chernoff y Hoeffding para calcular el intervalo donde se encontrarán los valores de diferentes variables. La ecuación definida en esa sección, $margen\_\varepsilon$ (ver ecuación 1.22 en la página 22), será la que empleemos para calcular los componentes calculados que a continuación enumeramos. Comenzaremos con una sucesión de cuatro componentes para llegar a estimar el error del árbol:

**Probabilidad de que una experiencia alcance la hoja $i$:** $W_i = (\text{ínf}(w_i), \sup(w_i))$

La probabilidad de que una experiencia alcance una determinada hoja ($i$) vendrá caracterizada por el número de experiencias que hayan llegado previamente ($r_i$) y por el total que se han procesado ($t_i$), siendo su valor estimado $r_i/t_i$. Este elemento calculado se define como un par en el que se incluyen los dos extremos del intervalo que captura todos los posibles valores que puede tener la probabilidad que se busca.

$W_i \in [0,1]^2$ donde $i \in HOJAS$
$W_i = (\text{ínf}(w_i), \sup(w_i))$ donde:

$$\begin{aligned} \text{ínf}(w_i) &= \begin{cases} \text{máx}\{0, r_i/t_i - margen\_\varepsilon(t_i, r_i/t_i, \delta)\} & \text{si } t_i \neq 0 \\ 0 & \text{si } t_i = 0 \end{cases} \\ \sup(w_i) &= \begin{cases} \text{mín}\{1, r_i/t_i + margen\_\varepsilon(t_i, r_i/t_i, \delta)\} & \text{si } t_i \neq 0 \\ 1 & \text{si } t_i = 0 \end{cases} \end{aligned}$$

**Probabilidad de que una experiencia que alcance la hoja $i$ esté etiquetada con la clase $z$:**
$P_{i,z} = (\text{ínf}(p_{i,z}), \sup(p_{i,z}))$

La probabilidad de que una experiencia alcance una hoja ($i$) y tenga la etiqueta de clase $z$ vendrá determinada por el número de experiencias que hayan llegado previamente a esa hoja con esa clase ($r_{i,z}$) y por el número de experiencias que hayan alcanzado dicha hoja ($r_i$), siendo su valor estimado $r_{i,z}/r_i$. Este elemento calculado se define como un par en el que se incluyen los dos extremos del intervalo que captura todos los posibles valores que puede tener la probabilidad que se busca.

$P_{i,z} \in [0,1]^2$ donde $i \in HOJAS$ y $z \in C$
$P_{i,z} = (\text{ínf}(p_{i,z}), \sup(p_{i,z}))$ donde:

$$\begin{aligned} \text{ínf}(p_{i,z}) &= \begin{cases} \text{máx}\{0, r_{i,z}/r_i - margen\_\varepsilon(r_i, r_{i,z}/r_i, \delta)\} & \text{si } r_i \neq 0 \\ 0 & \text{si } r_i = 0 \end{cases} \\ \sup(p_{i,z}) &= \begin{cases} \text{mín}\{1, r_{i,z}/r_i + margen\_\varepsilon(r_i, r_{i,z}/r_i, \delta)\} & \text{si } r_i \neq 0 \\ 1 & \text{si } r_i = 0 \end{cases} \end{aligned}$$

**Probabilidad de la clase mayoritaria en la hoja $i$:** $Q_i = (\text{ínf}(q_i), \sup(q_i))$

El elemento $Q_i$ es un par definido por dos elementos que delimitan el intervalo donde se encuentra la probabilidad de la clase mayoritaria en la hoja $i$. Los dos elementos son la cota inferior ($\text{ínf}(q_i)$) y la cota superior ($\sup(q_i)$).

Antes de dar la definición de cada una de las cotas debemos destacar que $\sup(q_i)$ no es igual a la máxima probabilidad de que un ejemplo sea clasificado como de la clase $z$ ni $\text{ínf}(q_i)$ es la mínima probabilidad. La principal restricción es que las probabilidades deben permanecer dentro de sus límites y que la suma de las diferentes combinaciones siempre debe ser 1.

Sea $Y_i$ un conjunto de vectores. Cada vector perteneciente a $Y_i$ se corresponde con un posible vector de clasificación en la hoja $i$. Así, $Y_i$ recoge todos los posibles vectores de clasificación que pueden darse en la hoja $i$. Cada vector en $Y_i$ está definido por $k$ componentes (recordamos que $k$ es el número de clases en el problema) y a cada una de ellas le corresponde un valor restringido en el intervalo definido por $[\text{ínf}(p_{i,j}), \sup(p_{i,j})]$. Además, la suma de las $k$ componentes de cada vector debe sumar 1 (la probabilidad conjunta de todas las opciones).

$$Y_i = \left\{ (y_{i,1}, \ldots, y_{i,k}) \in [0,1]^k \quad | \quad \sum_{j=1}^{k} y_{i,j} = 1 \ \wedge \ y_{i,j} \in [\text{ínf}(p_{i,j}), \sup(p_{i,j})] \quad \forall j \in \{1, \ldots, k\} \right\}$$

Sea $M_{i,j}$ el conjunto de todas las probabilidades de la $j$-ésima clase que se dan cuando dicha clase es la mayoritaria en la hoja $i$:

$$M_{i,j} = \{y_{i,j} \mid \vec{y}_i \in Y_i \ \wedge \ y_{i,j} \geq y_{i,t} \ \forall t \in \{1, \ldots, k\}\}$$

Sea $M_i$ la unión de todos los valores de probabilidad que alcanzan las clases cuando son mayoritarias en la hoja $i$:

$$M_i = \bigcup_{j=1}^{k} M_{i,j}$$

Ahora podemos definir las cotas inferior y superior para la probabilidad de la clase mayoritaria en la hoja $i$:

$$\begin{aligned} \text{ínf}(q_i) &= \text{mín}\, M_i \\ \sup(q_i) &= \text{máx}\, M_i \end{aligned}$$

**Error de clasificación estimado del árbol de decisión:** $error = (\text{ínf}(error), \sup(error))$

El elemento $error$ también se define como un par constituido por dos elementos que determinan las cotas estimadas para el error de clasificación cometido por el árbol de decisión. Dichas cotas son la cota inferior ($\text{ínf}(error)$) y la cota superior ($\sup(error)$). Se pueden usar diferentes enfoques para estimar el error que comete un árbol de decisión: basándose en el tipo de decisiones tomadas para inducirlo [Lan-2003], suponiendo que el error de clasificación seguirá una distribución binomial [Lan-2005] o utilizando la estructura alcanzada (anchura y profundidad) [Pic-2007]. En nuestro caso, para calcular las cotas que definen la estimación del error, usaremos dos de los elementos calculados descritos anteriormente: la probabilidad de que una experiencia alcance una hoja y la probabilidad de la clase mayoritaria.

Básicamente, $error$ se calcula sumando los errores individuales cometidos por cada hoja buscando la configuración del mejor caso para la cota inferior y del peor caso para la cota superior.

Sea $VECTORES\_ERROR$ un conjunto de vectores de error. Cada uno de estos vectores proporciona el error cometido por el conjunto de hojas que definen un árbol de decisión. La estructura del árbol de decisión cuyo error queremos estimar es única, pero los valores para ciertas variables en las hojas del árbol no son únicos: se encuentran definidos por intervalos. De este modo, aún teniendo una única estructura, podemos decir que nos encontramos con una gran variedad de árboles diferentes debido a que los valores de las variables no son únicos. Para cada una de las posibles combinaciones de valores en las variables, tenemos un árbol diferente, y para cada árbol existe un vector cuyas componentes son el error aportado por cada una de sus hojas. Cada vector tiene que cumplir una serie de restricciones: la suma de las probabilidades de alcanzar una hoja debe ser 1 y los valores seleccionados para las probabilidades de alcanzar una hoja ($w_i$) y de la clase mayoritaria ($q_i$) deben permanecer en sus intervalos. El conjunto $VECTORES\_ERROR$ recoge todos los posibles vectores con el error aportado por las hojas en todas las posibles configuraciones:

$$\begin{aligned} VECTORES\_ERROR = \Big\{ (y_1, \ldots, y_{|HOJAS|}) \in [0,1]^{|HOJAS|} \quad | \quad & y_i = w_i \cdot (1 - q_i) \ \wedge \\ & \sum_{i=1}^{|HOJAS|} w_i = 1 \ \wedge \\ & w_i \in [\text{ínf}(w_i), \sup(w_i)] \ \wedge \\ & q_i \in [\text{ínf}(q_i), \sup(q_i)] \\ & \forall i \in \{1, \ldots, |HOJAS|\} \Big\} \end{aligned}$$

Sea $VALORES\_ERROR$ el conjunto de todos los valores posibles para el error cometido en la clasificación por un árbol de decisión. Los valores se calculan sumando todos los componentes de los diferentes

vectores de error:

$$VALORES\_ERROR = \left\{ \sum_{i=1}^{|HOJAS|} y_i \mid \vec{y} \in VECTORES\_ERROR \right\}$$

Ahora podemos definir las cotas inferior y superior para el error de clasificación estimado del árbol de decisión ($error$):

$$\text{ínf}(error) = \text{mín } VALORES\_ERROR$$
$$\sup(error) = \text{máx } VALORES\_ERROR$$

Una vez visto el error del árbol, presentamos los componentes usados en el proceso de expansión:

**Probabilidad de que una experiencia que alcance la hoja virtual $(i,l,v)$ esté etiquetada con la clase $z$:**
$P_{(i,l,v),z} = (\text{ínf}(p_{(i,l,v),z}), \sup(p_{(i,l,v),z}))$

La probabilidad de que una experiencia alcance una hoja virtual ($(i,l,v)$) y tenga la etiqueta de clase $z$ vendrá determinada por el número de experiencias que hayan llegado previamente a esa hoja virtual con esa clase ($r'_{(i,l,v),z}$) y por el número de experiencias que hayan alcanzado dicha hoja virtual ($r'_{(i,l,v)}$), siendo su valor estimado $r'_{(i,l,v),z}/r'_{(i,l,v)}$. Este elemento calculado se define como un par en el que se incluyen los dos extremos del intervalo que captura todos los posibles valores que puede tener la probabilidad que se busca.

$P_{(i,l,v),z} \in [0,1]^2$ donde $(i,l,v) \in VIRTUALES$ y $z \in C$
$P_{(i,l,v),z} = (\text{ínf}(p_{(i,l,v),z}), \sup(p_{(i,l,v),z}))$ donde:

$$\text{ínf}(p_{(i,l,v),z}) = \begin{cases} \text{máx}\{0, r'_{(i,l,v),z}/r'_{(i,l,v)} - margen\_\varepsilon(r'_{(i,l,v)}, r'_{(i,l,v),z}/r'_{(i,l,v)}, \delta)\} & \text{si } r'_{(i,l,v)} \neq 0 \\ 0 & \text{si } r'_{(i,l,v)} = 0 \end{cases}$$

$$\sup(p_{(i,l,v),z}) = \begin{cases} \text{mín}\{1, r'_{(i,l,v),z}/r'_{(i,l,v)} + margen\_\varepsilon(r'_{(i,l,v)}, r'_{(i,l,v),z}/r'_{(i,l,v)}, \delta)\} & \text{si } r'_{(i,l,v)} \neq 0 \\ 1 & \text{si } r'_{(i,l,v)} = 0 \end{cases}$$

**Medida de desorden en la hoja $i$ dado el atributo $l$:**
$medida_i(l) = (\text{ínf}(medida_i(l)), \sup(medida_i(l)))$

El algoritmo que presentamos puede trabajar, al igual que hacen BOAT [Geh-1999] y otros algoritmos, con cualquier medida de desorden para identificar cuál es el mejor atributo para realizar una expansión. La medida de desorden es una función externa definida como:

$$medida : [0,1]^k \rightarrow \mathbb{R}^+ \cup \{0\}$$

y que podría ser configurada como un parámetro interno del algoritmo. En nuestro caso hemos seleccionado la "*entropía*" [Sha-1948; Pea-1979] como medida de desorden puesto que el comportamiento observado es satisfactorio y es una medida ampliamente utilizada en el campo de la minería de datos [Qui-1986; Qui-1993; Ram-2005c].

Sea $MEDIDAS_{(i,l,v)}$ el conjunto de todos los posibles valores para la medida de desorden en la hoja virtual $(i,l,v)$:

$$MEDIDAS_{(i,l,v)} = \{medida(\vec{y}) \mid \vec{y} \in Y_{(i,l,v)}\}$$

donde

$$Y_{(i,l,v)} = \Big\{ (y_{(i,l,v),1}, \ldots, y_{(i,l,v),k}) \in [0,1]^k \quad | \quad \sum_{t=1}^{k} y_{(i,l,v),t} = 1 \ \wedge \ y_{(i,l,v),t} \in [\text{ínf}(p_{(i,l,v),t}), \sup(p_{(i,l,v),t})] \ \forall t \in \{1,\ldots,k\} \Big\}$$

Ahora podemos definir las cotas inferior y superior de la medida de desorden en una hoja ($i$) dado un atributo ($l$):

$$\text{ínf}(medida_i(l)) = \frac{\sum_{v=1}^{m_l} \left( (\text{mín } MEDIDAS_{(i,l,v)}) \cdot r'_{(i,l,v)} \right)}{\sum_{v=1}^{m_l} r'_{(i,l,v)}}$$

$$\sup(medida_i(l)) = \frac{\sum_{v=1}^{m_l} \left( (\text{máx } MEDIDAS_{(i,l,v)}) \cdot r'_{(i,l,v)} \right)}{\sum_{v=1}^{m_l} r'_{(i,l,v)}}$$

**Atributo usado para expandir la hoja $i$:** $mejor\_atributo_i$

Antes de que algoritmo pueda decidir si la hoja $i$ puede expandirse necesitará identificar cuál será el atributo por el que se expandirá. Ese atributo no debe haber sido usado previamente (pertenecerá a los atributos sin usar) y será el que, considerando la medida de desorden, mayor mejora consiga.

Sea $MEJORES\_ATRIBUTOS_i$ el conjunto de los atributos más prometedores. Éstos serán aquellos atributos que tengan la menor medida de desorden en el peor caso ($\sup(medida_i(l))$):

$$\begin{aligned} MEJORES\_ATRIBUTOS_i = \quad \{ \quad & l \in atributos\_libres_i \mid \\ & \sup(medida_i(l)) \leq \sup(medida_i(l')) \\ & \forall l' \in atributos\_libres_i \} \end{aligned}$$

El mejor atributo se selecciona de entre los anteriores y será aquél cuya medida de desorden sea menor en el mejor caso ($\text{ínf}(medida_i(l))$):

$$\begin{aligned} mejor\_atributo_i = \text{mín} \quad \{ \quad & l \in MEJORES\_ATRIBUTOS_i \mid \\ & \text{ínf}(medida_i(l)) \leq \text{ínf}(medida_i(l')) \\ & \forall l' \in MEJORES\_ATRIBUTOS_i \} \end{aligned}$$

**Hoja que más contribuye al error de clasificación del árbol:** $peor\_hoja$

Denominamos $peor\_hoja$ a la hoja que mayor error introduce en el árbol de decisión. Esta será la hoja seleccionada para ser expandida haciendo que el beneficio sea máximo.

Sea $sup(\varepsilon_i)$ el mayor error cometido por la hoja $i$ del árbol:

$$\sup(\varepsilon_i) = \text{máx}\{y_i \mid \vec{y} \in Y\} \cdot \ (1 - inf(q_i))$$

donde

$$\begin{aligned} Y = \Big\{ (y_1, \ldots, y_{|HOJAS|}) \in [0,1]^{|HOJAS|} \quad \mid \quad & \sum_{t=1}^{|HOJAS|} y_t = 1 \ \wedge \\ & y_t \in [\text{ínf}(w_t), \sup(w_t)] \\ & \forall t \in \{1, \ldots, |HOJAS|\} \Big\} \end{aligned}$$

Puesto que $HOJAS$ está definido como $HOJAS \subset \mathbb{N}^+$, el 0 no forma parte de su conjunto y lo empleamos para indicar que no existe ninguna hoja que pueda ser considerada la peor. Por tanto, definimos la peor hoja ($peor\_hoja \in HOJAS \cup \{0\}$) como:

$$peor\_hoja = \text{mín}\,\{\{i \in HOJAS\_LIBRES \mid \sup(\varepsilon_i) \geq \sup(\varepsilon_j)\ \forall j \in HOJAS\_LIBRES\} \cup \{0\}\}$$

$$\text{donde } HOJAS\_LIBRES = \{i \in HOJAS \mid atributos\_libres_i \neq \emptyset\ \wedge\ rama_i > 0\}$$

### 2.2.4. Predicados

Los predicados controlan la ejecución del programa y sus principales cometidos son determinar el momento de detener la ejecución y permitir la expansión de hojas en el árbol de decisión. Existen cuatro predicados cuya finalidad ya se presentó en la sección 2.1 (página 26) y en el pseudocódigo de la figura 2.2 (página 29). En este apartado los definiremos de forma detallada utilizando los elementos calculados del apartado anterior.

Los predicados que describiremos son:

- *Condición_Parada*: evaluada sobre el árbol de decisión.
- *Condición_Completamente_Expandido*: evaluada sobre el árbol de decisión.
- *Condición_Expansión*: evaluada sobre el árbol de decisión.
- *Condición_Es_Expansible*: evaluada sobre una hoja del árbol de decisión.

#### *Condición_Parada*

Este predicado determina cuándo se alcanza la precisión mínima deseada por el usuario. Esa situación estará garantizada cuando el error de clasificación del árbol en el peor caso $\sup(error)$ esté por debajo del error máximo tolerado ($\varepsilon$).

$$Condición\_Parada : 2^{\mathbb{N}} \rightarrow \{V, F\}$$
$$Condición\_Parada(HOJAS) \equiv \sup(error) \leq \varepsilon$$

#### *Condición_Completamente_Expandido*

Este predicado es verdadero cuando el árbol ha sido completamente expandido y en las hojas no quedan más atributos libres por los que expandir. El único sentido de seguir muestreando experiencias sería ajustar aún más los intervalos en los que se mueven las variables definidas. Lo normal es que la ejecución del algoritmo no se detenga por este predicado (en los experimentos realizados no ha se ha producido nunca este caso), pero es necesario controlar todas las posibilidades.

$$Condición\_Completamente\_Expandido : 2^{\mathbb{N}} \rightarrow \{V, F\}$$
$$Condición\_Completamente\_Expandido(HOJAS) \equiv atributos\_libres_i = \emptyset \quad \forall i \in HOJAS$$

#### *Condición_Expansión*

Este predicado determina cuándo es posible expandir el árbol de decisión. Esto no quiere decir que la expansión se vaya a producir, porque para que se expanda también hay que comprobar que la hoja seleccionada sea expansible (*Condición_Es_Expansible*). El objetivo de este predicado es evitar el posible comportamiento asintótico del error de clasificación del árbol de decisión, que está acotado entre $\inf(error)$ y $\sup(error)$. Para no expandir bajo cualquier circunstancia e inducir árboles de decisión demasiado grandes de forma innecesaria, la primera opción que se puede plantear es permitir las expansiones sólo cuando el error de clasificación en el mejor caso $\inf(error)$ sea mayor que el error máximo permitido ($\varepsilon$).

Pero esto plantea el problema de que aparezca un comportamiento asintótico del intervalo de error alrededor del error máximo permitido. En ese caso el algoritmo no terminaría nunca dado que no se expandiría,

porque en el mejor caso no se superaría el error máximo permitido, ni se satisfarían los requisitos del usuario, porque el error en el peor caso no es menor que el máximo permitido. Para relajar esta condición y solventar el problema del comportamiento asintótico hemos definido una "*frontera de expansión*" (a partir de la cuál se podrá expandir) por debajo del error máximo permitido. La cercanía de esta frontera de expansión al valor de error máximo permitido se determina usando un parámetro interno del algoritmo que hemos llamado "*factor de expansión*" y lo hemos notado como $\gamma$. El valor de $\gamma$ está definido en el intervalo $(0, 1)$.

$$Condición\_Expansión : 2^{\mathbb{N}} \rightarrow \{V, F\}$$
$$Condición\_Expansión(HOJAS) \equiv \text{ínf}(error) \geq (1 - \gamma) \cdot \varepsilon$$

***Condición_Es_Expansible***

Este predicado, a diferencia de los tres anteriores, no se evalúa usando el árbol de decisión, sino una de sus hojas ($peor\_hoja$). El objetivo es detectar cuándo podemos afirmar que el mejor atributo para expandir una determinada hoja ($mejor\_atributo$) ha sido identificado. Como comentamos anteriormente, la elección de buenos atributos para expandir una hoja se convierte en una tarea fundamental para inducir árboles parecidos al óptimo. En este predicado incluimos dos condiciones para decidir cuándo un atributo se ha diferenciado claramente como el mejor entre todos: cuando la medida de desorden en el peor caso para ese atributo ($\sup(medida_i(mejor\_atributo_i))$) es inferior a la medida de desorden para el resto de atributos o cuando la distancia entre esos valores es pequeña. Para determinar esa distancia máxima el algoritmo cuenta con el parámetro llamado "*diferenciación de atributos*" notado como $d$. El valor de $d$ está definido en el intervalo $(0, 1)$ y para valores pequeños de $d$ se producen las selecciones más restrictivas.

$$Condición\_Es\_Expansible : HOJAS \cup \{0\} \rightarrow \{V, F\}$$
$$Condición\_Es\_Expansible(i) \equiv (i \neq 0) \ \wedge Diferenciado\_Mejor\_atrib(i)$$

donde:

$$Diferenciado\_Mejor\_atrib(i) : HOJAS \rightarrow \{V, F\}$$
$$Diferenciado\_Mejor\_atrib(i) \equiv \Bigg( \Big[ \sup(medida_i(mejor\_atributo_i)) \leq \text{ínf}(medida_i(l)) \quad \forall l \in (atributos\_libres_i - \{mejor\_atributo_i\}) \Big] \vee \Big[ |\sup(medida_i(mejor\_atributo_i)) - \text{mín}\{\text{ínf}(medida_i(l)) : l \in (atributos\_libres_i - \{mejor\_atributo_i\})\}| \leq d \Big] \Bigg)$$

En caso de que no exista ninguna hoja que cumpla la condición de ser la $peor\_hoja$, la condición se evaluará con el valor 0 que representa esa circunstancia ($i = 0$). Bajo esas condiciones, el predicado dirá que no es expansible.

### 2.2.5. Procedimientos

Una vez hemos presentado los componentes básicos, los calculados y los predicados, la última parte que queda por definir para describir completamente el algoritmo son los procedimientos. IADEM-0, al ser un algoritmo incremental, parte de un modelo muy sencillo y lo hace evolucionar hasta que induce el modelo final que más se ajuste al conocimiento en el conjunto de datos. Recordemos que el pseudocódigo del algoritmo se presentó en la figura 2.2 (página 29) y necesitamos definir los siguientes procedimientos:

- INICIALIZAR
- MUESTREAR_Y_RECALCULAR
- EXPANDIR_ÁRBOL

**INICIALIZAR**

Este procedimiento sólo se usa una vez al principio de la ejecución y su función es crear la estructura inicial, compuesta por una única hoja real (cuyo índice será 1) y tantas hojas virtuales como atributos tiene la descripción del problema, y darles los valores iniciales a todas las variables.

$HOJAS = \{1\}$
$atributos\_usados_1 = \{0\}$
$atributos\_libres_1 = \{1, 2, \ldots, a\}$
$N_1 = 0$
$A_1(0) = 0$
$V_{1,0} = 0$
$t_1 = 0$
$r_1 = 0$
$r_{1,z} = 0 \; \forall z \in C$
$total_1 = 0$
$rama_1 = 0$
$rama_{1,z} = 0 \; \forall z \in C$
$r'_{(1,r,v)} = 0 \; \forall r \in atributos\_libres_1, \; \forall v \in X_r$
$r'_{(1,r,v),z} = 0 \; \forall r \in atributos\_libres_1, \; \forall v \in X_r, \; \forall z \in C$

**MUESTREAR_Y_RECALCULAR**

Mientras no se den las condiciones necesarias para detener la ejecución del algoritmo, el proceso de muestrear experiencias y actualizar las variables almacenadas en el árbol de decisión se repetirá. La captura de las experiencias se hace por muestreo con reemplazamiento, pero si se trata de un flujo de datos, la extracción se haría tomándolas secuencialmente (también se podrían almacenar en una ventana de tamaño finito y muestrear con reemplazamiento de ella). Existe la posibilidad de muestrear una a una las experiencias, pero el número de pasos de actualización podría ser muy elevado antes de que se produzca una expansión, con el consiguiente retraso en la inducción del modelo. Por ello hemos introducido un parámetro interno en el algoritmo que controla el número de experiencias que son muestreadas en el procedimiento MUESTREAR_Y_RECALCULAR y lo hemos notado como $n$. El valor de $n$ es un número natural positivo ($n \in \mathbb{N}^+$). Cada vez que se produce un muestreo, obtenemos una nueva secuencia de experiencias $E \in \boldsymbol{E}$ con $E = \{e_1, e_2, \ldots, e_n\}$.

Sea $Corresponde$ un predicado que indica si la experiencia $e$ alcanza la hoja $i$, es decir:

$$Corresponde(i, e) \;\equiv\; x_u(e) = V_{i,u} \quad \forall u \in atributos\_usados_i$$

donde $x_u(e)$ es el valor del atributo $u$ en la experiencia $e$ y $V_{i,u}$ es el valor del atributo $u$ en la hoja $i$.

Con las nuevas experiencias muestreadas, vendrá nueva información que será distribuida por el árbol de decisión que se está induciendo. Los elementos contadores son actualizados del siguiente modo:

$t_i = t_i + n$
$r_i = r_i + |\{e \in E \mid Corresponde(i, e)\}|$
$r_{i,z} = r_{i,z} + |\{e \in E \mid Corresponde(i, e) \wedge c(e) = z\}|$
$total_i = total_i + n$
$rama_i = rama_i + |\{e \in E \mid Corresponde(i, e)\}|$
$rama_{i,z} = rama_{i,z} + |\{e \in E \mid Corresponde(i, e) \wedge c(e) = z\}|$
$r'_{(i,l,v)} = r'_{(i,l,v)} + |\{e \in E \mid Corresponde(i, e) \wedge x_l(e) = v\}|$
$r'_{(i,l,v),z} = r'_{(i,l,v),z} + |\{e \in E \mid Corresponde(i, e) \wedge x_l(e) = v \wedge c(e) = z\}|$

El resto de variables pertenecientes a los componentes básicos, es decir, los elementos estructurales, permanecen inalteradas puesto que se corresponden con la estructura del árbol, y ésta no se modifica en el proceso de muestrear y recalcular.

Una vez se han actualizado los contadores, los componentes calculados (subsección 2.2.3) que dependen de ellos son recalculados. Hemos de precisar que, para ser estrictos, no todos los componentes calculados necesitan ser recalculados en este momento. Los relacionados con la expansión de una hoja pueden recalcularse una vez se haya decidido que se puede intentar realizar una expansión.

Todas las experiencias muestreadas son olvidadas, permitiendo así que los requisitos de memoria tan sólo dependan de la estructura que almacena la información relevante sin tener en cuenta cuántas experiencias se hayan muestreado.

#### EXPANDIR_ÁRBOL

La parte fundamental en el proceso de inducción del árbol de decisión es la expansión de las hojas. Como ya hemos comentado, este proceso únicamente se realiza cuando hay evidencias de que la expansión tiene ciertos niveles de calidad (controlado por los predicados *Condición_Expansión* y *Condición_Es_Expansible*). La expansión se realiza a partir de una hoja que es identificada como la hoja que mayor error de clasificación introduce en el árbol ($peor\_hoja$).

Antes de describir el proceso de expansión del árbol, detallaremos cómo se expande una hoja. La expansión de una hoja ($i$) es un proceso sencillo en el que las hojas virtuales que se corresponden con el atributo seleccionado para realizar la expansión ($mejor\_atributo_i$) pasan a ser hojas reales. Eso significa que parte de sus variables serán reasignadas de las variantes virtuales a sus variantes reales y que habrá que construir nuevas hojas virtuales para cada una de las nuevas hojas reales. Este proceso, en el que se definen nuevas hojas reales, podemos expresarlo del siguiente modo:

$expandir\_hoja : \mathbb{N} \rightarrow 2^{\mathbb{N}}$
$expandir\_hoja(i) = \{|HOJAS| + 1, \ldots, |HOJAS| + m_b\}$ donde: $i \in HOJAS$
$b = mejor\_atributo_i$

y $\forall j \in expandir\_hoja(i)$ hacemos: $atributos\_usados_j = atributos\_usados_i \cup \{b\}$
$atributos\_libres_j = atributos\_libres_i - \{b\}$
$N_j = N_i + 1$
$A_j(d) = \begin{cases} A_i(d) & \text{if } d < N_j \\ b & \text{if } d = N_j \end{cases}$
$V_{j,u} = V_{i,u}\ \forall u \in atributos\_usados_i$
$V_{j,b} = j - |HOJAS|$
$t_j = total_i$
$r_j = r'_{(i,b,j-|HOJAS|)}$
$r_{j,z} = r'_{(i,b,j-|HOJAS|),z}$
$total_j = 0$
$rama_j = 0$
$rama_{j,z} = 0\ \forall z \in C$
$r'_{(j,l,v)} = 0\ \forall l \in atributos\_libres_j,\ \forall v \in X_l$
$r'_{(j,l,v),z} = 0\ \forall l \in atributos\_libres_j,\ \forall v \in X_l,\ \forall z \in C$

Una vez hemos definido cómo expandir una hoja, podemos explicar cómo se expande el árbol. El proceso consiste en retirar la hoja que ha sido expandida del conjunto de hojas que mantiene la descripción del árbol ($HOJAS$) y añadir las nuevas hojas reales:

$expandir\_árbol : 2^{\mathbb{N}} \rightarrow 2^{\mathbb{N}}$
$expandir\_árbol(HOJAS) = \Big[HOJAS \cup expandir\_hoja(peor\_hoja)\Big] - peor\_hoja$

Tras el proceso de expansión, los componentes calculados deben ser recalculados de nuevo, puesto que ha habido modificación en ellos. En realidad, al igual que ocurría con el procedimiento de muestrear y recalcular, no todos los componentes calculados deben ser recalculados. Sólo es imprescindible recalcular aquellos relacionados con las hojas reales para evaluar el nuevo intervalo de error del árbol.

### 2.2.6. IADEM-0 como sistema de predicción

En este apartado presentamos los conceptos de "*predicción*" y "*sistema de predicción*" y cómo se puede usar el árbol de decisión inducido por IADEM-0 como un "*sistema de predicción*".

#### Universo de predicciones y sistema de predicción

Dado un problema, definimos el "*universo de predicciones*" ($U_P$) como:

$$U_P = \left\{(y_1, y_2, \ldots, y_k) \in [0,1]^k \quad | \quad \sum_{z=1}^{k} y_z = 1\right\}$$

Un "*sistema de predicción*" ($P$) lo definimos como una función $P : U_O \rightarrow U_P$. De tal modo que, si $o$ es una observación ($o \in U_O$) y $P$ es un sistema de predicción, nuestra predicción de que la observación $o$ sea etiquetada como de la clase $z$ es $P_z(o)$:

$$P(o) = (P_1(o), P_2(o), \ldots, P_k(o)) \in U_P$$
$$\text{donde } P_z(o) \in [0,1]\ , z = 1, \ldots, k \text{ y } \sum_{z=1}^{k} P_z(o) = 1$$

#### IADEM-0 como un sistema de predicción

Como resultado de la ejecución del algoritmo IADEM-0 obtenemos un árbol de decisión, que está definido usando las hojas que forman la frontera del árbol ($HOJAS$). Debemos señalar que hemos considerado dos situaciones diferentes: cuando la hoja tiene información y cuando no la tiene. En el primer caso, la predicción se hace usando la propia información. En el segundo caso, en el que la hoja no dispone de información para hacer ninguna predicción, se utiliza la información de las hojas con las que comparte el mismo padre (hermanos). Utilizar la información de los hermanos es mejor que no usar ninguna información. El árbol de decisión inducido por IADEM-0 y usado como sistema de predicción es denominado ${}_{HOJAS}P$:

${}_{HOJAS}P : U_O \rightarrow U_P$

${}_{HOJAS}P(o) = ({}_{HOJAS}P_1(o), {}_{HOJAS}P_2(o), \ldots, {}_{HOJAS}P_k(o))$ donde:

$${}_{HOJAS}P_z(o) = \begin{cases} \dfrac{r_{i,z}}{r_i} & \text{si } r_i > 0 \\ & \text{donde } x_u(o) = V_{i,u}\ \forall u \in atributos\_usados_i \\ \dfrac{\sum\limits_{j \in MISMO\_PADRE(i)} r_{j,z}}{\sum\limits_{j \in MISMO\_PADRE(i)} r_j} & \text{si } r_i = 0 \\ & \text{donde } x_u(b) = V_{i,u}\ \forall u \in atributos\_usados_i \end{cases}$$

con $MISMO\_PADRE(i) = \left\{l \in HOJAS \mid V_{l,A_l(j)} = V_{i,A_i(j)}\ \wedge\ j \in \{1, \ldots, N_i - 1\}\right\} - \{i\}$

Una característica relevante de IADEM-0 como sistema de predicción es que, además del vector con la predicción estimada para una observación, la información se acompaña con un vector de márgenes de error. Definimos ${}_{HOJAS}M$ como la función que, dada una observación, ofrece los márgenes de error para los valores estimados (que son calculados mediante ${}_{HOJAS}P$):

${}_{HOJAS}M : U_O \rightarrow \left([0,1]^2\right)^k$

${}_{HOJAS}M(o) = ({}_{HOJAS}M_1(o), {}_{HOJAS}M_2(o), \ldots, {}_{HOJAS}M_k(o))$ donde:

$$
{}_{HOJAS}M_z(o) = \begin{cases} (\text{ínf}(p_{i,z}), \sup(p_{i,z})) & \text{si } r_i > 0 \\ & \text{donde } x_u(o) = V_{i,u} \\ & \forall u \in atributos\_usados_i \\ (\text{máx}\,\{0, {}_{HOJAS}P_z(o) - estError(o)\}, & \text{si } r_i = 0 \\ \text{mín}\,\{1, {}_{HOJAS}P_z(o) + estError(o)\}) & \text{donde } x_u(o) = V_{i,u} \\ & \forall u \in atributos\_usados_i \end{cases}
$$

con $estError(o) = margen\_\varepsilon\Big(\sum_{j \in MISMO\_PADRE(i)} r_j,\ {}_{HOJAS}P_z(o), \delta\Big)$

y $\quad MISMO\_PADRE(i) = \Big\{l \in HOJAS \mid V_{l,A_l(j)} = V_{i,A_i(j)} \ \wedge\ j \in \{1, \ldots, N_i - 1\}\Big\} - \{i\}$

Debemos señalar que $\sum_{j \in MISMO\_PADRE(i)} r_j \neq 0$ y ${}_{HOJAS}P_z(o) \neq 0$.

Con esta información adicional que se aporta al valor estimado de la predicción para una observación concreta se pueden estudiar mecanismos más complejos para predecir que no se basen únicamente en la predicción estimada.

## 2.3. Estudio de los parámetros internos

A lo largo de la sección anterior, en la que hemos descrito con detalle el algoritmo IADEM-0, han surgido diversos parámetros internos cuyo funcionamiento estudiamos a continuación. No los consideramos como entradas al algoritmo porque sus valores por defecto han sido fijados después de realizar diversos estudios sobre su influencia y no son configurables por el usuario. Los tres parámetros son:

- Número de experiencias tomadas en cada muestra: $n \in \mathbb{N}^+$
- Factor de expansión: $\gamma \in (0, 1]$
- Diferenciación de atributos: $d \in [0, 1]$

Para determinar los valores por defecto que deberíamos emplear para alcanzar un comportamiento adecuado en la mayoría de las situaciones hemos realizado diversas pruebas, de las cuales vamos a mostrar las más representativas, donde se puede apreciar el comportamiento del algoritmo frente a diversos valores para dichos parámetros internos. La experimentación que presentamos ha sido el resultado de inducir árboles de decisión a partir de un conjunto de datos generados artificialmente cuyas características son las siguientes: el problema está caracterizado por 10 atributos ($a = 10$) con 4 valores para cada atributo ($m_i = 4\ \forall i \in \{1, \ldots, a\}$) y un atributo de clase con 3 posibles valores ($k = 3$); y el conjunto de datos está formado por un millón de experiencias sin ruido que han sido generadas usando un árbol de decisión creado aleatoriamente con 130 hojas. Los resultados que se presentan se han calculado después de realizar una validación cruzada con diez particiones para cada prueba. Los valores usados para la entrada al algoritmo han sido $\varepsilon = 0,01$ y $\delta = 0,05$.

### 2.3.1. Número de experiencias tomadas en cada muestra: $n$

La principal repercusión del número de experiencias muestreadas y analizadas cada vez que se ejecuta el procedimiento MUESTREAR_Y_RECALCULAR se aprecia en el tiempo requerido para finalizar la ejecución del algoritmo. En la figura 2.3 presentamos el comportamiento del algoritmo dependiendo del valor dado a $n$ considerando dos aspectos relevantes: el número total de experiencias muestreadas y el tiempo consumido antes de satisfacer los requisitos del usuario.

En este caso no mostramos la precisión final alcanzada ni el tamaño del árbol porque se mantienen constantes independientemente del valor que se le dé al parámetro interno $n$. Observando los indicadores

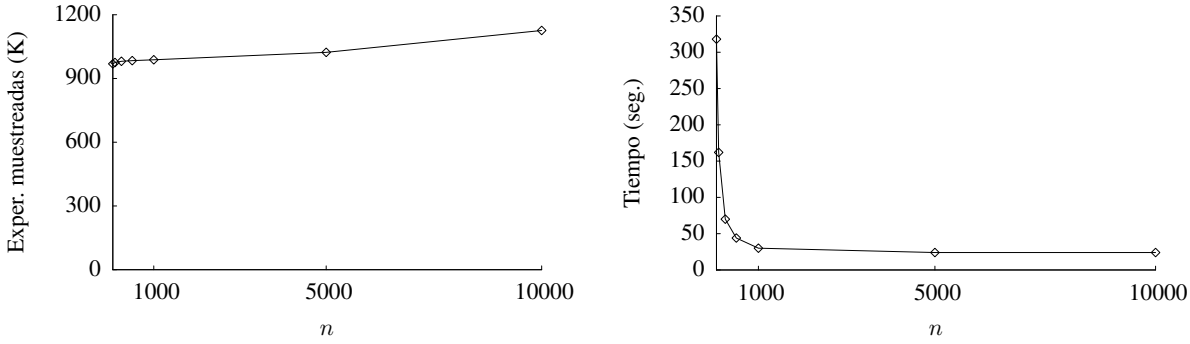



Figura 2.3: Comportamiento dependiendo del parámetro $n$.

que hemos usado para estudiar el comportamiento del algoritmo en función del número de experiencias tomadas en cada muestreo podemos destacar lo siguiente:

- La precisión alcanzada ha sido siempre superior al $99,00\,\%$, cumpliéndose los requisitos del usuario (que había establecido el error máximo tolerado en un $1,00\,\%$ puesto que $\varepsilon = 0,01$).
- El tamaño de los árboles inducidos ha estado siempre cerca de las $200$ hojas, algo mayor que el mínimo que podía alcanzarse (que eran las $130$ hojas del árbol con el que se generó el dataset), pero no han sido excesivamente grandes.
- El número total de experiencias muestreadas se mantiene casi inalterable, siendo el leve incremento producido cuando $n$ toma valores entre $5000$ y $10000$ consecuencia del exceso de experiencias tomadas en la última iteración antes de finalizar el algoritmo.
- El tiempo consumido por el algoritmo decrece considerablemente conforme $n$ toma valores mayores. Este comportamiento viene motivado por los repetidos e innecesarios recálculos producidos después de cada muestreo. La nueva información aportada después de cada muestreo no era suficiente para expandir el árbol de decisión, haciendo necesarios numerosos muestreos, que lo único que conseguían era realizar numerosos recálculos que dejaban el árbol de decisión en el mismo estado la mayor parte de las veces.

Observando los resultados mostrados, podemos concluir que asignar valores grandes (a partir de $1000$) al parámetro $n$ es una buena estrategia para reducir el tiempo necesario para inducir el árbol de decisión con la precisión requerida por el usuario. Por defecto, el valor usado para el parámetro $n$ ha sido fijado en $10000$.

### 2.3.2. Factor de expansión: $\gamma$

Como ya comentamos, la importancia del factor de expansión ($\gamma$) se pone de manifiesto en la facilidad ofrecida para considerar nuevas expansiones. Valores elevados de este parámetro interno relajarán la frontera de expansión ($(1-\gamma)\cdot\varepsilon$) y permitirán que el proceso de expansión sea considerado más veces (aunque el requisito de que el mejor atributo sea identificado claramente sigue siendo imprescindible). En la figura 2.4 presentamos el comportamiento del algoritmo dependiendo del valor dado a $\gamma$ considerando dos aspectos relevantes: el tamaño del árbol inducido y el número total de experiencias muestreadas antes de satisfacer los requisitos del usuario. En esta ocasión hemos considerado los resultados obtenidos teniendo en cuenta dos variantes: cuando se usa $n = 1000$ y cuando se usa $n = 10000$. Queremos así poner de manifiesto que, siempre que los valores usados para $n$ sean mayores que un mínimo, el resultado de la inducción variando $\gamma$ se mantiene independiente de $n$.

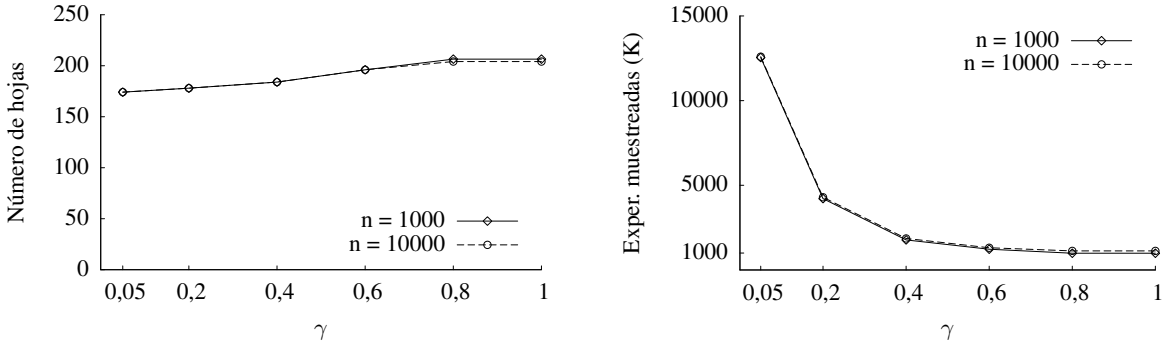



Figura 2.4: Comportamiento dependiendo del parámetro $\gamma$.

En este caso no mostramos las gráficas de la precisión final alcanzada ni el tiempo consumido por ser aspectos menos relevantes, aunque sí hablaremos de ellos en las consideraciones que a continuación enumeramos:

- La precisión alcanzada ha sido siempre superior a $99,00\,\%$ como era de esperar, puesto que el algoritmo se detiene al satisfacer los requisitos del usuario.
- El tamaño del árbol aumenta sensiblemente al aumentar el valor de $\gamma$, comportamiento lógico puesto que las condiciones para expandir el árbol se ven relajadas. En cualquier caso, el incremento no es excesivo.
- El número total de experiencias muestreadas crece conforme la frontera de expansión se acerca al nivel máximo de error tolerado (circunstancia que ocurre cuando se usan valores pequeños para $\gamma$). Recordemos que $\gamma$ está definido en el intervalo $(0, 1]$ para evitar situaciones en las que el intervalo de error sea asintótico con el error máximo tolerado por el usuario. Asignar valores a $\gamma$ próximos a $1$ hace que la frontera de expansión esté próxima a $0$ y, por lo tanto, se relaja la condición para considerar una expansión, haciendo que el árbol se induzca usando menos experiencias.
- El tiempo consumido por el algoritmo sigue un comportamiento similar al número de experiencias muestreadas dado que están íntimamente relacionadas: a mayor número total de experiencias muestreadas, mayor tiempo hará falta para procesarlas.

Observando los resultados mostrados, podemos concluir que asignar valores cercanos a $1$ al parámetro $\gamma$ es una buena estrategia para reducir el número total de experiencias necesarias antes de alcanzar el modelo requerido. Por defecto, el valor usado para el parámetro $\gamma$ ha sido fijado en $0,8$.

### 2.3.3. Diferenciación de atributos: $d$

Durante el estudio de este parámetro interno pretendemos justificar el valor por defecto usado para $d$ como hemos hecho con los demás parámetros, pero también queremos poner de manifiesto la relación que existe entre este parámetro y el número de experiencias muestreadas en cada paso ($n$). Las gráficas que presentamos en la figura 2.5 nos ayudarán a exponer el comportamiento del algoritmo en función del valor dado a $d$.

En este caso también hemos representado el tamaño del árbol inducido y el número de experiencias muestreadas antes de alcanzar el modelo buscado. No mostramos las gráficas de la precisión final alcanzada ni el tiempo consumido por ser aspectos menos relevantes, aunque sí hablaremos de ellos en las consideraciones que a continuación enumeramos:

- La precisión alcanzada ha sido siempre superior a $99,00\,\%$, como en los casos estudiados anteriormente, ya que el algoritmo se detiene al alcanzar la precisión mínima deseada.

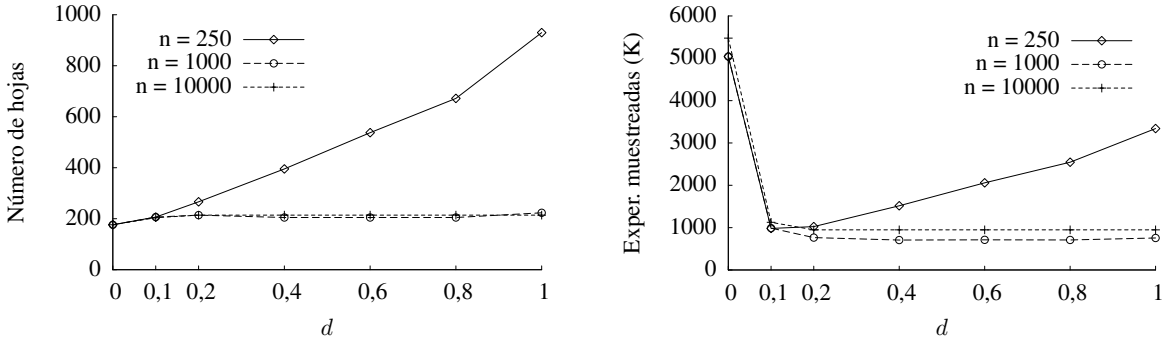



Figura 2.5: Comportamiento dependiendo del parámetro $d$.

- El tamaño del árbol inducido depende de $d$ y de $n$. Si nos fijamos en el caso en el que usamos el valor por defecto para $n$ ($n = 10000$), vemos cómo el tamaño del árbol se mantiene estable (con $214$ hojas), lo que nos confirma la buena elección del parámetro por defecto para $n$. Conforme $n$ se decrementa, el tamaño del árbol inducido se hace más sensible al valor empleado para $d$. La inducción de árboles de decisión con más hojas cuando $d$ es mayor se pone de manifiesto cuando $n$ toma valores pequeños y esto se debe a la combinación de dos circunstancias:

  1. Cuando damos valores grandes a $d$, estamos permitiendo un mayor solapamiento entre los atributos para considerar que el mejor atributo ha sido claramente identificado. Debido a esto, la calidad en las expansiones se reduce, lo que nos lleva a necesitar más expansiones para alcanzar un árbol con la mínima precisión requerida.
  2. Cuando el valor de $n$ es elevado, el número de experiencias muestreadas antes de cada nueva expansión también es elevado, lo que nos permite elegir el mejor atributo considerando más información. Esto hace que las expansiones sean de cierta calidad y no sea preciso inducir árboles tan grandes.

- Para el número total de experiencias muestreadas tenemos dos casos diferentes que considerar: cuando $d = 0$ o cuando $d > 0$. En el caso de asignar el valor $0$ a $d$, lo que estamos haciendo es exigir que el mejor atributo sea claramente diferente al resto, sin permitir en ningún caso solapamiento entre los atributos. Esa es la razón para que el número total de experiencias muestreadas sea tan elevado, se necesita mucha información para poder hacer afirmaciones tan exigentes. Relajando algo esa restricción y usando valores para $d$ cercanos a 0 conseguimos disminuir de forma considerable el número de experiencias necesario sin apenas repercutir en la calidad del árbol inducido. Cuando la restricción la relajamos demasiado (valores para $d$ cercanos a $1$) y lo combinamos con un valor de $n$ pequeño (que no nos asegure demasiada información nueva antes de una nueva expansión) nos encontramos con que es preciso muestrear un número elevado de experiencias antes de alcanzar el modelo requerido. La razón para que se dé este último caso es que las expansiones de baja calidad llevan a inducir mayores árboles, para lo que es preciso muestrear un mayor número de experiencias.

- El tiempo consumido por el algoritmo sigue un comportamiento similar al número de experiencias muestreadas dado que están íntimamente relacionadas: a mayor número total de experiencias muestreadas, mayor tiempo hará falta para procesarlas.

Observando los resultados mostrados, podemos concluir que asignar valores cercanos a $0$ al parámetro $d$ es una buena estrategia para reducir el número total de experiencias necesarias antes de alcanzar el modelo requerido, y conseguirlo además con un árbol de menor tamaño. Por defecto, el valor usado para el parámetro $d$ ha sido fijado en $0,1$.

## 2.4. Estudio de los argumentos de entrada

Después del estudio expuesto en la sección anterior acerca de los parámetros internos del algoritmo, cuyos valores están fijados por defecto y no son configurables por el usuario, vamos a estudiar cómo se comporta el algoritmo atendiendo a distintos valores para los argumentos de entrada al algoritmo (sin tener en cuenta el propio conjunto de datos).

Recordemos que los argumentos que establece el usuario están encaminados a exigir una precisión mínima del modelo que se va a inducir, asegurando que se alcanza con una determinada confianza. Dependiendo del problema en cuestión y de las necesidades del usuario (experto) estos criterios pueden variar. Una de las principales ventajas derivada de no requerir modelos excesivamente precisos radica en que los modelos inducidos son más sencillos (tienen menos hojas) y, por lo tanto, serán más fáciles de interpretar por los expertos. De igual forma, aunque puede no ser una ventaja tan relevante como la anterior, el tiempo consumido para alcanzar árboles de decisión con la precisión mínima deseada también se reduce.

### 2.4.1. Error máximo tolerado: $\varepsilon$

Dependiendo del error máximo tolerado por el usuario (o la precisión mínima necesaria), el modelo inducido a partir del conjunto de datos será más o menos fiel al conocimiento real que subyace en dicho conjunto de datos. Cuanto menos exigente sea el requisito del usuario, menos precisión alcanzará el modelo, más sencillo será y menos costoso resultará encontrar un modelo que satisfaga sus necesidades.

Para mostrar estas características hemos usado el mismo conjunto de datos empleado en la sección anterior (sección 2.3) y hemos usado el algoritmo configurando los parámetros internos con sus valores por defecto (expuestos en la misma sección anterior). En la figura 2.6 mostramos la precisión alcanzada después de usar diferentes valores para el argumento $\varepsilon$ (desde $0,0$ hasta $0,2$), siendo el valor de $\delta = 0,05$. Como era de esperar, la precisión alcanzada es siempre superior a la mínima precisión requerida. Adicionalmente, en la misma figura 2.6, hemos representado los diferentes niveles de error alcanzados para los diferentes valores de $\varepsilon$. Podemos observar en la gráfica derecha, como era de esperar, que la línea que representa el $\sup(error)$ está siempre por debajo de $\varepsilon$ (el error máximo tolerado), puesto que es el criterio principal para detener la ejecución del algoritmo: alcanzar la precisión mínima requerida por el usuario (*Condición_Parada*). Los valores de $\inf(error)$ también se encuentran, como es lógico, por debajo de los valores de $\sup(error)$. Lo realmente interesante es que el error real cometido (medido mediante la evaluación de los conjuntos de prueba en la validación cruzada) se encuentra en el intervalo de error calculado en el árbol de decisión inducido con el algoritmo IADEM-0. Es más, se encuentra más próximo a la cota inferior que a la superior.

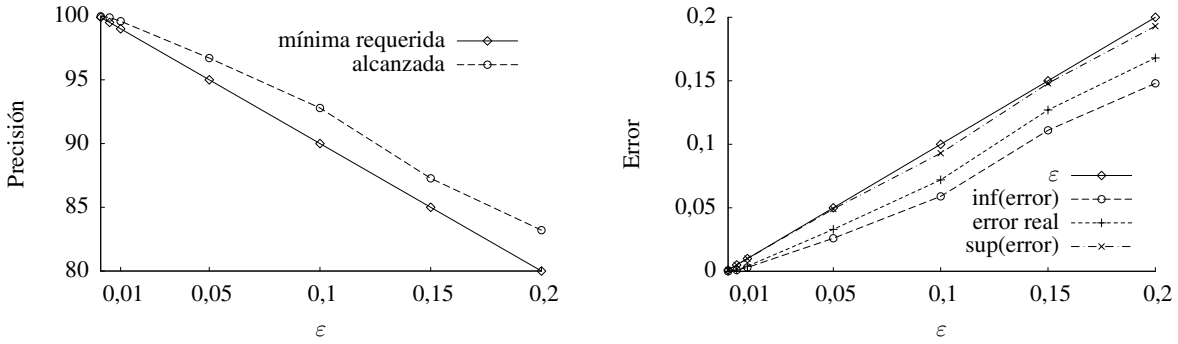



Figura 2.6: Comportamiento dependiendo de la entrada $\varepsilon$.

Al solicitar el usuario un menor límite máximo para el error tolerado, los modelos no tienen que ser tan precisos y, en consecuencia, al algoritmo le será menos costoso inducir modelos que alcancen los niveles deseados. Este aspecto lo podemos observar gráficamente en la figura 2.7. Al no requerir que sea tan preciso,

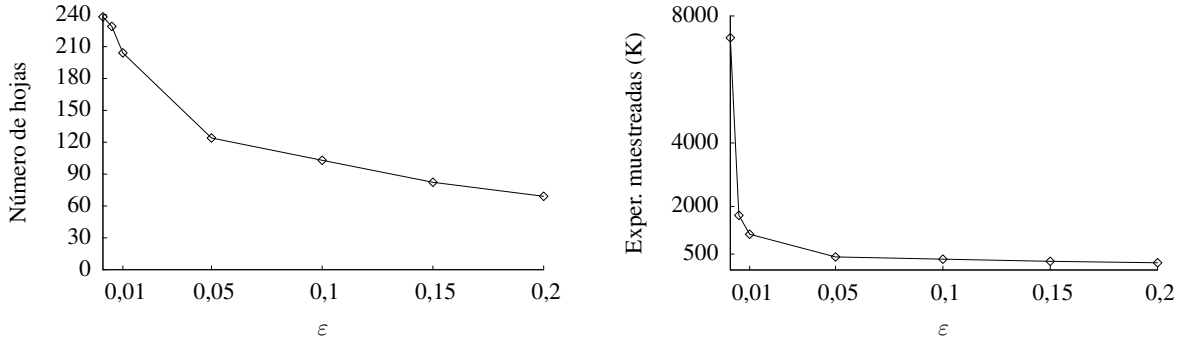



Figura 2.7: Comportamiento dependiendo del parámetro $\varepsilon$.

el modelo que se obtiene es más sencillo (el árbol es más pequeño teniendo menos ramas). También debemos considerar el tiempo requerido para alcanzar la solución que satisfaga los requisitos preestablecidos: cuanto más elevada sea la precisión exigida, mayor tiempo requerirá el algoritmo para alcanzarla. Debemos recordar que, al ser un algoritmo incremental, la solución se va construyendo conforme se dispone de nuevas experiencias y que, debido a la definición del algoritmo que presentamos, el principal criterio para detener el algoritmo es satisfacer los requisitos del usuario en cuanto a la precisión del modelo final inducido.

### 2.4.2. Confianza requerida: $(1 - \delta)$

Además del error máximo permitido, el usuario también debe establecer cuál es la confianza que desea tener acerca del resultado obtenido. La repercusión de este argumento de entrada no es tanta como la que tiene el argumento $\varepsilon$ y se limita principalmente al número total de experiencias muestreadas antes de alcanzar la solución deseada y, consecuentemente, al tiempo consumido.

En la figura 2.8 mostramos el tamaño final del árbol inducido y el número total de experiencias muestreadas dependiendo del valor que se le dé a $\delta$, siendo $\varepsilon = 0,01$. El rango de valores que hemos considerado para estudiar el comportamiento (desde $0,001$ hasta $0,2$) ha sido elegido de tal forma que se consideren los valores más comúnmente empleados. La precisión final alcanzada (no mostrada en la gráfica) y el tamaño del árbol inducido permanecen constantes para los diferentes valores asignados a $\delta$. Por contra, el número total de experiencias muestreadas aumenta conforme se exige mayor confianza en el resultado. Recordemos que la confianza es un factor determinante en la estimación de las cotas de concentración que sirven para determinar diversas variables propias del árbol de decisión y, por lo tanto, una mayor confianza implica que los intervalos para dichas variables aumenten y se requiera muestrear más experiencias para alcanzar los mismos intervalos que se alcanzaban cuando la confianza requerida era menor. La influencia del argu-

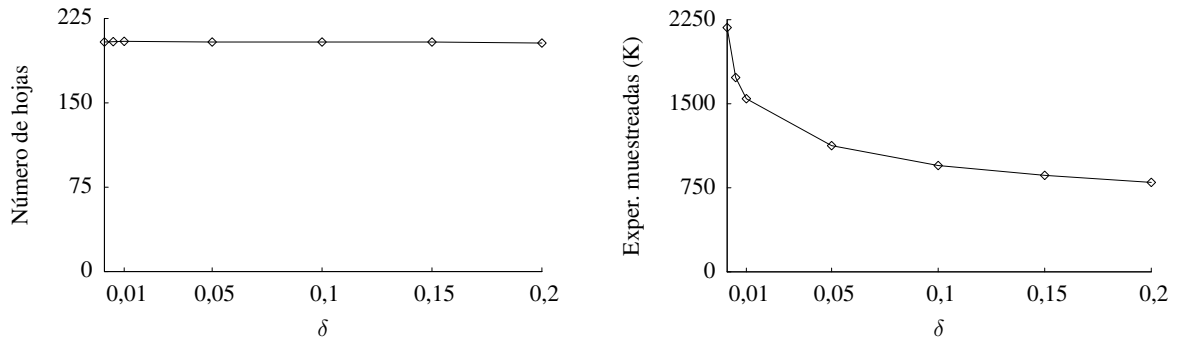



Figura 2.8: Comportamiento dependiendo del parámetro $\delta$.

mento $\delta$ en el tiempo consumido es similar a la influencia ejercida sobre el número total de experiencias muestreadas.

## 2.5. Estudio de la complejidad

Una vez hemos presentado el algoritmo IADEM-0 y hemos estudiado el comportamiento que se observa dependiendo de los valores asignados a los parámetros internos y a los argumentos de entradas, pasamos a realizar un estudio de la complejidad temporal y espacial.

### 2.5.1. Complejidad temporal

Para realizar el estudio de la complejidad temporal, primero presentaremos el coste computacional que supone incorporar la información aportada por una sola experiencia al modelo. Posteriormente justificaremos la ventaja de incorporar las experiencias usando grupos de experiencias (de tamaño $n$) y analizaremos qué complejidad es la que muestra el algoritmo en su conjunto.

A continuación analizamos el coste computacional que supone incorporar la información de una experiencia en el modelo y, para hacerlo de forma más sencilla, estudiaremos qué complejidad aporta cada uno de los predicados y procedimientos que definen el algoritmo. Para realizar el estudio utilizaremos algunas variables que a continuación describimos:

- $a$: número de atributos descritos en el problema.
- $b$: número máximo de valores diferentes en los atributos. Supone considerar, como peor caso, que todos los atributos tienen el mismo número de valores y este número es $b$.
- $k$: número de valores de la clase.
- $d$: profundidad máxima del árbol inducido hasta el momento.
- $h$: número máximo de hojas del árbol inducido con profundidad $d$. Se trata de una variable calculada que se definiría como $b^d$.

El primer procedimiento que nos encontramos, que sólo se ejecutará una vez, es el procedimiento INICIALIZAR. Su coste computacional es $\mathcal{O}(a \cdot b \cdot k)$ debido a que se deben inicializar todas las variables y el mayor número de variables se encuentran en las hojas virtuales. Habrá tantas hojas virtuales como atributos ($a$) y valores por atributo ($b$), y en cada hoja virtual habrá tantos contadores como clases ($k$). Este coste no es relevante para el algoritmo, porque lo realmente relevante es el coste del bucle (que se repetirá un determinado número de veces), pero lo presentamos para mostrar, por primera vez, que lo importante es la descripción del problema y no el tamaño del conjunto de datos (valor que no interviene en la fórmula).

El procedimiento MUESTREAR_Y_RECALCULAR es el más costoso, porque toma la información en la experiencia y necesita actualizar los componentes calculados. Como primera tarea debe localizar la hoja cuyos contadores tiene que actualizar. Eso supone, en el peor de los casos, hacer tantas consultas como la profundidad máxima del árbol, es decir, $\mathcal{O}(d)$. Actualizar los contadores sigue un esquema similar al descrito en el procedimiento INICIALIZAR, puesto que la mayor parte de los contadores se encuentran en las hojas virtuales. Concretamente, el coste de esta operación será $\mathcal{O}((a-d) \cdot b \cdot k)$ puesto que en la rama que llega a la hoja donde hay que actualizar la información ya han sido usados $d$ atributos (para los que no existen hojas virtuales). Con esto se puede observar que, cuanto más profunda sea la rama, mayor tiempo costará identificar la hoja, pero mucho menor será el tiempo de actualización de contadores. Es decir, las hojas más costosas de actualizar son aquellas más cercanas a la raíz, pero cuanto más crece el árbol, más se alejan y menos cuesta actualizarlas. La parte del procedimiento MUESTREAR_Y_RECALCULAR en la que se recalculan los componentes calculados es la más costosa del algoritmo. Podemos dividirla en tres partes: actualizar los componentes que sirven para calcular la estimación del error, localizar la hoja que más error introduce en el árbol y actualizar sus componentes calculados para el proceso de expansión.

El cálculo que conduce a la estimación del error en el árbol (cálculo de $\text{ínf}(error)$ y $\sup(error)$) es de orden $\mathcal{O}(\text{máx}\{k,h\} \cdot \log_2 \text{máx}\{k,h\})$ debido a que el proceso más costoso supondría una ordenación de $k$ elementos (número de clases) o de $h$ elementos (número de hojas en el árbol). Identificar cuál es la hoja que más error introduce en el árbol ($peor\_hoja$) es más sencillo y sólo supone buscar entre las hojas del árbol, por lo que su complejidad es $\mathcal{O}(h)$. Los componentes calculados que deben actualizarse para evaluar la expansión de una hoja (determinación del mejor atributo para expandir –$mejor\_atributo_i$–) no son los de todas las hojas, sino únicamente los de la hoja que acaba de ser seleccionada ($i = peor\_hoja$). El coste de esta operación es $\mathcal{O}((a-d) \cdot b \cdot k \log_2 k)$ puesto que están involucradas todas las hojas virtuales de una hoja real ($(a-d) \cdot b$) y hay que realizar una operación de ordenación de las contadores de clase en cada hoja virtual ($\mathcal{O}(k \log_2 k)$).

El último procedimiento del que resta mostrar su complejidad es EXPANDIR_ÁRBOL. Puesto que los valores que se usarán al cambiar las hojas virtuales y redefinirlas como reales ya fueron calculadas en el proceso MUESTREAR_Y_RECALCULAR, el único coste derivado de la ejecución de este procedimiento será el de crear los nuevos contadores de las nuevas hojas virtuales y el de actualizar los componentes calculados que estiman el error en el árbol de decisión. Para crear los nuevos contadores se precisa $\mathcal{O}((a-(d+1)) \cdot b^2 \cdot k)$ porque habrá que actualizar los contadores de $b$ nuevas hojas virtuales, en cada una de las cuales existirán $(a-(d+1))$ atributos libres, con $b$ valores cada uno, y existirán $k$ clases para cada atributo libre en cada nueva hoja virtual. Para dejar actualizados los componentes calculados que serán usados al inicio del siguiente bucle es preciso realizar un proceso de recálculo similar al realizado en el procedimiento MUESTREAR_Y_RECALCULAR siendo éste de complejidad $\mathcal{O}(\text{máx}\{k,h\} \cdot \log_2 \text{máx}\{k,h\})$.

El grueso de la computación se realiza en los procedimientos, porque, como mostramos a continuación, la evaluación de los predicados es prácticamente directa en todos los casos. Los predicados *Condición_Parada*, *Condición_Completamente_Expandido* y *Condición_Expansión* consumen tiempo constante porque sólo necesitan hacer una simple comparación. El predicado *Condición_Es_Expansible(peor_hoja)* es el único que consume algo más de tiempo ($\mathcal{O}(a-d)$) puesto que debe realizar tantas comparaciones como atributos libres queden en la hoja que se pretende expandir.

En definitiva, el mayor coste computacional se realiza, como es lógico, al recalcular todas los componentes del árbol de decisión, puesto que es necesario recorrer todas las hojas realizando distintas operaciones que suponen, en el peor caso, alguna reordenación. Por lo tanto, recoger la información de las experiencias una a una es muy costoso y es preferible agruparlas, tomándolas de bloque en bloque, para reducir el número de veces que se ejecuta el bucle. Ya presentamos, en la figura 2.3, una gráfica en la que se mostraba el tiempo consumido en función del valor asignado al parámetro interno $n$, y se podía observar cómo el tiempo consumido decrecía exponencialmente conforme mayor era el valor de $n$, hasta que prácticamente se estabilizaba a partir de un valor elevado (recordemos que elegimos por defecto el valor $n = 10000$).

Dependiendo de la dificultad para extraer conocimiento de un conjunto de datos se necesitarán muestrear más o menos experiencias, pero ese número no influye en la complejidad de una iteración del bucle. Como hemos mostrado, la complejidad de cada iteración del bucle únicamente depende del tipo de problema que se está abordando (número de atributos, de clases, etc.) y de la estructura del árbol inducido en cada momento (profundidad máxima, número de hojas, etc.).

Para tener una idea de la complejidad global del algoritmo podemos realizar la siguiente aproximación. El coste computacional vendrá determinado, principalmente, por el número de veces que se ejecute el bucle del algoritmo (cuya complejidad acabamos de presentar), y este número está en clara relación con el número total de experiencias que necesitan ser muestreadas antes de finalizar la inducción. Considerando que los argumentos de entrada al algoritmo son el error máximo tolerado por el usuario ($\varepsilon$) y la confianza requerida ($1-\delta$), es posible determinar aproximadamente qué número de experiencias es necesario para satisfacer los requisitos del usuario. Básicamente consiste en despejar de las fórmulas de las cotas de concentración resumidas en la función que calcula los márgenes de error ($margen\_\varepsilon$), y se puede observar que la relación es polinómica en $\frac{1}{\varepsilon}$ y $\frac{1}{\delta}$. Es decir, cuanto menor sea el error máximo tolerado (cuanto menor sea $\varepsilon$) o cuanto mayor sea la confianza requerida (cuanto menor sea $\delta$), mayor número de experiencias será necesario muestrear. Experimentalmente hemos comprobado que un mayor número de experiencias muestreadas supone un

incremento lineal en el tiempo consumido por el algoritmo para concluir la inducción.

La posibilidad de disponer de un estudio algo más completo sobre la complejidad temporal sería de cierta utilidad y, por tanto, en futuros trabajos de investigación estudiaremos los distintos factores que afectan a la complejidad y pretendemos establecer alguna comparación entre este algoritmo y otros que tengan un enfoque no incremental.

### 2.5.2. Complejidad espacial

La complejidad espacial que presenta el algoritmo IADEM-0 únicamente depende del problema que se esté tratando y de la estructura del árbol. De la descripción del problema, los factores relevantes son el número de atributos diferentes ($a$), el número de valores para cada atributo (supondremos que todos los atributos tienen el mismo número de valores $b$) y el número de clases ($k$). De la estructura del árbol es relevante cómo están distribuidas las hojas ($h$) y su profundidad máxima. Por simplicidad, supondremos que trabajamos con un árbol completo de profundidad $d$. Los requisitos de memoria que necesita este algoritmo para trabajar son $\mathcal{O}(h \cdot (a-d) \cdot b \cdot k)$ puesto que el grueso de los contadores, que son los que determinarán la complejidad, se encuentran en las hojas virtuales. En cada hoja virtual se almacena un vector de tamaño $k$ y el número de hojas virtuales depende de la profundidad del árbol, habiendo $(h \cdot (a-d) \cdot b)$ hojas virtuales en un árbol completo de profundidad $d$. De esta forma, el máximo espacio requerido para almacenar toda la información será $\mathcal{O}(h \cdot b \cdot k) = \mathcal{O}(b^a \cdot k)$, que es el caso de un árbol casi totalmente expandido ($d = a - 1$). Si estuviese totalmente expandido ($d = a$) no existirían hojas virtuales en la que almacenar información y ocuparía menos espacio que el anterior.

Como puede apreciarse, los requisitos de memoria tampoco se ven determinados por el tamaño del conjunto de datos que se esté procesando, sino por la descripción del problema y la estructura actual del árbol. En el peor caso, no influirá ni siquiera la estructura del árbol, puesto que únicamente dependerá del número de atributos ($a$), del número de valores por atributo ($b$) y del número de clases ($k$). Estas necesidades de memoria podrían suponer una limitación para el algoritmo si el problema que se está tratando tiene una alta dimensionalidad y el algoritmo no es capaz de encontrar una solución con un árbol relativamente pequeño (que minimice el número de hojas virtuales).

# Capítulo 3

# Mejorando la Inducción de Árboles de Decisión por Muestreo: IADEM-2

Hasta este momento hemos presentado el algoritmo IADEM-0 con bastante detalle pero, a pesar de que es un algoritmo con una serie de características bastante interesantes, cuenta con una serie de limitaciones que no permiten aplicarlo directamente bajo determinadas condiciones, como pueden ser conjuntos de datos con atributos continuos o en los que haya ruido. Una evolución del algoritmo IADEM-0 [Ram-2000; Ram-2001; Ram-2004] dio lugar al algoritmo IADEM [Cam-2002; Ram-2006a], que contempla la posibilidad de tratar conjuntos de datos en los que existe ruido y que presentaremos en la primera sección de este capítulo. Dos nuevas mejoras se incorporaron al algoritmo IADEM para dotarlo de la capacidad de tratar directamente conjuntos de datos con atributos continuos y para mejorar la predicción con información ya disponible en el árbol. Estas dos mejoras, junto a la anterior, dieron lugar a la variante del algoritmo llamada IADEMc [Cam-2006].

Agrupando todas estas mejoras y otras que se han incorporado con posterioridad, y que también describiremos en este capítulo, surge un nuevo algoritmo, el algoritmo IADEM-2 [Cam-2007]. En este capítulo describimos con el mismo nivel de detalle y siguiendo la misma estructura usada el capítulo anterior, las principales características del algoritmo que estamos presentando en esta parte de la tesis. Desarrollaremos un estudio del modo en que influyen los variaciones de los valores de los parámetros internos y fijaremos valores por defecto para todos ellos.

## 3.1. Tolerancia al ruido

Como ya se comentó en el capítulo anterior, el algoritmo IADEM-0 no es tolerante al ruido y está diseñado para ser utilizado con conjuntos de datos en los que no haya presencia de ruido. De ahí le viene el número "0" que forma parte del nombre. Cuando hablamos de ruido nos referimos a que una misma experiencia pueda aparecer clasificada de dos o más formas diferentes en el mismo conjunto de datos. Las razones para que esto ocurra pueden ser muy diversas (problemas en la captura de datos, no determinismo inherente al problema, ausencia de otros atributos que determinen completamente la clasificación, etc.) pero lo único que nos interesa es poder extraer conocimiento sobre la forma en que las experiencias son etiquetadas. Con aquellas experiencias que estén presentes en el conjunto de experiencias pero que tengan diferentes etiquetas para la clase, el objetivo será intentar clasificar las experiencias teniendo en cuenta las más frecuentes (la predicción se dará para la clase mayoritaria).

Como consecuencia de tal carencia, si el algoritmo IADEM-0 se usa para extraer conocimiento de un conjunto de datos en el que hubiese ruido, nunca se podría alcanzar un árbol cuyo error estimado esté por debajo del ruido en aquél. Por tanto, sería necesario un conocimiento previo del ruido por parte del usuario para solicitar un valor adecuado del error máximo tolerado (mayor que el ruido) o se produciría un estancamiento del aprendizaje en el nivel de error que nunca quedaría por debajo del nivel de ruido presente en el conjunto de datos.

Debido a esta limitación se hacía necesario encontrar alguna solución que permitiese extraer conocimiento de conjuntos de datos con ruido. La aportación definitiva que se incluye en el nuevo algoritmo que proponemos en este capítulo, y que ha recibido el nombre de IADEM-2, se presenta en la sección 3.2, pero antes de describirla presentamos en esta sección una solución previa que se adoptó enfocada únicamente a la detección del ruido. Las razones para presentar dicha solución, aunque no forme parte del algoritmo que proponemos, son dos: supuso una mejora para el algoritmo IADEM-0 y será empleada como punto de referencia para realizar el estudio experimental.

El hecho de haber supuesto una mejora para el algoritmo IADEM-0 se hace patente en diversos trabajos [Cam-2002; Ram-2006a]. Por haberle incorporado la tolerancia al ruido, la versión del algoritmo surgida tras la integración de tal mecanismo ha sido llamada IADEM [Cam-2002] (sin el "$-0$"). El nombre del algoritmo también concuerda con el acrónimo en versión inglesa de "*Incremental Algorithm Driven by Error Margins*" [Ram-2006a]. No haremos un estudio exhaustivo de este enfoque puesto que ha sido reemplazado por otro en la definición del algoritmo que presentaremos (IADEM-2), pero puede ser útil para otros algoritmos que en el futuro puedan surgir con un enfoque similar a IADEM-0. La propuesta de la versión IADEM nos resultará también de bastante utilidad para realizar el estudio comparativo que se llevará a cabo en el capítulo de experimentación, puesto que las limitaciones de IADEM-0 restringen demasiado el tipo de conjunto de datos que se podrían usar (aquellos que no presentasen ruido).

**Presentando una primera solución**

El comportamiento típico de la estimación del error cometido por el árbol para el peor caso ($\sup(error)$) es tender a $0$, es decir, conforme se analizan más experiencias y se expande el árbol, el modelo se ajusta más y el error estimado disminuye. Tras analizar el comportamiento del algoritmo IADEM-0 ante la presencia de ruido cuando se le exigía que alcanzase niveles de precisión que no eran factibles debido al propio ruido, se observó un comportamiento anómalo. Cuando el error estimado para el árbol se acercaba al nivel de ruido en el conjunto de experiencias, la cota superior del error estimado ($\sup(error)$) se comportaba de forma diferente. Aunque seguía disminuyendo conforme se analizaban más experiencias (comportamiento lógico debido a que las cotas de concentración se ajustan más), esta cota superior aumentaba cuando se producía una expansión. Los dos comportamientos pueden observarse en la figura 3.1: mientras que en la parte izquierda de la gráfica siempre se producen decrementos de $\sup(error)$ (haya o no expansión), en la parte derecha se producen incrementos de $\sup(error)$ en el momento de la expansión (comportamiento anómalo). El método que a continuación proponemos se basa en observar los incrementos de $\sup(error)$ cuando se producen expansiones y establecer unos criterios para evitar posibles ocurrencias esporádicas que no estén relacionadas con la detección de ruido.

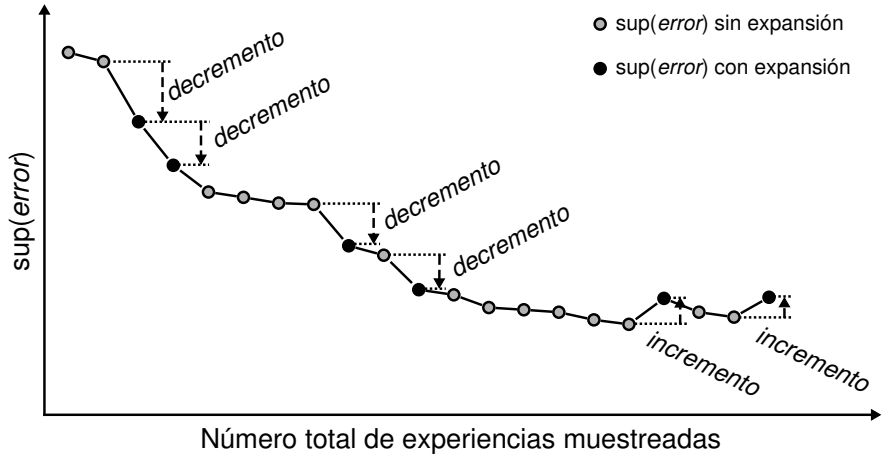



Figura 3.1: Evolución de la cota superior del error en un árbol de decisión generado por IADEM-0.

**Formalizando la solución aportada**

En la figura 3.2 hemos modificado el pseudocódigo del algoritmo IADEM-0 para incorporar un nuevo predicado en el bucle principal: "*Condición_Ruido*". Esta incorporación junto con la definición de los elementos necesarios para evaluar el nuevo predicado conforman la única modificación realizada al algoritmo para poder trabajar con conjuntos de datos que presenten ruido.

**Entrada:** $\varepsilon$ (error máximo tolerado), $\delta$ (confianza), $E$ (conjunto de datos)

1. INICIALIZAR
2. **mientras** ¬*Condición_Parada* ∧
   ¬*Condición_Completamente_Expandido* ∧
   ¬*Condición_Ruido* **hacer:**
2.1. MUESTREAR_Y_RECALCULAR
2.2. **si** ¬*Condición_Parada* ∧
   *Condición_Expansión* ∧
   *Condición_Es_Expansible*(*peor_hoja*) **entonces:**
2.2.1. EXPANDIR_ÁRBOL

**Salida:** $HOJAS$ (árbol de decisión)

Figura 3.2: Pseudocódigo del algoritmo IADEM.

La incorporación del predicado *Condición_Ruido* tan solo supone incluir un componente básico con información general (no pertenece a ninguna hoja) que almacene la historia más reciente acerca del comportamiento de las últimas expansiones. La información que almacenará serán las diferencias observadas en sucesivos valores de la cota superior del error estimado para el árbol ($\sup(error)$).

Hemos definido un vector, llamado $historia$, que almacena las diferencias entre el valor estimado para el error del árbol en el peor caso después de una expansión y antes de ella. Así, en la $j$-ésima posición del vector se guarda la diferencia correspondiente a la $j$-ésima última expansión. El tamaño del vector ($tamaño\_historia$) es mayor conforme más fácil es expandir, criterio que viene determinado, principalmente, por el parámetro interno $d$. Si se favorece la expansión (valores para $d$ cercanos a 1), el tamaño del vector debe ser mayor para poder estudiar más casos. La forma de almacenar las diferencias en el vector es la propia de una cola: conforme llegan nuevos valores (procedentes de nuevas expansiones) los más viejos se eliminan del vector.

La definición más formal de los elementos que son necesarios para evaluar el predicado que controla la detección del ruido es la siguiente:

- $tamaño\_historia = \max\{3, \lceil d \cdot 10 \rceil\}$
- $historia \in [-1, 1)^{tamaño\_historia}$
  con $historia_j = \sup(error)_{post\_j} - \sup(error)_{pre\_j}$ donde $\sup(error)_{post\_j}$ es el valor de $sup(error)$ después de haber realizado la $j$-ésima última expansión y $\sup(error)_{pre\_j}$ es el valor de $sup(error)$ justo antes de realizar dicha expansión. De esta forma, el vector $historia$ recoge de forma ordenada las diferencias producidas en las $j$ últimas expansiones, siendo $historia_1$ el valor de la diferencia producida en la última expansión e $historia_{tamaño\_historia}$ el valor de la última diferencia de la que se tiene constancia hasta el momento.

Como es de esperar, la inicialización de este vector se realiza en el procedimiento INICIALIZAR y su correspondiente actualización se realiza en el procedimiento EXPANDIR_ÁRBOL. Para no volver a reescribir los procedimientos completos, únicamente detallaremos cómo han de ser modificados para incluir el tratamiento del vector $historia$. Así, ambos procedimientos incorporarán las siguientes instrucciones:

- INICIALIZAR
  - inicializamos el vector poniendo todos sus valores a "$-1$" (que es el mejor caso, en el que todos los valores se corresponden con decrementos):
    $historia_j = -1 \quad \forall j \in \{1, \ldots, tamaño\_historia\}$
- EXPANDIR_ÁRBOL
  - antes de expandir hay que guardar el valor de $\sup(error)$:
    $previo\_sup\_error = \sup(error)$
  - después de haber expandido la hoja hay que incluir la diferencia en el vector (conservando los valores más recientes):
    $historia_j = historia_{j-1} \quad \forall j \in \{2, \ldots, tamaño\_historia\}$
    $historia_1 = \sup(error) - previo\_sup\_error$

Una vez hemos visto qué elementos son necesarios para evaluar el predicado encargado de detectar el ruido y cómo son modificados, podemos pasar a describir el propio predicado. Como se ha explicado previamente, el objetivo de este predicado consiste en contar el número de decrementos de la cota superior del error para el árbol (casos favorables) que se han producido en las últimas expansiones (y que están almacenados en el vector $historia$) y comprobar que se supere un umbral mínimo. Así definimos *Condición_Ruido* como sigue:

$$Condición\_Ruido : [-1, 1)^{tamaño\_historia} \rightarrow \{V, F\}$$
$$Condición\_Ruido(historia) \equiv$$
$$\left| \left\{ historia_j < 0 \mid j \in \{1, \ldots, tamaño\_historia\} \right\} \right| < \left\lceil \frac{tamaño\_historia - 2}{2} \right\rceil$$

Para poner un ejemplo del uso de este nuevo predicado, en el caso concreto de la configuración usada por defecto para IADEM-0 (que sigue siendo la misma para la versión IADEM) el valor para $d$ es $0,1$ y, por lo tanto, el número de diferencias de la cota superior del error estimado que se almacenan es $3$ (según la definición de $tamaño\_historia$). Para que se detecte ruido en este mismo ejemplo, el número de decrementos debe ser menor que $1$ (según la definición de $Condición\_Ruido$), es decir, deben ocurrir $3$ incrementos de la cota superior del error estimado de forma consecutiva para que se satisfaga el predicado que detecta la presencia de ruido.

Si nos fijamos en la complejidad introducida por este predicado, su repercusión es nula, tanto a efectos de tiempo como a efectos de memoria. El número de nuevas operaciones (actualización y evaluación del predicado) es constante en cada iteración del bucle y la actualización de las variables no siempre tiene que producirse (si no hay expansión no hay nada que actualizar). Por su parte, los nuevos requisitos de memoria se ven satisfechos con un vector de elementos de tamaño fijo (entre 3 y 10) y por una variable que almacena temporalmente el valor previo de $\sup(error)$, siendo su uso global en el algoritmo IADEM.

Debemos recordar que esta primera solución que acabamos de describir será mejorada por la que se propone en la subsección 3.2.2, y que ya formará parte del algoritmo IADEM-2 cuyas aportaciones detallamos en el resto de secciones que componen este capítulo (3.2, 3.3 y 3.4).

## 3.2. Mejorar la convergencia: IADEM-2

En esta sección agrupamos dos modificaciones que están encaminadas a mejorar la convergencia del algoritmo. Por un lado incrementaremos el número de expansiones que pueden realizarse inmediatamente después de cada muestreo (que anteriormente estaba limitado a una) y, por otro lado, sustituiremos el predicado encargado de detectar ruido que acabamos de presentar por una combinación de predicados que detendrán el algoritmo cuando se perciba que el aprendizaje se ha estabilizado o ha empezado a empeorar.

En la figura 3.3 presentamos el pseudocódigo del nuevo algoritmo IADEM-2 en el que se pueden apreciar las diferencias introducidas: inclusión de una lista con las peores hojas ($peores\_hojas$), las cuales intentarán ser expandidas antes del siguiente muestreo; definición de un predicado (*Condición_Finalización*) que agrupa todas las condiciones necesarias para controlar el bucle principal, incluyendo los nuevos predicados encargados de detectar la estabilización o el empeoramiento del modelo; y definición de un procedimiento para controlar las hojas que se intentarán expandir (RETIRAR_PEOR_HOJA).

**Entrada:** $\varepsilon$ (error máximo tolerado), $\delta$ (confianza), $E$ (conjunto de datos)

1. INICIALIZAR
2. **mientras** ¬*Condiciones_Finalización* **hacer:**
2.1. MUESTREAR_Y_RECALCULAR
2.4. **mientras** $peores\_hojas \neq (0, 0, \ldots, 0) \wedge$ ¬*Condiciones_Finalización* $\wedge$ *Condición_Expansión* **hacer:**
2.4.2. **si** *Condición_Es_Expansible*($peor\_hoja$) **entonces:**
2.4.2.1. EXPANDIR_ÁRBOL
2.4.3. RETIRAR_PEOR_HOJA

**Salida:** $HOJAS$ (árbol de decisión)

Figura 3.3: Pseudocódigo del algoritmo IADEM-2.

A continuación pasamos a estudiar detenidamente cada una de las aportaciones dirigidas a mejorar la convergencia. En la subsección 3.2.1 presentamos la forma en que se permite incrementar el número de expansiones que se pueden realizar inmediatamente después de cada muestreo y en la subsección 3.2.2 veremos los dos nuevos predicados encargados de detectar la estabilización o el empeoramiento del aprendizaje.

### 3.2.1. Incrementando el número de expansiones por muestreo

En el algoritmo IADEM-0 el número de expansiones después de haber procesado una muestra de experiencias (de tamaño $n$) estaba limitado a 1. Después del procedimiento MUESTREAR_Y_RECALCULAR sólo se intentaba la expansión de la hoja que mayor error introducía en el árbol ($peor\_hoja$). De esta forma la información disponible antes de cada expansión era bastante completa y las expansiones tenían bastante soporte. Aun así, existían principalmente dos problemas: el número total de experiencias muestreadas antes de alcanzar el modelo definitivo podía verse aumentada de forma innecesaria, y existía la posibilidad de que se produjese un estancamiento en el modelo sin avanzar hacia la solución.

El primer problema viene motivado porque existe la posibilidad de que haya más de una hoja en el árbol con suficiente información para ser expandida, pero que, por la definición del algoritmo, no pueda ser expandida en ese momento. Después de ejecutar el procedimiento MUESTREAR_Y_RECALCULAR, se dispone de la información necesaria para saber qué expansiones serían factibles, es decir, se pueden identificar las hojas en las que el mejor atributo se ha diferenciado lo suficiente para considerar a la hoja como expansible, pero únicamente se permite la expansión de una hoja. Sólo la $peor\_hoja$ en cada iteración del bucle es candidata a ser expandida. De esta forma puede haber hojas que en el futuro serán expandidas, porque ya tengan identificado y diferenciado al mejor atributo para su expansión, pero que tendrán que postergar su expansión hasta que se realicen los muestreos previos necesarios.

El segundo problema se deriva del hecho de que sólo se pueda expandir una hoja en cada iteración (la $peor\_hoja$) y de que es posible que el mejor atributo para realizarla no llegue nunca a distinguirse claramente como el mejor o su identificación se postergue demasiado. La identificación del mejor atributo para realizar la expansión de una hoja, tal y como se definió en el algoritmo IADEM-0, tiene ciertos defectos (que

comentaremos más detenidamente en la sección 3.3). De entre esos defectos destaca el hecho de que no se pueda identificar el mejor atributo para realizar la expansión (o se postergue considerablemente) si existiese otro atributo tan bueno como él puesto que no existe ningún mecanismo para romper el "*empate*". Como se puede deducir, el algoritmo no convergería a ninguna solución (o se vería muy retrasada) si la $peor\_hoja$ siempre fuese la misma (circunstancia perfectamente plausible) y el mejor atributo para expandirla no se pudiese diferenciar del resto (debido al problema del empate).

**Presentando la solución propuesta**

Tras haber visto los inconvenientes del procedimiento de expansión usado en el algoritmo IADEM-0, hemos elegido una dinámica diferente para la definición del algoritmo IADEM-2. Hemos permitido que, tras cada muestreo, se pueda expandir más de una hoja. En primer lugar se selecciona una lista con las hojas que más error introducen en el árbol ($peores\_hojas$) en una cantidad determinada por un parámetro interno ($num\_peores\_hojas$), cuya influencia se estudiará en el siguiente apartado. La lista de hojas se recalcula siempre en el procedimiento MUESTREAR_Y_RECALCULAR y está ordenada de forma descendente según el error introducido por cada una de las hojas. Una vez se ha seleccionado la lista, se intentan expandir las diferentes hojas de una en una (siguiendo el orden establecido) y siempre que las condiciones de finalización sigan sin cumplirse (¬*Condiciones_Finalización*).

**Formalizando la solución propuesta**

Para formalizar la solución propuesta tan solo es necesario definir cómo se calcula el nuevo componente $peores\_hojas$ (vector ordenado con los índices de las hojas que mayor error introducen en el árbol), redefinir el componente $peor\_hoja$ a partir del anterior (sigue siendo la hoja que más error introduce en el árbol) y explicar cómo son modificados estos elementos mediante el procedimiento RETIRAR_PEOR_HOJA, que elimina las hojas del conjunto $peores\_hojas$ una vez han sido usadas. Los nuevos componentes que definiremos (o redefiniremos) son calculados en cada iteración del bucle principal después de cada muestreo por el propio procedimiento MUESTREAR_Y_RECALCULAR y son modificados después de cada intento de expansión mediante el procedimiento RETIRAR_PEOR_HOJA.

Antes de definir los componentes calculados y el nuevo procedimiento, recordemos cómo se definían (al final de la subsección 2.2.3) algunos elementos que volverán a ser necesarios: el mayor error cometido por una hoja y el conjunto de hojas que pueden ser expandidas.

El mayor error cometido por la hoja $i$ del árbol, notado como $sup(\varepsilon_i)$ se definía del siguiente modo:

$$\sup(\varepsilon_i) = \max\{y_i \mid \vec{y} \in Y\} \cdot \ (1 - inf(q_i))$$

donde

$$Y = \Big\{(y_1, \ldots, y_{|HOJAS|}) \in [0,1]^{|HOJAS|} \quad | \quad \sum_{t=1}^{|HOJAS|} y_t = 1 \ \wedge \ y_t \in [\text{ínf}(w_t), \sup(w_t)] \ \forall t \in \{1, \ldots, |HOJAS|\}\Big\}$$

El conjunto de hojas que pueden ser expandidas se notaba como $HOJAS\_LIBRES$ y se definía así:

$$HOJAS\_LIBRES = \{i \in HOJAS \mid atributos\_libres_i \neq \emptyset \ \wedge \ rama_i > 0\}$$

Una vez hemos recordado los elementos que serán necesarios, podemos definir los componentes calculados y el nuevo procedimiento de la siguiente forma:

- $peores\_hojas \in (HOJAS \cup \{0\})^{num\_peores\_hojas}$ con $num\_peores\_hojas \in \mathbb{N}^+$
  $peores\_hojas = \{h_1, h_2, \ldots, h_{num\_peores\_hojas}\}$ donde

$$h_1 = \begin{cases} 0 & \text{si } HOJAS\_LIBRES = \emptyset \\ \text{mín } \{ \quad i \in HOJAS\_LIBRES \mid & \\ \qquad \sup(\varepsilon_i) \geq \sup(\varepsilon_j) & \\ \qquad \forall j \in HOJAS\_LIBRES \ \} & \text{si } HOJAS\_LIBRES \neq \emptyset \end{cases}$$

$$h_2 = \begin{cases} 0 & \text{si } HOJAS\_LIBRES - \{h_1\} = \emptyset \\ \text{mín } \{ \quad i \in HOJAS\_LIBRES - \{h_1\} \mid & \\ \qquad \sup(\varepsilon_i) \geq \sup(\varepsilon_j) & \\ \qquad \forall j \in HOJAS\_LIBRES - \{h_1\} \ \} & \text{si } HOJAS\_LIBRES - \{h_1\} \neq \emptyset \end{cases}$$

$$\vdots$$

$$h_{num\_peores\_hojas} = \begin{cases} 0 & \\ \text{si } HOJAS\_LIBRES - \{h_1, h_2, \ldots, h_{num\_peores\_hojas-1}\} = \emptyset & \\ \text{mín } \{i \in HOJAS\_LIBRES - \{h_1, h_2, \ldots, h_{num\_peores\_hojas-1}\} \mid & \\ \qquad \sup(\varepsilon_i) \geq \sup(\varepsilon_j) & \\ \qquad \forall j \in HOJAS\_LIBRES - \{h_1, h_2, \ldots, h_{num\_peores\_hojas-1}\} \} & \\ \text{si } HOJAS\_LIBRES - \{h_1, h_2, \ldots, h_{num\_peores\_hojas-1}\} \neq \emptyset & \end{cases}$$

Nótese que el vector con todos sus elementos iguales a 0 será el vector que represente que no quedan hojas disponibles para intentar ser expandidas. Por tanto, cuando el vector esté vacío ($peores\_hojas = (0, 0, \ldots, 0)$), la fase de expansión se dará por finalizada y se volverá a la fase de muestreo y recálculo (o se detendrá el algoritmo, según proceda).

- $peor\_hoja \in peores\_hojas$

  $peor\_hoja = h_1$

- RETIRAR_PEOR_HOJA

  En primer lugar, retira el primer elemento del conjunto con las peores hojas (hoja que será la que acabamos de intentar expandir) desplazando el resto de índices, de forma que en el siguiente paso sea seleccionado la hoja correspondiente:

  $$h_j = h_{j+1} \quad \forall j \in \{1, \ldots, num\_peores\_hojas - 1\}$$
  $$h_{num\_peores\_hojas} = 0$$

  En segundo lugar, recalcula la $peor\_hoja$ que será la siguiente hoja que se intentará expandir, según la definición que acabamos de dar ($peor\_hoja = h_1$).

**Estudio de la mejora y su complejidad**

El hecho de permitir que después de cada muestreo se pueda expandir más de una hoja ha supuesto una mejora real para el algoritmo, pero, como veremos a continuación, el número de hojas que se puedan expandir antes de cada nuevo muestro debe estar limitada. Para establecer ese límite hemos definido el parámetro interno llamado $num\_peores\_hojas$ y en la figura 3.4 presentamos cómo varía el comportamiento del algoritmo en función de ese parámetro. Para la elaboración de estas gráficas hemos usado el mismo conjunto de datos empleado en las secciones 2.3 y 2.4, manteniendo $\varepsilon = 0,01$, $\delta = 0,05$ y el resto de parámetros internos a sus valores por defecto.

En la figura 3.4 hemos representado el tamaño del árbol inducido, el número total de experiencias muestreadas antes de alcanzar el modelo buscado y el tiempo consumido. No mostramos la gráfica de la precisión final alcanzada, aunque sí hablaremos de ella en las consideraciones que a continuación enumeramos:

- La precisión alcanzada ha sido siempre superior a $99,00\,\%$, como era de esperar, puesto que el conjunto de datos no tiene ruido y el algoritmo se detiene al satisfacer los requisitos del usuario ($\varepsilon = 0,01$).

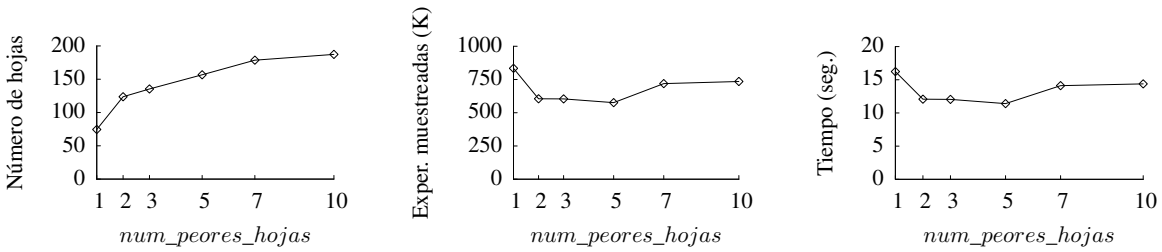



Figura 3.4: Comportamiento dependiendo del parámetro $num\_peores\_hojas$

- El tamaño del árbol aumenta conforme más hojas permitimos expandir después de cada muestreo (valores mayores para $num\_peores\_hojas$). Este comportamiento se debe principalmente a que el número de hojas que se están expandiendo es mayor que el mínimo necesario (que saldría de usar $num\_peores\_hojas = 1$). La razón es que las hojas que se están expandiendo (aquellas que han identificado claramente la idoneidad del mejor atributo para expandir) no siempre son las peores (conforme más avancemos en el conjunto ordenado $peores\_hojas$, menor será el error individual aportado por esa hoja al error de clasificación del árbol). Cuanto mayor sea el valor de $num\_peores\_hojas$, menor será la aportación a la mejora del modelo de las últimas hojas en la lista, con el consiguiente incremento del tamaño del árbol para alcanzar un modelo con la calidad requerida. En otros experimentos realizados, este incremento de tamaño no se pone tan de manifiesto para valores pequeños de $num\_peores\_hojas$ (valores de $num\_peores\_hojas$ entre 1 y 3 son poco sensibles a mostrar diferencias en el tamaño del árbol resultante).

- El número total de experiencias muestreadas, disminuye cuando permitimos realizar más de una expansión después de cada muestreo, siempre que limitemos su número. Como puede apreciarse, usando valores entre 2 y 5 para el parámetro interno $num\_peores\_hojas$, conseguimos reducir el número total de experiencias muestreadas. Si sólo permitimos una expansión, como ocurría en el algoritmo IADEM-0, estamos retrasando la inducción del modelo y se necesitará muestrear más experiencias. Por contra, si permitimos demasiadas expansiones, las expansiones de las hojas que menos error introducen en el modelo hacen que tengan que acompañarse de otras expansiones para alcanzar la calidad requerida, con el consiguiente aumento del tamaño del árbol (antes justificado) y del número total de experiencias muestreadas.

- El tiempo consumido hasta inducir el modelo presenta un comportamiento similar al observado en el número total de experiencias muestreadas. El principal motivo es que el tiempo consumido depende del número de experiencias muestreadas, pero también debemos considerar que, cuanto mayor sea $num\_peores\_hojas$, mayor número de hojas habrá que probar si son expansibles y mayor tiempo se invertirá.

Observando los resultados mostrados, podemos concluir que asignar valores pequeños, pero mayores que 1, al parámetro $num\_peores\_hojas$ es una buena estrategia para reducir el número total de experiencias necesarias antes de alcanzar el modelo requerido. Por defecto, el valor usado para el parámetro $num\_peores\_hojas$ ha sido fijado en 3.

Después de haber presentado formalmente la aportación realizada en este aspecto y tras haber realizado un estudio sobre el comportamiento del nuevo parámetro interno, comentamos las repercusiones que tiene este mecanismo desde el punto de vista de la complejidad. Atendiendo tanto a la complejidad temporal como espacial, la sobrecarga es mínima o nula. La cantidad de nuevas operaciones (de recálculo de componentes o de evaluación de predicados) que es preciso realizar en cada iteración del bucle principal se mantiene constante y tan solo se ve afectado el número de repeticiones por el valor del parámetro $num\_peores\_hojas$, que es el responsable del número de expansiones que se intentarán realizar. Cuanto mayor sea el valor de $num\_peores\_hojas$, más operaciones se realizarán entre cada dos muestreos, pero el tiempo empleado no

será desaprovechado mientras que los intentos de expansión conduzcan a expansiones que fuesen a realizarse con posterioridad. En cuanto a la complejidad espacial introducida podemos decir que es nula, puesto que sólo es necesario añadir el espacio necesario para almacenar un conjunto con las peores hojas, que será una estructura de tamaño constante y común para todo el algoritmo.

### 3.2.2. Detectando estabilización o empeoramiento del modelo

Ya se comentó en la sección 3.1 que una de las mayores deficiencias del algoritmo IADEM-0 es que no era tolerante al ruido. En esa misma sección se introdujo un método para detectar cuándo había ruido en el conjunto de datos en una cantidad superior al error máximo tolerado por el usuario ($\varepsilon$). Recordemos que la idea básica era estudiar si el valor estimado del error cometido por el árbol en el peor caso ($\sup(error)$) se incrementaba o se decrementaba cuando se producían expansiones.

Existen diversos problemas que se derivan del enfoque presentado en la sección 3.1 y que dio lugar al algoritmo IADEM. Podemos destacar dos situaciones que podrían llevar a que la detección del ruido se hiciese de forma prematura o de forma tardía: no se considera la cantidad en la que se ha visto incrementado el valor de $\sup(error)$ y es imprescindible que se produzca una expansión para evaluar dicho incremento.

Por una parte, valores muy pequeños de incrementos de $\sup(error)$ pueden hacer que se detecte ruido cuando el modelo aún podía ser algo más preciso. Se probaron diferentes variantes en las que se consideraba el comportamiento global de incrementos y decrementos, tanto en valores absolutos como relativos, pero aún así, seguía estando presente el segundo problema: sigue siendo imprescindible la ocurrencia de expansiones. Si no se producen expansiones, el predicado encargado de detectar ruido no recibirá nuevos datos (provenientes de nuevas expansiones) y la ejecución continuará sin que se detecte presencia de ruido. Como consecuencia, queda patente que es necesario buscar otro enfoque que evite los posibles casos anómalos que pueden darse al usar el anterior método para detectar ruido.

#### Presentando la solución propuesta

Puesto que la información almacenada en el árbol de decisión permite calcular el error cometido por el árbol para el peor caso ($\sup(error)$), podemos monitorizar su evolución, independientemente de si se produjeron expansiones o no, y tratar de detectar si hay convergencia en el aprendizaje. Es decir, proponemos sustituir la detección de comportamientos anómalos que sólo pueden estudiarse bajo determinadas circunstancias, por un estudio de la estabilización (convergencia) del aprendizaje que puede hacerse en todo momento. Existen diversos trabajos en esta línea, pero, como se afirma en el artículo de Provost, Jensen y Oates [Pro-1999a] o en la tesis de Castillo [Cas-2006], «*la detección de convergencia* [en el aprendizaje] *sigue siendo un problema abierto al que hace falta dedicar significativos esfuerzos de investigación*».

Una *curva de aprendizaje* muestra la relación entre el número de experiencias usadas para entrenar un modelo y la precisión alcanzada por ese modelo [Pro-1999a]. Como puede verse en la figura 3.5, esta curva la podemos mostrar en función de la precisión alcanzada o en función del error cometido. Este concepto de curva de aprendizaje puede extenderse, sin pérdida de generalidad, al aprendizaje incremental en el cuál, en vez de fijar el tamaño del conjunto de entrenamiento, se van procesando nuevas experiencias para mejorar el modelo. Las curvas de aprendizaje suelen presentar tres regiones: una primera en la que se produce un aprendizaje más rápido y que se corresponde con una pendiente pronunciada en la curva, una segunda en la que el aprendizaje ya no es tan rápido y que se corresponde con una pendiente suave, y una última región en la que el modelo apenas puede aprender nuevos conceptos y que se identifica con una meseta.

Uno de los enfoques más completos para estudiar el comportamiento de las curvas de aprendizaje desde la perspectiva de la teoría del aprendizaje computacional y de la estadística es el que se deriva de la dimensión de Vapnik-Chervonenkis [Vap-1971; Vap-1982]. La dimensión de VC permite predecir probabilísticamente una cota superior del error de generalización de un clasificador en función del error de entrenamiento y el número de experiencias con que se entrena. Aun así, se han identificado ciertas limitaciones [Bru-2004] a la hora de aplicar este enfoque en la práctica: sólo puede ser descrita para algoritmos de aprendizaje muy

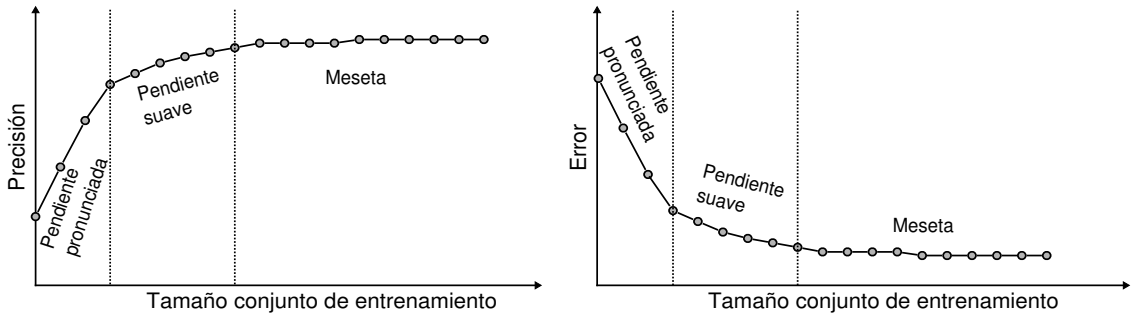



Figura 3.5: Curvas de aprendizaje teniendo en cuenta la precisión o el error.

sencillos (redes neuronales de una capa), no se mantiene constante para todos los algoritmos en función del número de experiencias con el que se entrena, etc.

El estudio de la curva de aprendizaje también se ha enfocado comúnmente desde la perspectiva de la extrapolación [Pro-1999a; Mee-2002]. Los métodos que intentan extrapolar la curva de aprendizaje usan los datos históricos para modelar su comportamiento buscando el mejor ajuste (usando todos los datos) o la mejor calidad de predicción (usando parte de los datos). Un estudio bastante extenso donde se comparan diversos métodos para la extrapolación de las curvas de aprendizaje puede encontrarse en el trabajo de Gu, Hu y Liu [Gu-2001]. En este mismo trabajo se recogen los principales usos que se le suelen dar a la información que se extraiga sobre la curva de aprendizaje: predecir cuál será la mayor precisión que se pueda alcanzar con el modelo, calcular cuántas experiencias harán falta para alcanzar la máxima precisión o determinar cuándo se puede detener la inducción de un modelo a tenor de la mayor ganancia en precisión que se pueda obtener.

Un enfoque más reciente para detectar la convergencia, basado en la detección del *comportamiento típico* de la curva de aprendizaje (la curva basada en la precisión es monótonamente creciente y cóncava o la curva basada en el error es monótonamente decreciente y convexa), puede encontrarse en el trabajo de Brumen y otros [Bru-2004] o en los de Castillo y Gama [Cas-2006; Cas-2006a]. Pero exigir que la curva de aprendizaje siga un comportamiento típico puede ser demasiado estricto. No todas las curvas de aprendizaje tienen que presentar ese comportamiento. Por ejemplo, análisis teóricos de curvas de aprendizaje basadas en mecanismos estadísticos [Wat-1993; Hau-1996] han demostrado que es posible que se produzcan descensos repentinos en la precisión (o incrementos repentinos en el error).

Lo que pretendemos con el método que a continuación presentamos es determinar a partir de qué punto podemos detener la inducción del algoritmo porque no se espere que la precisión pueda ser mejorada. Recordemos que nuestro interés no es detener la inducción lo antes posible porque suponemos que el tiempo que se tarde en alcanzar la solución o el número total de experiencias muestreadas no son lo prioritario. Estamos ante un enfoque incremental que va a extraer conocimiento de conjuntos de datos muy grandes o de flujos de datos potencialmente infinitos y lo que se pretende es alcanzar los modelos más precisos y sencillos que sea posible. El número de experiencias observadas o el tiempo consumido serán factores a considerar, pero no serán los prioritarios.

El enfoque que presentamos hace uso de la curva de aprendizaje basada en el error y toma como valores las estimaciones del error cometido en el peor caso ($\sup(error)$), cuyo cálculo es actualizado por el algoritmo constantemente. Por consiguiente, los valores de la curva de aprendizaje no son calculados mediante la proporción de experiencias que son mal clasificadas por el modelo actual. El método encargado de estudiar el comportamiento de la curva de aprendizaje sólo se preocupa de identificar cuándo se ha producido una meseta (una estabilización en el aprendizaje). No incluye como paso previo la detección del comportamiento típico de la curva de aprendizaje. Como se comentó en la sección 3.1 y se mostró en la figura 3.1, dicha curva puede presentar incrementos (situaciones anómalas) que distorsionarían el comportamiento típico de una curva de aprendizaje. Pero esas distorsiones son susceptibles de aparecer, más aún cuando el modelo está llegando a niveles de error cercanos al nivel de ruido en el conjunto de datos. Por esta razón el méto-

do que proponemos no espera a la aparición del comportamiento típico para detectar estabilización en el aprendizaje y tan sólo se centra en la detección de mesetas.

El estudio que realizamos sobre la curva de aprendizaje tiene elementos similares a los empleados por Castillo y Gama [Cas-2006; Cas-2006a] en lo que denominan *model-error learning curve*, pero es más sencillo y existen ciertas diferencias. Los valores que forman la curva son la primera diferencia: mientras ellos calculan la proporción de experiencias que son mal clasificadas por el modelo actual, nosotros usamos el valor estimado del error cometido en el peor caso ($\sup(error)$). Hemos de hacer notar, aunque se detallará más adelante, que los valores que forman la curva no son únicamente los valores del error después de cada iteración del bucle principal, sino cada modificación que sufra el árbol que se esta induciendo. Es decir, después de cada iteración en la que se hayan muestreado nuevas experiencias y no se hayan producido expansiones (se habrá añadido información al árbol) o después de cada expansión (se habrá producido una modificación en la estructura del árbol), se registrará el valor que tome $\sup(error)$. Recordemos que después de cada muestreo se intentan expandir un número determinado de hojas ($num\_peores\_hojas$), con lo que cada iteración del bucle producirá entre 1 y $num\_peores\_hojas$ nuevos valores que se incorporarán a la curva de aprendizaje (dependerá del número de expansiones que se produzcan). La otra gran diferencia es que hemos simplificado la detección de estabilización en el aprendizaje prescindiendo de la detección del comportamiento típico de la curva.

Antes de detallar formalmente la solución aportada debemos comentar que, además de la detección de estabilización, hemos incorporado un mecanismo bastante simple y eficaz para evitar un excesivo empeoramiento en el proceso de aprendizaje. Típicamente se evalúa cuándo parar la inducción incremental de modelos una vez alcanzada una meseta, pero el modelo se degradaría si por cualquier circunstancia el error empieza a incrementarse continuamente sin haberse detectado previamente ninguna meseta y no se detendría la inducción. Para evitar esta circunstancia controlamos en todo momento cuál fue el menor error estimado para el árbol y detendremos la inducción si se supera una cota por encima del mínimo. Debemos indicar que esta circunstancia no se ha dado en ninguno de los experimentos que mostraremos en el capítulo 4 donde se realiza el estudio experimental. Aun así, creemos necesario incorporar este mecanismo para evitar posibles degradaciones que pudieran surgir.

**Formalizando la solución propuesta**

En el pseudocódigo de la figura 3.3 hemos usado el predicado *Condición_Finalización* para controlar el cuerpo principal del bucle y para detener la fase de expansiones. Este predicado se define de la siguiente forma:

$$\begin{aligned} Condición\_Finalización \equiv\ & Condición\_Parada \vee \\ & Condición\_Completamente\_Expandido \vee \\ & Condición\_Estabilización \vee \\ & Condición\_Empeoramiento \end{aligned}$$

Hemos sustituido la condición que detectaba ruido (*Condición_Ruido*) en el pseudocódigo del algoritmo IADEM (en la figura 3.2) por la combinación de predicados que se encargan de detectar estabilización en el aprendizaje (*Condición_Estabilización*) o empeoramiento (*Condición_Empeoramiento*). Antes de definir estos dos predicados es preciso definir una serie de componentes y cómo se calculan.

Para la detección de estabilización en el aprendizaje usaremos un método geométrico más sencillo que el usado por Castillo y Gama [Cas-2006a] pero basado en él. El método no usa todos los valores de la curva de aprendizaje, sino que tan sólo emplea los más recientes. Cuando el valor del error cometido por el árbol se acerca al ruido en el conjunto de datos y no se puede alcanzar un modelo con mayor precisión, el valor de $\sup(error)$ tiene pequeñas fluctuaciones, incrementando o decrementando su valor; pero los valores se mantienen cercanos. Decidimos usar el área del triángulo determinado por los últimos valores de la curva de aprendizaje estableciendo un umbral mínimo para detectar una $meseta$. Esos valores son almacenados en el componente que se introdujo previamente con el nombre de $historia$ y cuyo tamaño será ahora definido

como un parámetro interno del algoritmo ($tamaño\_historia$). El uso que tiene el componente $historia$ es equivalente al que ofrecen en otros entornos las denominadas *ventanas* y usaremos indistintamente ambos términos. Cuando sucesivos puntos toman valores muy próximos, el área delimitada por el triángulo que une los dos extremos y el punto medio de la ventana será pequeña. En la figura 3.6 hemos representado una curva de aprendizaje en la que se han dibujado los triángulos delimitados por dos ventanas. En concreto el triángulo más a la izquierda puede visualizarse fácilmente puesto que su área es considerable, mientras que el otro triángulo está situado en la parte derecha de la gráfica y apenas puede distinguirse debido a su pequeña área. Los triángulos tendrán sus vértices etiquetados del siguiente modo: $(x_1, y_1)$ para el nodo más antiguo en la ventana, $(x_2, y_2)$ para el nodo en el centro de la ventana y $(x_3, y_3)$ para el nodo más reciente.

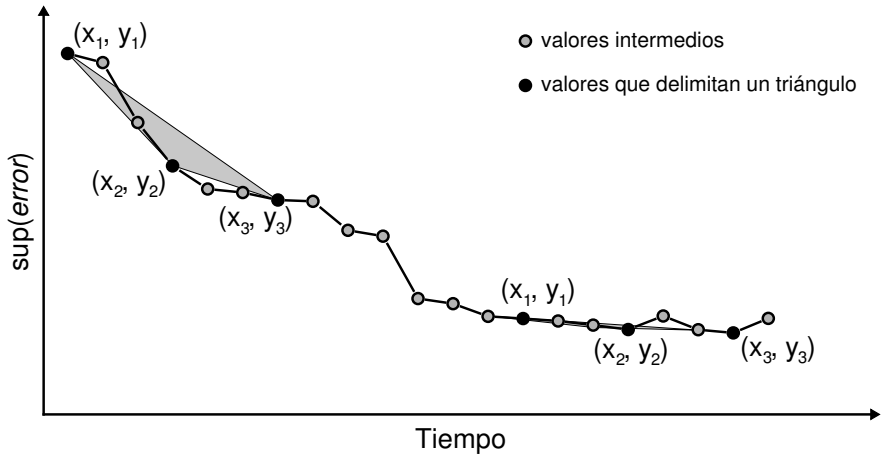



Figura 3.6: Áreas delimitadas en distintas ventanas de la curva de aprendizaje.

Debemos volver a definir el componente $historia$, que es el que mantiene la información necesaria sobre los valores de $\sup(error)$. Ahora no registrará las diferencias entre sucesivos valores de $\sup(error)$ sino que almacenará directamente sus valores. Será necesario incluir dos elementos que nos ayudarán a controlar la actualización de la historia ($num\_expansiones$) y la evaluación de su contenido ($contenido\_historia$).

También será preciso incluir dos elementos que serán imprescindibles para la evaluación de los predicados: uno registrará el número de mesetas consecutivas que se han producido ($num\_mesetas\_consec$ para el predicado *Condición_Estabilización*) y el otro almacenará el menor valor que haya alcanzado $\sup(error)$ ($min\_sup\_error$ para el predicado *Condición_Empeoramiento*). A continuación definimos todos los componentes necesarios incluyendo los vértices del triángulo contenido en $historia$:

- $tamaño\_historia \in \mathbb{N}^+ - \{1, 2\}$
- $historia \in (0, 1]^{tamaño\_historia}$
  donde $historia_1$ es el valor de $\sup(error)$ más antiguo almacenado y $historia_{tamaño\_historia}$ es el último valor de $\sup(error)$ que se ha registrado.
- $contenido\_historia \in \{0, \ldots, tamaño\_historia\}$: será el elemento que indique cuántos elementos forman parte de la $historia$. Hasta que la $historia$ no esté completa no puede usarse para evaluar el predicado de estabilización.
- $num\_expansiones \in \{0, \ldots, num\_peores\_hojas\}$: será el elemento donde se llevará la cuenta del número de expansiones que se han realizado entre cada dos muestreos. Recordemos que la $historia$ se actualiza después de cada expansión o antes de un nuevo muestreo si no se han producido expansiones.
- $num\_mesetas\_consec \in \mathbb{N}$ : llevará la cuenta del número de mesetas que se han producido de forma consecutiva. La definición formal del predicado $Meseta$ se dará más adelante, pero intuitivamente se corresponde con la aparición de una "*meseta*" en el comportamiento típico de la curva de aprendizaje.

- $min\_sup\_error \in (0, 1]$: registrará cuál ha sido el menor valor registrado por $\sup(error)$.
- $x_1, x_2, x_3 \in \{1, \ldots, tamaño\_historia\}$
  $x_1 = 1$
  $x_2 = \lceil tamaño\_historia/2 \rceil$
  $x_3 = tamaño\_historia$
- $y_1, y_2, y_3 \in (0, 1]$
  $y_1 = historia_{x_1}$
  $y_2 = historia_{x_2}$
  $y_3 = historia_{x_3}$

Los procedimientos deberán modificarse para mantener actualizados los distintos componentes que acabamos de definir. Antes de describir cómo son modificados debemos definir las operaciones que se realizan para actualizar el vector de historia, puesto que serán necesarias en más de una ocasión y facilitará la explicación. Así definimos:

$$actualizar\_historia \equiv historia_j = historia_{j+1} \quad \forall j \in \{1, \ldots, tamaño\_historia - 1\}$$

$$historia_{tamaño\_historia} = \sup(error)$$

$$contenido\_historia = \text{mín}\{tamaño\_historia, contenido\_historia + 1\}$$

$$num\_mesetas\_consec = \begin{cases} 0 & \text{si } \neg\, Meseta \\ num\_mesetas\_consec + 1 & \text{si } \quad Meseta \end{cases}$$

$$min\_sup\_error = \text{mín}\{\sup(error), min\_sup\_error\}$$

Procedemos, a continuación, a describir los cambios realizados en los procedimientos:

- INICIALIZAR
  - inicializamos el vector poniendo todos sus valores a "1" aunque en realidad no se lleguen a usar hasta que el contenido esté completo (se hayan registrado tantos valores de $\sup(error)$ como elementos haya en $historia$):
    $historia_j = 1 \quad \forall j \in \{1, \ldots, tamaño\_historia\}$
  - inicializamos los contadores que llevarán el número de expansiones que se han producido entre dos muestreos y el número de elementos registrados por $historia$:
    $contenido\_historia = 0$
    $num\_expansiones = 0$
    $num\_mesetas\_consec = 0$
    $min\_sup\_error = 1$
- MUESTREAR_Y_RECALCULAR
  - antes de muestrear ni recalcular ningún componente es preciso saber si no se produjo ninguna expansión en la última iteración, porque en ese caso es necesario añadir el último valor de $\sup(error)$ en $historia$:

    $$\text{si } (num\_expansiones = 0) \text{ entonces } actualizar\_historia$$

  - después de muestrear y antes de intentar realizar ninguna expansión reiniciamos el componente que controla el número de expansiones entre dos muestreos:

    $$num\_expansiones = 0$$

- EXPANDIR_ÁRBOL

  - después de cada expansión se siguen recalculando los componentes (incluyendo $\sup(error)$) y tenemos que actualizar la $historia$ y aumentar el contador de expansiones:

$$actualizar\_historia$$
$$num\_expansiones = num\_expansiones + 1$$

Una vez hemos definido los componentes necesarios y cómo se calculan, podemos definir el resto de elementos que son necesarios para evaluar los nuevos predicados. El área mínima para detectar estabilización ($área\_meseta$) que es un parámetro interno del algoritmo (cuyo comportamiento estudiaremos a continuación) y el área de un triángulo ($área$) se definen del siguiente modo:

- $área\_meseta \in \mathbb{R}^+$
- $área \in \mathbb{R}^+ \cup \{0\}$

$$área = \left|\frac{1}{2} \cdot (((x_2 \cdot y_1) - (x_1 \cdot y_2)) + ((x_3 \cdot y_2) - (x_2 \cdot y_3)) + ((x_1 \cdot y_3) - (x_3 \cdot y_1)))\right|$$

Pero detectar un área mínima no es suficiente para encontrarnos ante una estabilización del aprendizaje. Para tener una meseta no sólo es necesario que el área delimitada por el triángulo sea menor que un umbral, sino que, además, los puntos que definen dicho triángulo deben estar alineados formando un ángulo cercano a la horizontal. Para ello calcularemos la pendiente formada por los extremos de la ventana ($(x_1, y_1)$ y $(x_3, y_3)$) y fijaremos otro umbral mínimo ($pendiente\_meseta$) para considerar que es horizontal. El valor de este umbral vendrá determinado por el valor que se exija para considerar que el área del triángulo es pequeña ($área\_meseta$). Así definimos:

- $pendiente\_meseta \in \mathbb{R}^+$

  $pendiente\_meseta = +\sqrt{área\_meseta}$

- $pendiente \in \mathbb{R}^+ \cup \{0\}$

  $$pendiente = \left|\frac{y_3 - y_1}{x_3 - x_1}\right|$$

Usando estos elementos decimos que se ha producido una meseta en $historia$ cuando:

$$\begin{aligned} Meseta \quad \equiv \quad & (contenido\_historia = tamaño\_historia) \wedge \\ & (área \leq área\_meseta) \wedge \\ & (pendiente \leq pendiente\_meseta) \end{aligned}$$

Como último paso antes de formalizar los nuevos predicados debemos definir los umbrales que se tendrán en cuenta para detectar los comportamientos de "*estabilización*" o "*empeoramiento*". En el caso del umbral para detectar estabilización ($max\_mesetas\_consec$), su valor viene determinado por otros parámetros internos del algoritmo, mientras que el umbral para detectar empeoramiento ($max\_empeoramiento$) es explícitamente otro parámetro interno del algoritmo. Así:

- $max\_mesetas\_consec \in \mathbb{N}^+$
  $max\_mesetas\_consec = \text{máx}\{3, \lceil 10000/n \rceil\}$
- $max\_empeoramiento \in \mathbb{R}^+$

Una vez hemos definido todos los componentes que se necesitan para formalizar los nuevos predicados, pasamos a definirlos formalmente:

$$Condición\_Estabilización \equiv num\_mesetas\_consec \geq max\_mesetas\_consec$$

$$Condición\_Empeoramiento \equiv \sup(error) \quad \geq \quad min\_sup\_error + (min\_sup\_error \cdot max\_empeoramiento)$$

El valor de $max\_mesetas\_consec$ se ha definido de tal forma que la aparición esporádica de una meseta no provoque la detección de estabilización. Como mínimo hará falta detectar 3 mesetas consecutivas y el número máximo dependerá del número de experiencias muestreadas en cada paso ($n$). Cuanto menos experiencias se muestreen antes de intentar el proceso de expansión, más fácil será que no haya nueva información suficiente para expandir y que el valor de $\sup(error)$ cambie poco. Es por eso que, para valores pequeños de $n$, se tengan que exigir un mayor número de mesetas detectadas de forma consecutiva. La razón de usar 10000 en el numerador de la definición de $max\_mesetas\_consec$ es que el valor por defecto para el parámetro interno $n$ es también 10000. De esta forma, por defecto se exigirán 3 mesetas consecutivas, si $n$ fuese mayor se seguirían exigiendo 3 (para evitar mesetas esporádicas), pero si $n$ fuese menor haría falta detectar más mesetas consecutivas antes de que se produjese la detección de estabilización.

**Estudio de la mejora y su complejidad**

A partir de este momento, el estudio de los parámetros internos lo haremos usando un conjunto de datos diferente al que hemos usado con anterioridad. Ya que acabamos de introducir el tratamiento de ruido no tiene sentido seguir haciendo el estudio de los parámetros con conjuntos de datos sin ruido en los que no se puedan apreciar las características de cada parámetro interno. El nuevo conjunto de datos ha sido generado artificialmente con las siguientes características: el problema está caracterizado por 20 atributos ($a = 20$) con entre 4 y 6 valores para cada atributo ($m_i \in \{4, 5, 6\}\ \forall i \in \{1, \ldots, a\}$) y un atributo de clase con 4 posibles valores ($k = 4$); y el conjunto de datos está formado por un millón de experiencias siendo estas generadas usando un árbol de decisión creado aleatoriamente con 663 hojas habiendo introducido un ruido del 15 %. Los resultados que se presentan se han calculado después de realizar una validación cruzada con diez particiones para cada prueba. Los valores usados para la entrada al algoritmo han sido $\varepsilon = 0,01$ y $\delta = 0,05$ y los valores para los parámetros internos han sido los establecidos por defecto (excepto para aquellos que están siendo estudiados).

En este apartado hemos definido tres parámetros internos nuevos: dos se usan para la detección de estabilización en el aprendizaje y el otro se usa para detectar empeoramiento del modelo. Empezaremos estudiando cómo afecta al algoritmo el valor que se le dé al número de valores que se consideran en la $historia$ ($tamaño\_historia$). En la figura 3.7 mostramos cómo afecta a la precisión alcanzada por el modelo, al tamaño alcanzado y al número total de experiencias muestreadas.

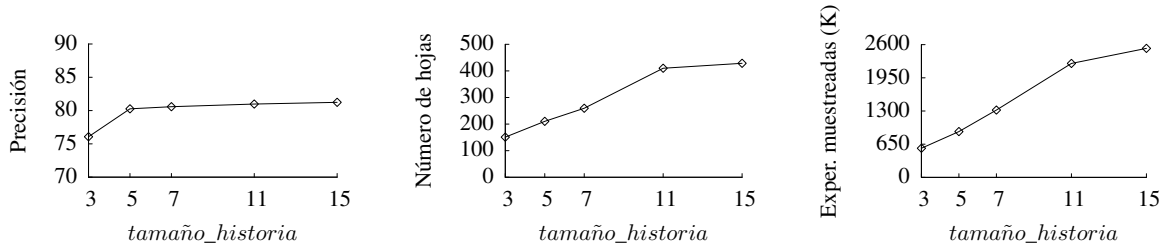



Figura 3.7: Comportamiento dependiendo del parámetro $tamaño\_historia$.

Las observaciones que se pueden hacer sobre las gráficas en la figura 3.7 pueden resumirse del siguiente modo:

- Si el tamaño de la ventana no es lo suficientemente grande (como ocurre en el caso de que el valor de $tamaño\_historia$ sea 3), la precisión alcanzada se ve mermada. La razón es que se produce una detección de la estabilización demasiado prematura. Hace falta un mayor tamaño de la ventana. A partir de valores de 5 ó 7, la precisión alcanzada apenas experimenta ligeros incrementos. Debemos hacer notar que la precisión alcanzada está lejos de parecerse a la solicitada por el usuario (recordemos que $\varepsilon = 0,01$), pero es lógico puesto que el ruido en el conjunto de datos es mucho mayor que el error máximo tolerado y el requisito del usuario nunca se podría ver satisfecho. La inducción se ha detenido por el predicado que estudia la estabilización.

- El tamaño del árbol aumenta conforme mayor es el tamaño de la ventana. Un mayor tamaño de la ventana hace que la detección de la estabilización se vea postergada (por ser más exigente). Esto es debido a que el área calculada para el triángulo definido en una ventana mayor será mayor que el área calculada con una ventana menor. Manteniendo el umbral necesario para considerar que el área es pequeña, lo que estamos haciendo al aumentar el tamaño de la ventana es exigir que los puntos que conforman el triángulo estén más alineados que en otros casos.

- El número total de experiencias muestreadas también aumenta cuanto mayor es la ventana. El motivo está en la misma línea del razonamiento anterior. Una mayor ventana hace más estricta la condición de estabilización y hace que se necesiten muestrear más experiencias para detener la inducción. El tiempo consumido hasta inducir el modelo, aunque no se muestre su gráfica en la figura, presenta un comportamiento similar al observado en el número total de experiencias muestreadas. La causa es la misma que hemos descrito anteriormente.

Teniendo en cuenta principalmente la precisión y el tamaño del árbol, el valor por defecto que hemos seleccionado para el parámetro $tamaño\_historia$ es 7. La razón principal es que con ese tamaño ya se alcanzan niveles de precisión máximos (o cercanos al máximo) y el tamaño del árbol sigue siendo reducido.

Pasamos ahora a estudiar el segundo parámetro interno que afecta al predicado encargado de detectar la estabilización en el aprendizaje: $área\_meseta$. En la figura 3.8 mostramos tres gráficas con el comportamiento observado en la precisión, la complejidad del modelo y el número de experiencias muestreadas.

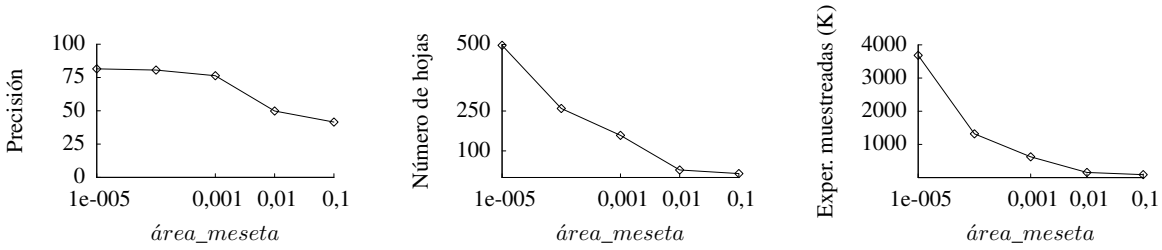



Figura 3.8: Comportamiento dependiendo del parámetro $área\_meseta$.

Resumimos a continuación las observaciones que se pueden hacer sobre las gráficas en la figura 3.8:

- Cuando el área mínima que debe alcanzar el triángulo definido con los puntos de la ventana es demasiado grande, la precisión decae rápidamente. El motivo es que el criterio de estabilización se activa de forma prematura. Es necesario requerir que el área mínima sea pequeña para obligar a que los puntos del triángulo se encuentren bastante alineados, lo que será señal de que estamos, efectivamente, ante una meseta (cumpliendo también que la meseta esté cerca de la horizontal).

- La consecuencia del razonamiento anterior nos lleva a que el tamaño del árbol sea menor cuanto mayor sea el área mínima requerida para detectar la meseta. Una detección prematura de la estabilización

hace que el modelo aún no recoja todo el conocimiento que puede extraerse, dando lugar a árboles de decisión demasiado simples.

- Por el mismo motivo anteriormente expuesto, tanto el número total de experiencias muestreadas como el tiempo consumido (que sigue un comportamiento similar aunque su gráfica no esté presente) son menores cuanto menor sea la exigencia impuesta para detectar la meseta.

Considerando el comportamiento que muestra el algoritmo en función de $área\_meseta$ hemos tomado el valor de $0,0001$ como valor por defecto para este parámetro interno. Consecuentemente, tenemos que $pendiente\_meseta = 0,01$, lo que se corresponde con una pendiente de poco más de medio grado $(0,5729)$ para considerar que los extremos de la ventana están próximos a la horizontal.

Por último estudiamos cómo afecta el valor que se le dé al parámetro interno $max\_empeoramiento$ a la inducción de árboles de decisión con el algoritmo IADEM-2. En la figura 3.9 volvemos a representar las mismas gráficas que en los casos anteriores: precisión, tamaño del árbol inducido y número total de experiencia muestreadas.

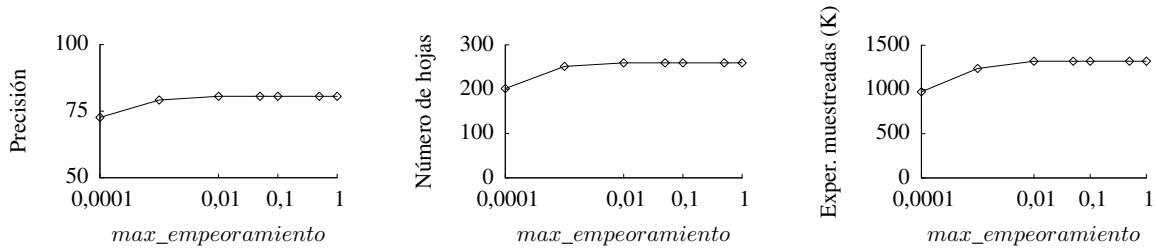



Figura 3.9: Comportamiento dependiendo del parámetro $max\_empeoramiento$.

Observando las gráficas de la figura 3.9 podemos sacar las siguientes conclusiones:

- Asignar valores pequeños al umbral para detectar empeoramiento puede provocar detenciones precipitadas en la ejecución del algoritmo que conducen a casos en los que la precisión aparece perjudicada. Debemos recordar que el umbral que se establece se usa de forma relativa a partir del valor de $\sup(error)$, permitiendo, por ejemplo, el valor de $0,1$ un empeoramiento del $10\,\%$ sobre el menor valor registrado.

- La misma razón anterior es la que influye en que el tamaño de los árboles sea menor: una detención prematura de la inducción no permite que el modelo termine de extraer todo el conocimiento de que es capaz. Un razonamiento idéntico es el que sirve para hacer que se muestreen menos experiencias y, consecuentemente, se consuma menos tiempo

A partir de las observaciones realizadas hemos fijado el valor por defecto para el parámetro interno $max\_empeoramiento$ en $0,1$, es decir, un $10\,\%$ de empeoramiento sobre el menor valor registrado de $\sup(error)$ ($min\_sup\_error$).

Después de haber presentado formalmente la aportación realizada en este aspecto y tras haber realizado este estudio sobre el comportamiento de los diferentes parámetros internos, comentamos las repercusiones que tienen estos mecanismos desde el punto de vista de la complejidad. Atendiendo tanto a la complejidad temporal como espacial, la sobrecarga es nula. Todas las operaciones se realizan sobre una estructura ($historia$) que tiene un tamaño máximo preestablecido ($tamaño\_historia$). Por tanto, el tiempo para hacer los cálculos que se necesitan para evaluar los predicados será constante y el espacio requerido para almacenar la información que se usará en dicha evaluación también será constante (y común a todo el algoritmo).

## 3.3. Mejorar la calidad del modelo: IADEM-2

En la sección anterior hemos descrito las aportaciones encaminadas a mejorar la convergencia del modelo y sus beneficios se mostrarán en el capítulo de experimentación (capítulo 4). Es ahora el turno de presentar las mejoras que persiguen mejorar la calidad del modelo intentando que sea más preciso al mismo tiempo que más sencillo. Para ello hemos incluido tres mejoras: hemos considerado los atributos continuos como tales sin tener que realizar un paso previo de discretización para poder procesarlos, hemos fijado un desbalanceo máximo para poder realizar una expansión y hemos reformulado el criterio para decidir cuándo se puede considerar la idoneidad del mejor atributo para realizar una expansión.

### 3.3.1. Considerando problemas con atributos continuos

La descripción del problema que habíamos considerado hasta este momento sólo contemplaba la posibilidad de usar atributos categóricos, pero existen infinidad de problemas reales en los que los atributos no son siempre categóricos y es habitual que aparezcan atributos continuos. Una de las soluciones más comunes es discretizar los atributos que sean continuos y convertirlos en categóricos. En el trabajo de Liu, Hussain, Tan y Dash [Liu-2002] puede encontrarse un buen resumen de las técnicas de *discretización*.

#### Presentando la solución propuesta

A la discretización previa y que sólo se realiza una vez antes de la fase de aprendizaje se la conoce como "*discretización estática*", mientras que a la discretización que se realiza conforme el modelo se va induciendo se la conoce como "*discretización dinámica*". Una comparación de ambos enfoques puede encontrarse en el trabajo de Dougherty, Kohavi y Sahami [Dou-1995]. Como ejemplos de la discretización estática podemos mencionar los métodos que realizan la partición basándose en la entropía [Cat-1991; Fay-1993] o los basados en la tasa de error [Hol-1993; Gir-2002]. Ejemplos de discretización dinámica tenemos en los algoritmos que inducen árboles de decisión como el ID3 [Qui-1986], el C4.5 [Qui-1993] o el VFDTc [Gam-2003; Gam-2006]. Dado que la discretización dinámica se desarrolla a la par que el proceso de inducción, los métodos de discretización dinámica suelen corresponderse con los métodos conocidos como de "*discretización local*" puesto que la discretización no es común para todo el conjunto de datos (que se correspondería con el enfoque "*global*" cuyos ejemplos más comunes son la discretización en intervalos de igual anchura o de igual frecuencia). La ventaja de la discretización local es que se ajusta a las necesidades del algoritmo conforme va siendo necesario sin imponer ninguna división a priori.

Existen métodos que realizan la discretización estática para grandes conjuntos de datos o para flujos de datos [Gam-2006a], haciendo la discretización de los atributos de forma independiente. Pero, si queremos aprovechar las particularidades de cada atributo en cada situación, es más conveniente usar un enfoque dinámico que discretice los atributos continuos según va siendo necesario y teniendo en cuenta las decisiones que se han tomado previamente por el algoritmo. La dificultad que presentan los algoritmos tradicionales para afrontar este problema es que necesitan realizar una operación de ordenación que es muy costosa computacionalmente [Cat-1991a].

Para abordar este problema en el campo del aprendizaje incremental se han seguido diferentes enfoques que evitan los enfoques tradicionales por ser demasiado costosos. Estos enfoques van desde el uso de histogramas, a la presunción de normalidad en la distribución, pasando por almacenar toda la información disponible. En el trabajo de Jin y Agrawal [Jin-2003] se usan histogramas con intervalos de igual anchura para almacenar la información que será necesaria para decidir sobre la mejor partición y, para limitar la memoria que usarán, se fija un límite máximo de intervalos podando los intervalos numéricos menos prometedores. Otra alternativa que se presenta es la discretización dinámica cualitativa propuesta por Mora, Fortes, Morales y Triguero [Mor-2000]. En esta aproximación no se establecen de forma estricta los puntos de corte que separan los intervalos contiguos, sino que se tiene en cuenta la evolución de la serie para ajustarlos. Por esa razón su uso está más orientado al área de las series temporales [Mor-2005]. En los trabajos de Gama, Medas y Rocha [Gam-2004] y Holmes, Kirkby y Pfahringer [Hol-2005] se asume que la distribución pre-

sentará un comportamiento normal y hacen uso de la aproximación gaussiana para realizar sus estimaciones. Con dicha aproximación consiguen disminuir los requisitos de memoria puesto que sólo precisan almacenar la media y la varianza. Contrapuesta a esta aproximación gaussiana, que presume la normalidad en la distribución, nos encontramos con otros trabajos de Gama, Rocha, Medas y Fernandes [Gam-2003; Gam-2006] en los que se emplea una estructura de datos para mantener todos los datos observados de forma ordenada y de la que sea sencillo extraer la información para seleccionar la mejor discretización (en dos intervalos). Esa estructura es un árbol binario que permite tiempos de acceso (o inserción) $\mathcal{O}(\log_2 n)$ en el mejor caso y $\mathcal{O}(n)$ en el peor caso (siendo $n$ el número de valores distintos almacenados en el árbol binario). Nosotros hemos recogido este último enfoque para adaptarlo a nuestro problema y dotar al algoritmo IADEM-2 con la capacidad de poder trabajar con atributos continuos (dicho enfoque ya ha sido usado en una modificación del algoritmo IADEM llamada IADEMc [Cam-2006]).

**Formalizando la solución propuesta**

Para recoger información sobre los diferentes valores que tomen los atributos continuos y mantenerla ordenada hemos optado por usar un árbol binario balanceado. Esta estructura de datos ofrece tiempos de acceso a los datos y de inserción de nuevos datos que son $\mathcal{O}(\log_2 n)$ para todos los casos [Pen-1998], mejorando así al árbol binario no balanceado.

Para tratar experiencias con atributos continuos ($atributos\_continuos$) el árbol de decisión se ha visto modificado y los nodos de decisión ya no estarán etiquetados únicamente con atributos categóricos. Se ha incluido un nuevo tipo de nodo de decisión que introduce un punto de corte numérico ($pc \in \mathbb{R}$) y del que dependerán dos ramas que se corresponderán con los valores menores o iguales que el punto de corte ($\leq pc$) o con los valores mayores que el punto de corte ($> pc$). Cuando el algoritmo decida que la mejor decisión a tomar en un nodo es expandir usando un atributo continuo ($X_j$) por un punto de corte concreto ($pc$), se creará un nodo dicotómico donde se instalará la pregunta "¿$X_j \leq pc$?", que tendrá dos posibles respuestas (ramas): $V$ (verdadero) o $F$ (falso). Siendo dicotómicas las divisiones que vengan de atributos continuos, éstos siempre serán considerados como atributos susceptibles de ser usados en cualquier hoja ($X_j \in atributos\_libres_i \quad \forall i \in HOJAS \quad \wedge \quad \forall X_j \in atributos\_continuos$) para así poder realizar cualquier discretización que suponga más de dos intervalos.

En la figura 3.10 hemos representado un ejemplo de árbol de decisión que usa atributos continuos en algunos nodos. Como puede apreciarse, un mismo atributo continuo puede ser usado más de una vez con distintos puntos de corte. La información necesaria para estudiar las expansiones se sigue guardando en las hojas virtuales, como ocurría cuando sólo se usaban atributos categóricos, pero ahora existe una diferencia importante. Para los atributos continuos ($l' \in atributos\_continuos$) no se conoce exactamente el número de valores diferentes ($diferentes_{l'} \in \mathbb{N}$) para los que habrá que guardar información y por eso se han sustituido las estructuras estáticas por estructuras dinámicas. En concreto, como ya se ha mencionado, hemos usado árboles binarios balanceados.

Recordemos que las hojas virtuales ya definidas para ser utilizadas con atributos categóricos se nombran como $(i, l, v)$, donde $i$ es la hoja real de la que depende y $v$ es el valor del atributo libre $l$ del que se está almacenando información ($v \in X_l$). Nombraremos con $(i, l', \text{MENOR\_IGUAL}(pc))$ a la hoja virtual que depende de la hoja $i$ y que almacena la información del número de experiencias que son menores o iguales que el punto de corte $pc$ en el atributo continuo $l'$; y nombraremos $(i, l', \text{MAYOR}(pc))$ a la que depende de la hoja $i$ y que almacena la información del número de experiencias que son mayores que el punto de corte $pc$ en el atributo continuo $l'$.

Llamaremos $l'_j$ al $j$-ésimo menor valor almacenado en el árbol binario balanceado para el atributo continuo $l'$ ($l'_1$ será el menor valor del atributo $l'$ y $l'_{diferentes_{l'}}$ será el mayor) y llamaremos $pc_{(l',p)}$ al $p$-ésimo menor punto de corte en el árbol binario balanceado para el atributo $l'$ ($pc_{(l',1)}$ será el menor punto de corte del atributo $l'$ y $pc_{(l',diferentes_{l'}-1)}$ será el mayor). Los puntos de corte serán los puntos intermedios entre dos valores reales, por tanto, definimos $pc_{(l',p)} = (l'_p + l'_{p+1})/2 \; \forall p \in \{1, \ldots, diferentes_{l'} - 1\}$.

La información contenida en el árbol binario balanceado será necesaria para calcular la información de las hojas virtuales resultantes de los atributos continuos. La razón para usar esta estructura es que se

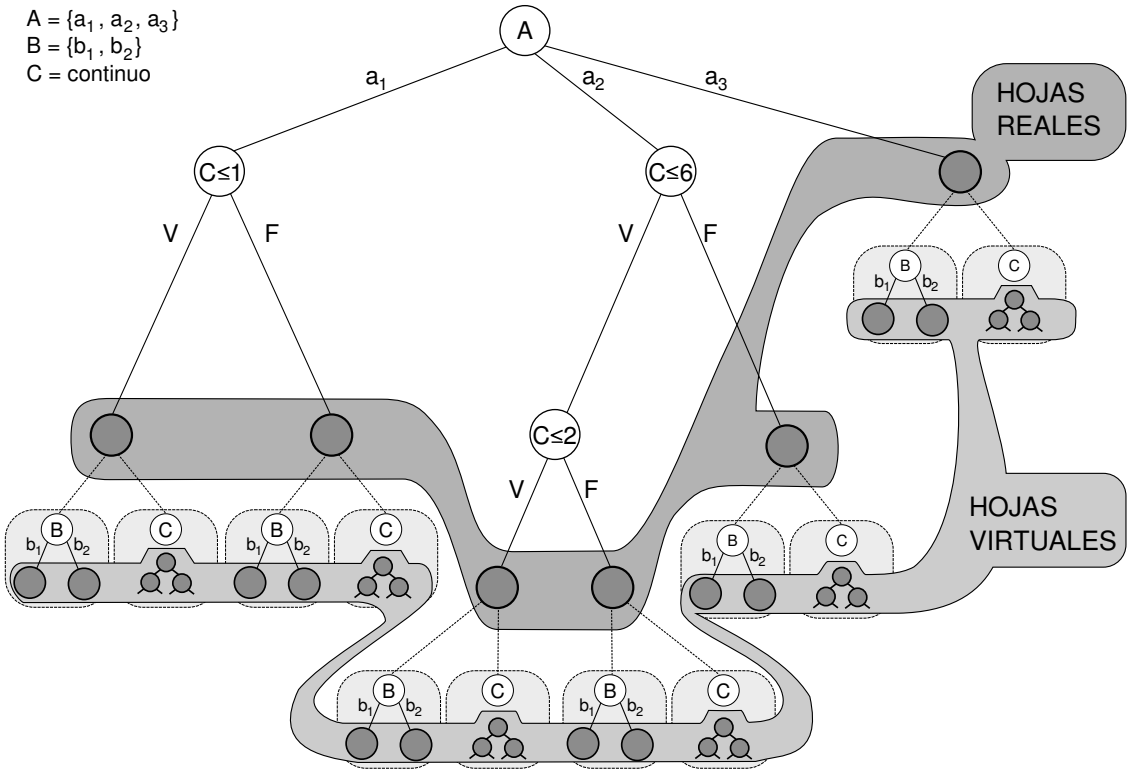



Figura 3.10: Ejemplo de árbol de decisión con atributos categóricos y continuos.

podrán realizar todos los cálculos para estudiar el mejor punto de corte de una forma eficiente (haciendo un único recorrido por el árbol que ya está ordenado). En cada hoja real del árbol se tendrán tantos árboles binarios balanceados como atributos continuos se hayan definido, independientemente de que se hayan usado anteriormente en la rama correspondiente. Por tanto, de cada hoja real dependerán las siguientes hojas virtuales:

- Por atributos categóricos: tantas como atributos libres estén sin usar ($|atributos\_libres_i|$) multiplicado por el número de valores de cada atributo ($m_l$ siendo $l \in atributos\_libres_i$).
- Por atributos continuos: tantas como atributos continuos existan ($|atributos\_continuos|$) multiplicado por 2 y por el número de puntos de corte de cada atributo ($diferentes_{l'} - 1$ siendo $l' \in atributos\_continuos$). Se multiplica por 2 porque existirán dos posibilidades por cada punto de corte: ser menor o igual, o ser mayor.

Con lo que acabamos de ver, para un atributo categórico ($l$) en una hoja virtual sólo tenemos que considerar una medida de desorden ($medida_i(l)$), mientras que para una hoja virtual con un atributo continuo ($l'$) tendremos que considerar una medida de desorden ($medida_i(l', pc)$) por cada punto de corte del atributo continuo ($pc \in \{pc_{(l',p)} | p \in \{1, \ldots, diferentes_{l'} - 1\}\}$). Antes de definir cómo se calculan esas medidas de desorden para todos los posibles puntos de corte debemos presentar la estructura que hemos usado para almacenar toda la información necesaria. En la figura 3.11 hemos representado un esquema con la información que se almacena para cada árbol binario balanceado. Tendrá un contador general que registrará el número de valores diferentes que se han observado (que se corresponderá con el número de nodos en el árbol binario): $diferentes_{l'} \in \mathbb{N}$. Cada uno de los nodos en el árbol binario lleva una información similar a la usada en las hojas virtuales de los atributos categóricos, pero con la peculiaridad de que hay que anotar cuál es el valor para el que se están contando los eventos. Definimos así los siguientes contadores:

- $v' \in \mathbb{R}$ que es un valor concreto del atributo continuo.

- $r'_{(i,l',v')} \in \mathbb{N}$ que lleva la cuenta del número de experiencias que tienen el valor $v'$ para el atributo $l'$.
- $r'_{(i,l',v'),z} \in \mathbb{N}$ que lleva la misma cuenta anterior pero considerando los distintos valores de la clase.

Los contadores $r'_{(i,l',v')}$ y $r'_{(i,l',v'),z}$ son equivalentes a los componentes básicos definidos en la subsección 2.2.2 y que se usan en las hojas virtuales de atributos categóricos. Para el cálculo de los componentes que definiremos a continuación también usaremos la información de los componentes básicos $rama_i$ y $rama_{i,z}$ que llevan, respectivamente, la cuenta del número de experiencias registradas por la hoja virtual y la misma cuenta pero considerando la etiqueta de clase (definidos también en la subsección 2.2.2).

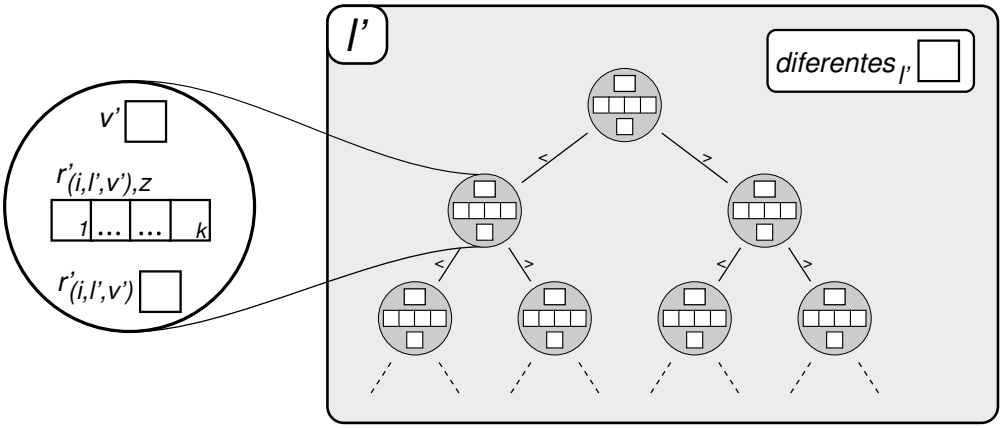



Figura 3.11: Información almacenada en un árbol binario balanceado (atributo $l'$).

Cuando se observa una nueva experiencia, se actualizan los nodos correspondientes en los árboles binarios balanceados que hay en cada una de las hojas virtuales. Si no existiese aún, se crearía un nuevo nodo actualizando sus contadores (todos valdrían 0 excepto los que deban ser actualizados). Un ejemplo con un caso particular de un árbol binario balanceado con la estructura que acabamos de describir puede observarse en la figura 3.12.

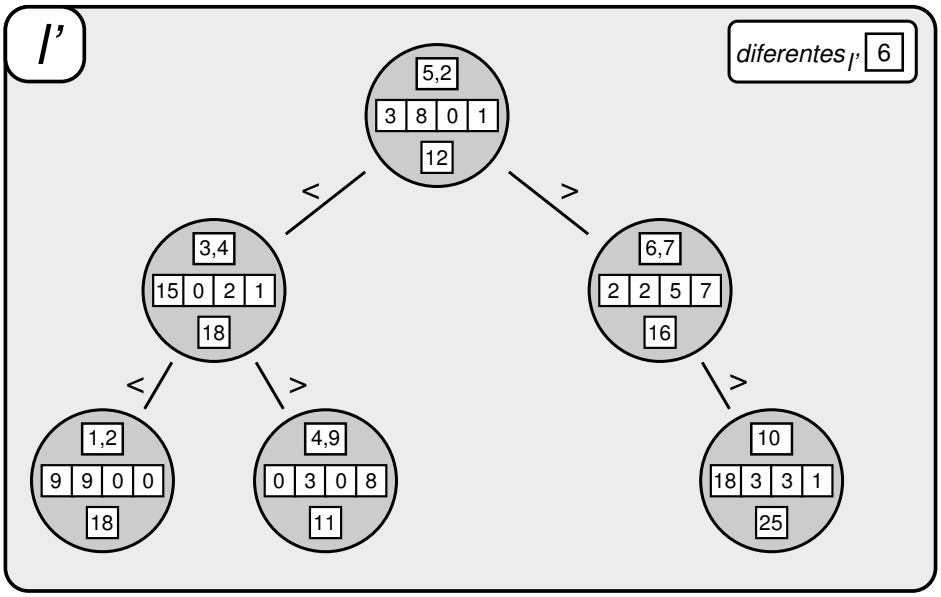



Figura 3.12: Ejemplo de un árbol binario balanceado (atributo $l'$).

Para calcular las medidas de desorden de un atributo categórico usamos las"*probabilidades de que una experiencia que alcance la hoja virtual* $(i, l, v)$ *esté etiquetada con la clase* $z$" ($P_{(i,l,v),z}$) siendo $l$ un atributo categórico y $v$ alguno de los valores que puede tomar dicho atributo ($v \in X_l$). Pero, para calcular las medidas de desorden de cada uno de los posibles puntos de corte tendremos que calcular la "*probabilidad de que una experiencia que alcance la hoja virtual* $(i, l', \text{MENOR\_IGUAL}(pc))$ *esté etiquetada con la clase* $z$" ($P_{(i,l',\text{MENOR\_IGUAL}(pc)),z}$) y la "*probabilidad de que una experiencia que alcance la hoja virtual* $(i, l', \text{MAYOR}(pc))$ *esté etiquetada con la clase* $z$" ($P_{(i,l',\text{MAYOR}(pc)),z}$). Los cálculos que realizaremos para calcular estas probabilidades siguen la misma línea del algoritmo de no ofrecer valores concretos, sino intervalos en los que se pueden encontrar dichas componentes calculados (los límites vendrán dados por los valores en el mejor y en el peor caso).

Veamos primero cómo se calcula la "*probabilidad de que una experiencia que alcance la hoja virtual* $(i, l', \text{MENOR\_IGUAL}(pc))$ *esté etiquetada con la clase* $z$":

$P_{(i,l',\text{MENOR\_IGUAL}(pc)),z} \in [0,1]^2$

$P_{(i,l',\text{MENOR\_IGUAL}(pc)),z} = (\text{ínf}(p_{(i,l',\text{MENOR\_IGUAL}(pc)),z}), \sup(p_{(i,l',\text{MENOR\_IGUAL}(pc)),z}))$

donde:

$$\text{ínf}(p_{(i,l',\text{MENOR\_IGUAL}(pc)),z}) = \begin{cases} \text{máx}\{0, \quad frec_{\leq pc,z} - \\ \qquad\qquad margen\_\varepsilon(total_{\leq pc}, frec_{\leq pc,z}, \delta)\} & \text{si } total_{\leq pc} \neq 0 \\ 0 & \text{si } total_{\leq pc} = 0 \end{cases}$$

$$\sup(p_{(i,l',\text{MENOR\_IGUAL}(pc)),z}) = \begin{cases} \text{mín}\{1, \quad frec_{\leq pc,z} + \\ \qquad\qquad margen\_\varepsilon(total_{\leq pc}, frec_{\leq pc,z}, \delta)\} & \text{si } total_{\leq pc} \neq 0 \\ 1 & \text{si } total_{\leq pc} = 0 \end{cases}$$

con $frec_{\leq pc,z} = \dfrac{parcial_{\leq pc,z}}{total_{\leq pc}}$ ,

$$parcial_{\leq pc,z} = \sum_{p=1}^{pc} r'_{(i,l',p),z} \quad \text{y}$$

$$total_{\leq pc} = \sum_{p=1}^{pc} r'_{(i,l',p)}.$$

El cálculo de la "*probabilidad de que una experiencia que alcance la hoja virtual* $(i, l', \text{MAYOR}(pc))$ *esté etiquetada con la clase* $z$" es similar al anterior:

$P_{(i,l',\text{MAYOR}(pc)),z} \in [0,1]^2$

$P_{(i,l',\text{MAYOR}(pc)),z} = (\text{ínf}(p_{(i,l',\text{MAYOR}(pc)),z}), \sup(p_{(i,l',\text{MAYOR}(pc)),z}))$

donde:

$$\text{ínf}(p_{(i,l',\text{MAYOR}(pc)),z}) = \begin{cases} \text{máx}\{0, \quad frec_{>pc,z} - \\ \qquad\qquad margen\_\varepsilon(total_{>pc}, frec_{>pc,z}, \delta)\} & \text{si } total_{>pc} \neq 0 \\ 0 & \text{si } total_{>pc} = 0 \end{cases}$$

$$\sup(p_{(i,l',\text{MAYOR}(pc)),z}) = \begin{cases} \text{mín}\{1, \quad frec_{>pc,z} + \\ \qquad\qquad margen\_\varepsilon(total_{>pc}, frec_{>pc,z}, \delta)\} & \text{si } total_{>pc} \neq 0 \\ 1 & \text{si } total_{>pc} = 0 \end{cases}$$

con $frec_{>pc,z} = \dfrac{parcial_{>pc,z}}{total_{>pc}}$ ,

$$parcial_{>pc,z} = \sum_{p=pc+1}^{diferentes_{l'}} r'_{(i,l',p),z} \quad \text{y}$$

$$total_{>pc} = \sum_{p=pc+1}^{diferentes_{l'}} r'_{(i,l',p)}.$$

La forma de calcular todas las probabilidades de forma eficiente es recorrer todos los posibles puntos de corte del árbol secuencialmente, desde el menor ($pc_{(l',1)}$) hasta el mayor ($pc_{(l',diferentes-1)}$), llevando almacenada la cuenta de experiencias de cada clase $z$ en ambos lados del punto de corte ($parcial_{\leq pc,z}$ y $parcial_{>pc,z}$) y el total de experiencias en cada lado del corte ($total_{\leq pc}$ y $total_{>pc}$). En la evaluación de cada nuevo punto de corte bastará con sumar las cantidades correspondientes del nodo que cambia de lado del corte a las variables que quedan en el lado "*menor o igual*" y restarle a las que quedan del lado "*mayor*".

Por ejemplo, para el atributo $l'$ en la hoja $i$ tenemos que, para el primer punto de corte ($pc_{(l',1)}$):

$$\begin{aligned} parcial_{\leq pc_{(l',1)},z} &= r'_{(i,l',l'_1),z} \\ parcial_{>pc_{(l',1)},z} &= rama_{i,z} - r'_{(i,l',l'_1),z} \\ total_{\leq pc_{(l',1)}} &= r'_{(i,l',l'_1)} \\ total_{>pc_{(l',1)}} &= rama_i - r'_{(i,l',l'_1)} \end{aligned}$$

Y calcular los valores para un punto de corte ($pc_{(l',x)}$) conociendo los del corte anterior ($pc_{(l',x-1)}$) sería tan sencillo como hacer lo siguiente:

$$\begin{aligned} parcial_{\leq pc_{(l',x)},z} &= parcial_{\leq pc_{(l',x-1)},z} + r'_{(i,l',l'_x),z} \\ parcial_{>pc_{(l',x)},z} &= parcial_{>pc_{(l',x-1)},z} - r'_{(i,l',l'_x),z} \\ total_{\leq pc_{(l',x)}} &= total_{\leq pc_{(l',x-1)}} + r'_{(i,l',l'_x)} \\ total_{>pc_{(l',x)}} &= total_{\leq pc_{(l',x-1)}} - r'_{(i,l',l'_x)} \end{aligned}$$

**Estudio de la mejora y su complejidad**

Puesto que esta mejora no incluye ningún parámetro interno que sea necesario estudiar para dar su configuración por defecto, una buena forma de presentar la mejora introducida es mostrar los resultados alcanzados por el algoritmo IADEM-2 usando la discretización estática de atributos continuos y usando discretización dinámica. Para la discretización estática hemos realizado una división de los atributos en 7 intervalos de igual anchura y para la discretización dinámica hemos usado el enfoque que acabamos de describir. En el siguiente cuadro (cuadro 3.1) se muestran los resultados de aplicar el algoritmo con las dos formas de discretizar a tres conjuntos de datos del repositorio de la UCI [New-1998]. En el capítulo donde presentamos la experimentación (capítulo 4) se muestran más datos sobre estos conjuntos de datos y su comparación con otros algoritmos, pero en este momento lo que queremos destacar es la mejora que se produce al usar la discretización dinámica propuesta. En el cuadro hemos nombrado como "IADEM-2 (discreto)" a la versión que usa los atributos continuos como si fueran categóricos (previamente discretizados en 7 intervalos) y llamamos "IADEM-2 (continuo)" a la versión propuesta (que usa la información almacenada en los árboles binarios balanceados).

Como puede observarse en el cuadro 3.1, la mejora derivada de usar los atributos continuos como tales y hacer divisiones dicotómicas es manifiesta en todos los aspectos. Los árboles que se inducen son más pequeños y, por lo tanto, más sencillos de interpretar. Aunque son más sencillos, recogen mejor el conocimiento, puesto que la precisión que se alcanza es bastante superior. La solución alcanzada no requiere que se muestreen tantas experiencias (salvo una excepción) lo que lleva a que se precise menos tiempo para converger a la solución. Un menor tiempo a pesar del mayor coste que supone almacenar la información en la estructura dinámica.

Puede llamar la atención la alta precisión alcanzada en el caso del conjunto de datos "*Ionosphere*" cuando el árbol inducido únicamente tiene una hoja, pero la razón reside en las aportaciones realizadas al proceso de predicción que comentaremos en la sección 3.4. El algoritmo IADEM-2 incorpora una serie de aportaciones como ya se introdujo al comenzar este capítulo, pero aún no hemos tenido oportunidad de presentarlas todas en detalle. A pesar de no haberlas descrito todas aún, en las ejecuciones del algoritmo IADEM-2 se consideran todas las aportaciones y se activan o desactivan las mejoras que están siendo estudiadas. En este

| Datos | Algoritmo | Hojas | Precisión | Experiencias (en miles) | Tiempo (en seg.) |
|---|---|---|---|---|---|
| Balance | | | | | |
| | IADEM-2 (discreto) | 9,40 ± 14,42 | 84,18 ± 11,52 | 348 | 4,18 |
| | IADEM-2 (continuo) | 8,10 ± 9,37 | 87,36 ± 4,49 | 380 | 3,55 |
| Ionosphere | | | | | |
| | IADEM-2 (discreto) | 68,20 ± 27,97 | 73,50 ± 6,34 | 367 | 26,34 |
| | IADEM-2 (continuo) | 1,00 ± 0,00 | 87,73 ± 4,70 | 238 | 8,62 |
| Pima-indians | | | | | |
| | IADEM-2 (discreto) | 258,40 ± 131,04 | 67,31 ± 3,21 | 966 | 24,15 |
| | IADEM-2 (continuo) | 8,10 ± 2,08 | 72,92 ± 6,19 | 428 | 5,17 |

Cuadro 3.1: Estudio de la mejora al trabajar directamente con atributos continuos.

caso, en el que estudiamos la influencia del uso de la discretización dinámica, ésta es la única mejora que activamos o desactivamos, pero la mejora realizada en el proceso de predicción (basado en clasificadores ingenuos de Bayes) está activa. Esta última mejora es la que permite alcanzar la alta precisión registrada a pesar de sólo disponer de una hoja en el árbol de decisión.

En el ámbito de la complejidad, usar esta mejora introduce algunas variantes con respecto al estudio realizado en la sección 2.5, donde únicamente se contemplaban atributos categóricos. Para analizar la complejidad introducida debemos considerar el peor caso y para ello definimos las siguientes variables:

- $a$: número de atributos descritos en el problema.
- $b'$: número máximo de valores diferentes en los atributos categóricos. Supone considerar, en el peor caso, que todos los atributos tienen el mismo número de valores y este número es $b'$. Esta variable es llamada $b$ en la sección 2.5, es decir, $b = b'$.
- $b''$: número máximo de valores diferentes en los atributos continuos. Supone considerar, en el peor caso, que todos los atributos tienen el mismo número de valores y este número es $b''$.
- $k$: número de valores de la clase.
- $d$: profundidad máxima del árbol inducido hasta el momento, y en el que aún no se ha utilizado ningún atributo categórico (con lo que es necesario almacenar toda la información referida a atributos categóricos y continuos).
- $h$: número máximo de hojas del árbol inducido con profundidad $d$. Se trata de una variable calculada que se definiría como $b^d$.

Al incluir el tratamiento de atributos continuos cambian los tiempos de cómputo necesarios para ejecutar distintos procedimientos y predicados. Los cambios relevantes son los que afectan al procedimiento MUESTREAR_Y_RECALCULAR puesto que hay que almacenar la información de una forma diferente y hay que recalcular los componentes necesarios para el proceso de expansión usando estructuras de datos diferentes. Para actualizar los contadores de todos los atributos que existen en las hojas virtuales del árbol (que en el peor caso aún no han usado ningún atributo categórico) se necesita una cantidad de tiempo $\mathcal{O}(a \cdot \text{máx}\{b', \log_2 b''\} \cdot k)$. Se ha sustituido $(a - d)$ (que aparecía en la sección 2.5) por $a$, porque en el peor caso no se han usado todavía ninguno de los atributos categóricos; y se ha incluido $(\text{máx}\{b', \log_2 b''\})$ porque insertar un valor concreto en el árbol binario balanceado que almacena la información de un atributo continuo consume un tiempo $\mathcal{O}(\log_2 b'')$. Para recalcular los componentes que se necesitan para el proceso de expansión se consume un tiempo $\mathcal{O}(a \cdot \text{máx}\{b', b''\} \cdot k \log_2 k)$. Se ha sustituido $(a - d)$ por $a$ por el mismo

motivo y se ha cambiado $b$ por $(\max\{b', b''\})$ porque actualizar los datos en el árbol binario balanceado se puede hacer en tiempo lineal dependiendo del número de valores diferentes que haya almacenado ($b''$).

La complejidad espacial se ve menos afectada en cuanto al espacio requerido para guardar la información de los atributos continuos, ya que únicamente se utiliza un vector de tamaño $k$ por cada valor diferente observado (habiendo un máximo de $b''$ valores diferentes). Lo que sí puede afectar al tamaño es que no se utilicen atributos categóricos para realizar las expansiones, lo que supondría un aumento en los requisitos de memoria que teníamos anteriormente. Al seguir considerando todos los atributos categóricos (porque no se ha utilizado ninguno) y todos los continuos (porque se pueden usar siempre), los requisitos de memoria aumentan en el peor caso. Así, la complejidad espacial al añadir el tratamiento de atributos continuos pasa a ser $\mathcal{O}(a \cdot 2^d \cdot \max\{b', b''\} \cdot k)$. Se ha sustituido $(a - d)$ por $a$ porque se consideran todos los atributos del problema y hemos cambiado $(h \cdot b)$ por $(2^d \cdot \max\{b', b''\})$ porque las expansiones son dicotómicas (reemplazamos $h$ por $2^d$) y porque debemos considerar el mayor número de valores diferentes (reemplazamos $b$ por $\max\{b', b''\}$). En este caso no es posible determinar un tamaño máximo para el árbol, como hicimos cuando el problema sólo usaba atributos categóricos, porque los atributos continuos siempre pueden ser usados para expandir y necesitamos mantener información para ellos.

### 3.3.2. Fijando un desbalanceo máximo para expandir

A la hora de realizar una expansión podemos encontrarnos en la situación de que una de las nuevas ramas que se creen concentren la mayor parte de las experiencias que se hayan observado. Mientras que esa mayor parte no sea excesivamente predominante sobre el resto, los niveles de desbalanceo no serán demasiado grandes. Por el contrario, cuando esa parte mayoritaria sea muy superior al resto, es fácil que el hecho de haber realizado la expansión apenas suponga una mejora, puesto que el árbol de decisión se hará más complejo (tendrá más ramas y serán más profundas) sin apenas establecer diferencias entre las distintas experiencias. La mayor parte de las experiencias que llegaban a una hoja seguirán llegando a una de las nuevas hojas (hija de la hoja que acaba de convertirse en nodo de decisión).

#### Presentando la solución propuesta

Una opción que se puede considerar para evitar la situación que acabamos de describir es exigir un tamaño mínimo para cada uno de los subconjuntos de experiencias que se generarían al realizar una expansión (cada subconjunto se correspondería con una de las nuevas ramas creadas). Esta estrategia entraría en el ámbito de los métodos de "*prepoda*". Existen multitud de métodos para realizar la poda de un árbol de decisión [Min-1989; Esp-1997; Her-2004], pero debido a que estamos en el campo del aprendizaje incremental, debemos fijarnos en métodos que estén basados en la prepoda. Un estudio comparando los enfoques de "*prepoda*" y "*postpoda*" puede encontrarse en la tesis doctoral de Frank [Fra-2000] y una de las conclusiones a la que se llega es que ambos enfoques ofrecen resultados similares cuando se usan para problemas reales, pero el coste computacional de la prepoda es mucho menor, puesto que la postpoda implica la expansión de árboles más complejos que posteriormente son simplificados. Por estas razones es preferible usar métodos de prepoda cuando estamos ante aprendizaje incremental.

Antes de decidir que una hoja sea expansible (*Condición_Es_Expansible(i)*) hay que comprobar que la rama que más experiencias concentraría si se expandiese, no superará el valor fijado por un umbral máximo ($max\_prop\_mayor$). No usamos un umbral mínimo porque, aunque para el caso de la división dicotómica estaríamos en un caso simétrico, en el caso de la expansión de un atributo categórico con más de dos valores lo interesante es que ninguna rama concentre la mayor parte de las experiencias.

#### Formalizando la solución propuesta

Para definir este nuevo requisito, previo a la expansión de una hoja, sólo es preciso definir una variable y un parámetro interno. La variable será $prop\_mayor$ y registrará la mayor proporción de experiencias que corresponden con un valor del atributo que se quiere expandir. Para su cálculo se usarán componentes

previamente definidos. El parámetro interno será llamado $max\_prop\_mayor$ y representará la máxima proporción de experiencias que se permitirá que se acumulen para un único valor.

La definición de la variable $prop\_mayor$ variará según se esté intentando expandir una hoja por un atributo categórico o continuo:

- expandiendo la hoja $i$ por el atributo categórico $l$:
  $prop\_mayor = \text{máx}\{r'_{(i,l,v)}/rama_i \mid v \in m_l\}$
- expandiendo la hoja $i$ por el atributo continuo $l'$ usando el punto de corte $pc$:
  $prop\_mayor = \text{máx}\{total_{\leq pc}/rama_i\ ,\ total_{>pc}/rama_i\}$

Habiendo definido $prop\_mayor$ sólo resta definir cómo se modifica la condición que controla la expansibilidad de una hoja:

$$Condición\_Es\_Expansible : HOJAS \cup \{0\} \rightarrow \{V, F\}$$
$$Condición\_Es\_Expansible(i) \equiv (i \neq 0)\ \wedge\ (prop\_mayor \leq max\_prop\_mayor) \wedge Diferenciado\_Mejor\_atrib(i)$$

**Estudio de la mejora y su complejidad**

La nueva condición que se ha añadido al predicado que evalúa si una hoja debe ser expandida ofrece mejoras, pero éstas son más o menos evidentes dependiendo del conjunto de datos del que se quiera extraer conocimiento y, principalmente, de si el problema está definido por atributos continuos o no. Para exponer los resultados derivados del estudio del parámetro interno $max\_prop\_mayor$ no hemos usado el mismo conjunto de datos con ruido que venimos usando, sino que hemos seleccionado un conjunto de datos con atributos continuos, que es donde mejor se aprecian los beneficios. El conjunto de datos en este caso concreto también ha sido elegido del repositorio de la UCI [New-1998]. Hemos seleccionado el generador de conjuntos de datos llamado "*Waveform*" que tiene 40 atributos continuos (19 de ellos irrelevantes), 3 valores de clase y no es completamente determinista, es decir, presenta ruido (en la sección 4.5 pueden encontrarse más detalles). En la figura 3.13 presentamos el comportamiento del algoritmo en función del parámetro $max\_prop\_mayor$ y observamos cómo repercute en la precisión, en el tamaño del árbol y en el número total de experiencias muestreadas.

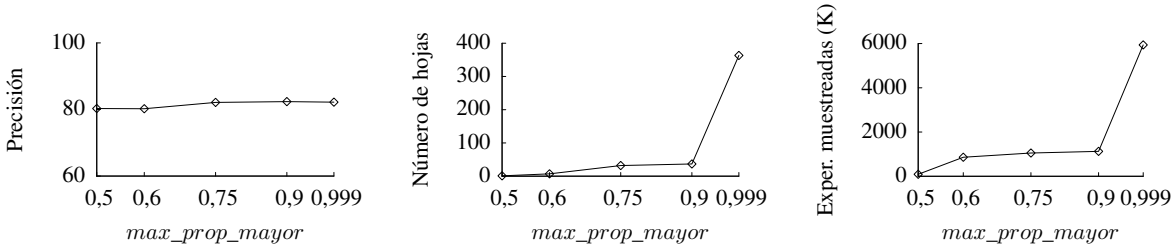



Figura 3.13: Comportamiento dependiendo del parámetro $max\_prop\_mayor$.

Considerando las gráficas presentes en la figura 3.13 podemos alcanzar las siguientes conclusiones:

- La precisión se mantiene bastante estable pero, para los valores de $max\_prop\_mayor$ menores que $0,75$, la precisión es levemente inferior. La razón de esa leve pérdida de precisión es que valores demasiado pequeños (entre $0,5$ ó $0,6$) provocan exigencias demasiado estrictas para las expansiones, haciendo que su número se reduzca tanto que no se alcancen los máximos niveles de precisión.
- Donde mejor se aprecia la restricción impuesta al desbalanceo máximo permitido para expandir es en el tamaño del árbol. Para valores pequeños de $max\_prop\_mayor$ (entre $0,5$ ó $0,6$), las expansiones se ven tan limitadas que el árbol es demasiado pequeño. Para valores entre $0,7$ y $0,9$ el comportamiento es mucho más estable, la precisión alcanzada es la máxima pero induciendo árboles de tamaño

bastante reducido. Si usamos valores cercanos a 1, lo que supondría permitir total libertad a las expansiones, puede suceder que se produzcan expansiones que hagan crecer el árbol sin aumentar la precisión, como puede observarse en la gráfica para el valor $0,999$.

- El mismo comportamiento observado para el tamaño del árbol se aprecia en cuanto al número total de experiencias muestreadas y al tiempo consumido. El hecho de permitir mayor libertad en las expansiones (valores cercanos a 1) conlleva realizar más expansiones y éstas a su vez exigen muestrear más experiencias.

Teniendo en cuenta el comportamiento observado para el parámetro $max\_prop\_mayor$, del que hemos mostrado un ejemplo en la figura 3.13, hemos seleccionado el valor de $0,75$ como valor por defecto para este parámetro. Con esto intentamos no acercarnos a ninguno de los extremos: ni permitir cualquier tipo de expansión, ni ser demasiado exigentes.

La modificación del predicado *Condición_Es_Expansible* no supone ningún coste adicional desde el punto de vista de la complejidad. En lo que se refiere a la complejidad temporal, la única diferencia es que supone realizar un número pequeño y constante de operaciones que no repercutirán en la complejidad del algoritmo. En cuanto a la complejidad espacial tampoco se verá afectada porque la información que se almacena no se ha incrementado.

### 3.3.3. Reformulando la idoneidad del mejor atributo para expandir

En la sección 3.2 introdujimos algunos problemas que pueden surgir de la forma en que se identifica la idoneidad del mejor atributo para expandir una hoja. El predicado $Diferenciado\_Mejor\_atrib(i)$, que se usa para determinar el predicado *Condición_Es_Expansible(i)*, tiene como objetivo indicar cuándo el mejor atributo para expandir la hoja $i$ ($mejor\_atributo_i$) es claramente mejor que el resto de atributos. Recordemos cómo se definía el mejor atributo ($mejor\_atributo_i$) y el predicado $Diferenciado\_Mejor\_atrib(i)$:

$$\begin{aligned} mejor\_atributo_i = \text{mín} \quad \{ \quad & l \in MEJORES\_ATRIBUTOS_i \mid \\ & \text{ínf}(medida_i(l)) \leq \text{ínf}(medida_i(l')) \\ & \forall l' \in MEJORES\_ATRIBUTOS_i\} \qquad \text{donde} \\ MEJORES\_ATRIBUTOS_i = \quad \{ \quad & l \in atributos\_libres_i \mid \\ & \sup(medida_i(l)) \leq \sup(medida_i(l')) \\ & \forall l' \in atributos\_libres_i\} \end{aligned}$$

$$\begin{aligned} & Diferenciado\_Mejor\_atrib(i) : HOJAS \rightarrow \{V, F\} \\ & Diferenciado\_Mejor\_atrib(i) \equiv \Big( \Big[ \sup(medida_i(mejor\_atributo_i)) \leq \text{ínf}(medida_i(l)) \\ & \qquad \forall l \in (atributos\_libres_i - \{mejor\_atributo_i\}) \Big] \\ & \qquad \vee \\ & \qquad \Big[ \mid \sup(medida_i(mejor\_atributo_i)) - \text{mín}\{\text{ínf}(medida_i(l)) : \\ & \qquad\quad l \in (atributos\_libres_i - \{mejor\_atributo_i\})\}\mid \leq d \Big] \Big) \end{aligned}$$

En la primera parte del predicado lo que se pedía era que el mejor atributo se diferenciase totalmente del resto. Como existía la posibilidad de que esa diferenciación tan drástica se demorase, o incluso no llegase a darse, se incluyó un parámetro interno ($d$) para flexibilizar la condición. Lo que se hizo fue permitir un nivel máximo de solapamiento entre el mejor atributo y el resto de atributos. En la figura 3.14 hemos representado dos gráficas con estos dos casos. En los ejemplos que presentamos hemos supuesto (sin pérdida de generalidad) que tenemos 4 atributos que pueden ser usados para la expansión de la hoja $i$ y que hemos nombrado con números consecutivos. Además los hemos ordenado según su calidad, usando los índices de forma creciente para representar el empeoramiento ($X_1$ es el mejor atributo y $X_4$ es el peor).

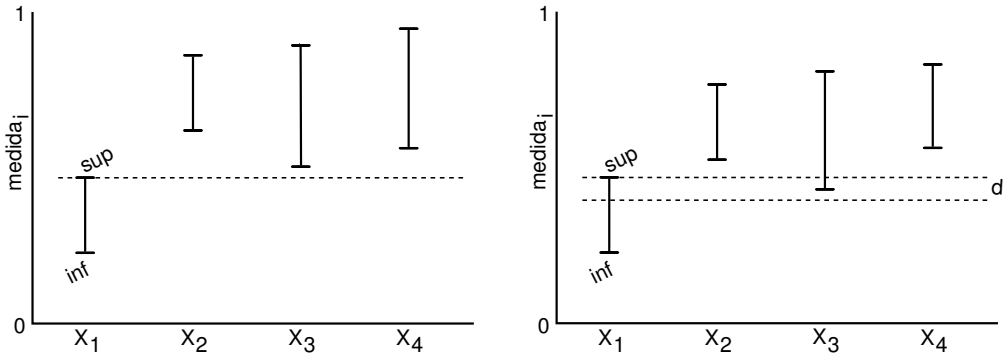



Figura 3.14: Dos casos en los que el predicado $Diferenciación\_Mejor\_atrib(i)$ sería cierto. En el caso de la izquierda por clara separación entre los atributos. En el caso de la derecha por un pequeño solapamiento, menor que $d$.

Si analizamos detenidamente el procedimiento que se sigue para determinar la idoneidad del mejor atributo nos encontramos con dos problemas (ambos representados en la figura 3.15):

- la idoneidad puede ser detectada con excesivo retraso: con que exista un sólo atributo que tenga el mismo comportamiento que el mejor atributo, éste nunca se diferenciará del resto puesto que al menos habrá uno que sea igual que él. La única opción que permitiría que el predicado de diferenciación se satisficiera sería que el tamaño del intervalo fuese menor que $d$. En el ejemplo de la figura 3.15 sería necesario que $\sup(medida_i(X_1)) - \inf(medida_i(X_2)) \leq d$, cosa que podría alcanzarse después de muestrear un número elevado de experiencias. Y aún así, el predicado estaría dando por cierto que el mejor atributo se ha diferenciado del resto, existiendo otro exactamente igual a él y del que no se ha diferenciado en la realidad.
- la idoneidad puede ser detectada con excesivo adelanto: al ser usado el valor de $d$ como valor absoluto, podremos encontrarnos en situaciones en las que los atributos en realidad no hayan llegado a diferenciarse (como en el caso de la derecha de la figura 3.15), pero que el predicado considere como válidas para indicar que el mejor atributo se ha diferenciado.

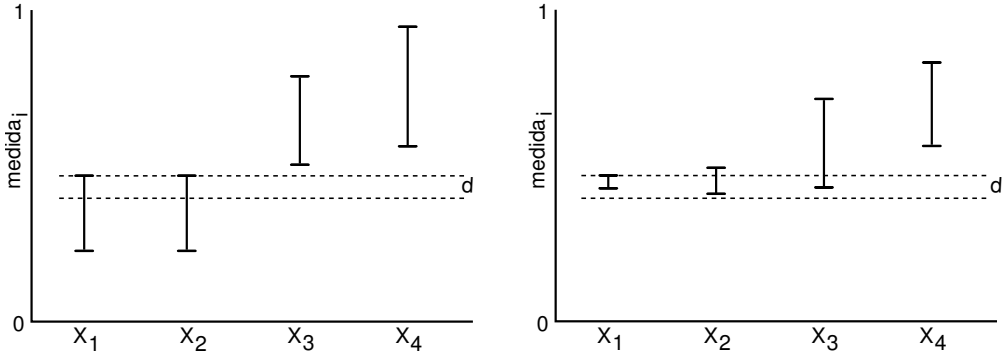



Figura 3.15: Caso en el que el predicado $Diferenciación\_Mejor\_atrib(i)$ podría ser siempre falso (izquierda) o en el que podría ser prematuramente cierto (derecha).

**Presentado la solución propuesta**

Es preciso resolver los problemas que acabamos de comentar porque, aunque es posible que al estudiar determinados conjuntos de datos no aparezcan, siempre serán susceptibles de ocurrir.

El problema de elegir el mejor atributo cuando hay dos que son parecidos ya se ha planteado en otros algoritmos de aprendizaje incremental como el VFDT [Dom-2000]. En este caso, los autores propusieron establecer una diferencia mínima entre las medidas de desorden de los dos mejores atributos (en su caso, los dos que tenían mayor ganancia de información). Con esta diferencia mínima (que llamaron $\tau$) conseguían que el modelo que se estaba induciendo no se estancase de forma indefinida, sino que presentase una velocidad mínima de crecimiento. Si no se producen expansiones es porque la diferencia entre los atributos es pequeña, pero, con el paso de las experiencias, llegará un momento en que la diferencia sea menor que el umbral establecido ($\tau$) y se expanda.

Este enfoque tiene sus defectos como presentaron Holmes, Kirby y Pfahringer [Hol-2005]. El problema de esta estrategia para romper el empate es que, bajo determinadas situaciones, se produce un excesivo número de expansiones sin la suficiente calidad, dando lugar a árboles demasiado complejos que no aportan nada desde el punto de vista de la precisión. Para resolver este inconveniente propusieron una modificación en la que se usaban los atributos mejor, segundo mejor y peor. Lo que estudiaban era si la diferencia entre los dos mejores atributos multiplicado por un factor (que llamaban $\alpha$) era menor que la diferencia entre el segundo mejor y el peor. Para evitar el mismo problema que tenía el método original de VFDT y el número de expansiones no fuese excesivo, proponían aumentar el límite de experiencias necesarias entre dos desempates consecutivos.

Pero con esta aportación también existen algunas situaciones que pueden provocar un mal funcionamiento del algoritmo. Una de ellas es que nunca se rompería un empate si todos los atributos presentasen un comportamiento similar, puesto que las diferencias entre el mejor y el segundo mejor y entre el segundo mejor y el peor no se diferenciarían lo suficiente. Y aunque el incremento del límite de experiencias necesarias entre dos desempates consecutivos sirve para paliar el problema, no consigue eliminar los desempates innecesarios, sino retrasarlos.

Viendo los problemas de los métodos anteriores y teniendo en cuenta que el algoritmo IADEM-0 no tiene valores concretos para las medidas de desorden, sino que estima los intervalos donde deben situarse, hemos decidido adoptar una estrategia diferente. La solución que formalizaremos a continuación está basada en la comparación del mejor atributo con el peor atributo. La determinación estricta del mejor atributo no es imprescindible para alcanzar árboles de calidad siempre que el mejor atributo sea realmente bueno. Al comparar los dos atributos buscaremos la ocurrencia de cualquiera de las siguientes dos condiciones:

- que sean muy diferentes: el peor atributo puede ser irrelevante en el peor caso o contener información útil en el mejor caso. De cualquier modo, el mejor atributo, al diferenciarse de él va a disponer de información significativa. Si el peor atributo no era relevante, el mejor atributo tendrá información que se diferencia de la que no es relevante. Si el peor atributo era relevante, aún mejor, porque el mejor atributo tendrá información más significativa que la de otro que ya era útil.

- que sean muy parecidos: al ser muy parecidos el mejor y el peor atributo estamos garantizando que el resto de atributos también serán parecidos al mejor. Por lo tanto, todos los atributos tendrán una información similar y, de entre la información que nos ofrecen, podríamos quedarnos con cualquiera. Para quedarnos con el mejor caso, usaremos el mejor atributo.

En el método que a continuación presentamos, seguiremos usando el parámetro interno $d$, pero su uso variará. Ahora será usado de forma relativa para estudiar la diferencia o parecido entre las medidas de desorden de dos atributos. La idea básica que hay detrás de la técnica que proponemos es estudiar que porcentaje de los intervalos del mejor atributo y del peor atributo son comunes. Para decir que son parecidos exigiremos que ese porcentaje común sea elevado y para decir que son diferentes exigiremos que sea pequeño.

**Formalizando la solución propuesta**

Para describir formalmente el método que proponemos sólo es necesario definir cómo se calcula el peor atributo, qué amplitud tiene un intervalo, cómo se calcula el porcentaje que tiene en común el mejor atributo y el peor, y cómo se definen los predicados necesarios para evaluar la idoneidad del mejor atributo. De este modo:

- El peor atributo para la expansión de la hoja $i$ se define de forma complementaria a como se hizo con el mejor atributo. Sea $PEORES\_ATRIBUTOS_i$ el conjunto de los atributos menos prometedores. Éstos serán aquellos atributos que tengan la mayor medida de desorden en el peor caso ($\sup(medida_i(l))$):

$$PEORES\_ATRIBUTOS_i = \{ \; l \in atributos\_libres_i \mid \sup(medida_i(l)) \geq \sup(medida_i(l')) \; \forall l' \in atributos\_libres_i \}$$

  El peor atributo se selecciona de entre los anteriores y será aquél cuya medida de desorden sea mayor en el mejor caso ($\text{ínf}(medida_i(l))$):

$$peor\_atributo_i = \text{mín} \; \{ \; l \in PEORES\_ATRIBUTOS_i \mid \text{ínf}(medida_i(l)) \geq \text{ínf}(medida_i(l')) \; \forall l' \in PEORES\_ATRIBUTOS_i \}$$

- La amplitud del intervalo donde puede encontrarse el valor de la medida de desorden de un atributo es la diferencia entre su valor para el peor caso y para el mejor caso y la notaremos como $||atributo||$. Así:

$$\begin{aligned} ||mejor\_atributo_i|| &= \sup(medida_i(mejor\_atributo_i)) - \text{ínf}(medida_i(mejor\_atributo_i)) \\ ||peor\_atributo_i|| &= \sup(medida_i(peor\_atributo_i)) - \text{ínf}(medida_i(peor\_atributo_i)) \end{aligned}$$

- El intervalo que tienen en común se calculará con la función $común_i$ que toma los índices de dos atributos libres de la hoja $i$ y que se define como:

$$\begin{aligned} &común_i : \mathbb{N} \times \mathbb{N} \to [0,1] \\ &común_i(a,b) = \text{máx}\{0, (\text{máx}\{\sup(i,a), \sup(i,b)\} - \text{mín}\{\text{ínf}(i,a), \text{ínf}(i,b)\}) - \\ &\qquad ((\text{máx}\{\sup(i,a), \sup(i,b)\} - \text{mín}\{\sup(i,a), \sup(i,b)\}) + \\ &\qquad (\text{máx}\{\text{ínf}(i,a), \text{ínf}(i,b)\} - \text{mín}\{\text{ínf}(i,a), \text{ínf}(i,b)\})) \end{aligned}$$

  donde, por simplicidad, hemos usado la siguiente nomenclatura:

$$\begin{aligned} &\sup(i,a) = \sup(medida_i(a)) \\ &\sup(i,b) = \sup(medida_i(b)) \\ &\text{ínf}(i,a) = \text{ínf}(medida_i(a)) \\ &\text{ínf}(i,b) = \text{ínf}(medida_i(b)) \end{aligned}$$

  Por comodidad y para unificar la notación con la amplitud de un intervalo, usaremos la siguiente notación para referirnos al intervalo de las medidas de desorden que tienen en común dos atributos:

$$||a \cap^i b|| = común_i(a,b)$$

  Por tanto, el porcentaje en común del mejor y peor atributo se define como:

$$||mejor\_atributo_i \cap^i peor\_atributo_i|| = común_i(mejor\_atributo_i, peor\_atributo_i)$$

- El predicado que evalúa si dos atributos ($a$ y $b$) en la hoja $i$ son diferentes se define del siguiente modo (recordemos que $d$ es el parámetro interno que ha sido redefinido para controlar la diferenciación o el parecido):

$$Diferentes_i : \mathbb{N} \times \mathbb{N} \times [0,1] \to \{V, F\}$$

$$Diferentes_i(a,b,d) \equiv \left[ \left( \frac{||a \cap^i b||}{||a||} \right) \leq d \right] \vee \left[ \left( \frac{||a \cap^i b||}{||b||} \right) \leq d \right]$$

- El predicado que evalúa si dos atributos ($a$ y $b$) en la hoja $i$ son parecidos se define del siguiente modo (recordemos que $d$ es el parámetro interno que ha sido redefinido para controlar la diferenciación o el parecido):

$$Parecidos_i : \mathbb{N} \times \mathbb{N} \times [0,1] \rightarrow \{V, F\}$$

$$Parecidos_i(a,b,d) \equiv \left[\left(\frac{||a \cap^i b||}{||a||}\right) \geq (1-d)\right] \ \wedge \left[\left(\frac{||a \cap^i b||}{||b||}\right) \geq (1-d)\right]$$

Una vez hemos descrito todos los elementos necesarios para formalizar la nueva condición que servirá para detectar la idoneidad de un atributo para realizar una expansión, podemos formalizar su definición. Al criterio de ser parecidos hemos añadido una amplitud máxima que no puede superar el intervalo del mejor atributo. La razón es que, si el intervalo es demasiado grande (circunstancia que suele darse cuando aún no se han muestreado suficientes experiencias y las cotas de concentración son demasiado amplias), los atributos aún no han terminado de definirse con suficiente información. De hecho, cuando aún no se han muestreado suficientes experiencias, todos los atributos tendrán intervalos tan grandes que ocuparán gran parte del intervalo entre 0 y 1 y serán muy parecidos. Expandir en esa circunstancia sería muy contraproducente para el resultado que se espera alcanzar. Considerando esta condición adicional, definimos el predicado como sigue:

$$\begin{aligned} &Diferenciado\_Mejor\_atrib(i) : HOJAS \rightarrow \{V, F\} \\ &Diferenciado\_Mejor\_atrib(i) \equiv Diferentes(mejor\_atributo_i, peor\_atributo_i) \ \vee \\ &\qquad \Big(Parecidos(mejor\_atributo_i, peor\_atributo_i) \ \wedge \\ &\qquad\ ||mejor\_atributo_i|| \leq d\Big) \end{aligned}$$

En la figura 3.16 hemos representado dos casos en los que el predicado $Diferenciado\_Mejor\_atrib(i)$ sería cierto. En la gráfica a la izquierda de la figura hemos representado un caso en el que el mejor atributo tiene un intervalo para la medida de desorden que es muy parecido al intervalo del mejor atributo. Además ese intervalo es pequeño (menor que $d$), por lo que el predicado diría que el mejor atributo sería idóneo para realizar una expansión. En la gráfica de la derecha de esa misma figura hemos representado un caso en el que los intervalos de ambos atributos son diferentes puesto que el mejor de ellos sólo tiene en común un $5\,\%$ de su intervalo con el peor.

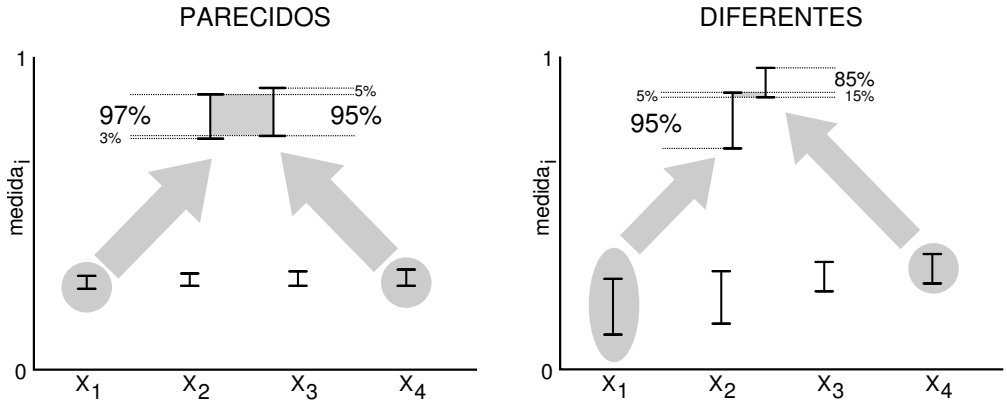



Figura 3.16: Dos casos en los que el predicado $Diferenciación\_Mejor\_atrib(i)$ sería cierto. A la izquierda porque los atributos mejor y peor son muy parecidos y a la derecha porque son muy diferentes. Recordemos que el valor por defecto para el parámetro interno $d$ es $0,05$.

**Estudio de la mejora y su complejidad**

Considerando que el uso del parámetro interno $d$ ha pasado de ser absoluto a ser relativo y que la forma de usarlo también se ha modificado, es preciso volver a realizar un estudio sobre dicho parámetro. En la figura 3.17 hemos representado el comportamiento del algoritmo en función de $d$ teniendo en cuenta la precisión, el tamaño del árbol de decisión inducido y el número de hojas muestreadas.

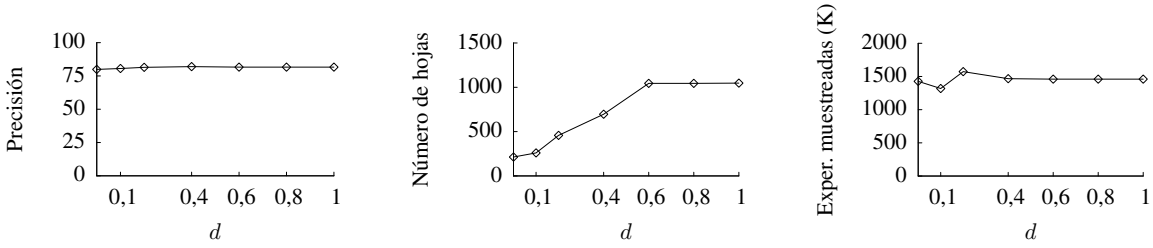



Figura 3.17: Comportamiento dependiendo del parámetro $d$.

Las observaciones que se pueden hacer sobre las gráficas en la figura 3.17 pueden resumirse del siguiente modo:

- La precisión alcanzada se mantiene prácticamente constante. Tan solo se aprecia un pequeño descenso en la precisión cuando $d$ toma el valor $0$, lo que se correspondería con la exigencia de una diferenciación estricta del mejor atributo en la que no se permitiría ninguna clase de solapamiento entre el mejor atributo y el peor. Pero, aún a pesar de esa pequeña diferencia, la precisión se mantiene muy estable.

- El tamaño del árbol aumenta conforme mayor es el valor de $d$. La razón se debe a que un mayor valor de $d$ lleva acompañado una mayor permisividad a la hora de hacer expansiones, lo que repercute en una menor calidad de éstas. Dado que las expansiones que se realizan son de peor calidad, el número de éstas tendrá que incrementarse para extraer el modelo más preciso posible. La elección de atributos de poca calidad lleva a que posteriormente tengan que ser corregidos por sucesivas expansiones. Llegado a un punto en la relajación de la condición para considerar que un atributo es idóneo para la expansión, los resultados se mantienen constantes, como se puede observar para valores de $d$ mayores que $0,6$.

- El número total de experiencias muestreadas permanece también bastante estable y las elecciones de atributos que fueron realizadas con poca información debido a la relajación del criterio, no afectan al número de experiencias que es necesario muestrear antes de detener la inducción. Debido a la relación que existe entre el número de experiencias muestreadas y el tiempo consumido, éste también se mantiene bastante estable independientemente del valor de $d$.

Teniendo en cuenta que la precisión apenas varía (excepto ligeramente para $d = 0$) y que el tamaño del árbol crece con el valor de $d$, el valor por defecto que hemos seleccionado para el parámetro $d$ es $0,1$, que coincide con el mismo valor que habíamos seleccionado cuando se usaba de forma absoluta.

La incorporación de este método para la verificar la idoneidad del mejor atributo sustituyendo al anterior no tiene repercusiones en la complejidad. En cuanto a la complejidad temporal lo único que supone es una variación en la evaluación del predicado $Diferenciación\_Mejor\_atrib(i)$ que no supone aumento en la complejidad. Si acaso, cierto decremento, puesto que ahora, en vez de comparar el mejor atributo con el resto de atributos, sólo se compara con el peor atributo; pero la diferencia no supone un descenso en el orden de complejidad temporal que se mantiene igual. En lo que se refiere a la complejidad espacial, sólo se han incluido algunas variables que únicamente serán empleadas para la evaluación del predicado y que no suponen tampoco ningún incremento en la complejidad espacial del algoritmo.

## 3.4. Mejorar la predicción: IADEM-2

En la subsección 2.2.6 presentamos el algoritmo IADEM-0 como un sistema de predicción en el que se usaba la clase mayoritaria como criterio para realizar dicha predicción. La idea de clasificar una observación ($o \in U_O$) usando la clase mayoritaria en la hoja a la que llegaría tal observación está bastante extendida y es muy común en la mayor parte de los árboles de decisión. Pero un enfoque que surgió hace algún tiempo, que se sigue aplicando y sobre el que se continúa trabajando en la actualidad consiste en realizar la predicción usando otro tipo de funciones que, considerando más información, intentan dar una predicción más ajustada. Cuando nos encontramos trabajando con árboles de decisión, la información de cada hoja sirve para definir la función que se usará en dicha hoja con objeto de realizar la predicción y, por tanto, las hojas son llamadas "*hojas funcionales*" [Gam-2001].

El árbol de decisión que induce el algoritmo IADEM-0 almacena gran cantidad de información en las hojas virtuales que es utilizada en la fase de expansión del árbol, pero no se usa para realizar las predicciones. En esta sección veremos cómo se puede aprovechar toda la información de que dispone el árbol para ofrecer predicciones de mayor calidad.

**Presentando la solución propuesta**

De entre los primeros trabajos que sustituyeron el esquema habitual de realizar la predicción usando la clase mayoritaria y emplearon otros más avanzado podemos destacar los "*árboles perceptron*" ("*perceptron trees*") [Utg-1989], que incluyen discriminadores lineales (perceptrones) en los nodos del árbol, o los "*árboles con clasificación ingenua de Bayes*" ("*NBTrees*") [Koh-1996], que incluyen clasificadores ingenuos basados en la regla de Bayes (naïve Bayes). Incluir estos clasificadores en las hojas permite realizar predicciones con más información y, por el hecho de combinar dos clasificadores (un árbol de decisión para la estructura y otro clasificador en las hojas del árbol), estos árboles de decisión también se llaman "*árboles de decisión híbridos*". Un trabajo que estudia los beneficios de realizar las predicciones con más información es el presentado por Seewald, Petrack y Widmer [See-2001], en el que estudian la mejora en la predicción usando diferentes tipos de clasificadores en las hojas (modelos basados en casos, naïve Bayes y regresión lineal). Otro trabajo, algo más general, es el presentado por Gama [Gam-2001] donde se estudia qué beneficio aporta usar funciones en cualquier nodo del árbol sin restringirlo sólo a las hojas y se muestra cómo el hecho de usar hojas funcionales supone una reducción en la varianza de los modelos.

Atendiendo a los beneficios que se derivan de incluir clasificadores en las hojas de los árboles de decisión, hemos decidido incorporar y adaptar este enfoque al nuevo algoritmo que desarrollamos y que mejorará al anterior IADEM-0. En nuestro caso, el modelo más conveniente para intentar realizar predicciones de más calidad es el que usa clasificadores ingenuos basados en la regla de Bayes (naïve Bayes)[Dud-1973; Lan-1992; Dud-2001], puesto que en las hojas virtuales ya se dispone de la información que se necesita para inducir este clasificador y, además, se ajusta muy bien al aprendizaje incremental. Por esta última razón, en el algoritmo VFDTc [Gam-2003; Gam-2006] también usan esta técnica para la predicción.

Una observación que se desea etiquetar alcanzará una de las hojas del árbol de decisión considerando las decisiones que etiquetan los nodos internos del mismo. Cuando un modelo únicamente emplea la frecuencia relativa de las clases en la hoja para realizar una predicción sobre la clase que debería etiquetar dicha observación, está dando una aproximación demasiado general, pues está considerando que todas las observaciones que alcancen la hoja serán similares. Pero, de esa observación que ha llegado a una hoja concreta, posiblemente, habrá varios atributos que no hayan sido utilizados en el camino hasta la hoja y que pueden utilizarse para realizar una predicción más informada. Es bajo estas condiciones cuando se usa el teorema de Bayes para calcular las probabilidades de alcanzar las distintas clases pero condicionadas por los valores de los atributos que aún no han sido usados en la observación concreta que se está estudiando. De esta forma se pretende realizar una predicción "*personalizada*" para cada observación. Gracias a esta "personalización" se consigue una mejora global del modelo en la tarea de predicción.

Este enfoque, además de realizar predicciones con más información, es muy apropiado para ser utilizado en el ámbito del aprendizaje incremental debido a diversas razones entre las que podemos destacar: que tiene

naturaleza incremental, ya que lo único que necesita para ser calculada son los contadores que se actualizan incrementalmente; que se comporta bien en problemas con datos heterogéneos y que se ha revelado como un algoritmo muy eficiente para conjuntos de datos pequeños [Dom-1997]. En este punto debemos señalar que el conjunto de datos que trata el algoritmo de inducción de árboles de decisión no es el mismo conjunto de datos que el que aborda el clasificador basado en Bayes. Para empezar, la información en cada una de las hojas donde se calculará este clasificador únicamente registra un subconjunto del problema y, mientras que el árbol de decisión trata con problemas potencialmente grandes, el clasificador basado en Bayes sólo tiene que calcularse para un problema que dependerá del número de valores para un atributo y del número de clases (problema de tamaño reducido y siempre menor que el original).

**Formalizando la solución propuesta**

A continuación indicamos qué elementos de los que ya se encuentran definidos son los que necesitaremos para calcular las probabilidades condicionadas y cómo son usadas para aplicar el teorema de Bayes.

Aplicando el teorema de Bayes lo que pretendemos es calcular una expresión que sea proporcional a la probabilidad de que una observación esté etiquetada con una determinada clase, dada la propia observación. Cuando una observación alcanza una hoja, habrá una serie de atributos que hayan sido usados, pero otros no ($atributos\_libres_i$). La información de que se dispone en cada hoja virtual es información sobre los atributos categóricos que aún no han sido usados o sobre los atributos continuos que siempre pueden ser usados.

Calcular la probabilidad condicionada de una clase dada un atributo categórico es algo directo, pero hacerlo para un atributo numérico es algo más complejo. Para este último caso, hemos usado la misma aproximación planteada por Domingos y Pazzani [Dom-1997] debido a su sencillez y a que las aproximaciones que realiza son de bastante calidad. Esta aproximación consiste en dividir el intervalo de cada atributo continuo en 10 intervalos de igual anchura (o en menos si existen menos valores observados para dicho atributo).

La probabilidad de que una observación $o$ sea etiquetada con la clase $z$, notado como $P(z \mid o)$, es, aplicando el teorema de Bayes, proporcional a la siguiente expresión:

$$P(z \mid o) \propto P(z) \cdot \prod P(x_l(o) \mid z)$$

donde

- $z$ es cualquiera de los valores de clase
- $l$ es cualquiera de los atributos libres en la hoja a la que llega la observación $o$
- $x_l(o)$ es el valor que toma el atributo libre $x_l$ en la observación $o$
- $P(z)$ es la probabilidad de la clase $z$ en la hoja a la que llega la observación $o$ y se calcula usando la siguiente expresión (ya usada en la subsección 2.2.6):

$$P(z) = \frac{r_{i,z}}{r_i}$$

- $P(x_l(o) \mid z)$ es la probabilidad de que, dada la clase $z$, el atributo libre $x_l$ tome el valor que indica la observación $o$ y se calcula usando la siguiente expresión que aprovecha la información que ya estaba presente en las hojas virtuales:

$$P(x_l(o) \mid z) = \frac{r'_{(i,l,x_l(o)),z}}{rama_{i,z}}$$

Debemos indicar que $x_l(o)$ tendrá una interpretación diferente según si estamos tratando con un atributo categórico o continuo:

- categórico: $x_l(o)$ tomará valores en su dominio, es decir, $x_l(o) \in X_l = \{1, \ldots, m_l\}$

- continuo: $x_l(o)$ tomará su valor dependiendo del número de intervalos en los que sea dividido el atributo $x_l$ en la hoja a la que se llega mediante la observación $o$ y del valor que tenga ese atributo en la observación. El número de intervalos de igual anchura que surgen de dividir el atributo $x_l$ en la hoja correspondiente se define como $\text{mín}\{diferentes_{x_l}, 10\}$.

Recordemos que en la subsección 2.2.6 se introdujo el árbol de decisión inducido por IADEM-0 como un sistema de predicción ${}_{HOJAS}P$. Ahora debemos hacer lo mismo pero considerando el nuevo criterio para realizar las predicciones. Así, definimos ${}_{HOJAS}P\_NB$ como el nuevo sistema de predicción que usa el teorema de Bayes:

${}_{HOJAS}P\_NB : U_O \to U_P$

${}_{HOJAS}P\_NB(o) = ({}_{HOJAS}P\_NB_1(o), {}_{HOJAS}P\_NB_2(o), \ldots, {}_{HOJAS}P\_NB_k(o))$ donde:

$$ {}_{HOJAS}P\_NB_z(o) = \begin{cases} \dfrac{P(z \mid o)}{\sum_{j=1}^{k} P(j \mid o)} & \text{si } r_i > 0 \\ \dfrac{\sum\limits_{j \in MISMO\_PADRE(i)} r_{j,z}}{\sum\limits_{j \in MISMO\_PADRE(i)} r_j} & \text{si } r_i = 0 \\ & \text{donde } x_u(b) = V_{i,u}\ \forall u \in atributos\_usados_i \end{cases} $$

donde $MISMO\_PADRE(i) = \{l \in HOJAS \mid V_{l,A_l(j)} = V_{i,A_i(j)} \ \wedge \ j \in \{1, \ldots, N_i - 1\}\} - \{i\}$

En caso de que no se disponga de información en la hoja, el criterio es el mismo que el que se usaba anteriormente: usar la información de la frecuencia en los padres.

Al igual que se definió un sistema que ofrecía márgenes de error para el sistema de predicción previo (${}_{HOJAS}M$), ahora se puede definir el mismo sistema pero considerando el nuevo enfoque. Lo llamaremos ${}_{HOJAS}M\_NB$ y se define del siguiente modo:

${}_{HOJAS}M\_NB : U_O \to \left([0,1]^2\right)^k$

${}_{HOJAS}M\_NB(o) = ({}_{HOJAS}M\_NB_1(o), {}_{HOJAS}M\_NB_2(o), \ldots, {}_{HOJAS}M\_NB_k(o))$ donde:

$$ {}_{HOJAS}M\_NB_z(o) = \begin{cases} (\text{ínf}(P(z \mid o)), \sup(P(z \mid o))) & \text{si } r_i > 0 \\ (\text{máx}\{0, {}_{HOJAS}P_z(o) - estError(o)\}, & \text{si } r_i = 0 \\ \ \text{mín}\{1, {}_{HOJAS}P_z(o) + estError(o)\}) & \text{donde } x_u(o) = V_{i,u} \\ & \forall u \in atributos\_usados_i \end{cases} $$

donde:

$$ \text{ínf}(P(z \mid o)) = \frac{\text{ínf}'(P(z \mid o)) \cdot \prod \text{ínf}'(P(x_l(o) \mid z))}{\sum_{j=1}^{k} P(j \mid o)} $$

$$ \text{ínf}'(P(z \mid o)) = \begin{cases} \text{máx}\{0, P(z \mid o) - margen\_\varepsilon(r_i, P(z \mid o), \delta)\} & \text{si } r_i \neq 0 \\ 0 & \text{si } r_i = 0 \end{cases} $$

$$ \text{ínf}'(P(x_l(o) \mid z)) = \begin{cases} \text{máx}\{0, P(x_l(o) \mid z) - margen\_\varepsilon(rama_{i,z}, P(x_l(o) \mid z), \delta)\} & \text{si } rama_{i,z} \neq 0 \\ 0 & \text{si } rama_{i,z} = 0 \end{cases} $$

$$\sup(P(z\,|\,o)) = \frac{\sup'(P(z\,|\,o))\cdot\prod \sup'(P(x_l(o)\,|\,z))}{\sum_{j=1}^{k} P(j\,|\,o)}$$

$$\sup'(P(z\,|\,o)) = \begin{cases} \min\{1, P(z\,|\,o) + margen\_\varepsilon(r_i, P(z\,|\,o), \delta)\} & \text{si } r_i \neq 0 \\ 1 & \text{si } r_i = 0 \end{cases}$$

$$\sup'(P(x_l(o)\,|\,z)) = \begin{cases} \min\{1, P(x_l(o)\,|\,z) + margen\_\varepsilon(rama_{i,z}, P(x_l(o)\,|\,z), \delta)\} & \text{si } rama_{i,z} \neq 0 \\ 1 & \text{si } rama_{i,z} = 0 \end{cases}$$

$$estError(o) = margen\_\varepsilon\Big(\sum_{j\in MISMO\_PADRE(i)} r_j,\ {}_{HOJAS}P_z(o), \delta\Big)$$

$$MISMO\_PADRE(i) = \{l \in HOJAS \mid V_{l,A_l(j)} = V_{i,A_i(j)} \ \wedge\ j \in \{1,\ldots,N_i - 1\}\} - \{i\}$$

**Estudio de la mejora y su complejidad**

Usar este tipo de predicción que emplea información más específica para etiquetar la clase de una observación no supone ninguna sobrecarga durante la ejecución del algoritmo, puesto que los contadores se seguirán actualizando igual que lo hacían antes. La única sobrecarga aparecerá en el momento de realizar la predicción, situación que comentaremos a continuación, pero antes queremos ilustrar cómo mejora la predicción ofrecida por el modelo usando las hojas funcionales que incorporan clasificadores ingenuos de Bayes.

Existe la posibilidad de limitar el uso de dichas hojas funcionales en base al número de experiencias que han sido registradas en cada una de ellas como se hace en otros algoritmos (por ejemplo VFDTc [Gam-2006]) pero en nuestro caso no hemos establecido ninguna limitación. Usar el clasificador ingenuo de Bayes en cualquier caso sin considerar ningún número mínimo de experiencias observadas no perjudica a la predicción: esa es la filosofía del teorema de Bayes. Cualquier dato de que se disponga puede ser usado porque permite que las decisiones se produzcan con más información y sean, al menos, tan buenas como las decisiones tomadas sin esa información. En los experimentos que hemos realizado también se observa ese comportamiento. En la figura 3.18 hemos representado la precisión alcanzada por el modelo estableciendo diferentes valores para el número mínimo de experiencias que se ha debido observar en una hoja para que se pueda usar su clasificador ingenuo de Bayes. Si no se usase, se predeciría con la clase mayoritaria. El conjunto de datos es el mismo que hemos usado anteriormente con 20 atributos categóricos y un 15 % de ruido.

En la figura 3.18 únicamente hemos representado la precisión, puesto que es lo único que varía por el hecho de usar el enfoque que estamos presentado, pero, para facilitar la visualización, hemos incluido dos gráficas de la precisión: la de la izquierda representa el comportamiento para un rango de valores más amplio y el de la derecha se centra en los valores más cercanos a 0. Como puede observarse, a partir de un valor (en este caso cercano a las 20000 experiencias), deja de usarse la predicción que usa el clasificador ingenuo de Bayes y el comportamiento es estable a partir de él. Pero la predicción mejora cuanto más nos acerquemos a 0, que sería el punto en el que se usa el clasificador ingenuo de Bayes siempre. Por lo tanto, no hemos incluido ningún parámetro interno que controle esta enfoque, que será usado independientemente del número de experiencias que hayan sido observadas en la hoja correspondiente.

Si consideramos el incremento en la complejidad que se deriva de este enfoque podemos apreciar que, como se comentó anteriormente, no supone ninguna variación en la fase de entrenamiento. El incremento en la complejidad se produce en el momento de realizar la predicción para una experiencia ya que, en vez de calcular las frecuencias relativas para las diferentes clases tenemos que aplicar el teorema de Bayes para cada una de las clases. Cada aplicación del teorema de Bayes supone que, en vez de hacer el cálculo de una

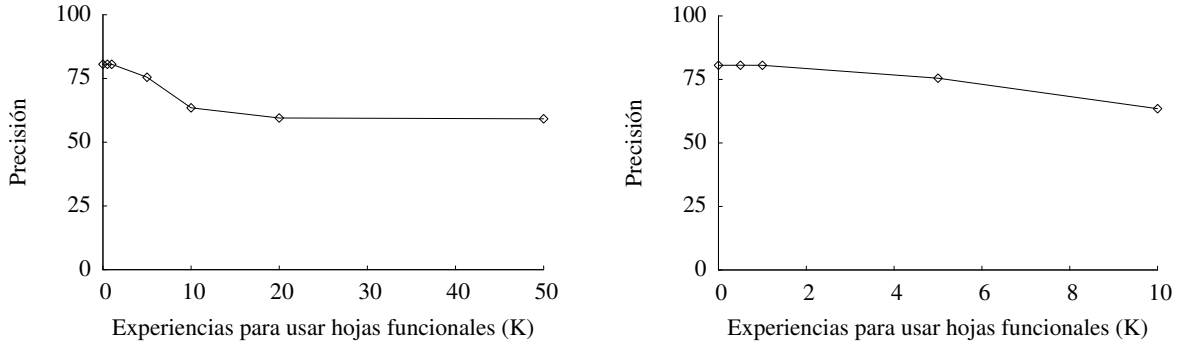



Figura 3.18: Comportamiento dependiendo del número de experiencias mínimo que han debido ser observadas por una hoja para usar la predicción basada Bayes ingenuo. El número de experiencias está expresado en miles (K).

frecuencia, es preciso hacer el cálculo de esa frecuencia más un cálculo por cada uno de los atributos que no hayan sido usados en la hoja correspondiente. Además, como los atributos continuos pueden considerarse en todo momento como atributos libres, es preciso hacer una discretización de los atributos continuos en un máximo de 10 intervalos de igual anchura. Por lo tanto, estaríamos pasando de una complejidad $\mathcal{O}(k)$, donde $k$ es el número de clases, a una complejidad $\mathcal{O}(k \cdot a \cdot b'')$ donde $a$ es el número de atributos del problema y $b''$ es el máximo número de valores diferentes de los atributos continuos.

En la práctica, este incremento en la complejidad temporal que supone dar la predicción para una observación puede reducirse drásticamente. Típicamente, la fase en la que se usa el modelo para predecir es posterior a la fase en la que se induce el modelo. Por lo tanto, sería factible generar, a partir del modelo, un árbol que ya tuviese calculadas todas las frecuencias de todos los valores de los atributos, incluyendo las discretizaciones de los atributos continuos. De esa forma, el incremento en la complejidad pasaría de $\mathcal{O}(k)$ a $\mathcal{O}(k \cdot a)$.

# Capítulo 4

# Estudio Experimental

Tras haber realizado una detallada descripción del algoritmo IADEM-2 es necesario evaluar sus características y compararlas con las propias de otros algoritmos. En este capítulo recogemos el estudio experimental que nos permitirá discernir las ventajas que se pueden esperar al utilizar el algoritmo IADEM-2 para extraer conocimiento y que también nos servirá para identificar las limitaciones que presenta dicho algoritmo. Para ello empezaremos describiendo, en la sección 4.1, los algoritmos que van a intervenir en las comparaciones, el tipo de experimentos que se van a realizar y otros detalles relevantes. En las secciones 4.2 y 4.3 realizaremos un estudio del comportamiento del algoritmo atendiendo al tamaño del conjunto de datos y al ruido presente en él. A continuación, en las secciones 4.4 y 4.5, mostraremos los resultados obtenidos al aplicar diferentes algoritmos a distintos tipos de conjuntos de datos (reales y sintéticos, grandes y pequeños, con ruido y sin ruido) y finalmente, en la sección 4.6, haremos un resumen de las principales características que se hayan observado en el estudio experimental.

## 4.1. Detalles de la experimentación

Antes de comentar los experimentos que se han realizado sobre los distintos conjuntos de datos y los resultados que se han obtenido es necesario describir cómo hemos realizado dichos experimentos. En los apartados que componen esta sección daremos los detalles que se refieren a los algoritmos que se han usado y los elementos que han sido estudiados. Además, presentaremos un nuevo índice que trata de concentrar en una única medida los elementos interesantes que son objeto de estudio, permitiendo realizar comparaciones más sencillas que recojan toda la información relevante.

### 4.1.1. Algoritmos usados en el estudio experimental

En este apartado enumeraremos los algoritmos que se han usado en el estudio experimental y detallaremos la características más interesantes de cada uno de ellos para justificar su elección.

La referencia para realizar las comparaciones será el algoritmo IADEM-2 por ser el centro de esta parte de la tesis. Se trata de seleccionar algoritmos cuya implementación esté disponible para ser usada y que reúnan una serie de características que nos permitan evaluar las diferentes cualidades del algoritmo en cuestión: IADEM-2. El hecho de que necesitemos algoritmos que tengan disponible su implementación (independientemente del lenguaje de programación empleado) presenta la primera gran limitación a la que nos enfrentamos a la hora de realizar la selección.

Puesto que IADEM-2 es un algoritmo que induce árboles de decisión como modelo para representar el conocimiento que se está generando, hemos restringido el tipo de algoritmos seleccionados a aquellos que también inducen árboles de decisión. La razón principal es que a la hora de realizar las comparaciones, éstas son más sencillas y, sobre todo, mucho más comprensibles que las que se producirían si eligiésemos otros modelos. También hay que destacar que el tratamiento que hacen del espacio de búsqueda, aunque no sean exactamente iguales por ser algoritmos diferentes, sigue un enfoque con características similares.

También es necesario considerar las características que presenta el algoritmo IADEM-2 para buscar algoritmos que puedan ser comparados con él en todos o casi todos los casos. Así, se hace aconsejable elegir algoritmos que sean capaces de trabajar sobre conjuntos de datos en los que puedan aparecer tanto atributos categóricos como continuos, y que sean capaces de trabajar con grandes volúmenes de datos. Para evaluar el efecto que tienen los diferentes modelos de memoria de experiencias sobre el aprendizaje incremental, hemos seleccionado los algoritmos de forma que se evalúen los tres modelos: memoria total, memoria parcial y sin memoria.

A ninguno de los algoritmos que se han usado les ha sido modificada su configuración por defecto, es decir, la realización de los experimentos ha sido realizada con los algoritmos tal y como se encuentran disponibles. Considerando los requisitos que acabamos de exponer, pasamos a enumerar los algoritmos que se han utilizado:

- **J48**: es la implementación del algoritmo C4.5 [Qui-1993] presente en la herramienta Weka [Wit-2005] disponible en Internet (`http://www.cs.waikato.ac.nz/ml/weka`). Este algoritmo induce árboles de decisión e incorpora un mecanismo de post-poda para hacerlos más comprensibles y para mejorar la generalización del modelo. Es capaz de extraer conocimiento de conjuntos de datos con atributos categóricos y continuos. Los nodos de decisión etiquetados con atributos categóricos tienen tantas ramas como valores haya definidos para dicho atributo y los que están etiquetados con atributos continuos presentan divisiones dicotómicas (pudiendo ser usado el mismo atributo continuo tantas veces como sea necesario en la misma rama). Este algoritmo usa un modelo de memoria total, por lo que necesita almacenar todas las experiencias en la memoria principal antes de inducir un modelo.

- **ITI-total**: es la variante del algoritmo ITI que usa un modelo de memoria total (es la variante por defecto). Este algoritmo fue diseñado por Utgoff [Utg-1994] y, tras diversos cambios [Utg-1997], su versión más moderna data de 2001, la cual puede encontrarse en `http://www-lrn.cs.umass.edu/iti/index.html`. Este algoritmo ya incorpora la incrementalidad como una de sus características, a diferencia del J48 anterior. El tipo de árboles de decisión que induce no es igual que el de ninguno de los otros algoritmos que usaremos en el apartado de experimentación. Todos los nodos de decisión son binarios, tanto los de atributos categóricos como los de atributos continuos. Esta diferencia puede permitir que aparezcan árboles más pequeños: dado un árbol de decisión binario, si quisiésemos obtener el árbol equivalente usando árboles de decisión que expanden los atributos categóricos usando todos los valores y no realizando agrupaciones binarias, obtendríamos un árbol con un mayor número de ramas (o igual en el caso de que los atributos categóricos sólo presentasen dos valores). Además de esta salvedad, hemos de destacar que la forma en que se hace incremental al algoritmo no es similar a las que expondremos a continuación y que son más comunes. El algoritmo toma las experiencias del conjunto de datos de una en una (no por lotes como hacen otros algoritmos) y, después de tomar cada experiencia, puede producirse una reestructuración del árbol para hacerlo igual al que se habría obtenido si se hubiesen tomado todas las experiencias observadas hasta ese momento en un único bloque. Esta forma de procesar las experiencias supondrá un elevado consumo de tiempo antes de concluir la inducción, como se podrá observar en distintos casos. Existe la posibilidad de ejecutar el algoritmo por lotes, cargando todas las experiencias de una sola vez, pero hemos preferido no usar esta característica porque estaríamos privando al algoritmo de su carácter incremental.

- **ITI-parcial**: es la variante del algoritmo ITI que usa un modelo de memoria parcial. El algoritmo dispone de la posibilidad de usar un enfoque llamado "*modo de corrección de errores*" ("*error correction mode*") en el que las únicas experiencias que se almacenan son aquellas que han sido incorrectamente clasificadas por el árbol. Hemos considerado interesante emplear esta variante para considerar un algoritmo que induce árboles de decisión y que usa un modelo de memoria parcial.

- **VFDTc**: este algoritmo propuesto por Gama y otros [Gam-2003; Gam-2006] ha servido como base para determinadas aportaciones que se han presentado en los capítulos anteriores y, aunque algunas de sus ideas han sido modificadas para aprovechar mejor las características propias del algo-

ritmo IADEM-2, sigue teniendo aspectos en común con él. Es un algoritmo incremental, sin memoria de experiencias, que induce árboles de decisión. El tipo de expansiones que realiza (múltiples ramas para atributos categóricos y dos ramas para atributos continuos) es la misma que la del algoritmo J48 y que la del algoritmo IADEM-2. Incorpora hojas funcionales que inducen clasificadores ingenuos de Bayes, permitiendo elegir entre la predicción clásica (usando la clase mayoritaria) y la predicción de las hojas funcionales. Siempre consideraremos el mejor resultado de los dos ofrecidos. La implementación de este algoritmo está disponible en la siguiente dirección `http://www.liacc.up.pt/~jgama/ales2/vfdtc`.

- **IADEM**: para evaluar las aportaciones introducidas al algoritmo IADEM-0 [Ram-2000; Ram-2001; Ram-2004] y que han dado lugar al nuevo algoritmo IADEM-2 [Cam-2007], nos hemos visto en la necesidad de utilizar el algoritmo IADEM [Cam-2002; Ram-2006a], que únicamente se diferencia de IADEM-0 en que permite tratar conjuntos de datos en los que haya presencia de ruido (como se expuso en la sección 3.1). Si no lo hiciésemos así, y usásemos directamente el algoritmo IADEM-0, el tipo de experimentos que podríamos abordar sería demasiado limitado. Las entradas al algoritmo, además del conjunto de datos se han configurado del siguiente modo: $\varepsilon = 0,01$ y $\delta = 0,05$. Los parámetros internos toman sus valores por defecto cuyo estudio se realizó en los capítulos 2 y 3.

- **IADEM-2**: es el algoritmo que estamos presentando y con el que queremos comparar el resto. Su implementación está disponible en la dirección `http://iaia.lcc.uma.es/~jcampo`. Las entradas al algoritmo son las mismas que las del algoritmo IADEM: $\varepsilon = 0,01$ y $\delta = 0,05$.

Todos los algoritmos han sido ejecutados en un ordenador que dispone de un procesador un Intel® Xeon™ con una frecuencia de CPU de $3,2$ GHz. La memoria ha sido limitada para usar un máximo de 512MB, con la intención de evaluar las capacidades de cada algoritmo bajo las mismas condiciones. Únicamente, en determinadas ocasiones (que se indicarán claramente) hemos aumentado el límite de memoria para evaluar los resultados que obtendrían determinados algoritmos que no eran capaces de completar la inducción con el tamaño de memoria previamente fijado.

### 4.1.2. Elementos observados para el estudio experimental

Existen diversas características que resultan interesantes para evaluar las propiedades de los algoritmos. Cuando la tarea es inducir los modelos más sencillos que mejor capturen el conocimiento y mejores resultados ofrezcan cuando se usen para predecir, las características más relevantes son la precisión alcanzada y la complejidad del modelo. En nuestro caso estudiaremos la precisión como el porcentaje de experiencias que son correctamente clasificadas y la complejidad del modelo la evaluaremos teniendo en cuenta el número de hojas que tenga el árbol de decisión. Por cada hoja del árbol existe una rama y cada rama se corresponde con una regla en la que hay tantas decisiones como nodos internos tenga esa rama. Además de estas dos características puede resultar interesante observar el comportamiento del algoritmo teniendo en cuenta el tiempo que tarda en inducir el modelo y el número de experiencias que procesa antes de hacerlo. En el caso de los algoritmos J48, ITI y VFDTc el número de experiencias procesadas es conocido (tantas como experiencias haya en el conjunto de datos) pero en el caso de IADEM e IADEM-2, este número puede variar debido a que se realiza un muestreo con reemplazamiento.

Para realizar los experimentos hemos realizado una validación cruzada con $10$ particiones y los resultados que se muestran son los obtenidos de hacer la media de las $10$ particiones. Además, en el caso de los valores de precisión y del número de hojas, también mostramos el valor de la desviación típica. Para realizar las comparaciones resulta aconsejable la consideración de algún test estadístico [Her-2004a]. Hemos usado el test de Wilcoxon [Wil-1945; Sie-1988] debido a que es un test estadístico no paramétrico que no exige normalidad en la distribución. Lo hemos calculado usando la herramienta estadística R [R-2005] exigiendo una confianza de $0,95$. Para indicar diferencias significativas entre los resultados de dos algoritmos hemos establecido como algoritmo de referencia a IADEM-2 y hemos usado los símbolos $\oplus$ y $\ominus$. Con el símbolo $\oplus$ indicamos que el resultado es significativamente mejor que el obtenido por IADEM-2; con el símbolo

$\ominus$ indicamos que el resultado es significativamente peor y si no usamos ningún símbolo indicamos que no existen diferencias significativas.

### 4.1.3. Índice combinado para el estudio de los elementos más relevantes

Acabamos de mencionar que se medirán tanto la precisión como la complejidad del árbol (considerando el número de hojas) como las características más relevantes del modelo que se induzca, pero, evaluar esas características de forma aislada no siempre resulta sencillo. Si un modelo es el que tiene menor número de hojas y al mismo tiempo es el que alcanza una menor precisión, está claro que ese modelo será el peor. Cuando, dados dos modelos, la precisión sea similar, preferiremos el que tenga menor número de hojas y si lo que tenemos son dos árboles con un número similar de hojas, lo que preferiremos será aquél que nos ofrezca la mejor precisión. El problema surge cuando ninguna de las dos características es similar a otra. En ese caso nos interesará combinar ambas características en una nueva medida que recoja todas las características interesantes al mismo tiempo.

Para evaluar estas mismas características en los árboles de decisión generados por el algoritmo CIDIM [Ram-2005c], un algoritmo no incremental, Ramos, del Campo y Morales propusieron el uso de un índice combinado ($IC$) que recogía la mejora producida en ambas características, considerando como modelo de partida uno con las características mínimas exigibles. En la figura 4.1 hemos representado un esquema con los cálculos necesarios para estimar dicho índice combinado. Las características mínimas exigibles en cuanto a precisión es que, al menos, sea mayor que la probabilidad de la clase mayoritaria (modelo de predicción más sencillo). En lo que se refiere al tamaño del árbol, lo mínimo exigible es que tenga, como máximo, tantas hojas como el árbol que induzca el algoritmo ID3[Qui-1986], que es un algoritmo que induce árboles de decisión bastante grandes.

$Error$ = 100 - porcentaje de acierto con el árbol inducido
$ErrorMayor$ = 100 - probabilidad de la clase mayoritaria (fija para cada experimento)
$Mejora = ((ErrorMayor - Error)/ErrorMayor)$

$Hojas$ = número de hojas en el árbol inducido
$HojasID3$ = número de hojas que tiene el árbol de decisión inducido por el algoritmo ID3
$Reducción = (HojasID3 - Hojas)/HojasID3$

$Índice\_Combinado\ (IC) = Mejora \times Reducción$

Figura 4.1: Cálculo del índice combinado $Índice\_Combinado\ (IC)$.

Existe un área de conocimiento que se dedica a tratar estos problemas en los que se tienen más de un elemento a considerar y no se tiene ninguna medida que recoja las características de forma global. Recibe el nombre genérico de "*toma de decisiones con múltiples criterios*" (sus siglas en inglés son *MCDM*). Este área puede dividirse en dos categorías diferentes según sus objetivos [Hwa-1981]:

- *Toma de decisiones con múltiples atributos o características* (sus siglas en inglés son *MADM*): se usa para la selección de la combinación de atributos (o características) que maximice unos objetivos implícitos. Los criterios vienen definidos por los atributos o características y el número de alternativas (modelos) es finito puesto que los posibles valores de las características ya han sido definidos (o calculados).

- *Toma de decisiones con múltiples objetivos* (sus siglas en inglés son *MODM*): se usa para diseñar métodos que deben satisfacer una serie de restricciones que se presentan de forma explícita. Los

Sea $m$ el número de alternativas (modelos) a considerar.
Sea $A_i$ cada una de las alternativas donde $i \in \{1, \ldots, m\}$.
Sea $n$ el número de características a considerar en cada alternativa (modelo).
Sea $C_j$ cada una de las características donde $j \in \{1, \ldots, n\}$.
Sea $X$ la matriz con los valores de todas las características para todas las alternativas (modelos):

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ x_{21} & x_{22} & \ldots & x_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ x_{m1} & x_{m2} & \ldots & x_{mn} \end{bmatrix}$$

Sea $W$ la matriz de pesos para ponderar las diferentes características.
Sea $w_j$ cada uno de los pesos donde $j \in \{1, \ldots, n\}$, $w_j \in [0, 1]$ y $\sum_{j=1}^{n} w_j = 1$
Sea $r_{ij}$ el porcentaje normalizado de la mejora de la característica $C_j$ en la alternativa $A_i$:

$$r_{ij} = \begin{cases} \dfrac{x_{ij}}{\max\limits_{i} x_{ij}} & \text{si } C_j \text{ es una característica beneficiosa} \\ \dfrac{\min\limits_{i} x_{ij}}{x_{ij}} & \text{si } C_j \text{ es una característica costosa} \end{cases}$$

con $i \in \{1, 2, \ldots, m\}$ y $j \in \{1, 2, \ldots, n\}$

Definimos el valor con la medida global calculada del siguiente modo:

$$V_i = \sum_{j=1}^{n} w_j \cdot r_{ij} \quad \text{con } i \in \{1, 2, \ldots, m\}$$

Figura 4.2: Cálculo del método de ponderación aditiva simple (SAW).

criterios vienen definidos por los objetivos y el número de alternativas es, a priori, infinito puesto que la solución aún no se ha calculado.

Dado nuestro problema, en el que ya disponemos de las características de los modelos que se han inducido y lo que queremos es compararlos para decidir cuál tiene en global mejores características, nos encontramos ante un problema de *toma de decisiones con múltiples atributos o características*. Existen múltiples trabajos sobre este tema, pero en un artículo de Zanakis, Solomon, Wishart y Dublish [Zan-1998] podemos encontrar una comparativa de los métodos más relevantes en ese ámbito. Una de las críticas que se le hacen a estos métodos, como se plantea en el trabajo de Gershon y Duckstein [Ger-1983], es su inconsistencia, es decir, la diferencia de resultados que surgen de aplicar diferentes métodos ante el mismo problema. De hecho, para cada problema suele estudiarse qué método puede resultar más apropiado [Noh-2003; Yeh-2003].

El índice combinado $IC$ podría considerarse un método de toma de decisiones con múltiples atributos, pero tiene la particularidad de exigir el establecimiento de unos valores de referencia mínimos. La precisión mínima exigible sí puede ser calculada, pero el número de hojas inducido por el algoritmo ID3 no es aplicable a problemas cuyo conjunto de datos sea demasiado grande. Es por eso necesario que establezcamos un nuevo índice combinado que sea útil, al menos para el caso que nos concierne: el aprendizaje incremental.

Un método bastante sencillo para la toma de decisiones con múltiples atributos y que ofrece buenos resultados [Zan-1998] es el de "*ponderación aditiva simple*" (sus siglas en inglés son *SAW*) cuyo cálculo presentamos en la figura 4.2. Su funcionamiento básicamente consiste en realizar una suma ponderada de los porcentajes de rendimiento de cada una de las características del individuo (modelo) que se está estudiando. Presenta una serie de propiedades que resultan interesantes como no ser sensible al cambio de escala en las características que está considerando, o que los pesos tomen valores entre 0 y 1 y que su suma sea

igual a 1. Pero carece de otras como asignar el valor 0 al individuo que presente los peores valores en todas las características y el valor 1 a aquél que presente los mejores valores, con lo que se conseguiría la mayor diferenciación entre los dos individuos extremos (esta característica sí aparece en el método llamado "*técnica para asignar orden de preferencia por parecido a la solución ideal*" cuyas siglas en inglés son *TOPSIS* [Hwa-1981]). También aparece como una desventaja el hecho de que las medidas se vean afectadas dependiendo de la distancia al origen de los valores de las características, en vez de considerar la amplitud del intervalo. Por ejemplo, una característica no interviene de igual forma en el cálculo global de la medida cuando dos individuos toman los valores 110 y 120 que cuando toma los valores 1010 y 1020, a pesar de que el intervalo de mejora es, para ambos casos, de 10 unidades. El método de ponderación aditiva simple considera que todas las características podrían tomar valores mayores o igual que 0, y en ese caso, el comportamiento anterior estaría justificado. Sin embargo, en nuestro caso eso no es deseable puesto que no siempre podemos garantizar que las características puedan tomar valores a partir de 0, de hecho, en ningún caso existirá una característica cuyo peor valor sea 0 puesto que la precisión será mayor que 0 y el tamaño del árbol también.

Para solventar las carencias observadas en el método de ponderación aditiva simple hemos hecho una sencilla modificación que da lugar al método de "*ponderación aditiva relativa*" (sus siglas en inglés serían *RAW*). En la figura 4.3 presentamos cómo se calcula. Usaremos este método para calcular un nuevo índice combinado que puede emplearse con los modelos generados por algoritmos incrementales y que evite tener que establecer unos valores de referencia mínimos. Ésta medida será usada en las secciones 4.4 y 4.5 para evaluar los modelos inducidos por los diferentes algoritmos.

Sea $m$ el número de alternativas (modelos) a considerar.
Sea $A_i$ cada una de las alternativas donde $i \in \{1, \ldots, m\}$.
Sea $n$ el número de características a considerar en cada alternativa (modelo).
Sea $C_j$ cada una de las características donde $j \in \{1, \ldots, n\}$.
Sea $X$ la matriz con los valores de todas las características para todas las alternativas (modelos):

$$X = \begin{bmatrix} x_{11} & x_{12} & \ldots & x_{1n} \\ x_{21} & x_{22} & \ldots & x_{2n} \\ \ldots & \ldots & \ldots & \ldots \\ x_{m1} & x_{m2} & \ldots & x_{mn} \end{bmatrix}$$

Sea $W$ la matriz de pesos para ponderar las diferentes características.
Sea $w_j$ cada uno de los pesos donde $j \in \{1, \ldots, n\},\ w_j \in [0, 1]$ y $\sum_{j=1}^{n} w_j = 1$
Sea $r_{ij}$ el porcentaje normalizado de la mejora de la característica $C_j$ en la alternativa $A_i$:

$$r_{ij} = \begin{cases} \dfrac{\max\limits_i x_{ij} \ - \ x_{ij}}{\max\limits_i x_{ij} \ - \ \min\limits_i x_{ij}} & \text{si } C_j \text{ es una característica beneficiosa} \\ \\ \dfrac{x_{ij} - \min\limits_i x_{ij}}{\max\limits_i x_{ij} \ - \ \min\limits_i x_{ij}} & \text{si } C_j \text{ es una característica costosa} \end{cases}$$

con $i = 1, 2, \ldots, m$ y $j = 1, 2, \ldots, n$

Definimos el valor con la medida global calculada del siguiente modo:

$$V_i = 1 - \sum_{j=1}^{n} w_j \cdot r_{ij} \quad \text{con } i = 1, 2, \ldots, m.$$

Figura 4.3: Cálculo del método de ponderación aditiva relativa (RAW).

En este capítulo usaremos ambos métodos eligiendo la precisión y el tamaño del árbol de decisión como las características relevantes para calcular el índice combinado. Puesto que estamos más interesados en conseguir árboles precisos antes que árboles pequeños, los pesos que hemos usado han sido: $0,7$ para la precisión y $0,3$ para el tamaño. De esta forma tendremos las dos características en cuenta, pero daremos prioridad a la precisión.

## 4.2. Estudio del comportamiento en función del tamaño del conjunto de datos

El primer estudio que realizaremos para observar el comportamiento del algoritmo IADEM-2 y del resto de algoritmos tendrá como objetivo el tamaño del conjunto de datos. Para ello usaremos un generador de conjuntos de datos que existe en el repositorio de la UCI [New-1998]. El generador en cuestión recibe el nombre de "*LED*" debido a que las experiencias reflejan el estado de un display de 7 segmentos (LEDs). Se define mediante $24$ atributos binarios (con valores $0$ ó $1$) de los cuales $17$ son irrelevantes y los $7$ restantes son los que se usan para determinar la clase, que puede tomar $10$ valores diferentes (los $10$ dígitos diferentes que presenta un display de $7$ segmentos). Al ser binarios los atributos del problema, la diferencia en el tamaño de los árboles que pudiera derivarse del tipo de expansión que usan los algoritmos (binaria el ITI y multivaluada el resto) se elimina, favoreciendo la comparación. La principal ventaja de usar este generador es que podemos obtener conjuntos de datos de diferentes tamaños de un problema real y estudiar el comportamiento de los algoritmos ante diversas condiciones.

Los primeros datos que mostramos están recogidos en la figura 4.4 y muestran la precisión, el tamaño del árbol (número de hojas), el número total de experiencias observadas para inducir el modelo y el tiempo consumido. El conjunto de datos generados no presenta ruido y, por lo tanto, una precisión del $100\,\%$ es alcanzable. Hemos generado los modelos usando distintos tamaños para el conjunto de datos (desde $50000$ experiencias hasta 2 millones – 50K hasta 2000K–).

En las gráficas de la figura 4.4 podemos destacar las siguientes observaciones:

- la precisión que alcanzan todos los modelos es siempre del $100\,\%$, independientemente del algoritmo y del tamaño del conjunto de experiencias. Con esto vemos cómo, ante la ausencia de ruido, todos los algoritmo son capaces de inducir modelos que clasifican correctamente el conjunto de datos al completo. La única salvedad que hemos de comentar, puesto que no se aprecia correctamente en esta gráfica (se aprecia mejor en la gráfica que representa el tiempo consumido), es que el algoritmo J48 no es capaz de inducir ningún modelo cuando trata con conjuntos de datos mayores de $1$ millón de experiencias debido a que se queda sin memoria. Lo mismo le ocurre a las dos variantes del algoritmo ITI, que no son capaces de completar la inducción cuando el tamaño es de 2 millones de experiencias. Aquí se pueden observar las carencias propias de los algoritmos que usan algún tipo de modelo de memoria, ya sea parcial o total: es posible que no sean capaces de afrontar problemas cuyo tamaño sea excesivamente grande.

- el tamaño alcanzado por todos los modelos es de $10$ hojas. La única salvedad es el caso del algoritmo VFDTc cuando el conjunto de datos tiene menos de $250$K experiencias puesto que induce árboles más pequeños. El menor tamaño se debe a que, con pocas experiencias, el algoritmo no es capaz de alcanzar la confianza necesaria para realizar las expansiones que le lleven a inducir el árbol con las $10$ hojas. A pesar de ello, la precisión sigue siendo máxima y esto es debido al uso de las hojas funcionales. Si se usase la predicción por la clase mayoritaria en los casos en los que VFDTc aún no ha inducido el árbol con $10$ hojas, la precisión sería del $50\,\%$ (para el caso en que el conjunto de datos tuviese $50$K experiencias) o del $70\,\%$ (para el caso en que tuviese $100$K experiencias). Este es uno de los primeros ejemplos en los que se pueden observar los beneficios del uso de hojas funcionales.

- el número de experiencias observadas por los algoritmos J48, ITI (en sus dos variantes) y VFDTc es siempre el correspondiente al tamaño del conjunto de datos del que tienen que extraer conocimien-

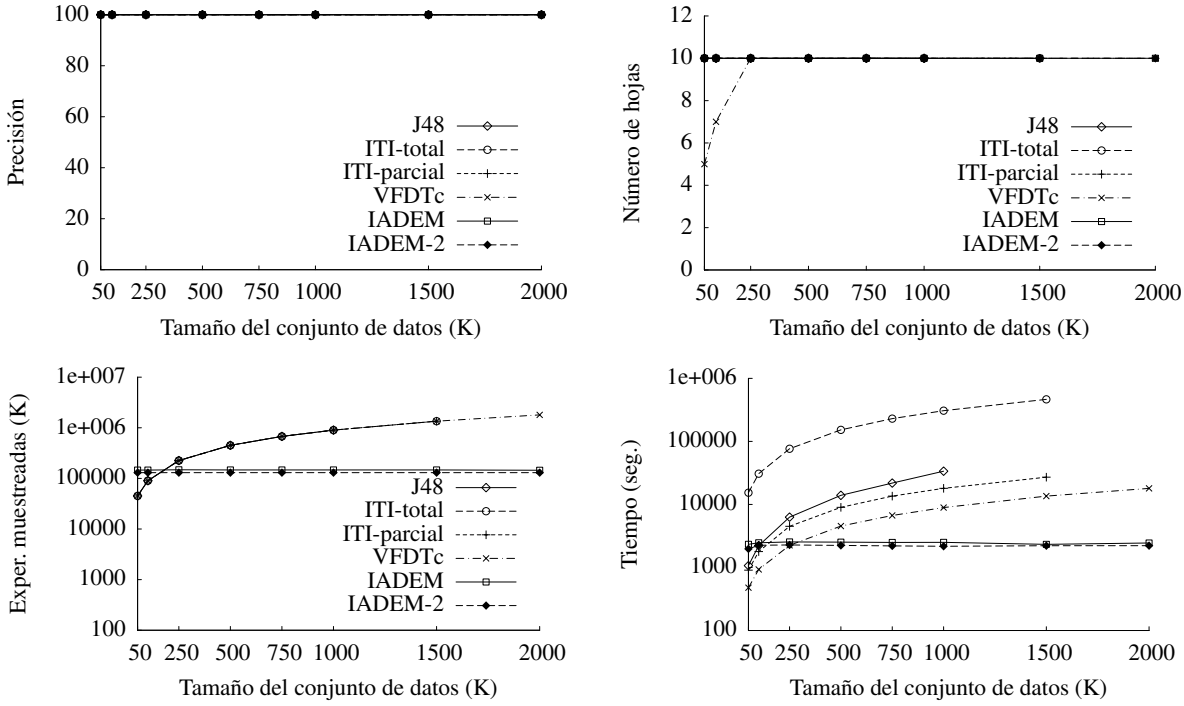



Figura 4.4: Gráficas del valor medio de la precisión, del tamaño del árbol, del número total de experiencias muestreadas y del tiempo consumido en función del tamaño del conjunto de datos (expresado en miles). Los conjuntos de datos han sido creados con el generador *LED* y sin presencia de ruido.

to. Por contra, el número de experiencias muestreadas por el algoritmo IADEM y por el algoritmo IADEM-2 se mantiene de forma bastante estable independientemente del tamaño del conjunto de datos. Esta es una de las principales ventajas que hemos podido observar hasta el momento propias de este tipo de algoritmos: el algoritmo muestra un comportamiento satisfactorio incluso en los casos en los que el conjunto de datos es pequeño, debido a que se realizan muestreos con reemplazamiento hasta que el modelo sea finalmente inducido. Es posible que el total de experiencias procesadas (muestreadas) sea mayor que el total de experiencias en el conjunto de datos pero, para un mismo problema, ese tamaño suele ser estable. La ventaja surge en el momento en el que el número de experiencias necesarias para inducir el modelo es menor que el tamaño del conjunto de datos, puesto que el proceso de inducción no necesitará observar el conjunto completo y podrá detenerse de forma anticipada, mejorando al resto de algoritmos a partir de ese punto.

- el tiempo consumido introduce algo de información nueva que no hemos podido observar en las gráficas anteriores. La característica más relevante sigue siendo la diferenciación entre los algoritmos J48, ITI (en sus dos variantes) y VFDTc y los algoritmos IADEM e IADEM-2. Mientras los primeros necesitan tiempo lineal para inducir el modelo, los segundos sólo precisan tiempo constante. La razón es la que acabamos de comentar anteriormente: con IADEM e IADEM-2, una vez se alcanza el modelo deseado (con la confianza requerida), la inducción se detiene sin tener que consumir el conjunto completo de experiencias. Como ese modelo se alcanza consumiendo un número similar de experiencias, independientemente del tamaño del conjunto de datos, el tiempo consumido también se mantiene estable.

  En cuanto a las comparaciones entre los algoritmos J48, ITI (en sus dos variantes) y VFDTc podemos observar que VFDTc es el algoritmo más rápido y que el más lento es ITI-total, debido principalmente al mecanismo que lo hace incremental (recordemos que adquiere las experiencias de una en una). ITI-parcial se comporta mejor en este caso que no tiene ruido debido a que el modelo induce, con pocas

experiencias, un árbol que las clasifica correctamente y ya no necesita almacenar ninguna más en su modelo de memoria parcial.

Acabamos de presentar un experimento en el que hemos usado conjuntos de datos en los que no había ruido pero, el generador de conjuntos de datos de *LED* también permite crearlos con ruido. En la figura 4.5 hemos representado las mismas gráficas del caso anterior, pero han sido obtenidas después de aplicar los algoritmos a conjuntos de datos de diferentes tamaños en los que se ha introducido un $10\,\%$ de ruido.

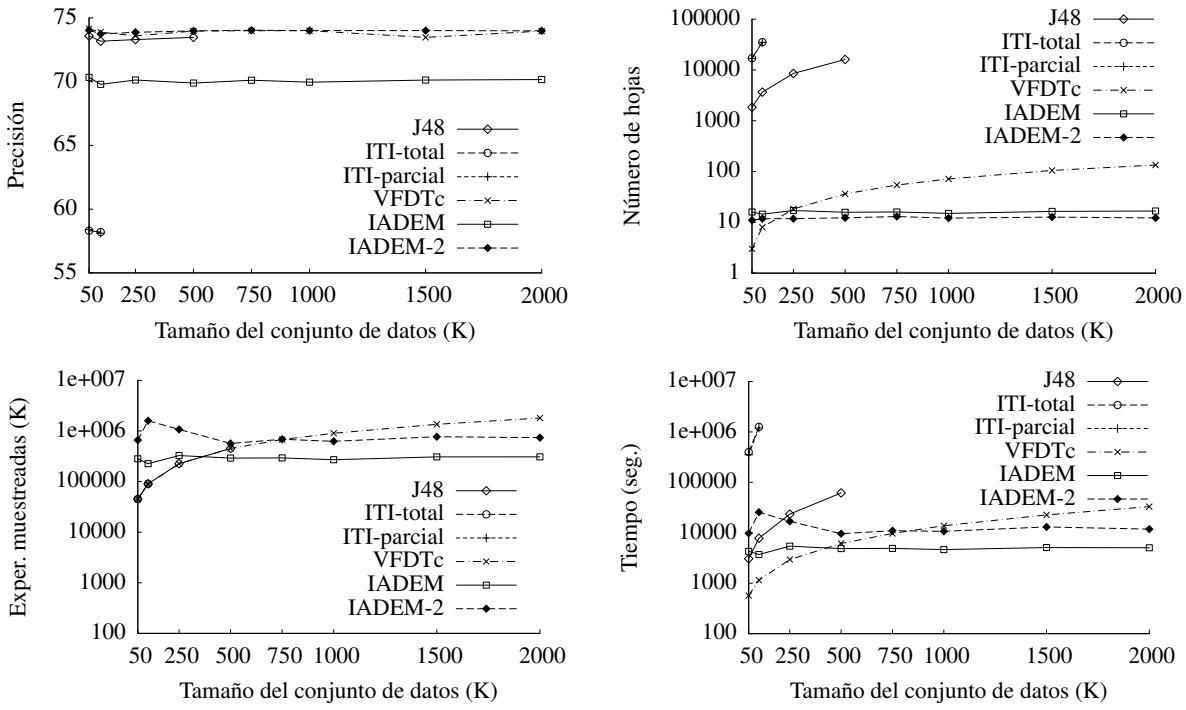



Figura 4.5: Gráficas del valor medio de la precisión, del tamaño del árbol, del número total de experiencias muestreadas y del tiempo consumido en función del tamaño del conjunto de datos (expresado en miles). Los conjuntos de datos han sido creados con el generador *LED* y con presencia de ruido del $10\,\%$.

En las gráficas de la figura 4.5 podemos destacar las siguientes observaciones:

- la precisión que alcanzan los modelos ya no es del $100\,\%$ como es lógico. Los mejores valores para la precisión lo alcanza el modelo inducido con el algoritmo IADEM-2. Muy cerca de estos resultados, siendo a veces iguales, se encuentran los modelos inducidos por el algoritmo VFDTc. El último algoritmo que ha completado la inducción de un modelo y no se ha visto limitado por la memoria en ningún caso ha sido el algoritmo IADEM, aunque su precisión no ha sido tan elevada. Aquí volvemos a apreciar la principal ventaja que se deriva del uso de hojas funcionales: el incremento en la precisión.

  En cuanto a los algoritmos J48 e ITI (en sus dos variantes) debemos indicar que sufren las limitaciones de memoria en mayor grado debido al ruido introducido. Mientras que antes, cuando no existía ruido, eran capaces de concluir la inducción en casi todos los casos, ahora no ocurre igual. El modelo inducido por J48 alcanza una precisión similar, aunque algo inferior, a la alcanzada por IADEM-2 y VFDTc. Por su parte, los modelos inducidos por las dos variantes del algoritmo ITI consiguen una precisión bastante alejada de la del resto de modelos.

- el tamaño de los árboles inducidos por los algoritmos IADEM e IADEM-2, al igual que ocurría en el conjunto de datos sin ruido, se mantiene estable y apenas presenta diferencias entre los dos modelos (siendo el inducido por IADEM-2 ligeramente menor). En cuanto a los modelos generados por los

algoritmos J48, ITI (en sus dos variantes) y VFDTc observamos un crecimiento lineal con respecto al tamaño del árbol. Este crecimiento es más acusado para las variantes del algoritmo ITI y no puede deberse al tipo de expansión binaria que realiza este algoritmo puesto que este conjunto de datos únicamente presenta atributos binarios. El motivo es un sobreajuste en la presencia de ruido, que es el único factor añadido que diferencia este experimento del anterior. Los modelos inducidos por el algoritmo J48 también son bastante grandes. El excesivo tamaño de los árboles inducidos por los modelos J48 e ITI (en sus dos variantes) son la causa de que el tamaño del conjunto de datos que pueden tratar (por razones de requisitos de memoria) se vea reducido con respecto al caso en que no había ruido. Ahora, el problema de almacenar todas las experiencias en memoria se ve agravado por la necesidad de almacenar una estructura (árbol de decisión) que es cada vez más grande. En este momento podemos mencionar otro problema que puede aparecer en los algoritmos que usan el mecanismo de post-poda (como J48): los requisitos de memoria. Puesto que el árbol es expandido más allá de lo necesario para después ser podado, es posible que la memoria disponible fuese suficiente para almacenar el modelo que resultaría después de ser podado, pero no antes, con la consiguiente imposibilidad de finalizar la inducción.

En cuanto a los modelos inducidos por algoritmos que usan cotas de concentración, es decir, VFDTc, IADEM e IADEM-2, observamos que su tamaño es mucho menor. La idea de retrasar las expansiones hasta contar con garantías de que su realización es conveniente, permite inducir árboles de menor tamaño, principalmente debido a que no se necesita corregir las expansiones prematuras realizadas con poca información.

- el número de experiencias observadas por los algoritmos J48, ITI (en sus dos variantes) y VFDTc sigue correspondiéndose con el tamaño del conjunto de datos del que tienen que extraer conocimiento. Por el mismo motivo expuesto en el experimento anterior donde no había presencia de ruido, el número de experiencias muestreadas por el algoritmo IADEM y por el algoritmo IADEM-2 se mantiene de forma bastante estable independientemente del tamaño del conjunto de datos. Lo que se puede observar es que, ante el aumento de ruido (de un $0\,\%$ a un $10\,\%$), el número de experiencias necesarias se estabiliza en valores superiores, es decir, los algoritmos IADEM e IADEM-2 necesitan más experiencias para inducir el modelo cuando existe ruido. En estos casos, la detención del algoritmo no estará motivada por alcanzar la precisión deseada por el usuario (recordemos $\varepsilon = 0,01$) que no puede ser alcanzada debido al ruido, sino por la estabilización en el aprendizaje. La necesidad de estar seguro de que se ha producido una estabilización es lo que repercute en una mayor cantidad de experiencias muestreadas.

  Analizando el comportamiento de los algoritmos IADEM e IADEM-2 se puede observar que el número de experiencias muestreadas por IADEM-2 es bastante mayor que las muestreadas por IADEM (prácticamente el doble). La razón de este aumento se debe a las múltiples mejoras introducidas, pero principalmente se debe a la forma en que se decide la idoneidad del mejor atributo. Con el nuevo criterio adoptado para decidir cuándo el mejor atributo para realizar una expansión es realmente diferente del resto (o parecido) puede ser necesario almacenar más información, lo que supone muestrear más experiencias; pero ese criterio hace que la calidad de las expansiones mejore, lo que se aprecia en el menor tamaño de los árboles inducidos. En este experimento pasamos de un promedio de $16$ hojas en los modelos inducidos por IADEM a un promedio de $12$ hojas en los inducidos por IADEM-2, lo que supone una reducción del $25\,\%$.

- el tiempo consumido por los diferentes algoritmos tiene un comportamiento similar al observado en el experimento anterior en el que no existía ruido. Sólo es preciso destacar algunas diferencias. El tiempo empleado por las dos variantes del algoritmo ITI es ahora similar y es debido a la presencia de ruido en el conjunto de datos. La variante que usa memoria parcial sólo almacena las experiencias que no son correctamente clasificadas y este número iba reduciéndose conforme aprendía cuando no existía ruido, pero, ante la presencia de ruido, el número de experiencias almacenadas sigue aumentando y presenta un comportamiento similar a la variante con un modelo de memoria total. La otra diferencia

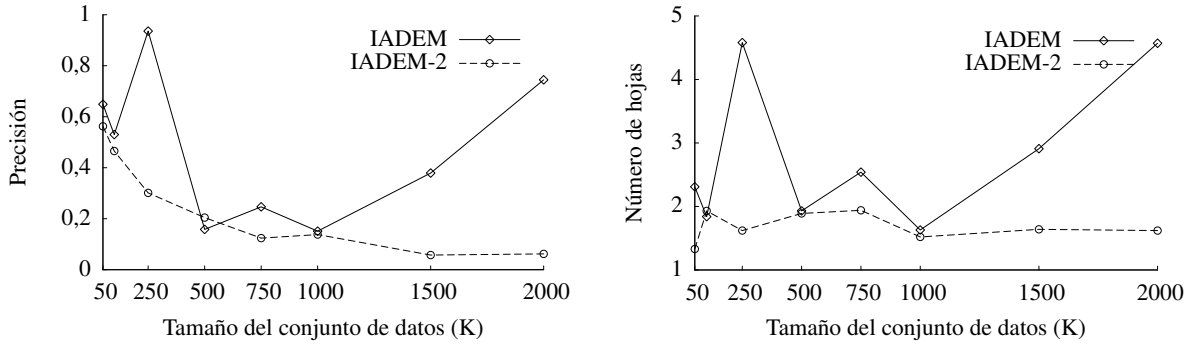



Figura 4.6: Gráficas de la desviación típica de la precisión y del tamaño del árbol en función del tamaño del conjunto de datos (expresado en miles). Los conjuntos de datos han sido creados con el generador *LED* y con presencia de ruido del $10\,\%$.

a destacar es la necesidad del algoritmo IADEM-2 de muestrear más experiencias que IADEM, por lo que el tiempo consumido antes de finalizar la inducción también se incrementa y este es el motivo de la diferencia entre ambos tiempos de ejecución.

Para completar el estudio del comportamiento de los algoritmos en función del tamaño del conjunto de datos hemos considerado interesante añadir un último dato. En la figura 4.6 hemos representado la evolución de la desviación típica observada en los algoritmos IADEM e IADEM-2 para la precisión y para el tamaño del árbol. Como puede observarse, además de las ventajas ya mencionadas que presenta el algoritmo IADEM-2 sobre el algoritmo IADEM, podemos apreciar cómo la desviación típica es menor y más estable en los modelos inducidos por el algoritmo IADEM-2.

## 4.3. Estudio del comportamiento en función del ruido en el conjunto de datos

En la sección anterior estudiamos cómo afecta el tamaño de los conjuntos de datos a la inducción de modelos con los diferentes algoritmos que estamos estudiando. Pero tan sólo consideramos dos casos diferentes en lo referido al nivel de ruido: sin ruido y con un ruido limitado. En esta sección centraremos el estudio en analizar, fijando un tamaño para el conjunto de datos, cómo se comportan los algoritmos ante la presencia de ruido en el conjunto de datos. El tamaño seleccionado ha sido de $1$ millón de experiencias y los niveles de ruido van desde el $0\,\%$ hasta el $30\,\%$. En este caso tan solo consideraremos los algoritmos VFDTc, IADEM e IADEM-2 debido a que el resto (J48 y las dos variantes de ITI) únicamente son capaces de inducir árboles de decisión cuando el conjunto de datos no tiene ruido, por las razones que se expusieron en la sección anterior. En la figura 4.7 presentamos las gráficas con los valores medios de la precisión, del tamaño del árbol, del número total de experiencias observadas y del tiempo consumido.

Después de analizar las gráficas presentadas en la figura 4.7 podemos comentar lo siguiente:

- la precisión, como es lógico, disminuye conforme aumentamos el ruido en el conjunto de datos. Los algoritmos que inducen los modelos más precisos son VFDTc e IADEM-2, cuyas curvas se solapan en la gráfica. Algo más alejada se encuentra la precisión de los modelos inducidos por IADEM, debiéndose principalmente a que no usa hojas funcionales para la predicción.

- el tamaño de los árboles inducidos es el mismo cuando el conjunto de datos no tiene ruido (10 hojas) pero, en cuanto añadimos algo de ruido, el tamaño de los árboles inducidos por VFDTc se incrementa rápidamente. Por su parte, los árboles creados por los algoritmos IADEM e IADEM-2 presentan un tamaño que crece lentamente conforme más ruido se introduce en el conjunto de datos, pero, aún

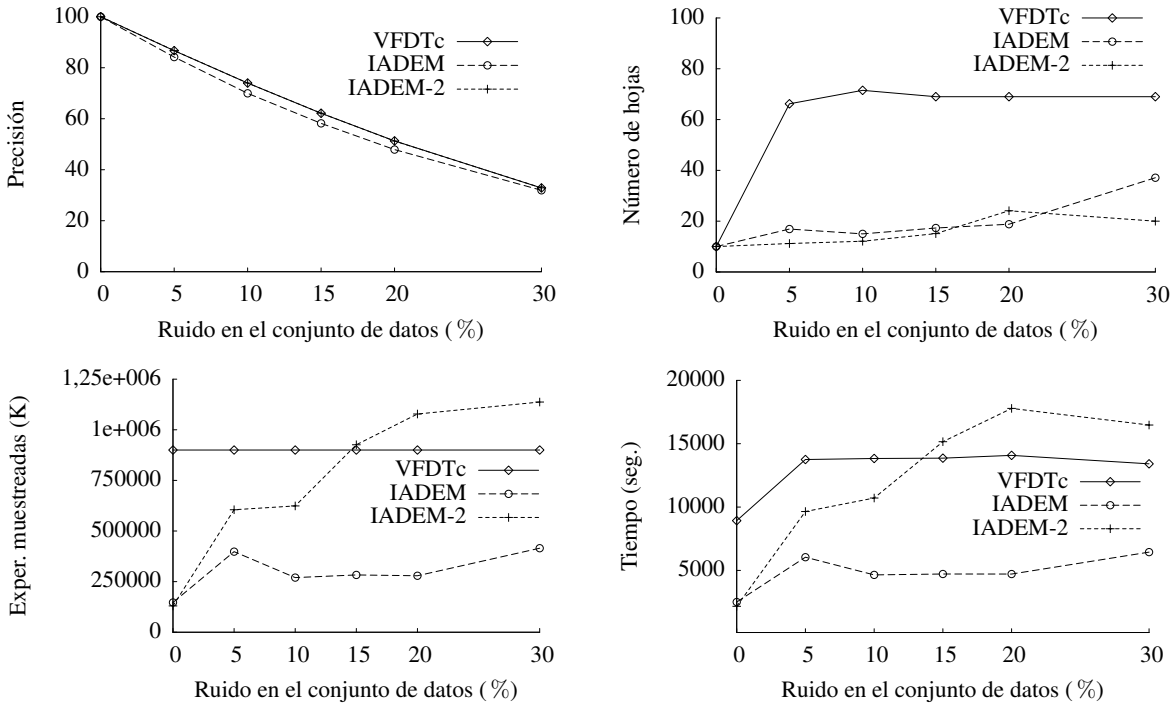



Figura 4.7: Gráficas del valor medio de la precisión, del tamaño del árbol, del número total de experiencias muestreadas y del tiempo consumido en función del ruido introducido en el conjunto de datos. Los conjuntos de datos han sido creados con el generador *LED* y con un tamaño de 1 millón de experiencias.

con ese crecimiento, se mantienen muy por debajo de los creados por el algoritmo VFDTc. Entre las razones que motivan este comportamiento se pueden destacar que los algoritmos que realizan muestreo con reemplazamiento (IADEM e IADEM-2) no realizan una expansión hasta tener garantías de su idoneidad y a que detectan la estabilización en el aprendizaje, deteniendo la inducción antes de que el árbol siga creciendo de forma innecesaria.

- el número total de experiencias observadas por el algoritmo VFDTc se mantiene constante, como ocurría en los experimentos previos, puesto que consume todas las experiencias disponibles en el conjunto de datos (en este caso 900K que son las $\frac{9}{10}$ partes del original ya que estamos realizando una validación cruzada con 10 particiones). Los algoritmos que no usan siempre el mismo número de experiencias son IADEM e IADEM-2 por las mismas razones expuestas anteriormente: sólo muestrean tantas experiencias como sean necesarias. En este caso, el número de experiencias usadas por IADEM se mantiene de forma bastante estable para todos los casos, mientras que las usadas por IADEM-2 son cada vez más, conforme mayor es el nivel de ruido en el conjunto de datos. La justificación principal viene del cambio que supuso pasar del detector de ruido presente en IADEM al detector de estabilización en el aprendizaje presente en IADEM-2. Cuanto mayor es el ruido, más experiencias necesita IADEM-2 para asegurarse de que el aprendizaje se ha estabilizado y no va a conseguir ser mejorado. La ventaja de detener antes la inducción en el caso del algoritmo IADEM se pierde al comprobar que la precisión que se alcanza no es tan elevada.

- el tiempo consumido por el algoritmo VFDTc se mantiene estable, salvo para el caso en el que el conjunto de datos no tiene ruido donde el tiempo consumido es menor. La razón para que este tiempo sea menor es el tamaño del árbol, que en el caso de ausencia de ruido es menor. Cuando el tamaño del árbol permanece estable y el número de experiencias observadas también, el tiempo empleado en realizar la inducción también permanecerá estable. Las observaciones del tiempo consumido por los

algoritmos IADEM e IADEM-2 son similares a las ofrecidas para el caso del número de experiencias muestreadas, puesto que están relacionadas: muestrear más experiencias supone consumir más tiempo para procesarlas.

## 4.4. Evaluación con conjuntos de datos de tamaño reducido

En esta sección evaluaremos el comportamiento del algoritmo IADEM-2 y el resto de algoritmos que se están usando en la comparación usando conjuntos de datos reales y de reducido tamaño tomados del repositorio de la UCI [New-1998]. Hemos usado 3 cuyos atributos son categóricos y otros 3 con atributos continuos. Aunque el algoritmo IADEM-2 y casi todos los demás que estamos usando para el estudio experimental están pensados para trabajar con grandes conjuntos de datos, la razón para hacer este tipo de experimentos es permitirnos evaluar cómo se comporta el algoritmo IADEM-2 ante conjuntos de datos reales para los que no ha sido diseñado expresamente. El muestreo con reemplazamiento que se utiliza para observar tantas experiencias como sean necesarias debería bastar para que el algoritmo indujese modelos de cierta calidad.

Empezaremos describiendo los 3 conjuntos de datos cuyos atributos son categóricos (hemos usado los nombres en inglés para facilitar su referencia al repositorio de la UCI):

- *Breast cancer*: este conjunto de datos ha sido aportado por el Instituto de Oncología del "University Medical Centre" en Ljubljana (Eslovenia) y agradecemos a M. Zwitter y M. Soklic su aportación. Dispone de 286 experiencias y está definido por 9 atributos categóricos con múltiples valores (pueden ser ordenados o no). El atributo de clase puede tomar 2 valores ("recurrencia" o "no recurrencia").

- *Mushroom*: los datos de este conjunto han sido tomados de "*The Audubon Society Field Guide to North American Mushrooms*" [Lin-1981] y en él se describen determinadas características físicas de diferentes hongos. Dispone de 8124 experiencias y está definido por 22 atributos categóricos con múltiples valores. El atributo de clase puede tomar 2 valores ("comestible" o "venenoso").

- *Nursery*: este conjunto de datos ha sido extraído a partir de un modelo de decisión jerárquico ([Boh-1990]) que fue diseñado para priorizar solicitudes en escuelas de enfermería. Dispone de 12960 experiencias y está definido por 8 atributos categóricos con múltiples valores. El atributo de clase puede tomar 5 valores ("no recomendado", "recomendado", "muy recomendado", "prioritario" y "especialmente prioritario").

En el cuadro 4.1 hemos presentado los datos correspondientes a las medias y desviaciones típicas de los experimentos realizados en cada una de las validaciones cruzadas. En él podemos destacar los siguientes aspectos:

- en cuanto a la precisión, observamos que los modelos más precisos son alcanzados o por el algoritmo J48 o por las variantes del algoritmo ITI, aunque no siempre son los más precisos al mismo tiempo para todos los conjuntos de datos (por ejemplo con el conjunto de datos *Breast cancer*, mientras J48 es el más preciso, las variantes del ITI son las menos precisas). Tiene sentido que estos algoritmos sean los que alcancen la mayor precisión, puesto que almacenan en memoria todas las experiencias y pueden realizar una inducción más exhaustiva. En lo que se refiere a los modelos inducidos por el algoritmo IADEM-2 se puede apreciar que suelen alcanzar precisiones cercanas a la de los mejores modelos, siendo el modelo para el conjunto de datos de *Nursery* el que presenta una precisión más alejada del mejor modelo. Comparando la precisión de los modelos inducidos por IADEM e IADEM-2 es necesario destacar que la precisión de este último siempre es mejor que la del primero, y normalmente de forma significativa.

- en lo que se refiere al tamaño de los árboles inducidos podemos apreciar que se dan tres situaciones diferentes (las tres que son posibles): cuando el algoritmo J48 induce árboles pequeños, IADEM-2

| Datos | Algoritmo | Precisión | | | Hojas | | | Experiencias (en miles) | Tiempo (en seg.) |
|---|---|---|---|---|---|---|---|---|---|
| Breast Cancer | J48 | 73,44 | ± 6,73 | | 9,80 | ± 7,08 | ⊕ | 0,257 | 0,02 |
| | ITI-total | 65,06 | ± 9,75 | ⊖ | 92,10 | ± 3,18 | ⊕ | 0,257 | 0,01 |
| | ITI-parcial | 64,34 | ± 11,35 | ⊖ | 88,80 | ± 4,71 | ⊕ | 0,257 | 0,00 |
| | VFDTc | 70,32 | ± 7,06 | | 1,00 | ± 0,00 | ⊕ | 0,257 | 0,00 |
| | IADEM | 70,64 | ± 9,66 | | 312,40 | ± 104,16 | ⊖ | 1241 | 13,50 |
| | IADEM-2 | 71,75 | ± 8,52 | | 216,70 | ± 66,43 | | 581 | 6,56 |
| Mushroom | | | | | | | | | |
| | J48 | 100,00 | ± 0,00 | ⊕ | 25,00 | ± 0,00 | ⊖ | 7,311 | 0,06 |
| | ITI-total | 100,00 | ± 0,00 | ⊕ | 12,00 | ± 0,00 | ⊕ | 7,311 | 0,00 |
| | ITI-parcial | 100,00 | ± 0,00 | ⊕ | 9,20 | ± 1,14 | ⊕ | 7,311 | 0,00 |
| | VFDTc | 99,41 | ± 0,28 | ⊖ | 17,00 | ± 0,00 | | 7,311 | 0,05 |
| | IADEM | 99,22 | ± 0,15 | ⊖ | 17,60 | ± 0,52 | ⊖ | 74 | 1,41 |
| | IADEM-2 | 99,89 | ± 0,12 | | 17,00 | ± 0,00 | | 79 | 1,30 |
| Nursery | | | | | | | | | |
| | J48 | 97,10 | ± 0,46 | ⊕ | 356,60 | ± 10,92 | ⊖ | 11,664 | 0,12 |
| | ITI-total | 99,68 | ± 0,18 | ⊕ | 273,10 | ± 2,28 | ⊖ | 11,664 | 0,00 |
| | ITI-parcial | 99,61 | ± 0,20 | ⊕ | 273,80 | ± 7,58 | ⊖ | 11,664 | 0,01 |
| | VFDTc | 87,96 | ± 0,60 | ⊖ | 19,00 | ± 0,00 | ⊕ | 11,664 | 0,06 |
| | IADEM | 90,66 | ± 0,57 | ⊖ | 48,20 | ± 4,54 | ⊖ | 399 | 6,24 |
| | IADEM-2 | 96,37 | ± 0,63 | | 40,10 | ± 9,85 | | 549 | 8,68 |

Cuadro 4.1: Valores medios y desviaciones típicas de las validaciones cruzadas para los conjuntos de datos *Breast cancer*, *Mushroom* y *Nursery*. El símbolo ⊕ indica que el resultado es significativamente mejor que el alcanzado por IADEM-2 y el símbolo ⊖ indica que es significativamente peor.

los genera grandes (*Breast cancer*), cuando J48 los induce grandes, IADEM-2 los genera pequeños (*Nursery*) y, por último, ambos algoritmos generan árboles de tamaño similar (*Mushroom*). Por tanto, las comparaciones de los modelos inducidos por IADEM-2 no son tan directas cuando las comparamos con J48. Algo similar ocurre cuando lo comparamos con los modelos inducidos por las variantes de ITI: algunas veces son mucho mayores, otras son similares y otras son mucho menores. Lo que sí es una constante es que los árboles más pequeños siempre los induce el algoritmo VFDTc y esto es debido al reducido tamaño de los conjuntos de datos. Al ser tan pequeños, sobre todo *Breast cancer*, el algoritmo no dispone de información suficiente para realizar ninguna expansión. Aún así, como se puede apreciar, las precisiones alcanzadas no son siempre las peores debido al uso de hojas funcionales. Antes de pasar a comentar el último punto, debemos señalar que los árboles inducidos por IADEM son en todos los casos, y de forma significativa, mayores que los inducidos por IADEM-2.

- debido a las diferentes maneras en que son usados los conjuntos de datos para inducir los modelos, el número de experiencias observadas por los algoritmos es también diferente. Se puede apreciar cómo el número de experiencias usadas por J48, las dos variantes de ITI y VFDTc es siempre el mismo: el total del conjunto de experiencias. Por su parte, IADEM e IADEM-2, como se comentó anteriormente, muestrean tantas experiencias como sea necesario hasta satisfacer los requisitos del usuario o hasta que se detecte estabilización en el aprendizaje. De ahí el mayor número de experiencias que deben usar antes de detener la inducción. Una consecuencia inmediata de esto es el mayor tiempo requerido por estos dos últimos algoritmos. Entre ellos, en principio, no podemos decir que ninguno sea el que menos experiencias consuma puesto que se van alternando.

Habiendo presentado los características de forma individual, podemos realizar el estudio comparativo agrupando las características más relevantes en índices combinados que nos resuman en un único valor la calidad de los modelos. En el cuadro 4.2 hemos recogido los índices combinados calculados con los métodos de ponderación aditiva simple ($SAW$) y de ponderación aditiva relativa ($RAW$) que se describieron en la subsección 4.1.3. La razón de usar los dos y no sólo uno ha sido justificar la necesidad de introducir el método relativo que mejora al simple. Recordemos que las características elegidas para calcular el índice combinado han sido la precisión, con un peso de $0,7$, y el tamaño del árbol, con un peso de $0,3$. Los pesos han sido asignados de esta forma porque, para nosotros, la precisión tiene más importancia que el tamaño del árbol.

| Datos | Algoritmo | SAW | | RAW | |
|---|---|---|---|---|---|
| | | $IC$ | Puesto | $IC$ | Puesto |
| Breast Cancer | | | | | |
| | J48 | 0,73 | 2º | 0,99 | 1º |
| | ITI-total | 0,62 | 5º | 0,27 | 5º |
| | ITI-parcial | 0,62 | 6º | 0,22 | 6º |
| | VFDTc | 0,97 | 1º | 0,76 | 2º |
| | IADEM | 0,67 | 4º | 0,48 | 4º |
| | IADEM-2 | 0,69 | 3º | 0,66 | 3º |
| Mushroom | | | | | |
| | J48 | 0,81 | 6º | 0,70 | 4º |
| | ITI-total | 0,93 | 2º | 0,95 | 2º |
| | ITI-parcial | 1,00 | 1º | 1,00 | 1º |
| | VFDTc | 0,86 | 4º | 0,32 | 5º |
| | IADEM | 0,85 | 5º | 0,14 | 6º |
| | IADEM-2 | 0,86 | 3º | 0,75 | 3º |
| Nursery | | | | | |
| | J48 | 0,70 | 6º | 0,55 | 4º |
| | ITI-total | 0,72 | 4º | 0,77 | 2º |
| | ITI-parcial | 0,72 | 5º | 0,77 | 3º |
| | VFDTc | 0,92 | 1º | 0,30 | 6º |
| | IADEM | 0,75 | 3º | 0,44 | 5º |
| | IADEM-2 | 0,82 | 2º | 0,78 | 1º |

Cuadro 4.2: Valores de los índices combinados ($IC$) calculados con los métodos *SAW* y *RAW* y su orden de preferencia para los conjuntos de datos *Breast cancer*, *Mushroom* y *Nursery*.

Observando los valores del cuadro 4.2 describiremos la principal diferencia que se puede apreciar entre los órdenes de preferencia asignados por los dos métodos. Esta diferencia radica en que los pesos ejerzan un papel más influyente y realmente sirvan para lo que fueron considerados. El primer ejemplo de un funcionamiento no deseado del método de ponderación aditivo simple ($SAW$) puede verse en el conjunto de datos *Breast cancer*. Aunque el algoritmo VFDTc induce un árbol de tamaño mínimo (1 hoja) con una precisión aceptable, no parece lógico que esté en el primer puesto en el orden de preferencia. Como hemos indicado, para nosotros lo más importante es la precisión (a la que damos un peso de $0,7$) y existe un algoritmo, J48, que induce un árbol algo más grande (con solo $8,8$ hojas de diferencia en media) pero bastante más preciso (más de un $3\,\%$). Parece más razonable que el algoritmo que ocupe el primer puesto para este problema en concreto sea J48 y no VFDTc. Este orden sí es asignado por el método de ponderación aditivo relativo ($RAW$). Otro caso en que se puede observar el mismo comportamiento es en el conjunto de datos *Nursery*. Tampoco parece razonable que, aunque el modelo inducido por VFDTc sea el más pequeño, sea el que se coloque en el primer puesto, ya que es el modelo menos preciso. Si es el modelo más pequeño y el menos

preciso, el valor de su índice combinado debería corresponderse con la importancia asignada por el usuario a los pesos. En este caso concreto, usando el método de ponderación aditiva relativa ($RAW$), su valor sería de $0,3$ ($0,0$ de $0,7$ posibles por ser el menos preciso y $0,3$ de $0,3$ posibles por ser el más pequeño). Por lo tanto, también en este caso parece que el método de ponderación aditiva relativa asigna un orden de preferencia más lógico.

Centrándonos en el orden asignado por los métodos empleados y dejando atrás los razonamientos que han motivado la inclusión de un nuevo método para la toma de decisiones con múltiples atributos (o características), pasamos a comentar los aspectos más relevantes. Como puede observarse, los modelos inducidos por el algoritmo IADEM-2 siempre se encuentran entre los tres primeros (incluso considerando el método de ponderación aditiva simple), estando también por encima del algoritmo IADEM, por lo que las ventajas de haber incorporado todas las aportaciones incluidas en el capítulo 3 parecen estar justificadas experimentalmente.

Una vez que hemos estudiado los conjuntos de datos con atributos categóricos, pasamos a estudiar aquellos que presentan atributos continuos para así poder evaluar la incorporación de esta característica en el algoritmo IADEM-2. Los conjuntos de datos empleados han sido los siguientes (usamos de nuevo sus nombres originales en inglés):

- *Balance scale*: este conjunto de datos fue generado para modelar los resultados de experimentos psicológicos [Sie-1976]. Dispone de $625$ experiencias y está definido por $4$ atributos continuos. El atributo de clase puede tomar $3$ valores ("izquierda", "equilibrado" o "derecha").

- *Ionosphere*: los datos que forman este conjunto proceden de los recogidos por un radar en Goose Bay, Labrador (Canadá) que medía los electrones libres en la ionosfera para detectar si existía alguna estructura o no [Sig-1989]. Dispone de $351$ experiencias y está definido por $34$ atributos continuos. El atributo de clase puede tomar $2$ valores ("bueno" o "malo").

- *Pima-indians diabetes*: este conjunto de datos procede del "National Institute of Diabetes and Digestive and Kidney Diseases" y está basado en un estudio de la diabetes en la tribu india llamada "*pima*" [Smi-1988]. Dispone de $768$ experiencias y está definido por $8$ atributos continuos. El atributo de clase puede tomar $2$ valores ("positivo" y "negativo").

En el cuadro 4.3 hemos presentado los datos correspondientes a las medias y desviaciones típicas de los experimentos realizados en cada una de las validaciones cruzadas. Antes de destacar los aspectos más relevantes debemos indicar que el algoritmo IADEM no es aplicable en estos casos puesto que no está diseñado para trabajar con atributos continuos. Existía la posibilidad de realizar una discretización estática previa, pero los resultados no serían comparables, así que hemos optado por no aplicar el algoritmo IADEM a estos conjuntos de datos. Los aspectos más destacados son:

- en cuanto a la precisión, observamos que los modelos más precisos ya no son siempre los inducidos por J48 o por las variantes de ITI. Los modelos más precisos son los inducidos por VFDTc a pesar de contar únicamente con una hoja. En estos casos, la predicción únicamente se realiza contando con la información almacenada en los árboles binarios que existen en la hoja y aplicando el clasificador ingenuo de Bayes a las particiones de igual anchura que se hagan para dichos atributos continuos. En lo que se refiere a los modelos inducidos por el algoritmo IADEM-2 se puede apreciar que ahora se dan casos en los que la precisión está próxima a los mejores valores (*Balance scale*) y se dan otros casos en los que está cercana a los peores (*Ionosphere*).

- en lo que se refiere al tamaño de los árboles inducidos podemos apreciar que los árboles inducidos por VFDTc e IADEM-2 son los más pequeños. Parece que la forma de realizar las divisiones dicotómicas empleada en estos dos algoritmos resulta más eficaz debido a que esperan hasta tener cierto soporte estadístico antes de expandir. Los modelos inducidos por J48 y las dos variantes de ITI son siempre significativamente mayores que los inducidos por IADEM-2.

| Datos | Algoritmo | Precisión | | Hojas | | Experiencias (en miles) | Tiempo (en seg.) |
|---|---|---|---|---|---|---|---|
| Balance Scale | J48 | 77,93 ± 4,08 | ⊖ | 39,30 ± 3,23 | ⊖ | 0,562 | 0,05 |
| | ITI-total | 76,64 ± 6,81 | ⊖ | 161,90 ± 6,47 | ⊖ | 0,562 | 0,00 |
| | ITI-parcial | 74,23 ± 4,75 | ⊖ | 159,90 ± 8,12 | ⊖ | 0,562 | 0,00 |
| | VFDTc | 91,67 ± 3,53 | ⊕ | 1,00 ± 0,00 | | 0,562 | 0,00 |
| | IADEM | No aplicable | | | | | |
| | IADEM-2 | 87,36 ± 4,49 | | 8,10 ± 9,37 | | 380 | 3,55 |
| Ionosphere | | | | | | | |
| | J48 | 88,30 ± 6,54 | | 13,60 ± 1,78 | ⊖ | 0,315 | 0,10 |
| | ITI-total | 87,18 ± 6,20 | | 23,40 ± 2,07 | ⊖ | 0,315 | 0,00 |
| | ITI-parcial | 89,46 ± 4,67 | | 22,20 ± 1,55 | ⊖ | 0,315 | 0,00 |
| | VFDTc | 89,44 ± 4,29 | | 1,00 ± 0,00 | | 0,315 | 0,02 |
| | IADEM | No aplicable | | | | | |
| | IADEM-2 | 87,73 ± 4,70 | | 1,00 ± 0,00 | | 238 | 8,62 |
| Pima-indians Diabetes | J48 | 73,70 ± 5,34 | | 21,50 ± 8,42 | ⊖ | 0,691 | 0,06 |
| | ITI-total | 72,92 ± 5,01 | | 133,50 ± 5,64 | ⊖ | 0,691 | 0,00 |
| | ITI-parcial | 68,23 ± 5,24 | ⊖ | 131,00 ± 6,60 | ⊖ | 0,691 | 0,00 |
| | VFDTc | 75,66 ± 3,73 | | 1,10 ± 0,32 | ⊕ | 0,691 | 0,00 |
| | IADEM | No aplicable | | | | | |
| | IADEM-2 | 72,92 ± 6,19 | | 8,10 ± 2,08 | | 428 | 5,17 |

Cuadro 4.3: Valores medios y desviaciones típicas de las validaciones cruzadas para los conjuntos de datos *Balance scale*, *Ionosphere* y *Pima-indians diabetes*. El símbolo ⊕ indica que el resultado es significativamente mejor que el alcanzado por IADEM-2 y el símbolo ⊖ indica que es significativamente peor.

- al igual que ocurría en los experimentos realizados con conjuntos de datos cuyos atributos eran categóricos, se pueden apreciar las mismas diferencias observadas entre IADEM-2 y el resto de algoritmos en lo que se refiere al número de experiencias observadas y al tiempo consumido. La razón sigue siendo la misma: IADEM-2 muestrea tantas experiencias como sea necesario hasta satisfacer los requisitos del usuario o hasta que se detecte estabilización en el aprendizaje.

Habiendo presentado los características de forma individual, podemos realizar el estudio comparativo agrupando las características más relevantes en índices combinados que nos resuman en un único valor la calidad de los modelos. En el cuadro 4.2 hemos vuelto a mostrar los índices combinados calculados con los métodos de ponderación aditiva simple ($SAW$) y de ponderación aditiva relativa ($RAW$). Recordemos que las características elegidas para calcular el índice combinado siguen siendo la precisión, con un peso de $0,7$, y el tamaño del árbol, con un peso de $0,3$.

No volveremos a estudiar tan detenidamente las razones para incluir el nuevo método de ponderación aditiva relativa ($RAW$), pero sí queremos indicar un nuevo ejemplo. En el caso del conjunto de datos *Ionosphere*, el método de ponderación aditiva simple ($SAW$) asigna el segundo puesto al algoritmo IADEM-2 cuando, debido a la precisión que presenta el modelo inducido por él, no debería ocupar un puesto tan alto. Es cierto que el árbol está compuesto por una única hoja, siendo el árbol más pequeño (junto con el árbol inducido por VFDTc), pero la característica más interesante sigue siendo la precisión y la precisión del modelo generado por IADEM-2 dista bastante de ser la mejor en este caso. Es justo que el segundo puesto lo ocupe ITI-parcial (como se deduce del índice combinado calculado con RAW) puesto que induce el modelo más preciso con un árbol no excesivamente grande.

Centrándonos en el orden asignado por los métodos empleados, pasamos a comentar los aspectos más

| Datos | Algoritmo | SAW | | RAW | |
|---|---|---|---|---|---|
| | | $IC$ | Puesto | $IC$ | Puesto |
| Balance | | | | | |
| Scale | J48 | 0,60 | 3º | 0,38 | 3º |
| | ITI-total | 0,59 | 4º | 0,10 | 4º |
| | ITI-parcial | 0,57 | 5º | 0,00 | 5º |
| | VFDTc | 1,00 | 1º | 1,00 | 1º |
| | IADEM | No aplicable | | | |
| | IADEM-2 | 0,70 | 2º | 0,81 | 2º |
| Ionosphere | | | | | |
| | J48 | 0,71 | 4º | 0,48 | 3º |
| | ITI-total | 0,69 | 5º | 0,00 | 5º |
| | ITI-parcial | 0,71 | 3º | 0,72 | 2º |
| | VFDTc | 1,00 | 1º | 1,00 | 1º |
| | IADEM | No aplicable | | | |
| | IADEM-2 | 0,99 | 2º | 0,47 | 4º |
| Pima-indians | | | | | |
| | J48 | 0,70 | 3º | 0,77 | 2º |
| | ITI-total | 0,68 | 4º | 0,44 | 4º |
| | ITI-parcial | 0,63 | 5º | 0,01 | 5º |
| | VFDTc | 1,00 | 1º | 1,00 | 1º |
| | IADEM | No aplicable | | | |
| | IADEM-2 | 0,72 | 2º | 0,73 | 3º |

Cuadro 4.4: Valores de los índices combinados ($IC$) calculados con los métodos *SAW* y *RAW* y su orden de preferencia para los conjuntos de datos *Balance scale*, *Ionosphere* y *Pima-indians diabetes*.

relevantes. Como puede observarse, el modelo inducido por el algoritmo IADEM-2 siempre se encuentra en los puestos intermedios (2º, 4º y 3º). Nunca destaca como el mejor, pero tampoco es nunca el peor.

Considerando lo expuesto en esta sección en la que hemos estudiado el comportamiento del algoritmo IADEM-2 con conjuntos de datos pequeños, para los que no fue expresamente diseñado el algoritmo, podemos concluir que los modelos inducidos tienen una calidad comparable con la ofrecida por otros algoritmos. La única desventaja que presenta frente a ellos es el tiempo consumido, pero esto es debido a la forma en que se toman las experiencias y se detiene el algoritmo.

## 4.5. Evaluación con grandes conjuntos de datos

Una vez hemos visto que el algoritmo IADEM-2 tiene un comportamiento aceptable cuando trabaja con conjuntos de datos de tamaño reducido, pasamos a estudiar su comportamiento con conjuntos de datos de gran tamaño, para lo que ha sido diseñado originalmente. Los conjuntos de datos que hemos usado han sido creados mediante generadores propios o mediante generadores disponibles en el repositorio de la UCI. Hemos incluido conjuntos de datos sintéticos creados por nosotros mismos para poder observar diferentes características que no era posible evaluar con los generadores disponibles en el repositorio de la UCI. Todos los conjuntos generados por nosotros reciben el nombre de "*Sint-*$x$" donde la $x$ es un número y han sido generados a partir de árboles de decisión creados de forma aleatoria. En primer lugar definíamos una descripción del problema (con tantos atributos, valores y clases como deseásemos) y posteriormente se generaba, aleatoriamente, un árbol con una profundidad máxima en la que los nodos eran etiquetados con atributos seleccionados también de forma aleatoria. A continuación damos los detalles de los conjuntos de

datos empleados:

- *Sint-1*: el problema definido antes de crear el árbol tiene 20 atributos con múltiples valores (entre 5 y 7) y 4 valores para el atributo de clase. El árbol generado aleatoriamente tiene una profundidad máxima de 5 niveles y un total de 398 hojas. El conjunto de datos ha sido generado sin incluirle ruido y su tamaño es de 1 millón de experiencias.

- *Sint-2*: el problema definido antes de crear el árbol tiene también 20 atributos con múltiples valores (entre 3 y 5) y 4 valores para el atributo de clase. El árbol generado aleatoriamente tiene una profundidad máxima de 5 niveles y un total de 100 hojas. El conjunto de datos ha sido generado sin incluirle ruido y tiene un tamaño de 100000 experiencias (100K). La peculiaridad de este conjunto de datos es que se ha generado evitando que las clases estuviesen balanceadas, en concreto, la cuarta clase sólo aparece alrededor del 3 % de las veces (cuando le correspondería un 25 %).

- *Sint-3*: el problema definido antes de crear el árbol tiene también 20 atributos con múltiples valores (entre 4 y 6) y 4 valores para el atributo de clase. El árbol generado aleatoriamente tiene una profundidad máxima de 7 niveles y un total de 663 hojas. El conjunto de datos ha sido generado incluyéndole un 15 % de ruido y su tamaño es de 1 millón de experiencias. Este conjunto de datos representa un conocimiento bastante más complejo que ninguno de los anteriores debido al gran tamaño del árbol y al ruido introducido.

- *LED*: este conjunto de datos se describe en el libro "*Classification and Regression Trees*" [Bre-1984] (entre las páginas 43 y 49) y ya lo hemos usado anteriormente en las secciones 4.2 y 4.3. Como se indicó anteriormente, el generador de este conjunto de datos recibe el nombre de "*LED*" debido a que las experiencias reflejan el estado de un display de 7 segmentos (LEDs). Se define mediante 24 atributos binarios (con valores 0 ó 1) de los cuales 17 son irrelevantes y los 7 restantes son los que se usan para determinar la clase, que puede tomar 10 valores diferentes (los 10 dígitos diferentes que presenta un display de 7 segmentos). Hemos preparado un conjunto de datos con 1 millón de experiencias y con un 10 % de ruido usando el generador disponible en el repositorio de la UCI.

- *Waveform*: este conjunto de datos también se describe en el libro "*Classification and Regression Trees*" [Bre-1984] (entre las páginas 49 y 55). El generador crea experiencias con 40 atributos continuos de los cuales 19 son irrelevantes. La clase puede tomar 3 valores, cada uno correspondiente a uno de los 3 tipos de onda creados. Usando el generador disponible en el repositorio de la UCI hemos creado un conjunto de datos con 1 millón de experiencias. En la creación de este conjunto de datos se introduce ruido.

En el cuadro 4.5 hemos presentado los datos correspondientes a las medias y desviaciones típicas de los experimentos realizados en cada una de las validaciones cruzadas. Existen algunos algoritmos cuya ejecución no terminaba correctamente debido a las limitaciones impuestas al tamaño máximo de memoria disponible. Para permitir que induzcan modelos y así poder comparar los resultados hemos ampliado dichos límites y los hemos indicado en el cuadro marcándolos con asteriscos ('*'). Los algoritmos han sido el J48 en los casos en los que había ruido y VFDTc en el caso del conjunto de datos *Waveform*. Para J48 hemos aumentado directamente la memoria hasta el máximo disponible (2GB) y entonces ha podido concluir su ejecución. Para VFDTc, como suponíamos que el límite no estaría lejos del límite anterior (512MB), lo hemos aumentado gradualmente hasta conseguir que la ejecución se realizase correctamente. El límite usado por VFDTc para el conjunto de datos *Waveform* ha sido de 605MB. Las ejecuciones de las variantes de ITI con los conjuntos de datos en los que había ruido no concluían ni ampliando al máximo el límite de memoria (2GB) y por eso no disponemos de datos para ellas.

Los aspectos más destacados que se pueden observar en el cuadro 4.5 son los siguientes:

- observando los conjuntos de datos sin ruido:

| Datos | Algoritmo | Precisión | | Hojas | | Experiencias (en miles) | Tiempo (en seg.) |
|---|---|---|---|---|---|---|---|
| Sint-1 | | | | | | | |
| 0 % ruido | J48 | 100,00 ± 0,00 | ⊕ | 398,00 ± 0,00 | ⊖ | 900 | 27,26 |
| | ITI-total | 99,66 ± 0,03 | ⊖ | 6461,80 ± 250,77 | ⊖ | 900 | 6523,28 |
| | ITI-parcial | 99,98 ± 0,02 | ⊖ | 1914,50 ± 809,52 | ⊖ | 900 | 5983,93 |
| | VFDTc | 99,79 ± 0,05 | ⊖ | 468,60 ± 7,59 | ⊖ | 900 | 18,78 |
| | IADEM | 99,79 ± 0,05 | ⊖ | 426,90 ± 34,80 | ⊖ | 2261 | 73,66 |
| | IADEM-2 | 99,997 ± 0,00 | | 394,00 ± 0,00 | | 834 | 28,77 |
| Sint-2 | | | | | | | |
| 0 % ruido | J48 | 100,00 ± 0,00 | ⊕ | 104,50 ± 14,23 | ⊕ | 90 | 1,80 |
| desbalanc. | ITI-total | 99,75 ± 0,06 | ⊖ | 615,00 ± 13,86 | ⊖ | 90 | 205,72 |
| | ITI-parcial | 99,99 ± 0,02 | ⊕ | 235,70 ± 35,83 | | 90 | 58,06 |
| | VFDTc | 94,48 ± 6,11 | ⊖ | 190,40 ± 52,17 | ⊕ | 90 | 1,70 |
| | IADEM | 99,84 ± 0,04 | | 169,50 ± 75,37 | ⊕ | 895 | 29,00 |
| | IADEM-2 | 99,86 ± 0,04 | | 241,00 ± 0,00 | | 817 | 26,83 |
| Sint-3 | | | | | | | |
| 15 % ruido | J48** | 84,09 ± 0,10 | ⊕ | 43460,60 ± 656,92 | ⊖ | 900 | 54,28 |
| | ITI-total | Sin memoria | | | | | |
| | ITI-parcial | Sin memoria | | | | | |
| | VFDTc | 70,25 ± 6,38 | ⊖ | 1528,80 ± 325,61 | ⊖ | 900 | 23,58 |
| | IADEM | 80,02 ± 2,97 | | 719,60 ± 127,95 | ⊖ | 6208 | 241,01 |
| | IADEM-2 | 80,58 ± 2,96 | | 259,50 ± 105,92 | | 1319 | 51,49 |
| LED | | | | | | | |
| 10 % ruido | J48** | 73,56 ± 0,13 | ⊖ | 29997,90 ± 129,40 | ⊖ | 900 | 113,43 |
| | ITI-total | Sin memoria | | | | | |
| | ITI-parcial | Sin memoria | | | | | |
| | VFDTc | 74,00 ± 0,11 | ⊖ | 71,50 ± 1,35 | ⊖ | 900 | 13,84 |
| | IADEM | 70,01 ± 0,12 | ⊖ | 15,40 ± 1,58 | ⊖ | 268 | 4,07 |
| | IADEM-2 | 74,19 ± 0,13 | | 12,40 ± 1,43 | | 848 | 12,56 |
| Waveform | | | | | | | |
| con ruido | J48** | 80,86 ± 0,04 | ⊖ | 40457,20 ± 197,85 | ⊖ | 900 | 6888,85 |
| | ITI-total | Sin memoria | | | | | |
| | ITI-parcial | Sin memoria | | | | | |
| | VFDTc* | 82,73 ± 0,10 | ⊕ | 245,80 ± 2,86 | ⊖ | 900 | 158,97 |
| | IADEM | No aplicable | | | | | |
| | IADEM-2 | 82,13 ± 0,41 | | 32,00 ± 6,45 | | 1051 | 341,17 |

Cuadro 4.5: Valores medios y desviaciones típicas de las validaciones cruzadas para los conjuntos de datos *Sint-1*, *Sint-2*, *Sint-3*, *LED* y *Waveform*. El símbolo ⊕ indica que el resultado es significativamente mejor que el alcanzado por IADEM-2 y el símbolo ⊖ indica que es significativamente peor. Los algoritmos marcados con asteriscos han necesitado un incremento en el límite de la memoria disponible para poder ejecutarse (VFDTc* ha necesitado 605MB y J48** ha podido ser ejecutado tras aumentar el límite hasta 2GB).

- el único algoritmo que es capaz de inducir un modelo exactamente igual al que se usó para generar el conjunto de datos *Sint-1* es J48. Su modelo tiene 398 hojas (igual que el original) y no se equivoca en las predicciones. De igual forma, este algoritmo es el que induce el modelo más similar al usado para crear el conjunto de datos *Sint-2*, ya que consigue una precisión del 100 % con tan sólo 104, 5 hojas. Debemos recordar que este último conjunto de datos, además

no está balanceado, lo que complica para algunos algoritmos la inducción del modelo. Además de los resultados tan buenos observados para el algoritmo J48 podemos destacar que el algoritmo IADEM-2 consiguió inducir un árbol similar al original para el conjunto de datos *Sint-1* pero que detuvo la inducción al satisfacer los requisitos del usuario (recordemos que $\varepsilon = 0,01$) sin llegar a sobrepasar el límite de las $398$ hojas del modelo generador.

- al considerar el conjunto de datos *Sint-2*, que no está balanceado, vemos que la precisión alcanzada por casi todos los modelos es cercana al $100\,\%$, excepto para VFDTc. En este punto queremos destacar el uso que se le puede dar a los algoritmos IADEM e IADEM-2 a la hora de extraer conocimiento de conjuntos de datos sin ruido, pero cuya proporción de clases no esté balanceada. Debido a que el usuario puede exigir un nivel máximo de error tolerado ($\varepsilon$), el algoritmo seguirá muestreando experiencias hasta alcanzarlo (excepto que se estabilice el aprendizaje) y esto hace ideal a este algoritmo para tratar problemas desbalanceados para los que otros algoritmos incrementales podrían no dar la solución requerida.
- el número de experiencias usadas antes de inducir el modelo sigue siendo superior, por lo general, al usar IADEM o IADEM-2, pero nos gustaría destacar un aspecto. Cuando el número de experiencias es muy superior, como ocurre en el conjunto de datos *Sint-2*, el tiempo consumido también es superior; sin embargo, cuando el número de experiencias es comparable, por ejemplo para *Sint-1*, el tiempo consumido por IADEM-2 puede ser similar al consumido por J48.

- observando los conjuntos de datos con ruido:

  - los árboles inducidos por J48 son considerablemente mayores que el resto, pero no es de extrañar dado que este algoritmo no está preparado para trabajar con grandes conjuntos de datos. Los árboles inducidos por VFDTc también son bastante más grandes que los inducidos por IADEM-2, siendo las diferencias estadísticamente significativas. Los árboles más pequeños han sido siempre los inducidos por IADEM-2 sin que por ello presenten un mal comportamiento en el apartado de precisión puesto que es la opción que menos error comete o está cerca de serlo.
  - el número de experiencias que necesita el algoritmo IADEM-2 sigue dependiendo de la dificultad de los conceptos a aprender y no del tamaño del conjunto de datos. Es por eso que, para diferentes problemas, se observan diferentes comportamientos manteniendo un consumo de tiempo factible en todos los casos.

Habiendo presentado los características de forma individual, podemos realizar el estudio comparativo agrupando las características más relevantes en índices combinados que nos resuman en un único valor la calidad de los modelos. En el cuadro 4.6 hemos vuelto a mostrar los índices combinados calculados con los métodos de ponderación aditiva simple ($SAW$) y de ponderación aditiva relativa ($RAW$). Recordemos que las características elegidas para calcular el índice combinado siguen siendo la precisión, con un peso de $0,7$, y el tamaño del árbol, con un peso de $0,3$.

El principal cambio de orden observado entre los métodos de ponderación aditiva simple ($SAW$) y relativa ($RAW$) se produce en el conjunto de datos *Sint-1* entre J48 e IADEM-2. La razón para que J48 sea el primero en este caso parece lógica: aunque hay una diferencia de $4$ hojas y únicamente de $0,003\,\%$ en el porcentaje de precisión, es razonable que J48 ocupe el primer lugar, puesto que estamos valorando la precisión como la característica más importante.

Centrándonos en el orden asignado por los métodos empleados, pasamos a comentar los aspectos más relevantes. Como puede observarse, los modelos inducidos por el algoritmo IADEM-2 normalmente se encuentran entre los primeros puestos (1º ó 2º) estando su índice combinado cerca del mejor cuando no lo es él. Tan sólo alcanza un puesto intermedio cuando se trata del conjunto de datos *Sint-2*, el conjunto de datos desbalanceado, pero su resultado está muy cerca del mejor.

Considerando lo expuesto en esta sección en la que hemos estudiado el comportamiento del algoritmo IADEM-2 con conjuntos de datos grandes, para cuyo estudio fue diseñado el algoritmo, podemos concluir que los modelos inducidos por él son de una gran calidad.

| Datos | Algoritmo | SAW | | RAW | |
|---|---|---|---|---|---|
| | | $IC$ | Puesto | $IC$ | Puesto |
| Sint-1 | | | | | |
| 0 % ruido | J48 | 1,00 | 2º | 1,00 | 1º |
| | ITI-total | 0,72 | 6º | 0,00 | 6º |
| | ITI-parcial | 0,76 | 5º | 0,88 | 3º |
| | VFDTc | 0,95 | 4º | 0,56 | 5º |
| | IADEM | 0,98 | 3º | 0,57 | 4º |
| | IADEM-2 | 1,00 | 1º | 0,99 | 2º |
| Sint-2 | | | | | |
| 0 % ruido | J48 | 1,00 | 1º | 1,00 | 1º |
| desbalanc. | ITI-total | 0,75 | 6º | 0,67 | 5º |
| | ITI-parcial | 0,83 | 3º | 0,92 | 3º |
| | VFDTc | 0,83 | 5º | 0,25 | 6º |
| | IADEM | 0,88 | 2º | 0,94 | 2º |
| | IADEM-2 | 0,83 | 4º | 0,90 | 4º |
| Sint-3 | | | | | |
| 15 % ruido | J48** | 0,70 | 3º | 0,70 | 3º |
| | ITI-total | Sin memoria | | | |
| | ITI-parcial | Sin memoria | | | |
| | VFDTc | 0,64 | 4º | 0,29 | 4º |
| | IADEM | 0,77 | 2º | 0,79 | 2º |
| | IADEM-2 | 0,97 | 1º | 0,82 | 1º |
| LED | | | | | |
| 10 % ruido | J48** | 0,69 | 4º | 0,59 | 3º |
| | ITI-total | Sin memoria | | | |
| | ITI-parcial | Sin memoria | | | |
| | VFDTc | 0,75 | 3º | 0,97 | 2º |
| | IADEM | 0,90 | 2º | 0,30 | 4º |
| | IADEM-2 | 1,00 | 1º | 1,00 | 1º |
| Waveform | | | | | |
| con ruido | J48** | 0,68 | 3º | 0,00 | 3º |
| | ITI-total | Sin memoria | | | |
| | ITI-parcial | Sin memoria | | | |
| | VFDTc* | 0,74 | 2º | 1,00 | 1º |
| | IADEM | No aplicable | | | |
| | IADEM-2 | 0,99 | 1º | 0,78 | 2º |

Cuadro 4.6: Valores de los índices combinados ($IC$) calculados con los métodos *SAW* y *RAW* y su orden de preferencia para los conjuntos de datos *Sint-1*, *Sint-2*, *Sint-3*, *LED* y *Waveform*. Los algoritmos marcados con asteriscos han necesitado un incremento en el límite de la memoria disponible para poder ejecutarse (VFDTc* ha necesitado 605MB y J48** ha podido ser ejecutado tras aumentar el límite hasta 2GB).

## 4.6. Conclusiones derivadas del estudio experimental

En esta sección recogeremos las características más relevantes que pueden atribuirse al algoritmo IADEM-2. En primer lugar veremos para qué tipo de problemas está especialmente indicado su uso y a continuación veremos las dificultades que se pueden presentar y que pueden hacer que su desempeño no sea el deseado.

### 4.6.1. Ventajas derivadas del uso de IADEM-2

A lo largo de todo este capítulo hemos venido indicando las ventajas que se apreciaban en el algoritmo IADEM-2, pero aprovechamos ahora para esquematizarlas en los siguientes puntos:

- Precisión elevada: en todos los experimentos que hemos presentado se puede comprobar cómo los valores de la precisión alcanzada por los modelos inducidos con el algoritmo IADEM-2 son los mejores o se encuentran cerca de serlo. Sólo en algunos casos concretos (principalmente en conjuntos de datos pequeños) la precisión aparece algo más alejada del mejor resultado, pero en ningún experimento su valor ha sido el peor.

- Mayor interpretabilidad del modelo: los árboles de decisión inducidos por IADEM-2 son bastante más pequeños que el resto de árboles de decisión inducidos por otros algoritmos. Tan solo en algunos experimentos se inducen árboles más grandes, pero son minoritarios. Al ser los árboles de decisión más pequeños, y presentar menos hojas, los expertos tendrán más facilidad de interpretar el conocimiento recogido en ellos.

- Independencia del tamaño del conjunto de datos: el número total de experiencias muestreadas por el algoritmo IADEM-2 no depende del tamaño del conjunto de datos, sino de la complejidad del conocimiento que reside en él y de las características requeridas por el usuario. De esta forma es posible concluir la inducción del árbol de decisión con menos experiencias que las que usarían otros algoritmos. Este aspecto, como veremos a continuación, también tiene su contrapartida en forma de limitación.

- Tratamiento de conjuntos de datos con desbalanceo en la proporción de clases: la posibilidad que se le brinda al usuario para limitar el error máximo tolerado en el modelo que se induzca puede servir para forzar al algoritmo IADEM-2 a encontrar soluciones que otros algoritmos posiblemente no encontrasen. Si para detener la inducción el algoritmo IADEM-2, éste se ve forzado a alcanzar una precisión mínima y, para alcanzarla, debe expandir hasta las hojas en las que hay un desbalanceo en la proporción de clases, lo hará, aunque, a priori, no aportarían demasiado al modelo.

### 4.6.2. Limitaciones actuales en el uso de IADEM-2

Para terminar el capítulo recogemos también las características que pueden hacer que el algoritmo se comporte de forma irregular y para las que no ha sido diseñado. Algunas de ellas, como comentaremos en el último capítulo de esta parte de la tesis, serán las futuras líneas de investigación. Las limitaciones actuales en el uso de IADEM-2 pueden provenir de:

- Independencia del tamaño del conjunto de datos: ya comentamos este aspecto como una ventaja cuando el algoritmo IADEM-2 necesita muestrear menos experiencias que el tamaño total del conjunto de datos. La contrapartida aparece cuando ese número de experiencias es mayor. Esto suele ocurrir cuando los conjuntos de datos son pequeños, cuyo tratamiento no está considerado entre las tareas del algoritmo que hemos diseñado. Aún así, a pesar del mayor coste computacional que tiene intentar extraer conocimiento de este tipo de conjunto de datos, los resultados nos indican que los modelos que se induzcan tendrán, por lo general, una calidad similar a la alcanzada con otros algoritmos.

- Alta dimensionalidad del problema a resolver: cuando el problema que se intente resolver presente un gran número de atributos o el número de valores que puedan tomar sea elevado, el algoritmo IADEM-2 puede que ocupe toda la memoria disponible y no sea capaz de finalizar la inducción. La razón no es más que la estructura propia del algoritmo que almacena contadores para todos los valores de todos los atributos en todas las hojas del árbol. Cuando cualquiera de éstos sea excesivamente grande, los recursos disponibles no serán suficientes. Este caso no se ha producido en ninguno de los experimentos que hemos realizado, pero su aparición es factible. Hemos de indicar que esta es una de las líneas futuras de trabajo que contemplamos y que comentaremos en el siguiente capítulo.

- Presencia de atributos con valores desconocidos: una característica que no ha sido abordada directamente por el algoritmo IADEM-2 es el tratamiento de valores desconocidos que pudiesen aparecer en los diferentes atributos que definen los conjuntos de problemas. La única forma disponible, hasta el momento, en el algoritmo IADEM-2 es la de añadir un nuevo valor al atributo que presenta valores desconocido y asignar ese nuevo valor a todos los valores desconocidos del atributo en cuestión. Pero este enfoque es demasiado básico y no aprovecha toda la información disponible como pueden hacer otro métodos [For-2006]. Entre nuestras futuras líneas de investigación hemos contemplado la incorporación de métodos más sofisticados que intenten paliar la pérdida de información originada por los valores desconocidos en determinados atributos.

- Presencia de cambio de concepto en el conjunto de datos: una de las dificultades que siempre ha estado latente y que últimamente está empezando a ser tratada desde diversas perspectivas es el cambio de concepto [Wid-1996; Gam-2006]. El algoritmo IADEM-2 no es capaz de tratar el cambio de concepto de forma explícita. La única forma en que se puede recoger sería en los contadores de las hojas virtuales y cuando la información antigua se pierda al realizar una expansión, pero la estructura inicial no cambiaría y los contadores tardarían demasiado tiempo en recoger el cambio de concepto (hasta que el nuevo concepto haya sido observado durante más tiempo que el anterior). Hemos de señalar que en esta línea también están dirigidos nuestros futuros trabajos de investigación.

# Capítulo 5

# Resumen de la parte I
# Summary of part I

El objetivo de esta parte de la tesis ha sido presentar un nuevo algoritmo incremental basado en cotas de concentración al que hemos llamado IADEM-2. Hemos introducido sus antecedentes (capítulos 1 y 2), hemos descrito las aportaciones introducidas (capítulo 3) y hemos realizado un exhaustivo estudio experimental para evaluar su comportamiento (capítulo 4). A continuación presentamos un breve resumen de todos los temas tratados en esta parte de la tesis.

Las cotas de concentración [Dia-2001; Jan-2002; Chu-2006] constituyen uno de los enfoques más extendidos para determinar el intervalo en el que debe encontrarse el valor real de una determinada variable. Las cotas de concentración ofrecen márgenes de error estimados entre los que debe encontrarse el valor real de la variable que se estudia teniendo en cuenta su valor medio (estará entre 0 y 1) y el número de muestras que se han usado para calcular dicho valor medio. Además, el cálculo de los márgenes de error se realiza considerando una *confianza* mínima $(1 - \delta)$. En el capítulo 1 hemos estudiado dos de las cotas más utilizadas en el área de aprendizaje computacional: la cota de Chernoff [Che-1952] y la de Hoeffding [Hoe-1963]. Después de expresarlas de una forma más uniforme que hiciese más sencillo su uso, hemos estudiado los beneficios que se pueden esperar cuando son combinadas. El margen de error ofrecido por ambas cotas es diferente y, dependiendo de los valores usados en el cálculo, el menor margen de error será ofrecido por una cota o por la otra. En el estudio que hemos realizado se puede concluir que la estimación más ajustada para el margen de error la ofrece la cota de Chernoff cuando el valor medido (usando una muestra), cuyo margen de error se quiere calcular, es menor que $1/6$. Por contra, cuando ese valor medido es mayor que $1/6$, la cota de Hoeffding es la que ofrece el margen de error más ajustado.

Para extraer conocimiento de grandes volúmenes de datos, cada vez más frecuentes, existen diversos enfoques, pero el "*muestreo*" [Fel-1968] se ha revelado como uno de los más empleados para abordar dicha tarea. Existen multitud de tipos de muestreo, desde el muestreo "*estratificado*" [Cat-1991] hasta el muestreo "*activo*" [Saa-2004], pasando por el "*doble*" muestreo [Cox-1952] y otros. El tipo de muestreo en el que estamos interesados se conoce como muestreo "*secuencial*" [Gav-2001], "*dinámico*" [Joh-1996] o "*adaptativo*" [Lip-1990]. Al usar este tipo de muestreo, el tamaño de la muestra no se fija a priori, sino que se toman muestras del conjunto de datos de forma iterativa. Con cada nueva muestra se evalúan los criterios que deben ser satisfechos y, cuando sean alcanzados, la ejecución se detiene habiendo logrado satisfacer dichos requisitos. El algoritmo que presentamos usa el enfoque de muestreo secuencial (adaptativo o dinámico) aprovechando sus beneficios y tomando (por muestreo con reemplazamiento) la cantidad de experiencias necesarias hasta alcanzar un modelo con la calidad mínima requerida por el usuario.

Para el desarrollo del algoritmo IADEM-2 hemos partido del algoritmo IADEM-0 que fue desarrollado, en su versión inicial, por Ramos y Morales [Ram-2000; Ram-2001]. La implementación inicial propuesta para ese algoritmo ha sido sustituida por una definición general del funcionamiento del mismo [Cam-2002], abstrayendo los procedimientos explícitos previos y dotando de total libertad al diseñador a la hora de implementar cualquier versión. El proceso de abstracción llevado a cabo ha permitido mejorar el algoritmo,

contemplando casos extremos que anteriormente no fueron considerados y refinando los cálculos bajo diversas condiciones. El algoritmo IADEM-0 representa el conocimiento extraído usando árboles de decisión que son inducidos considerando las hojas reales del propio árbol y un conjunto de hojas virtuales que recopilan información un nivel más allá de la frontera del árbol. Una de las principales características de este algoritmo es que, en todo momento, se tiene una estimación del error que comete el modelo en el peor y en el mejor caso. Dicha estimación resulta útil porque servirá para detener la inducción del árbol cuando satisfaga los requisitos del usuario. De esta forma, el usuario puede indicarle al algoritmo el nivel máximo de error que permite que se produzca en el modelo y la confianza en esa estimación. La utilidad de permitir que el usuario exija el nivel de error máximo radica en que se puedan obtener modelos más sencillos que sean fácilmente interpretables. Otra característica que encontramos en el algoritmo IADEM-0 es que el vector de predicción que se calcula, cuando se desea conocer la clase de una observación, no es la única información ofrecida. Al vector con los valores estimados de la predicción le acompañan otros dos vectores que delimitan el intervalo en el que se encontrará el vector de predicción con una confianza previamente fijada.

Para completar el desarrollo del algoritmo, se ha realizado un estudio intenso sobre los valores que deben tomar los diferentes parámetros internos. Distintas características del algoritmo se pueden configurar pero requieren de cierto dominio por parte del usuario. Para hacer más fácil el manejo del algoritmo, esos parámetros reciben una serie de valores por defecto que son eficaces en la mayoría de las situaciones, liberando al usuario de tener que asignar valores a argumentos cuyo significado puede no conocer en detalle. Estos mismos valores son los que se han usado en la configuración para realizar el estudio experimental. Hemos incorporado un detallado estudio de cómo afectan dichos parámetros al comportamiento del algoritmo y hemos asignado los valores por defecto que mejores resultados dan para la mayor parte de las circunstancias. Al estudio de la mejor combinación de parámetros hemos incorporado un análisis básico sobre la complejidad temporal y espacial del algoritmo.

A pesar de los buenos resultados que se alcanzan con el algoritmo IADEM-0, carece de una serie de características que limitan su uso: su funcionamiento puede no ser adecuado cuando trabaja con conjuntos de datos en los que haya ruido, no es capaz de procesar directamente experiencias con atributos continuos (precisándose una discretización estática previa), y no usa toda la información de que dispone para realizar predicciones más fiables. Estas limitaciones han sido superadas y otras mejoras han sido incorporadas en el diseño del algoritmo IADEM-2. Un paso intermedio entre IADEM-0 e IADEM-2 fue el desarrollo del algoritmo IADEM [Ram-2006a] y del algoritmo IADEMc [Cam-2006] en el que se solucionan parcialmente los defectos. El algoritmo que resuelve los problemas antes descritos y mejora sustancialmente el proceso de expansión ha sido llamado IADEM-2 [Cam-2007] y es el núcleo de esta parte de la tesis.

El algoritmo IADEM-2 mejora la convergencia, incrementa la calidad de los modelos inducidos y aumenta la precisión. Para mejorar la convergencia del algoritmo se ha modificado la forma en que se expanden las hojas y, sobre todo, se ha incluido un criterio para detectar estabilizaciones o empeoramientos en el aprendizaje simplificando el método geométrico presentado por Castillo [Cas-2006]. La calidad de los modelos inducidos se ha visto mejorada por un nuevo mecanismo que evalúa la idoneidad del mejor atributo para ser expandido, por la incorporación de técnicas que permitan procesar experiencias con atributos continuos de forma directa (adaptando la propuesta por Gama, Rocha y Medas [Gam-2003]) y por el establecimiento de un desbalanceo máximo antes de poder expandir una hoja. La precisión del modelo ha podido ser aumentada gracias a información de la que ya se disponía en el árbol de decisión pero que no era usada: básicamente consiste en generar un clasificador ingenuo de Bayes en cada hoja con la información que ya existía en ellas como se describe en la propuesta de Gama, Rocha y Medas [Gam-2003].

Cada una de las mejoras que se ha introducido también dispone de parámetros que sirven para variar su funcionamiento. Al mismo tiempo que se iban introduciendo las mejoras, hemos mostrado cómo la modificación de los parámetros correspondientes afecta al algoritmo. Igual que para el algoritmo IADEM-0 se hizo un estudio de las repercusiones que tenía cada parámetro y se determinó qué valor podía ser el más adecuado para ser usado en el mayor número de casos, para el algoritmo IADEM-2 también hemos realizado un estudio similar. Además del estudio de los parámetros, hemos incluido, para cada aportación, un análisis de la complejidad temporal y espacial que supone añadir cada una de ellas.

Para terminar esta parte de la tesis hemos llevado a cabo un extenso estudio experimental con idea de mostrar cómo se comporta el nuevo algoritmo que proponemos en el mayor número de situaciones posible. Como complemento al estudio individual de las características relevantes de los modelos que se iban a inducir, hemos incorporado un nuevo método en el campo de la toma de decisiones para problemas de múltiples características (sus siglas en inglés son *MADM* [Hwa-1981]) que hemos llamado *ponderación aditiva relativa*. Su función es resumir, en un único índice, las características relevantes que suelen ser consideradas de forma aislada e independiente.

Hemos comprobado que el comportamiento del algoritmo para conjuntos de datos de diferentes tamaños y con diferentes niveles de ruido es muy satisfactoria. De ese estudio merece ser destacado la estabilidad observada en cuanto a la precisión alcanzada, el tamaño de los árboles inducidos y el número de experiencias que necesitan ser muestreadas antes de concluir la inducción. El algoritmo se adapta a la complejidad del conocimiento que está extrayendo del conjunto de datos y actúa en consecuencia, muestreando más experiencias cuanto más complejo sea el conocimiento.

En cuanto a las pruebas realizadas con pequeños conjuntos de datos reales (tomados del repositorio de la UCI) hemos de comentar que los modelos inducidos por IADEM-2 son comparables con los modelos inducidos por otros algoritmos, aunque el tiempo requerido para su inducción se incrementa mucho. Esta demora es lógica ya que el algoritmo no ha sido diseñado para tratar con conjuntos de datos pequeños y para inducir el modelo necesita muestrear tantas experiencias como sea preciso hasta que haya evidencias estadísticas de su conveniencia. En lo que se refiere a los resultados obtenidos al tratar con grandes conjuntos de datos (creados usando generadores tomados del repositorio de la UCI y otros desarrollados por nosotros mismos) hemos de indicar que los resultados son bastante positivos. Los modelos inducidos por IADEM-2 son los más simples, lo que repercute en una mejor interpretabilidad, y alcanzan las mejores cotas de precisión o están cercas de ellas.

Para terminar este resumen debemos mencionar que se ha hecho una valoración objetiva de las ventajas y limitaciones del algoritmo. Como ventajas podemos destacar las siguientes: se consigue una precisión elevada, los modelos son más fácilmente interpretables (porque son más pequeños), es independiente del tamaño del conjunto de datos y puede tratar conjuntos de datos con desbalanceo en la proporción de clases. Como limitaciones hemos de considerar las siguientes: la independencia del tamaño del conjunto de datos (que puede verse también como una limitación si el conjunto es demasiado pequeño), la alta dimensionalidad del problema a resolver, la presencia de atributos con valores desconocidos y de cambio de concepto en el conjunto de datos. La alta dimensionalidad del problema podría ser abordada de distintas formas según lo que queramos reducir. Se podrían filtrar los atributos menos relevantes utilizando técnicas de selección de características [Rui-2005a]. Si lo que se pretende es reducir el tamaño de los árboles binarios balanceados que contienen la información de los atributos continuos, se podría fijar un número máximo de valores diferentes o una precisión máxima. Si lo que se pretende es reducir el número de hojas reales que almacenen información, se podría optar por un enfoque similar al propuesto por Domingos y Hulten [Dom-2000] en el que las hojas menos prometedores son apartadas del proceso, al menos de forma temporal. En cuanto al tratamiento de valores desconocidos que puedan aparecer en diferentes atributos, estudiaremos la conveniencia de modificar los criterios de expansión de forma que se puedan asignar valores concretos adaptando el enfoque presentado por Fortes, Mora, Morales y Triguero [For-2006]. Para tratar el cambio de concepto existen también multitud de enfoques, pero nuestros primeros pasos los dirigiremos a utilizar algún mecanismo (como pueda ser EDDM [Bae-2006]) que sea capaz de detectar en las distintas hojas cuándo se ha producido un cambio en los conceptos que se estaban observando. Además de paliar estas limitaciones incorporando nuevas estrategias, también deseamos profundizar en el estudio de métodos que mejoren el tratamiento de los atributos continuos, tanto a la hora de realizar posibles expansiones con más de dos intervalos (eliminando la limitación de expansiones dicotómicas) como a la hora de usar la información disponible para producir mejores predicciones. Como último punto a destacar en lo que se refiere a futuros trabajos planteados para mejorar el conocimiento interno del propio algoritmo, nos planteamos un estudio en profundidad de la repercusión que puedan tener las características del problema y del conjunto de datos en la complejidad del algoritmo.

**Summary of part I**

*The aim of this part of the thesis has been the presentation of a new algorithm based on concentration bounds and called IADEM-2. We have introduced some basic ideas (chapters 1 and 2) needed to describe the new contributions (chapter 3) and then, we have done an extensive experimental work (chapter 4) in order to evaluate the behaviour of the algorithm. The outstanding characteristics are described below.*

*Concentration bounds [Dia-2001; Jan-2002; Chu-2006] are one of the approaches most extensively used when it is needed to know which is the interval where a concrete variable must be. Concentration bounds calculate an estimation of the error margins for a given variable considering its average value (between* $0$ *and* $1$*) and the number of examples that have been used to calculate such average value. Furthermore, that calculation is done using a minimum confidence* $(1-\delta)$ *set by the user. In chapter 1 we have studied two concentration bounds widely used in the area of Machine Learning: Chernoff's bound [Che-1952] and Hoeffding's bound [Hoe-1963]. They have been reformulated in order to make uniform the expressions of both formulas and, therefore, we have achieved a simpler way to use them. Then, we have combined both bounds and studied the benefits of that combination. The margins of error calculated by both bounds are different and, depending on the values used in such calculation, the minimum margin will be reached by one bound or by the other one. We have done a study to compare both bounds and we have reached the conclusion that the estimation calculated by Chernoff's bound is tighter when the average value of the variable that is being considered is less than* $1/6$*. On the other hand, when that average value is bigger than* $1/6$*, the tighter estimation is the one calculated by Hoeffding's bound.*

*Knowledge discovery from databases (or data streams) is becoming an area more and more important because the volume of data is increasingly growing. There are different approaches to deal with very large datasets (or data streams) but one of the most extensively used is the "*sampling*" method [Fel-1968]. There is a wide range of sampling methods, from "*stratified*" sampling [Cat-1991] to "*active*" sampling [Saa-2004] or "*double*" sampling [Cox-1952] among other methods. The kind of sampling most interesting for us is the "*sequential*" sampling [Gav-2001], also known as "*dynamic*" sampling [Joh-1996] or "*adaptive*" sampling [Lip-1990]. Using this sampling method, the size of the sample does not have to be set a priori, but the algorithm can take different samples from the dataset in an iterative way. The evaluation of the requirements that must be satisfied before stopping the execution of the algorithm are evaluated after taking every new sample. Thus, the algorithm will stop when that requirements are achieved. The algorithm that we are presenting in this part of the thesis takes advantage of this characteristic. It samples with replacement from the original dataset until the user's requirements are satisfied and the induced model presents the minimum quality desired.*

*For designing the algorithm IADEM-2 we have considered the previous algorithm IADEM-0, whose first version was developed by Ramos and Morales [Ram-2000; Ram-2001]. The original implementation proposed for IADEM-0 has been substituted by a general definition where it can be seen a description of its working process [Cam-2002]. In that description we have made an abstraction of the previous explicit procedures and we have provided a new approach, much more general, in order to relax the implementation restrictions. In this way, the designer will have at his disposal a formal description of the algorithm and it will be able to implement it using any programming language. Furthermore, that abstraction process has favoured some improvements in the algorithm because new extreme cases have been considered and some calculations have been reformulated.*

*IADEM-0 is an algorithm that learns from datasets and uses decision trees to represent the acquired knowledge. Two kinds of leaves are used in the border of the tree: the real leaves and the virtual ones. The real leaves are the leaves that constitute the real border of the tree. Every real leaf has as many virtual leaves as attributes are unused in the branch that ends in that leaf. Thus, the set of virtual leaves that corresponds to a real leaf represents all the possible expansions for that real leaf and they register all the information that will be needed to do an expansion. In this way, IADEM-0 uses the information in the leaves (real and virtual) to decide how the border of the tree evolves.*

*In this algorithm, a very important point are the user's requirements about the maximum level of error*

*that is allowed and the confidence required. Thus, the user must set two arguments for the algorithm: the maximum allowed error for the tree that is going to be induced ($\varepsilon$) and the desired confidence ($1-\delta$). Using the information stored in the leaves (real leaves) and calculating some statistics and their error margins, IADEM-0 maintains, at every moment, an estimation of the superior and inferior bounds of the error in the tree. That estimation is very useful because it will be needed to stop the execution when the user's requirements are satisfied. Thus, the user can get the simplest model available for the minimum accuracy demanded, because, once the model achieves the accuracy requested, the model does not keep growing.*

*Other characteristic offered by IADEM-0 is the possibility of getting more information when the decision tree is used to predict. When it is used to predict the class of an observation, the model gives one prediction vector with the estimated values for the different classes, but this vector can be completed with two other vectors that delimit the intervals for that estimations in the best case and in the worst one.*

*In order to complete the description of the algorithm we have made some experiments. They are focused to decide which values are the most appropriate for the different internal parameters. Some characteristics of the algorithm can be configured by the user setting those parameters, but this configuration requires that the user has a deep knowledge of the algorithm. This circumstance is not usual, so we have fixed the values for that parameters to simplify the use of IADEM-0. We have made a detailed study of the behaviour of the algorithm considering diverse configurations, and we have explained how different values affect to it. Once we know the importance of every parameter, we have fixed the default values to achieve the best results in a wide variety of cases. This study has been completed with a basic analysis of the temporal and spatial complexity.*

*Although IADEM-0 achieves good results in many cases, it lacks some characteristics that limit its performance: it may work inappropriately when the datasets have noise, it can not deal with continuous attributes in a direct form (it needs a previous step of discretization), and it does not use all the available information in the decision tree to do the prediction process. These limitations have been overcome and some other improvements have been included in the design of the algorithm IADEM-2. Two algorithms have been developed between IADEM-0 and IADEM-2 and they have solved partially some of the limitations. They are IADEM [Ram-2006a] and IADEMc [Cam-2006]. But the algorithm that overcomes all the limitations and improves substantially the extraction of knowledge is the algorithm called IADEM-2 [Cam-2007], that is the core of this part of the thesis.*

*IADEM-2 improves the convergence of the algorithm, increases the quality of the induced models and makes them more accurate. The expansion process has been changed to improve the convergence of the algorithm, but more important is the inclusion of new criteria to detect the stabilization of the learning process or its worsening. These new criteria are based on a simplified version of the geometrical method proposed by Castillo [Cas-2006]. The quality of the models have been improved by the inclusion of different mechanisms: the use of a new method that detects when a concrete expansion must be made, the incorporation of mechanisms that allow working with continuous attributes in a direct form (adapting the proposal of Gama, Rocha and Medas [Gam-2003]) and the establishment of a maximum imbalance level before expanding a leaf. The accuracy of the model have been improved using some information in the decision tree that was previously wasted. Basically, the prediction process can be improved inducing naive Bayes classifiers in every leaf by using the information stored in the virtual leaves. These classifiers change the leaves into "*functional*" leaves as proposed Gama, Rocha and Medas [Gam-2003].*

*The improvements that we have proposed also include some internal parameters that can be used to modify their behaviours. As we did with the parameters defined in IADEM-0, we study the performance of the algorithm considering the new parameters and default values are set for them. We have included an analysis of the temporal and spatial complexity too.*

*To conclude the description of IADEM-2 we have made an extensive experimental study to show how the new algorithm performs in a wide variety of cases. To complete the comparisons carried out with the individual characteristics studied, we have defined a new method in the area of multiple attribute decision making (MADM [Hwa-1981]) and we have called it "*relative additive weighting*" (RAW). Its aim is to resume in one unique measure the characteristics that are relevant and that usually are considered individually.*

*Considering the results shown in the experimental section, we can conclude that IADEM-2 performs very well when it is used to extract knowledge from datasets of different size and with different levels of noise. From those results we can remark the stability as the most important characteristic. The decision trees induced present a very stable behaviour for three aspects: accuracy, size of the decision tree, and number of examples sampled to induce the model. The algorithm can adapt the induction of the decision tree taking into account the complexity of the knowledge that is being extracted and it samples more examples when the knowledge is more complex.*

*With respect to the results achieved by IADEM-2 when it deals with small datasets (taken from UCI Repository), we must note that the models induced are comparable with the models induced by other algorithms, although the time needed to finish the induction is much more higher. This delay is logical because IADEM-2 has not been designed to deal with small datasets and it needs more examples than other algorithms before getting statistical evidence of the convenience of stopping the induction. Considering the results observed when the algorithm is used with very large datasets (created using generators taken from UCI Repository or developed by ourselves), we must note that the results are very positive. The decision trees induced by IADEM-2 are the smallest, what affects in a better comprehensibility, and the accuracies achieved are the best ones, or they are near to the best ones.*

*To conclude this summary we must mention that it has been made one objective evaluation where the advantages and limitations of the algorithm are enumerated. As advantages we must consider the following ones: a high level of accuracy is achieved, the decision trees are easily understandable (because they are small), it is independent on the size of the datasets and it can deal with imbalanced datasets. As limitations we must consider the following ones: the independence of the size of the dataset also can be seen as a limitation if the dataset is too small, the high dimensionality of the problem to be solved and the presence of missing values or concept drift. The high dimensionality problem could be faced from different perspectives depending on what it is intended to be reduced. If we want to reduce the size of the balanced binary trees that store the information of the continuous attributes, we could set a maximum number of different values or a maximum precision. If we want to reduce the number of leaves that store information, we could use one approach similar to the one proposed by Domingos and Hulten [Dom-2000] and remove, temporarily, the least promising leaves. With respect to dealing with missing attributes, we will study the convenience of modifying the expansion predicates in order to give concrete values using an approach based on the one proposed by Fortes, Mora, Morales and Triguero [For-2006]. Finally, there are multiple methods to deal with concept drift, but our first steps will be directed in the way of using some mechanism (like EDDM [Bae-2006]) in the nodes of the decision tree. Thus, we will try to detect the concept drift and selectively remove the obsolete concepts. Besides overcoming different limitations, we also want to keep working in some methods to improve the processing of continuous attributes, both in expansion process (substituting dichotomic expansions by multiple intervals expansions), both in prediction process. As final point to be emphasized, we must note our interest in doing an extensive study about the repercussion of the different characteristics of a problem (or dataset) in the complexity of the algorithm.*

# Parte II

# Algoritmos Incrementales basados en Sistemas Multiclasificadores

# Capítulo 6

# Sistemas Multiclasificadores para Aprendizaje Incremental

En esta segunda parte de la tesis presentamos una serie de aportaciones realizadas a los sistemas multiclasificadores preparados para realizar un aprendizaje incremental. En este capítulo comenzaremos exponiendo las características que presentan este tipo de sistemas en su versión no incremental y las cualidades que los hacen tan interesantes. Para introducir los métodos que se usarán en la definición de los sistemas multiclasificadores que proponemos, se expondrán previamente, en la sección 6.2, una serie de algoritmos que hemos desarrollado y cuyas características y resultados nos permiten esperar un comportamiento bastante aceptable. En la sección 6.3 describiremos los principales enfoques que se han desarrollado para dotar a los sistemas multiclasificadores de la posibilidad de extraer conocimiento de grandes conjuntos de datos (o flujos de datos) y realizar un aprendizaje incremental. Al comentar dichos enfoques presentaremos las características más interesantes de cada uno de ellos y justificaremos las decisiones tomadas para desarrollar el primer sistema multiclasificador incremental que proponemos: MultiCIDIM-DS [Cam-2007a].

## 6.1. Sistemas multiclasificadores

Los *sistemas multiclasificadores* [Die-2000a] se han revelado como un mecanismo eficaz que combina un conjunto de clasificadores para producir un modelo algo más complejo pero también más preciso. La principal idea que subyace a los sistemas multiclasificadores es la posibilidad de mejorar la precisión de un modelo individual (clasificador básico) combinándolo con otros, sean del mismo tipo o de diferentes tipos, y pretendiendo que, en conjunto, sean capaces de evitar determinados errores en la predicción de observaciones [Han-1990]. De forma intuitiva se puede explicar esta mejora diciendo que, para una observación en concreto, es posible que un clasificador individual se equivoque, pero al usar más de un clasificador, es posible que la predicción correcta sea alcanzada por la mayoría de clasificadores, aunque exista una minoría que se equivoque. Combinando esas predicciones, la ofrecida en común por la mayoría de clasificadores básicos puede evitar errores individuales.

El principal objetivo de los sistemas multiclasificadores es aumentar el poder predictivo, pero se consigue a costa de mermar el poder descriptivo. Estos sistemas alcanzan, por lo general, una precisión mayor que la que alcanzan los clasificadores básicos individualmente, pero la interpretabilidad de un clasificador básico es, típicamente, mayor que la que se pueda alcanzar teniendo que combinar múltiples modelos. A pesar de este aumento en la dificultad para interpretar los sistemas multiclasificadores, existe una ventaja y es que se aumenta el poder expresivo, es decir, el modelo multiclasificador puede sintetizar conocimiento que antes no podía ser recogido en un único modelo básico.

Algo fundamental para inducir sistemas multiclasificadores de calidad, es decir, para generar modelos más precisos, es inducir clasificadores básicos que sean precisos y diferentes entre ellos [Han-1990]. Al decir que deben ser precisos nos referimos a que su tasa de error sea mejor que la propia de hacer predicciones aleatorias y con modelos diferentes nos referimos a que estos modelos deben cometer errores diferentes

para diferentes subconjuntos del espacio de búsqueda (errores no correlacionados [Jac-1995]). Si todos los modelos en el sistema multiclasificador cometiesen los mismos errores para las mismas experiencias, no tendría sentido disponer de múltiples modelos. El objetivo de tener varios, como ya hemos indicado, es que para los casos en los que un clasificador básico pueda equivocarse, existan otros que puedan darnos alternativas correctas.

A la hora de diseñar algoritmos que generen sistemas multiclasificadores hay que considerar dos puntos fundamentales [Gam-1999; Her-2004]: cómo son generados los distintos clasificadores básicos que componen el sistema multiclasificador y cómo son combinados para realizar una predicción conjunta. En la figura 6.1 hemos representado un esquema básico y resumido cómo se realizan estas tareas, de las que hablaremos con más detalle en los siguientes apartados (6.1.1 y 6.1.2).

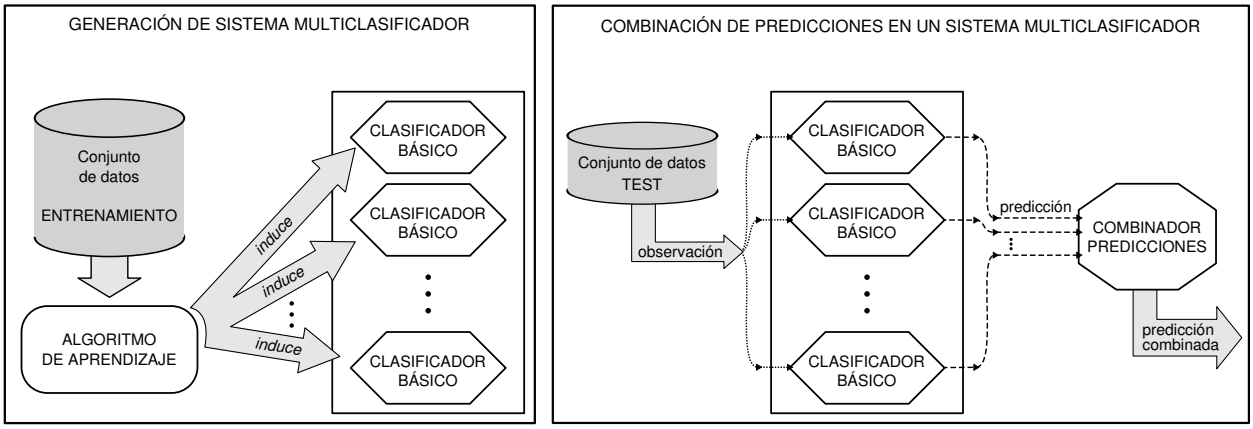



Figura 6.1: Esquemas simples de la generación de un sistema multiclasificador (izquierda) y de la combinación de las predicciones de los distintos clasificadores básicos que componen el sistema multiclasificador (derecha).

### 6.1.1. Métodos para la generación de sistemas multiclasificadores

Los trabajos en el ámbito de los sistemas multiclasificadores han sido numerosos y siguen surgiendo debido a su interés y a las ventajas que aportan. Es difícil recoger todos los enfoques propuestos de forma exhaustiva, pero podemos presentar diferentes características propias de los mismos que sirven para categorizarlos. Si atendemos a la homogeneidad de los modelos que componen el sistema multiclasificador [Gam-1999] podemos diferenciar entre aquellos que combinan clasificadores básicos homogéneos y aquellos que combinan clasificadores heterogéneos. Existe una gran variedad de modelos que pueden formar parte del sistema multiclasificador siempre que sean precisos y diferentes entre ellos: redes neuronales [Han-1990], clasificadores ingenuos de Bayes [Zhe-1998], árboles de decisión [Die-2000], máquinas de vectores soporte [Kim-2003], etc. Como exponentes más reconocidos y utilizados entre los que usan modelos homogéneos tenemos "*bagging*" [Bre-1996] y "*boosting*" [Fre-1996; Fre-1997]. Por su parte, en el grupo de métodos que usan modelos heterogéneos en la composición del sistema multiclasificador podemos destacar el método de "*generalización apilada*" ("*stacked generalization*") [Wol-1992] o el de "*generalización en cascada*" ("*cascade generalization*") [Gam-2000].

Como ya se ha expuesto, la diversidad de los modelos, sean del mismo tipo (homogéneos) o de tipos diferentes (heterogéneos), es un requisito necesario para conseguir sistemas multiclasificadores. Para alcanzar dicha diversidad, los algoritmos de aprendizaje deben lograr, mediante alguna fórmula, que los modelos que se generen no sean idénticos. Basándose en los diferentes elementos que pueden ser manipulados para conseguir que los algoritmos induzcan modelos diferentes, Dietterich propuso otra clasificación con cuatro

grupos [Die-2000a]: los métodos que manipulan los conjuntos de datos de entrenamiento, los que manipulan los atributos de entrada, los que manipulan las clases y los que introducen algún tipo de aleatoriedad en el proceso de aprendizaje. *Bagging* y *boosting* forman parte del grupo de algoritmos que consiguen la variedad manipulando el conjunto de datos de entrenamiento, aunque lo hacen de forma diferente: *bagging* no modifica la distribución de los datos de entrenamiento, mientras que *boosting* sí lo hace. La idea de modificar el conjunto de experiencias hay que combinarla con el uso de algoritmos que sean bastante sensibles a pequeñas variaciones del mismo, con el objetivo de producir modelos bastante diferentes con pequeñas variaciones en el conjunto de entrenamiento. A los algoritmos que presentan esta "*inestabilidad*" frente a pequeñas variaciones se les conoce como "*aprendices débiles*". En el grupo de métodos que manipulan los atributos de entrada podemos destacar los "*bosques de decisión*" [Ho-1998] que seleccionan un pequeño conjunto de dimensiones. Aunque la clase podría considerarse como atributo de entrada y su manipulación ser englobada en el grupo anterior, ocupa un lugar diferente en esta clasificación puesto que se considera a la clase como el objetivo de salida. Uno de los métodos perteneciente a este grupo sería el propuesto por el propio Dietterich y por Bakiri [Die-1995] en el que se generan subconjuntos mediante la partición aleatoria de las clases del conjunto original. En el último grupo de esta clasificación se introduce aleatoriedad a la hora de tomar las decisiones en el algoritmo de aprendizaje. En este grupo podemos destacar el algoritmo de "*aleatorización*" ("*randomization*"), también propuesto por Dietterich [Die-2000], en el que la expansión de una hoja por un atributo se hace de forma aleatoria seleccionando de entre los mejores atributos disponibles. Otro ejemplo en este mismo grupo es la inducción de "*multi-árboles de decisión*" [Est-2003] donde la aleatoriedad se introduce a la hora de seleccionar la partición con la que continuar la generación de la estructura.

### *Bagging* y *boosting*

Puesto que *bagging* y *boosting* han sido dos de los mecanismos más extendidos y sobre los que se han desarrollado multitud de variantes, procedemos a dar los detalles más significativos de cada uno de ellos. Además, como se verá en las siguientes secciones, parte de las aportaciones que presentaremos se basan en el esquema de *bagging* y en los resultados experimentales hemos usado ambos enfoques como modelos de referencia.

El método *bagging*, propuesto por Breiman [Bre-1996], es un método sencillo para generar diversos clasificadores a partir de un conjunto de datos. Su nombre viene de *bootstrap aggregation*, que resume la filosofía del método: generar diferentes subconjuntos de entrenamiento realizando muestreo con reemplazamiento a partir del original. Sería lógico pensar que si los subconjuntos generados son parecidos, los modelos inducidos usando un mismo algoritmo también fuesen parecidos y no cumpliesen con el requisito de ser diferentes entre ellos. Pero esto no ocurre siempre de este modo, ya que existen algoritmos de inducción que son muy sensibles a pequeñas variaciones en el conjunto de entrenamiento, por ejemplo, algunos algoritmos de inducción de árboles de decisión. Esta característica de inestabilidad es la que da nombre a los algoritmos conocidos como "*aprendices débiles*" y se ve acentuada cuando el subconjunto de entrenamiento tiene un tamaño limitado. Normalmente, el tipo de combinación de las predicciones que se usa es la votación mayoritaria: se elige la clase que haya sido seleccionada por más clasificadores individuales de entre los que componen el sistema multiclasificador.

El método *boosting*, propuesto por Freund y Schapire [Fre-1996; Fre-1997], asigna pesos a las experiencias en el conjunto de entrenamiento. El objetivo del método *boosting* es inducir un modelo que minimice la suma de los pesos de las experiencias que no se clasifiquen correctamente. En un principio, los pesos de las diferentes experiencias tendrán el mismo valor y los errores serán igual de importantes. Pero en cada iteración, los pesos de las experiencias que han sido incorrectamente clasificadas se aumentarán para darles más importancia. De esta forma, en cada iteración se le da más importancia a las experiencias que no han sido clasificadas correctamente en pasos anteriores. En este modelo, la combinación de los clasificadores básicos no es tan simple como con *bagging*. Ahora se tiene en cuenta el error estimado de cada modelo para calcular la clase de la observación que se está intentando predecir. Sumando los pesos de los modelos que eligen la misma clase, se selecciona aquella que acumule mayor valor.

Existen diversos trabajos que comparan los resultados alcanzados por *bagging* y *boosting* bajo diferentes circunstancias [Qui-1996; Bau-1999; Die-2000]. Como conclusiones generales llegan a decir que *boosting* alcanza mejores resultados que *bagging*, pero no en todos los casos. Los casos en los que *boosting* no alcanza resultados tan buenos son los problemas en los que existe ruido. La razón es sencilla, cuando existe ruido, *boosting* se centra en intentar aprender experiencias que no siempre están etiquetadas con la misma clase, que son aquellas que han sido mal clasificadas en las iteraciones anteriores. Al centrarse en aprender de estas experiencias, cosa que no es posible puesto que son ruidosas, deja de aprender de otras experiencias que sí eran susceptibles de aportar nuevo conocimiento.

### 6.1.2. Métodos para combinar los modelos en los sistemas multiclasificadores

Al igual que existen multitud de métodos para generar los modelos (clasificadores básicos) que componen un sistema multiclasificador, también existen multitud de métodos para combinar dichos modelos y usarlos para predecir la clase de una observación. Existen diversas formas de agruparlos, pero podemos fijarnos en dos aspectos (como se propone en la tesis de Gama [Gam-1999]): qué tipo de información se usa para realizar la predicción y qué modelos participan en dicha predicción.

Atendiendo al tipo de información que se usa podemos diferenciar los métodos en dos grupos: los modelos que "*votan*" por una clase concreta y los métodos que "*fusionan*" los vectores de predicción ofrecidos por los distintos modelos. Dentro de los métodos que realizan votación, se pueden distinguir aquellos que realizan una "*votación uniforme*" o los que realizan una "*votación ponderada*". La votación uniforme es aquella en la que la clase predicha por cada uno de los modelos tiene el mismo peso, y la predicción combinada es aquella que sume mayor número de "*votos*". Como ejemplo de este tipo de votación tenemos al método *bagging*. Para la votación ponderada existen multitud de técnicas que consideran diferentes características, pero la idea básica es que los diferentes modelos (o las partes que lo componen) reciben un peso determinado (calculado en función de ciertas características) con el que se intenta potenciar a los mejores clasificadores. Como ejemplos de estos métodos tenemos el método *boosting*, que ya mencionamos anteriormente, o el método de "*combinación bayesiana*" [Bun-1990] que asigna los pesos asociados a las probabilidades condicionadas de los modelos y a la precisión de las reglas que lo componen.

Los métodos de "*fusión*" forman el otro enfoque en el que los modelos no dan un único voto, sino que ofrecen un vector de predicción. Al disponer de los vectores de predicción calculados por los diferentes modelos, es necesario "*fusionarlos*" en un nuevo vector de predicción para ofrecer una única salida. Los mecanismos para definir ese nuevo vector son múltiples [Kit-1998]: cálculo de la suma, de la media, del producto, de la mediana, del valor máximo, del valor mínimo, etc.

Existen estudios que han comparado diversos métodos usados para combinar los modelos [Ali-1996; Kun-2002], pero la elección del mejor método dependerá del problema que se está tratando y de las características propias del sistema multiclasificador.

La otra característica que diferencia en dos grandes grupos a los sistemas multiclasificadores es la forma en que se seleccionan los modelos que van a ser usados en la predicción. Si se usan todos los modelos disponibles en el sistema multiclasificador, como puede ser el caso de *bagging* o *boosting*, estamos ante los "*métodos estáticos*"!estáticos. Por otro lado, si los modelos que son combinados para realizar la predicción se eligen expresamente dependiendo de la observación que se vaya a clasificar, los métodos se llaman "*dinámicos*". La razón que motiva la aparición de estos modelos es que los errores no están uniformemente distribuidos por todo el universo de experiencias y habrá regiones en las que unos clasificadores se comporten mejor, y otras en las que sean otros clasificadores los que hayan aprendido mejor el concepto. Seleccionar los modelos más precisos para cada observación que se quiera clasificar puede ser un enfoque que incremente la precisión global alcanzada por el sistema multiclasificador.

En este caso, la mayor variedad de propuestas aparece en el grupo de los métodos dinámicos. Así, podemos destacar, entre otros, los métodos que estudian la "*aplicabilidad del modelo* ("*model applicability induction*") [Ort-1996], los "*aprendices compuestos*" ("*composite learner*") [Tin-1997] o el método "*SCANN*" [Mer-1998]. El método propuesto por Ortega [Ort-1996], *model applicability induction*, presenta el uso de un modelo adicional que acompañe al modelo clasificador y que sea capaz de aprender en qué cir-

cunstancias el modelo clasificador realiza clasificaciones correctas y en cuáles no. De esta forma, se podrán seleccionar los modelos que mejor se ajusten a la observación cuya clase se intenta predecir. El objetivo de los aprendices compuestos (*composite learner*) propuestos por Ting [Tin-1997] es similar: elegir los clasificadores que alcancen los mayores niveles de precisión, caracterizándolos mediante una medida que se basa en las distancias entre clases diferentes y entre clases iguales. El método *SCANN* [Mer-1998], presentado por Merz, usa la transformación de la matriz que recoge todas las predicciones de todos los clasificadores para todas las experiencias. Esa transformación se usa para encontrar modelos redundantes y para caracterizar qué modelos son mejores en los diferentes subconjuntos en los que se pueda dividir el universo de experiencias.

## 6.2. Sistemas multiclasificadores basados en CIDIM

Tras haber descrito las características básicas que presentan los sistemas clasificadores, los requisitos mínimos necesarios para que sean útiles (precisión de los modelos básicos y diversidad de éstos) y las ventajas que pueden aportar, pasamos a describir algunos sistemas clasificadores propios que nos servirán de base para los nuevos algoritmos que propondremos en esta parte de la tesis. Estos sistemas clasificadores usan como algoritmo de aprendizaje para inducir los clasificadores básicos el algoritmo CIDIM. Este algoritmo, inicialmente propuesto por Ramos, Morales y Villalba [Ram-2000a; Ram-2001], ha sido objeto de posteriores estudios [Ram-2005c] y ha sido aplicado a casos reales con resultados satisfactorios [Jer-2003; Rui-2005], debido principalmente a sus propiedades: inducción de árboles de decisión precisos y de reducido tamaño.

Aprovechando estas características y añadiéndole la particularidad de que, por el propio diseño del algoritmo CIDIM, es sencillo obtener diferentes árboles de decisión, aun usando el mismo conjunto de datos, resulta apropiado diseñar métodos que utilicen CIDIM como algoritmo de aprendizaje para inducir los clasificadores básicos. Como consecuencia de diversos enfoques se han diseñado diferentes sistemas multiclasificadores entre los que tenemos los que siguen un enfoque más clásico, como M-CIDIM [Ram-2005b], E-CIDIM [Ram-2005a] o FE-CIDIM [Ram-2006], o los que usan múltiples sistemas multiclasificadores en sucesivas capas para refinar la clasificación, como ML_m-CIDIM [Ram-2005d] o ML-bin_m-CIDIM [Ram-2005].

No es objeto de esta tesis realizar un estudio detallado sobre el algoritmo CIDIM ni de los diferentes sistemas multiclasificadores que se han diseñado aprovechando sus ventajas, pero consideramos que es necesario describirlos, aunque sea de forma breve, para apoyar nuestra decisión de desarrollar los nuevos algoritmos incrementales que proponemos basándonos también en CIDIM y justificar la necesidad de incorporarles un enfoque incremental. Los trabajos que exponemos a continuación son el propio algoritmo CIDIM (subsección 6.2.1) y el sistema multiclasificador FE-CIDIM (subsección 6.2.2), que es el último sistema multiclasificador basado en CIDIM que hemos desarrollado hasta el momento.

### 6.2.1. CIDIM

El algoritmo CIDIM ha sido desarrollado para extraer conocimiento de conjuntos de datos cuyos atributos sean categóricos (pueden estar ordenados o no) y representar el modelo inducido mediante árboles de decisión. El diseño de este algoritmo permite, además de alcanzar una elevada precisión, que los árboles inducidos tengan un tamaño reducido. Existen tres ideas básicas que fundamentan el comportamiento de CIDIM y que presentaremos con algo más de detalle en los siguientes apartados: la división del conjunto de entrenamiento, la agrupación de valores consecutivos y un control local para detener la inducción. El nombre de CIDIM surge de algunas de estas características, puesto que su nombre es el acrónimo de "*Control de Inducccción por DIvisión Muestral*" [Ram-2005b] (o el acrónimo en inglés de "*Control of Induction by sample DIvision Method*" [Ram-2005c]).

**División del conjunto de entrenamiento**

Los algoritmos clásicos para la inducción de árboles de decisión (TDIDT [Qui-1986; Qui-1993]) suelen dividir el conjunto de experiencias en dos subconjuntos: el subconjunto de entrenamiento (con el que se induce el árbol de decisión) y el subconjunto de test (con el que se comprueban los resultados). El algoritmo CIDIM realiza una división adicional sobre el subconjunto de entrenamiento ($E$) y obtiene dos nuevos subconjuntos con la misma frecuencia de clases y de tamaño similar. De estos nuevos conjuntos uno es usado para la inducción o construcción del modelo (llamado $CNS$) y otro es utilizado para las tareas de control (llamado $CLS$).

En la figura 6.2 se puede observar una descripción más formal de estos subconjuntos usados por el algoritmo CIDIM en el proceso de expansión. Cada nodo del árbol de decisión tiene sus correspondientes subconjuntos $CNS$ y $CLS$ y, cuando se realiza una expansión, éstos subconjuntos se dividen atendiendo a los nuevos hijos que se acaban de generar. De esta forma, el tamaño de ambos conjuntos va disminuyendo conforme se profundiza en el árbol. Lo importante de estos subconjuntos radica en el uso que se hace de ellos para formar grupos de valores consecutivos y para evaluar la condición de control local a cada nodo en el proceso de expansión.

$$
\begin{array}{l}
|CNS| \neq 0 \\
|CLS| \neq 0 \\
CNS \cap CLS = \emptyset \\
CNS \cup CLS = E \\
0 \leq |CNS| - |CLS| \leq 1 \\
0 \leq |\{e \in CNS \mid C(e) = i\}| - |\{e \in CLS \mid C(e) = i\}| \leq 1 \quad \forall i \in CLASES \\
\quad \text{donde } C(e) \text{ es la clase de la experiencia } e
\end{array}
$$

Figura 6.2: Subconjuntos $CNS$ y $CLS$.

**Agrupación de valores consecutivos**

Normalmente, al expandir una hoja utilizando un atributo categórico, se añaden tantos hijos como valores presente el atributo seleccionado. Pero, si éste es ordenado, la realización de un tratamiento previo que detecte agrupaciones de valores consecutivos podría reducir el número de nuevos hijos, y, por tanto, el tamaño final del árbol. El algoritmo CIDIM contempla distintos comportamientos para realizar una expansión en función del tipo de atributo que vaya a ser utilizado en dicha expansión. Si el atributo no es ordenado, la expansión se realiza de la forma tradicional, creando un nuevo hijo por cada valor; pero, si el atributo es ordenado, CIDIM usa un algoritmo voraz, basado en una división dicotómica recursiva, para encontrar agrupaciones de valores. Inicialmente existe un único grupo con todos los valores y en cada paso se evalúa si una división en dos grupos produciría alguna mejora. Este proceso es recursivo y continúa hasta que no haya mejora o hasta que sólo quede un valor por grupo. Para realizar las divisiones se usa el subconjunto $CNS$ y se decide si una división produce alguna mejora mediante una medida de desorden (que llamamos $md\_división$). Usando esta método, CIDIM busca cuál es la mejor división para cada uno de los atributos ordenados que quedan sin usar. Al igual que hace el algoritmo BOAT [Geh-1999] o el algoritmo IADEM-2 [Cam-2007], CIDIM no está condicionado a usar siempre la misma medida de desorden y ésta puede configurarse. Tras una serie de pruebas empíricas hemos determinado que la entropía es una buena medida de desorden para ser usada en el proceso de división y, por tanto, ésta será la que usemos por defecto.

Para decidir qué par atributo-división es el mejor de todos los posibles en la hoja que se va a expandir también se usa otra medida de desorden (que llamamos $md\_mejora$). Ésta tampoco es fija, sino que puede elegirse libremente como ocurre con la medida $md\_división$. Puesto que los resultados obtenidos con la entropía se encuentran entre los mejores, usaremos también esta medida de desorden en el proceso de control, siendo así $md\_división$ igual a $md\_mejora$. Cuando se ha elegido el mejor par atributo-división, se

realiza un control local de inducción (que presentamos en el siguiente apartado) para evaluar la conveniencia de realizar la expansión correspondiente. Este control es local a cada hoja puesto que un mismo atributo puede estar dividido en distintas agrupaciones de valores dependiendo de las experiencias almacenadas en cada hoja.

**Control local de inducción**

La forma más sencilla para detener la expansión de los árboles de decisión es aquella que controla que todas las experiencias en una hoja estén etiquetadas con la misma clase, pero esto suele generar árboles demasiado grandes. Para evitar esto, algunos algoritmos introducen criterios externos que detienen la ejecución y que se conocen como mecanismos de "*prepoda*" [Min-1989; Esp-1997]. El algoritmo CIDIM usa la siguiente condición local como mecanismo de poda: una hoja se expande sólo si se produce una mejora en la precisión calculada sobre el subconjunto de control ($CLS$). Esta condición se supervisa de forma local en cada hoja y tiene en cuenta dos índices: un índice absoluto ($I_A$) y otro relativo ($I_R$) (ver ecuaciones 6.1 y 6.2). Para expandir una hoja es preciso que alguno de los dos índices mejore y que ninguno empeore. Los índices absoluto y relativo se definen del siguiente modo:

$$I_A = \frac{\sum_{i=1}^{N} CORRECT(e_i)}{N} \tag{6.1}$$

$$I_R = \frac{\sum_{i=1}^{N} P_{C(e_i)}(e_i)}{N} \tag{6.2}$$

donde $N$ es el número de experiencias en $CLS$, $e$ es una experiencia, $C(e)$ es la clase de la experiencia $e$, $P_m(e)$ es la probabilidad de la clase $m$ para la experiencia $e$ en la hoja usando el subconjunto $CNS$, y $CORRECT(e) = 1$ si $P_{C(e_i)} = max\{P_1(e), P_2(e), ..., P_k(e)\}$ ó 0 en otro caso.

**Algoritmo CIDIM**

Una vez hemos presentado los componentes fundamentales del algoritmo podemos describirlo con más detalle. A continuación, en la figura 6.3, mostramos el pseudocódigo del algoritmo CIDIM:

**Entrada:** $E$ (conjunto de datos),
$md\_división$ (medida de desorden), $md\_mejora$ (medida de desorden)

1. $\{CNS, CLS\} = División\_Dicotómica\_Aleatoria(E)$
2. $Raiz = Nueva\_Hoja(CNS, CLS)$
3. $SinEtiqueta = \{Raiz\}$
4. **mientras** $SinEtiqueta \neq \emptyset$ **hacer:**
4.1. $Hoja = Seleccionar\_Aleatoriamente(SinEtiq)$
4.2. $Atrib\_\&\_Div = Mejor\_Atributo\_División(Hoja, md\_división, md\_mejora)$
4.3. **si** $Mejora\_Precisión(Hoja, Atrib\_\&\_Div)$ **entonces:**
4.3.1. $Hijos = Expandir(Hoja, Atrib\_\&\_Div)$
4.3.2. $SinEtiqueta = SinEtiqueta \cup Hijos$
4.4. $SinEtiqueta = SinEtiqueta - \{Hoja\}$

**Salida:** $Raiz$ (raíz del árbol de decisión inducido)

Figura 6.3: Pseudocódigo del algoritmo CIDIM.

El algoritmo CIDIM recibe como entrada un conjunto de datos de entrenamiento ($E$) y dos medidas de desorden ($md\_división$ y $md\_mejora$) que, por defecto, serán la entropía. Después de extraer el conocimiento del conjunto de entrenamiento y de inducir un árbol de decisión, devuelve su nodo raíz ($Raiz$).

El primer paso del algoritmo consiste en realizar una división dicotómica aleatoria del conjunto de entrenamiento ($División\_Dicotómica\_Aleatoria(E)$) y construir los dos subconjuntos que serán usados para la inducción del árbol de decisión ($CNS$ y $CLS$). Esta división mantiene la frecuencia de clases del conjunto original ($E$) y cumple los requisitos expuestos en la figura 6.2. Después de realizar la división, se construye la raíz del árbol usando los subconjuntos $CNS$ y $CLS$, y se incluye como único elemento provisional del conjunto que contendrá las hojas sin etiquetar ($SinEtiqueta$). Desde ese momento se repite un proceso iterativo que continúa mientras queden hojas sin etiquetar ($SinEtiqueta \neq \emptyset$). El proceso comienza seleccionando una de esas hoja de forma aleatoria ($Hoja = Seleccionar\_Aleatoriamente$ $(SinEtiqueta)$) y calculando, para dicha hoja, cuál es el par atributo-división que mayor mejora produce ($Atrib\_\&\_Div = Mejor\_Atributo\_División(\ Hoja, md\_división, md\_mejora)$). Dicho par se calcula mediante la división dicotómica recursiva antes descrita, usando el subconjunto de construcción ($CNS$) y las dos medidas de desorden. Una vez se ha determinado cuál es el mejor par atributo-división, se comprueba si la mejor expansión que se puede realizar aumenta la precisión calculada usando el subconjunto de control ($CLS$) y los índices descritos en las ecuaciones 6.1 y 6.2. Si la expansión no se debe realizar, la hoja se etiqueta como "*no expansible*" (y se retira del conjunto de hojas que aún no han sido etiquetadas); pero si la expansión mejora la precisión, se añaden al árbol los hijos correspondientes al atributo seleccionado según la división realizada.

Tras realizar diversos estudios experimentales, que no vamos a detallar en esta tesis pero que pueden encontrarse en distintos trabajos [Ram-2005c; Ram-2006b], se puede observar que los árboles de decisión que se inducen son bastante precisos (comparables con otros algoritmos) y, además, son bastante pequeños (más que los alcanzados por otros algoritmos). Estas características son de bastante utilidad puesto que además de modelos precisos, se obtienen modelos fácilmente interpretables por los usuarios.

### 6.2.2. FE-CIDIM

Después de haber presentado el algoritmo CIDIM y de observar sus características, se puede llegar a la conclusión de que este algoritmo es bastante indicado para ser usado como algoritmo de aprendizaje en la inducción de los clasificadores básicos que componen un sistema multiclasificador. Utilizándolo de esta forma hemos definido distintos sistemas multiclasificadores, desde el m-CIDIM [Ram-2005d] que es el modelo más básico, hasta el FE-CIDIM [Ram-2006] que es el método más avanzado que hemos desarrollado hasta el momento.

La razón que motiva el uso de CIDIM como algoritmo de aprendizaje para generar sistemas multiclasificadores se basa en la apreciación hecha por Hansen y Salamon [Han-1990]: para generar un buen sistema clasificador es necesario que los clasificadores básicos cumplan sean precisos y sean diferentes entre ellos. Los árboles de decisión inducidos por CIDIM son precisos y, debido a la división dicotómica aleatoria del conjunto de entrenamiento ($División\_Dicotómica\_Aleatoria(E)$), que se realiza como primer paso del algoritmo, los árboles de decisión generados pueden ser muy diferentes incluso usando el mismo conjunto de entrenamiento ($E$). Además, al usar CIDIM obtenemos, como ventaja adicional, que el sistema multiclasificador puede ser más fácilmente interpretable porque los árboles de decisión son de un tamaño reducido.

En esta subsección presentamos el algoritmo FE-CIDIM [Ram-2006] que genera sistemas multiclasificadores y que ha sido diseñado usando como base otro algoritmo llamado E-CIDIM [Ram-2005a]. La única diferencia existente entre ambos algoritmos es que FE-CIDIM incluye un factor de mejora que controla el proceso de inducción y consigue resultados similares a E-CIDIM (incluso mejores) en un tiempo mucho menor. En la figura 6.4 se puede observar el pseudocódigo del algoritmo FE-CIDIM de forma detallada. A continuación indicaremos, de forma resumida, las claves sobre el funcionamiento de este algoritmo.

El algoritmo FE-CIDIM se caracteriza porque induce, de forma iterativa, diferentes clasificadores básicos usando el algoritmo CIDIM. Dichos clasificadores serán diferentes en la mayoría de los casos debido al proceso de aleatorización que sufre el conjunto de entrenamiento por parte del algoritmo CIDIM. Existe un número máximo de árboles que compondrán el sistema multiclasificador y su valor se indica con el argumento $max\_num\_árboles$. En la primera fase del proceso de inducción se introducen, en el conjunto de clasificadores básicos, la mitad del máximo número de árboles que formarán parte del conjunto. De esta

forma se inicializa el sistema multiclasificador de una forma más directa que la utilizada en el proceso general (controlada por el bucle "mientras") y se garantiza una cantidad mínima de clasificadores básicos en el sistema.

El procedimiento de inducción de nuevos árboles de decisión que son susceptibles de ser incluidos en el sistema multiclasificador se detiene cuando, durante un número determinado de intentos consecutivos ($max\_intentos\_fallidos$), no se consigue mejorar ninguno de los árboles de decisión que ya existían. La única diferencia que se ha introducido en FE-CIDIM y que lo diferencia de E-CIDIM, acelarando la convergencia, es la modificación del criterio que controla cuánto debe mejorar el clasificador nuevo al peor para ser incluido en el sistema multiclasificador. En el algoritmo E-CIDIM cualquier pequeña diferencia que hiciese que la tasa de error del nuevo árbol inducido fuese menor que la del peor árbol en el sistema multiclasificador suponía la inclusión de aquél. Como consecuencia, el contador que controlaba el número de intentos consecutivos en los que no se había incluido ningún nuevo clasificador básico ($Intentos\_fallidos$) era reiniciado con el valor 0 y esto solía postergar la finalización del algoritmo. Para acelerar la convergencia, FE-CIDIM incluye un factor de mejora ($\gamma \in (0, 1]$) que se usa para permitir únicamente la inclusión de aquellos nuevos clasificadores básicos que sean mejores que el peor en un porcentaje mínimo. De esta forma se evita la inclusión de muchos árboles de decisión y se acelera la inducción del sistema multiclasificador definitivo.

**Entrada:** $E$ (conjunto de datos), $max\_num\_árboles$ (máximo número de árboles)
$max\_intentos\_fallidos$ (máximo número de intentos fallidos), $\gamma$ (factor de mejora)

1. $Conjunto = \emptyset$
2. $num\_inicial\_árboles = \lceil max\_num\_árboles/2 \rceil$
3. **desde** 1 **hasta** $num\_inicial\_árboles$ **hacer:**
3.1. $Nuevo\_árbol = CIDIM(E, entropia, entropia)$
3.2. $Conjunto = Conjunto \cup Nuevo\_árbol$
4. $Intentos\_fallidos = 0$
5. **mientras** $Intentos\_fallidos < max\_intentos\_fallidos$ **hacer:**
5.1. $Peor\_árbol = \{ x \in Conjunto \mid tasa\_acierto(x) < tasa\_acierto(y) \; \forall y \in Conjunto \}$
5.2. $Nuevo\_árbol = CIDIM(E, entropia, entropia)$
5.3. **si** $tasa\_error(Nuevo\_árbol) < \gamma \cdot tasa\_error(Peor\_árbol)$ **entonces:**
5.3.1. $Intentos\_fallidos = 0$
5.3.2. $Conjunto = Conjunto \cup Nuevo\_árbol$
5.3.3. **si** $|Conjunto| > max\_num\_árboles$ **entonces:**
5.3.3.1. $Conjunto = Conjunto - Peor\_árbol$
5.4. **si no:**
5.4.1. $Intentos\_fallidos = Intentos\_fallidos + 1$

**Salida:** $Conjunto$ (sistema multiclasificador inducido)

Figura 6.4: Pseudocódigo del algoritmo FE-CIDIM.

Tras realizar diversos estudios experimentales, que no vamos a detallar en esta tesis pero que pueden encontrarse en distintos trabajos [Ram-2005a; Ram-2006], se puede observar que los sistemas multiclasificadores que se inducen utilizando CIDIM como algoritmo de aprendizaje básico son bastante precisos (comparable con otros métodos) y, además, su tamaño es menor que el que obtienen otros algoritmos.

## 6.3. Adaptando los sistemas multiclasificadores al aprendizaje incremental

Hemos presentado, en las secciones anteriores (6.1 y 6.2), distintos aspectos que hacen de los sistemas multiclasificadores un enfoque bastante útil. Pero aún no hemos presentado una de sus características más interesantes: son fácilmente modificables para realizar un aprendizaje incremental. Los algoritmos que inducen los sistemas multiclasifadores pueden ser fácilmente adaptados para procesar conjuntos de experiencias de forma incremental, ya sea tomándola por bloques o de una en una. Entre los primeros trabajos que contemplan esta posibilidad podemos destacar los de Schapire [Sch-1990] o Freund [Fre-1995] en los que se plantea la posibilidad de usar la técnica de "*boosting*" para realizar este tipo de aprendizaje. Con algunas diferencias entre sus trabajos, lo que proponían era, básicamente, "*filtrar*" las experiencias que se iban tomando y utilizar, principalmente, aquellas que aún no eran correctamente clasificadas. Como método se proponía el uso de la técnica de "*boosting*" pero, como señalan Domingo y Watanabe [Dom-2000a], el enfoque que usaban no era adaptativo puesto que se necesitaba conocer (y configurar) una cota inferior con la ventaja que se podía esperar al usar un algoritmo de aprendizaje (aprendiz débil). Para usar una variante adaptativa, una de las mejores opciones es el algoritmo AdaBoost [Fre-1996; Fre-1997], que ha sido usado por Breiman [Bre-1999], y modificado por Domingo y Watanabe [Dom-2000a], para definir métodos que induzcan sistemas multiclasificadores con un enfoque incremental.

Esta primera aproximación en la que se aplica la técnica de "*boosting*" mediante filtrado (en inglés "*boosting by filtering*") presenta un defecto que podría limitar su uso incremental cuando se tratan grandes conjuntos de datos: el número de clasificadores que componen el sistema multiclasificador no ha sido limitado. Con cada iteración, en la que se construye un conjunto de entrenamiento para inducir un nuevo clasificador básico, se añade un nuevo modelo al sistema, y esto podría requerir una cantidad de memoria demasiado elevada. Para superar esta limitación se podría optar por un enfoque que únicamente conserve los últimos modelos, lo que sería como definir una ventana de un determinado tamaño en la que se guardarían los modelos más recientes. Esta solución es la que aportan Fan, Stolfo y Zhang [Fan-1999], además de proponer dos nuevos tipos de filtrado alternativos (uno aleatorio y otro que divide el conjunto de datos).

Esta solución también puede plantear algún inconveniente debido a que únicamente se almacenan los últimos clasificadores básicos. Un enfoque, ligeramente diferente, que solucionaría esta posible dificultad es el que proponen Street y Kim [Str-2001] en el algoritmo SEA. La forma en la que se determina qué clasificadores deben formar parte del sistema multiclasificador es más elaborada que la de otros algoritmos y se plantea la inclusión de un nuevo clasificador básico únicamente cuando haya evidencia de que es mejor que alguno de los que ya existen en el sistema multiclasificador. Además, para mantener limitada la cantidad de memoria necesaria, el tamaño del sistema multiclasificador también es limitado y, si al añadir un multiclasificador básico se supera el límite, se elimina alguno de los clasificadores básicos. La forma en la que se seleccionan las experiencias también varía porque no usa el enfoque de *boosting*, sino que se induce un nuevo clasificador básico con cada nuevo bloque de experiencias que se procese. De esta forma se sigue pudiendo extraer conocimiento de grandes conjuntos de datos (y flujos de datos) de forma natural. En la figura 6.5 hemos representado un esquema simple del proceso de inducción del sistema multiclasificador y de su uso para realizar predicciones. Como puede apreciarse (en el esquema de la izquierda), el conjunto de entrenamiento usado en cada paso no se usa únicamente para inducir un nuevo clasificador básico, sino que también es usado para actualizar los que ya existían. Esa actualización no afecta al modelo en sí, sino que se usa para actualizar distintas medidas que sirven para controlar su calidad como, por ejemplo, la tasa de error. Esas medidas que controlan la calidad del modelo, junto con otras que describen en sí al propio modelo, son las que servirán para decidir cuándo un clasificador básico es mejor que otro y puede reemplazarse el peor para dejar espacio al nuevo. Si observamos el esquema de la derecha en la misma figura 6.5, podemos apreciar que el método para combinar la predicción de los clasificadores básicos mantiene una estructura similar al que se pueda usar en los sistemas multiclasificadores no incrementales.

En el pseudocódigo de la figura 6.6 hemos expresado la estructura básica que presentaría un algoritmo basado en el enfoque de Street y Kim y que nos resultará útil porque este enfoque es básicamente igual al que emplearemos en la siguiente sección para definir un nuevo algoritmo.

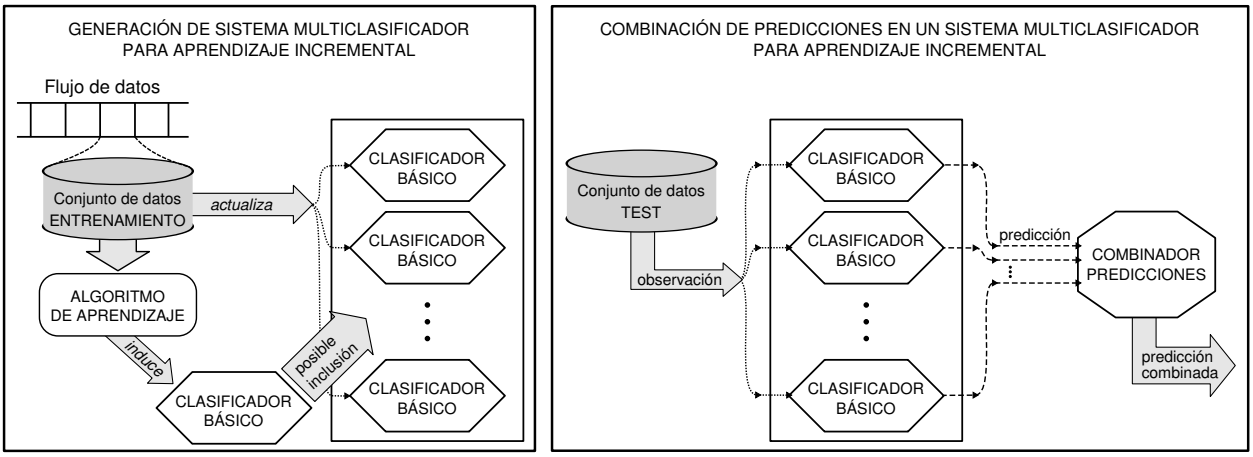



Figura 6.5: Esquemas simples de la generación de un sistema multiclasificador incremental siguiendo un enfoque secuencial (izquierda) y de la combinación de las predicciones de los distintos clasificadores básicos que componen dicho sistema multiclasificador (derecha).

**Entrada:** $E$ (conjunto de datos),
$algoritmo\_aprendizaje$ (algoritmo para inducir los clasificadores básicos),
$tam\_bloque$ (tamaño del bloque de datos por iteración),
$max\_num\_clasif$ (máximo número de clasificadores básicos)

1. $Conjunto = \emptyset$
2. **mientras** $E \neq \emptyset$ **hacer:**
2.1 $Entrenamiento \leftarrow$ Extraer las primeras $tam\_bloque$ experiencias de $E$ (son eliminadas de $E$)
2.2. $Clasificador_i \leftarrow$ Inducir un nuevo clasificador con $algoritmo\_aprendizaje$ usando $Entrenamiento$
2.3. $Conjunto = Conjunto \cup \{Clasificador_{i-1}\}$
2.4. **Para todo** $x \in Conjunto$ **hacer:**
2.4.1 Actualizar $tasa\_acierto(x)$ usando $Entrenamiento$
2.5. **si** $|Conjunto| > max\_num\_clasif$ **entonces:**
2.5.1. $Peor\_clasificador = \{x \in Conjunto \mid tasa\_acierto(x) < tasa\_acierto(y) \; \forall y \in Conjunto \wedge x \neq y\}$
2.5.2. $Conjunto = Conjunto - \{Peor\_clasificador\}$

**Salida:** $Conjunto$ (sistema multiclasificador inducido)

Figura 6.6: Pseudocódigo del método general para inducir sistemas multiclasificadores incrementales.

Los enfoques que hemos descrito hasta este momento se clasificarían como métodos de "*generación secuencial*" según la definición dada por Fern y Givan [Fer-2003]. Reciben este nombre porque los distintos clasificadores básicos que tratan de incluirse en el sistema multiclasificador son inducidos uno tras otro (secuencialmente) usando las experiencias del conjunto de datos según se van construyendo los conjuntos de entrenamiento. Si se usan algoritmos de aprendizaje no incrementales para inducir los clasificadores básicos, como ocurre en los algoritmos que hemos citado anteriormente, la forma más adecuada para generarlos es la secuencial. Existe otro enfoque que usa algoritmos de aprendizaje incremental para inducir y mantener actualizados los clasificadores básicos. Este enfoque realiza una "*generación en paralelo*" del sistema multiclasificador, puesto que todas las experiencias que se procesen serán utilizadas por el algoritmo incremental

para actualizar todos los modelos. Como ejemplos de este tipo de generación tenemos los algoritmos "*online bagging*" y "*online boosting*" propuestos por Fern y Givan [Fer-2003] y por Oza [Oza-2001; Oza-2005]. Entre las principales diferencias existentes entre estos trabajos está el tipo de modelo que generan los algoritmos de aprendizaje (árboles de decisión en el trabajo de Fern y Givan y redes neuronales y clasificadores de Bayes en el trabajo de Oza), pero lo realmente importante es que los modelos se generan y actualizan mediante algoritmos incrementales. Para realizar este tipo de generación, los autores cuentan con la ventaja que supone el hecho de contar con procesadores que puedan trabajar en paralelo, pero, ni estos sistemas están siempre disponibles, ni los sistemas de generación secuencial están incapacitados para aprovechar esa misma ventaja de multiprocesamiento.

## 6.4. MultiCIDIM-DS: un sistema multiclasificador para aprendizaje incremental basado en CIDIM

Habiendo presentado distintas formas que existen para extender las capacidades de los sistemas multiclasificadores y permitirles desarrollar un aprendizaje incremental, presentamos a continuación el primer algoritmo que proponemos en esta parte de la tesis. Su nombre es MultiCIDIM-DS [Cam-2007a] y se debe a que el algoritmo de aprendizaje que usa para inducir los algoritmos básicos es CIDIM [Ram-2005c] y a que está preparado para trabajar con grandes conjuntos de datos o flujos de datos ("-DS" por "*data stream*"). El proceso básico que lo define es el que hemos presentado en la figura 6.6, donde el argumento $algoritmo\_aprendizaje$ es definido con el algoritmo CIDIM.

Su esquema de funcionamiento es similar al que se utiliza en el algoritmo SEA [Str-2001] pero con pequeñas variaciones. La elección de este enfoque se debe a diversas circunstancias que detallamos a continuación. La primera, que nos hace descartar todos los enfoques basados en boosting, es que el algoritmo de inducción básico que vamos a usar (CIDIM) no está diseñado para minimizar los errores ponderados de la forma en que los presenta *boosting*. En segundo lugar, el enfoque seleccionado no presenta la limitación de que el sistema multiclasificador pueda requerir cada vez más memoria por el hecho de añadir nuevos clasificadores básicos y no eliminar ninguno antiguo. Por último, en cada momento considera cuáles son los clasificadores básicos más precisos y no los elimina simplemente por el hecho de ser más antiguos. Las variaciones más destacadas que lo diferencian del algoritmo SEA son la elección del algoritmo CIDIM en lugar del algoritmo C4.5 [Qui-1993] y la modificación del criterio para comparar los modelos. En el algoritmo SEA no se precisa que el clasificador que se elimine sea el peor de todos, sólo se indica que sea alguno peor que el nuevo; y el criterio utilizado por el algoritmo SEA para determinar qué modelo es peor no considera únicamente la tasa de acierto, sino que la compara con otros indicadores más elaborados.

Como hemos indicado, el argumento de entrada $algoritmo\_aprendizaje$ ya está determinado, pero aún existen otros dos argumentos que determinan el comportamiento del algoritmo, además del propio conjunto de datos. Nos referimos al número de experiencias que compondrán cada uno de los subconjuntos que serán usados para inducir cada uno de los clasificadores básicos ($tam\_bloque$) y al número máximo de clasificadores básicos que se permitirá almacenar ($max\_num\_clasif$). A continuación, en las figuras 6.7 y 6.8, hemos presentado un análisis del comportamiento del algoritmo considerando ambas variables y analizando cómo afectan a la mejora de la precisión (que es el objetivo principal de los sistemas multiclasificadores).

En ambas figuras hemos representado los mismos datos y la única diferencia que existen entre ellas es la presencia de ruido. Para hacer las gráficas de la figura 6.7 hemos usado el conjunto de datos *Sint-1*, que fue presentado en la sección 4.5 (1 millón de experiencias, 20 atributos categóricos con entre 5 y 7 valores, 4 clases y sin ruido), y para las gráficas de la figura 6.8 hemos usado el conjunto de datos *Sint-3*, que fue presentado en la misma sección (1 millón de experiencias, 20 atributos categóricos con entre 4 y 6 valores, 4 clases y 15 % de ruido).

En la figura 6.7, donde el conjunto de datos no tiene ruido, se puede observar que la variación del parámetro $tam\_bloque$ influye bastante en la inducción de un sistema que sea capaz de clasificar correctamente el conjunto de datos. El mismo comportamiento se puede observar en la figura 6.8, en la que se usaron conjuntos de datos con ruido.

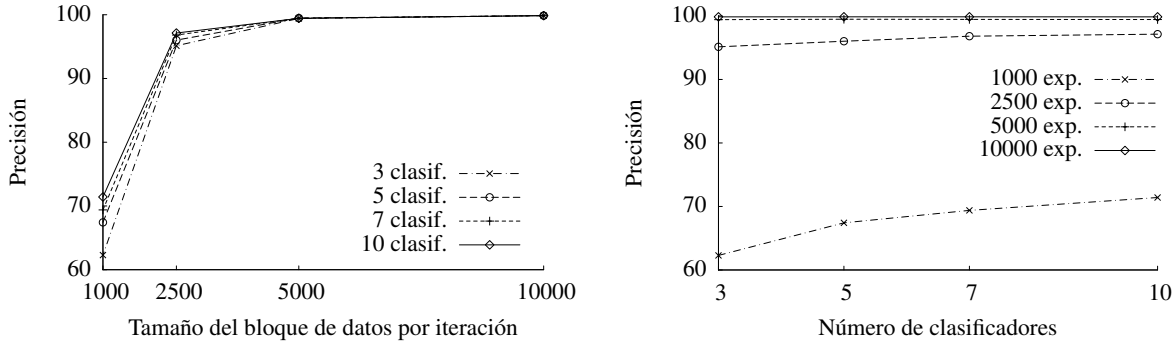



Figura 6.7: Comportamiento dependiendo del tamaño del bloque de datos por iteración ($tam\_bloque$) y del número máximo de clasificadores ($max\_num\_clasif$) usando un conjunto de datos sin ruido (*Sint-1*)

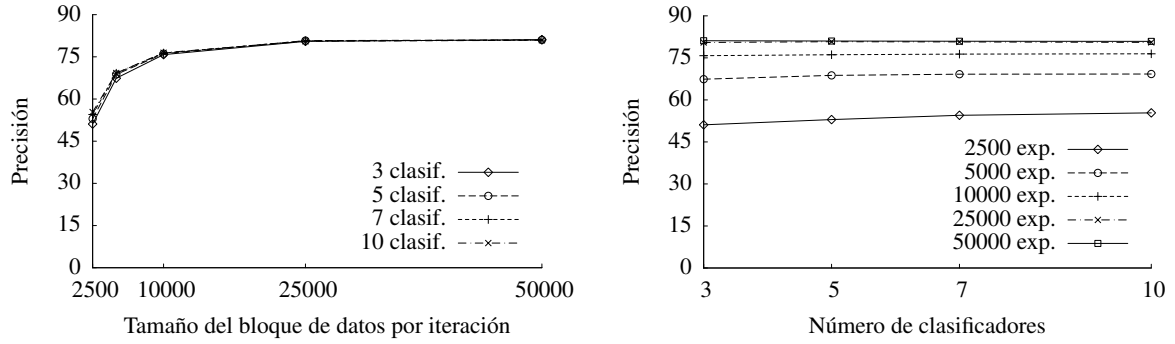



Figura 6.8: Comportamiento dependiendo del tamaño del bloque de datos por iteración ($tam\_bloque$) y del número máximo de clasificadores ($max\_num\_clasif$) usando un conjunto de datos con un $15\,\%$ de ruido (*Sint-3*)

El máximo número de clasificadores que intervienen no es demasiado relevante, pero se pueden apreciar ligeras diferencias dependiendo de su valor. Se observa que los sistemas multiclasificadores más precisos son los que tienen más modelos (entre 7 y 10) pero las diferencias son cada vez menores entre ellos cuanto más modelos los conformen. Para la realización de los experimentos que presentaremos en el capítulo 8 hemos usado como valor por defecto para este argumento ($max\_num\_clasif$) el valor 10 puesto que en los estudios preliminares ofrece precisiones elevadas y no se aprecian apenas diferencias cuando se usan valores mayores. Además, los sistemas multiclasificadores típicos también usan alrededor de 10 clasificadores básicos normalmente.

Para el argumento $tam\_bloque$ no es tan sencillo intentar establecer un valor por defecto. Como se puede observar, pasando de un valor determinado, la precisión se estabiliza (incluso la dependencia del argumento $max\_num\_clasif$ se hace menos importante), pero ese valor no puede ser fijado para todas las situaciones de una forma tan simple. Hemos observado que el valor de $tam\_bloque$ tiene que ser mayor cuanto más ruido se introduce en el conjunto de datos, consideración lógica puesto que a mayor complejidad del problema, mayor número de experiencias necesitará el algoritmo de aprendizaje para inducir clasificadores básicos precisos. Como estrategia hemos optado por tomar el valor de 10000 para los experimentos en los que no exista ruido y 50000 para los experimentos en los que pueda existir ruido. Un análisis más detallado y proponer estrategias adaptativas son parte de nuestras futuras líneas de investigación.

# Capítulo 7

# Mejorando los Sistemas Multiclasificadores mediante Filtros Correctores

En este capítulo presentamos un método para mejorar la precisión alcanzada por los sistemas multiclasificadores y que está diseñado para poder ser usada especialmente en el ámbito del aprendizaje incremental. En primer lugar, en la sección 7.1, presentaremos dicho método que puede ser aplicado a cualquier mecanismo que induzca sistemas multiclasificadores para aprendizaje incremental y, a continuación, en la sección 7.2, describiremos una implementación concreta de este método que ha dado lugar al algoritmo llamado MultiCIDIM-DS-CFC [Cam-2007a].

## 7.1. Clasificación mediante filtros correctores

En los enfoques que hemos expuesto anteriormente, los métodos inducen sistemas multiclasificadores en los que los modelos existentes son combinados para clasificar (predecir) una observación. Aunque existen métodos que inducen modelos de diferente naturaleza (heterogéneos), los que hemos presentado y son más comunes inducen siempre el mismo tipo de modelo (homogéneos). Para ello sólo es preciso definir un tipo de algoritmo de aprendizaje que los induzca. Los diferentes métodos expuestos también presentan diferentes formas de combinar las predicciones que, como se comentó en la subsección 6.1.2, pueden ser la "*votación*" usando la clase mayoritaria o la "*fusión* de los vectores de predicción para generar uno nuevo. Lo que sí tienen en común es que usan las predicciones que ofrecen todos los clasificadores básicos y que los define como "*métodos estáticos*" (según la definición dada en la misma subsección 6.1.2).

En contraposición a los métodos estáticos existen los "*métodos dinámicos*" que únicamente combinan los clasificadores básicos más relevantes para cada caso concreto. Esta modificación en la forma en que se combinan los clasificadores básicos entraría en el ámbito de la "*teoría de la revisión*" [Fla-1989; Moo-1993] y un trabajo directamente relacionado con los sistemas multiclasificadores es el realizado por Ortega [Ort-1996]. En este trabajo se propone el uso de unos "*árbitros*" que actúen activando o desactivando los clasificadores básicos que pueden ser combinados cada vez que se realiza una predicción en concreto. Para cada modelo inducido se crea un modelo adicional ("*árbitro*") que es inducido por otro algoritmo de aprendizaje (puede ser el mismo) y que extraerá conocimiento sobre las zonas del universo de experiencias que han sido correctamente aprendidas y las que no. Para extraer ese conocimiento usará el conjunto de experiencias modificado en el que se ha sustituido la clase original por otra binaria. Esa nueva clase binaria recogerá la información de clasificaciones correctas o incorrectas del clasificador en cuestión para la experiencia considerada. Usando el conocimiento que cada "*árbitro*" tiene sobre su clasificador básico, el sistema multiclasificador decidirá qué clasificadores básicos funcionan correctamente y deben ser activados para participar en la combinación de las predicciones. En el mismo trabajo de Ortega se plantean dos métodos para combinar los modelos que hayan sido activados: MMM, en el que se realiza por votación mayoritaria, y MMC, en el que se considera únicamente la votación del modelo que más confianza muestre (según el árbitro correspondiente).

El principal inconveniente de este trabajo es que se realiza desde una perspectiva no incremental, pero su filosofía es la misma que la que utilizaremos nosotros para mejorar la clasificación usando filtros correctores (CFC) en un ambiente incremental. En el trabajo de Street y Kim [Str-2001], donde se presenta el algoritmo SEA, que sí es incremental, se planteó el uso de algún tipo de mecanismo para realizar una combinación *dinámica* pero, como ellos mismos comentan, no observaron ninguna mejora consistente y abandonaron esa línea de trabajo. Con el método que proponemos a continuación se consigue mejorar la precisión de los sistemas multiclasificadores de manera consistente y para el caso incremental. Al método lo hemos llamado "*clasificación mediante filtros correctores*" (CFC) porque los clasificadores que pueden intervenir en la combinación son "*filtrados*" para "*corregir*" una posible predicción errónea. En la figura 7.1 presentamos dos esquemas: uno con la forma en que se genera el sistema multiclasificador usando los filtros correctores (arriba) y otro con la forma en que son combinados para ofrecer la predicción (abajo).

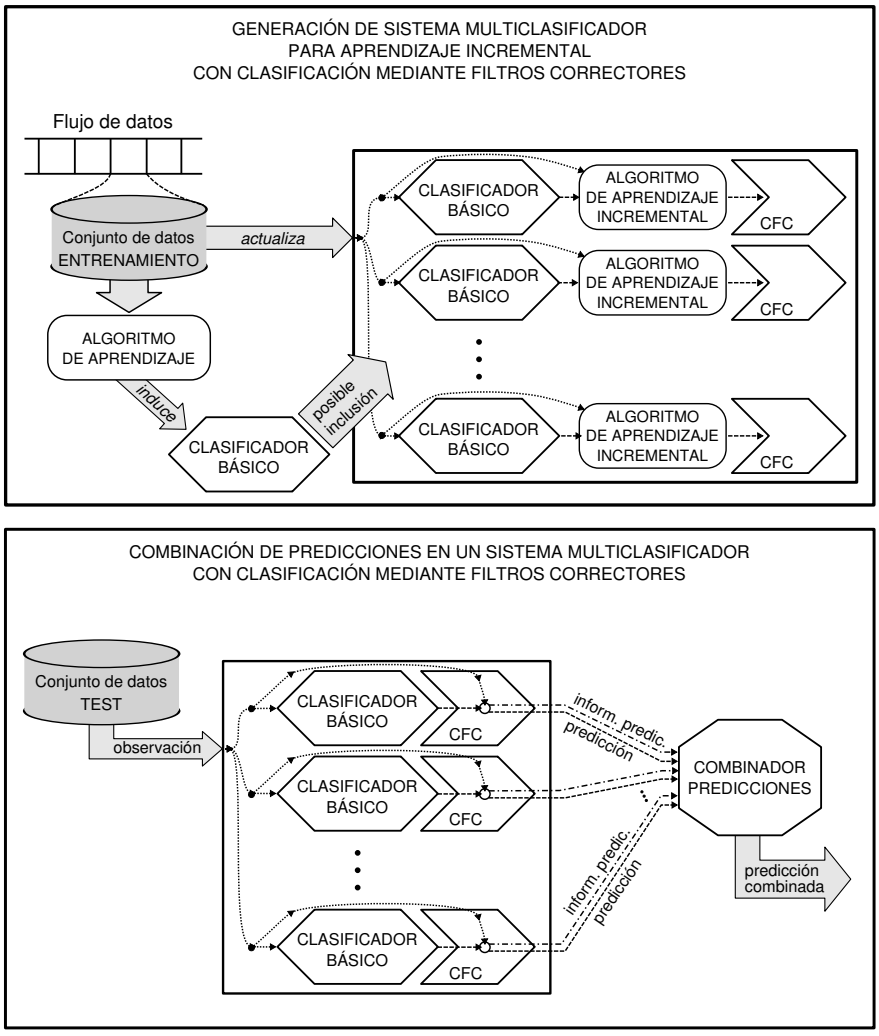



Figura 7.1: Esquemas simples de la generación de un sistema multiclasificador incremental con filtros correctores (arriba) y de la combinación de las predicciones de los distintos clasificadores básicos que componen dicho sistema (abajo).

Para explicar la forma en que se induce un sistema multiclasificador que incluya filtros correctores también hemos incluido un pseudocódigo, que presentamos en la figura 7.2. Como puede apreciarse, el funcionamiento del método está basado en el que se presentó anteriormente en la figura 6.5 y sólo incluye las modificaciones necesarias para contemplar el uso de los filtros correctores. Para cada uno de los clasificado-

res básicos que se incluyan en el sistema multiclasificador se creará, en el momento mismo de la inclusión, un modelo que recogerá el conocimiento sobre qué tipo de experiencias son correctamente clasificadas por el clasificador básico. Para poder llevar a cabo esta tarea es preciso incluir un nuevo argumento de entrada en el algoritmo: un algoritmo de aprendizaje incremental ($algoritmo\_aprendizaje\_incremental$). Su función será que cada filtro corrector aprenda, de forma incremental, qué experiencias son las que clasifica correctamente el clasificador básico que le corresponde. Se necesita que este aprendizaje sea incremental porque es posible que un clasificador básico, que no es incremental y permanece inalterable desde que se induce, permanezca en el sistema multiclasificador durante muchas iteraciones (situación que se dará mientras su tasa de acierto sea mayor que la de algún otro). Cuanto más tiempo se encuentre en el sistema multiclasificador, más conjuntos de entrenamiento habrán servido para incrementar el conocimiento y la única forma de incorporarlo todo, sin desechar nada, es usando un algoritmo incremental. Si consideramos el funcionamiento de este método desde la perspectiva del tipo de generación que se lleva a cabo debemos decir que sería una mezcla entre *generación secuencial* y *generación paralela*. Sería secuencial en la forma de añadir nuevos clasificadores básicos, porque con cada nuevo conjunto de experiencias se crea un único nuevo clasificador básico. Sería paralela en la forma de añadir conocimiento a los filtros correctores, porque éstos son actualizados todos en cada iteración con el mismo conjunto de experiencias.

**Entrada:** $E$ (conjunto de datos),
$algoritmo\_aprendizaje$ (algoritmo para inducir los clasificadores básicos),
$algoritmo\_aprendizaje\_incremental$ (algoritmo para inducir los filtros correctores),
$tam\_bloque$ (tamaño del bloque de datos por iteración),
$max\_num\_clasif$ (máximo número de clasificadores básicos)

1. $Conjunto = \emptyset$
2. **mientras** $E \neq \emptyset$ **hacer:**
2.1 $Entrenamiento \leftarrow$ Extraer las primeras $tam\_bloque$ experiencias de $E$ (son eliminadas de $E$)
2.2. $Clasificador_i \leftarrow$ Inducir un nuevo clasificador con $algoritmo\_aprendizaje$ usando $Entrenamiento$
2.3. $Conjunto = Conjunto \cup \{Clasificador_{i-1}\}$
2.4. **Para todo** $x \in Conjunto$ **hacer:**
2.4.1 Actualizar $tasa\_acierto(x)$ usando $Entrenamiento$
2.4.2 $Entrenamiento\_CFC \leftarrow$ completar experiencias en $Entrenamiento$ incorporando la clasificación hecha por $x$
2.4.3 **si** $CFC_x$ existe **entonces:**
2.4.3.1 $CFC_x \leftarrow$ actualizar $CFC_x$ con $algoritmo\_aprendizaje\_incremental$ usando $Entrenamiento\_CFC$
2.4.4 **si no:**
2.4.4.1 $CFC_x \leftarrow$ inducir nuevo filtro corrector con $algoritmo\_aprendizaje\_incremental$ usando $Entrenamiento\_CFC$
2.5. **si** $|Conjunto| > max\_num\_clasif$ **entonces:**
2.5.1. $Peor\_clasificador = \{x \in Conjunto \mid tasa\_acierto(x) < tasa\_acierto(y) \; \forall y \in Conjunto \wedge x \neq y\}$
2.5.2. $Conjunto = Conjunto - \{Peor\_clasificador\}$

**Salida:** $Conjunto$ (sistema multiclasificador inducido)

Figura 7.2: Pseudocódigo del método general para inducir sistemas multiclasificadores incrementales con filtros correctores.

El proceso que se sigue para combinar los predicciones de los diferentes clasificadores no es ninguna de las propuestas por Ortega. En nuestro caso no nos quedamos únicamente con la mejor ni usamos la votación mayoritaria de las que clasifiquen correctamente la observación en cuestión. En la figura 7.3 hemos presentado el proceso que se debe seguir para calcular el vector de predicción para una observación usando los filtros correctores. Se presentan dos comportamientos diferentes dependiendo de si existe algún clasificador básico que clasifique correctamente la observación (según indique su filtro corrector correspondiente), o no existe ninguno. En caso de que exista al menos un clasificador cuyo filtro corrector diga que es capaz de clasificar la observación correctamente ($Prob\_Mejor\_Prediccion \geq 0{,}5$), se identifica qué porcentaje de acierto tiene el mejor clasificador usando la información en los filtros correctores ($Prob\_Mejor\_Prediccion$). El vector de predicción será el resultado de normalizar la suma ponderada de los vectores de predicción de los clasificadores básicos cuyo porcentaje de acierto esté cerca del mejor (en un $10\,\%$). En futuros trabajos realizaremos un estudio más detallado de la importancia de esta cercanía a la probabilidad de acierto del mejor clasificador básico, pero debemos indicar que, de los estudios que hemos realizado, se puede deducir que un valor aconsejable estaría entre $0\,\%$ y $10\,\%$. Por otro lado, en caso de que ningún clasificador sea capaz de clasificar correctamente la observación (según la información ofrecida por los filtros correctores), el vector de predicción será, básicamente, el vector inverso del que resultaría de combinar todos los clasificadores básicos. Si todos los clasificadores fallan, todos intervendrán en la creación del nuevo vector de predicción, pero, como todos fallan, invertiremos dicho vector para que la clase mayoritaria sea la minoritaria y viceversa.

**Entrada:** $Conjunto$ (sistema multiclasificador con filtros correctores),
$o$ (observación)

1. $N \leftarrow$ número de clasificadores básicos en $Conjunto$
2. $c_i \leftarrow$ vector de predicción para $o$ usando el $i$-ésimo clasificador básico en $Conjunto$
3. $f_i \leftarrow$ probabilidad de clasificar correctamente la observación $o$ usando el $i$-ésimo clasificador básico en $Conjunto$ (se calcula usando el $i$-ésimo filtro corrector)
4. $F = \{f_i | 1 \geq i \geq N\}$
5. $Prob\_Mejor\_Prediccion = \text{máx}\{F\}$
6. $Min\_Valor\_Para\_Fusion = \text{máx}(0{,}5, (Prob\_Mejor\_Prediccion - (Prob\_Mejor\_Prediccion * 0{,}1)))$
7. $Vector\_Pred = (0, 0, \ldots, 0)$
8. **si** $Prob\_Mejor\_Prediccion \geq 0{,}5$ **entonces:**
8.1. **desde** $i = 1$ **hasta** $N$ **hacer:**
8.1.1. **si** $f_i \geq Min\_Valor\_Para\_Fusion$ **entonces:**
8.1.1.1. $Vector\_Pred = Vector\_Pred + c_i * f_i$
9. **sino:**
9.1. **desde** $i = 1$ **hasta** $N$ **hacer:**
9.1.1. $Vector\_Pred = Vector\_Pred + c_i * (1 - f_i)$
9.2. **desde** $j = 1$ **hasta** $|Vector\_Pred|$ **hacer:**
9.2.1. $Vector\_Pred_j = 1 \,/\, Vector\_Pred_j$
10. $Vector\_Pred = normalizar(Vector\_Pred)$

**Salida:** $Vector\_Pred$ (vector de predicción)

Figura 7.3: Proceso de predicción usando filtros correctores.

## 7.2. MultiCIDIM-DS-CFC: aplicando la clasificación mediante filtros correctores

A partir del método que acabamos de describir para mejorar la precisión alcanzada por los sistemas multiclasificadores orientados para realizar aprendizaje incremental, definimos el algoritmo MultiCIDIM-DS-CFC. Como se puede deducir del nombre, este algoritmo usa como base el algoritmo MultiCIDIM-DS (presentado en la sección 6.4) y le incorpora la capacidad de clasificar usando filtros correctores ("-CFC").

Para definir este algoritmo, lo único que es necesario concretar es el algoritmo incremental que será usado para actualizar la información de los filtros correctores. Dada nuestra experiencia con el algoritmo IADEM-2 y después de observar los buenos resultados alcanzados, hemos optado por seleccionar este algoritmo para definir el argumento de entrada $algoritmo\_aprendizaje\_incremental$. El resto de argumentos de entrada tendrán un tratamiento similar al explicado para el algoritmo MultiCIDIM-DS.

Del algoritmo MultiCIDIM-DS-CFC en sí no hay nada más que comentar puesto que todo lo necesario para definirlo ya ha sido explicado. Lo único que resta es evaluar su comportamiento y confirmar que el uso de los filtros correctores realmente mejora la precisión. Para ello hemos realizado un estudio experimental bastante extenso y cuyos resultados presentamos en el siguiente capítulo (capítulo 8).

# Capítulo 8

# Estudio Experimental

Al igual que hicimos en la parte I de esta tesis, después de exponer las aportaciones que presentamos para mejorar los sistemas multiclasificadores realizando la tarea de clasificación mediante filtros correctores (CFC), procedemos a realizar un estudio experimental en el que se pongan de manifiesto las características observables, ya sean ventajas o limitaciones. Para ello empezaremos describiendo, en la sección 8.1, los algoritmos que van a intervenir en las comparaciones, el tipo de experimentos que se van a realizar y otros detalles relevantes. En la sección 8.2 realizaremos un estudio de la mejora observada al usar los filtros correctores y analizaremos cómo afecta dependiendo de la configuración del algoritmo. A continuación, en la sección 8.3, mostraremos los resultados obtenidos al aplicar los diferentes algoritmos a distintos tipos de conjuntos de datos. En este estudio experimental se pondrán de manifiesto determinadas características, pero hemos de señalar que el principal objetivo de este estudio será poner de manifiesto la ventaja que supone el hecho de incorporar los filtros correctores en el sistema multiclasificador. Finalmente, en la sección 8.4, haremos un resumen de las principales características que se hayan observado en el estudio experimental.

## 8.1. Detalles de la experimentación

En esta sección describiremos las características más relevantes de la experimentación que se ha realizado, justificando la elección de los algoritmos usados, de los elementos observados y de los conjuntos de datos empleados. En términos generales, la experimentación realizada será similar a la realizada en la parte I de la tesis, pero presentará ciertas diferencias: se han elegido otros algoritmos más adecuados para realizar el estudio comparativo, se ha dejado de estudiar algún elemento que ha perdido relevancia y la variedad de conjuntos de datos se ha reducido para ajustarlo a las características observables en este caso concreto. Seguiremos usando el nuevo índice combinado propuesto en la subsección 4.1.3, puesto que se revela como una herramienta útil para resumir diversas características en una sola medida y facilitar así la interpretación de los resultados.

### 8.1.1. Algoritmos usados en el estudio experimental

En este apartado enumeramos los algoritmos que se han usado en el estudio experimental y detallamos la características más interesantes de cada uno de ellos para justificar su elección.

La referencia para realizar las comparaciones será el algoritmo MultiCIDIM-DS-CFC por recoger las aportaciones presentadas en esta parte de la tesis. Se trata de seleccionar algoritmos cuya implementación esté disponible para ser usada y que reúnan una serie de características que nos permitan evaluar las diferentes cualidades del algoritmo en cuestión: MultiCIDIM-DS-CFC. De nuevo, al igual que ocurría en el estudio experimental realizado en el capítulo 4, el hecho de que necesitemos algoritmos que tengan disponible su implementación (independientemente del lenguaje de programación empleado) presenta la primera gran limitación que nos encontramos a la hora de realizar la selección.

El algoritmo MultiCIDIM-DS-CFC (y MultiCIDIM-DS) induce sistemas multiclasificadores de forma incremental, pero, a pesar de que existen trabajos sobre diferentes sistemas multiclasificadores incrementales [Str-2001; Fer-2003], no ha sido posible utilizar ninguno de ellos en nuestro estudio. La razón es que no se encuentran implementaciones disponibles y, para paliar parcialmente esta deficiencia, hemos utilizado como referencia dos algoritmos incrementales que no son multiclasificadores y dos sistemas multiclasificadores que no son incrementales. A continuación daremos detalles sobre ellos, pero antes queremos destacar que la principal característica que se desea poner de manifiesto, más que el hecho de comparar MultiCIDIM-DS-CFC con otros algoritmos, es la mejora introducida por el uso de filtros correctores en sistemas multiclasificadores dedicados al aprendizaje incremental.

Puesto que MultiCIDIM-DS-CFC es un algoritmo que induce árboles de decisión como clasificador básico, ya que usa CIDIM como algoritmo de aprendizaje, hemos restringido el tipo de algoritmos seleccionados a aquellos que también inducen árboles de decisión, ya sea formando parte de un sistema multiclasificador o no. Las razones son similares a las expuestas en el estudio experimental realizado en la parte anterior de esta tesis: facilitar las comparaciones entre los modelos y que el tratamiento que hacen los algoritmos del espacio de búsqueda sea lo más parecido posible.

A ninguno de los algoritmos que se han usado les ha sido modificada su configuración por defecto, es decir, los experimentos han sido realizados con los algoritmos tal y como se encuentran disponibles. Considerando los requisitos que acabamos de exponer, pasamos a enumerar los algoritmos que se han utilizado:

- **VFDTc**: este algoritmo propuesto por Gama y otros [Gam-2003; Gam-2006] ya se ha usado como algoritmo de referencia en el anterior estudio experimental y es uno de los algoritmos incrementales, no basado en sistemas multiclasificadores, que usaremos para establecer una referencia de lo que se puede esperar de cualquier algoritmo de aprendizaje incremental. La implementación de este algoritmo está disponible en la siguiente dirección: `http://www.liacc.up.pt/~jgama/ales2/vfdtc`.

- **IADEM-2**: es el segundo algoritmo incremental no basado en sistemas multiclasificadores que usaremos como referencia. Es uno de los núcleos de esta tesis, ha sido ampliamente detallado, y aprovecharemos este estudio experimental para evaluar su comportamiento con otros algoritmos diferentes de los anteriormente usados. Su implementación está disponible en la siguiente dirección: `http://iaia.lcc.uma.es/~jcampo`. Las entradas al algoritmo son las mismas que las usadas anteriormente: $\varepsilon = 0,01$ y $\delta = 0,05$.

- **Bagging-J48**: es el algoritmo *bagging* [Bre-1996] que usa como algoritmo de aprendizaje básico para inducir los clasificadores básicos al algoritmo C4.5 [Qui-1993]. Hemos usado la implementación disponible en la herramienta Weka [Wit-2005] (`http://www.cs.waikato.ac.nz/ml/weka`) y por eso recibe el nombre de "Bagging-J48", porque usa la implementación de C4.5 llamada J48 como clasificador básico del algoritmo *bagging*. Este algoritmo será el más parecido, en términos de estrategia de generación del sistema multiclasificador, al algoritmo Multi-CIDIM-DS-CFC. Aun así, no será directamente comparable, como señalaremos en la sección 8.3, puesto que no dispone de ningún enfoque incremental. Por esa razón, mientras el algoritmo Multi-CIDIM-DS-CFC procesará el conjunto de datos de entrenamiento de forma incremental y por bloques, Bagging-J48 lo hará de una sola vez, disponiendo de más información para la inducción de cada uno de los clasificadores básicos pero consumiendo más tiempo. Por defecto, el número de iteraciones que se producen será de 10, que coincide con el parámetro interno por defecto que se usa en Multi-CIDIM-DS-CFC para almacenar un máximo de 10 clasificadores básicos.

- **Boosting-J48**: es el algoritmo *boosting* [Fre-1996; Fre-1997] que usa como algoritmo de aprendizaje básico para inducir los clasificadores básicos al algoritmo C4.5. Hemos usado, de nuevo, la implementación disponible en la herramienta Weka y por eso recibe el nombre de "Boosting-J48", porque usa la implementación de C4.5 llamada J48 como clasificador básico del algoritmo *boosting*. Este algoritmo no es tan similar a Multi-CIDIM-DS-CFC como lo es Bagging-J48, pero, por tratarse de un sistema multiclasificador, también hemos considerado oportuno utilizarlo. Este sistema, tampoco

es incremental y procesará el conjunto de experiencias de forma similar a Bagging-J48, es decir, de una sola vez. Por defecto, el número de iteraciones que se producen también será de 10, volviendo a coincidir con el parámetro interno por defecto que se usa en Multi-CIDIM-DS-CFC para almacenar un máximo de 10 clasificadores básicos.

- **Multi-CIDIM-DS**: este algoritmo es el presentado en la sección 6.4. Se trata de un sistema multiclasificador para aprendizaje incremental, cuyo algoritmo de aprendizaje básico es CIDIM [Ram-2005c] y que está diseñado para procesar la entrada por bloques, induciendo un nuevo clasificador básico con cada nuevo bloque de datos que sea recibido. Su implementación está disponible en la dirección `http://iaia.lcc.uma.es/~jcampo`. El único argumento de entrada, el tamaño del bloque necesario para inducir cada uno de los clasificadores básicos ($tam\_bloque$), tomará el valor de 10000 cuando trate un conjunto de datos sin ruido y 50000 cuando trate un conjunto de datos con ruido. Estos valores han sido elegidos por las razones expuestas en la sección 6.4.

- **Multi-CIDIM-DS-CFC**: este algoritmo es el que hemos presentado en la sección 7.2 y el que incorpora las principales aportaciones realizadas en esta parte de la tesis. Lo que le diferencia del algoritmo anterior es el uso de filtros correctores para la clasificación, que, en este caso, son generados por el algoritmo incremental IADEM-2. Su implementación también está disponible en la dirección `http://iaia.lcc.uma.es/~jcampo` y la configuración será idéntica a la empleada en el algoritmo Multi-CIDIM-DS.

De los algoritmos que acabamos de describir y que se usarán en este estudio experimental, cuatro son incrementales (VFDTc, IADEM-2, MultiCIDIM-DS y MultiCIDIM-DS-CFC) y cuatro generan sistemas multiclasificadores (Bagging-J48, Boosting-J48, MultiCIDIM-DS y MultiCIDIM-DS-CFC). Como se puede apreciar, los únicos que están en los dos grupo (generan sistemas multiclasificadores siguiendo un aprendizaje incremental) son MultiCIDIM-DS y MultiCIDIM-DS-CFC, los algoritmos que queremos comparar en primer término. Los otros algoritmos se han incluido para que sirvan de referencia o modelo a la hora de evaluar determinados aspectos que se comentarán en las siguientes secciones.

Todos los algoritmos han sido ejecutados en el mismo ordenador utilizado en la parte I. Su procesador es un Intel® Xeon™ con una frecuencia de CPU de $3,2$ GHz y la memoria ha sido limitada para seguir usando un máximo de 512MB, con la intención de evaluar las capacidades de cada algoritmo bajo las mismas condiciones. Únicamente, en determinadas ocasiones (que se indicarán claramente) hemos aumentado el límite de memoria para evaluar los resultados que obtendrían determinados algoritmos que no eran capaces de completar la inducción con el tamaño de memoria previamente fijado.

### 8.1.2. Elementos observados para el estudio experimental

Las características que observaremos en este estudio experimental serán casi las mismas que las observadas en el estudio experimental del capítulo 4. Las más relevantes seguirán siendo la precisión y el tamaño medio de los árboles de decisión, puesto que los modelos más precisos y más sencillos serán los más útiles, pero existen una serie de diferencias que debemos aclarar antes de avanzar en el desarrollo del estudio experimental. La primera es que la sencillez de los modelos ya no tiene el mismo significado que anteriormente. Para los algoritmos VFDTc e IADEM-2 sigue siendo el tamaño del árbol de decisión inducido, pero en el caso de los sistemas multiclasificadores, debido a que no se cuenta con un único árbol, el tamaño que se calculará será el tamaño medio de los clasificadores básicos que componen cada sistema multiclasificador. La segunda diferencia es que no hemos considerado el número de experiencias observadas por los diferentes algoritmos debido a que en todos los casos (excepto para IADEM-2) se consume el conjunto de datos por completo. Por último, el tiempo también ha sido considerado, pero debemos puntualizar que en el caso de los sistemas multiclasificadores no incrementales (Bagging-J48 y Boosting-J48), el tiempo consumido no puede ser directamente comparado con el resto de algoritmos, debido a que la inducción de cada uno de los clasificadores básicos se realiza considerando el conjunto de datos completo cada vez.

Para realizar los experimentos hemos realizado una validación cruzada con 10 particiones y los resultados que se mostrarán son los obtenidos al hacer la media de las 10 particiones. Además, en el caso de los valores de precisión y del número de hojas, también mostraremos el valor de la desviación típica. Para realizar las comparaciones resulta aconsejable la consideración de algún test estadístico [Her-2004a]. Hemos usado el test de Wilcoxon [Wil-1945; Sie-1988] debido a que es un test no paramétrico que no exige normalidad en la distribución. Lo hemos calculado usando la herramienta estadística R [R-2005] exigiendo una confianza de $0,95$. Para indicar diferencias significativas entre los resultados de dos algoritmos hemos establecido como algoritmo de referencia a IADEM-2 y hemos usado los símbolos $\oplus$ y $\ominus$. Con el símbolo $\oplus$ indicamos que el resultado es significativamente mejor que el obtenido por IADEM-2, con el símbolo $\ominus$ indicamos que el resultado es significativamente peor y si no usamos ningún símbolo indicamos que no existen diferencias significativas.

### 8.1.3. Conjuntos de datos usados en el estudio experimental

En este estudio experimental, a diferencia del llevado a cabo en el capítulo 4, sólo consideraremos conjuntos de datos grandes. El algoritmo que presentamos toma bloques de experiencias del conjunto de datos de entrenamiento y, aunque puede funcionar tomando bloques pequeños si el conjunto de datos es pequeño, está diseñado para trabajar con grandes volúmenes de datos (o flujos de datos). Hemos realizado experimentos en los que hemos considerado conjuntos de datos pequeños, y los resultados indican que el sistema multiclasificador MultiCIDIM-DS-CFC ofrece resultados aceptables y comparables a los de otros algoritmos en numerosos casos, pero exige una calibración individualizada del tamaño del bloque ($tam\_bloque$) que se debe utilizar para inducir cada clasificador básico. Además, si el conjunto es demasiado pequeño, también existe la posibilidad de que inducir cada uno de los 10 árboles de decisión con la décima parte del conjunto original no ofrezca clasificadores básicos suficientemente precisos, que es una de las propiedades necesarias para conseguir sistemas multiclasificadores precisos [Han-1990].

Por lo tanto, los conjuntos de datos empleados, y que resumimos a continuación, han sido los mismos que los empleados en la sección 4.5 exceptuando el conjunto "*Waveform*", debido a que presenta atributos continuos y CIDIM, el algoritmo que induce los clasificadores básicos, aún no está preparado para trabajar con ese tipo de atributos. Así, los conjuntos de datos usados son los siguientes:

- *Sint-1*: creado usando un árbol de decisión generado aleatoriamente (como se explicó en la sección 4.5). El problema definido antes de crear el árbol tiene 20 atributos con múltiples valores (entre 5 y 7) y 4 valores para el atributo de clase. El árbol generado aleatoriamente tiene una profundidad máxima de 5 niveles y un total de 398 hojas. El conjunto de datos ha sido generado sin incluirle ruido y tiene un tamaño de 1 millón de experiencias.

- *Sint-2*: el problema definido antes de crear el árbol tiene también 20 atributos con múltiples valores (entre 3 y 5) y 4 valores para el atributo de clase. El árbol generado aleatoriamente tiene una profundidad máxima de 5 niveles y un total de 100 hojas. El conjunto de datos ha sido generado sin incluirle ruido y tiene un tamaño de 100000 experiencias (100K). La peculiaridad de este conjunto de datos es que se ha generado evitando que las clases estuviesen balanceadas, en concreto, la cuarta clase sólo aparece alrededor del 3 % de las veces (cuando le correspondería un 25 %).

- *Sint-3*: el problema definido antes de crear el árbol tiene también 20 atributos con múltiples valores (entre 4 y 6) y 4 valores para el atributo de clase. El árbol generado aleatoriamente tiene una profundidad máxima de 7 niveles y un total de 663 hojas. El conjunto de datos ha sido generado incluyéndole un 15 % de ruido y tiene un tamaño de 1 millón de experiencias. Este conjunto de datos representa un conocimiento bastante más complejo que ninguno de los anteriores debido al gran tamaño del árbol y al ruido introducido.

- *LED*: este conjunto de datos ha sido creado con un generador que recibe el nombre de "*LED*" debido a que las experiencias reflejan el estado de un LED de 7 segmentos. Se define mediante 24 atributos

binarios (con valores 0 ó 1) de los cuales 17 son irrelevantes y los 7 restantes son los que se usan para determinar la clase, que puede tomar 10 valores diferentes (los 10 dígitos diferentes que presenta un LED de 7 segmentos). Hemos preparado un conjunto de datos con 1 millón de experiencias y con un 10 % de ruido usando el generador disponible en el repositorio de la UCI.

## 8.2. Evaluación de la mejora al clasificar mediante filtros correctores

La principal característica que deseamos evaluar es la mejora que supone la incorporación la clasificación mediante filtros correctores, abreviado como "CFC". Para ello hemos representado en la figura 8.1 la precisión alcanzada por los sistemas multiclasificadores MultiCIDIM-DS (que no usa CFC) y MultiCIDIM-DS-CFC (que sí usa CFC) considerando dos conjuntos de datos diferentes: uno con ruido (derecha) y otro sin él (izquierda).

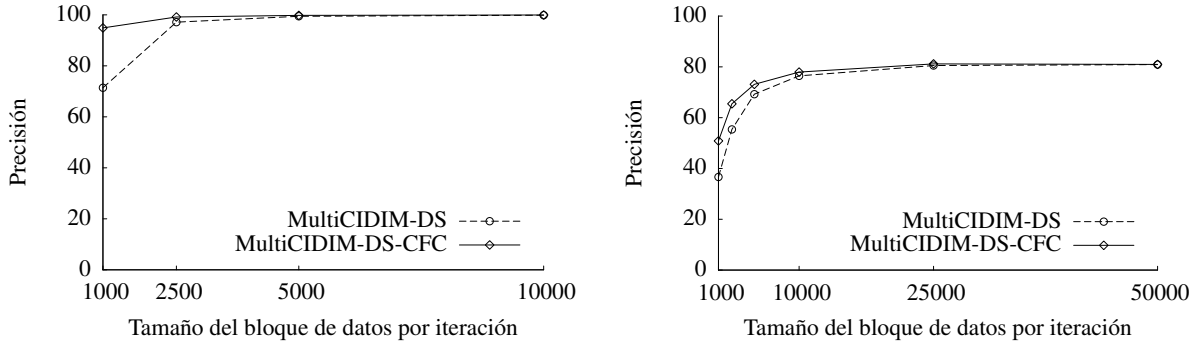



Figura 8.1: Precisión alcanzada por MultiCIDIM-DS y MultiCIDIM-DS-CFC usando conjuntos de datos sin ruido (izquierda) y con ruido (derecha).

Si observamos los resultados alcanzados al aplicar ambos algoritmos al conjunto de datos *Sint-1* (gráfica a la izquierda de la figura 8.1), en el que no hay presencia de ruido, podemos observar que el algoritmo que alcanza más precisión es el que usa los filtros correctores (MultiCIDIM-DS-CFC) y la mejora sobre el que no lo usa (MultiCIDIM-DS) se hace más patente cuanto menos preciso es el sistema multiclasificador. Al usar tamaños de bloque de datos más pequeños, el algoritmo de aprendizaje que induce los clasificadores básicos (en este caso CIDIM) consigue crear modelos menos precisos y eso repercute en la precisión global del sistema multiclasificador, pero es en esos casos en los que más se aprecia la ventaja que tiene el incluir la clasificación mediante filtros correctores. Usarlos siempre supone una ventaja, pero ésta es más evidente cuando el sistema multiclasificador no ha alcanzado los niveles máximos de precisión. Dado que es difícil asignar un valor adecuado al tamaño del bloque para todos los casos, usar los filtros correctores en cualquier caso parece bastante útil. En este caso concreto, aunque parece que a partir de un valor de 10000 para el tamaño del bloque la precisión no mejora, no es así, sigue mejorándose, y, a pesar de que la diferencia es pequeña en valor absoluto (por estar cerca del máximo que es 100), sigue siendo importante en términos relativos como puede observarse en la gráfica izquierda de la figura 8.2, donde se muestra la tasa de mejora de la precisión en términos relativos.

Considerando la precisión alcanzada para el conjunto de datos que presenta ruido, *Sint-3* (gráfica a la derecha de la figura 8.1), las conclusiones son similares: usar los filtros correctores mejora la precisión para todos los tamaños de bloque considerados, pero se hace más patente cuando dicho tamaño es más pequeño. La razón es la misma, la mejora introducida al emplear los filtros correctores se hace mayor cuanto más lejos está el sistema multiclasificador de la precisión alcanzable.

Analizando la figura 8.2 se puede observar, quizás más claramente, cómo disminuye la mejora introducida al clasificar mediante filtros correctores al aumentar el tamaño del bloque de experiencias que se usa para inducir los clasificadores básicos. La razón en sí no es el tamaño del bloque de experiencias, sino la

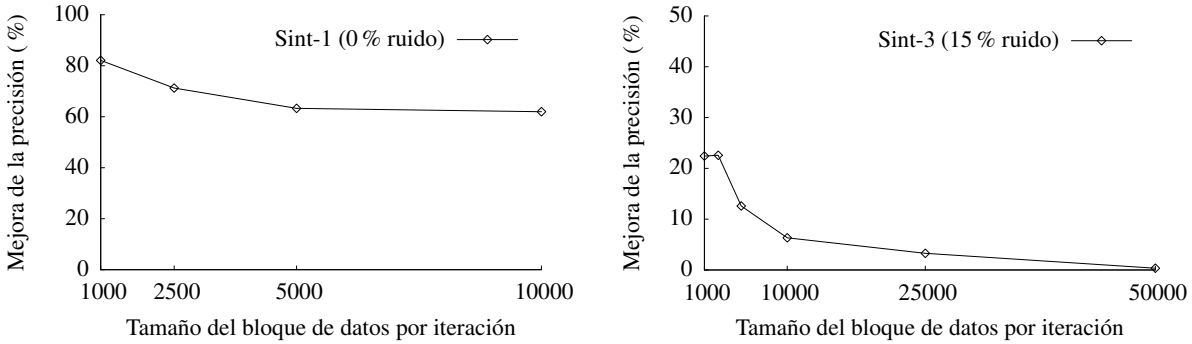



Figura 8.2: Tasas de mejoras de precisión alcanzadas por el algoritmo MultiCIDIM-DS-CFC sobre el algoritmo MultiCIDIM-DS usando conjuntos de datos sin ruido (izquierda) y con ruido (derecha).

precisión alcanzada por los clasificadores básicos al ser inducidos con ese conjunto de experiencias. Por lo tanto, aunque el uso de los filtros correctores parece aumentar la precisión bajo diferentes circunstancias, la mayor ventaja se puede esperar cuando se aplique a sistemas multiclasificadores que aún no hayan alcanzado su máxima precisión.

## 8.3. Evaluación con grandes conjuntos de datos

Una vez hemos visto que el uso de los filtros correctores mejora generalmente la precisión de los sistemas multiclasificadores, realizaremos un estudio en el que compararemos los resultados alcanzados por distintos algoritmos con los conjuntos de datos anteriormente descritos.

En el cuadro 8.1 hemos presentado los datos correspondientes a las medias y desviaciones típicas de los experimentos realizados en cada una de las validaciones cruzadas. Existen algunos algoritmos cuya ejecución no terminaba correctamente debido a las limitaciones impuestas al tamaño máximo de memoria disponible. Para poder usar sus resultados hemos ampliado dichos límites y los hemos indicado en la tabla marcándolos con asteriscos (‘*’). Estos algoritmos han sido los sistemas multiclasificadores no incrementales que usaban como algoritmo de aprendizaje el J48, es decir, Bagging-J48 y Boosting-J48. Al aumentar el límite de memoria hasta los 2GB, las ejecuciones han terminado (excepto en un caso), pero debemos resaltar la dificultad que tienen los algoritmos no incrementales para trabajar con grandes conjuntos de datos.

En el cuadro 4.5 se pueden observar diferentes resultados, que tienen distintos enfoques según estemos tratando los conjuntos de datos con ruido o sin ruido. Por ello, destacamos las características más relevantes, agrupándolas según la presencia de ruido, a continuación:

- observando los conjuntos de datos sin ruido (*Sint-1* y *Sint-2*):

  - los únicos algoritmos capaces de inducir modelos, en este caso sistemas multiclasificadores, que alcancen una precisión del $100\,\%$ son Bagging-J48 y Boosting-J48. Esto se debe a que el algoritmo de aprendizaje que induce los clasificadores básicos, J48, es capaz de inducir el árbol original usado para la generación del conjunto de datos (con muy ligeras diferencias de tamaño en el caso de *Sint-2*). Este aspecto ya se detectó en el anterior estudio experimental mostrado en la sección 4.5. Lo llamativo en los modelos inducidos por MultiCIDIM-DS y MultiCIDIM-DS-CFC para el conjunto de datos *Sint-1* es la alta precisión alcanzada, sobre todo con MultiCIDIM-DS-CFC, a pesar de que el tamaño medio de los árboles en el sistema multiclasificador es menor (incluso significativamente) que el del árbol original. Con esto puede apreciarse la ventaja de usar los filtros correctores, pero aún más, la precisión del multiclasificador inducido con MultiCIDIM-DS-CFC es siempre mejor que la obtenida con MultiCIDIM-DS y lo es de forma significativa.

| Datos | Algoritmo | Precisión | | Hojas | | Tiempo (en seg.) |
|---|---|---|---|---|---|---|
| Sint-1 | | | | | | |
| 0 % ruido | VFDTc | 99,79 ± 0,05 | ⊖ | 468,60 ± 7,59 | ⊖ | 18,78 |
| | IADEM2 | 99,997 ± 0,00 | ⊕ | 394,00 ± 0,00 | ⊖ | 28,77 |
| | Bagging-J48 | 100,00 ± 0,00 | ⊕ | 398,00 ± 0,00 | ⊖ | 275,29 |
| | Boosting-J48 | 100,00 ± 0,00 | ⊕ | 398,00 ± 0,00 | ⊖ | 37,03 |
| | MultiCIDIM-DS | 99,86 ± 0,06 | ⊖ | 345,05 ± 7,38 | | 45,19 |
| | MultiCIDIM-DS-CFC | 99,95 ± 0,02 | | 345,05 ± 7,38 | | 83,33 |
| Sint-2 | | | | | | |
| 0 % ruido | VFDTc | 94,48 ± 6,11 | ⊖ | 190,40 ± 52,17 | | 1,70 |
| desbalanc. | IADEM2 | 99,86 ± 0,04 | ⊖ | 241,00 ± 0,00 | ⊖ | 26,83 |
| | Bagging-J48 | 100,00 ± 0,00 | ⊕ | 109,45 ± 6,86 | ⊕ | 14,49 |
| | Boosting-J48 | 100,00 ± 0,00 | ⊕ | 104,50 ± 14,23 | ⊕ | 2,10 |
| | MultiCIDIM-DS | 99,78 ± 0,36 | ⊖ | 187,49 ± 91,37 | | 3,21 |
| | MultiCIDIM-DS-CFC | 99,92 ± 0,12 | | 187,49 ± 91,37 | | 4,22 |
| Sint-3 | | | | | | |
| 15 % ruido | VFDTc | 70,25 ± 6,38 | ⊖ | 1528,80 ± 325,61 | | 27,66 |
| | IADEM2 | 80,58 ± 2,96 | | 259,50 ± 105,92 | ⊕ | 51,49 |
| | Bagging-J48** | 84,14 ± 0,12 | ⊕ | 67626,04 ± 214,63 | ⊖ | 436,88 |
| | Boosting-J48** | 83,16 ± 0,13 | ⊕ | 134866,69 ± 218,84 | ⊖ | 619,55 |
| | MultiCIDIM-DS | 80,89 ± 0,99 | | 1354,19 ± 39,25 | | 55,75 |
| | MultiCIDIM-DS-CFC | 80,96 ± 1,07 | | 1354,19 ± 39,25 | | 80,88 |
| LED | | | | | | |
| 10 % ruido | VFDTc | 74,00 ± 0,11 | | 71,50 ± 1,35 | ⊕ | 13,84 |
| | IADEM2 | 74,01 ± 0,14 | | 12,10 ± 1,52 | ⊕ | 10,72 |
| | Bagging-J48** | 73,94 ± 0,13 | | 36150,17 ± 197,38 | ⊖ | 924,86 |
| | Boosting-J48** | Sin memoria | | | | |
| | MultiCIDIM-DS | 73,96 ± 0,22 | | 118,35 ± 5,27 | | 39,64 |
| | MultiCIDIM-DS-CFC | 73,98 ± 0,14 | | 118,35 ± 5,27 | | 68,49 |

Cuadro 8.1: Valores medios y desviaciones típicas de las validaciones cruzadas para los conjuntos de datos *Sint-1*, *Sint-2*, *Sint-3* y *LED*. El símbolo ⊕ indica que el resultado es significativamente mejor que el alcanzado por MultiCIDIM-DS-CFC y el símbolo ⊖ indica que es significativamente peor. Los algoritmos marcados con asteriscos han necesitado un incremento en el límite de la memoria disponible para poder ejecutarse (Bagging-J48** y Boosting-J48** han podido ser ejecutados tras aumentar el límite hasta 2GB).

Debemos considerar además que esta mejora se produce siendo el sistema multiclasificador bastante preciso, que es la circunstancia en la que menos se aprecian las ventajas del uso de los filtros correctores como hemos explicado en la sección anterior (sección 8.2).

- comparando el sistema multiclasificador que proponemos, MultiCIDIM-DS-CFC, con algoritmos incrementales que inducen un único modelo, VFDTc y IADEM-2, podemos observar que, salvo en un caso (conjunto de datos *Sint-1* con el algoritmo IADEM-2), la precisión alcanzada por el sistema multiclasificador es la mejor y lo es de forma significativa. Además, el tamaño medio de los árboles inducidos en el sistema multiclasificador es menor que los árboles inducidos por VFDTc o IADEM-2, y suele serlo de forma también significativa. Esta es una de las ventajas aportadas por el uso de CIDIM como algoritmo de aprendizaje básico: produce árboles precisos con un tamaño generalmente reducido.
- aunque el tiempo consumido por Bagging-J48 es muy superior al empleado por el resto de al-

goritmos, no debemos ver esto como un problema. Ya comentamos anteriormente que Bagging-J48 (y también Boosting-J48) usan el conjunto de datos completo para inducir cada uno de los clasificadores básicos, por tanto, no son comparables los tiempos consumidos por estos dos algoritmos con los tiempos empleados por el resto de algoritmos. Este razonamiento se repetirá para los conjuntos de datos en los que existe ruido.

- el tiempo empleado por MultiCIDIM-DS-CFC para inducir el sistema multiclasificador es mayor que el que necesita MultiCIDIM-DS, aunque es siempre menor del doble. Es lógico puesto que a la tarea de inducir los distintos clasificadores básicos, que es común a ambos algoritmos, el sistema que usa los filtros correctores (MultiCIDIM-DS-CFC) tiene que añadir la tarea de crear y entrenar dichos filtros correctores. Aún así, la sobrecarga de generar los filtros correctores y entrenarlos con las experiencias modificadas no suponen más tiempo del que emplea el algoritmo CIDIM en inducir los clasificadores básicos.

- observando los conjuntos de datos con ruido (*Sint-3* y *LED*):

  - los sistemas multiclasificadores inducidos por Bagging-J48 y Boosting-J48 son los más precisos con el conjunto de datos *Sint-3*. Lo que conviene destacar de los sistemas multiclasificadores generados por Bagging-J48 y Boosting-J48 es que los niveles de precisión tan altos que se alcanzan son debidos a que el algoritmo básico, J48, considera todo el conjunto de datos antes de inducir el clasificador básico. Al considerar tanta información, para lo que necesita que se aumente el límite de memoria hasta los 2GB si se desea que complete la inducción, la precisión es elevada, pero destaquemos también el elevado tamaño de los árboles que componen el sistema multiclasificador. Los sistemas clasificadores inducidos por Bagging-J48 y Boosting-J48 tienen un tamaño medio de los árboles de decisión entre $50$ y $100$ veces mayor que los inducidos por MultiCIDIM-DS-CFC con el conjunto de datos *Sint-3*, lo que los hace prácticamente incomprensibles para cualquier experto.
  - si nos fijamos en el conjunto de datos *LED* se aprecia que las precisiones son similares para todos los algoritmos, pero la diferencia en el tamaño de los árboles de decisión inducidos es considerable, como ocurría en el caso anterior. El tamaño medio de los árboles en el sistema multiclasificador inducido por Bagging-J48 es $300$ veces mayor que el inducido por MultiCIDIM-DS-CFC y la precisión es similar. La razón principal ya ha sido comentada en el punto anterior: Bagging-J48 no es un algoritmo incremental, consume todo el conjunto de experiencias para inducir cada uno de los clasificadores básicos, y éstos son bastante grandes.
  - las diferencias entre las precisiones alcanzadas por los algoritmos MultiCIDIM-DS y MultiCIDIM-DS-CFC existe, siendo favorable a MultiCIDIM-DS-CFC, pero no son significativas. Debemos recordar que, como se apuntó en la sección anterior (sección 8.2), las diferencias son más pequeñas cuanto más experiencias se usan para inducir cada clasificador básico y, en este caso, por ser un conjunto de datos con ruido, estamos usando $50000$ experiencias, un número lo suficientemente elevado para que los clasificadores básicos sean bastante precisos.
  - debemos observar la dificultad de los algoritmos no incrementales (Bagging-J48 y Boosting-J48) para afrontar grandes conjuntos de datos en los que hay ruido. Esa dificultad reside en la forma en que se procesa el conjunto de datos, lo que exige mayores requisitos de memoria (incluso con 2GB es posible que no terminen) para almacenar en memoria tanto el conjunto de datos como el modelo que se está induciendo.
  - el tiempo consumido por Bagging-J48 y Boosting-J48 sigue siendo muy superior al empleado por el resto de algoritmos, pero debemos recordar que no son comparables por la forma en que se inducen los clasificadores básicos. Los tiempos empleados por MultiCIDIM-DS-CFC para inducir los sistemas multiclasificadores siguen siendo mayores que los que necesita MultiCIDIM-DS, pero continúan siendo menores que el doble, como ocurría con los conjuntos de datos en los que no había ruido.

Habiendo presentado los características de forma individual, podemos realizar el estudio comparativo agrupando las características más relevantes en índices combinados que nos resuman en un único valor la calidad de los modelos. En el cuadro 8.2 hemos vuelto a mostrar los índices combinados calculados con los métodos de ponderación aditiva simple ($SAW$) y de ponderación aditiva relativa ($RAW$). Recordemos que las características elegidas para calcular el índice combinado siguen siendo la precisión, con un peso de $0,7$, y el tamaño del árbol, con un peso de $0,3$.

| Datos | Algoritmo | SAW | | RAW | |
|---|---|---|---|---|---|
| | | $IC$ | Puesto | $IC$ | Puesto |
| Sint-1 | | | | | |
| 0 % ruido | VFDTc | 0,92 | 6º | 0,00 | 6º |
| | IADEM2 | 0,96 | 3º | 0,87 | 3º |
| | Bagging-J48 | 0,96 | 4º | 0,87 | 1º |
| | Boosting-J48 | 0,96 | 4º | 0,87 | 1º |
| | MultiCIDIM-DS | 1,00 | 2º | 0,54 | 5º |
| | MultiCIDIM-DS-CFC | 1,00 | 1º | 0,82 | 4º |
| Sint-2 | | | | | |
| 0 % ruido | VFDTc | 0,83 | 6º | 0,11 | 6º |
| desbalanc. | IADEM2 | 0,83 | 5º | 0,68 | 5º |
| | Bagging-J48 | 0,99 | 2º | 0,99 | 2º |
| | Boosting-J48 | 1,00 | 1º | 1,00 | 1º |
| | MultiCIDIM-DS | 0,87 | 4º | 0,79 | 4º |
| | MultiCIDIM-DS-CFC | 0,87 | 3º | 0,81 | 3º |
| Sint-3 | | | | | |
| 15 % ruido | VFDTc | 0,66 | 4º | 0,90 | 4º |
| | IADEM2 | 1,00 | 1º | 1,00 | 3º |
| | Bagging-J48** | 0,01 | 5º | 0,15 | 5º |
| | Boosting-J48** | 0,01 | 6º | 0,00 | 6º |
| | MultiCIDIM-DS | 0,76 | 3º | 1,00 | 2º |
| | MultiCIDIM-DS-CFC | 0,76 | 2º | 1,00 | 1º |
| LED | | | | | |
| 10 % ruido | VFDTc | 0,75 | 2º | 0,92 | 2º |
| | IADEM2 | 1,00 | 1º | 1,00 | 1º |
| | Bagging-J48** | 0,70 | 5º | 0,00 | 5º |
| | Boosting-J48** | Sin memoria | | | |
| | MultiCIDIM-DS | 0,73 | 4º | 0,53 | 4º |
| | MultiCIDIM-DS-CFC | 0,73 | 3º | 0,70 | 3º |

Cuadro 8.2: Valores de los índices combinados ($IC$) calculados con los métodos *SAW* y *RAW* y su orden de preferencia para los conjuntos de datos *Sint-1*, *Sint-2*, *Sint-3* y *LED*. Los algoritmos marcados con asteriscos han necesitado un incremento en el límite de la memoria disponible para poder ejecutarse (Bagging-J48** y Boosting-J48** han podido ser ejecutados tras aumentar el límite hasta 2GB).

Los únicos cambios de orden que se observan entre los métodos de ponderación aditiva simple ($SAW$) y relativa ($RAW$) se producen en el conjunto de datos *Sint-1* entre Bagging-J48, Boosting-J48 y MultiCIDIM-DS-CFC, y en el conjunto de datos *Sint-3* entre IADEM-2, MultiCIDIM-DS y MultiCIDIM-DS-CFC. De nuevo, los valores calculados con el método de ponderación aditiva relativa parecen más lógicos si queremos que prevalezca la precisión como una característica más importante a la hora de considerar precisión y tamaño como los aspectos relevantes.

Centrándonos en el orden asignado por los métodos empleados, podemos señalar que el comportamien-

to de MultiCIDIM-DS-CFC, aunque no alcanza los puestos más altos cuando los conjuntos de datos no presentan ruido, consigue valores altos y cercanos a los mejores. En los casos en los que hay ruido sí parece conseguir mejores puestos. Un hecho interesante, aunque era de esperar después de ver los resultados del cuadro 8.1, es que MultiCIDIM-DS-CFC siempre consigue un puesto mejor que MultiCIDIM-DS. Decimos que era de esperar porque teniendo el mismo tamaño los clasificadores básicos, la precisión de MultiCIDIM-DS-CFC siempre es mayor que la de MultiCIDIM-DS.

Considerando lo expuesto en esta sección, en la que hemos estudiado el comportamiento del algoritmo MultiCIDIM-DS-CFC y lo hemos comparado con diferentes algoritmos usando distintos tipos de conjuntos de datos, podemos concluir que los sistemas multiclasificadores inducidos por él son similares al resto en algunos casos y mejores en otros, pero nunca presentan un mal comportamiento. Aun así, lo más destacado no es la comparación anterior realizada con otros algoritmos, sino que hemos visto cómo la incorporación de los filtros correctores en los sistemas multiclasificadores puede resultar provechosa a la hora clasificar.

## 8.4. Conclusiones derivadas del estudio experimental

En esta sección resumimos las principales características del algoritmo MultiCIDIM-DS-CFC, y las extrapolamos al uso de filtros correctores en general. Primeramente expondremos qué ventajas tiene e indicaremos para qué problemas puede ser útil su empleo, y a continuación veremos las dificultades que se pueden presentar y hacer que el resultado de su uso no sea el deseado.

### 8.4.1. Ventajas derivadas del uso de los filtros correctores

- Incremento de la precisión: en los experimentos que hemos presentado se puede comprobar cómo los valores de la precisión alcanzada al usar los filtros correctores para la clasificación son mejores que los observados cuando no se usan dichos filtros. Hemos comentado que esas mejoras se producen incluso en los casos en los que el sistema multiclasificador ya es bastante preciso y que las mayores mejoras se observan cuando el sistema multiclasificador aún no se ha aproximado a sus máximos niveles de precisión. Pero, en cualquier caso, el uso de la clasificación mediante filtros correctores (CFC) se plantea como algo aconsejable puesto que mejora la precisión (independientemente del nivel de precisión que tenga el sistema multiclasificador) y no se ha observado ningún caso en el que el uso de los filtros correctores que proponemos haya empeorado la precisión del sistema multiclasificador.

- Buena interpretabilidad de los clasificadores básicos: esta característica es propia del algoritmo MultiCIDIM-DS, bien use filtros correctores o no, debido a que los árboles de decisión inducidos por CIDIM son bastante pequeños y suelen ser menores que los inducidos por otros algoritmos de aprendizaje (como J48) [Ram-2005c]. Por esa razón los sistemas multiclasificadores generados con CIDIM como algoritmo de aprendizaje para la inducción de los clasificadores básicos suelen ser más comprensibles que los inducidos por otros algoritmos. Es cierto que en determinados casos existen algoritmos incrementales que inducen modelos individuales aún más sencillos, como es el caso de IADEM-2, pero eso se debe a que la estrategia para inducir el árbol de decisión es diferente y no son directamente comparables (los modelos individuales no tienen la misma estructura que los modelos múltiples).

### 8.4.2. Limitaciones actuales en el uso de filtros correctores

Para terminar el capítulo presentamos algunas limitaciones del algoritmo MultiCIDIM-DS-CFC. Estas limitaciones, que ya hemos identificado, formarán parte de nuestras futuras líneas de investigación, junto con otras que comentaremos brevemente en el resumen del capítulo 9. Las limitaciones actuales en el uso de MultiCIDIM-DS-CFC pueden provenir de:

- Presencia de atributos continuos: esta limitación es propia del algoritmo MultiCIDIM-DS-CFC (y de MultiCIDIM-DS) y se debe a que CIDIM, el algoritmo de aprendizaje básico, no es capaz de tratar directamente atributos continuos (sería preciso una discretización previa). No poder trabajar directamente con atributos continuos no viene determinado por el esquema definido para usar filtros correctores, sino por el tipo de implementación concreta que se elija para el algoritmo de aprendizaje básico y para el algoritmo que induce los filtros correctores. Si ambos son capaces de trabajar con atributos continuos sin discretización previa, el sistema multiclasificador también los podrá usar directamente.

- Incremento en los recursos necesarios: la memoria necesaria para almacenar el sistema multiclasificador y el tiempo consumido se ven aumentados al usar los filtros correctores. Es lógico puesto que, además de almacenar los clasificadores básicos, se induce, para cada uno de ellos, otro clasificador incremental que estudiará su comportamiento. Aún así, en los experimentos realizados la memoria nunca ha sido un inconveniente, pudiendo completar la inducción en todos los casos y el único aspecto que se ha visto empeorado ha sido el tiempo consumido, que, a pesar de ser mayor, nunca ha pasado del doble empleado por el modelo que no usa filtros correctores.

- Presencia de cambio de concepto en el conjunto de datos: el cambio de concepto no centra nuestro interés en esta tesis, aunque está cobrando importancia también en el ámbito de los sistemas multiclasificadores [Wan-2003; Sch-2007] y es una de las futuras líneas de investigación que abordaremos. En este apartado debemos decir que, considerando el cambio de concepto, se espera que el comportamiento del algoritmo MultiCIDIM-DS-CFC sea mejor que el que pueda presentar el algoritmo IADEM-2, puesto que un cambio de concepto llevará a que los nuevos clasificadores que se induzcan para MultiCIDIM-DS-CFC con cada nuevo bloque de experiencias reemplacen a los anteriores, renovando así los conceptos almacenados en el sistema multiclasificador y adaptándose más rápidamente al nuevo concepto. En el siguiente capítulo, en el que se detallan más extensamente las futuras líneas de investigación, comentaremos algunas ideas para abordar este problema.

# Capítulo 9

# Resumen de la parte II
# Summary of part II

En esta segunda parte de la tesis hemos tratado el aprendizaje incremental basado en sistemas multiclasificadores, proponiendo en primer lugar un nuevo algoritmo basado en CIDIM que es capaz de extraer conocimiento de grandes conjuntos de datos (o de flujos de datos) y al que hemos llamado MultiCIDIM-DS [Cam-2007a]. Además de haber desarrollado este nuevo algoritmo, hemos propuesto un método para mejorar la clasificación que realizan los sistemas multiclasificadores cuando se usan para el aprendizaje incremental. Para ello hemos definido el concepto de "*clasificación mediante filtros correctores*" que se ha sido usado en el diseño de otro nuevo algoritmo, basado en el anterior, al que hemos llamado MultiCIDIM-DS-CFC [Cam-2007a]. A continuación presentamos un breve resumen de los temas abordados en esta parte de la tesis incluyendo las aportaciones realizadas y las futuras líneas de investigación.

Los sistemas multiclasificadores han supuesto un enfoque bastante utilizado para conseguir clasificadores más precisos, aunque a costa de sacrificar cierta comprensibilidad del modelo. En ellos se combinan diferentes modelos básicos, ya sean del mismo tipo (homogéneos) o de diferente tipo (heterogéneos), para extraer conocimiento de los diversos subconjuntos que forman el espacio de experiencias. La generación de estos modelos básicos, o clasificadores básicos, puede realizarse de muy distintas maneras, como se observa en la gran cantidad de trabajos desarrollados [Bre-1996; Fre-1996; Die-2000a]. En la sección 6.1 hemos presentado los fundamentos básicos que hacen de este enfoque una herramienta tan útil y hemos realizado un breve recorrido detallando distintos métodos para generar sistemas multiclasificadores, destacando entre ellos *bagging* [Bre-1996] y *boosting* [Fre-1996]. La sencillez que caracteriza al algoritmo *bagging* no está reñida con las mejoras que alcanza y su metodología es similar a la que hemos empleado en el desarrollo de los sistemas multiclasificadores que hemos propuesto, pero con la salvedad de que han sido adaptados para poder realizar un aprendizaje incremental. En esa misma sección 6.1 también se han presentado diversos métodos para combinar los modelos que componen los sistemas multiclasificadores. Éstos pueden ser diferenciados de distintas formas: según "*voten*" usando las predicciones mayoritarias o "*fusionen*" los vectores de predicción, y según usen todos los clasificadores básicos ("*métodos estáticos*") o sólo los más relevantes en cada caso ("*métodos dinámicos*"). Este último enfoque, que sólo usa los modelos relevantes, ha sido especialmente útil para desarrollar la clasificación usando los filtros correctores que hemos propuesto. Continuando con los sistemas multiclasificadores, y antes de introducir las propuestas realizadas para extender su uso a grandes conjuntos de datos (o flujos de datos), se han presentado, en la sección 6.2, los sistemas multiclasificadores basados en CIDIM [Ram-2005c] porque han servido de referencia para definir los dos algoritmos propuestos. La particularidad de estos sistemas multiclasificadores, y particularmente la del sistema FE-CIDIM [Ram-2006], es la alta precisión que se consigue utilizando sistemas cuyos clasificadores básicos son árboles de decisión con una elevada precisión y cuyo tamaño es bastante reducido.

Una vez se han presentado los conceptos básicos de los sistemas multiclasificadores y se han descrito algunos ejemplos concretos, en la sección 6.3 se describen las modificaciones que se les pueden hacer a estos métodos para conseguir trabajar con grandes conjuntos de datos y permitirles realizar un aprendizaje

incremental. Uno de los enfoques más sencillos es no tomar todas las experiencias existentes en el conjunto de datos, lo que podría producir complicaciones debido a la demanda de excesivos recursos, sino tomarlas por bloques [Str-2001]. El conjunto de datos se convierte así en una fuente de experiencias de la que se van tomando secuencialmente bloques del mismo tamaño, que son usados sucesivamente para inducir diferentes clasificadores básicos que se combinarán en el sistema multiclasificador. Este enfoque tan simple es el que hemos usado como punto de partida para los algoritmos que proponemos debido a que reduce los requisitos de memoria a una cantidad bastante limitada, independientemente del tamaño del conjunto de datos, y a que ofrece unos resultados de bastante calidad. El primer sistema multiclasificador que hemos propuesto, haciendo uso del enfoque anteriormente descrito, es el algoritmo llamado MultiCIDIM-DS. En la sección 6.4 se describe este sistema multiclasificador para aprendizaje incremental basado en el algoritmo CIDIM y también se realiza un estudio de cómo se comporta ante diferentes situaciones dependiendo de cómo haya sido configurado.

Tomando el anterior enfoque para desarrollar sistemas multiclasificadores con carácter incremental se describe, en la sección 7.1, una aportación bastante interesante para mejorar la clasificación alcanzada por estos sistemas mediante el uso de lo que hemos llamado "*filtros correctores*". La idea de usar filtros correctores es similar a la propuesta por Ortega [Ort-1996] para sistemas multiclasificadores clásicos, en los que una serie de modelos adicionales tienen la función de aprender bajo qué condiciones es mejor usar unos clasificadores básicos u otros. Con esta información es posible realizar predicciones más orientadas considerando qué características tiene la observación concreta cuya clase se quiere predecir. La principal carencia que ofrece este enfoque es que no está diseñado para ser usado en ámbitos que requieren un aprendizaje incremental. En el trabajo de Street y Kim [Str-2001], que como ya hemos indicado sí tiene carácter incremental, se menciona la posibilidad de ejercer algún tipo de control sobre los clasificadores que intervienen en la predicción con la misma intención de Ortega, es decir, buscando la posibilidad de hacer "*dinámico*" el método de combinación de predicciones. Pero, según su propia experiencia, parece que no encontraron ningún método que supusiese una mejora. Esa intención de mejora es la que hemos recogido en nuestro trabajo consiguiendo algunos resultados positivos. Lo que se propone al usar los filtros correctores para la clasificación es considerar los mejores clasificadores básicos que deben ser combinados en cada caso concreto para obtener la mejor predicción posible. El objetivo de este enfoque es que sólo intervengan los clasificadores básicos que tengan conocimiento fiable. La determinación de cuáles son esos mejores clasificadores la realiza otro modelo adicional (filtro corrector) que ha sido entrenado para extraer conocimiento sobre las condiciones en las que mejor se comporta cada clasificador básico. Lo único que se necesita para inducir ese modelo, cuando se usa para realizar aprendizaje incremental, es que sea inducido por un algoritmo que también sea incremental.

En la sección 7.2 se describe una implementación concreta de la idea de usar filtros correctores, usando el algoritmo CIDIM para inducir los clasificadores básicos que formarán el sistema multiclasificador y aprovechando el algoritmo IADEM-2 para caracterizar a los modelos inducidos por CIDIM. Así, se obtiene un sistema multiclasificador en el que los clasificadores básicos son precisos y diferentes entre ellos (requisitos necesarios para conseguir un buen sistema multiclasificador [Han-1990]) y de reducido tamaño, al tiempo que se dispone de un mecanismo para elegir los mejores modelos para clasificar en cada momento la observación que se esté considerando.

Después de implementar los algoritmos propuestos, se ha desarrollado un extenso estudio experimental (capítulo 8) en el que se han comparado los resultados obtenidos con los alcanzados mediante otros algoritmos y, sobre todo, se ha puesto de manifiesto la mejora que se alcanza al introducir los filtros correctores en un sistema multiclasificador. En la comparación con otros algoritmos debemos decir que los resultados han sido bastante satisfactorios, siendo, generalmente, similares al resto. La mejora apreciada al incorporar el uso de los filtros correctores se ha revelado de forma clara al comparar los resultados alcanzados por el algoritmo MultiCIDIM-DS, que no usa los filtros, y MultiCIDIM-DS-CFC, que sí los usa. Esta mejora es más pronunciada cuanto menos preciso sea el sistema multiclasificador al que se le aplica, y, en los experimentos realizados, nunca se ha observado un empeoramiento al usar los filtros correctores.

Entre las principales características que se pueden observar de las aportaciones que hemos introducido

se pueden destacar una serie de ventajas. La primera sería el hecho de que los sistemas multiclasificadores inducidos por el algoritmo MultiCIDIM-DS son bastante precisos y más sencillos de comprender puesto que los clasificadores básicos que lo componen son más simples que los inducidos por otros algoritmos. En segundo lugar, debemos mencionar el incremento de precisión alcanzado al incluir los filtros correctores para controlar la fusión de las predicciones dadas por los diferentes clasificadores básicos. Esta nueva predicción, en la que se emplea únicamente la información más relevante, está mucho más orientada a resolver el caso concreto que se le presenta al sistema multiclasificador y permite alcanzar resultados más precisos. Por último, como aspecto más destacado, tenemos que el uso de filtros correctores no está limitado a la implementación concreta que hemos presentado, sino que puede ser aplicado a cualquier sistema multiclasificador. Enlazando con esta última ventaja debemos mencionar que una de nuestras futuras líneas de investigación es la inclusión de este enfoque en otros algoritmos, para evaluar cómo les afecta y qué condiciones son las que más peso tienen a la hora de conseguir una mayor mejora.

Tampoco debemos olvidar que el uso de los filtros correctores lleva acompañado una serie de limitaciones, algunas de las cuales esperamos abordar en nuevos trabajos de investigación. Entre dichas limitaciones podemos citar el mayor coste computacional que conlleva (debido a que hay que entrenar nuevos modelos adicionales) y un tratamiento del cambio de concepto que podría ser claramente mejorable. El cambio de concepto, como indicamos anteriormente, no se ha incluido en esta tesis como uno de los temas fundamentales, pero es bien sabido que recientemente está cobrando gran interés en la comunidad científica, incluso en el campo de los sistemas multiclasificadores [Wan-2003; Sch-2007]. El uso de sistemas multiclasificadores para realizar aprendizaje incremental se ve extendido de una forma natural con la capacidad para tratar problemas en los que exista cambio de concepto: basta con eliminar los clasificadores básicos que no se correspondan con el concepto actual y sustituirlos por nuevos clasificadores. El problema que puede surgir es el tiempo que se tarde en sustituir todos los clasificadores que ya no son adecuados, tiempo durante el cual el sistema multiclasificador es muy posible que no sea capaz de realizar predicciones correctas. Esta limitación sería fácilmente superable y se podrían eliminar los clasificadores básicos que ya no sean correctos, de forma inmediata, induciendo los filtros correctores mediante algoritmos de aprendizaje incremental capaces de detectar cambio de concepto. Al hacerlo así, en cuanto un filtro corrector detecte que el clasificador básico que está supervisando deja de clasificar del mismo modo en que se hacía antes, puede avisar de un posible cambio de concepto y ser eliminado del sistema multiclasificador. Esta idea que acabamos de comentar será otra de nuestras futuras líneas de trabajo para mejorar los sistemas multiclasificadores para aprendizaje incremental.

**Summary of part II**

*In this second part of the thesis, we have studied the incremental learning approach from the perspective of multiple classifier systems and we have proposed a new algorithm, based on CIDIM [Ram-2005c], that has been designed to extract knowledge from large datasets (or from data streams). It is called MultiCIDIM-DS [Cam-2007a]. But, more important than the proposal of this algorithm is the new method developed to improve the accuracy of the multiple classifier systems by using "*correction filters for classification*" (CFC). This new method has been used to implement the algorithm called MultiCIDIM-DS-CFC [Cam-2007a]. Next we summarize the topics studied in this part of the thesis including the contributions proposed and future research.*

*The multiple classifier systems have represented one approach widely used to induce more accurate classifiers, although the comprehensibility of the model decrease a little. They combine different basic models, being of the same type (homogeneous) or being of different types (heterogeneous), to extract knowledge from the diverse subsets that form the search space. The methods used to create the different basic classifiers that are combined in the multiple classifier systems are very diverse and many strategies can be considered [Bre-1996; Fre-1996; Die-2000a]. In section 6.1 we present the basic ideas that made this approach so useful and we briefly describe several algorithms used to induce multiple classifier systems, emphasizing*

*bagging [Bre-1996] and boosting [Fre-1996]. The simplicity that characterize the bagging algorithm is not incompatible with the improvements achieved by it, and its methodology is very similar to the one used in the design of the algorithms that we have proposed, but considering that these are qualified to do an incremental learning.*

*In the same section 6.1 some methods to combine the predictions given by the basic classifiers are presented too. These methods can be grouped in different categories depending on different criteria. If we consider how the information given by every basic classifiers is used, we have methods that "*vote*" using the majority class predicted by every classifier, or methods that "*fuse*" the prediction vectors to create a new one. If we consider which basic classifiers give their information to be combined, we have "*static methods*" that use all the basic classifiers and "*dynamic methods*" that only use the classifiers that have relevant information for the concrete case that is being considered. The last approach, the one that only uses the basic classifiers with relevant information, will be specially useful to design correction filters to classify. Before presenting the new algorithms for incremental learning based on CIDIM, in section 6.2, we have presented our previous non-incremental works in this area. The FE-CIDIM [Ram-2006] algorithm is described because it is useful to understand some qualities that can be observed in the new incremental algorithms based on CIDIM: high accuracy, and relatively simple comprehensibility of the model.*

*Once we have described the basic concepts of the multiple classifier systems, and we have presented some concrete examples, we explain, in section 6.3, how these systems can be extended to deal with large datasets using an incremental approach. One of the most simplest ways of doing that extension is taking batches of examples [Str-2001] instead of taking the whole dataset. Thus, the dataset becomes a source of examples from where the algorithms take equal-sized batches sequentially. Every batch is used by the base learning algorithm to induce one basic classifiers that will be combined in the multiple classifier system. This is the approach that we have used as our starting point to design the new algorithms proposed because the memory requirements are minimized (independently of the dataset size) and the results are of high quality (high accuracy and low complexity). The first multiple classifier system that we propose, using a simplified approach, is the achieved by the algorithm MultiCIDIM-DS. In section 6.4 this algorithm is described and some experimental results are presented to study the behaviour of MultiCIDIM-DS under diverse circumstances.*

*Considering the previous approach used to develop multiple classifier systems with an incremental methodology, we present, in section 7.1, one contribution that can be very useful to every multiple classifier system (not only for the algorithm proposed). The new method uses what we have called "*correction filters*". The aim pursued with these filters is similar to the one pursued by Ortega [Ort-1996] over classical multiple classifier systems. He included new classifiers called "*referee predictors*" that provided a mechanism for deciding which existing base classifiers should be combined to form the prediction. Those decisions where made using the knowledge extracted by the referee predictors, because they know which classifiers performs well in diverse situations. But this method can not be used for incremental learning. In the work proposed by Street and Kim [Str-2001], that had an incremental approach, they tried to use some kind of "*gated voting*" procedure. The idea was similar to the one used by Ortega, but they observed «no consistent improvement over simple voting». We have taken up again the intention of improving the prediction process used by the multiple classifier systems for incremental learning, and we have achieved positive results. Our proposal shares the same philosophy of "*referee predictors*" because it tries to know when a concrete basic classifier can be expected to show a good performance. But the difference lies in our incremental approach. We induce our "*correction filters*" using incremental learning algorithms. Thus, every new example (present in the batches) can be used by the correction filters to continue the learning.*

*In section 7.2 we concrete the previous idea. CIDIM is the algorithm used to induce the basic classifiers and IADEM-2 is the incremental algorithm used to induce the correction filters. This concrete implementation has been called MultiCIDIM-DS-CFC. Thus, the new algorithm shows some qualities taken from CIDIM and IADEM-2. The basic classifiers that compound the system will be accurate and diverse (requirements needed to get a good multiple classifier system [Han-1990]), and they will be small. In addition, by using correction filters, we can improve the prediction process because only the relevant basic classifiers*

*would be combined to form the global prediction vector.*

*After describing the new algorithms proposed, we have studied their performance and the results obtained have been presented in chapter 8. We have compared the systems induced by MultiCIDIM-DS and MultiCIDIM-DS-CFC with other algorithms (incremental algorithms and multiple classifier systems). But more important is the improvement shown in the prediction accuracy when correction filters are used. Considering the comparison with other algorithms we can say that the results are quite satisfactory because, in general terms, the systems induced by our algorithms are comparable with the rest of models. The improvement observed when using correction filters is clearly evident when we compare the results achieved by MultiCIDIM-DS (without filters) and MultiCIDIM-DS-CFC (with filters). The improvement appeared in every experiment, but it was higher when the accuracy achieved by the multiple classifier system (without filter) was lower.*

*Some advantages can be emphasized from the characteristics watched in the contributions. The first one would be the accuracy of the models induced by MultiCIDIM-DS combined with the simplicity of the system, what facilitates its comprehensibility. Secondly, the improvement of the prediction process using correction filters leads to a higher accuracy of the system. Finally, the flexibility of the use of correction filters is the most important aspect, because this approach can be used in a wide variety of multiple classifier systems. Taking this into account, we must note that one of our future researches is oriented towards the inclusion of this approach in other systems. Thus, we would try to evaluate which conditions are the most relevant to obtain a better improvement of the accuracy.*

*We have talked about advantages, but the algorithms that we have proposed also present some limitations. Some of them are intended to be overcome in future works. Between the limitations we must mention a higher computational cost (because new models need to be induced) and a poor treatment of the concept drift. Concept drift, as we have explained, has not been included in the thesis as a central topic, but it is getting great relevance in the areas of Machine Learning and Data Mining. This relevance can be also seen in the field of multiple classifier systems [Wan-2003; Sch-2007]. This kind of systems can be naturally extended to deal with concept drift, because basic classifiers that represent obsolete knowledge can be easily substituted by new basic classifiers induced with new batches of examples. The main problem is how much time is spent until every old basic classifier is replaced. During that interval of time, the multiple classifier system has new and old concepts, and the prediction can be potentially wrong. This limitation can be overcome if we use correction filters induced by incremental algorithms with the ability of detecting concept drift. If we use that kind of incremental algorithm to induce the correction filters, they can check continuously the kind of concepts learnt by every basic classifiers. But the most important, when they detect some change in the concepts that have been learnt, they can warn to the multiple classifier system and remove the obsolete classifiers immediately. This idea is another research line that we will consider in the future.*

# Parte III

# Conclusiones

# Capítulo 10

# Conclusiones y Futuras Líneas de Trabajo
# Conclusions and Future Work

En esta tesis hemos presentado distintas aportaciones en el ámbito del aprendizaje incremental dirigidas a mejorar diversos aspectos de diferentes algoritmos y métodos. En este capítulo final recogemos las principales conclusiones que se han ido presentando a lo largo de la tesis y resumimos las futuras líneas de investigación por las que tenemos previsto continuar.

*In this thesis we have presented some contributions to the field of incremental learning. They have been developed to improve different characteristics of diverse methods and algorithms. In this final chapter we summarize the most important conclusions that have been presented through this thesis and we enumerate our main future research lines.*

## 10.1. Conclusiones // Conclusions

Esta tesis ha sido dividida en dos grandes bloques para presentar las aportaciones realizadas agrupándolas según el enfoque utilizado. En primer lugar (parte I) hemos introducido un nuevo algoritmo incremental que se basa en el uso de cotas de concentración para extraer conocimiento de grandes conjuntos de datos (o flujos de datos). Este primer enfoque se ha visto completado por un segundo (parte II) en el que se ha propuesto un método para mejorar el aprendizaje de los sistemas multiclasificadores con carácter incremental.

El nuevo algoritmo incremental que hemos propuesto en la primera parte y que se basa en el uso de cotas de concentración se llama IADEM-2. Se trata de un algoritmo que no usa ningún modelo de memoria para almacenar las experiencias y que representa el conocimiento extraído mediante árboles de decisión. Como consecuencia, los únicos requisitos de memoria necesarios son aquellos que permitan almacenar el modelo que se está induciendo. El algoritmo ha sido diseñado para muestrear con reemplazamiento las experiencias del conjunto de datos (o flujo de datos) del que se desea obtener conocimiento, haciendo al algoritmo independiente del número de experiencias que formen dicho conjunto de datos. Utilizando la información almacenada en el árbol de decisión que se está induciendo y combinándola con los márgenes de error que ofrecen las cotas de concentración, el algoritmo IADEM-2 dispone de un conocimiento muy útil: las estimaciones de error cometido por el árbol de decisión para el mejor y el peor caso. Estas estimaciones son de gran utilidad para el funcionamiento del algoritmo y pueden ser aprovechadas por el usuario para obtener información adicional sobre el conocimiento extraído. Además, dichas estimaciones permitirán a los usuarios fijar unos requisitos mínimos en cuanto a la calidad exigida al modelo que se induzca.

El algoritmo IADEM-2 se ha diseñado para trabajar en gran diversidad de situaciones. Está especialmente preparado para extraer conocimiento de grandes conjuntos de datos, aunque, como se ha podido comprobar, no presenta inconvenientes cuando los conjuntos de datos son pequeños. Esto viene motivado por el uso del muestreo con reemplazamiento para obtener las experiencias con las que se inducirá el modelo

deseado. Los conjuntos de datos con los que se trabaje pueden venir definidos tanto con atributos categóricos como con atributos continuos, ya que el algoritmo está preparado para ello. Otra circunstancia que puede darse, pero que no provoca un funcionamiento anómalo del algoritmo, es la presencia de ruido en el conjunto de datos, puesto que dispone de mecanismos que le sirven para extraer la mayor cantidad de conocimiento antes de dar por finalizada la inducción del árbol de decisión. Este algoritmo también incorpora el uso de hojas funcionales que le permitan realizar predicciones más precisas usando la información más relevante disponible para cada caso. Como particularidad adicional, al vector con la predicción estimada le acompañan vectores que delimitan (con la confianza requerida) los márgenes de error de cada estimación.

Uniendo todas estas características en un único algoritmo, nos permite presentarlo como un buen candidato para ser usado cuando se trate de extraer conocimiento siguiendo un enfoque incremental. Sus bondades han sido puestas de manifiesto en los diferentes experimentos que se han realizado, destacando sobre todas su estabilidad en cuanto a los altos niveles de precisión alcanzados, al tamaño de los árboles de decisión generados y al número de experiencias necesarias para inducir el modelo que refleje el conocimiento extraído.

En la segunda parte de la tesis hemos abordado el aprendizaje incremental en el ámbito de los sistemas multiclasificadores y también hemos realizado algunas aportaciones. La primera ha sido incluir el carácter incremental en un sistema multiclasificador basado en el algoritmo CIDIM. Como resultado de la incorporación de un enfoque incremental que permita extraer conocimiento de grandes conjuntos de datos (o flujos de datos) con sistemas multiclasificadores previos, desarrollamos el algoritmo MultiCIDIM-DS. Este método procesa las experiencias disponibles en el conjunto de datos tomándolas por bloques, de forma que los requisitos de memoria son drásticamente reducidos, limitándose a los que son necesarios para almacenar el sistema multiclasificador que se está induciendo y el bloque considerado en cada paso.

Además de dotar al anterior sistema multiclasificador de la posibilidad de realizar un aprendizaje incremental, también hemos propuesto un nuevo método que permite mejorar la precisión alcanzada por este tipo de sistemas. La utilización concreta de este método para extender el algoritmo MultiCIDIM-DS ha permitido desarrollar un nuevo algoritmo llamado MultiCIDIM-DS-CFC, pero lo más interesante de este método es que su uso no está restringido a ningún sistema multiclasificador en concreto, sino que puede ser utilizado de forma mucho más general. La idea fundamental en la que se basa este método es poder realizar las predicciones combinando únicamente los clasificadores básicos que disponen de información útil en cada momento. Para ello se ha introducido el concepto de "*filtro corrector*" cuya función es informar al sistema multiclasificador de cuáles son los clasificadores básicos que deben combinarse para alcanzar la mejor predicción posible. Estos filtros correctores son modelos que se inducen de forma incremental, y que tratan de aprender qué subconjunto del espacio de experiencias es el que mejor ha aprendido cada clasificador básico.

Del estudio experimental realizado con los nuevos sistemas multiclasificadores propuestos sólo podemos concluir que su comportamiento es comparable con los que muestran otros algoritmos, pero existe una circunstancia que sí merece ser destacada y es la constatación de que el uso de filtros correctores mejora la precisión alcanzada.

Para terminar, debemos indicar que los diferentes algoritmos propuestos en las dos partes de la tesis tienen enfoques incrementales pero, dependiendo del problema que se esté tratando, puede ser más útil usar un enfoque u otro. Entre los diversos aspectos que pueden influir tenemos el tamaño del conjunto de datos: si éste no es demasiado grande, puede ocurrir que los algoritmos que inducen sistemas multiclasificadores no induzcan modelos completos (por falta de experiencias para inducir todos los clasificadores básicos que se hayan solicitado), haciéndose más aconsejable el uso del algoritmo IADEM-2. También es aconsejable usar este último algoritmo si se consideran atributos continuos, puesto que el algoritmo de aprendizaje utilizado por los sistemas multiclasificadores (CIDIM) aún no puede tratar este tipo de atributos de forma directa. Si lo que se desea es tener un algoritmo que trabaje de forma continuada procesando todas las experiencias que se vayan registrando, el enfoque más adecuado sería el que usa los sistemas multiclasificadores (MultiCIDIM-DS-CFC) puesto que toma bloques iterativamente y no se detiene aunque el aprendizaje se haya estabilizado (como ocurriría con el algoritmo IADEM-2). Esto permitiría, además, una mayor adaptabilidad de los modelos a los nuevos conceptos que pudiesen aparecer debido a que MultiCIDIM-DS-CFC puede renovar un clasificador básico después de observar cada bloque de experiencias.

**Conclusions**

*This thesis has been divided in two main parts to present the different contributions. Thus, we have grouped them according to the used approach. In the first part (part I) we have introduced a new incremental algorithm based on concentration bounds. This first contribution has been completed with a second approach (part II) where we have proposed a new method to improve the incremental learning using multiple classifier systems.*

*In the first part of the thesis we have presented a new incremental algorithm based on concentration bounds. It has been called IADEM-2. It is an incremental algorithm with no example memory which knowledge base is represented using decision trees. IADEM-2 receives examples to learn from, but once they are processed they are forgotten. No example is saved, only the relevant information for IADEM-2 is kept using counters stored in the decision tree. Thus, the memory requirements of this algorithm only depend on the size of the decision tree and its associated counters. This way of processing data lets us deal with any kind of dataset or data stream independently of size. The examples used by the algorithm are sampled with replacement from a dataset (or they can be taken directly from a data stream). Using the information stored in the decision tree and calculating the margins of error provided by the concentration bounds, the algorithm IADEM-2 has a very useful information. It has an estimation of the error of the decision tree that is being induced for the best case and for the worst one. These estimations are very useful for the execution of the algorithm and they can be used to provide more information to the user. Furthermore, with those estimations the users can set the minimum level of quality that must have the induced model.*

*IADEM-2 has been designed to work in a wide variety of situations. It is specifically prepared to extract knowledge from very large datasets, although it has no problem when the datasets are small (as has been shown in the experimental results). This property is due to the kind of mechanism used to get the examples. It uses sampling with replacement and it will continue taking new examples while the induced model has not reached the desired precision. Datasets used in the process can be defined by categorical or continuous attributes, because different kinds of expansions can be considered. The execution of the algorithm does not present an anomalous behaviour even in the presence of noise because some methods have been incorporated to extract as much knowledge as possible. Another relevant point for IADEM-2 is the ability of using functional leaves that produce more accurate results when the model is used to predict.*

*Considering all these advantages, we can present IADEM-2 as a good algorithm when it is necessary to extract knowledge using an incremental approach. Its qualities have been shown in the experimental section and we must emphasize the stability as one of the main qualities. The behaviour is quite stable when we consider the accuracy, the size of the tree or the number of examples needed to induce the model.*

*In the second part of the thesis we have tackled the field of multiple classifier systems for incremental learning and we have also contributed with some ideas and developments. One of the new algorithms proposed includes the incremental ability to a previous multiple classifier system based on CIDIM. As result we get the algorithm MultiCIDIM-DS that can extract knowledge from very large datasets. It takes batches of examples from the dataset in a sequential way and minimizes the requirements of memory needed to store the dataset.*

*In addition, this new extension of capabilities proposed in MultiCIDIM-DS have been completed with the development of a new method. It has been designed to improve the accuracy achieved by multiple classifier systems for incremental learning. This method is based in what we have called "*correction filters*" but its use is not restricted to the previous algorithm. The proposed filters provide a mechanism for deciding which existing base classifiers should be combined to form the prediction. They are induced by incremental algorithms using the training examples and the classifications made by the basic classifiers. Correction filters have been used to create the algorithm MultiCIDIM-DS-CFC using MultiCIDIM-DS as basic algorithm, but the proposed approach can be extended to any other multiple classifier system designed for incremental learning.*

*We have made different experiments and the results show that the performance of the new algorithms is comparable with the performance of other algorithms. But there is a circumstance that must be empha-*

*sized: in every experiment, the accuracy achieved by MultiCIDIM-DS-CFC is better than the one achieved by MultiCIDIM-DS. So we can determine that using correction filters may improve the accuracy of other multiple classifier systems.*

*We must note that the different algorithms proposed in this thesis have been designed to do an incremental learning, but, depending on the kind of problem, one algorithm could be more useful than the other (or vice versa). The size of the dataset is one of the aspects that can influence on the decision. If the dataset is small (or not very large) the algorithms based on multiple classifier systems could induce incomplete models (because there are insufficient examples to induce all the basic classifiers needed to complete the multiple classifier system). In that case it would be preferable to use the IADEM-2 algorithm. It would also be preferable to use IADEM-2 when there are continuous attributes in the dataset because CIDIM, the base learner used to induce the multiple classifier system, can not deal directly with them. If the algorithm is going to be used to extract knowledge using an on-line approach, MultiCIDIM-DS-CFC will show a more appropriate performance because it takes batches of examples continuously and it does not stop the execution when the learning process is stabilized (as IADEM-2 does). Furthermore, the model induced by MultiCIDIM-DS-CFC could present a higher level of adaptability because every basic classifier can be renovated using new batches of examples.*

## 10.2. Futuras Líneas de Trabajo // Future Work

Durante la exposición de las diferentes aportaciones realizadas se han ido mencionando diversas líneas de investigación que suscitan nuestro interés. Algunas son más específicas de uno u otro enfoque y otras son comunes a ambos.

Una limitación que no ha sido observada en la experimentación que hemos realizado, pero que es susceptible de aparecer en el algoritmo IADEM-2 sería consecuencia de una alta dimensionalidad del conjunto de datos. Este problema podría ser resuelto mediante técnicas de preprocesado de los datos, pero, entre nuestros objetivos futuros se encuentra el modificar el algoritmo IADEM-2 para afrontar esta dificultad de una forma directa. Contemplamos la posibilidad de controlar el tamaño del árbol de decisión inducido de forma que sólo se consuma memoria para almacenar la información que sea más necesaria, inhabilitando temporalmente aquellas ramas que sean menos prometedoras. Otra característica que sería deseable mejorar en este algoritmo es el tratamiento que reciben los atributos continuos, tanto en el proceso de inducción del árbol como en el proceso de predicción. Para aumentar la comprensibilidad del modelo sería deseable sustituir la división dicotómica usada para los atributos continuos por otro tipo de división que produjese múltiples intervalos. Así mismo, el uso que se hace, durante el proceso de predicción, de la información almacenada para este tipo de atributo sería susceptible de ser mejorado. Sin dejar el tema de los atributos continuos, una extensión del algoritmo CIDIM, que sería bastante útil para aumentar el tipo de conjuntos de datos con los que se pudiese trabajar, sería la de incorporar el tratamiento de atributos continuos de forma directa, sin necesidad de una discretización previa como ocurre en la versión actual.

Otras dos extensiones que serían de gran utilidad para ambos enfoques (el basado en cotas de concentración y el basado en sistemas multiclasificadores) serían la posibilidad de trabajar sobre conjuntos de datos en los que existiesen atributos con valores desconocidos y la capacidad de adaptar los modelos inducidos a posibles cambios de concepto. El tratamiento de atributos con valores desconocidos ampliaría aún más el ámbito de aplicación de nuestros algoritmos y la incorporación de mecanismos para adaptar los modelos ante cambios de conceptos se hace indispensable a la hora de trabajar con flujos de datos reales. Los enfoques para incluir la capacidad de extraer conocimiento a pesar de la existencia de cambios de concepto serán diferentes según el algoritmo en cuestión. Nuestra primera intención para incorporar esta característica al algoritmo IADEM-2 será incluir algún mecanismo en los nodos del árbol de decisión de forma que se pueda realizar una pérdida selectiva del conocimiento obsoleto. La extensión del algoritmo MultiCIDIM-DS-CFC para incorporar esta cualidad podría ser más directa y estudiaremos si bastaría con el empleo de algoritmos incrementales que detecten el cambio de concepto para generar los filtros correctores.

Para terminar, debemos mencionar nuestro interés en continuar desarrollando nuevos métodos y algo-

ritmos en el área del aprendizaje incremental, incorporando también técnicas evolutivas. Estas técnicas se han mostrado bastante útiles en el caso no incremental y consideramos que son fácilmente adaptables para extenderlas con el enfoque incremental.

**Future work**

*In this thesis we have described different contributions, but we also have presented different lines of research that we will consider in our future works.*

*One limitation that is susceptible of appearing when using IADEM-2 is the high dimensionality of the problem. This limitation has not appeared in any experiment that we have done, but it must be faced in order to make more robust our algorithm. This problem could be solved using preprocessing techniques, but our intention is to include some method that let IADEM-2 deal with that kind of problem in a direct way. We will study the possibility of controlling the size of the decision tree using only the relevant leaves and removing temporarily the least relevant ones. Other characteristic that could be improved is the treatment done with the continuous attributes, during the induction process and during the prediction process. Trying to increase the comprehensibility of the model that represent the knowledge, we will study the possibility of substituting the dichotomic division used with continuous attributes by another one based on multiple intervals. Besides, we will try to improve the use of the information stored for every continuous attribute when it is used for predicting. Another future work related with continuous attributes is our intention of including some method in CIDIM to work directly with that kind of attribute.*

*We will consider two more extensions that will affect to both approaches presented in this thesis (based on concentration bound and based on multiple classifier systems): working with datasets where some values could be missed and extracting knowledge from datasets where the concepts could change. The treatment of attributes with missing values will widen the applicability of the algorithms, and the inclusion of methods to deal with concept drift is necessary to work with real data streams. The methods proposed to include these extensions to our algorithms will depend on the nature of the algorithm. In the case of IADEM-2 our first attempt will be the insertion of some detecting method in the structure of the decision tree. Thus, when any concept drift is detected, only the corresponding branch will be removed. In the case of MultiCIDIM-DS, the modification could be simpler. It may be enough with the use of incremental algorithms with the ability of detecting concept drift to induce the correction filters.*

*Finally, we must declare our intention of continuing our research developing new methods and algorithms using an evolutionary approach. The evolutionary programming has shown a very good performance for non incremental learning and we consider that it could be easily adapted.*

# Parte IV

# Apéndices

# Apéndice A

# Formulación explícita de la cota de Chernoff

La expresión de la cota de Chernoff que hemos usado en el desarrollo de esta tesis ha sido la presentada en las fórmulas 1.3 y 1.4 (en la página 16), que constituían un corolario derivado de la cota general de Chernoff. Las ecuaciones definidas en el teorema que presenta la cota general de Chernoff para ambas cotas (cota superior en la ecuación 1.1 y cota inferior en la ecuación 1.2) no ofrecían la comodidad suficiente para ser usadas directamente, y por ello hemos usado las presentadas en el corolario.

El objetivo de este apéndice es demostrar que las fórmulas del corolario son algo menos restrictivas que las que forman el teorema, pero que se derivan de ellas. Comenzaremos demostrando la ecuación de la cota inferior por ser más sencilla (tomada de [Dia-2001]) y posteriormente demostraremos la ecuación para la cota superior (tomada parcialmente de [Dia-2001], [Mot-1995] y [Zha-2002] y completada en este apéndice).

## A.1. Cota inferior

De las desigualdades que aparecen en la ecuación que presentamos a continuación, la primera es la propia del teorema (ecuación 1.2), y la segunda es la que queremos demostrar (ecuación 1.4).

$$\Pr[X_s < (1-\beta)\mu_s] \leq \left(\frac{e^{-\beta}}{(1-\beta)^{1-\beta}}\right)^{\mu_s} \leq e^{-\frac{\beta^2\mu_s}{2}} \qquad \text{para } 0<\beta<1 \tag{A.1}$$

Al ser la cota inferior, los valores de $\beta$ van a pertenecer al intervalo $(0,1)$ y usando el desarrollo de Taylor tenemos:

$$(1-\beta)\cdot\ln(1-\beta) \;=\; -\beta+\frac{\beta^2}{2}+O(\beta^3)$$

lo que implica que:

$$(1-\beta)^{(1-\beta)} \;>\; e^{\frac{\beta^2}{2}-\beta}$$

Sustituyendo en la ecuación A.1 demostramos que:

$$\begin{aligned}
\Pr[X_s < (1-\beta)\mu_s] \leq \left(\frac{e^{-\beta}}{(1-\beta)^{1-\beta}}\right)^{\mu_s} &\leq \left(\frac{e^{-\beta}}{e^{\frac{\beta^2}{2}-\beta}}\right)^{\mu_s} \\
\left(\frac{e^{-\beta}}{(1-\beta)^{1-\beta}}\right)^{\mu_s} &\leq \left(e^{-\beta-(\frac{\beta^2}{2}-\beta)}\right)^{\mu_s} \\
\left(\frac{e^{-\beta}}{(1-\beta)^{1-\beta}}\right)^{\mu_s} &\leq e^{-\frac{\beta^2\mu_s}{2}}
\end{aligned}$$

## A.2. Cota superior

Al igual que ocurría con la cota inferior, de las desigualdades que aparecen en la ecuación que presentamos a continuación, la primera es la propia del teorema (ecuación 1.1), y la segunda es la que queremos demostrar (ecuación 1.3).

$$\Pr[X_s > (1+\beta)\mu_s] \leq \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s} \leq e^{-\frac{\beta^2\mu_s}{3}} \quad \text{para } 0<\beta<1 \tag{A.2}$$

$$\Pr[X_s > (1+\beta)\mu_s] \leq \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s} \leq e^{-\frac{\beta^2\mu_s}{2+\beta}} \quad \text{para } \beta \geq 1 \tag{A.3}$$

Como puede apreciarse, la demostración que debemos realizar es similar a la realizada en el caso de la cota inferior con la salvedad de que la segunda desigualdad está dividida en dos casos:

- **Primera rama** (ecuación A.2):

  Como paso preliminar demostraremos que para $\beta \in [0,1]$:

  $$(1+\beta)\ln(1+\beta) \geq \beta + \frac{\beta^2}{3} \tag{A.4}$$

  Dividiremos la demostración de la ecuación anterior en tres casos:

  1. para $\beta = 0$. Es trivial, puesto que:

     $$(1+0)\ln(1+0) \geq 0 + 0^2/3 \quad \Rightarrow \quad \ln(1) \geq 0$$

  2. para $\beta = 1$. También es trivial, puesto que:

     $$(1+1)\ln(1+1) \geq 1 + 1^2/3 \quad \Rightarrow \quad 2\ln(2) \geq 1 + 1/3 \quad \Rightarrow \quad 1,386 \geq 1,333$$

  3. para $\beta \in (0,1)$. Sea $f(\beta) = \beta + \frac{\beta^2}{3} - (1+\beta)\ln(1+\beta)$. Entonces, $f'(\beta) = \frac{3}{2}\beta - \ln(1+\beta)$. Como $f$ es continua, $f(0) = 0$, $f(1) = -0,529$ y las raíces de $f'$ son $\beta = 0$ y $\beta = 1,144$, sabemos que $f$ tiene que ser decreciente en el intervalo real $(0,1)$. El hecho de que $f$ sea decreciente, junto con que $f(0) = 0$, hacen que $f(\beta) \leq 0$ para $\beta \in [0,1]$. Por lo tanto, la ecuación A.4 también es cierta para $\beta \in (0,1)$.

  Sabiendo que la ecuación A.4 es válida para $\beta \in [0,1]$ podemos sustituirla en la ecuación A.2:

  $$\begin{aligned}
  \Pr[X_s > (1+\beta)\mu_s] \leq \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s} &\leq \left(\frac{e^{\beta}}{e^{\beta+\frac{\beta^2}{3}}}\right)^{\mu_s} \\
  \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s} &\leq \left(e^{\beta-(\beta+\frac{\beta^2}{3})}\right)^{\mu_s} \\
  \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s} &\leq e^{-\frac{\beta^2\mu_s}{3}}
  \end{aligned}$$

- **Segunda rama** (ecuación A.3):

  Como paso preliminar demostraremos que para $\beta \geq 1$:

  $$\frac{(1+\beta)\ln(1+\beta) - \beta}{\beta^2} \geq \frac{1}{2+\beta} \tag{A.5}$$

Primero reduciremos la inecuación a otra más sencilla

$$\begin{aligned}\frac{(1+\beta)\ln(1+\beta)-\beta}{\beta^2} &\geq \frac{1}{2+\beta}\\ (1+\beta)\ln(1+\beta) &\geq \frac{2\beta+2\beta^2}{2+\beta}\\ \ln(1+\beta) &\geq \frac{2\beta}{2+\beta}\end{aligned}$$

Sea $g(\beta)=\ln(1+\beta)-\frac{2\beta}{2+\beta}$. Entonces $g'(\beta)=\frac{1}{1+\beta}-\frac{4}{(2+\beta)^2}$. Como $g(1)=0,0265$ y $g$ es creciente puesto que $g'$ es positiva (porque $g'(\beta)\geq 0$ para $\beta\geq 1$), sabemos que la ecuación A.5 es cierta para $\beta\geq 1$.

Antes de completar la demostración de esta segunda rama (ecuación A.3) debemos introducir el siguiente teorema, tomado de la página 72 del libro "*Randomized algorithms*" [Mot-1995]:

TEOREMA 3
Para $0<\beta\leq U$,

$$\left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s}\leq e^{-\beta^2\mu_s\cdot c(U)}\ , \quad \text{donde } c(U)=[(1+U)\ln(1+U)-U]/U^2$$

Usando este teorema (igualando $U=\beta$) y sabiendo que la ecuación A.5 es válida para $\beta\geq 1$, podemos hacer la siguiente sustitución en la ecuación A.2:

$$\begin{aligned}\Pr[X_s>(1+\beta)\mu_s]\leq\left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s} &\leq e^{-\beta^2\mu_s\cdot[(1+\beta)\ln(1+\beta)-\beta]/\beta^2}\\ \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s} &\leq e^{-\beta^2\mu_s\cdot\frac{1}{2+\beta}}\\ \left(\frac{e^{\beta}}{(1+\beta)^{1+\beta}}\right)^{\mu_s} &\leq e^{-\frac{\beta^2\mu_s}{2+\beta}}\end{aligned}$$

# Apéndice B

# Publicaciones // Publications

En este apéndice presentamos un resumen esquemático de los trabajos realizados según su relación con las distintas partes de esta tesis. De esta forma se podrá ver gráficamente la evolución de los diferentes algoritmos y localizar los trabajos publicados. Así, en la sección B.1 veremos las publicaciones relacionadas con el aprendizaje incremental basado en cotas de concentración, en la sección B.2 las que están basadas en sistemas multiclasificadores y en la sección B.3 presentaremos otras publicaciones, no directamente relacionadas con los algoritmos desarrollados, pero que serán referencia para algunos de nuestros futuros trabajos de investigación.

*In this appendix we summarize, in a schematic way, the publications related to the different parts of this thesis. Our intention is to show graphically the evolution of the algorithms that we propose. Thus, in section B.1 we present the publications related to the incremental learning approach based on concentration bounds, in section B.2 we enumerate the ones based on multiple classifier systems, and finally, in section B.3 we describe some other publications, not directly related to the algorithms proposed, but they will be used as reference to continue with some future research.*

## B.1. Publicaciones relacionadas con el aprendizaje incremental basado en cotas de concentración

Los trabajos publicados que están relacionados con el algoritmo que proponemos en la parte I de esta tesis, IADEM-2, son cinco y a continuación damos una breve descripción de cada uno de ellos. Para facilitar la estructuración hemos incluido la figura B.1 donde se pueden ver gráficamente las diferentes aportaciones realizadas que han conducido a la definición definitiva del nuevo algoritmo que proponemos: IADEM-2.

Una breve descripción de cada artículo puede encontrarse a continuación:

- ***IADEM-0: Un nuevo algoritmo incremental*** **[Ram-2004]**: resume el algoritmo original, IADEM-0, propuesto por Ramos y Morales [Ram-2000; Ram-2001], sobre el que hemos desarrollado las diferentes aportaciones propuestas. En él se presentan las ideas principales, se actualiza el contexto y se enumeran las conclusiones observadas después de realizar un nuevo estudio experimental.

- ***Inducción de árboles de decisión por muestreo tolerante al ruido*** **[Cam-2002]**: en este trabajo, un proyecto de fin de carrera, se realiza la primera extensión del algoritmo IADEM-0 y se le dota de mecanismos para trabajar con conjuntos de datos en los que hubiese ruido. Como resultado se obtiene el algoritmo IADEM. En este trabajo también se realiza una descripción formal del algoritmo, mostrando su funcionamiento sin centrarnos en ninguna implementación concreta, y se solucionan determinados casos extremos que no fueron considerados anteriormente. Para terminar, se incluye un estudio detallado de los parámetros que intervienen en el algoritmo y se proponen sus valores por defecto.

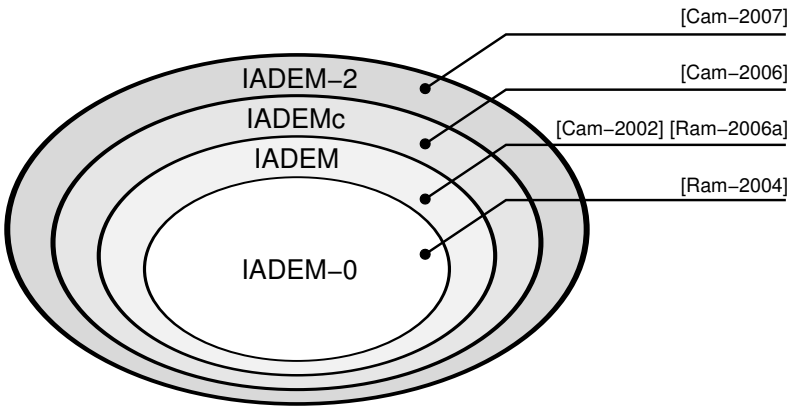



Figura B.1: Esquema de los publicaciones propias relacionadas con el aprendizaje incremental basado en cotas de concentración.
*Diagram with our publications related to incremental learning approach based on concentration bounds.*

- ***Incremental algorithm driven by error margins*** **[Ram-2006a]**: este artículo recoge las características principales del trabajo anterior y realiza un estudio experimental más avanzado y enfocado a realzar las ventajas derivadas del uso del algoritmo IADEM.

- ***Improving prediction accuracy of an incremental algorithm driven by error margins*** **[Cam-2006]**: este trabajo introduce dos nuevas mejoras al algoritmo IADEM y da lugar al algoritmo llamado IADEMc. Las dos aportaciones introducidas son la posibilidad de trabajar con atributos continuos directamente sin tener que realizar discretizaciones previas y la mejora de la predicción usando información que ya estaba disponible en el árbol de decisión que representa el conocimiento.

- ***Improving the performance of an incremental algorithm driven by error margins*** **[Cam-2007]**: este artículo recoge los aspectos fundamentales del algoritmo IADEM-2 presentados en las secciones 3.2, 3.3 y 3.4. Además, le acompaña un estudio experimental bastante completo en el que se muestran los ventajas y limitaciones propias del algoritmo.

## *Publications related to incremental learning based on concentration bounds*

*Our publications related to the incremental algorithm proposed in the part I of this thesis, IADEM-2, are five and we will give a brief description of them below. Before, we have included the figure B.1 to facilitate the organization.*

*Next we give the description for different works and papers:*

- **IADEM-0: Un nuevo algoritmo incremental [Ram-2004]**: *it summarizes the original algorithm, IADEM-0, introduced by Ramos and Morales [Ram-2000; Ram-2001]. It is the basis of the different contributions that we propose. In this work the main ideas (including a new context) are presented and a new experimental study is carried out.*

- **Inducción de árboles de decisión por muestreo tolerante al ruido [Cam-2002]**: *in this work, it is proposed the first extension to IADEM-0: a new method to detect noise is inserted into the algorithm. Thus, datasets with noise can be used too. As a result we design the algorithm called IADEM. Besides the new method, a formal description of the algorithm is given, showing how it works without any reference to any concrete implementation. Some extreme cases are also considered. Finally, a detailed*

*experimental section is described and an analysis of the internal parameters is done (setting the default values for them).*

- **Incremental algorithm driven by error margins [Ram-2006a]**: *this paper collects the main ideas from the previous work and a more advanced experimental analysis is done in order to enhance the advantages that can be expected by using IADEM.*

- **Improving prediction accuracy of an incremental algorithm driven by error margins [Cam-2006]**: *this work introduces two new contributions to IADEM and describes the new algorithm IADEMc. The two contributions are the capability of working with datasets where some attributes are continuous (without doing previous discretization) and the improvement incorporated to the prediction process (using more information available in the decision tree).*

- **Improving the performance of an incremental algorithm driven by error margins [Cam-2007]**: *this paper describes the main contributions described in sections 3.2, 3.3 and 3.4, and presents the algorithm proposed in this thesis: IADEM-2. In addition, a complete experimental section shows the main advantages (and limitations) of the algorithm.*

## B.2. Publicaciones relacionadas con el aprendizaje incremental basado en sistemas multiclasificadores

Los trabajos publicados que están relacionados con los algoritmos que proponemos en la parte II de esta tesis, MultiCIDIM-DS y MultiCIDIM-DS-CFC, son nueve. Existen artículos relacionados más directamente y otros menos directamente, pero todos los que hemos agrupado en esta sección tienen, al menos, una de las siguientes características en común: se han realizado teniendo como base al algoritmo CIDIM y/o son sistemas multiclasificadores. En la figura B.2 hemos representado una gráfica en la que intentamos esquematizar los distintos trabajos y cómo se relacionan unos con otros. A continuación daremos una breve descripción de todos ellos, pero antes debemos indicar que los artículos y trabajos más directamente relacionados con MultiCIDIM-DS y MultiCIDIM-DS-CFC son los que parten del núcleo central de la gráfica (donde se encuentra CIDIM) y se disponen por debajo de él.

Una breve descripción de cada artículo puede encontrarse a continuación:

- ***Induction of decision trees using an internal control of induction*** **[Ram-2005c]**: presenta con mayor detalle el algoritmo CIDIM propuesto por Ramos, Morales y Villalba [Ram-2000a; Ram-2001] y se realiza un estudio experimental bastante más extenso y completo, analizando e identificando las características propias del algoritmo. Las principales características de este algoritmo, que lo harán bastante útil para ser usado en sistemas multiclasificadores, son que induce árboles de decisión bastante precisos y, al mismo tiempo, de reducido tamaño. Este comportamiento se debe, entre otros factores, a que el conjunto de entrenamiento se divide en dos, uno para inducir el modelo y otro para controlar (validar) el aprendizaje.

- ***Modelado de un robot móvil basado en aprendizaje inductivo*** **[Rui-2005]**: este artículo describe una aplicación práctica del algoritmo CIDIM en el que se modelan ciertas características de un robot aprovechando las reglas extraídas del árbol de decisión generado por dicho algoritmo.

- ***E-CIDIM: Ensemble of CIDIM classifiers*** **[Ram-2005a]**: en este trabajo se propone un sistema multiclasificador que usa al algoritmo CIDIM como algoritmo de aprendizaje básico para inducir los clasificadores básicos. Se caracteriza porque la inducción de clasificadores básicos continua mientras se encuentre alguno que sea mejor (más preciso) que cualquiera de los existentes en el sistema multiclasificador. Los resultados obtenidos son bastante positivos, induciendo sistemas bastante precisos y cuyos clasificadores básicos son bastante pequeños (y comprensibles).

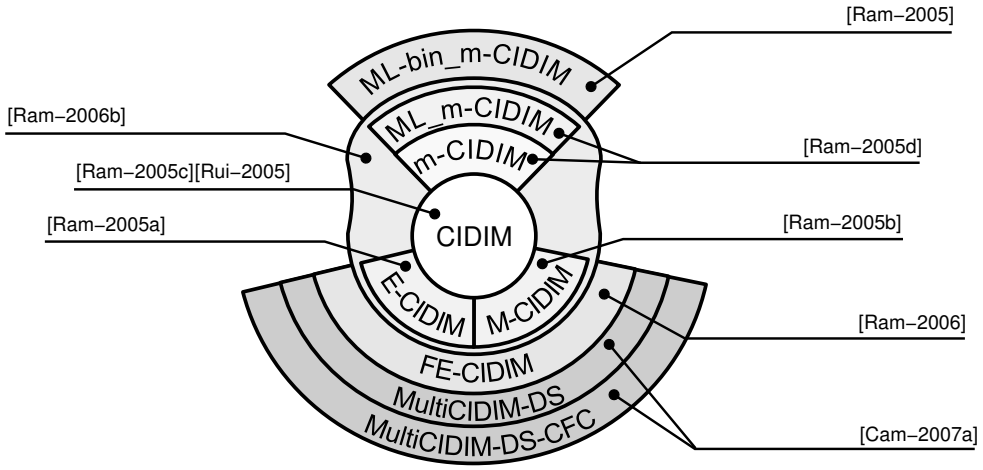



Figura B.2: Esquema de los publicaciones propias relacionadas con el aprendizaje incremental basado en sistemas multiclasificadores.
*Diagram with our publications related to incremental learning approach based on multiple classifier systems.*

- ***Inducción de árboles de decisión con CIDIM: nuevos enfoques*** **[Ram-2005b]**: en este trabajo se recopilan los avances introducidos en los algoritmos CIDIM y E-CIDIM y se propone otra variante para crear un sistema multiclasificador que aproveche las particularidades del algoritmo CIDIM. Tal sistema multiclasificador recibe el nombre de M-CIDIM y su aportación consiste en reordenar el conjunto de entrenamiento en pasos sucesivos para que el algoritmo CIDIM reciba conjuntos de entrenamiento y control que sean ortogonales con los anteriores. El objetivo de esta reordenación es aumentar la variabilidad de los conjuntos de entrenamiento (y control) al máximo para conseguir modelos que sean lo más diferente posible entre ellos.

- ***ML-CIDIM: Multiple layers of multiple classifier systems based on CIDIM*** **[Ram-2005d]**: en este trabajo se presenta el clasificador basado en CIDIM más básico que se pueda diseñar, m-CIDIM, pero sobre él se desarrolla, ML_m-CIDIM, que alcanza resultados muy prometedores. Consiste en generar múltiples capas de sistemas multiclasificadores, donde cada una de ellas usa el conjunto de datos y la información de las capas anteriores para ser creada. Esta idea de usar la información generada en otras capas, que se puede considerar como *meta*-aprendizaje, fue la que inspiró el enfoque utilizado para diseñar los sistemas multiclasificadores con filtros correctores.

- ***Sistemas multiclasificadores y de aprendizaje por capas basados en CIDIM*** **[Ram-2006b]**: en este artículo se puede encontrar una comparación de los anteriores algoritmos y sistemas multiclasificadores (CIDIM, E-CIDIM, M-CIDIM y ML_m-CIDIM) realizada entre ellos y comparándolos con otros algoritmos para caracterizar sus ventajas y observar bajo qué condiciones es preferible usar uno u otro enfoque.

- ***Aprendizaje por capas basado en sistemas multiclasificadores*** **[Ram-2005]**: este trabajo presenta una aportación hecha al algoritmo ML_m-CIDIM y lo que se propone es usar la misma idea de generar distintas capas de sistemas multiclasificadores pero modificando la información que pasa de capa a capa. Se observa que el algoritmo propuesto, ML-bin_m-CIDIM, ofrece un comportamiento mejor por el simple hecho de usar un enfoque binario a la hora de generar la información que pasará de capa a capa.

- ***FE-CIDIM: Fast ensemble of CIDIM classifiers*** **[Ram-2006]**: en este artículo se presenta, partiendo del algoritmo E-CIDIM expuesto en los artículos [Ram-2005a] y [Ram-2005b], un sistema multiclasificador aún más evolucionado, el algoritmo FE-CIDIM. Alcanza una precisión comparable con la que alcanza el algoritmo E-CIDIM (a veces mejor), pero consumiendo un tiempo mucho menor.

- ***Incremental learning with multiple classifier systems using correction filters for classification*** **[Cam-2007a]**: en este artículo partimos de nuestra experiencia con los sistemas multiclasificadores basados en CIDIM para presentar el concepto de clasificación mediante filtros correctores (CFC). Las ideas básicas presentadas en los capítulos 6 y 7 de la parte II de la tesis son descritas en este trabajo pero de forma mucho más resumida. Las dos aportaciones que incluye son la modificación del sistema multiclasificador FE-CIDIM para que pueda ser incremental (MultiCIDIM-DS) y la inclusión de filtros correctores para mejorar la precisión alcanzada por el sistema (MultiCIDIM-DS-CFC).

## *Publications related to incremental learning based on multiple classifier systems*

*Our publications related to the incremental algorithm proposed in the part II of this thesis, MultiCIDIM-DS and MultiCIDIM-DS-CFC, are nine. The works and papers grouped in this section are related to the basic classifier algorithm CIDIM and/or related to multiple classifier systems based on CIDIM. We also have included a figure to facilitate the description of the publications. In figure B.2 we summarize all the publications related to the algorithms proposed in the part II of this thesis, but we must notice that the works most directly related are those that goes from the center of the figure to the bottom.*

*Next we give the description for different works and papers:*

- **Induction of decision trees using an internal control of induction [Ram-2005c]**: *this paper describes, in a detailed way, the algorithm CIDIM, proposed by Ramos, Morales and Villalba [Ram-2000a; Ram-2001] and it presents a very complete experimental section where the main characteristics of the algorithm are identified and analyzed. The most relevant qualities of CIDIM, what makes it a very useful learning algorithm to be used in any multiple classifier system, are the following: it induces very accurate decision trees and, at the same time, they are very small. This behaviour it is motivated, in part, by the division of the training dataset in two subsets, one for the induction of the decision tree and the other one to control (validate) the learning process.*

- **Modelado de un robot móvil basado en aprendizaje inductivo [Rui-2005]**: *this work describes one real application of CIDIM. It is used to model some characteristics of a robot using the few rules produced in the decision tree induced by the algorithm.*

- **E-CIDIM: Ensemble of CIDIM classifiers [Ram-2005a]**: *in this paper one multiple classifier system, that uses CIDIM as the learning algorithm to induce the basic classifiers, is described. It is characterized by the iterative induction of basic classifiers until none of the new basic classifiers improves the performance (mainly accuracy) of the previous ones. The results achieved are very positive, and the induced systems are very accurate and easily understandable because of the small size of the basic classifiers.*

- **Inducción de árboles de decisión con CIDIM: nuevos enfoques [Ram-2005b]**: *contributions proposed in the algorithms CIDIM and E-CIDIM are completed in this work with a new variant that takes advantages from the special design of CIDIM. That new algorithm is called M-CIDIM and its contribution is the mechanism used to resort the training dataset during consecutive steps in order to maximize the differences between the decision trees induced by CIDIM. The aim of this resort (orthogonalization) is to take advantage, as much as possible, of the variability of models induced by CIDIM.*

- **ML-CIDIM: Multiple layers of multiple classifier systems based on CIDIM [Ram-2005d]**: *in this work we present the most basic multiple classifier system based on CIDIM, m-CIDIM, but it is*

*used to develop a very advanced new algorithm. ML_m-CIDIM is an algorithm that induces multiple layers of multiple classifier systems, and every layer uses the information provided by the previous one to improve the global system. The idea of using the information provided by other classifier (*meta-*learning) inspired the approach used to design the new algorithms proposed in the part II of this thesis.*

- **Sistemas multiclasificadores y de aprendizaje por capas basados en CIDIM [Ram-2006b]**: *in this paper an exhaustive comparison between CIDIM, E-CIDIM, M-CIDIM, ML_m-CIDIM and other algorithms can be found. Here can be clearly identified the advantages of every algorithm and determined which approach is the most suitable one for every situation.*

- **Aprendizaje por capas basado en sistemas multiclasificadores [Ram-2005]**: *this work describes the algorithm ML-bin_m-CIDIM, one evolution of the algorithm ML_m-CIDIM that uses the same idea of inducing several layers of multiple classifier systems. The difference, what improves the performance shown by ML_m-CIDIM, is the binary approach adopted to pass the information from one layer to the following one.*

- **FE-CIDIM: Fast ensemble of CIDIM classifiers [Ram-2006]**: *this paper presents another multiple classifier system that uses CIDIM as base classifier. It is the most advanced non-incremental system that we have created until this moment and it is called FE-CIDIM. It is based on E-CIDIM (described in [Ram-2005a] and [Ram-2005b]) and its peculiarity is the speed. The accuracy achieved by the new algorithm is comparable (sometimes better) to the previous one, but the time needed to induce the multiple classifier system is much lesser.*

- **Incremental learning with multiple classifier systems using correction filters for classification [Cam-2007a]**: *here we collect the contributions explained in the chapters 6 and 7. Using our experience with multiple classifier systems based on CIDIM we have designed a new algorithm (MultiCIDIM-DS) that can deal with very large dataset in an incremental way. Besides this new algorithm, we present a more interesting contribution: the use of "*correction filters*" to improve the accuracy achieved by the system. This interesting contribution has been used to define the algorithm MultiCIDIM-DS-CFC.*

## B.3. Publicaciones relacionadas con otras futuras líneas de investigación

Aunque los trabajos que comentamos a continuación no estén directamente relacionados con las aportaciones realizadas en esta tesis, sí que tienen importancia para algunos de los futuros desarrollos que consideramos afrontar. Es por eso que describimos brevemente los siguientes trabajos:

- ***Algoritmos evolutivos para clasificación de experiencias usando árboles de decisión* [Cam-2004]**: en este trabajo se presenta un mecanismo para inducir árboles de decisión mediante el empleo de algoritmos evolutivos. Su nombre es CADMAE y la principal ventaja que se consigue con este tipo de enfoque es que los árboles de decisión suelen ser igual de precisos que los inducidos por otros algoritmos, pero, además, acostumbran a ser más pequeños. Esto se debe a la forma en que se induce una población de árboles de decisión considerando múltiples alternativas y no fijando de forma definitiva la estructura de los árboles. También contemplamos la necesidad de continuar nuestros trabajos futuros por esta línea que ofrece de forma natural la posibilidad de utilizar un enfoque incremental y muy posiblemente con la particularidad de poder hacer frente a los cambios de concepto.

- ***Inducción de multiclasificadores basados en árboles de decisión usando algoritmos evolutivos* [Cam-2005]**: aprovechando nuestra experiencia con los sistemas clasificadores y disponiendo de una población de árboles de decisión generados por el algoritmo anterior, CADMAE, se hace casi directo el uso de diversos árboles de decisión como clasificadores básicos para un sistema multiclasificador.

Así surge el algoritmo M-CADMAE, con el que también esperamos continuar desarrollando nuevos enfoques para mejorar los sistemas multiclasificadores incluso en el ámbito de aprendizaje incremental con cambios de concepto.

- ***Early drift detection method*** **[Bae-2006]**: en este artículo se presenta un mecanismo para la detección de cambios de conceptos basada en la variación de la velocidad con que cambia la distribución que se está observando, llamado EDDM. Estudiando la variación de la velocidad se puede anticipar la predicción de un cambio de concepto, circunstancia que será de gran utilidad para el algoritmo que tiene que adaptar su modelo a los nuevos conceptos. Este trabajo inspirará los primeros desarrollos que hagamos para intentar dotar al algoritmo IADEM-2 y al MultiCIDIM-DS-CFC de capacidad para tratar el cambio de concepto.

## *Publications related with other future researches*

*Although the papers that we present below are not directly related to the contributions proposed in this thesis, they are important for our future research. This is the reason of describing them in this section:*

- **Algoritmos evolutivos para clasificación de experiencias usando árboles de decisión [Cam-2004]**: *in this work we use evolutionary algorithms to induce decision trees. The name of the algorithm is CADMAE and its main characteristic is the quality of the induced models. They usually are accurate decision trees and, in addition, they usually are smaller than any other model induced by other algorithms. The reason for such good performance is the method used to induce them. The structure of the tree is not fixed, different trees in the population can have different structures and the best ones arise. We will continue our developments on incremental learning with this method, that it is easily adaptable to support this approach, including the capability of dealing with concept drift.*

- **Inducción de multiclasificadores basados en árboles de decisión usando algoritmos evolutivos [Cam-2005]**: *using our experience with multiple classifier systems and having a population of decision trees induced by the previous algorithm, the extension to combine different decision trees is almost direct. The algorithm that selects the most suitable decision trees induced by CADMAE and creates a multiple classifier system is called M-CADMAE. We expect to continue our research with this algorithm too.*

- **Early drift detection method [Bae-2006]**: *this paper describes a novel method to detect concept drift using the speed variation observed in the distribution of examples that is being studied. The method is called EDDM and, studying that variation, it can detect the concept drift earlier. This work will be, together with other approaches, the reference used to introduce the capability of dealing with concept drift in the algorithms proposed in this thesis: IADEM-2 and MultiCIDIM-DS-CFC.*

# Índice alfabético

# Bibliografía


[Agr-1996] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen y I. Verkamo. ADVANCES IN KNOWLEDGE DISCOVERY AND DATA MINING, capítulo Fast discovery of association rules, páginas 307–328. AAAI Press. 1996.

[Aha-1991] D. W. Aha, D. Kibler y M. K. Albert. INSTANCE-BASED LEARNING ALGORITHMS. *Machine Learning*, 6, páginas 37–66. 1991.

[Ali-1996] K. M. Ali y M. J. Pazzani. ERROR REDUCTION THROUGH LEARNING MULTIPLE DESCRIPTIONS. *Machine Learning*, 24, páginas 173–202. 1996.

[Ant-1997] M. H. G. Anthony y N. L. Biggs. COMPUTATIONAL LEARNING THEORY. Cambridge University Press. 1997.

[Bae-2006] M. Baena-García, J. del Campo-Ávila, A. Bifet, R. Fidalgo, R. Gavaldà y R. Morales-Bueno. EARLY DRIFT DETECTION METHOD. En *Proceedings of the 4th International Workshop on Knowledge Discovery from Data Streams (IWKDDS-2006)*, páginas 77–86. 2006.

[Bau-1999] E. Bauer y R. Kohavi. AN EMPIRICAL COMPARISON OF VOTING CLASSIFICATION ALGORITHMS: BAGGING, BOOSTING, AND VARIANTS. *Machine Learning*, 36, páginas 105–142. 1999.

[Ber-1924] S. Bernstein. ON A MODIFICATION OF CHEBYSHEV'S INEQUALITY AND OF THE ERROR FORMULA OF LAPLACE. *Ann. Sci. Inst. Savantes Ukraine, Sect. Math.*, 1, páginas 38–49. 1924.

[Ber-2003] M. Berthold y D. J. Hand. INTELLIGENT DATA ANALYSIS. AN INTRODUCTION. Springer-Verlag, 2 edición. 2003.

[Bif-2006] A. Bifet y R. Gavaldà. LEARNING FROM TIME-CHANGING DATA WITH ADAPTIVE WINDOWING. Informe técnico, Universidad Politécnica de Cataluña. 2006.

[Boh-1990] M. Bohanec y V. Rajkovič. DEX: AN EXPERT SYSTEM SHELL FOR DECISION SUPPORT. *Sistemica*, 1 (1), páginas 145–157. 1990.

[Bra-1998] I. Bratko, T. Urbančič y C. Sammut. MACHINE LEARNING AND DATA MINING: METHODS AND APPLICATIONS, capítulo Behavioural cloning of control skill, páginas 335–350. John Wiley & Sons Ltd. 1998.

[Bre-1984] L. Breiman, J. H. Friedman, R. A. Olshen y C. J. Stone. CLASSIFICATION AND REGRESSION TREES. Wadsworth International Group. 1984.

[Bre-1996] L. Breiman. BAGGING PREDICTORS. *Machine Learning*, 24 (2), páginas 123–140. 1996.

[Bre-1999] L. Breiman. PASTING SMALL VOTES FOR CLASSIFICATION IN LARGE DATABASES AND ON-LINE. *Machine Learning*, 36, páginas 85–103. 1999.

[Bru-2004] B. Brumen, I. Golob, H. Jaakkola, T. Welzer y I. Rozman. EARLY ASSESSMENT OF CLASSIFICATION PERFORMANCE. En *Proceedings of the 2nd workshop on Australasian Information Security, Data Mining and Web Intelligence, and Software Internationalisation (ACSW Frontiers-2004)*, páginas 91–96. Australian Computer Society, Inc. 2004.

[Bun-1990] W. L. Buntime. A THEORY OF LEARNING CLASSIFICATION RULES. Tesis Doctoral, University of Technology, Sidney. 1990.

[Cam-2002] J. del Campo-Ávila. INDUCCIÓN DE ÁRBOLES DE DECISIÓN POR MUESTREO TOLERANTE AL RUIDO. Proyecto Fin de Carrera. Universidad de Málaga, España. 2002.

[Cam-2004] J. del Campo-Ávila, C. Cotta y G. Ramos-Jiménez. ALGORITMOS EVOLUTIVOS PARA CLASIFICACIÓN DE EXPERIENCIAS USANDO ÁRBOLES DE DECISIÓN. En *Actas del III Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB-2004)*, páginas 187–194. 2004.

[Cam-2005] J. del Campo-Ávila, G. Ramos-Jiménez y C. Cotta. INDUCCIÓN DE MULTICLASIFICADORES BASADOS EN ÁRBOLES DE DECISIÓN USANDO ALGORITMOS EVOLUTIVOS. En *Actas del IV Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB-2005)*, páginas 689–696. 2005.

[Cam-2006] J. del Campo-Ávila, G. Ramos-Jiménez, J. Gama y R. Morales-Bueno. IMPROVING PREDICTION ACCURACY OF AN INCREMENTAL ALGORITHM DRIVEN BY ERROR MARGINS. En *Proceedings of the 4th International Workshop on Knowledge Discovery from Data Streams (IWKDDS-06)*, páginas 57–66. 2006.

[Cam-2007] J. del Campo-Ávila, G. Ramos-Jiménez, J. Gama y R. Morales-Bueno. IMPROVING THE PERFORMANCE OF AN INCREMENTAL ALGORITHM DRIVEN BY ERROR MARGINS. *Intelligent Data Analysis*. 2007. Aceptado para ser publicado.

[Cam-2007a] J. del Campo-Ávila, G. Ramos-Jiménez y R. Morales-Bueno. INCREMENTAL LEARNING WITH MULTIPLE CLASSIFIER SYSTEMS USING CORRECTION FILTERS FOR CLASSIFICATION. *Lecture Notes in Computer Science*, Aceptado para ser publicado.

[Cas-2006] G. Castillo. ADAPTIVE LEARNING ALGORITHMS FOR BAYESIAN NETWORK CLASSIFIERS. Tesis Doctoral, University of Aveiro, Portugal. 2006.

[Cas-2006a] G. Castillo y J. Gama. AN ADAPTIVE PREQUENTIAL LEARNING FRAMEWORK FOR BAYESIAN NETWORK CLASSIFIERS. *Lecture Notes in Artificial Intelligence*, 4213, páginas 67–78. 2006.

[Cat-1991a] J. Catlett. MEGAINDUCTION: A TEST FLIGHT. En *Proceedings of the 8th International Workshop on Machine Learning*, páginas 596–599. 1991.

[Cat-1991] J. Catlett. MEGAINDUCTION: MACHINE LEARNING ON VERY LARGE DATABASES. Tesis Doctoral, Basser Department of Computer Science, University of Sydney, Australia. 1991.

[Cer-2006] P. Cerrito. INTRODUCTION TO DATA MINING USING SAS ENTERPRISE MINER. SAS Press. 2006.

[Cha-1999] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer y R. Wirth. CRISP-DM 1.0. STEP-BY-STEP DATA MINING GUIDE. CRISP-DM consortium. 1999.

[Che-1952] H. Chernoff. A MEASURE OF ASYMPTOTIC EFFICIENCY FOR TESTS OF A HYPOTHESIS BASED ON THE SUM OF OBSERVATIONS. *Annals of Mathematical Statistics*, 23, páginas 493–507. 1952.

[Chu-2006] F. Chung y L. Lu. CONCENTRATION INEQUALITIES AND MARTINGALE INEQUALITIES: A SURVEY. *Internet Mathematics*, 3 (1), páginas 79–127. 2006. Aceptado para ser publicado.

[Cox-1952] D. R. Cox. ESTIMATION BY DOUBLE SAMPLING. *Biometrika*, 39 (3/4), páginas 217–227. 1952.

[Dan-1997] G. B. Dantzing y M. N. Thapa. LINEAR PROGRAMMING: INTRODUCTION. Springer-Verlag. 1997.

[Dia-2001] J. Díaz, J. Petit y M. Serna. HANDBOOK OF RANDOMIZED COMPUTING, tomo II, capítulo A guide to concentration bounds, páginas 1–45. Kluwer Academic. 2001.

[Die-2000a] T. G. Dietterich. ENSEMBLE METHODS IN MACHINE LEARNING. *Lecture Notes in Computer Science*, 1857, páginas 1–15. 2000.

[Die-2000] T. G. Dietterich. AN EXPERIMENTAL COMPARISON OF THREE METHODS FOR CONSTRUCTING ENSEMBLES OF DECISION TREES: BAGGING, BOOSTING AND RANDOMIZATION. *Machine Learning*, 40, páginas 139–157. 2000.

[Die-2003] T. G. Dietterich. NATURE ENCYCLOPEDIA OF COGNITIVE SCIENCE, capítulo Machine learning. Macmillan. 2003.

[Die-1995] T. G. Dietterich y G. Bakiri. SOLVING MULTICLASS LEARNING PROBLEMS VIA ERROR-CORRECTING OUTPUT CODES. *Journal of Artificial Intelligence Research*, 2, páginas 263–286. 1995.

[Dod-1929] H. Dodge y H. Romig. A METHOD OF SAMPLING INSPECTION. *The Bell System Technical Journal*, 8, páginas 613–631. 1929.

[Dom-1999] C. Domingo, R. Gavaldà y O. Watanabe. ADAPTIVE SAMPLING METHODS FOR SCALING UP KNOWLEDGE DISCOVERY ALGORITHMS. *Lecture Notes in Computer Science*, 1721, páginas 172–183. 1999.

[Dom-2000a] C. Domingo y O. Watanabe. MADABOOST: A MODIFICATION OF ADABOOST. En *Proceedings of the 13th Annual Conference on Computational Learning Theory (COLT-2000)*, páginas 180–189. 2000.

[Dom-2002] C. Domingo, R. Gavaldà y O. Watanabe. ADAPTIVE SAMPLING METHODS FOR SCALING UP KNOWLEDGE DISCOVERY ALGORITHMS. *Data Mining and Knowledge Discovery*, 6, páginas 131–152. 2002.

[Dom-1997] P. Domingos y M. Pazzani. ON THE OPTIMALITY OF THE SIMPLE BAYESIAN CLASSIFIER UNDER ZERO-ONE LOSS. *Machine Learning*, 291, páginas 103–130. 1997.

[Dom-2000] P. Domingos y G. Hulten. MINING HIGH-SPEED DATA STREAMS. En *Proceedings of the ACM 6th International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD-2000)*, páginas 71–80. 2000.

[Dou-1995] J. Dougherty, R. Kohavi y M. Sahami. SUPERVISED AND UNSUPERVISED DISCRETIZATION OF CONTINUOUS FEATURES. En *Machine Learning: Proceedings of the 12th International Conference*, páginas 194–202. 1995.

[Dud-1973] R. O. Duda y P. E. Hart. PATTERN CLASSIFICATION AND SCENE ANALYSIS. Wiley-Interscience. 1973.

[Dud-2001] R. O. Duda, P. E. Hart y D. G. Stork. PATTERN CLASSIFICATION. John Wiley & Sons. 2001.

[Esp-1997] F. Esposito, D. Malerba y G. Semeraro. A COMPARATIVE ANALYSIS OF METHODS FOR PRUNING DECISION TREES. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19 (5), páginas 476–491. 1997.

[Est-2003] V. Estruch, C. Ferri, J. Hernández-Orallo y M. Ramírez-Quintana. BEAM SEARCH EXTRACTION AND FORGETTING STRATEGIES ON SHARED ENSEMBLES. *Lecture Notes in Computer Science*, 2709, páginas 206–216. 2003.

[Fan-1999] W. Fan, S. J. Stolfo y J. Zhang. THE APPLICATION OF ADABOOST FOR DISTRIBUTED, SCALABLE AND ON-LINE LEARNING. En *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-1999)*, páginas 362–366. 1999.

[Fay-1993] U. M. Fayyad y K. B. Irani. MULTI-INTERVAL DISCRETIZATION OF CONTINUOUS-VALUED ATTRIBUTES FOR CLASSIFICATION LEARNING. En *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI-1993)*, tomo 2, páginas 1022–1027. 1993.

[Fay-1996] U. M. Fayyad, G. Piatetsky-Shapiro y P. Smyth. ADVANCES IN KNOWLEDGE DISCOVERY AND DATA MINING, capítulo From data mining to knowledge discovery: An overview, páginas 1–30. AAAI Press. 1996.

[Fel-1968] W. Feller. AN INTRODUCTION TO PROBABILITY THEORY AND ITS APPLICATIONS. John Wiley & Sons. 1968.

[Fer-2003] A. Fern y R. Givan. ONLINE ENSEMBLE LEARNING: AN EMPIRICAL STUDY. *Machine Learning*, 53, páginas 71–109. 2003.

[Fer-2005] F. Ferrer-Troyano y J. S. Aguilar-Ruiz. MINERÍA DE DATA STREAMS: CONCEPTOS Y PRINCIPALES TÉCNICAS. En *Actas de las Jornadas de Técnicas de Minería de Datos, Universidad de Castilla La Mancha*. 2005.

[Fer-2005a] F. Ferrer-Troyano, J. S. Aguilar-Ruiz y J. C. Riquelme. INCREMENTAL RULE LEARNING AND BORDER EXAMPLES SELECTION FROM NUMERICAL DATA STREAMS. *Journal of Universal Computer Science*, 11 (8), páginas 1426–1439. 2005.

[Fis-1998] D. H. Fisher y J. C. Schlimmer. MODELS OF INCREMENTAL CONCEPT LEARNING: A COUPLED RESEARCH PROPOSAL. Informe Técnico CS-88-05, Vanderbilt University. 1998.

[Fla-1989] N. S. Flann y T. G. Dietterich. A STUDY OF EXPLANATION-BASED METHODS FOR INDUCTIVE LEARNING. *Machine Learning*, 4 (2), páginas 187–226. 1989.

[For-2006] I. Fortes, L. Mora-López, R. Morales-Bueno y F. Triguero. INDUCTIVE LEARNING MODELS WITH MISSING VALUES. *Mathematical and Computer Modelling*, 44, páginas 790–806. 2006.

[Fra-2000] E. Frank. PRUNING DECISION TREES AND LISTS. Tesis Doctoral, University of Waikato, New Zealand. 2000.

[Fre-2002] A. Freitas. DATA MINING AND KNOWLEDGE DISCOVERY WITH EVOLUTIONARY ALGORITHMS. Springer-Verlag. 2002.

[Fre-1995] Y. Freund. BOOSTING A WEAK LEARNING ALGORITHM BY MAJORITY. *Information and Computation*, 121, páginas 256–285. 1995.

[Fre-1996] Y. Freund y R. E. Schapire. EXPERIMENTS WITH A NEW BOOSTING ALGORITHM. En *Proceedings of the 13th International Conference on Machine Learning (ICML-1996)*, páginas 146–148. 1996.

[Fre-1997] Y. Freund y R. E. Schapire. A DECISION-THEORETIC GENERALIZATION OF ON-LINE LEARNING AND AN APPLICATION TO BOOSTING. *Journal of Computer and System Sciences*, 55, páginas 119–139. 1997.

[Fre-1998] Y. Freund. SELF BOUNDING LEARNING ALGORITHMS. En *Proceedings of the 11th Annual Conference on Computational Learning Theory (COLT-1998)*, páginas 247–258. 1998.

[Gam-1999] J. Gama. COMBINING CLASSIFICATION ALGORITHMS. Tesis Doctoral, University of Porto, Portugal. 1999.

[Gam-2000] J. Gama y P. Brazdil. CASCADE GENERALIZATION. *Machine Learning*, 41, páginas 315–343. 2000.

[Gam-2001] J. Gama. FUNCTIONAL TREES. *Lecture Notes in Artificial Intelligence*, 2226, páginas 59–73. 2001.

[Gam-2003] J. Gama, R. Rocha y P. Medas. ACCURATE DECISION TREES FOR MINING HIGH-SPEED DATA STREAMS. En *Proceedings of the 9th ACM International Conference on Knowledge Discovery and Data mining (SIGKDD-2003)*, páginas 523–528. 2003.

[Gam-2004] J. Gama, P. Medas y R. Rocha. FOREST TREES FOR ON-LINE DATA. En *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC-2004)*, páginas 632–636. ACM Press. 2004.

[Gam-2006] J. Gama, R. Fernandes y R. Rocha. DECISION TREES FOR MINING DATA STREAMS. *Intelligent Data Analysis*, 10 (1), páginas 23–45. 2006.

[Gam-2006a] J. Gama y C. Pinto. DISCRETIZATION FROM DATA STREAMS: APPLICATIONS TO HISTOGRAMS AND DATA MINING. En *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC-2006)*, páginas 662–667. 2006.

[Gav-2001] R. Gavaldà y O. Watanabe. SEQUENTIAL SAMPLING ALGORITHMS: UNIFIED ANALYSIS AND LOWER BOUNDS. *Lecture Notes In Computer Science*, 2264, páginas 173–188. 2001.

[Geh-1999] J. Gehrke, V. Ganti, R. Ramakrishnan y W. Loh. BOAT - OPTIMISTIC DECISION TREE CONSTRUCTION. En *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data*, páginas 169–180. 1999.

[Ger-1983] M. E. Gershon y L. Duckstein. MULTIOBJECTIVE APPROACHES TO RIVER BASIN PLANNING. *Journal of Water Resource Planning*, 109, páginas 13–28. 1983.

[Gir-2002] R. Giráldez, J. Aguilar-Ruiz, J. Riquelme, F. Ferrer-Troyano y D. Rodríguez. DISCRETIZATION ORIENTED TO DECISION RULES GENERATION. En *Proceedings of the 6th International Conference on Knowledge-Based Intelligent Information Engineering Systems*, páginas 275–279. 2002.

[Gir-2000] C. Giraud-Carrier. A NOTE ON THE UTILITY OF INCREMENTAL LEARNING. *AI Communications*, 13 (4), páginas 215–223. 2000.

[Gre-1992] R. Greiner y I. Jurisica. A STATISTICAL APPROACH TO SOLVING THE EBL UTILITY PROBLEM. En *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-1992)*, páginas 241–248. 1992.

[Gre-1996] R. Greiner. PALO: A PROBABILISTIC HILL-CLIMBING ALGORITHM. *Artificial Intelligence*, 84, páginas 177–208. 1996.

[Gu-2001] B. Gu, F. Hu y H. Liu. MODELLING CLASSIFICATION PERFORMANCE FOR LARGE DATA SETS: AN EMPIRICAL STUDY. *Lecture Notes in Computer Science*, 2118, páginas 317–328. 2001.

[Hag-1990] T. Hagerup y C. Rüb. A GUIDED TOUR OF CHERNOFF BOUNDS. *Information Processing Letters*, 33, páginas 305–308. 1990.

[Han-1997] D. Hand. CONSTRUCTION AND ASSESSMENT OF CLASSIFICATION RULES. John Wiley & Sons. 1997.

[Han-2000] J. Han y M. Kamber. DATA MINING: CONCEPTS AND TECHNIQUES. Morgan Kaufmann. 2000.

[Han-1990] L. K. Hansen y P. Salamon. NEURAL NETWORK ENSEMBLES. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12, páginas 993–1001. 1990.

[Hau-1992] D. Haussler. DECISION THEORETIC GENERALIZATIONS OF THE PAC MODEL FOR NEURAL NET AND OTHER LEARNING APPLICATIONS. *Information and Computation*, 100, páginas 78–150. 1992.

[Hau-1996] D. Haussler, M. Kearns, H. S. Seung y N. Tishby. RIGOROUS LEARNING CURVE BOUNDS FROM STATISTICAL MECHANICS. *Machine Learning*, 25, páginas 195–236. 1996.

[Her-2004] J. Hernández-Orallo, M. J. Ramírez-Quintana y C. Ferri-Ramírez. INTRODUCCIÓN A LA MINERÍA DE DATOS. Prentice Hall. 2004.

[Her-2004a] F. Herrera, C. Hervás, J. Otero y L. Sánchez. TENDENCIAS DE LA MINERÍA DE DATOS EN ESPAÑA, capítulo Un estudio empírico preliminar sobre los tests estadísticos más habituales en el aprendizaje automático, páginas 403–412. Red Española de Minería de Datos. 2004.

[Hes-2002] Z. Heszberger, J. Zátonyi y J. Bíró. EFFICIENT CHERNOFF-BASED RESOURCE ASSESSMENT TECHNIQUES IN MULTI-SERVICE NETWORKS. *Telecommunication Systems*, 20 (1-2), páginas 59–80. 2002.

[Ho-1998] T. K. Ho. C4.5 DECISION FORESTS. En *Proceedings of the 14th International Conference on Pattern Recognition (ICPR-1998)*, páginas 545–549. 1998.

[Hoe-1963] W. Hoeffding. PROBABILTY INEQUALITIES FOR SUMS OF BOUNDED RANDOM VARIABLES. *Journal of the American Statistical Association*, 58, páginas 13–30. 1963.

[Hol-2005] G. Holmes, R. Kirkby y B. Pfahringer. TIE BREAKING IN HOEFFDING TREES. En *Proceedings of the 2nd International Workshop on Knowledge Discovery in Data Streams (IWKDDS-2005)*. 2005.

[Hol-1993] R. Holte. VERY SIMPLE CLASSIFICATION RULES PERFORM WELL ON MOST COMMONLY USED DATASETS. *Machine Learning*, 11, páginas 63–91. 1993.

[Hou-1991] W. Hou, G. Ozsoyoglu y E. Dogdu. ERROR-CONSTRAINED COUNT QUERY EVALUATION IN RELATIONAL DATABASES. En *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, páginas 278–287. ACM Press. 1991.

[Hun-1966] E. B. Hunt, J. Marin y P. Stone. EXPERIMENTS IN INDUCTION. Academic Press. 1966.

[Hun-1993] K. J. Hunt. CLASSIFICATION BY INDUCTION: APLICATIONS TO MODELLING AND CONTROL OF NON-LINEAR DYNAMIC SYSTEMS. *Intelligent Systems Engineering*, 2 (4), páginas 231–245. 1993.

[Hwa-1981] C. L. Hwang y K. L. Yoon. MULTIPLE ATTRIBUTE DECISION MAKING: METHODS AND APPLICATIONS. A STATE-OF-THE-ART SURVEY. Springer Verlag. 1981.

[Hya-1971] L. Hyafil y R. L. Rivest. CONSTRUCTING OPTIMAL BINARY DECISION TREES IS NP-COMPLETE. *Information Processing Letters*, 5, páginas 15–17. 1971.

[Jac-1995] R. A. Jacobs. METHODS FOR COMBINING EXPERTS' PROBABILITY ASSESSMENTS. *Neural Computation*, 7, páginas 867–888. 1995.

[Jan-2002] S. Janson. ON CONCENTRATION OF PROBABILITY. En *Contemporary Combinatorics*, tomo 10 de *Bolyai Society Mathematical Studies*, páginas 289–301. Springer. 2002.

[Jer-2003] J. M. Jerez-Aragonés, J. A. Gómez-Ruiz, G. Ramos-Jiménez, J. Muñoz-Pérez y E. Alba-Conejo. A COMBINED NEURAL NETWORK AND DECISION TREES MODEL FOR PROGNOSIS OF BREAST CANCER RELAPSE. *Artificial Intelligence in Medicine*, 27 (1), páginas 45–63. 2003.

[Jer-2004] C. Jermaine, A. Pol y S. Arumugam. ONLINE MAINTENANCE OF VERY LARGE RANDOM SAMPLES. En *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, páginas 299–310. 2004.

[Jin-2003] R. Jin y G. Agrawal. EFFICIENT DECISION TREE CONSTRUCTION ON STREAMING DATA. En *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, páginas 571–576. 2003.

[Joh-1996] G. H. John y P. Langley. STATIC VERSUS DYNAMIC SAMPLING FOR DATA MINING. En *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-1996)*, páginas 367–370. 1996.

[Kea-1994] M. J. Kearns y U. V. Vazirani. AN INTRODUCTION TO COMPUTATIONAL LEARNING THEORY. MIT Press. 1994.

[Kim-2003] H. Kim, S. Pang, H. Je, D. Kim y S. Y. Bang. CONSTRUCTING SUPPORT VECTOR MACHINE ENSEMBLE. *Pattern Recognition*, 36, páginas 2757–2767. 2003.

[Kit-1998] J. Kittler. COMBINING CLASSIFIERS: A THEORETICAL FRAMEWORK. *Pattern Analysis & Applications*, 1 (1), páginas 18–27. 1998.

[Kiv-1994] J. Kivinen y H. Mannila. THE POWER OF SAMPLING IN KNOWLEDGE DISCOVERY. En *In Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Theory (PODS-1994)*, páginas 77–85. 1994.

[Kod-1987] Y. Kodratoff, M. Manago y J. Blythe. GENERALIZATION AND NOISE. *International Journal of Man-Machine Studies*, 27 (2), páginas 181–204. 1987.

[Koh-1996] R. Kohavi. SCALING UP THE ACCURACY OF NAIVE BAYES CLASSIFIERS: A DECISION-TREE HYBRID. En *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-1995)*, páginas 202–207. 1996.

[Kon-1998] I. Kononenko, I. Bratko y M. Kukar. MACHINE LEARNING AND DATA MINING: METHODS AND APPLICATIONS, capítulo Application of machine learning to medical diagnosis, páginas 389–405. John Wiley & Sons Ltd. 1998.

[Kun-2002] L. I. Kuncheva. A THEORETICAL STUDY ON SIX CLASSIFIER FUSION STRATEGIES. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24 (2), páginas 281–286. 2002.

[Lan-2003] J. Langford y A. Blum. MICROCHOICE BOUNDS AND SELF BOUNDING LEARNING ALGORITHMS. *Machine Learning*, 51, páginas 165–179. 2003.

[Lan-2005] J. Langford. TUTORIAL ON PRACTICAL PREDICTION THEORY FOR CLASSIFICATION. *Journal of Machine Learning Research*, 6, páginas 273–306. 2005.

[Lan-1992] P. Langley, W. Iba y K. Thompson. AN ANALYSIS OF BAYESIAN CLASSIFIERS. En *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI-1992)*, páginas 223–228. 1992.

[Lim-2000] T. S. Lim, W. Y. Loh y Y. S. Shih. A COMPARISON OF PREDICTION ACCURACY, COMPLEXITY, AND TRAINING TIME OF THIRTY-THREE OLD AND NEW CLASSIFICATION ALGORITHMS. *Machine Learning*, 48, páginas 203–228. 2000.

[Lin-1981] G. H. Lincoff. THE AUDUBON SOCIETY FIELD GUIDE TO NORTH AMERICAN MUSHROOMS. New York: Knopf. 1981.

[Lip-1990] R. J. Lipton, J. F. Naughton y D. A. Schneider. PRACTICAL SELECTIVITY ESTIMATION THROUGH ADAPTIVE SAMPLING. En *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, páginas 1–11. 1990.

[Liu-2002] H. Liu, F. Hussain, C. L. Tan y M. Dash. DISCRETIZATION: AN ENABLING TECHNIQUE. *Data Mining and Knowledge Discovery*, 6, páginas 393–423. 2002.

[Lop-2005] M. López-Valdés y E. Mayordomo. DIMENSION IS COMPRESSION. *Lecture Notes in Computer Science*, 3618, páginas 676–685. 2005.

[Mal-2000] M. A. Maloof y R. S. Michalski. SELECTING EXAMPLES FOR PARTIAL MEMORY LEARNING. *Machine Learning*, 41 (1), páginas 27–52. 2000.

[Man-1998] M. Manago y E. Auriol. MACHINE LEARNING AND DATA MINING: METHODS AND APPLICATIONS, capítulo Application of inductive learning and case-based reasoning for troubleshooting industrial machines, páginas 173–183. John Wiley & Sons Ltd. 1998.

[Man-1994] H. Mannila, H. Toivonen y I. Verkamo. EFFICIENT ALGORITHMS FOR DISCOVERING ASSOCIATION RULES. En *Proceedings of the AAAI Workshop on Knowledge Discovery in Databases (KDD-1994)*, páginas 181–192. 1994.

[Mar-1994] O. Maron y A. Moore. HOEFFDING RACES: ACCELERATING MODEL SELECTION SEARCH FOR CLASSIFICATION AND FUNCTION APPROXIMATION. En *Advances in Neural Information Processing Systems*, tomo 6, páginas 59–66. 1994.

[May-2007] E. Mayordomo. EFFECTIVE FRACTAL DIMENSION IN ALGORITHMIC INFORMATION THEORY. En *New Computational Paradigms: Changing Conceptions of What is Computable*. Springer-Verlag. 2007. Aceptado para ser publicado.

[McD-1989] C. McDiarmid. ON THE METHOD OF BOUNDED DIFFERENCES. En *Surveys in Combinatorics*, tomo 141 de *London Mathematical Society Lecture Note Series*, páginas 148–188. Cambridge University Press. 1989.

[Mee-2002] C. Meek, B. Thiesson y D. Heckerman. THE LEARNING-CURVE SAMPLING METHOD APPLIED TO MODEL-BASED CLUSTERING. *Journal of Machine Learning Research*, 2, páginas 397–418. 2002.

[Meh-1996] M. Mehta, R. Agrawal y J. Rissanen. SLIQ: A FAST SCALABLE CLASSIFIER FOR DATA MINING. *Lecture Notes in Computer Science*, 1057, páginas 18–32. 1996.

[Mer-1998] C. J. Merz. CLASSIFICATION AND REGRESSION BY COMBINING MODELS. Tesis Doctoral, University of California, Irvine. 1998.

[Mic-1985] R. S. Michalski. KNOWLEDGE REPAIR MECHANISMS: EVOLUTION VS. REVOLUTION. En *Proceedings of the 3rd International Workshop on Machine Learning*, páginas 116–119. 1985.

[Mic-1998] R. S. Michalski, I. Bratko y M. Kubat, editores. MACHINE LEARNING AND DATA MINING: METHODS AND APPLICATIONS. John Wiley & Sons Ltd. 1998.

[Mic-1994] D. Michie, D. Spiegelhalter y C. Taylor. MACHINE LEARNING, NEURAL AND STATISTICAL CLASSIFICATION. Ellis Horwood. 1994.

[Min-1989] J. Mingers. AN EMPIRICAL COMPARISON OF PRUNING METHODS FOR DECISION TREE INDUCTION. *Machine Learning*, 4, páginas 227–243. 1989.

[Moo-1993] R. J. Mooney. INDUCTION OVER THE UNEXPLAINED: USING OVERLY-GENERAL DOMAIN THEORIES TO AID CONCEPT LEARNING. *Machine Learning*, 10 (1), páginas 79–110. 1993.

[Mor-2000] L. Mora-López, I. Fortes-Ruiz, R. Morales-Bueno y F. Triguero-Ruiz. DYNAMIC DISCRETIZATION OF CONTINUOUS VALUES FROM TIME SERIES. *Lecture Notes in Artificial Intelligence*, 1810, páginas 280–291. 2000.

[Mor-2005] L. Mora-López, J. Mora, R. Morales-Bueno y M. S. de Cardona. MODELING TIME SERIES OF CLIMATIC PARAMETERS WITH PROBABILISTIC FINITE AUTOMATA. *Environmental Modelling & Software*, 20, páginas 753–760. 2005.

[Mot-1995] R. Motwani y P. Raghavan. RANDOMIZED ALGORITHMS. Cambridge University Press. 1995.

[Mur-1998] S. K. Murthy. AUTOMATIC CONSTRUCTION OF DECISION TREES FROM DATA: A MULTI-DISCIPLINARY SURVEY. *Data Mining and Knowledge Discovery*, 2, páginas 345–389. 1998.

[Nau-1991] G. E. Naumov. NP-COMPLETENESS OF PROBLEMS OF CONSTRUCTION OF OPTIMAL DECISION TREES. *Soviet Physics: Doklady*, 36 (4), páginas 270–271. 1991.

[New-1998] D. J. Newman, S. Hettich, C. L. Blake y C. J. Merz. UCI REPOSITORY OF MACHINE LEARNING DATABASES. University of California, Irvine, Department of Information and Computer Sciences. 1998.
`http://www.ics.uci.edu/~mlearn/MLRepository.html`.

[Noh-2003] J. Noh y K. M. Lee. APPLICATION OF MULTIATTRIBUTE DECISION-MAKING METHODS FOR THE DETERMINATION OF RELATIVE SIGNIFICANCE FACTOR OF IMPACT CATEGORIES. *Environmental Management*, 31 (5), páginas 633–641. 2003.

[Nor-1981] D. A. Norman. PERSPECTIVES ON COGNITIVE SCIENCE, capítulo What is cognitive science?, páginas 1–11. Ablex Publishing Corp. 1981.

[Nun-1991] M. Núñez. THE USE OF BACKGROUND KNOWLEDGE IN DECISION TREE INDUCTION. *Machine Learning*, 6, páginas 231–250. 1991.

[Nun-2005] M. Núñez, R. Fidalgo y R. Morales-Bueno. ON-LINE LEARNING OF DECISION TREES IN PROBLEMS WITH UNKNOWN DYNAMICS. *Lecture Notes in Artificial Intelligence*, 3789, páginas 443–453. 2005.

[Oka-1958] M. Okamoto. SOME INEQUALITIES RELATING TO THE PARTIAL SUM OF BINOMIAL PROBABILITIES. *Annals of the Institute of Statistical Mathematics*, 10 (1), páginas 29–36. 1958.

[Ort-1996] J. Ortega. EXPLOITING MULTIPLE EXISTING MODELS AND LEARNING ALGORITHMS. En *Proceedings of the Integrating Multiple Learned Models for Improving and Scaling Machine Learning Algorithms Workshop (IMLM-1996)*. 1996.

[Oza-2001] N. C. Oza. ONLINE ENSEMBLE LEARNING. Tesis Doctoral, Department of Electrical Engineering and Computer Science, University of California, Berkeley. 2001.

[Oza-2005] N. C. Oza. ONLINE BAGGING AND BOOSTING. En *Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics*, tomo 3, páginas 2340–2345. 2005.

[Par-2002] S. Parthasarathy. EFFICIENT PROGRESSIVE SAMPLING FOR ASSOCIATION RULES. En *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM-2002)*, páginas 354–361. 2002.

[Pen-1998] R. Peña-Marí. DISEÑO DE PROGRAMAS. FORMALISMO Y ABSTRACCIÓN. Prentice Hall. 1998.

[Pea-1979] J. Pearl. ENTROPY, INFORMATION AND RATIONAL DECISIONS. *Policy Analysis and Information Systems*, 3 (1), páginas 93–109. 1979.

[Per-2001] C. Pérez. TÉCNICAS ESTADÍSTICAS CON SPSS. Prentice Hall. 2001.

[Pic-2007] C. Pichuka, R. S. Bapi, C. Bhagvati, A. K. Pujari y B. L. Deekshatulu. A TIGHTER ERROR BOUND FOR DECISION TREE LEARNING USING PAC LEARNABILITY. En *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI-2007)*, páginas 1011–1016. 2007.

[Pol-2001] R. Polikar, L. Udpa, S. Udpa y V. Honavar. LEARN++: AN INCREMENTAL LEARNING ALGORITHM FOR SUPERVISED NEURAL NETWORKS. *IEEE Transactions on Systems, Man, and Cybernetics*, 31, páginas 497–508. 2001.

[Pro-1999] F. Provost y V. Kolluri. A SURVEY OF METHODS FOR SCALING UP INDUCTIVE ALGORITHMS. *Data Mining and Knowledge Discovery*, 2, páginas 131–169. 1999.

[Pro-1999a] F. Provost, D. Jensen y T. Oates. EFFICIENT PROGRESSIVE SAMPLING. En *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-1999)*, páginas 23–32. 1999.

[Qui-1979] J. R. Quinlan. EXPERT SYSTEMS IN THE MICROELECTRONIC AGE, capítulo Discovering rules by induction from large collections of examples, páginas 168–201. Edinburgh University Press. 1979.

[Qui-1983] J. R. Quinlan. LEARNING EFFICIENT CLASSIFICATION PROCEDURES AND THEIR APPLICATION TO CHESS END GAMES. En *Machine Learning: An Artificial Intelligence Approach*, páginas 463–482. Tioga. 1983.

[Qui-1986] J. R. Quinlan. INDUCTION OF DECISION TREES. *Machine Learning*, 1, páginas 81–106. 1986.

[Qui-1993] J. R. Quinlan. C4.5: PROGRAMS FOR MACHINE LEARNING. Morgan Kaufmann. 1993.

[Qui-1996] J. R. Quinlan. BAGGING, BOOSTING, AND C4.5. En *Proceedings of the 13th National Conference on Artificial Intelligence and 8th Innovative Applications of Artificial Intelligence Conference (AAAI/IAAI-1996)*, páginas 725–730. 1996.

[Rag-1988] P. Raghavan. PROBABILISTIC CONSTRUCTION OF DETERMINISTIC ALGORITHMS: APPROXIMATING PACKING INTEGER PROGRAMS. *Journal of Computer and System Sciences*, 37 (2), páginas 130–143. 1988.

[Ram-2000] G. Ramos-Jiménez y R. Morales-Bueno. A NEW METHOD FOR INDUCTION DECISION TREES BY SAMPLING. En *Proceedings of the Neurocolt Workshop on Applications of Learning Theory*. 2000.

[Ram-2000a] G. Ramos-Jiménez, R. Morales-Bueno y A. Villalba-Soria. CIDIM. CONTROL OF INDUCTION BY SAMPLE DIVISION METHODS. En *Proceedings of the International Conference on Artificial Intelligence (ICAI-2000)*, páginas 1083–1087. 2000.

[Ram-2001] G. Ramos-Jiménez. NUEVOS DESARROLLOS EN APRENDIZAJE INDUCTIVO. Tesis Doctoral, Universidad de Málaga, España. 2001.

[Ram-2004] G. Ramos-Jiménez, R. Morales-Bueno y J. del Campo-Ávila. TENDENCIAS DE LA MINERÍA DE DATOS EN ESPAÑA, capítulo IADEM-0: Un nuevo algoritmo incremental, páginas 91–98. Red Española de Minería de Datos. 2004.

[Ram-2005] G. Ramos-Jiménez, J. del Campo-Ávila y R. Morales-Bueno. APRENDIZAJE POR CAPAS BASADO EN SISTEMAS MULTICLASIFICADORES. En *Actas de la XI Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA-2005)*, páginas 113–122. 2005.

[Ram-2005a] G. Ramos-Jiménez, J. del Campo-Ávila y R. Morales-Bueno. E-CIDIM: ENSEMBLE OF CIDIM CLASSIFIERS. *Lecture Notes in Artificial Intelligence*, 3584, páginas 108–117. 2005.

[Ram-2005b] G. Ramos-Jiménez, J. del Campo-Ávila y R. Morales-Bueno. INDUCCIÓN DE ÁRBOLES DE DECISIÓN CON CIDIM: NUEVOS ENFOQUES. En *Actas del III Taller de Minería de Datos y Aprendizaje (TAMIDA-2005)*, páginas 13–20. 2005.

[Ram-2005c] G. Ramos-Jiménez, J. del Campo-Ávila y R. Morales-Bueno. INDUCTION OF DECISION TREES USING AN INTERNAL CONTROL OF INDUCTION. *Lecture Notes in Computer Science*, 3512, páginas 795–803. 2005.

[Ram-2005d] G. Ramos-Jiménez, J. del Campo-Ávila y R. Morales-Bueno. ML-CIDIM: MULTIPLE LAYERS OF MULTIPLE CLASSIFIER SYSTEMS BASED ON CIDIM. *Lecture Notes in Artificial Intelligence*, 3642, páginas 138–146. 2005.

[Ram-2006] G. Ramos-Jiménez, J. del Campo-Ávila y R. Morales-Bueno. FE-CIDIM: FAST ENSEMBLE OF CIDIM CLASSIFIERS. *International Journal of Systems Science*, 37 (13), páginas 939–947. 2006.

[Ram-2006a] G. Ramos-Jiménez, J. del Campo-Ávila y R. Morales-Bueno. INCREMENTAL ALGORITHM DRIVEN BY ERROR MARGINS. *Lecture Notes in Artificial Intelligence*, 4265, páginas 358–362. 2006.

[Ram-2006b] G. Ramos-Jiménez, J. del Campo-Ávila y R. Morales-Bueno. SISTEMAS MULTICLASIFICADORES Y DE APRENDIZAJE POR CAPAS BASADOS EN CIDIM. *Inteligencia Artificial. Revista Iberoamericana de IA*, 10 (29), páginas 49–58. 2006.

[Rap-2000] W. J. Rapaport. ENCYCLOPEDIA OF COMPUTER SCIENCE, capítulo Cognitive science. John Wiley and Sons Ltd. 2000.

[RAE-2001] Real Academia Española. DICCIONARIO DE LA LENGUA ESPAÑOLA. Espasa-Calpe, 22 edición. 2001.

[Rei-1988] R. E. Reinke y R. S. Michalski. INCREMENTAL LEARNING OF CONCEPT DESCRIPTIONS: A METHOD AND EXPERIMENTAL RESULTS. En *Machine Intelligence*, tomo 11, páginas 435–454. 1988.

[Rok-2005] L. Rokach y O. Maimon. TOP-DOWN INDUCTION OF DECISION TREES CLASSIFIERS – A SURVEY. *IEEE Transactions on Systems, Man, and Cybernetics – Part C: Applications and Reviews*, 35 (4), páginas 476–487. 2005.

[Rui-2005] J. Ruiz-Gómez, G. Ramos-Jiménez, J. del Campo-Ávila, A. García-Cerezo y R. Morales-Bueno. MODELADO DE UN ROBOT MÓVIL BASADO EN APRENDIZAJE INDUCTIVO. En *Actas de las VI Jornadas de Transferencia Tecnológica de Inteligencia Artificial (TTIA-2005)*, páginas 175–182. 2005.

[Rui-2005a] R. Ruiz, J. S. Aguilar-Ruiz, J. C. Riquelme y N. Díaz-Díaz. ANALYSIS OF FEATURE RANKINGS FOR CLASSIFICATION. *Lecture Notes in Computer Science*, 3646, páginas 362–372. 2005.

[Rum-1986] D. E. Rumelhart, G. E. Hinton y R. J. Williams. LEARNING INTERNAL REPRESENTATIONS BY ERROR PROPAGATION. En *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, tomo 1, páginas 318–362. MIT Press. 1986.

[Saa-2004] M. Saar-Tsechansky y F. Provost. ACTIVE SAMPLING FOR CLASS PROBABILITY ESTIMATION AND RANKING. *Machine Learning*, 54, páginas 153–178. 2004.

[Sch-1990] R. E. Schapire. THE STRENGTH OF WEAK LEARNABILITY. *Machine Learning*, 5, páginas 197–227. 1990.

[Sch-2002] T. Scheffer y S. Wrobel. FINDING THE MOST INTERESTING PATTERNS IN A DATABASE QUICKLY BY USING SEQUENTIAL SAMPLING. *Journal of Machine Learning Research*, 3, páginas 833–862. 2002.

[Sch-1986] J. C. Schlimmer y D. H. Fisher. A CASE STUDY OF INCREMENTAL CONCEPT INDUCTION. En *Proceedings of the 5th National Conference on Artificial Intelligence (AAAI-1986)*, páginas 496–501. 1986.

[Sch-1995] J. P. Schmidt, A. Siegel y A. Srinivasan. CHERNOFF-HOEFFDING BOUNDS FOR APPLICATIONS WITH LIMITED INDEPENDENCE. *SIAM Journal on Discrete Mathematics*, 8 (2), páginas 223–250. 1995.

[Sch-2007] M. Scholz y R. Klinkenberg. BOOSTING CLASSIFIERS FOR DRIFTING CONCEPTS. *Intelligent Data Analysis*. 2007. Aceptado para ser publicado.

[Scu-2006] D. Sculley y C. E. Brodley. COMPRESSION AND MACHINE LEARNING: A NEW PERSPECTIVE ON FEATURE SPACE VECTORS. En *Proceedings of the Data Compression Conference (DCC-2006)*, páginas 332–341. 2006.

[See-2001] A. K. Seewald, J. Petrak y G. Widmer. HYBRID DECISION TREE LEARNERS WITH ALTERNATIVE LEAF CLASSIFIERS: AN EMPIRICAL STUDY. En *Proceedings of the 14th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2001)*, páginas 407–411. 2001.

[Sha-1996] J. C. Shafer, R. Agrawal y M. Mehta. SPRINT: A SCALABLE PARALLEL CLASSIFIER FOR DATA MINING. En *Proceedings of the 22th International Conference on Very Large Data Bases (VLDB-1996)*, páginas 544–555. 1996.

[Sha-1948] C. E. Shannon. A MATHEMATICAL THEORY OF COMMUNICATION. *The Bell System Technical Journal*, 27, páginas 379–423, 623–656. 1948.

[Sie-1988] S. Siegel. ESTADÍSTICA NO PARAMÉTRICA. Trillas. 1988.

[Sie-1976] R. S. Siegler. THREE ASPECTS OF COGNITIVE DEVELOPMENT. *Cognitive Psychology*, 8, páginas 481–520. 1976.

[Sig-1989] V. G. Sigillito, S. P. Wing, L. V. Hutton y K. B. Baker. CLASSIFICATION OF RADAR RETURNS FROM THE IONOSPHERE USING NEURAL NETWORKS. *Johns Hopkins APL Technical Digest*, 10, páginas 262–266. 1989.

[Smi-1981] E. E. Smith y D. L. Medin. CATEGORIES AND CONCEPTS. Harvard University Press. 1981.

[Smi-1988] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler y R. S. Johannes. USING THE ADAP LEARNING ALGORITHM TO FORECAST THE ONSET OF DIABETES MELLITUS. En *Proceedings of the 12th Symposium on Computer Applications and Medical Care (SCAMC-1988)*, páginas 261–265. 1988.

[Sri-1995] R. Srikant y R. Agrawal. MINING GENERALIZED ASSOCIATION RULES. En *Proceedings of the 21st International Conference on Very Large Data Bases (VLDB-1995)*, páginas 407–419. 1995.

[Sri-1997] R. Srikant y R. Agrawal. MINING GENERALIZED ASSOCIATION RULES. *Future Generation Computer Systems*, 13 (2-3), páginas 161–180. 1997.

[Str-2001] W. N. Street y Y. Kim. A STREAMING ENSEMBLE ALGORITHM (SEA) FOR LARGE-SCALE CLASSIFICATION. En *Proceedings of the 7th ACM SIGKDD international conference on Knowledge Discovery and Data Mining (KDD-2001)*, páginas 377–382. 2001.

[R-2005] R. Development Core Team. R: A LANGUAGE AND ENVIRONMENT FOR STATISTICAL COMPUTING. R Foundation for Statistical Computing, Vienna, Austria. 2005. ISBN 3-900051-07-0.
`http://www.R-project.org`.

[The-2007] T. Theodosopoulos. A REVERSION OF THE CHERNOFF BOUND. *Statistics & Probability Letters*, 77, páginas 558–565. 2007.

[Tin-1997] K. M. Ting. DECISION COMBINATION BASED ON THE CHARACTERISATION OF PREDICTIVE ACCURACY. *Intelligent Data Analysis*, 1, páginas 181–205. 1997.

[Toi-1996] H. Toivonen. SAMPLING LARGE DATABASES FOR ASSOCIATION RULES. En *Proceedings of the 22nd International Conference on Very Large Data Bases (VLDB-1996)*, páginas 134–145. 1996.

[Utg-1989] P. Utgoff. PERCEPTRON TREES: A CASE STUDY IN HYBRID CONCEPT REPRESENTATIONS. *Connection Science*, 1, páginas 377–391. 1989.

[Utg-1988] P. E. Utgoff. ID5: AN INCREMENTAL ID3. En *Proceedings of the 5th International Conference on Machine Learning (ICML-1988)*, páginas 107–120. 1988.

[Utg-1994] P. E. Utgoff. AN IMPROVED ALGORITHM FOR INCREMENTAL INDUCTION OF DECISION TREES. En *Proceedings of the 11th International Conference on Machine Learning (ICML-1994)*, páginas 318–325. 1994.

[Utg-1997] P. E. Utgoff, N. C. Berkman y J. A. Clouse. DECISION TREE INDUCTION BASED ON EFFICIENT TREE RESTRUCTURING. *Machine Learning*, 29 (1), páginas 5–44. 1997.

[Val-1984] L. G. Valiant. A THEORY OF THE LEARNABLE. *Communications of the ACM*, 27 (11), páginas 1134–1142. 1984.

[Vap-1971] V. N. Vapnik y A. Y. Chervonenkis. ON THE UNIFORM CONVERGENCE OF RELATIVE FREQUENCIES OF EVENTS TO THEIR PROBABILITIES. *Theory of Probability and its Applications*, 16 (2), páginas 264–280. 1971.

[Vap-1982] V. N. Vapnik. ESTIMATION OF DEPENDENCES BASED ON EMPIRICAL DATA. Springer-Verlag, New York. 1982.

[Wal-1947] A. Wald. SEQUENTIAL ANALYSIS. Wiley. 1947.

[Wan-2003] H. Wang, W. Fan, P. Yun y J. Han. MINING CONCEPT-DRIFTING DATA STREAMS USING ENSEMBLE CLASSIFIERS. En *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, páginas 226–235. 2003.

[Wan-2005] H. Wang, S. Parthasarathy, A. Ghoting, S. Tatikonda, G. Buehrer, T. Kurc y J. Saltz. DESIGN OF A NEXT GENERATION SAMPLING SERVICE FOR LARGE SCALE DATA ANALYSIS APPLICATIONS. En *Proceedings of the 19th Annual International Conference on Supercomputing*, páginas 91–100. 2005.

[Wat-1993] T. L. Watkin, A. Rau y M. Biehl. THE STATISTICAL MECHANICS OF LEARNING A RULE. *Reviews of Modern Physics*, 65 (2), páginas 499–556. 1993.

[Wid-1996] G. Widmer y M. Kubat. LEARNING IN THE PRESENCE OF CONCEPT DRIFT AND HIDDEN CONTEXT. *Machine Learning*, 23, páginas 69–101. 1996.

[Wid-1960] B. Widrow y M. E. Hoff. ADAPTIVE SWITCHING CIRCUITS. En *IRE WESCON Convention Record*, páginas 96–104. 1960.

[Wil-1945] F. Wilcoxon. INDIVIDUAL COMPARISONS BY RANKING METHODS. *Biometrics Bulletin*, 1, páginas 80–83. 1945.

[Win-1994] W. L. Winston. OPERATIONS RESEARCH: APPLICATIONS AND ALGORITHMS. Duxbury Press. 1994.

[Wit-2005] I. H. Witten y E. Frank. DATA MINING: PRACTICAL MACHINE LEARNING TOOLS AND TECHNIQUES. Morgan Kaufmann, segunda edición. 2005.

[Wol-1992] D. H. Wolpert. STACKED GENERALIZATION. *Neural Networks*, 5, páginas 241–259. 1992.

[Wro-1997] S. Wrobel. AN ALGORITHM FOR MULTI-RELATIONAL DISCOVERY OF SUBGROUPS. En *Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery*, páginas 78–87. 1997.

[Ye-2003] N. Ye. THE HANDBOOK OF DATA MINING. Lawrence Erlbaum Associates. 2003.

[Yeh-2003] C. H. Yeh. THE SELECTION OF MULTIATTRIBUTE DECISION MAKING METHODS FOR SCHOLARSHIP STUDENT SELECTION. *International Journal of Selection and Assessment*, 11 (4), páginas 289–296. 2003.

[Zan-1998] S. H. Zanakis, A. Solomon, N. Wishart y S. Dublish. MULTI-ATTRIBUTE DECISION MAKING: A SIMULATION COMPARISON OF SELECT METHODS. *European Journal of Operational Research*, 107, páginas 507–529. 1998.

[Zha-2002] H. Zhang. THE ADDITIVE VS. MULTIPLICATIVE CHERNOFF BOUNDS. A FEW OTHER FORMS OF CHERNOFF-HOEFFDING BOUNDS. Lecture Notes for Course CS 174: Combinatorics and Discrete Probability University of California, Berkeley. 2002.

[Zhe-1998] Z. Zheng. NAIVE BAYESIAN CLASSIFIER COMMITTEES. *Lecture Notes in Computer Science*, 1398, páginas 196–207. 1998.