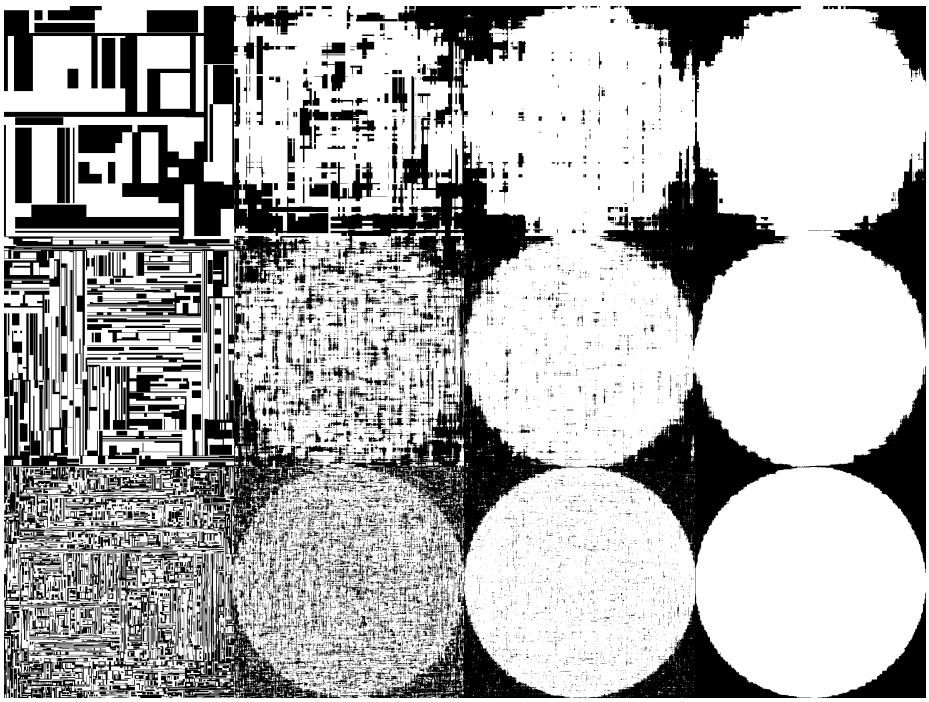
Universidad Autónoma de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Informática

# CLASIFICACIÓN MEDIANTE CONJUNTOS



TESIS DOCTORAL
FEBRERO 2006

GONZALO MARTÍNEZ MUÑOZ
DIRECTOR: ALBERTO SUÁREZ GONZÁLEZ

*A Lucia, Pietro y Nora*

# Agradecimientos

Agradezco muy sinceramente a mi Director de Tesis, D. Alberto Suárez González por su disponibilidad y apoyo durante todo el desarrollo de esta tesis. Sus sugerencias para orientar el trabajo de investigación y su lectura rigurosa de esta memoria han sido muy valiosas.

Agradezco a Pilar Rodríguez por sus consejos y por animarme a hacer la tesis en el Departamento.

Muchas gracias a Eduardo Pérez, lector designado por el Departamento, por su lectura minuciosa que ha contribuido a mejorar este documento.

Quiero agradecer a Francisco Rodríguez por su disponibilidad y por permitirme utilizar tiempo de CPU para realizar parte de los experimentos contenidos en esta tesis. Muchas gracias a Alejandro Sierra por facilitarme código fuente que he utilizado en algunos experimentos. Gracias también a Luis Fernando Lago que me ayudó técnica y moralmente con el arranque de esta memoria de tesis.

Agradezco a Jordi, mi compañero de despacho, por su buena compañía, por ponerme al día sobre los *clásicos de la informática*, así como por su apoyo con LaTeX en la recta final de la tesis.

Un agradecimiento a Raúl, con quien he compartido tantos años en la Autónoma: en los barracones del colegio Príncipe de Asturias antes, luego en la Facultad de Físicas y ahora como profesores de esta universidad.

Gracias a Antonio, que también ha compartido conmigo los años de la Facultad y muchos más.

Mis agradecimientos a la "gente del office", y en particular a Ana, Estrella, Alejandro, Paco, Paco, Pablo, Almudena, Ruth, Mariano, Elisa, Eugenio por los buenos ratos pasados juntos en el Departamento. Con sus tertulias de muy variada naturaleza, me han ayudado a desconectar del trabajo y de la tesis durante comidas y cafés.

También quiero agradecer a este Departamento y a los compañeros con los que he compartido asignatura.

Gracias a mis padres, hermanos, familia, *famiglia* y amigos.

Muchas gracias también a Lucia por lidiar con las fieras en Italia durante el verano mientras yo luchaba con la tesis en Madrid.

Figura 1: **Portada.** Mosaico compuesto por mapas de clasificación para un problema perfectamente separable de dos clases delimitadas por una circunferencia para el conjunto de clasificadores *class-switching* ($p = 0.4$). La columna y fila de la figura determinan el número de árboles combinados dentro del conjunto de clasificadores y el número de ejemplos utilizados para el entrenamiento respectivamente. Se han combinado $1$, $11$, $101$ y $1001$ árboles ( *de izquierda a derecha*) y se han utilizado $300$, $3000$ y $30000$ ejemplos de entrenamiento (*de arriba a abajo*)

# Índice general

# Índice de cuadros

# Índice de figuras

# Capítulo 1

# Introducción

Un clasificador es un sistema capaz de diferenciar elementos de acuerdo con sus características y agruparlos en órdenes o clases. La tarea es sencilla si se conocen las reglas para asignar una etiqueta de clase a dichos elementos a partir de sus atributos. El problema que se aborda en esta tesis en inducir las reglas de clasificación, cuando éstas son desconocidas, a partir de la información contenida en un conjunto de datos de entrenamiento. Este proceso de adquisición de conocimiento es denominado aprendizaje a partir de ejemplos o aprendizaje automático inductivo.

Para obtener este sistema de reglas se han diseñado un gran número de algoritmos de reconocimiento de patrones. Estos algoritmos se pueden dividir en dos grandes grupos. Por un lado, se encuentran los que parten de un conjunto de datos para los que se desconocen las clases en las que se pueden agrupar (clasificación no supervisada). Estas técnicas tratan de deducir cómo se agrupan los datos de acuerdo con sus características para proponer un esquema de clasificación. Por otro lado están los algoritmos de aprendizaje supervisado, en los que se dispone de un conjunto de datos con ejemplos de entrenamiento que han sido etiquetados previamente. El objetivo del aprendizaje supervisado es predecir la etiqueta de un nuevo elemento basándose en los atributos que lo caracterizan y utilizando las reglas inducidas a partir del conjunto de entrenamiento. Dentro del aprendizaje supervisado se distinguen dos tipos de problemas, dependiendo de la naturaleza de la etiqueta de clase. Se habla de clasificación cuando las posibles etiquetas de clase toman valores de entre un conjunto discreto. En caso de que los valores sean continuos se trata de un problema de regresión.

Las investigaciones presentadas en esta tesis versan sobre el diseño de algoritmos que generan modelos de clasificación partiendo de un conjunto de datos etiquetados. Con el fin de delimitar el ámbito de aplicación de los algoritmos propuestos haremos una serie de suposiciones sobre los problemas que se analizan. Primero, consideramos que los datos de entrenamiento utilizados por el algoritmo han sido obtenidos aleatoriamente por muestreo a

partir de las distribuciones de probabilidad (desconocidas) del problema. Asimismo, suponemos que las distribuciones de probabilidad de los problemas que analizamos tienen una variación suave. Además, suponemos que el muestreo de los ejemplos de entrenamiento se ha realizado con una frecuencia suficientemente alta como para que las distintas regiones del espacio de atributos relevantes a la clasificación estén bien representadas. Finalmente, dado que los modelos generados son estáticos, consideramos que las distribuciones de probabilidad del problema son estacionarias, es decir, que no cambian con el tiempo.

En general, el proceso completo de un sistema de reconocimiento automático se puede dividir en recolección de la información, selección y codificación de atributos, elección del algoritmo a aplicar y construcción y validación del modelo [Duda *et al.*, 2001]. Este proceso se representa esquemáticamente en la figura 1.1 y se describe a continuación.

El primer paso de todo sistema de reconocimiento de patrones es la recolección de la información relevante al problema mediante sensores u otros medios. En muchas ocasiones el diseñador del sistema de reconocimiento no podrá actuar sobre esta fase del diseño ya que el planteamiento del problema puede ser posterior a la recogida de la información.

A continuación, se debe elegir los atributos y codificarlos. Este paso es crítico, ya que no se podrán generar modelos eficaces si no se seleccionan características relevantes al problema de clasificación. El conocimiento experto sobre el problema puede ayudar a identificar los atributos más adecuados y facilitar así la tarea del algoritmo de clasificación. Tanto en la fase de recolección de la información como en la de codificación se puede introducir ruido en los datos, sea por errores de asignación de etiquetas, o por atributos cuyos valores son erróneos debido a fallos en los detectores, etc. Estos errores en las primeras fases generalmente limitan la fiabilidad de los modelos obtenidos.

Una vez que se dispone de los datos codificados se debe elegir el modelo que se considere más adecuado para el problema. Éste debe ser lo suficientemente complejo como para capturar la información contenida en los ejemplos y suficientemente robusto como para no ser sensible a fluctuaciones de muestreo u otros tipos de ruido en los datos. Generalmente, se tiene una preferencia (sesgo) por el modelo más sencillo posible que explique los ejemplos de entrenamiento (*navaja de Occam*, [Blumer *et al.*, 1990]). Este sesgo aplicado a aprendizaje automático indica que a igual error en los ejemplos disponibles para el entrenamiento se debe elegir el modelo menos complejo. Sin embargo, es importante hacer notar que esta preferencia no conduce necesariamente a la construcción de un clasificador que generalice mejor: un ejemplo no visto en entrenamiento estará bien clasificado exactamente por la mitad de las hipótesis compatibles con el conjunto de entrenamiento en problemas de dos clases con atributos discretos [Mitchell, 1980; 1990]. De hecho, se puede demostrar que, realizando un promedio uniforme sobre todos los problemas de clasificación, el error esperado de generalización cometido por todos los algoritmos de clasificación es el mismo (*No Free Lunch Theorem*, [Wolpert, 1995]). Por tanto, la elección del modelo de clasificación se debe basar en elegir familias de clasificadores cuyo sesgo permita identificar preferentemente patrones del mismo tipo que los que

aparecen en el problema concreto de clasificación que se esté abordando [Mitchell, 1980; 1990]. La experiencia previa en el diseño de estos sistemas puede ser muy útil para la rápida determinación del modelo a utilizar. Una vez elegido el modelo, éste se entrena con los datos de ejemplo y posteriormente se valida usando datos independientes de los empleados en el aprendizaje. Si los resultados no son los esperados y el modelo comete más errores de lo deseable entonces hay que replantearse uno o varios de los pasos previos. Es posible que haya que ajustar los parámetros del algoritmo ((a) en la figura 1.1) o que el modelo elegido no tenga un sesgo que le permita captar regularidades en el problema (b) o que los atributos no se hayan escogido correctamente (c) o que se haya partido de una información espuria o no relevante para el problema de clasificación (d). En cualquiera de estos casos se deberá retomar el proceso desde el punto donde se ha detectado el fallo.

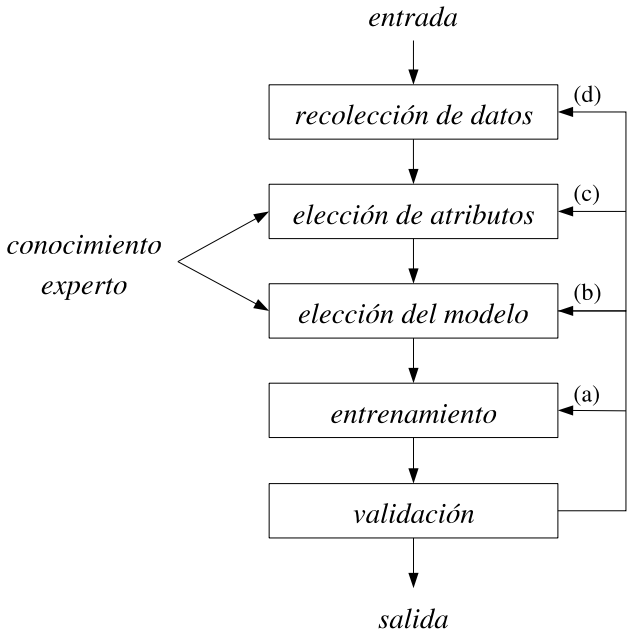



Figura 1.1: Diseño de un sistema de reconocimiento de patrones (adaptado de [Duda *et al.*, 2001])

El aprendizaje automático abarca una multitud de técnicas y de aplicaciones tanto de apoyo al experto como para sistemas autónomos. A continuación destacaremos algunas aplicaciones:

- Identificación de coberturas terrestres con imágenes de satélite. La generación automática o semi-automática de mapas de usos del suelo a partir de imágenes multiespectrales de satélite se basa en la clasificación de los valores digitales de los píxeles

que componen la imagen de una determinada porción de la superficie terrestre. En esta aplicación, el algoritmo es entrenado con unas muestras de "verdad terreno" (zonas de la imagen para las se conoce el uso del suelo mediante un muestreo previo sobre el terreno) para que reconozca la respuesta espectral de distintas coberturas del suelo: alfalfa, maíz, bosque de ribera, etc. Una vez terminado el entrenamiento, el ordenador clasifica todos los píxeles de la imagen a partir de sus valores digitales en las distintas bandas espectrales, generando así un mapa temático o de usos del suelo. Este tipo de proceso permite una actualización relativamente rápida y precisa de los mapas de uso del suelo, sin tener que recurrir a la foto interpretación. Un ejemplo de aplicación de estos mapas temáticos es la localización espacial de cultivos y la estimación de volúmenes de agua consumida en regadío, al multiplicar el área ocupada por cada cultivo por las dotaciones estándares de consumo de agua del cultivo [De Stefano y Montesinos, 2000].

- Biometría (*"métodos automáticos que analizan determinadas características humanas con el fin de identificar y autentificar a las personas"* [Tapiador Mateos *et al.*, 2005]). Es otro campo de aplicación del reconocimiento de patrones que ha recibido mucha atención en estos últimos años. La biometría incluye técnicas de reconocimiento automático de huella dactilar, iris, retina, escritura manuscrita, cara, voz. En definitiva, cualquier rasgo humano que pueda servir para la identificación de un individuo. Aparte de aplicaciones muy específicas de diversos colectivos profesionales, como la biometría forense en entornos judiciales, existe una serie de aplicaciones que se pueden implantar en la vida cotidiana de forma relativamente sencilla. Así por ejemplo, el reconocimiento en línea de firmas manuscritas puede ser una herramienta muy útil para hacer más segura cualquier tipo de transacción donde la firma sea lo que identifica a la persona, como es el cobro de un cheque o los pagos con tarjeta [Jain *et al.*, 2002].

- Detección de fraude en transacciones con tarjetas de pago. Otro enfoque para evitar este tipo de fraudes, que están implantando los grandes bancos y corporaciones de tarjetas de crédito, se basa en analizar la información de la transacción en sí más que en la autenticación del individuo. Se trata de un problema complejo por el volumen de datos con que se trabaja y porque el porcentaje de transacciones fraudulentas es muy bajo con respecto al total de transacciones, lo que hace que éstas sean difíciles de identificar [Chan *et al.*, 1999]. Además, la detección de fraude es un problema cambiante. Los infractores cambian a menudo sus hábitos para intentar eludir a los sistemas de detección de fraude [Fawcett y Provost, 1997]. Por tanto, los modelos utilizados se deberán actualizar periódicamente o deberán ser capaces de adaptarse a los cambios en el concepto a aprender. La información que se utiliza para la identificación de este tipo de fraude incluye datos del tipo: últimas transacciones realizadas, cuantías de las mismas, frecuencia de transacciones de la tarjeta, establecimientos

donde se realizan, etc [Dorronsoro *et al.*, 1997]. Estos sistemas no sólo permiten asignar un nivel de riesgo a cada transacción para así impedir la transacción (en sistemas en línea) o bloquear la tarjeta para futuras transacciones (cuando los sistemas operan sobre transacciones ya aceptadas) sino que también permiten identificar puntos de entrada de operaciones (comercios) donde se han podido realizar copias de tarjetas o donde se realizan transacciones fraudulentas habitualmente.

- Medicina. Es otro campo de aplicación muy importante de las diversas técnicas de reconocimiento de patrones. Existen varias revistas específicas, como *Artificial Intelligence in Medicine* o *Methods of Information in Medicine*. Gran parte de las aplicaciones se centran en la diagnosis y prognosis de pacientes. A partir de datos existentes de diagnósticos certeros se generan modelos que dan apoyo al especialista para el diagnóstico de futuros pacientes. Es deseable que estos sistemas tengan una precisión comparable o mejor que los médicos especialistas y que sean capaces de generar conocimiento transparente y hacer diagnósticos justificados. En general, un médico no cambiará su diagnóstico por el que propone un sistema experto si éste no es capaz de proporcionarle (junto con el diagnostico) los atributos o elementos que le han llevado a tomar esa decisión [Kononenko, 2001].

- Detección de correo comercial no solicitado (*spam*). La detección del correo basura se incluye ya en muchas aplicaciones de gestión del correo electrónico. Se trata de un problema difícil ya que enfoques basados en el mero filtro de mensajes que contienen determinadas palabras clave no dan buenos resultados. Esto se debe a que el formato y contenido del correo basura es cambiante [Fawcett, 2003]. Los emisores del correo basura modifican sus misivas para intentar eludir los filtros existentes de los clientes y servidores de correo. Por tanto, una buena herramienta de clasificación de mensajes deberá ser capaz de adaptarse a un concepto cambiante [Fawcett, 2003].

- Reconocimiento de caracteres. Actualmente, con la compra de cualquier escáner, el fabricante adjunta un software de reconocimiento de caracteres (sistemas OCR, *Optical character recognition* [Mori *et al.*, 1992]). Son herramientas de clasificación que parten de una imagen que contiene texto. Primero, la imagen es segmentada en bloques que corresponden a caracteres. Posteriormente se intenta identificar qué carácter hay en un bloque determinado para asignarle el código ASCII correspondiente. De este modo se puede disponer de documentos en formato texto en lugar de imágenes con texto no procesable. Este problema de clasificación ha sido abordado desde multitud de enfoques, destacamos [Mao, 1998] por utilizar conjuntos de clasificadores que son el tema principal de esta tesis.

- Otros ejemplos de aplicación interesantes incluyen predicción de fallos en discos duros a partir de atributos medidos por los propios discos. Entre los atributos utilizados se encuentran errores de lectura/escritura, altura de la cabeza lectora más alta o baja

de lo debido, temperatura, etc [Murray *et al.*, 2005]; categorización de texto [Schapire y Singer, 2000]; detección automática de interpretes: sistema entrenado sobre piezas de Chopin interpretadas por 22 pianistas expertos. El clasificador obtenido es capaz de identificar al interprete independientemente de la pieza que se le presente con una precisión mucho mayor que la que pueda dar un humano [Stamatatos y Widmer, 2005]; detección de fraude de clonación de tarjetas de móvil [Fawcett y Provost, 1997].

El objetivo de las investigaciones cuyos resultados se describen en este informe de tesis es el desarrollo y mejora de herramientas de clasificación supervisada de carácter general y aplicables a los problemas aquí expuestos. En concreto, el trabajo desarrollado explora diferentes aspectos de los conjuntos de clasificadores (*ensembles of classifiers*). Estas técnicas constituyen una de las cuatro direcciones fundamentales del aprendizaje automático identificadas por Dietterich [Dietterich, 1998b]. En dicho artículo Dietterich propone como problemas abiertos la mejora del error de clasificación mediante conjuntos de clasificadores, los métodos de escalado de algoritmos de aprendizaje supervisado, el aprendizaje por refuerzo y el aprendizaje de modelos estocásticos complejos. El desarrollo de conjuntos de clasificadores es un campo de investigación de gran actividad que ha dado lugar a multitud de publicaciones: [Freund y Schapire, 1995; Breiman, 1996a; Quinlan, 1996a; Breiman, 1998; Schapire *et al.*, 1998; Skurichina y Duin, 1998; Breiman, 1999; Bauer y Kohavi, 1999; Sharkey, 1999; Breiman, 2000; Dietterich, 2000b; Webb, 2000; Breiman, 2001; Rätsch *et al.*, 2001; Fürnkranz, 2002; Rätsch *et al.*, 2002; Bryll *et al.*, 2003; Hothorn y Lausen, 2003; Kim *et al.*, 2003; Chawla *et al.*, 2004; Martínez-Muñoz y Suárez, 2004b; Valentini y Dietterich, 2004; Hall y Samworth, 2005; Martínez-Muñoz y Suárez, 2005b]. Esta gran actividad se debe sobre todo a las significativas mejoras en la precisión de clasificación que se pueden obtener con esta técnica de sencilla implementación. Un conjunto de clasificadores clasifica nuevos ejemplos por decisión conjunta de sus componentes. Las decisiones de los clasificadores individuales se combinan, mediante voto, para obtener una clasificación final. Normalmente, de esta combinación resulta un conjunto de clasificadores que tiene más precisión que cada uno de los clasificadores de los que está compuesto. Obviamente, si se combinan clasificadores similares entre sí, la precisión del conjunto será aproximadamente igual a la de sus componentes. Por tanto, para mejorar el resultado de la clasificación por parte del conjunto, lo importante es generar clasificadores diversos cuyos errores no estén correlacionados, de forma que, al combinarlos, los errores de éstos tiendan a compensarse.

En esta tesis se proponen nuevos métodos de generación de conjuntos de clasificadores y heurísticas para la mejora por ordenación y poda de conjuntos generados con *bagging*. En concreto, las contribuciones realizadas en el trabajo son:

1. Se han propuesto tres nuevos métodos basados en el algoritmo de construcción de árboles *Algoritmo de crecimiento y poda iterativos* (IGP) [Gelfand *et al.*, 1991]. Este

algoritmo genera un árbol de decisión mediante la división de los datos de entrenamiento en dos subconjuntos. Una vez dividido el conjunto, se usa uno de los subconjuntos para hacer crecer el árbol y el otro para podarlo. El proceso se repite hasta alcanzar la convergencia, intercambiando los papeles de los conjuntos de datos en cada una de las iteraciones. Los métodos propuestos basados en IGP aprovechan el hecho de que distintas divisiones de los datos generan árboles diferentes. Esto permite que clasificadores generados con distintas particiones iniciales del conjunto de entrenamiento se puedan combinar para formar un conjunto de clasificadores, sin que sea necesario realizar remuestreos o introducir perturbaciones en el algoritmo de construcción del árbol, que generalmente reducen la capacidad de generalización de los árboles individuales generados. Los experimentos realizados ilustran que los métodos propuestos basados en el algoritmo IGP dan resultados equivalentes o mejores que otros métodos existentes (*bagging* y *boosting*) en los conjuntos de datos explorados. Presentan además un importante ahorro computacional respecto a conjuntos creados con árboles CART.

2. La diversidad entre los clasificadores incluidos en un conjunto de clasificadores es uno de los aspectos clave en el diseño de conjuntos de clasificadores [Dietterich, 2000a]. Se han realizado numerosos análisis sobre la dependencia entre la diversidad de los clasificadores individuales que forman parte del conjunto y la capacidad de generalización del conjunto [Dietterich, 2000b; Kuncheva y Whitaker, 2003]. A partir de estos trabajos y de un artículo de Breiman en el que se propone la modificación de las etiquetas de clase para generar conjuntos de clasificadores [Breiman, 2000], se ha propuesto un nuevo método de construcción de conjuntos de clasificadores. Este algoritmo, denominado *class-switching*, genera clasificadores con errores de entrenamiento no correlacionados mediante el uso de datos de entrenamiento en los que se han realizado modificaciones aleatorias de las etiquetas de clase. Asimismo, se muestra que para problemas de dos clases la evolución del error en el conjunto de entrenamiento con el número de clasificadores del conjunto *class-switching* se puede describir como un proceso de Bernoulli. El modelo de este proceso es independiente del problema de clasificación. Por otro lado el método *class-switching* muestra errores de generalización menores que *bagging* y equivalentes o menores que *boosting* en los conjuntos de datos analizados. Para alcanzar el nivel asintótico de error del conjunto es necesario generar conjuntos con un número elevado de clasificadores (en torno a 1000 clasificadores en los conjuntos estudiados).

3. Los conjuntos de clasificadores normalmente muestran un error de generalización que inicialmente disminuye a medida que se incrementa el número de clasificadores incluidos en el conjunto. Asintóticamente el error se estabiliza en un valor constante. Basándonos en las correlaciones entre los clasificadores del conjunto planteamos la hipótesis de que se puede modificar el orden de agregación original del conjunto de

forma que el error de generalización alcance un mínimo para un número de clasificadores menor que el del conjunto original completo. En este mínimo el error estaría por debajo del error asintótico del conjunto completo. Seleccionando este número de clasificadores se podría construir un subconjunto de clasificadores de menor tamaño y con mejor capacidad de generalización que el conjunto original. Este procedimiento de poda del conjunto mitigaría parcialmente algunos inconvenientes en el uso de los conjuntos de clasificadores, como son su abultado tamaño y menor velocidad de clasificación respecto a los clasificadores individuales de los que están compuestos. Estos aspectos han sido identificados por Dietterich como un problema abierto dentro de la investigación en conjuntos de clasificadores [Dietterich, 1998b]. Los experimentos realizados muestran que la ordenación de los clasificadores dentro de *bagging* es una herramienta útil para la identificación de subconjuntos de clasificadores más eficientes que el conjunto completo tanto en error de generalización como en velocidad de clasificación.

Los algoritmos diseñados han sido probados usando bases de datos sintéticas y bases de datos provenientes de distintos campos de aplicación contenidas en la colección de problemas de UCI [Blake y Merz, 1998].

Todo el desarrollo, tanto de los algoritmos de clasificación y de ordenación propuestos como de algunos de los algoritmos de referencia (*bagging* y *boosting*), ha sido realizado utilizando el lenguaje orientado a objetos C++ [Stroustrup, 1997].

La presente memoria describe el desarrollo de esta investigación en los siguientes capítulos:

En el capítulo 2 se presenta una introducción a la clasificación. Se describen los algoritmos de construcción de árboles de decisión CART (*Classification And Regression Trees*) [Breiman *et al.*, 1984] y C4.5 [Quinlan, 1993]. Además se describen brevemente los distintos grupos de técnicas existentes para la creación de conjuntos de clasificadores y se introducen los algoritmos de construcción de conjuntos de clasificadores que han sido desarrollados. Posteriormente, en este capítulo, se describen varios enfoques teóricos que permiten entender las razones por las que este tipo de algoritmos reduce el error de clasificación con respecto a los clasificadores elementales de los que están compuestos. Por una parte, se muestra el análisis de dichos algoritmos utilizando la descomposición del error en términos de sesgo (*bias*) y de varianza (*variance*). Por otra parte, se muestra cómo el aumento de los márgenes de clasificación que obtienen estos algoritmos puede explicar su funcionamiento. Finalmente, se describen y analizan en detalle algunos de los algoritmos de creación de conjuntos de clasificación más difundidos y que mejores resultados obtienen, como son *bagging* [Breiman, 1996a], *boosting* [Freund y Schapire, 1995], *wagging* [Bauer y Kohavi, 1999], *randomization* [Dietterich y Kong, 1995] o los bosques aleatorios (*random forests*) Forest-RI y Forest-RC [Breiman, 2001].

A continuación, esta tesis se estructura en dos partes que describen las distintas contribuciones realizadas. En una primera parte (capítulos 3 y 4) se detallan los nuevos métodos

de construcción de conjuntos de clasificadores desarrollados.

En el capítulo 3 se presentan los nuevos algoritmos de creación de conjuntos de clasificadores basados en el algoritmo IGP. Primero se describe el algoritmo de construcción de árboles IGP (*Iterative Growing and Pruning Algorithm*) [Gelfand *et al.*, 1991] que es utilizado para construir los clasificadores base en los conjuntos de clasificadores propuestos. A continuación se describen en detalle los tres algoritmos de construcción de clasificadores propuestos: conjunto de árboles IGP, *boosting* con arboles IGP y comités de árboles IGP. Posteriormente se muestran y describen los resultados de experimentos realizados utilizando *bagging*, *boosting* y los algoritmos propuestos.

El capítulo 4, también dentro de la primera parte, describe el método de generación de conjuntos *class-switching* por modificación aleatoria de etiquetas de clase. Para problemas de dos clases se analiza su funcionamiento modelizando la evolución del error de entrenamiento con el número de clasificadores del conjunto como un proceso de Bernoulli. Posteriormente se ilustra el funcionamiento del método *class-switching* mediante un sencillo ejemplo clasificación. Finalmente se compara experimentalmente el método *class-switching* con *bagging* y *boosting* en 15 problemas de clasificación.

La segunda parte de este trabajo de tesis (capítulo 5) presenta una serie de heurísticas de ordenación de conjuntos de clasificadores que permiten la poda de los mismos. Las heurísticas que se proponen son: reducción de error, medida de complementariedad, minimización de distancias de margen, ordenación por ángulos y ordenación basada en *boosting*. Posteriormente se muestran los resultados de probar estas heurísticas bajo distintas condiciones para analizar en detalle su comportamiento.

En el capítulo 6 se resumen los resultados obtenidos y se presentan las conclusiones globales del trabajo. Además se esbozan algunas futuras líneas de investigación.

En el apéndice A se muestran en detalle las características de las bases de datos utilizadas en las distintas pruebas experimentales llevadas a cabo a lo largo de este trabajo de investigación.

# Capítulo 2

# Clasificación

## 2.1. Clasificación supervisada y teoría de Bayes

En un problema de clasificación supervisada se parte de un conjunto $L$ de $N$ ejemplos etiquetados de la siguiente forma:

$$L = \{(\boldsymbol{x}_i, y_i), i = 1, 2, ..., N, y_i \in \{1, 2, \ldots, C\}\} \ , \tag{2.1}$$

donde cada ejemplo $(\boldsymbol{x}_i, y_i)$ está descrito por un vector de atributos $\boldsymbol{x}_i$ y una etiqueta de clase $y_i$ perteneciente a alguna de las $C$ clases del problema $\{1, 2, \ldots, C\}$. El vector de atributos puede incluir atributos categóricos o cuantitativos. Los categóricos son atributos cuyos valores no tienen un orden relevante al problema de clasificación (p. ej. el estado civil de una persona puede ser soltero, casado, viudo, etc. y generalmente se codificaría con un atributo de este tipo). Los atributos cuantitativos son atributos numéricos o cuyos valores tienen un orden relevante al problema de clasificación (p. ej. la edad de una persona). El objetivo de un algoritmo de clasificación es construir un clasificador que, dado un nuevo ejemplo sin etiquetar caracterizado por el vector de atributos $\boldsymbol{x}$ (no incluido necesariamente en $L$), prediga la clase $y$ a la que pertenece usando el conocimiento contenido en el conjunto de datos inicial $L$.

Una amplia descripción de los distintos métodos de clasificación y aprendizaje automático en general se pueden encontrar en las siguientes referencias: [Mitchell, 1997],[Duda *et al.*, 2001] y [Theodoridis, 2003]. Algunos grandes grupos de algoritmos de clasificación son: árboles de decisión [Breiman *et al.*, 1984; Quinlan, 1986; 1993], discriminantes lineales [Duda *et al.*, 2001], clasificadores basados en la teoría de Bayes como Naive-Bayes o redes bayesianas [Pearl, 1988; Jensen, 1996], vecinos más próximos y clasificadores basados en instancias [Aha *et al.*, 1991], redes neuronales [Haykin, 1999], máquinas de soporte vectorial [Vapnik, 1995; Burges, 1998], etc. Los

conjuntos de clasificadores, que son el tema central de esta tesis, pueden ser considerados como meta-clasificadores ya que no generan una hipótesis directamente sino que combinan las hipótesis obtenidas por otros algoritmos de clasificación [Wolpert, 1990; Freund y Schapire, 1995; Breiman, 1996a; Quinlan, 1996a]. En este capítulo se describe el funcionamiento de los árboles de decisión, que es el algoritmo de clasificación utilizado como base en este trabajo. En particular, se presenta en detalle el funcionamiento del algoritmo de creación de árboles CART, [Breiman *et al.*, 1984] y más someramente el algoritmo de construcción de árboles de decisión C4.5 [Quinlan, 1993].

Antes de describir los árboles de decisión, es oportuno hacer una breve descripción de las teorías estadísticas en las que se basan los algoritmos de resolución de problemas de clasificación y, más concretamente de la teoría de decisión de Bayes. Esta teoría parte de la hipótesis de que los problemas de clasificación se pueden analizar en términos probabilísticos. Consideremos un problema de clasificación en el que no se conoce el valor de ninguno de los atributos $\boldsymbol{x}$. ¿Cómo clasificaríamos un objeto del que no se conocen sus atributos pero sí las probabilidades a priori de pertenencia a una clase? Si debemos tomar una decisión lo mejor es optar por la clase más probable. Por ejemplo si un médico sabe que, para una enfermedad dada, el porcentaje de personas que sobreviven es del 90 % y le preguntan (sin conocer los resultados de los análisis) si un paciente concreto con dicha enfermedad sobrevivirá, el médico puede decir que es probable que sí. Esta cuantificación de la fiabilidad del diagnóstico en ausencia de otra evidencia se denomina probabilidad a priori y la denotaremos por $P(j)$, donde $j$ es el índice de la clase. La regla de decisión óptima para cuando no se conoce ningún atributo del objeto pero se conocen las probabilidades a priori de las clases a clasificar queda expresada matemáticamente como

$$j^{optima} = \underset{j}{argmax}\ P(j)\ . \tag{2.2}$$

Sin embargo en la mayoría de casos disponemos de más información para tomar una decisión. Un médico normalmente espera a conocer los resultados de los análisis para pronunciarse sobre un paciente concreto. Por tanto lo que realmente se quiere conocer es la probabilidad de pertenecer a cada una de las clases dado un valor para el vector de atributos, es decir, la probabilidad a posteriori $P(j|\boldsymbol{x})$. Consideremos que el vector de atributos $\boldsymbol{x}$ es una variable aleatoria cuya distribución en el espacio de atributos depende de la clase a la que pertenece. Definamos la distribución $p(\boldsymbol{x}|j)$ como la función de densidad de probabilidad para $\boldsymbol{x}$ dada la clase $j$. La probabilidad a posteriori se puede calcular a partir de $p(\boldsymbol{x}|j)$ y de las probabilidades a priori $P(j)$ mediante la regla de Bayes

$$P(j|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|j)P(j)}{p(\boldsymbol{x})} \tag{2.3}$$

donde

$$p(\boldsymbol{x}) = \sum_{j=1}^{C} p(\boldsymbol{x}|j)P(j) \ . \tag{2.4}$$

El criterio que minimiza la probabilidad de equivocarse tomando una decisión es escoger aquella clase que sea más probable para un vector de atributos $\boldsymbol{x}$, es decir

$$\text{Decidir } j \text{ si } P(j|\boldsymbol{x}) > P(k|\boldsymbol{x}) \text{ para todo } k \neq j \ . \tag{2.5}$$

Para un conjunto de datos se minimiza la probabilidad de error si y sólo si tomamos las decisiones de acuerdo con la ec. (2.5). Esta probabilidad mínima de error se denomina error de Bayes.

Para entender por qué el error de Bayes es el error mínimo alcanzable para cualquier problema de clasificación consideremos un problema de decisión unidimensional con $x$ como único atributo y con dos posibles clases 1 y 2. El clasificador divide el espacio en dos regiones $\mathcal{R}_1$ y $\mathcal{R}_2$ a las que asigna la clase 1 e 2 respectivamente. Por tanto el clasificador cometerá un error para la observación $x$ si $x \in \mathcal{R}_1$ y $x$ es de clase 2 o si $x \in \mathcal{R}_2$ y $x$ es de clase 1. La probabilidad de error para una clase $j$ es el resultado de multiplicar la probabilidad con que aparece dicha clase (probabilidad a priori $P(j)$) por la probabilidad con que aparece la clase en la región $\mathcal{R}_k$, donde el clasificador predice $k$ con $k \neq j$, esto es $P(x \in \mathcal{R}_k|j)$ (no confundir con la distribución de probabilidad $p(x|j)$ para la que utilizamos una notación con $p$ minúscula). La probabilidad de error total es

$$\begin{aligned} P(error) = P(x \in \mathcal{R}_1|2)P(2) + P(x \in \mathcal{R}_2|1)P(1) = \\ = \int_{\mathcal{R}_1} p(x|2)P(2)dx + \int_{\mathcal{R}_2} p(x|1)P(1)dx \ . \end{aligned} \tag{2.6}$$

En la construcción gráfica realizada en la figura 2.1 se observa que el valor $P(error)$ alcanza su mínimo cuando la división entre las regiones $\mathcal{R}_1$ y $\mathcal{R}_2$ se hace para $x = x_{bayes}$ ya que la región más obscura de la figura 2.1 no entra en la integral. Asimismo, se puede ver cómo es imposible reducir el error a cero, ya que hay intervalos (zonas rayadas en la figura 2.1) donde un mismo valor de $x$ puede corresponder a dos clases y por tanto lo único que se puede hacer es intentar minimizar la probabilidad de error según el resultado de la ec. (2.5).

En problemas reales el obtener la frontera óptima de división entre clases casi nunca es tarea fácil. En estos casos, generalmente, se puede estimar con cierta precisión las probabilidades a priori $P(j)$, pero no es fácil deducir las distribuciones de probabilidad de las clases $p(x|j)$ a partir de unos datos de entrenamiento limitados. El objetivo, por tanto, de la clasificación supervisada es construir un clasificador a partir de unos datos de entrenamiento etiquetados cuyo error sea lo menor posible, siendo el error de Bayes la cota inferior de dicho error.

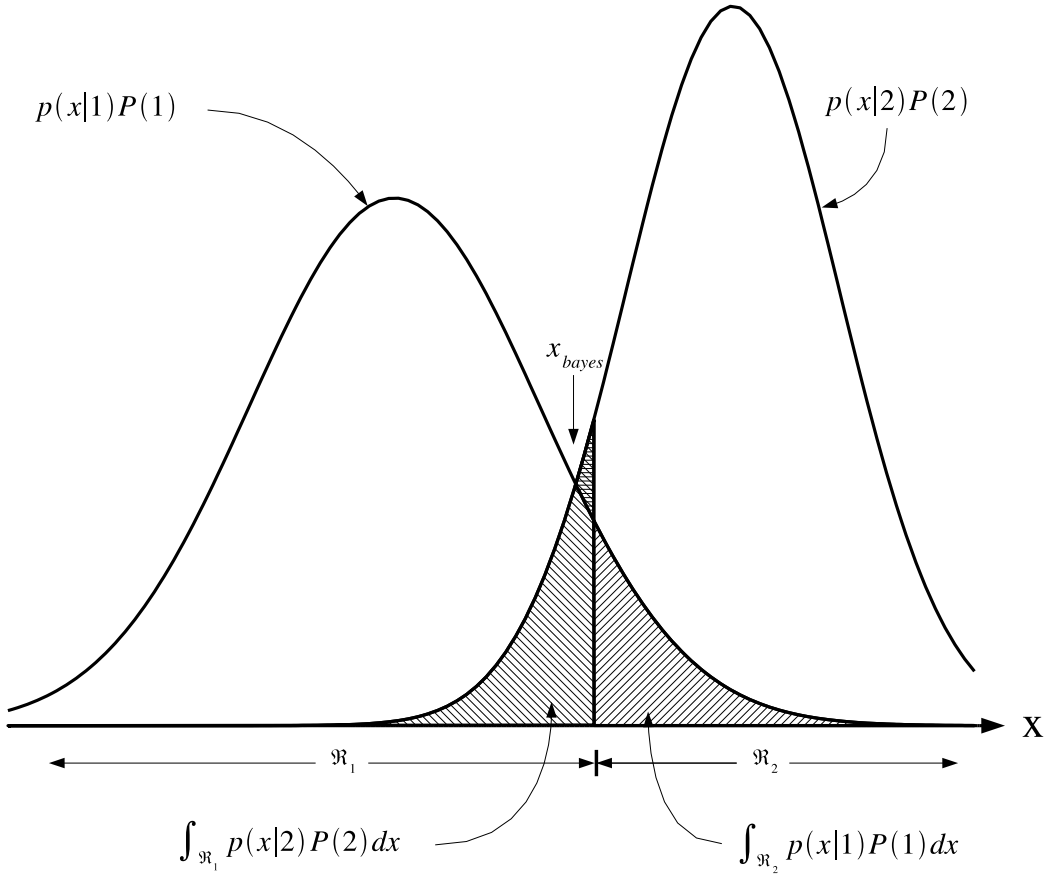



Figura 2.1: Distribuciones de probabilidad para un problema unidimensional de dos clases y probabilidad de error (*zonas rayadas*)

## 2.2. Árboles de decisión: CART y C4.5

En esta sección se describe el procedimiento general para la construcción de árboles de decisión centrándose principalmente en el algoritmo CART (*Classification And Regression Trees*) [Breiman *et al.*, 1984]. Se indican también de manera somera las características del algoritmo C4.5 [Quinlan, 1993] sobre todo en aquellos aspectos en los que difiere de CART.

Un árbol de decisión, que denotaremos por $T$, es un cuestionario jerárquico (un cuestionario en el cual la respuesta a una pregunta determina cuál es la siguiente pregunta) mediante el cual los ejemplos caracterizados por el vector $\boldsymbol{x}$ son asignados a regiones disjuntas del espacio de atributos. Cada una de estas regiones lleva asociada una etiqueta de clase $j$. Los ejemplos asignados por el cuestionario a dicha región son clasificados con la

clase $j$ correspondiente a la etiqueta de clase de dicha región. El cuestionario se puede representar mediante un árbol de decisión en el que a cada nodo interno $t$ se le asocia una de las preguntas del cuestionario jerárquico. La pregunta inicial del cuestionario se asocia al nodo raíz. Cada una de las regiones disjuntas en las que queda dividido el espacio de características corresponde a un nodo final o nodo hoja $t \in \hat{T}$, donde $\hat{T}$ denota a los nodos terminales del árbol $T$. Las divisiones utilizadas en CART son binarias: cada nodo interno $t$ tiene asociados dos nodos hijos $t_L$ y $t_R$ (nodo izquierdo y derecho respectivamente) cada uno de los cuales corresponde a respuesta (verdadero o falso) a la pregunta del nodo. En otro tipo de árboles de decisión, como C4.5, los nodos internos pueden tener más de dos descendientes. A cada uno de los nodos $t$ del árbol se le asocia una etiqueta $j(t)$ de clase que se elige de acuerdo con la clase mayoritaria de entre los ejemplos $\boldsymbol{x}_i$ de $L$ que pertenecen a la región definida por el nodo $t$, esto es

$$j(t) = \underset{j}{argmax}\ p(j|t)\ , \tag{2.7}$$

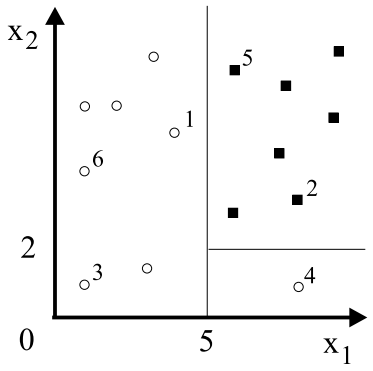
donde $p(j|t)$ es el la estimación de la probabilidad de que un ejemplo caracterizado por el vector de atributos $\boldsymbol{x}$ sea de clase $j$ dado que estamos en la región definida por el nodo $t$. Esta estimación se hace calculando el porcentaje de ejemplos de entrenamiento de clase $j$ que han sido asignados al nodo $t$,

$$p(j|t) = \frac{N_j(t)}{N(t)}\ , \tag{2.8}$$

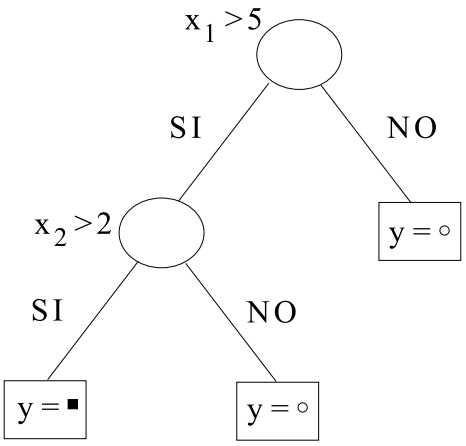
donde $N_j(t)$ es el número de ejemplos de clase $j$ asignados al nodo $t$ y $N(t)$ es el número total de ejemplos asignados al nodo $t$.

La construcción del árbol a partir del conjunto de datos de entrenamiento $L$ se hace mediante un proceso recursivo. Consideremos un nodo $t$ que es terminal en el momento actual del proceso de crecimiento del árbol. Este nodo corresponde a una región del espacio de atributos $U(t)$. A partir del nodo $t$ se generan dos hijos $(t_L, t_R)$ mediante un test booleano sobre los atributos. Esta división subdivide la región original $U(t)$ en dos regiones disjuntas $U(t_L)$ y $U(t_R)$ correspondientes a los nodos hijos $t_L$ y $t_R$. La división de los datos en las regiones $U(t_L)$ y $U(t_R)$ permite realizar una asignación más certera de la clase. Sin embargo, la subsiguiente división del espacio se hace con menos datos, por lo que está sujeta a mayor incertidumbre a causa de posibles errores de muestreo.

En la figura 2.2 se muestra un ejemplo de un árbol de decisión que divide el espacio de atributos en regiones correspondientes a dos clases: círculo y cuadrado. El gráfico de la parte superior izquierda de la figura muestra el espacio de atributos del problema de clasificación. En él se representan ejemplos de ambas clases. Algunos de estos ejemplos han sido enumerados en una tabla a la derecha del gráfico, indicando la clase a la que pertenecen. En el espacio de atributos también se han dibujado las líneas de división del espacio




| | $x_1$ | $x_2$ | y |
|---|---|---|---|
| 1 | 4 | 6 | ○ |
| 2 | 8 | 4 | ■ |
| 3 | 1 | 1 | ○ |
| 4 | 8 | 1 | ○ |
| 5 | 6 | 9 | ■ |
| 6 | 1 | 5 | ○ |
| … | … | … | … |




Reglas:
Si $x_1 > 5$ y $x_2 > 2$ entonces Cuadrado
si no entonces Círculo

Figura 2.2: Ejemplo de árbol de decisión

que genera el árbol de la parte inferior de la figura. Este árbol de decisión representa una solución posible para la división de ambas clases. Como se puede observar en el ejemplo de la figura 2.2 los árboles de decisión también se pueden representar como reglas. En este ejemplo el árbol de decisión corresponde a la regla

| | |
|---|---|
| Si $x_1 > 5$ y $x_2 > 2$ | la clase es Cuadrado |
| En caso contrario | la clase es Círculo . |

Esta correspondencia entre los árboles de decisión y conjuntos de reglas es una ventaja a la hora de la interpretación del modelo y de las decisiones generadas por el mismo.

Para conseguir la partición del espacio de atributos en regiones correspondientes a las distintas clases, los árboles de decisión utilizan una estrategia del tipo *divide y vencerás*. El resultado es que el espacio de atributos es segmentado. Para los atributos cuantitativos,

la estrategia más utilizada es dividir el espacio mediante hiperplanos, aunque también se podría dividir utilizando separaciones no lineales [Ittner y Schlosser, 1996]. En el ejemplo de la figura 2.2, dado que es un espacio bidimensional, estas divisiones son rectas. Para los atributos categóricos, las divisiones se realizan mediante particiones en subconjuntos de los distintos valores de los atributos. Las divisiones del espacio de atributos cuantitativos se pueden realizar utilizando bien hiperplanos de separación ortogonales a los ejes o bien oblicuos. Las divisiones ortogonales corresponden a preguntas sobre sólo uno de los atributos del espacio (como en el ejemplo de la figura 2.2) y son de la forma "¿$x_m \leq c$?" donde $m$ es el índice del atributo y el umbral de decisión, $c$, está dentro del rango de valores que puede tomar el atributo $x_m$. Estas divisiones se pueden calcular rápidamente por lo que se utilizan en la mayoría de algoritmos de creación de árboles de decisión. Otro posible método, implementado en CART, consiste en hacer divisiones oblicuas a los ejes. Estas divisiones corresponden a preguntas sobre el valor de una combinación lineal de los atributos ("¿$\sum_{m=1}^{N} a_m x_m \leq c$?"). Las divisiones oblicuas son mucho más expresivas que las divisiones paralelas a los ejes y pueden reflejar de manera más precisa las distribuciones de los datos. Las divisiones ortogonales son un caso particular de las oblicuas en las que todos los coeficientes excepto uno son nulos. Sin embargo el cálculo de la división oblicua óptima en cada nodo es más complicado, ya que el espacio de búsqueda de posibles divisiones es mayor. En CART las divisiones oblicuas se calculan con un método bastante eficaz y eficiente pero que no garantiza que la división sea óptima. En cualquier caso, el coste computacional de este tipo de divisiones es mucho mayor que el de las divisiones ortogonales. Además el hecho de utilizar tests más expresivos puede llevar a un sobreajuste a los datos de entrenamiento.

Para los atributos categóricos CART realiza preguntas de la forma "¿$x_m \in V$?" donde $V$ en un subconjunto de todos los posibles valores que puede tomar el atributo $x_m$. C4.5 puede generar divisiones de los atributos categóricos para cada nodo interno en más de dos subconjuntos y consecuentemente se obtienen más de dos nodos hijo.

La jerarquía de tests divide el espacio de atributos en regiones disjuntas: cada ejemplo se asigna a un solo nodo hijo dependiendo de la respuesta al test en el nodo padre. Otra posible arquitectura son los árboles de decisión borrosos donde cada ejemplo es asignado a todos los nodos hijos con un distinto grado de pertenencia [Chang y Pavlidis, 1977; Quinlan, 1993; Janikow, 1998; Suárez y Lutsko, 1999; Haskell *et al.*, 2004].

El tipo y los parámetros de la pregunta que determinan la división del espacio $U(t)$ de un nodo cualquiera $t$ se eligen mediante la minimización de una función local de coste. Esta función debe dividir el espacio $U(t)$ en dos regiones, $U(t_L)$ y $U(t_R)$, donde exista mayor homogeneidad de clases. El uso del error como función de coste podría parecer a priori la elección más acertada. Sin embargo, este criterio presenta dos inconvenientes [Breiman *et al.*, 1984]. El primero consiste en que es posible que ninguna de las divisiones posibles del espacio reduzca el error. Esto ocurre cuando en el nodo padre hay mayoría de ejemplos de una clase y todas las divisiones conducen a nodos hijos con mayoría de la misma clase.

El segundo defecto es menos cuantificable. Parece que este criterio no genera divisiones beneficiosas para el proceso global de construcción del árbol [Breiman *et al.*, 1984]. En el algoritmo CART [Breiman *et al.*, 1984] se elige una función local de coste $i(t)$ que selecciona para cada nodo $t$ la pregunta que maximiza la variación de la impureza del nodo para todas las divisiones posibles del conjunto de datos pertenecientes a $U(t)$. La variación de la impureza, $\Delta i(t)$, se define como

$$\Delta i(t) = i(t) - (i(t_L)p_L + i(t_R)p_R) \ ,$$

donde $p_R$ y $p_L$ son la proporción de ejemplos de contenidos en $U(t)$ que, después de la división, caen en los nodos hijos $t_R$ y $t_L$ respectivamente, esto es

$$p_L = \frac{p(t_L)}{p(t)} \ , \ p_R = \frac{p(t_R)}{p(t)} \ , \ p(t) = \frac{N(t)}{N} \ .$$

La función de impureza $i(t)$ se define en función de las probabilidades $p(j|t)$ de cada clase dentro de la región definida por el nodo $t$

$$i(t) = i(p(1|t), \ldots, p(C|t)) \ . \tag{2.9}$$

En [Breiman *et al.*, 1984] se establecen una serie de propiedades que debe cumplir la función de impureza definida en la ec. (2.9). Estas son:

1. Debe ser máxima sólo en el punto $(1/C, 1/C, \ldots, 1/C)$. Es decir, la impureza de un nodo es máxima si la distribución de ejemplos de cada clase es uniforme.

2. Debe alcanzar mínimos únicamente en los puntos: $(1, 0, \ldots, 0), (0, 1, \ldots, 0), \ldots$ y $(0, 0, \ldots, 1)$. Esto indica que la impureza de un nodo es mínima si sólo existen datos de una clase (nodo puro).

3. Suponiendo que todas las clases son equivalentes, debe ser una función simétrica en $p(1|t), p(2|t), \ldots$ y $p(C|t)$.

Una función de impureza $i(t)$ que cumpla estos criterios tiene la propiedad que $\Delta i(t) \geq 0$ para todo $t$ y toda posible división del espacio [Breiman *et al.*, 1984]. Esto es, la impureza nunca se incrementa cuando se hace crecer el árbol independientemente de cómo se elijan las divisiones. En cualquier caso se buscarán divisiones del espacio de atributos que conduzcan a la mayor homogeneidad de clases posible dentro de los nodos hijos. Se busca por tanto maximizar $\Delta i(t)$ con respecto a las divisiones posibles del espacio, $S$. Esto es

$$\max_{s \in S} \Delta i(s, t) = \max_{s \in S} [i(t) - (i(t_L)p_L + i(t_R)p_R)] \ . \tag{2.10}$$

La búsqueda del test óptimo para atributos continuos usando divisiones ortogonales a los ejes puede parecer costosa computacionalmente ya que el umbral puede tomar cualquier valor del rango de los reales. Sin embargo, sólo existe un número finito de divisiones que conduzcan a particiones de los datos de entrenamiento distintas. Consideremos un atributo ordinal $x_m$. Dado que estamos trabajando con un número $N$ finito de datos de entrenamiento, este atributo tiene como máximo $N$ valores distintos, que ordenados y eliminando los valores repetidos, los podemos denotar por $\{v_1, v_2, \ldots, v_n\}$ con $n \leq N$ y con $v_i < v_{i+1}$. Para cualquier umbral de corte $c$ elegido entre dos valores contiguos $v_i$ y $v_{i+1}$ se obtiene la misma variación de impureza (ec. (2.10)), ya que se divide el conjunto $\{v_1, v_2, \ldots, v_n\}$ en los mismos subconjuntos $\{v_1, v_2, \ldots, v_i\}$ y $\{v_{i+1}, v_{i+2}, \ldots, v_n\}$. Por tanto, el número de divisiones a comprobar para cada atributo ordinal usando divisiones ortogonales a los ejes es $n-1$ (cuyo valor máximo es $N-1$). El umbral elegido por CART una vez seleccionada la división es el punto medio ($(v_i + v_{i+1})/2$). En C4.5 se toma el umbral con valor $v_i$ con el fin de que los umbrales del árbol sean valores que aparecen en el conjunto de entrenamiento. La complejidad computacional de este enfoque para calcular la división óptima aumenta a medida que se incrementa el número de ejemplos $N$. Cuando el número de ejemplos $N$ supera un umbral prefijado $N_0$ para algún nodo interno de árbol, el algoritmo CART aplica submuestreos de los datos originales. Este submuestreo genera un nuevo conjunto de datos de tamaño $N_0$ con aproximadamente la misma distribución de clases que el conjunto de tamaño $N$. Sobre este nuevo conjunto se calcula el umbral de la división que posteriormente se aplica a todos los ejemplos para continuar con el proceso de construcción del árbol.

El análisis de las posibles divisiones para los atributos categóricos es más complejo, ya que el número de posibles subconjuntos no triviales para un atributo con $S$ posibles valores es como mínimo de $2^{S-1} - 1$. Esto hace inviable la evaluación de todas las posibles divisiones a partir de valores de $S$ no muy grandes. Breiman *et al.* demuestran que para problemas de clasificación de dos clases la búsqueda se puede realizar con un algoritmo cuya complejidad es de orden $S$ [Breiman *et al.*, 1984]. En caso contrario, CART, hace búsqueda exhaustiva. En C4.5 se utiliza una heurística para determinar estos subconjuntos.

La función de impureza, $i(t)$, que se elige en CART es el criterio de Gini. Este criterio cumple las propiedades previamente expuestas y viene definida por

$$i(t) = \sum_{i \neq j} p(i|t)p(j|t) \ , \tag{2.11}$$

donde los índices $i$ y $j$ del sumatorio son etiquetas de clase. El algoritmo C4.5 utiliza un criterio basado en la teoría de la información con $i(t) = -\sum_{j=1}^{Y} p(j|t) \log_2(p(j|t))$ (*gain criterion*). Como alternativa, Quinlan presenta una variante de la ec. (2.10) normalizada por la información que contiene cada división (*gain ratio criterion*) que evita que en problemas con atributos multivaluados se obtengan divisiones en los nodos internos del árbol con muchos nodos hijos. Se ha visto que los distintos criterios para la selección de las divisiones

del árbol generan árboles cuya capacidad de generalización es similar [Breiman *et al.*, 1984; Mingers, 1989b]. Las mayores diferencias de los distintos criterios se obtienen en el tamaño de los árboles obtenidos. En concreto *gain ratio criterion* es uno de los criterios que genera árboles más compactos [Mingers, 1989b].

Veamos según el criterio de Gini por qué en el ejemplo de la figura 2.2 se ha elegido como primera división del árbol $x_1 > 5$ y no $x_2 > 2$. Para ello hay que calcular la impureza del nodo raíz antes de la división y las impurezas de los nodos hijos después de hacer estas dos divisiones. Para simplificar el proceso utilizaremos sólo los datos presentados en la tabla de la figura 2.2. Partiendo de la estimación dada por la ec. (2.8) para $p(j|t)$ se obtiene que la impureza en el nodo raíz $t$ según la ec. (2.11) es

$$i(t) = 4/6 \times 2/6 = 2/9 \ .$$

Las impurezas de los nodos hijos después de la división $x_1 > 5$ son

$$\begin{aligned} i(t_L) &= 1/3 \times 2/3 = 2/9 \qquad \text{si } (x_1 > 5) \text{ (nodo izquierdo)} \\ i(t_R) &= 3/3 \times 0/3 = \ \ 0 \qquad \text{si } (x_1 \leq 5) \text{ (nodo derecho)} \end{aligned}$$

y la variación de impureza para la división $x_1 > 5$ es

$$\Delta i(t) = 2/9 - ((2/9) \times 1/2 + 0 \times 1/2) = 1/9$$

donde la proporción de ejemplos que se asigna a cada nodo es $p_R = p_L = 1/2$. Para la división $x_2 > 2$ se tiene: $i(t_L) = 2/4 \times 2/4 = 1/4$, $i(t_R) = 2/2 \times 0/2 = 0$, $p_L = 4/6$ y $p_R = 2/6$. Por lo que para $x_2 > 2$ la variación de impureza queda:

$$\Delta i(t) = 2/9 - ((1/4) \times 4/6 + 0 \times 2/6) = 1/18 \ .$$

Dado que $1/18 < 1/9$ tenemos que $x_1 > 5$ reduce más la impureza que $x_2 > 2$ y por tanto se elige como primera división del árbol según el criterio de Gini.

La subdivisión del espacio continúa de acuerdo con el procedimiento especificado hasta que, o bien se satisface un criterio de parada (prepoda), o bien se alcanzan todos los nodos terminales con ejemplos de una única clase (nodos puros), o no existe una división tal que los dos nodos hijos tengan algún dato. En general no se utilizan los criterios de prepoda (como por ejemplo $\Delta i(t) \leq \beta$), ya que detiene el proceso de división prematuramente en algunos nodos y demasiado tarde en otros, siendo difícil hacer que el crecimiento se pare uniformemente en todas las ramas del árbol de forma óptima [Breiman *et al.*, 1984]. La opción más utilizada es hacer crecer el árbol hasta que todos los nodos sean puros. Esto lleva a la generación de un árbol que se ajusta demasiado a los datos de entrenamiento pero que, a menudo, cuando se le presentan nuevos datos para clasificar, no tiene la suficiente

capacidad de generalización. Por ello hay que podarlo posteriormente. Mediante la poda generalmente se mejora la capacidad predictiva del árbol [Mingers, 1989a; Esposito *et al.*, 1997]. En el proceso de poda se eliminan nodos de la zona terminal del árbol, donde las preguntas se han generado con menos ejemplos y por tanto se tiene menos certeza de su validez.

Un modo sencillo para podar el árbol es estimar el error de clasificación en cada nodo utilizando un conjunto de datos independiente al utilizado para hacer crecer el árbol (*reduce error pruning*). Para ello se dividen los datos de entrenamiento $L$ en dos grupos $L_1$ y $L_2$, y se utiliza $L_1$ para generar el árbol y $L_2$ para podarlo. Para podar un árbol generado con el subconjunto $L_1$ se compara, con respecto al conjunto de datos $L_2$, el número de errores cometidos en un nodo interno $t$ con la suma de los errores de los nodos terminales que penden de $t$. Se poda si el error es mayor o igual en los nodos terminales que penden de $t$ que en el nodo $t$. El error que comete un árbol $T$ para un conjunto de datos cualquiera $L$ viene definido por la suma de los errores en todos sus nodos terminales

$$R(T, L) = \sum_{t \in \hat{T}} R(t, L) \ . \tag{2.12}$$

$R(t, L)$ es el error de un nodo $t \in T$ con respecto a un conjunto de datos $L$. A partir del conjunto de datos $L$ se puede estimar su valor mediante la expresión

$$R(t, L) = \frac{M(t, L)}{N} \ , \tag{2.13}$$

donde $M(t, L)$ es el número de ejemplos de $L$ tal que $\boldsymbol{x}_n \in U(t)$ y cuya clase $y$ es diferente de la clase $j(t)$ que predice el nodo y $N$ es el número de ejemplos de $L$. El criterio de poda queda como sigue:

$$R(t, L) \leq \sum_{u \in \hat{T},\, U(u) \subset U(t)} R(u, L) \ . \tag{2.14}$$

De acuerdo con este criterio de poda un árbol generado con unos datos $L$ no puede ser podado con el mismo conjunto de datos $L$ ya que no se obtendría poda alguna. En el proceso de crecimiento del árbol la impureza del árbol disminuye. Por tanto, también disminuye el error de los datos de entrenamiento utilizados. El inconveniente que presenta este método de poda es que reduce el número de ejemplos utilizados para el proceso de generación del árbol lo que no es recomendable [Esposito *et al.*, 1997]. Lo ideal es utilizar todos los ejemplos disponibles tanto para construir el árbol como para podarlo.

En el algoritmo CART los árboles se generan utilizando todos los datos disponibles en $L$ mientras que, para podar el árbol, se utiliza un criterio de poda (denominado poda de coste-complejidad) que tiene en cuenta, además del error, la complejidad del árbol. La idea detrás de este criterio es que, en general, para árboles con un error similar en $L$,

tendrá mayor capacidad de generalización aquél con menor complejidad, y para árboles con complejidad similar, tendrá mayor capacidad de generalización aquél con un error menor en $L$. Por tanto, el objetivo es llegar a un compromiso entre error y complejidad. En el algoritmo CART, la complejidad de un árbol $T$ se estima utilizando el número de nodos terminales del árbol $|\hat{T}|$. Posteriormente se elige el árbol podado $T^*$ que minimice la siguiente función de coste-complejidad:

$$\text{mín}\, R_\alpha(T^*), \qquad R_\alpha(T) = \alpha|\hat{T}| + \sum_{u\in\hat{T}} R(u, L)\ , \tag{2.15}$$

donde el parámetro $\alpha$ determina los pesos relativos en la función de coste del error y de la complejidad. El árbol podado $T^*$ que minimiza la ecuación (2.15) para un valor de $\alpha$ lo denotaremos como $T(\alpha)$. Variando $\alpha$ de 0 a infinito se puede obtener una familia de árboles podados. Esta familia es de tamaño finito dado que el árbol tiene un número finito de nodos. Para $\alpha = 0$ no se obtiene poda alguna, ya que $\alpha = 0$ significa que la complejidad no es penalizada y el árbol completo $T$ es el de menor error en $L$. Por otro lado, existe un $\alpha_K$ tal que para $\alpha \geq \alpha_K$ el árbol se podaría hasta el nodo raíz. Entre estos valores hay intervalos para el valor de $\alpha$ que nos definen una familia de posibles árboles podados a partir de $T$:

$$T = T_0 \geq T_1 \geq \cdots \geq T_K = raiz(T)$$

Donde:
-$T_0$ se obtiene para $\alpha < a_1$
-$T_k$ se obtiene para $\alpha_k \leq \alpha < a_{k+1}$ con $k = 1, 2, \ldots, K-1$
-$T_K$ se obtiene para $\alpha \geq a_K$ (2.16)

El siguiente paso es estimar el intervalo de $\alpha$ que nos da el árbol podado óptimo según la ecuación (2.15). En CART $\alpha$ se estima construyendo árboles auxiliares por validación cruzada. Para ello se dividen los datos $L$ en un número $V$ de grupos disjuntos (normalmente $V = 10$) tal que

$$\begin{gathered} L = L_1 \cup L_2 \cup \cdots \cup L_V \qquad \text{y} \\ \oslash = L_i \cap L_j \ \text{ para } \ i = 1, 2, \ldots, V \text{ con } i \neq j\ . \end{gathered} \tag{2.17}$$

Posteriormente, y utilizando los siguientes conjuntos de datos $L^{(v)} = L - L_v$ para $v = 1, 2, ..., V$, se construyen $V$ árboles que denominaremos $T^{(v)}$ para $v = 1, 2, ..., V$. De esta forma cada árbol es generado con un $100(V-1)/V$ por ciento de los datos. Por tanto, cada árbol $T^{(v)}$ dispone de un $100/V$ por ciento de datos (esto es, el conjunto $L_v$) que no se ha utilizado para crecer el árbol y que se puede usar para estimar parámetros óptimos del árbol $T^{(v)}$. Por ejemplo, podemos calcular el parámetro óptimo $\alpha$ para podar

el árbol $T^{(v)}$. Para ello es suficiente calcular la familia de árboles podados que minimizan la ec. (2.15) para cada intervalo posible de $\alpha$ para $L^{(v)}$ tal como viene definido en la ec. (2.16). Posteriormente se elige, de la familia de árboles generados, el árbol que tenga menor error para el conjunto de datos $L_v$ estimado con la ec. (2.12). El árbol con error mínimo definirá el intervalo de poda $\alpha$ para el árbol $T^{(v)}$.

Sin embargo, es necesario estimar el valor de $\alpha$ óptimo para podar el árbol $T$ construido con todos los datos. Se podría utilizar la media de los $\alpha$ obtenidos para cada uno de los $V$ árboles $T^{(v)}$ para podar el árbol $T$. El problema que presenta esta solución es que los distintos valores de $\alpha$ para los árboles $T^{(v)}$ y para el árbol $T$ no tienen por qué ser equivalentes, por lo que la media de los $\alpha$ óptimos de los árboles $T^{(v)}$ puede dar un valor inválido para $T$. Se deberá buscar, por tanto, un valor de $\alpha$ de entre los intervalos de $\alpha$ que determinan la poda del árbol $T$. La solución que adopta CART es la siguiente: para cada uno de los árboles $T^{(v)}$ y para un valor de $\alpha$ dentro de cada uno de los intervalos de $\alpha$ de $T$ se obtiene el árbol podado $T^{(v)}(\alpha)$ utilizando los datos $L^{(v)}$ siguiendo el criterio de poda de la ecuación (2.15). A continuación se estima el error de cada uno de estos árboles $T^{(v)}(\alpha)$ con respecto al conjunto de datos independientes $L_v$ con la ec. (2.12). Finalmente se elige el valor de $\alpha$ que minimiza el error medio de los árboles podados $T^{(v)}(\alpha)$, esto es

$$\min_k R^{cv}(T^*) = \min_k \frac{1}{V} \sum_{v=1}^{V} R_v(T^{(v)}(\sqrt{\alpha_k \alpha_{k+1}})) \ , \quad k = 1, 2, \ldots, K-1 \tag{2.18}$$

donde $R_v$ es el error cometido con respecto al conjunto de datos $L_v$ utilizando la ecuación (2.12) y donde los valores de $\alpha$ dentro de cada intervalo de poda del árbol $T$ utilizados vienen dados por $\sqrt{\alpha_k \alpha_{k+1}}$. El valor de $\alpha$ que minimiza la ec. (2.18) junto con la ec. (2.15) nos determinan el árbol $T(\alpha)$ podado a partir de $T$.

El algoritmo C4.5 usa criterio de poda basado en una estimación pesimista del error de cada nodo (poda basada en error). Para ello substituye el número de errores cometidos en cada nodo por el límite superior de confianza de una distribución binomial (donde los ejemplos del nodo $N(t)$ son los ensayos y los errores del nodo $M(t, L)$ son los "éxitos" de la distribución binomial) multiplicado por el número de ejemplos del nodo $N(t)$. En la exhaustiva comparativa de distintos métodos de poda realizada por Esposito *et al.* observaron que la poda basada en error de C4.5 tiende a podar menos de lo necesario mientras que la poda de coste-complejidad de CART tiende a generar árboles más pequeños de la poda óptima [Esposito *et al.*, 1997]. La poda pesimista que implementa C4.5 tiene la ventaja de que es computacionalmente muy rápida aunque en determinados problemas genera árboles que no generalizan bien [Mingers, 1989a]. Por otro lado la poda por validación cruzada de coste-complejidad es más lenta pero presenta la ventaja de proporcionar una familia de árboles que puede ser analizada y comparada por un experto humano [Mingers, 1989a].

## 2.3. Conjuntos de clasificadores

Los conjuntos de clasificadores (*ensembles of classifiers*) son sistemas que clasifican nuevos ejemplos combinando las decisiones individuales de los clasificadores de los que están compuestos. Los conjuntos de clasificadores se construyen en dos fases: en una primera fase, la fase de entrenamiento, se genera una serie de clasificadores (a cada uno de ellos lo denominaremos clasificador individual o clasificador base) con un algoritmo concreto (que denominaremos algoritmo base). En una segunda fase se combinan las distintas hipótesis generadas. La precisión del conjunto puede ser mucho mayor que la precisión de cada uno de los miembros en los que está compuesto como han demostrado multitud de estudios [Freund y Schapire, 1995; Breiman, 1996a; Quinlan, 1996a; Breiman, 1998; Schapire *et al.*, 1998; Skurichina y Duin, 1998; Breiman, 1999; Bauer y Kohavi, 1999; Sharkey, 1999; Breiman, 2000; Dietterich, 2000b; Webb, 2000; Breiman, 2001; Rätsch *et al.*, 2001; Fürnkranz, 2002; Rätsch *et al.*, 2002; Bryll *et al.*, 2003; Hothorn y Lausen, 2003; Kim *et al.*, 2003; Chawla *et al.*, 2004; Martínez-Muñoz y Suárez, 2004b; Valentini y Dietterich, 2004; Hall y Samworth, 2005; Martínez-Muñoz y Suárez, 2005b]. Esta mejora se podrá obtener únicamente si los clasificadores individuales son suficientemente diversos: combinar clasificadores idénticos no conlleva ninguna mejora; de hecho se obtendría la misma respuesta que cada clasificador base. Por tanto para construir un conjunto de clasificadores, hay que elegir el algoritmo base y diseñar una metodología que sea capaz de construir clasificadores que cometan errores distintos en los datos de entrenamiento.

Las distintas técnicas desarrolladas para la generación de conjuntos de clasificadores (primera fase) se pueden agrupar en [Dietterich, 1998b; 2000a]:

- **Técnicas basadas en remuestreo de los datos de entrenamiento**: Algunos de los métodos de generación de conjuntos de clasificadores más importantes, como *boosting* [Freund y Schapire, 1995] y *bagging* [Breiman, 1996a], pertenecen a está categoría. Este grupo de técnicas introduce perturbaciones en los datos de entrada (eliminación de ejemplos, repetición de ejemplos, distintas ponderaciones de los ejemplos, etc.) para obtener cada uno de los clasificadores individuales. La variabilidad requerida dentro del conjunto es obtenida mediante modificaciones de la distribución de entrenamiento (que se supone debe parecerse a la distribución real) y así inducir variaciones en los clasificadores individuales. Para que se genere la suficiente variabilidad entre clasificadores, el algoritmo base debe tener una cierta inestabilidad frente a los cambios. Los árboles de decisión poseen características que los convierte en buenos algoritmos base para este grupo de técnicas ya que pequeñas variaciones en los datos de entrenamiento pueden hacer que las estructuras de los árboles generados sean completamente diferentes.

  Generalmente se considera que los algoritmos de clasificación vecino más próximo y discriminante lineal no son adecuados [Breiman, 1996a; Dietterich, 1998b].

Estos clasificadores son bastante estables frente a modificaciones de los datos de entrenamiento y no se obtendría la variedad de clasificadores necesaria para que el conjunto generado mejore la capacidad de generalización del clasificador base. Bajo determinadas condiciones, se pueden construir conjuntos de discriminantes lineales que mejoran el rendimiento de un sólo clasificador. Esto se consigue solamente en situaciones donde el clasificador lineal se hace muy inestable, como se muestra experimentalmente en la referencia [Skurichina y Duin, 1998]. En otra referencia de los mismos autores [Skurichina y Duin, 2002] se hace un estudio detallado de distintos tipos de discriminantes lineales y de conjuntos de clasificadores (*bagging*, *boosting* y subespacios aleatorios) para hacer una "guía de uso". En esta guía indican con qué conjuntos de clasificadores se pueden obtener mejoras respecto al discriminante lineal individual dependiendo del tamaño del conjunto de entrenamiento. Vecino más próximo junto con *bagging* estándar obtiene los mismos resultados que vecino más próximo ejecutado sobre todos los datos [Breiman, 1996a]. Sin embargo, se ha visto recientemente que se pueden obtener mejoras significativas combinando vecino más próximo junto con *bagging* siempre que el tamaño del conjunto remuestreado contenga menos del 50 % de los ejemplos originales. Además se puede demostrar que si el porcentaje de remuestreo tiende a $0$ mientras que los datos de entrenamiento tienden a infinito entonces el error del conjunto de *bagging* con vecinos próximos tiende al error de Bayes [Hall y Samworth, 2005].

*Boosting* construye clasificadores mediante la asignación de pesos a los ejemplos de forma adaptativa. En cada iteración de *boosting* se construye un clasificador que intenta compensar los errores cometidos previamente por otros clasificadores. Para lograr que cada nuevo clasificador mejore los resultados en regiones donde fallan los anteriores se utiliza un conjunto de datos ponderado cuyos pesos son actualizados tras cada iteración: se incrementan los pesos de los ejemplos mal clasificados por el último clasificador y se reducen los pesos de los bien clasificados. *Boosting* puede o bien utilizar todos los ejemplos ponderados para construir cada clasificador (*boosting* con *reweighting*), o bien hacer un remuestreo ponderado (*boosting* con *resampling*) donde tengan más probabilidad de aparecer en la muestra los ejemplos con mayor peso. En cualquier caso, el algoritmo de clasificación base se encuentra con un conjunto de entrenamiento con ejemplos con distinta importancia relativa. De hecho, cada nuevo clasificador individual se centra en la clasificación de los ejemplos más difíciles que han sido erróneamente clasificados por los clasificadores previos. *Boosting* es uno de los métodos más eficientes para la construcción de conjuntos de clasificadores. Sin embargo, presenta dificultades de generalización en algunos problemas y cuando los datos tienen ruido en la asignación de etiquetas de clase [Quinlan, 1996a; Opitz y Maclin, 1999; Dietterich, 2000b].

Otra técnica ampliamente utilizada es *bagging* (*Bootstrap sampling and aggregation*) [Breiman, 1996a]. *Bagging* no utiliza ningún tipo de ponderación de los datos. Cada clasificador del conjunto se obtiene utilizando una muestra aleatoria con repetición del mismo número de ejemplos que el conjunto de datos de entrenamiento (muestra *bootstrap*). En media, cada muestra contiene el $63.2\,\%$ de los datos originales y el resto son ejemplos repetidos. Por tanto, en *bagging*, cada clasificador se genera con un conjunto reducido de los datos de entrenamiento. Esto significa que los clasificadores individuales son algo peores que los clasificadores construidos con todos los datos. Esta peor capacidad de generalización se compensa mediante la combinación de los clasificadores. *Bagging* es generalmente más robusto que *boosting* frente a fallos en las asignaciones de etiquetas de clase y generalmente mejora el error del algoritmo base [Quinlan, 1996a; Opitz y Maclin, 1999; Dietterich, 2000b].

- **Manipulación de los atributos**: Esta técnica descarta selectivamente el uso de atributos de los datos de entrada para construir los clasificadores individuales. De esta forma se construyen clasificadores en distintos subespacios de atributos. La selección de los atributos a eliminar se debe hacer cuidadosamente, ya que, si eliminamos algún atributo importante, la precisión de los clasificadores puede verse afectada [Tumer y Ghosh, 1996]. Otro ejemplo de este tipo de técnicas se presenta en [Ho, 1998] donde para construir cada clasificador se descarta un subconjunto aleatorio de los atributos de entrada. De esta forma cada clasificador individual trabaja sobre un subespacio aleatorio de atributos originales. Un enfoque similar se sigue en el método *attribute bagging* donde se generan subconjuntos aleatorios de atributos de tamaño fijo para construir cada clasificador [Bryll *et al.*, 2003].

- **Manipulación de las etiquetas de clase**: Cada clasificador individual es construido usando una recodificación de las etiquetas de clase de los datos de entrenamiento. En [Dietterich y Bakiri, 1995] se presenta el método de conjuntos ECOC (*Error-Correcting Output Codes*). Esta técnica reasigna aleatoriamente las $Y$ clases de un problema en dos clases ficticias para construir cada clasificador. En [Schapire, 1997] se aplica ECOC junto con *AdaBoost* para dar buenos resultados. En el método *round robin* se genera un clasificador para cada par de clases del problema [Fürnkranz, 2002]. De esta forma se transforma un problema de $C$ clases en $C(C-1)/C$ problemas de dos clases. Estos métodos, sin embargo, tienen la limitación de poder ser aplicados sólo a problemas de clasificación con muchas clases. Otros algoritmos, en vez de hacer una reasignación, intercambian aleatoriamente las clases de los ejemplos de entrenamiento [Breiman, 2000; Martínez-Muñoz y Suárez, 2005b] para construir cada clasificador base de forma que no tienen limitaciones con el número de clases del problema.

- **Técnicas basadas en la introducción de aleatoriedad en el algoritmo de aprendizaje**: Esta familia de técnicas introduce un cierto grado de aleatoriedad en el algoritmo base de aprendizaje, de forma que dos ejecuciones distintas con los mismos datos resultan en dos clasificadores diferentes. En general, esta técnica empeora la precisión del algoritmo de clasificación a cambio de obtener una mayor variabilidad en los clasificadores obtenidos para poder combinarlos. Un ejemplo de este tipo de técnicas es *randomization*, método que elige al azar entre las $k$ mejores preguntas que se pueden hacer en el nodo de un árbol de decisión [Dietterich y Kong, 1995; Dietterich, 2000b] o Forest-RI que en cada nodo selecciona la mejor pregunta dentro de un subconjunto aleatorio reducido de los atributos de entrada [Breiman, 2001]. Otro ejemplo consiste en generar las divisiones en los nodos internos del árbol de manera completamente aleatoria, tanto en la selección del atributo como en la elección del umbral de corte [Fan *et al.*, 2003]. El problema con estas técnicas es determinar la cantidad o el tipo de aleatoriedad a introducir en el algoritmo de forma que produzca el efecto deseado; esto es, aumentar variabilidad sin empeorar demasiado la precisión de cada uno de los clasificadores, lo que llevaría a no obtener mejora con el conjunto de clasificadores.

Existe otra familia de algoritmos denominada bosques aleatorios (*random forests*) [Breiman, 2001] que puede incorporar características de las diversas técnicas previamente expuestas. Se trata de técnicas de conjuntos de clasificadores que utilizan específicamente árboles de decisión como algoritmo base. Breiman define un bosque aleatorio como un clasificador compuesto por árboles de decisión donde cada árbol $h_t$ ha sido generado a partir del conjunto de datos de entrenamiento y de un vector $\mathbf{\Theta}_t$ de números aleatorios idénticamente distribuidos e independientes de los vectores $\mathbf{\Theta}_1, \mathbf{\Theta}_2, \ldots, \mathbf{\Theta}_{t-1}$ previamente utilizados para generar los clasificadores $h_1, h_2, \ldots, h_{t-1}$ respectivamente. Ejemplos de bosques aleatorios son: *bagging* usando árboles [Breiman, 1996a], subespacios aleatorios [Ho, 1998], *randomization* [Dietterich y Kong, 1995; Dietterich, 2000b], Forest-RI y Forest-RC [Breiman, 2001], *double-bagging* usando árboles [Hothorn y Lausen, 2003] o *class-switching* usando árboles [Martínez-Muñoz y Suárez, 2005b].

En lo que se refiere a la fase de combinación de clasificadores se pueden agrupar los distintos algoritmos de acuerdo con su arquitectura como [Jain *et al.*, 2000]:

- **Paralela**: Todos los clasificadores base son invocados y sus decisiones son combinadas. La mayoría de los conjuntos de clasificación pertenecen a esta categoría. Una extensión de este grupo es la arquitectura paralela ponderada (*gated parallel*) donde la salida de cada clasificador base es seleccionada o ponderada de acuerdo con algún criterio de combinación. En *bagging* y *boosting* la combinación final se hace por voto

no ponderado y ponderado respectivamente. El método MLE (*Mixtures of Local Experts*) [Jacobs *et al.*, 1991] entrena un clasificador (a la vez que el resto de elementos del conjunto) que selecciona a uno de los clasificadores para tomar la decisión final y hace que los clasificadores base tiendan a especializarse en distintas subtareas del problema (*local experts*). El método *stacking* [Wolpert, 1990] entrena un clasificador que aprende a combinar las salidas de los distintos clasificadores base; este enfoque también se adopta en [Todorovski y Džeroski, 2003] donde se construye un meta-árbol para combinar los elementos del conjunto. Otra variante consiste en generar un clasificador "árbitro" por cada elemento que dependiendo del ejemplo a clasificar dé un valor de confianza del clasificador base [Ortega *et al.*, 2001]. También es habitual el uso de distintas funciones de combinación de los clasificadores base. En [Kittler *et al.*, 1998] y [Kuncheva *et al.*, 2001] se muestran dos comparativas muy completas del uso de distintas funciones de combinación.

- **En cascada** (*cascading*): Los clasificadores del conjunto se invocan secuencialmente hasta que el patrón es clasificado [Gama y Brazdil, 2000; Pudil *et al.*, 1992]. Generalmente, los clasificadores base son incompatibles entre sí en el sentido de que se entrenan sobre conjuntos de datos con distintos atributos como, por ejemplo, las salidas de los clasificadores precedentes o con menos clases en cada paso. Por eficiencia estos conjuntos tienden a colocar los clasificadores rápidos y menos precisos al inicio seguidos de clasificadores más complejos y precisos.

- **Jerárquica**: Los clasificadores se organizan en una estructura de tipo árbol que determina el clasificador a invocar dependiendo del patrón a clasificar [Jordan y Jacobs, 1994]. Sólo se invoca por tanto un clasificador. Esta es una arquitectura muy flexible que utiliza clasificadores especializados en distintas regiones del espacio de atributos.

En el presente trabajo nos hemos centrado en arquitecturas paralelas con combinación final mediante voto. Esto es, considerando que se han generado una serie de $T$ clasificadores $h_1, h_2, \ldots, h_T$ su combinación mediante voto para obtener la clasificación final se puede formular como

$$H(\boldsymbol{x}) = \underset{j}{argmax} \sum_{t=1}^{T} w_t I(h_t(\boldsymbol{x}) = j) \ , \tag{2.19}$$

donde $I(.)$ es la función indicador que devuelve 1 si el argumento es verdadero y 0 en caso contrario y donde $w_t$ es un peso estático (no dependiente del vector de atributos $\boldsymbol{x}$) asignado al clasificador $h_t$.

### 2.3.1. Algoritmos propuestos

#### Nuevos métodos de generación de conjuntos de clasificadores

Dentro del presente trabajo se presentan cuatro nuevos métodos de construcción de conjuntos de clasificadores. Los tres primeros están basados en la variabilidad intrínseca de un algoritmo de construcción de árboles de decisión. Este algoritmo de construcción de árboles se denomina Algoritmo de Crecimiento y Poda Iterativos (*Iterative Growing and Pruning Algorithm*) y fue desarrollado por Gelfand *et al.* [Gelfand *et al.*, 1991]. El Algoritmo de Crecimiento y Poda Iterativos genera árboles de decisión —a los que haremos referencia a lo largo de la tesis como árboles IGP— dividiendo los datos en dos subconjuntos disjuntos de aproximadamente mismo tamaño y similar distribución de clases. IGP es un algoritmo iterativo que utiliza un subconjunto para hacer crecer el árbol y otro para podarlo alternando los papeles de los subconjuntos en cada iteración. El algoritmo tiene la propiedad de que diferentes divisiones de los datos generan árboles distintos, a pesar de haber sido construidos con el mismo conjunto de datos de entrenamiento.

El primero de los métodos propuestos, al que denominaremos conjunto IGP, aprovecha la variabilidad del algoritmo IGP para construir un conjunto de clasificadores a partir de distintas divisiones aleatorias de los datos. De este modo todos los clasificadores del conjunto se construyen usando todos los datos. Este método no se puede incluir en ninguna de las técnicas tradicionales para generar conjuntos de clasificadores que modifican, o bien los datos creando una visión parcial del problema, o bien un algoritmo de clasificación para generar variabilidad. El diseño del algoritmo de construcción de los clasificadores incorpora el mecanismo que asegura la variabilidad de los árboles generados sin necesidad de hacer remuestreos de los datos. Este hecho, combinado con la precisión equivalente de los clasificadores IGP con respecto a CART [Gelfand *et al.*, 1991] debería conducir a una mayor precisión del conjunto de clasificadores.

El segundo método es un conjunto de clasificadores de tipo *boosting* modificado para utilizarlo con árboles IGP. Esta técnica se incluye dentro de las técnicas de muestreo de los datos de entrenamiento.

El tercer método que hemos desarrollado une los dos métodos anteriores sustituyendo en el primero de ellos los árboles IGP por conjuntos de clasificadores de *boosting* con árboles IGP. Es decir, se trata de un conjunto de conjuntos de clasificadores.

Por último e independientemente de los tres métodos anteriores, se ha desarrollado un método que intercambia las clases del conjunto de entrenamiento aleatoriamente para construir cada clasificador base. Las modificaciones de las clases de los ejemplos se hacen de forma que todos los clasificadores construidos presenten el mismo error de clasificación en el conjunto de entrenamiento, pero en ejemplos distintos, aumentando así la variabilidad entre clasificadores. Este método pertenece al grupo de algoritmos que manipula las etiquetas de clase (sec. 2.3).

**Poda de conjuntos de clasificadores**

La segunda parte de este trabajo se centra en el análisis de los clasificadores del conjunto una vez generados para determinar cuáles son necesarios y reducir (podar) el número final de clasificadores del conjunto. Para ello se ha desarrollado una serie de procedimientos heurísticos que ordenan los clasificadores dentro del conjunto para posteriormente quedarse con los $\tau$ primeros. En este trabajo de tesis se han aplicado estos procedimientos a conjuntos generados con *bagging* obteniendo no sólo conjuntos de clasificadores más pequeños sino que además son más precisos que el conjunto completo. Otra ventaja adicional de los conjuntos podados es que clasifican más rápidamente los ejemplos y ocupan menos espacio en memoria.

## 2.4. Análisis del funcionamiento de conjuntos de clasificadores

Existen tres razones fundamentales que explican los generalmente buenos resultados que se obtienen utilizando conjuntos de clasificadores [Dietterich, 1998b; 2000a]. Estas razones son estadísticas, computacionales y de capacidad expresiva. Las razones estadísticas aplican a problemas de clasificación donde no se dispone de datos de entrenamiento suficientes para que el algoritmo de clasificación obtenga la mejor hipótesis. Los motivos computacionales se dan cuando, a pesar de disponer de datos suficientes, el algoritmo de clasificación no es capaz de llegar a la solución óptima; como puede ser el caso de una red neuronal que queda atrapada en un mínimo local. Por último, las causas expresivas aparecen cuando la solución del problema no está contenida en el espacio "efectivo" de hipótesis del algoritmo; donde el espacio "efectivo" de búsqueda del algoritmo viene limitado tanto por su capacidad expresiva real como por el hecho de que se dispone de un número finito de ejemplos de entrenamiento. En cualquiera de los tres casos, y mediante las técnicas de generación de conjuntos de clasificadores expuestas, se pueden generar clasificadores, cuya capacidad expresiva es limitada en el problema, que compensan sus limitaciones al ser combinados en un conjunto de clasificadores.

En la figura 2.3 se muestra gráficamente un problema de clasificación correspondiente a etiquetar ejemplos pertenecientes a dos clases separadas por una parábola mediante árboles de decisión que dividen el espacio con hiperplanos perpendiculares a los ejes. Se puede observar cómo la solución de este problema de clasificación no está contenida en el espacio de hipótesis de los árboles de decisión utilizados. En el gráfico de la izquierda se muestran tres soluciones propuestas por distintos árboles CART al problema. En el gráfico de la derecha se puede ver cómo la combinación de estas tres soluciones aproxima mucho mejor la frontera de decisión real. En este caso la solución combinada sigue estando en el espacio de hipótesis de los árboles de decisión, lo que no ocurriría con otros clasificadores base como

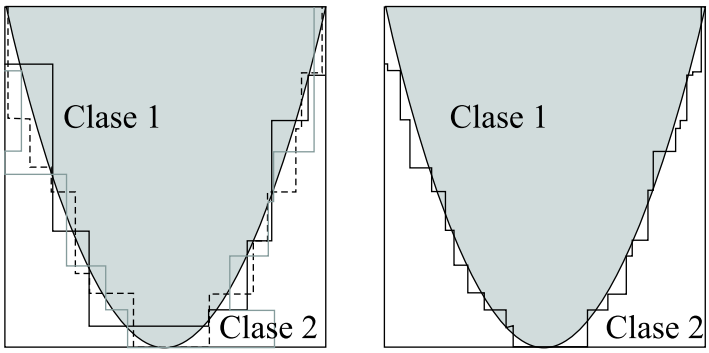



Figura 2.3: En el gráfico de la izquierda muestra tres aproximaciones en escalera a una división en parábola entre dos clases realizadas mediante *boosting*. El gráfico de la derecha muestra la combinación de las tres soluciones. Generado con *boosting*, errores de los árboles individuales con los datos de test=4.9 % 7.1 % y 6.7 % error conjunto 2.8 %

por ejemplo discriminantes lineales. Dado que, la solución combinada sigue estando en el espacio de hipótesis de los árboles de decisión, entonces ¿Por qué no intentar construir un algoritmo capaz de obtener directamente esta solución combinada sin tener que pasar por la generación varias hipótesis? Quinlan en su artículo [Quinlan, 1998] abordó este problema de manera inversa. Para ello, creó un árbol de decisión a partir de tres árboles obtenidos mediante *boosting*. Para obtener el árbol combinado colgó de las hojas del primer árbol el segundo árbol y de todas las hojas de todos los segundos árboles (colgados del primer árbol) colgó el tercer árbol. Finalmente asignó las clases a las hojas de árbol resultante teniendo en cuenta por qué hojas de los árboles 1, 2 y 3 había que pasar. Este árbol de decisión combinado es equivalente, en el espacio de hipótesis, al voto de los tres árboles por separado. Quinlan observó que el árbol combinado tenía muchos nodos hoja a los que la jerarquía de tests no asignaba ningún dato de los utilizados para construir los tres árboles. Sin embargo, si se podaban esos nodos, el error del árbol aumentaba hasta anular los beneficios obtenidos mediante la combinación. Esto nos lleva a pensar que es más sencillo generar varios clasificadores y combinarlos para que compensen sus errores que generar un clasificador único que, como en el caso de árboles de decisión, debería generar nodos que no contuvieran ningún dato de entrenamiento.

Parece por tanto que la combinación de clasificadores mediante voto hace que los errores de éstos se compensen. Generalmente se considera que en un problema de clasificación binario el error del conjunto tiende a 0 a medida que crece el número de clasificadores siempre que se cumplan las siguientes condiciones: (i) que los errores de los clasificadores individuales estén por debajo de $0.5$ y (ii) que los errores de los clasificadores no estén correlacionados [Dietterich, 1998b].

La primera condición no es estrictamente necesaria. En realidad, la condición necesaria para alcanzar error 0 es que la probabilidad dentro del espacio de posibles hipótesis de clasificar cada ejemplo sea menor de $0.5$ [Esposito y Saitta, 2003; 2004]. Ambas cantidades están relacionadas: ningún ejemplo tendrá probabilidad mayor de $0.5$ de ser clasificado correctamente si todas las hipótesis tienen un error mayor de $0.5$. Lo que nos dice la observación realizada por Saitta y Esposito es que puede existir un pequeño número de clasificadores dentro del conjunto con error mayor de $0.5$ pero que contribuyan positivamente a la reducción del error del conjunto. Consideremos por ejemplo la decisión proporcionada por cinco clasificadores distintos para tres ejemplos en un problema de dos clases: $\{0, 1, 1\}$, $\{0, 1, 1\}$, $\{1, 1, 0\}$, $\{1, 0, 1\}$ y $\{1, 0, 0\}$, donde $1$ indica clasificación correcta y $0$ incorrecta. Se puede ver cómo los cuatro primeros clasificadores presentan un error de $33\,\%$ y que al combinarlos clasifican bien el segundo y tercer ejemplo pero no el primero, en el que se produce un empate. Al añadir el quinto clasificador con error $66\,\% > 50\,\%$ el empate se deshace y el conjunto pasa a clasificar correctamente los tres ejemplos.

La segunda condición (que los errores de los clasificadores no estén correlacionados) es intuitivamente necesaria: si los clasificadores base son todos iguales, ¿qué sentido tiene hacer un conjunto? Sin embargo, no se ha encontrado ninguna medida de diversidad con validez general que correlacione el error de generalización con la diversidad entre clasificadores de forma clara. Una forma de ver gráficamente la relación entre diversidad y precisión de los clasificadores son los diagramas de kappa-error [Margineantu y Dietterich, 1997]. En estos diagramas, para cada par de clasificadores del conjunto se calcula un punto donde la $y$ es el error medio cometido por los clasificadores y la $x$ es su diversidad medida con el estadístico kappa. Este estadístico mide el acuerdo entre dos clasificadores de forma que: $\kappa = 0$ indica que el acuerdo entre los dos clasificadores es igual al esperado en el caso de que la coincidencia sea aleatoria; $\kappa = 1$ indica que los clasificadores clasifican igual todos los ejemplos y $\kappa < 0$ indica un acuerdo menor que el aleatorio. En la figura 2.4 se pueden ver dos diagramas de kappa-error para *bagging* (izquierda) y *boosting* (derecha) de tamaño 100 y entrenados sobre el conjunto *Twonorm*. Los puntos arriba a la izquierda en estos diagramas indican pares de clasificadores muy distintos entre sí y con un alto error de clasificación mientras que los puntos abajo a la derecha representan pares de clasificadores similares y con un error bajo. Se puede observar como la nube de puntos generada por *boosting* es mucho más extensa que la que genera *bagging*. Esta mayor capacidad para generar una diversidad de clasificadores podría explicar el mejor funcionamiento de *boosting* sobre *bagging* en conjuntos no ruidosos.

En el estudio realizado por [Kuncheva y Whitaker, 2003] se analizan 10 medidas de diversidad de conjuntos de clasificadores para analizar su correlación con el error del conjunto (4 de ellas basadas en promedios sobre pares de clasificadores y 6 globales del conjunto). Sus experimentos, sin embargo, fueron desalentadores y mostraron la incapacidad de estas medidas para predecir las variaciones de error del conjunto.

La mejora que obtienen los algoritmos de conjuntos de clasificadores también se ha

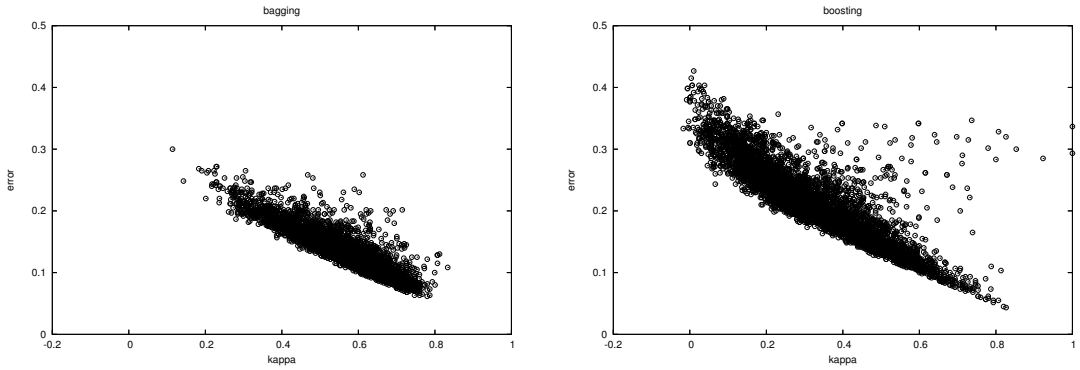



Figura 2.4: Diagramas de kappa-error para *bagging* (*izquierda*) y *boosting* (*derecha*) entrenados en el conjunto *Twonorm*

intentado formalizar teóricamente al menos desde otros dos puntos de vista. Por una parte se ha estudiado desde el punto de vista de dividir el error entre el sesgo (*bias*) y la varianza del algoritmo (*variance*). El origen de esta descomposición es el análisis de ajuste funcional mediante regresión donde la división entre sesgo y varianza son cantidades positivas bien definidas. La media de varias regresiones nunca incrementa el error esperado y reduce el término de varianza sin modificar el error de sesgo. Para clasificación la división entre estos dos términos no está tan bien definida. De hecho se han propuesto varias definiciones [Kong y Dietterich, 1995; Kohavi y Wolpert, 1996; Breiman, 1996b; Friedman, 1997] pero ninguna parece tener todas las propiedades deseables. Por otra parte, la mejora que consiguen los conjuntos de clasificación se ha analizado estudiando la distribución de los márgenes de los datos de entrenamiento, donde el margen de un ejemplo de entrenamiento es la diferencia entre los votos recibidos por la clase correcta y los votos recibidos por la clase incorrecta más votada [Schapire *et al.*, 1998].

### 2.4.1. Sesgo y varianza

Según el punto de vista del sesgo y la varianza, el error que comete un algoritmo de clasificación se puede dividir en: error de Bayes, error debido al sesgo del algoritmo de clasificación y error debido a la varianza del algoritmo, esto es

$$Error = Error\ de\ Bayes + sesgo + varianza\ . \tag{2.20}$$

Analicemos esta descomposición del error en detalle. Por una parte, el error de Bayes (ec. (2.6)) es un error inherente al problema de clasificación y por tanto irreducible. Viene dado por el solapamiento de las distribuciones de las clases en el espacio de atributos. En las zonas de solapamiento, donde dos o más clases pueden existir, es imposible el determinar

con seguridad la clase de cada nuevo ejemplo. Para reducir el error al mínimo en las zonas de solapamiento hay que clasificar cada punto de acuerdo a la distribución más probable: esto define el error mínimo de Bayes (ec. (2.5)). Sin embargo, para conjuntos de datos reales, en los que desconocemos las distribuciones de los datos y donde disponemos de un número limitado de ejemplos, el cálculo de este límite inferior puede no ser posible. No sucede así en conjuntos de datos generados artificialmente para los que disponemos de las reglas que generan los datos y por tanto el error de Bayes se puede calcular o estimar con precisión. La dificultad que existe en determinar el error de Bayes hace que muchas de las definiciones de sesgo y varianza engloben de alguna manera el error de Bayes dejando la definición del error como sigue

$$Error = sesgo + varianza\ . \tag{2.21}$$

Los otros dos miembros de la ecuación (2.20), sesgo y varianza, son la parte del error causada por el algoritmo de clasificación. El sesgo indica la parte del error debida a la tendencia central del algoritmo errónea, mientras que la varianza determina la parte del error debida a desviaciones en relación a la tendencia central del algoritmo. Se define la tendencia central de un algoritmo de clasificación para un vector $\boldsymbol{x}$ como la clase con mayor probabilidad de selección por los clasificadores construidos a partir de la distribución de posibles conjuntos de entrenamiento.

La medida de estas dos cantidades es útil para analizar la naturaleza del error de un algoritmo. Por una parte las diferencias en las predicciones que hace un algoritmo cuando es entrenado con distintos conjuntos de entrenamiento, dado que sólo hay una clase correcta, limita el error mínimo que podemos alcanzar (varianza). Por otra parte el diseñar algoritmos que presenten pocas variaciones para distintos conjuntos de entrenamiento no es garantía de una disminución del error, ya que puede ser que el algoritmo sea también estable en el error. Es decir, que tenga una tendencia central errónea (sesgo).

Para muchos algoritmos de conjuntos de clasificadores se han efectuado medidas del sesgo y la varianza, y se han comparado con el sesgo y varianza del algoritmo base [Bauer y Kohavi, 1999; Breiman, 1996b; Webb, 2000; Breiman, 2000]. De esta forma se puede explicar el origen de la disminución del error con respecto al algoritmo base. Generalmente, los algoritmos de conjuntos de clasificadores tienden a disminuir el error de varianza, ya que el proceso de votación hace que éstos sean más estables en sus decisiones que los clasificadores individuales. Además, los conjuntos de clasificadores que usan procesos adaptativos para generarse (ej. *boosting*) también pueden reducir el sesgo, ya que el proceso adaptativo hace que no cometan siempre los mismos errores (realmente también podrían aumentar el sesgo cuando los conjuntos de entrenamiento tienen datos etiquetados incorrectamente).

El hecho de que sesgo y varianza no estén bien definidos para problemas de clasificación ha llevado a la aparición de múltiples definiciones como se puede ver en [Webb, 2000]. De entre ellas aquí mostramos la definición de Breiman [Breiman, 1996b] por ser

sencilla e intuitiva. Definamos la notación brevemente. Sea $\Gamma$ un algoritmo de clasificación. Sea $\mathcal{L}$ la distribución de posibles conjuntos de entrenamiento: $\Gamma(\mathcal{L})$ es la distribución de clasificadores generados tras aplicar el algoritmo $\Gamma$ a la distribución $\mathcal{L}$. Además dada la distribución del problema $(X, Y)$, $\Gamma(\mathcal{L})(X)$ devuelve la distribución de clases obtenida por el algoritmo $\Gamma$ con la distribución de conjuntos de entrenamiento $\mathcal{L}$. A continuación se muestran las definiciones probabilísticas de Breiman utilizando el error de Bayes y sin utilizarlo (definiciones (2.20) y (2.21) respectivamente)

$$\begin{aligned} sesgo_B &= P_{(Y,X),\mathcal{L}}((\Gamma(\mathcal{L})(X) \neq Y) \wedge (\Gamma(\mathcal{L})(X) \neq C_{Y,X}^{Bayes}) \wedge (\Gamma(\mathcal{L})(X) = C_{\Gamma,\mathcal{L}}^{o}(X))) \\ var_B &= P_{(Y,X),\mathcal{L}}((\Gamma(\mathcal{L})(X) \neq Y) \wedge (\Gamma(\mathcal{L})(X) \neq C_{Y,X}^{Bayes}) \wedge (\Gamma(\mathcal{L})(X) \neq C_{\Gamma,\mathcal{L}}^{o}(X))) \\ sesgo &= P_{(Y,X),\mathcal{L}}((\Gamma(\mathcal{L})(X) \neq Y) \wedge (\Gamma(\mathcal{L})(X) = C_{\Gamma,\mathcal{L}}^{o}(X))) \\ var &= P_{(Y,X),\mathcal{L}}((\Gamma(\mathcal{L})(X) \neq Y) \wedge (\Gamma(\mathcal{L})(X) \neq C_{\Gamma,\mathcal{L}}^{o}(X))) \ , \end{aligned}$$

donde $C_{X,Y}^{Bayes}$ es el clasificador de Bayes para la distribución del problema $(X, Y)$ y $C_{\Gamma,\mathcal{L}}^{o}$ es la tendencia central del algoritmo $\Gamma$ para la distribución de conjuntos de datos de entrenamiento $\mathcal{L}$. La primera de estas definiciones indica que el sesgo para un algoritmo $\Gamma$, una distribución del problema $(X, Y)$ y una distribución de conjuntos de entrenamiento extraída de $(X, Y)$, $\mathcal{L}$, es igual a la probabilidad $P_{(Y,X),\mathcal{L}}$ de que el algoritmo se equivoque $(\Gamma(\mathcal{L})(X) \neq Y)$ y que su predicción coincida con la de la tendencia central del algoritmo $(\Gamma(\mathcal{L})(X) = C_{\Gamma,\mathcal{L}}^{o}(X))$, siempre que este error no lo cometa también el clasificador de Bayes $(\Gamma(\mathcal{L})(X) \neq C_{X,Y}^{Bayes}(X))$.

### 2.4.2. Márgenes

Otro procedimiento para explicar la mejora que se obtiene con los conjuntos de clasificadores se describe en [Schapire *et al.*, 1998]. Según este análisis, la mejora de los conjuntos de clasificación está relacionada con la distribución de los márgenes de clasificación de los ejemplos de entrenamiento. El margen de clasificación de un ejemplo de entrenamiento para un conjunto de clasificadores es la diferencia de votos que ha recibido la clase correcta del ejemplo y el número de votos recibidos por la clase incorrecta más votada. De acuerdo con esta definición, si el margen de un ejemplo es positivo, el ejemplo estará bien clasificado. Si el margen es negativo, esto significa que una clase incorrecta tiene más votos que la clase correcta y, por tanto, que el conjunto de clasificadores lo clasificará mal.

Con el fin de estudiar el margen de forma general para conjuntos con cualquier número de clasificadores Schapire *et al.* proponen una definición en la que se dividen los votos de los clasificadores por el número de clasificadores del conjunto haciendo que la suma de todos los votos sea 1. Con esta definición el margen de clasificación normalizado (a partir de este momento simplemente "margen") de cada ejemplo queda definido en el intervalo

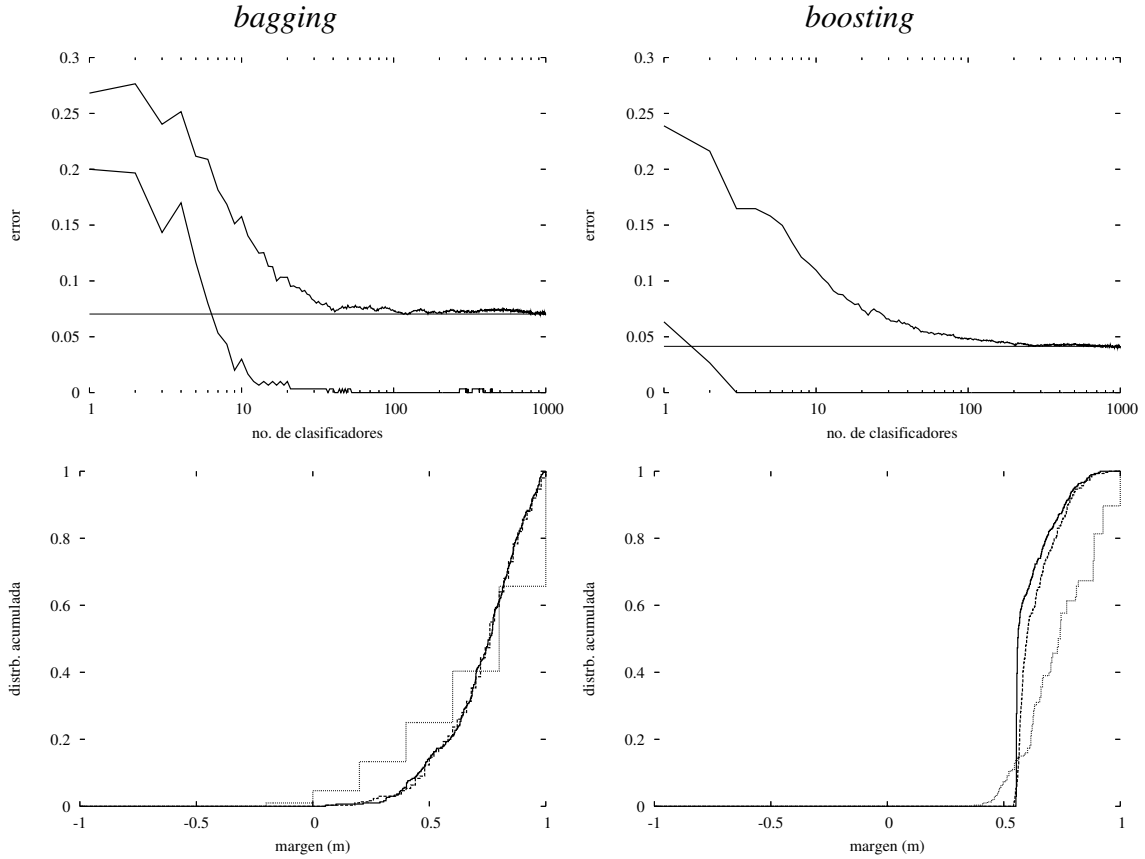



Figura 2.5: Curvas de error y gráficos de distribuciones de márgenes para *bagging* y *boosting* con CART como algoritmo base y para el conjunto de datos *Twonorm* (más detalles en el texto)

[-1, 1], independientemente del número de clasificadores del conjunto. Si todos los clasificadores han votado la clase correcta el margen es 1 y es -1 si todos los clasificadores han votado a una misma clase incorrecta.

El análisis basado en el margen puede explicar resultados como los obtenidos en el problema *Twonorm* mostrado en la figura 2.5. En las gráficas superiores de la figura se muestra el error de clasificación que obtienen *bagging* y *boosting* en función del número de clasificadores utilizando el algoritmo de construcción de árboles CART como algoritmo base. Cada uno de estos gráficos de error contiene dos curvas: la superior corresponde al error para los datos de test y la inferior corresponde al error para los datos de entrenamiento. Adicionalmente, se muestra con una línea horizontal el error final del conjunto con 1000 clasificadores.

En estos gráficos se puede observar cómo el error del conjunto de clasificadores generados con *boosting* disminuye a 0 para el conjunto de datos de entrenamiento tras generar 3 clasificadores. Se podría pensar que no podemos obtener más información de unos datos que clasificamos perfectamente. Sin embargo, se puede ver cómo el error en los datos de test sigue disminuyendo durante muchas más iteraciones. Schapire *et al.* en su artículo [Schapire *et al.*, 1998] explican este hecho utilizando la definición de margen. Consideran que se puede seguir aprendiendo a pesar de haber alcanzado error cero para los datos de entrenamiento ya que los márgenes de estos datos siguen aumentando. Al incrementar el número de clasificadores clasificamos "mejor" los datos de entrenamiento en el sentido de que aumentamos las diferencias entre la clase correcta y la segunda clase más votada. Esto se puede ver en los gráficos inferiores de la figura 2.5. En estos gráficos se representa para *bagging* y *boosting* la distribución acumulada de márgenes para todo el conjunto de datos de entrenamiento. Las curvas representan en función del margen $m$ la fracción de datos de entrenamiento cuyo margen es inferior o igual a $m$ con $m \in [-1, 1]$. Cada uno de estos gráficos muestra tres curvas con la distribución del margen para: 10 (línea de puntos), 100 (línea de trazos, parcialmente oculta) y 1000 (línea continua) clasificadores. Para *boosting* se puede ver cómo el margen cambia sustancialmente cuando se pasa de 10 a 100 clasificadores a pesar de que el error en los datos de entrenamiento sigue siendo cero. Podemos ver cómo en *boosting* para 100 clasificadores todos los ejemplos de entrenamiento tienen un margen superior a $0.5$. Es decir, todos los datos están clasificados con una mayor seguridad. También podemos observar cómo *bagging* también aumenta los márgenes de los ejemplos de entrenamiento que están más cerca de $0$, pero lo hace de forma más suave. Tras añadir 1000 clasificadores vemos que con *bagging* los ejemplos de entrenamiento tienen márgenes prácticamente en todo el rango $[0, 1]$ mientras que *boosting* en el rango $[0.55, 0.9]$. Es decir, *boosting* sacrifica clasificar perfectamente algunos ejemplos para reducir el margen de los ejemplos más difíciles aumentando así el margen mínimo en el conjunto de entrenamiento, donde el margen mínimo viene determinado por el ejemplo con menor margen.

La explicación basada en margen es cuanto menos incompleta. Casi siempre es posible obtener un árbol de decisión con error cero en el conjunto de entrenamiento. Si lo copiamos $K$ veces para dar lugar a un conjunto de clasificadores tendremos un conjunto que clasifica todos los datos de entrenamiento con margen 1. Sin embargo, este conjunto es probable que no alcance las capacidades de generalización de *bagging* o *boosting*. Usando programación lineal [Grove y Schuurmans, 1998] mostraron experimentalmente que aumentar el margen mínimo no sólo no disminuye el error de generalización sino que muy frecuentemente lo aumenta. En [Mason *et al.*, 2000] observaron que el margen mínimo no es un factor crítico para determinar el error de generalización. Otro contraejemplo, es el conjunto de clasificadores que se presenta en el capítulo 4 de este trabajo que tiende a ser más efectivo cuando el margen mínimo y medio en entrenamiento es menor.

## 2.5. *Bagging* y bosques aleatorios

Una de las técnicas más eficaces para la construcción de conjuntos de clasificadores, desarrollada por Breiman [Breiman, 1996a], es *bagging* (*Bootstrap sampling and aggregation*). Esta técnica se incluye dentro del grupo que muestrean los datos de entrenamiento para obtener cada uno de los clasificadores base (sec. 2.3). En la figura 2.6 se muestra el pseudocódigo de *bagging*. Cada clasificador base se genera a partir de un conjunto de datos obtenido por muestreo aleatorio con reemplazo del conjunto de datos de entrenamiento y con el mismo número de ejemplos que éste. Este algoritmo está basado en la técnica estadística *bootstrap*, que sirve para la estimación de cantidades estadísticas a partir de muestras obtenidas con repetición de la muestra original aleatoriamente [Efron y Tibshirani, 1994]. En *bagging* cada clasificador se construye con un subconjunto de los datos originales en el que con alta probabilidad hay ejemplos repetidos. Para estimar cuántos de estos ejemplos distintos tienen, en media, cada una de las muestras generadas vamos a calcular la probabilidad de que un ejemplo aparezca en la muestra. Esta probabilidad es igual a 1 menos la probabilidad de que no aparezca

$$P = 1 - \left(\frac{N-1}{N}\right)^N ,$$

donde $N$ es el número de ejemplos del conjunto de entrenamiento y $(N-1)/N$ es la probabilidad de que un elemento no sea elegido en una tirada y está elevado a $N$, que es el número de extracciones que se realizan. Esta probabilidad tiende a $1 - 1/e$ cuando $N$ tiende a infinito

$$\lim_{N\to\infty} \left(\frac{N-1}{N}\right)^N = \lim_{N\to\infty} \left(1 - \frac{1}{N}\right)^N = e^{-1} = 0.3679 .$$

Por tanto si cada uno de los ejemplos tiene una probabilidad $1 - 1/e$ de aparecer en una muestra entonces se tiene que, en media, cada muestra contiene un $63.2\,\%$ de los datos originales y el resto son ejemplos repetidos. Por tanto, en *bagging*, cada clasificador individual se genera con un número de ejemplos menor que el número inicial de ejemplos de entrenamiento. Esto hace que los clasificadores individuales utilizados en *bagging* normalmente tengan un error de generalización peor que el del clasificador construido con todos los datos. Sin embargo, al combinar la decisión de estos clasificadores se compensan en parte sus errores lo que habitualmente se traduce en mejoras en la capacidad de generalización respecto a la de un sólo clasificador construido con todos los datos.

La combinación de *bagging* con árboles de decisión como clasificadores base entra dentro de la definición de bosques aleatorios (*random forests*), donde el vector $\mathbf{\Theta}$ contiene $N$ números enteros aleatorios generados entre $1$ y $N$ para hacer el muestreo *bootstrap*.

**Entradas:**

Conjunto de entrenamiento $L$ de tamaño $N$

Número de clasificadores $T$

**Salida:**

$H(\boldsymbol{x}) = \underset{y}{argmax} \sum_{t=1}^{T} I(h_t(\boldsymbol{x}) = y)$

1. for $t = 1$ to $T$ $\{$
2. $L_{bs}$ = MuestreoBootstrap($L$)
3. $h_t$ = ConstruyeClasificador($L_{bs}$)
4. $\}$

Figura 2.6: Pseudocódigo de *bagging*

### 2.5.1. Consideraciones sobre *bagging*

Los conjuntos de clasificadores construidos mediante *bagging* presentan un error de generalización menor que el de los algoritmos base en la mayoría de los conjuntos de datos en los que se ha probado en la literatura [Breiman, 1996a; Quinlan, 1996a; Bauer y Kohavi, 1999; Dietterich, 2000b; Webb, 2000]. Además es un algoritmo robusto frente a ruido o fallos en las etiquetas de clase de los ejemplos [Quinlan, 1996a; Dietterich, 2000b].

La reducción del error con respecto al algoritmo base utilizado se debe a la reducción en varianza [Bauer y Kohavi, 1999; Webb, 2000]. Según la interpretación habitual, la eficacia de *bagging* en reducir el error es mayor cuando los clasificadores individuales tienen errores de sesgo pequeños y a la vez presentan errores de varianza grandes [Breiman, 1998; Bauer y Kohavi, 1999]. *Bagging* no reduce la parte de error debida al sesgo del algoritmo base. El error de sesgo es debido a que la tendencia central del algoritmo es errónea. Parece lógico que *bagging* no reduzca el sesgo ya que el conjunto mantiene la tendencia central del algoritmo base: los cambios de muestreo en *bagging* no son lo suficientemente grandes como para que el algoritmo base cambie su sesgo. De igual manera, dado que *bagging* estabiliza mediante voto la tendencia central del algoritmo base, se obtiene mejora en la reducción de varianza, ya que la combinación de clasificadores hace que la clasificación sea más estable. Estas últimas observaciones son correctas siempre que el mecanismo de *bootstrap* utilizado para obtener distintas muestras funcione lo suficientemente bien como para que las muestras generadas se aproximen suficientemente a muestras independientes. Según Schapire *et al.* [Schapire *et al.*, 1998] el procedimiento de *bootstrap* de *bagging* puede fallar en obtener muestreos aproximadamente independientes cuando se tienen distribuciones de datos muy simples. Los conjuntos de datos más utilizados (colección de problemas UCI [Blake y Merz, 1998]) en los que se ha probado *bagging* no deben presentar este problema dado que los resultados obtenidos son, en general, buenos, y casi nunca

aumentan el error del clasificador base.

En otro estudio [Grandvalet, 2004] (aplicado a regresión solamente) se muestra cómo *bagging* realiza una nivelación (*equalization*) de la influencia de los ejemplos reduciendo la importancia de los puntos de palanca (*leverage points*), aquéllos que tienen gran influencia en los regresores. Esto explica por qué *bagging* es más robusto frente a puntos anómalos (*outliers*) en contraste con otros algoritmos. Sin embargo en este estudio muestran cómo *bagging* puede ser perjudicial cuando los puntos de palanca no son anómalos sino beneficiosos para la estimación.

Como ya hemos mencionado, *bagging* descarta en media un $36.8\,\%$ de los datos de entrenamiento para construir cada clasificador base. A este conjunto se le denomina conjunto *out-of-bag*. Los conjuntos *out-of-bag* pueden ser utilizados para hacer buenas estimaciones del error de generalización del conjunto [Breiman, 1996c]. La estimación *out-of-bag* del error de generalización consiste en utilizar para cada ejemplo sólo las predicciones de los clasificadores que no han visto ese ejemplo. De esta forma el error sobre cada ejemplo se calcula agregando sólo las predicciones de estos clasificadores. Para calcular la estimación del error de generalización del conjunto se promedian estos errores sobre todos los datos de entrenamiento. Este método tiene la ventaja de ser muy eficiente computacionalmente con respecto a otros métodos utilizados para calcular el error de generalización, como validación cruzada, que deben generar clasificadores adicionales. Otro método eficiente para calcular el error de generalización aplicado a conjuntos *bagging* para regresión de describe en [Wolpert y Macready, 1999].

*Double-bagging* es una variante de *bagging* que aprovecha el conjunto *out-of-bag* de cada muestreo *bootstrap* para construir un discriminante lineal [Hothorn y Lausen, 2003]. Posteriormente, construye a partir de la muestra *bootstrap* el clasificador base usando los atributos originales del problema junto con las variables obtenidas por el discriminante lineal que ha usado el conjunto *out-of-bag*. El conjunto de clasificadores resultante obtiene resultados equivalentes a un discriminante lineal cuando las clases son separables linealmente y equivalentes a *bagging* en caso contrario.

Es interesante hacer notar que en *bagging* el número total de veces que ha aparecido cada ejemplo en entrenamiento sumado sobre todos los muestreos *bootstrap* no es constante, aunque tiende a equilibrarse al aumentar el número de clasificadores. Sin embargo, en una ejecución típica de *bagging* con 100 clasificadores no es difícil que haya ejemplos que aparezcan el doble de veces que otros [Christensen *et al.*, 2003]. En esta referencia se presenta una variante de *bagging* que consiste en forzar a que el número de veces que aparece cada ejemplo en el proceso total de construcción del conjunto sea constante.

En cuanto al estudio del margen, en [Schapire *et al.*, 1998] se muestra que *bagging* aumenta el margen cuando se incrementa el número de clasificadores. Sin embargo, este aumento ocurre lentamente, o al menos más lentamente que en *boosting*. Esto parece lógico ya que *bagging* es un algoritmo "neutro" con los ejemplos, es decir, construye clasificadores sin tener en cuenta ninguna información ni de los clasificadores previamente construidos

ni de los ejemplos de entrenamiento utilizados para construir cada clasificador. Esto hace que *bagging* se pueda implementar fácilmente en paralelo ya que la construcción de cada clasificador base es completamente independiente del resto de clasificadores. Se puede, por tanto, generar cada clasificador base en un proceso distinto y combinarlos al final.

## 2.6. *Boosting*

Otra de las técnicas más difundidas y eficaces para la construcción de conjuntos de clasificadores es *Boosting* [Freund y Schapire, 1995]. *Boosting* es una técnica que convierte cualquier aprendiz débil en uno fuerte [Schapire, 1990] (donde por clasificador débil se entiende aquel clasificador que consigue un error un poco mejor que predicción aleatoria mientras que fuerte es aquel método que clasifica bien el concepto excepto por una pequeña fracción de ejemplos). *Boosting* aprovecha el comportamiento de los clasificadores base previamente construidos para generar los siguientes. Breiman designó en [Breiman, 1996b] este tipo de algoritmos adaptativos con el nombre de *arcing* (*adaptively resample and combine*). En *boosting* este proceso adaptativo se consigue asignando pesos a los ejemplos de entrenamiento y modificando dichos pesos de acuerdo con los resultados del último clasificador generado. La modificación de pesos se hace de forma que los ejemplos mal clasificados por un clasificador aumenten en importancia para construir el siguiente clasificador. *Boosting* es el primer algoritmo de *arcing* desarrollado y el más difundido, aunque no el único. Existen otros algoritmos de *arcing* como el algoritmo *arc-x4* desarrollado por Breiman [Breiman, 1998], que funciona también dando pesos a los ejemplos. En *arc-x4*, después de construir cada clasificador se modifican los pesos multiplicándolos por $1 + m(i)^4$, donde $m(i)$ es el número de veces que el ejemplo $i$ ha sido mal clasificado por todos los anteriores clasificadores, y normalizando los pesos posteriormente. De aquí en adelante denominaremos a los algoritmos adaptativos como algoritmos tipo *boosting* por ser éste el término utilizado para definir al primer algoritmo de este tipo.

Veamos ahora el funcionamiento de *AdaBoost.M1* [Freund y Schapire, 1995], uno de los primeros algoritmos de *boosting* desarrollados. El pseudocódigo de este algoritmo se muestra en la fig. 2.7. Dado un conjunto de datos de entrenamiento $i = 1, 2, ..., N$ y un conjunto de clasificadores a construir $t = 1, 2, ..., T$, se asocia un peso por dato de entrenamiento y clasificador, $w_t[i]$, inicializando los pesos iniciales según $w_1[i] = 1/N$ (línea 1). Es decir, al principio, todos los ejemplos tienen igual importancia para construir el primer clasificador individual. A continuación se realiza un bucle de $T$ iteraciones donde se construye cada clasificador individual. Dentro del bucle: se construye un clasificador base $h_t$ usando todos los datos de entrenamiento ponderados con pesos $w_t$ (línea 3); se calcula el error $\epsilon_t$ para el clasificador $h_t$ con respecto a los datos de entrenamiento $L$ como la suma de los pesos de los ejemplos mal clasificados (línea 4); Si $\epsilon_t \geq 0.5$ o $\epsilon_t = 0$ entonces el proceso termina, descartando el último clasificador si $\epsilon_t \geq 0.5$ y dándole el máximo peso si

**Entradas:**

Conjunto de entrenamiento $L$ de tamaño $N$
Número de clasificadores $T$

**Salida:**

$H(\boldsymbol{x}) = \underset{y}{argmax} \sum_{t=1}^{T} \log(1/\beta_t) I(h_t(\boldsymbol{x}) = y)$

1. asignar $\boldsymbol{w}_1[i] = 1/N,\ i = 1, \ldots, N$
2. for t=1 to T {
3. $h_t$ = ConstruyeClasificador($L$, $\boldsymbol{w}_t$)
4. $\epsilon_t$ = Error($h_t$, $L$, $\boldsymbol{w}_t$)
5. $\beta_t = \epsilon_t/(1 - \epsilon_t)$
6. if ($\epsilon_t \geq 0.5$ or $\epsilon_t = 0$) {
7. if ($\epsilon_t \geq 0.5$) desechar $h_t$
8. break
9. }
10. for j=1 to N {
11. if ($h_t(\boldsymbol{x}_j) \neq y_j$) then $w_{t+1}[j] = w_t[j]\beta_t$
12. else $w_{t+1}[j] = w_t[j]$
13. }
14. Normalizar($\boldsymbol{w}_{t+1}$)
15. }

Figura 2.7: Pseudocódigo de *AdaBoost.M1*

$\epsilon_t = 0$. Si no, se calculan los pesos $w_{t+1}$ para construir $h_{t+1}$ de forma que $h_t$ con los pesos $w_{t+1}$ tenga un error igual a $0.5$. Esto se consigue reduciendo los pesos de los ejemplos bien clasificados por un factor $\epsilon_t/(1 - \epsilon_t)$ y normalizando (líneas 10–14). Finalmente, la clasificación del conjunto se obtiene a través del voto ponderado de todos los clasificadores $h_t$ mediante $\log((1 - \epsilon_t)/\epsilon_t) = \log(1/\beta_t)$. Esto significa que clasificadores con menor error tienen más peso en el proceso de votación.

Como vemos en los líneas 6–9 de la figura 2.7, el algoritmo puede detenerse antes de construir todos los clasificadores previstos. Esto sucede cuando:

- Se alcanza un clasificador con error 0. Si un clasificador alcanza error cero significa que no comete ningún error en los datos de entrenamiento. Continuar, por tanto, el proceso de generación de clasificadores no tendría ningún sentido, ya que los pesos no se alterarían y se volvería a obtener el mismo clasificador una y otra vez (siempre que el algoritmo base sea determinista). Además, un clasificador con error cero tiene peso infinito en el proceso de votación y por tanto será el clasificador que determine

la decisión del conjunto.

- Otra posibilidad es que un clasificador $h_t$ tenga un error ponderado mayor de $0.5$. Este error es mayor que el error de una predicción aleatoria en conjuntos equilibrados de dos clases, lo que reduciría la eficacia del conjunto de clasificadores (aunque hemos visto que esto no siempre es así, sec. 2.4). Además, el que un clasificador tenga un error mayor de $0.5$ significa que el algoritmo base de clasificación no ha sido capaz de generar una división que resuelva el problema de clasificación con los nuevos pesos.

El incremento del peso de los ejemplos mal clasificados conduce a que en las sucesivas iteraciones el algoritmo base se centre en intentar clasificar correctamente estos ejemplos previamente mal clasificados. La variación de pesos $w_t$ es tal que el error del clasificador $h_t$ usando los pesos $w_{t+1}$ (calculados por modificación de $w_t$) es $0.5$. Comprobemos ahora esta afirmación. Primero ordenemos los ejemplos de forma que los primeros $N_{et}$ ejemplos sean los datos mal clasificados por el clasificador $h_t$ y el resto de datos $(N - N_{et})$ sean los bien clasificados. De esta forma podemos escribir el error $\epsilon_t$ del clasificador $h_t$ como

$$\epsilon_t = \sum_{i=1}^{N_{et}} w_t[i] = 1 - \sum_{i=N_{et}+1}^{N} w_t[i] \ . \tag{2.22}$$

Por otra parte los pesos $w_{t+1}$ deben de normalizarse después de dividir a los ejemplos mal clasificados por $(1 - \epsilon_t)/\epsilon_t$, esto es que su suma debe dar 1, por tanto

$$\begin{aligned} 1 = \sum_{i=1}^{N} w_{t+1}[i] = \sum_{i=1}^{N_{et}} w_{t+1}[i] + \sum_{i=N_{et}+1}^{N} w_{t+1}[i] = \\ = K \left( \sum_{i=1}^{N_{et}} w_t[i] \frac{1-\epsilon_t}{\epsilon_t} + \sum_{i=N_{et}+1}^{N} w_t[i] \right) = K \left( \epsilon_t \frac{1-\epsilon_t}{\epsilon_t} + (1-\epsilon_t) \right) = 2K(1-\epsilon_t) \ , \end{aligned}$$

donde se han sustituido las sumatorios por el error definido en la ec. (2.22) y donde $K$ es la constante de normalización que, despejando, se obtiene

$$K = \frac{1}{2(1-\epsilon_t)}$$

Finalmente podemos observar cómo la suma de los pesos $w_{t+1}$ de los ejemplos mal

clasificados por el clasificador $h_t$ es $1/2$:

$$\sum_{i=1}^{N_{et}} w_{t+1}[i] = K \left( \sum_{i=1}^{N_{et}} w_t[i] \frac{1-\epsilon_t}{\epsilon_t} \right) = \frac{1}{2(1-\epsilon_t)} \left( \sum_{i=1}^{N_{et}} w_t[i] \frac{1-\epsilon_t}{\epsilon_t} \right) =$$

$$= \frac{1}{2\epsilon_t} \sum_{i=1}^{N_{et}} w_t[i] = \frac{1}{2\epsilon_t} \epsilon_t = \frac{1}{2} \, .$$

De este desarrollo se puede ver cómo la modificación de pesos del tercer paso se puede hacer de forma equivalente y en un solo paso dividiendo los pesos de los ejemplos bien clasificados por $2(1-\epsilon_t)$ y por $2\epsilon_t$ los pesos de los mal clasificados, como mostraron Bauer y Kohavi [Bauer y Kohavi, 1999]. Esta derivación demuestra que no es necesario realizar la normalización de los pesos tras cada iteración.

### 2.6.1. Consideraciones sobre *boosting*

*Boosting* ha mostrado ser uno de los métodos más eficientes para la construcción de conjuntos de clasificadores [Quinlan, 1996a; Opitz y Maclin, 1999; Bauer y Kohavi, 1999; Dietterich, 2000b; Webb, 2000]. En numerosos problemas de clasificación, es uno de los que mejores resultados obtiene. Sin embargo, hay una serie de problemas de clasificación donde su rendimiento es inferior a *Bagging* [Quinlan, 1996a; Webb, 2000]. Donde mayores dificultades encuentra *boosting* es en conjuntos de datos con ruido, bien porque hay atributos cuyo valor es erróneo o bien porque hay ejemplos con la clase mal asignada. Todas estas observaciones anómalas (*outliers*) hacen que *boosting* tenga dificultades de generalización [Quinlan, 1996a; Opitz y Maclin, 1999; Dietterich, 2000b]. Este comportamiento parece lógico, ya que *boosting* incrementa el peso de los ejemplos mal clasificados sin tener en cuenta si esos ejemplos están mal clasificados porque son ejemplos difíciles o simplemente porque son datos anómalos. Este problema se afronta en [Rätsch *et al.*, 2001] donde proponen un algoritmo de *boosting* regularizado que evita construir hipótesis usando conjuntos de datos donde unos pocos ejemplos tengan la mayoría del peso.

Otro problema que presenta *Boosting* es de agotamiento (*underflow*). Como hicieron ver Bauer y Kohavi en [Bauer y Kohavi, 1999] se trata de un problema técnico que, a menudo, es omitido en las descripciones de algoritmos de *boosting*. El problema aparece cuando se tienen instancias bien clasificadas en bastantes iteraciones. Si, además, el error de clasificación es pequeño entonces las instancias bien clasificadas en $n$ iteraciones verán reducido su peso en aproximadamente un factor $2^n$, lo que puede llevar al agotamiento. La solución propuesta en [Bauer y Kohavi, 1999] es usar un valor mínimo de forma que cuando algún peso baja de ese valor mínimo se le asigna el valor mínimo. Webb en su implementación de *boosting* utilizó un valor mínimo de $10^{-8}$ [Webb, 2000]. Para reducir

el agotamiento también es preferible utilizar la modalidad de variación de pesos que sólo necesita un paso, esto es, dividir los ejemplos bien clasificados por $2(1-\epsilon_t)$ y por $2\epsilon_t$ los mal clasificados. Esto evita realizar, como hemos visto, el paso de la normalización.

La reducción del error en *boosting* se puede explicar como una reducción del sesgo del algoritmo y de la varianza como ha sido demostrado tanto en [Bauer y Kohavi, 1999] como en [Webb, 2000]. La mejora en varianza, se explica ya que la combinación de clasificadores mediante voto hace que el clasificador final sea más estable. Además, y al contrario que en *bagging*, *boosting* no hereda la tendencia central errónea del algoritmo base (que da lugar al sesgo), ya que cada clasificador individual se intenta construir de forma que no cometa los mismos errores que los clasificadores previamente generados.

*Boosting* es un método mucho más agresivo con el margen que *bagging*, como se muestra en [Schapire *et al.*, 1998]. Esto ocurre así porque, en *boosting*, la construcción de cada clasificador se centra más en los ejemplos mal clasificados anteriormente. De hecho, los ejemplos con mayor peso son los que han sido más veces mal clasificados y, por tanto, es probable que correspondan a ejemplos con un margen menor. De hecho, *boosting* se puede analizar como un algoritmo que realiza un descenso por gradiente de una función de error que optimiza el margen [Rätsch *et al.*, 2001]. Pero, ¿realmente la reducción del margen de los ejemplos de entrenamiento garantiza la reducción del error de test? Como ya hemos mencionado varias investigaciones han mostrado que aumentar el margen mínimo de clasificación no asegura mejores prestaciones de generalización. Por otra parte, existen una serie de trabajos de investigación que acotan en función del margen el error de generalización de *boosting* [Schapire *et al.*, 1998; Rätsch *et al.*, 2001] . Estos límites superiores de generalización se basan en la teoría del aprendizaje PAC (*Probably and Approximately Correct*) y dan cotas bastante holgadas para el error de generalización.

Finalmente, *boosting* tiene otro tipo de limitaciones. Dado que se usan pesos en los ejemplos, el algoritmo base de clasificación que se escoja ha de permitir el uso de pesos. Si esto no es así se puede simular construyendo el clasificador $h_t$ a partir de un conjunto de entrenamiento obtenido mediante un muestreo *bootstrap* ponderado usando los pesos $\boldsymbol{w}_t$ (*boosting* con remuestreo). Sin embargo, esta variante es menos efectiva que la de usar los pesos directamente en el algoritmo base [Quinlan, 1996a]. Por otra parte, *boosting* es un algoritmo secuencial donde para construir cada clasificador es necesario haber construido el anterior. Por tanto *boosting* no se puede paralelizar.

## 2.7. Otros conjuntos de clasificadores

### 2.7.1. *Wagging*

La técnica denominada *wagging* (*weight aggregation*), propuesta en [Bauer y Kohavi, 1999], es una técnica de construcción de clasificadores que, al igual que *boosting*, utiliza pesos en los ejemplos de entrenamiento. Sin embargo, en *wagging*, al contrario que en *boosting*, los pesos de los ejemplos no se asignan de forma adaptativa. En *wagging* cada clasificador base se genera utilizando pesos aleatorios para los ejemplos. Estos pesos se obtienen aleatoriamente a partir de una distribución normal con media uno. En general habrá ejemplos a los que se les reduce el peso a 0 lo que en la práctica significa que son eliminados del conjunto de datos de entrenamiento. Este método es más próximo a *bagging* que a *boosting* ya que cada clasificador utiliza un número limitado de ejemplos dependiente de la desviación estándar del ruido gaussiano que se añade. Bauer y Kohavi mostraron que con desviaciones estándar entre 2 y 3 los resultados de *wagging* y *bagging* eran muy parecidos [Bauer y Kohavi, 1999]. Webb introdujo una variante de *wagging* donde los pesos de los ejemplos no se reducen a 0 como en la formulación original [Webb, 2000]. De esta forma se consigue que todos los ejemplos sean usados para construir todos los clasificadores. La función que se utiliza en [Webb, 2000] para asignar los pesos a los ejemplos es la distribución continua de Poisson dada por

$$Poisson() = -\log\left(\frac{Random(1,999)}{1000}\right) , \tag{2.23}$$

donde $Random(min, max)$ devuelve un número aleatorio entre $min$ y $max$.

Al igual que en *bagging* la reducción del error de este algoritmo es básicamente debida a la reducción en varianza. Sin embargo, *wagging* no es tan efectivo como *bagging* para reducir la varianza, sea usando ruido gaussiano [Bauer y Kohavi, 1999] o una distribución de Poisson [Webb, 2000]. Como en *bagging*, tampoco hay reducción en el sesgo ya que este algoritmo no tiene ningún mecanismo para evitar la tendencia central errónea del algoritmo base. Al igual que *bagging*, este algoritmo se puede implementar en paralelo de forma bastante sencilla.

### 2.7.2. *Multiboosting*

*Multiboosting* consiste en combinar *wagging* y *boosting* [Webb, 2000]. En *multiboosting* se realizan varias inicializaciones de los pesos usando la distribución de Poisson (ec. (2.23)) y, a partir de cada una de ellas, realiza un proceso de *boosting*. A cada proceso de *boosting* independiente que parte de una inicialización aleatoria de los pesos se le llama comité. La clasificación final del conjunto utiliza el voto ponderado de todos los clasificadores obtenidos.

En los resultados mostrados en [Webb, 2000] se ve que *multiboosting* obtiene, en media, mejores resultados que *bagging*, *boosting* y *wagging*. *multiboosting* consigue reducir prácticamente el sesgo como *boosting* y la varianza como *wagging*. Es un algoritmo más estable que *boosting* frente a ruido aunque no tanto como *bagging* ya que en alguno de los conjuntos analizados incrementa el error del algoritmo base.

### 2.7.3. *Randomization*

Este método introducido por Dietterich y Kong [Dietterich y Kong, 1995; Dietterich, 2000b] construye árboles de decisión que seleccionan de manera aleatoria las particiones que se hacen en cada nodo. Para ello selecciona al azar entre las 20 mejores particiones de los datos en cada nodo. Todos los elementos del conjunto se construyen usando todos los datos del conjunto de entrenamiento. Se trata de un método perteneciente a la categoría de los bosques aleatorios (*random forests*), introducida por Breiman, así como al grupo de técnicas que introducen a aleatoriedad en el algoritmo de aprendizaje (sec. 2.3).

En la referencia [Dietterich, 2000b] se realizan una serie de experimentos para evaluar el funcionamiento de *randomization* bajo distintas condiciones. De los resultados obtenidos se concluye que *randomization* obtiene resultados ligeramente superiores a *bagging* aunque claramente inferiores a *boosting*. Sin embargo, y al igual que *bagging*, *randomization* es robusto frente al ruido. Para ver esto introdujeron ruido en las etiquetas de clase de algunas bases de datos (cambiando el 5 %, 10 % y 20 % de las etiquetas de los ejemplos por otras etiquetas del problema). Con estas configuraciones pudieron observar que *randomization* iguala los resultados de *boosting* con una tasa de ruido pequeña (5 %). Para ruidos en las etiquetas más altos *randomization* supera claramente a *boosting* aunque no a *bagging*.

### 2.7.4. Forest-RI y Forest-RC

Estos dos algoritmos similares y de tipo bosques aleatorios (*random forests*) se basan en modificar de manera aleatoria las entradas que recibe cada nodo para hacer las divisiones [Breiman, 2001]. El método Forest-RI consiste en seleccionar al azar en cada nodo un subconjunto de tamaño fijo $F$ de los atributos de entrada sobre los que realizar la división. En los experimentos realizados se usan dos valores de $F = 1, \log_2 M + 1$ con $M$ siendo en número de variables de entrada. El método Forest-RC genera $F$ nuevos atributos para cada nodo sobre los que calcular la partición. Estos $F$ atributos son generados aleatoriamente con una combinación lineal que contiene $L$ de los atributos originales con coeficientes generados aleatoriamente y a partir de una distribución uniforme en el intervalo $[-1, 1]$. Los valores utilizados para los experimentos fueron $F = 2$ y $8$ y $L = 3$. Además, estos métodos se conjugan con *bagging* de forma que cada árbol se construye sobre una muestra *bootstrap* del conjunto de entrenamiento.

Los resultados que obtienen estos métodos son excelentes, más teniendo en cuenta que

no realizan ningún tipo de proceso adaptativo a los datos como en *boosting*. Ambos algoritmos presentan resultados competitivos e incluso mejores que *boosting*, siendo a la vez robustos frente al ruido en las etiquetas de clase.

# Parte I

# Nuevos conjuntos de clasificadores

# Capítulo 3

# Conjuntos de árboles de crecimiento y poda iterativos

*En el presente capítulo se presentan tres nuevos métodos de construcción de conjuntos de clasificadores que se caracterizan por usar sin modificaciones todos los datos de entrenamiento para construir cada uno de los clasificadores del conjunto. El algoritmo base, presentado en [Gelfand* et al.*, 1991], construye un árbol de decisión de forma iterativa a partir de un conjunto de datos que se divide en dos subconjuntos. En cada iteración, uno de los dos subconjuntos se utiliza para hacer crecer el árbol a partir del árbol de decisión obtenido en la iteración anterior. Una vez que se ha hecho crecer el árbol hasta su tamaño máximo éste se poda usando el otro subconjunto de datos. Los papeles de los subconjuntos se intercambian en cada iteración. Este proceso converge a un árbol final que es estable con respecto a la secuencia de pasos de crecimiento y poda. Para generar una variedad de clasificadores en el conjunto se crean tantas divisiones aleatorias de los ejemplos en dos subconjuntos como árboles se quieran construir. Basándose en este procedimiento hemos propuesto tres nuevos métodos de construcción de conjuntos de clasificadores: conjunto IGP, boosting IGP y comités IGP. Estos métodos obtienen buenos resultados de clasificación en varias bases de datos estándar con un coste computacional menor que los conjuntos basados en CART.*

## 3.1. Introducción

Como ya hemos visto en el capítulo previo, el estudio de los conjuntos de clasificadores es un tema de gran actividad dentro del aprendizaje supervisado. Esta actividad ha sido motivada por las mejoras que se pueden obtener con estas técnicas sencillas. Estos métodos tienen como objetivo obtener un conjunto de clasificadores diversos que cuando combinan sus decisiones obtienen mayor precisión que los clasificadores individuales. A menudo este aumento de diversidad viene acompañado con un deterioro de la capacidad de

generalización de los clasificadores individuales. Así por ejemplo en *bagging* se descartan datos de entrenamiento para generar cada clasificador individual. En *boosting* se modifica la distribución de los pesos de los ejemplos, lo que puede llevar a generar clasificadores que ajustan correctamente los datos con los pesos modificados pero que obtienen un error elevado en el problema original.

En este capítulo se presentan tres nuevos métodos de construcción de clasificadores que introducen diversidad sin reducir la eficiencia del algoritmo base. Se basan en la variabilidad intrínseca del algoritmo de crecimiento y poda iterativos (*Iterative Growing and Pruning Algorithm*, IGP) [Gelfand *et al.*, 1991], un método de construcción de árboles de decisión basado en repetir secuencias de crecimiento y poda. IGP genera árboles de decisión dividiendo los datos de entrenamiento en dos subconjuntos de aproximadamente igual tamaño y distribución de clases. IGP usa iterativamente uno de los subconjuntos para hacer crecer el árbol y el otro para podarlo, intercambiando los papeles de los subconjuntos en cada iteración. Este algoritmo tiene la propiedad de que, a pesar de partir del mismo conjunto, distintas divisiones de los datos de entrenamiento generan árboles distintos. La inestabilidad del algoritmo IGP junto con el hecho de que se utilizan todos los datos para construir cada clasificador individual debería permitir construir conjuntos con buena capacidad de generalización.

## 3.2. Algoritmo de aprendizaje

### 3.2.1. Algoritmo base, árboles IGP

El algoritmo de crecimiento y poda iterativos es un algoritmo desarrollado por Gelfand *et al.* para la construcción de árboles de decisión [Gelfand *et al.*, 1991]. Este algoritmo tiene la propiedad, al igual que CART [Breiman *et al.*, 1984], de que utiliza todos los datos para hacer crecer y para podar el árbol. El pseudocódigo del algoritmo IGP se muestra en la figura 3.1.

Previamente a la construcción de un árbol con IGP se divide el conjunto de datos de entrenamiento $L$ en dos subconjuntos, $L_1$ y $L_2$, de aproximadamente igual tamaño y con distribuciones de clases aproximadamente iguales. Una vez divididos los datos, el algoritmo IGP utiliza uno de los subconjuntos para hacer crecer el árbol y el otro para podarlo. La secuencia de crecimiento y poda es repetida intercambiando los papeles de los subconjuntos en cada iteración. Es decir, primero se genera un árbol $T_0$ usando $L_1$, una vez generado el árbol $T_0$, éste se poda hasta el tamaño óptimo $T_0^*$ con respecto al subconjunto $L_2$. Una vez que la primera poda se ha completado, los papeles de los subconjuntos de datos son intercambiados y se hace crecer un nuevo árbol $T_1$ a partir de los nodos terminales de $T_0^*$ utilizando $L_2$. A continuación $T_1$ se poda a su tamaño óptimo con respecto a $L_1$. Los papeles de los subconjuntos de crecimiento y de poda se van intercambiando hasta que dos árboles podados consecutivos son de igual tamaño. Se ha demostrado que esta secuencia

converge [Gelfand *et al.*, 1991]. La demostración de la convergencia de este proceso se basa en que después de cada iteración el árbol podado resultante $T_k^*$ contiene o es igual al árbol podado previo $T_{k-1}^*$ ($T_{k-1}^* \leq T_k^*$) siempre que la clase $j(t)$ de cada nodo t se elija, al igual que en CART, por mayoría, y en caso de empate, se asigne la clase del nodo padre de $t$. Cuando los árboles $T_{k-1}^*$ y $T_k^*$ son iguales entonces el proceso ha convergido. Para la condición de parada del algoritmo (paso 8 de la figura 3.1) es suficiente comparar los tamaños de dos árboles sucesivos ya que se puede demostrar que la secuencia va incrementando el tamaño de los árboles y los árboles están anidados [Gelfand *et al.*, 1991].

Al igual que CART, IGP utiliza el criterio de Gini (ec. (2.11), sección 2.2) para seleccionar las divisiones en los nodos internos, en el proceso de crecimiento del árbol.

**Entradas:**
Conjunto de entrenamiento $L$ dividido en $L_1$ y $L_2$
**Salida:**
$T^*$

1. Usar $L_1$ para generar un árbol $T_0$
2. $T_0^*$ = Podar($T_0$, $L_2$)
3. Asignar k:=1
   do {
4. if k es par Asignar i:=1 y j:=2 si no Asignar i:=2 y j:=1
5. Usar $L_i$ para generar un árbol $T_k$ a partir de los nodos terminales de $T_{k-1}^*$
6. $T_k^*$ = Podar($T_k$, $L_j$)
7. k:=k+1
8. } while($|T_k^*| <> |T_{k-1}^*|$)
9. Asignar $T^*$:= $T_k^*$

Figura 3.1: Pseudocódigo de árbol IGP

El método de poda utilizado en el algoritmo IGP se presenta en la figura 3.2. Este método devuelve el árbol más preciso con respecto a un conjunto de datos $L$. Los nodos se procesan de abajo a arriba empezando por los nodos terminales y procediendo de tal manera que un nodo no se procesa hasta que todos sus hijos han sido procesados. Para cada nodo $t$ se compara el error del nodo $R(t, L)$ y el error del subárbol que cuelga de $t$, $S(t, L)$. El error del subárbol se define como la suma de los errores de los nodos terminales que tienen al nodo $t$ como antecesor común. Un nodo se poda si el error del subárbol que pende de $t$ es mayor o igual al error del nodo $t$ con respecto a un conjunto de datos $L$ (ecuación (2.14)). Este método de poda es más rápido que la poda basada en validación cruzada que utiliza CART, en la cual es preciso construir árboles auxiliares para determinar los parámetros de poda.

**Entradas:**
Árbol $T$
Conjunto de datos $L$
**Salida:**
Árbol podado $T^*$

Para cada nodo $t \in T$, ordenados en modo que cada nodo
1. se procesa sólo después de que sus nodos hijos se hayan procesado:
2. Si $t \in \hat{T}$ entonces $S(t, L) = R(t, L)$
3. Si no
4. Asignar $S(t, L) = S(t_L, L) + S(t_R, L)$
5. Si $S(t, L) \geq R(t, L)$ entonces
6. Podar nodo $t$
7. Asignar $S(t, L) = R(t, L)$

Figura 3.2: Método de poda de IGP

### 3.2.2. Conjuntos basados en IGP

La eficacia de *bagging* para reducir el error de generalización es elevada cuando el clasificador base tiene un sesgo bajo y alta varianza [Breiman, 1998]. El algoritmo IGP es inestable con respecto a cómo son asignados los ejemplos a los dos subconjuntos de datos utilizados por dicho algoritmo. Por consiguiente la variabilidad necesaria para la construcción de los conjuntos de clasificadores se puede obtener empleando distintas particiones del conjunto de entrenamiento generadas aleatoriamente. Este mecanismo no se puede utilizar en otros algoritmos de construcción de árboles de decisión, como CART o C4.5, ya que el ordenamiento o agrupamiento de los datos no modifica su funcionamiento. Realmente en CART puede haber pequeñas variaciones a la hora de elegir el árbol final de entre la familia de árboles definida en la ec. (2.16) si se realizan particiones distintas para la validación cruzada. Los conjuntos de clasificadores que usan C4.5 o CART como clasificadores base generan variabilidad introduciendo una perturbación no intrínseca del algoritmo o datos (muestreo *bootstrap* en *bagging*, poderación de los pesos de los ejemplos, etc). Esta perturbación generalmente deteriora la eficacia en clasificación de los árboles individuales. A continuación se presentan tres métodos para construir conjuntos de clasificadores basados en árboles IGP que aprovechan la inestabilidad intrínseca de su proceso de construcción.

### Conjunto IGP

El primer método basado en árboles IGP que proponemos genera un conjunto de clasificadores en el que cada árbol IGP se crea con una subdivisión aleatoria diferente de los datos (ver pseudocódigo en figura 3.3). De este modo, la variabilidad de los clasificadores del conjunto IGP es intrínseca al algoritmo de construcción de árboles y no se impone de manera *ad-hoc*. Además la inestabilidad no se consigue a costa de reducir la precisión de los árboles de decisión individuales como en *bagging* o en *boosting*.

**Entradas:**
Conjunto de entrenamiento $L$ de tamaño $N$
Número de clasificadores $T$
**Salida:**
$H(\boldsymbol{x}) = \underset{y}{argmax} \sum_{t=1}^{T} I(h_t(\boldsymbol{x}) = y)$

1. for $t = 1$ to $T$ $\{$
2. Dividir aleatoriamente $L$ en $L_1$ y $L_2$
3. $h_t$ = IGP($L_1$, $L_2$)
4. $\}$

Figura 3.3: Pseudocódigo de conjunto IGP

### *Boosting* IGP

También se ha desarrollado un algoritmo basado en *boosting* utilizando arboles IGP (figura 3.4). Se trata de un método adaptativo donde los pesos de los ejemplos se modifican dentro de cada uno de los dos subconjuntos en los que se han dividido los datos para el algoritmo IGP. A diferencia del método anterior, en este algoritmo sólo se genera una particion inicial de los datos. La modificación de pesos se hace siguiendo la regla de *boosting* dentro de cada subconjunto de datos. Un aspecto crucial en el diseño del algoritmo es la elección del criterio de parada adecuado. El método *boosting* IGP propuesto puede parar cuando un clasificador, en el conjunto de entrenamiento, alcanza error cero o un error mayor que $0.5$ ponderado con los pesos asignados a los ejemplos (figura 3.4 líneas 6–9). Sin embargo, esto no es suficiente, dado que los pesos se adaptan dentro de cada subconjunto de datos y por tanto hay que comprobar también que el error ponderado del árbol generado no supere $0.5$ o alcance el error $0$ en cada uno de los subconjuntos. De no hacerlo así, al modificar los pesos de los ejemplos (líneas 15 y 16), se podrían aumentar los pesos de los ejemplos bien clasificados y reducir los pesos de los ejemplos mal clasificados para alguno de los subconjuntos. Si se para la ejecución cuando esto sucede entonces el algoritmo a

**Entrada:**

Conjunto de entrenamiento $L$ dividido en $L_1$ y $L_2$
Número de clasificadores $T$

**Salida:**

$H(\boldsymbol{x}) = \underset{y}{argmax} \sum_{t=1}^{T} I(h_t(\boldsymbol{x}) = y)$

```
1.  asignar w_1[i] = 1/N, i = 1,...,N
2.  for t=1 to T {
3.    h_t = ArbolIGP(L_1, L_2, w_t^1, w_t^2)
4.    ε_t = Error(h_t, L, w_t)
5.    β_t = ε_t/(1 − ε_t)
6.    if (ε_t ≥ 0.5 or ε_t = 0) {
7.      if (ε_t ≥ 0.5) desechar h_t
8.      break
9.    }
10.   for k=1 to 2 {
11.     ε_k = Error(h_t, L_k, w_t^k)
12.     if (ε_k ≥ 0.5 or ε_k = 0) asignar w_{t+1}^k[i] = 1/N, i = 1,...,N_k
13.     else {
14.       for j=1 to N_k {
15.         if (h_t(x_j^k) ≠ y_j^k) then w_{t+1}^k[j] = w_t^k[j]/2ε_k
16.         else w_{t+1}^k[j] = w_t^k[j]/2(1 − ε_k)
17.       }
18.     }
19.   }
20. }
```

Figura 3.4: Pseudocódigo de *boosting* IGP

menudo se detiene sin llegar al numero de clasificadores propuesto. Con el fin de evitar una parada prematura del algoritmo se ha introducido una pequeña variación que consiste en reasignar los pesos de los ejemplos del subconjunto a $1/N$ cuando el error ponderado del último clasificador generado supera $0.5$ (o alcanza error 0) en el subconjunto (línea 12). Si se reinicializan los pesos dentro de los dos subconjuntos de datos en una misma iteración se volvería a la situación inicial y consecuentemente se generarían de nuevo los mismos clasificadores. Sin embargo, la posibilidad de que esto suceda es muy baja. Esta situación es sólo posible cuando el error ponderado en un subconjunto es 0 y en el otro mayor de $0.5$, situación muy improbable ya que se usan ambos subconjuntos para hacer crecer y podar el

árbol. No se puede dar el caso en el que el error ponderado supere $0.5$ (o alcance error 0) en ambos subconjuntos ya que en estos casos el algoritmo se habría parado previamente en la línea 8. En este método se han seguido las propuestas de [Bauer y Kohavi, 1999] para evitar el agotamiento (*underflow*) en las operaciones con los pesos (ver sección 2.6).

### Comités de árboles IGP

Finalmente, proponemos un algoritmo que combina los conjuntos IGP (figura 3.3) con *boosting* IGP (figura 3.4). El algoritmo completo se muestra en la figura 3.5. Para combinarlos se substituye el clasificador base del conjunto IGP por un conjunto generado con *boosting* IGP. De este modo, los dos algoritmos se complementan con la idea de aprovechar la capacidad de reducir el error de los algoritmos de *boosting* y la estabilidad frente al ruido de los algoritmos de tipo *bagging*. Sin embargo, no hemos usado *bagging* directamente sino el conjunto IGP que utiliza todos los datos de entrenamiento para crear cada árbol. El algoritmo propuesto es similar a *multiboosting* (descrito en la sección 2.7.2) que parte de la idea de combinar la capacidad para reducir la varianza de *wagging* con la capacidad para reducir el sesgo de *boosting* [Webb, 2000].

**Entrada:**
Conjunto de entrenamiento $L$ de tamaño $N$
Número de comités $T_1$
Número de clasificadores por comité $T_2$
**Salida:**
$H(\boldsymbol{x}) = \underset{y}{argmax} \sum_{t=1}^{T_1} I(h_t(\boldsymbol{x}) = y)$

1. for $t = 1$ to $T_1$ {
2. Dividir aleatoriamente $L$ en $L_1$ y $L_2$
3. $C_t$ = *Boosting*IGP($L_1$, $L_2$, $T_2$)
4. }

Figura 3.5: Pseudocódigo de comités IGP

El algoritmo propuesto consiste en reemplazar la línea 3 de la figura 3.3 por el algoritmo *boosting* IGP. Cada uno de los clasificadores base dentro del conjunto principal lo denominaremos comité siguiendo la terminología introducida en [Webb, 2000]. Este algoritmo tiene, aparte del conjunto de datos $L$, otros dos parámetros: El parametro $T_1$ indica el número de clasificadores base a generar, en este caso el número de comités a generar con *boosting* IGP; El parámetro $T_2$ identifica el número de clasificadores a construir dentro del *boosting* IGP, esto es, el número de miembros de los que se compone cada comité. La salida

del algoritmo son $T_1$ comités que votan para obtener la clasificación final y está compuesta en total por $T = T_1 \times T_2$ árboles IGP.

La decisión final del conjunto se toma en dos etapas. Primero, cada comité toma una decisión consultando a sus miembros y, posteriormente, las decisiones de los comités se combinan de nuevo mediante voto para dar lugar a la decisión final.

Este método se puede ver como un algoritmo intermedio entre los algoritmos descritos previamente. De hecho, si se ejecuta con un comité ($T_1 = 1$), se recupera el *boosting* IGP. Y si se ejecuta con varios comités de un solo miembro ($T_2 = 1$), entonces recuperamos el conjunto IGP.

## 3.3. Resultados experimentales

Los algoritmos propuestos han sido evaluados en una serie de conjuntos de datos de problemas de aplicación obtenidos de la colección de problemas de UCI [Blake y Merz, 1998]. Estos son: *Breast Cancer Wisconsin*, *Pima Indian Diabetes*, *German Credit*, *Sonar* y *Waveform*. Para evitar efectos espurios debidos a la ausencia de valores para algunos atributos, se han elegido conjuntos de datos con todos los registros completos. Asimismo, para analizar la eficacia del conjunto IGP en función del tamaño del conjunto de datos de entrenamiento hemos realizado un estudio más detallado con el conjunto sintético *Waveform*, propuesto en [Breiman *et al.*, 1984].

El cuadro 3.1 muestra las características de los conjuntos seleccionados. Las columnas 2 y 3 dan el número de ejemplos de entrenamiento y test respectivamente. La columna 4 muestra el número de atributos del problema y la columna 5 el número de clases. Más detalles sobre las bases de datos seleccionadas se pueden encontrar en el apéndice A.

Cuadro 3.1: Características de los conjuntos de datos

| **Problema** | **Entrenamiento** | **Test** | **Atributos** | **Clases** |
|---|---|---|---|---|
| Breast Cancer Wisconsin | 500 | 199 | 9 | 2 |
| Pima Indian Diabetes | 500 | 268 | 8 | 2 |
| German Credit | 600 | 400 | 24 | 2 |
| Sonar | 120 | 88 | 60 | 2 |
| Waveform | 300 | 5000 | 21 | 3 |

Los algoritmos propuestos (conjunto IGP, *boosting* IGP y comités IGP) han sido comparados con *bagging* y *boosting* basados en CART. El tamaño de todos los conjuntos se ha fijado en $T = 99$ clasificadores ($T_1 \times T_2 = 11 \times 9$ para comités IGP). Como hemos mencionado se han realizado dos tipos de experimentos. Primero, se ha medido la eficacia

de los algoritmos con respecto al número de clasificadores generando una serie de 99 clasificadores para cada ejecución y conjunto. El segundo experimento ha sido diseñado para estudiar la dependencia del error de generalización con el número de datos utilizados en la fase de entrenamiento. Esta prueba se ha realizado para el conjunto IGP y para *bagging* utilizando árboles CART como algoritmo base. Además, también se ha medido el error de generalización de los clasificadores individuales CART e IGP. Sus tasas de error nos servirán como medida de referencia para los conjuntos de clasificadores y establecer si éstos mejoran al algoritmo base.

Para cada conjunto de datos del cuadro 3.1: (i) Se generan $N = 50$ conjuntos de entrenamiento aleatorios con los tamaños especificados en el cuadro 3.1; (ii) cada algoritmo se ejecuta $N$ veces, una vez por conjunto de datos de entrenamiento; (iii) finalmente, su capacidad de generalización se mide en el conjunto de test promediando sobre las $N$ ejecuciones. De esta forma los distintos algoritmos trabajan sobre las mismas particiones y los errores son directamente comparables. Se ha usado la prueba-t de Student pareada de dos colas para determinar si las diferencias son estadísticamente significativas: La prueba-t de Student mide la probabilidad (valor-p) de que dos poblaciones tengan igual media asumiendo que las diferencias entre las poblaciones es una variable aleatoria con una distribución aproximadamente normal o con una distribución t. Valores-p en torno a $10$–$1\,\%$ son valores habitualmente utilizados para determinar una diferencia estadísticamente relevante [Ross, 1987]. Varios estudios [Salzberg, 1997; Dietterich, 1998a; Nadeau y Bengio, 2003] critican el uso de la prueba-t de Student para determinar diferencias significativas en aprendizaje automático. Las críticas se basan en que normalmente las diferencias entre las poblaciones no proceden de una variable aleatoria con distribución aproximadamente normal o con distribución t en las configuraciones típicas de los experimentos de aprendizaje automático. Además se argumenta que la prueba-t está lejos de su valor nominal. Esto hace que a menudo se obtengan diferencias cuando no las hay (error de tipo I). Por ello hemos preferido siempre dar el valor-p obtenido (en vez de simplemente resaltar los resultados) y hemos utilizado un umbral para considerar las diferencias entre algoritmos como significativas más restrictivo de lo habitual (valor-p$< 0.5\,\%$). Además, se puede ver cómo en muchos de los casos los valores-p son mucho menores que $0.5\,\%$.

El cuadro 3.2 muestra los resultados de ejecutar individualmente los clasificadores CART e IGP. El mejor algoritmo para cada conjunto de datos se ha marcado en negrita. En la última columna del cuadro se muestran los valores de la prueba-t de Student pareada de dos colas. Se puede observar que las diferencias no son significativas entre ambos métodos (valor-p$< 0.005$), aunque CART construye árboles con menor error de generalización que IGP en 4 de los 5 conjuntos de datos. También hay que destacar que el error del árbol CART es menor que el del *bagging* CART con un solo clasificador (primera fila del cuadro 3.3). Esto se debe a que el CART individual se ha construido usando todos los elementos del conjunto de entrenamiento, mientras que el muestreo *bootstrap* usado para generar los árboles del conjunto selecciona en media un $63.2\,\%$ de los datos originales. Este efecto no

Cuadro 3.2: Error medio en % para los clasificadores individuales (desviación estándar entre paréntesis)

| | CART | IGP | valores-p |
|---|---|---|---|
| Breast W. | 5.9(1.8) | **5.6(1.6)** | 0.35 |
| Diabetes | **25.9(2.5)** | 26.3(2.5) | 0.38 |
| German | **27.0(2.0)** | 28.3(2.1) | 0.0061 |
| Sonar | **30.1(4.0)** | 30.5(5.2) | 0.65 |
| Waveform | **30.1(2.0)** | 30.6(1.7) | 0.31 |

se produce en el caso del conjunto IGP, ya que los árboles se construyen con el mismo conjunto de datos de entrenamiento dentro del conjunto de clasificadores que cuando se ejecuta individualmente.

El cuadro 3.3 presenta los errores de generalización como la media de las 50 ejecuciones para los distintos conjuntos de clasificadores en tres secciones con las desviación estándar entre paréntesis. Las tres secciones del cuadro presentan los resultados para conjuntos con 1, 9 y 99 árboles, respectivamente. En la primera sección se muestran los resultados del conjunto IGP y *bagging* CART con 1 clasificador. Las secciones segunda y tercera están divididas en 5 filas que muestran los resultados para *bagging* CART, conjunto IGP, *boosting* CART, *boosting* IGP y comités IGP respectivamente. El mejor resultado para cada sección se ha resaltado en negrita. Para calcular el error de los comités IGP para $T = 9$ se han usado los 3 primeros árboles de los 3 primeros comités, lo que es equivalente a usar $T_1 = 3$ y $T_2 = 3$. Asimismo, los conjuntos de clasificadores fueron analizados secuencialmente para obtener el error de clasificación de 1 a 99 clasificadores. Estos resultados se muestran en las figuras 3.6–3.8. De nuevo, los comités IGP se han procesado en manera distinta. Para este algoritmo solo se han representado los siguientes puntos $T_1 \times T_2$: $3 \times 3$, $11 \times 3$, $11 \times 5$, $11 \times 7$ y $11 \times 9$. En todos los casos se han usado sólo conjuntos de tamaño impar para evitar empates en los procesos de votación.

### Descripción de los resultados

De forma general podemos ver que los conjuntos de clasificadores mejoran la clasificación con respecto al clasificador base. Como excepción a resaltar están los algoritmos *boosting* CART y *boosting* IGP para el conjunto de datos *Pima Indian Diabetes*. Este comportamiento coincide con el observado en estudios previos [Bauer y Kohavi, 1999] y confirma los problemas de generalización de *boosting* en ciertos problemas de clasificación considerados ruidosos. Esto contrasta con el comportamiento de *bagging* CART y el conjunto IGP que no presentan este problema en ninguno de los conjuntos de datos estudiados.

Comparando el conjunto IGP con *bagging* CART se puede ver en el cuadro 3.3 que

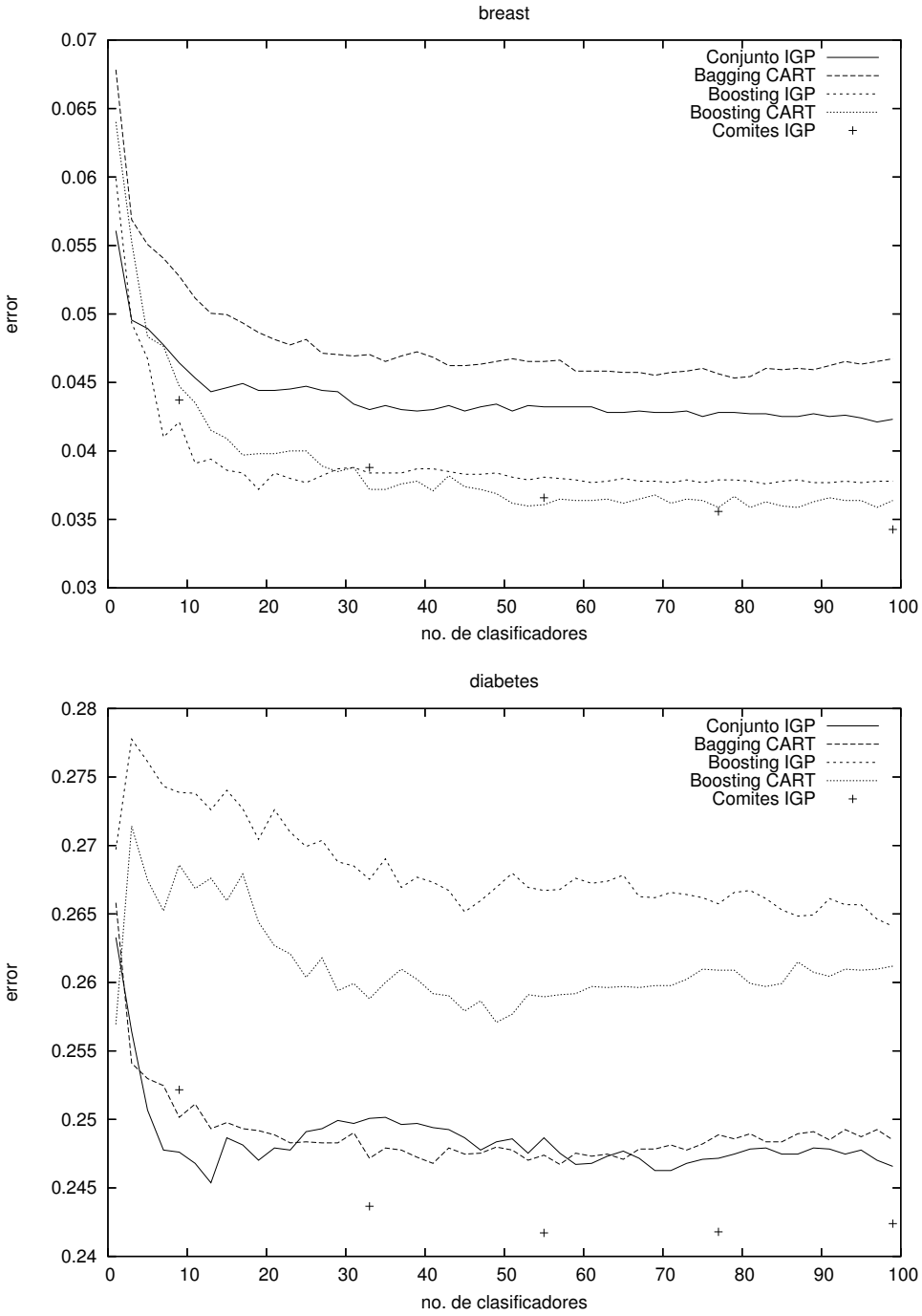



Figura 3.6: Evolución del error con respecto al número de clasificadores para los conjuntos de datos *Breast Cancer Wisconsin* (*gráfico superior*) y *Pima Indian Diabetes* (*gráfico inferior*)

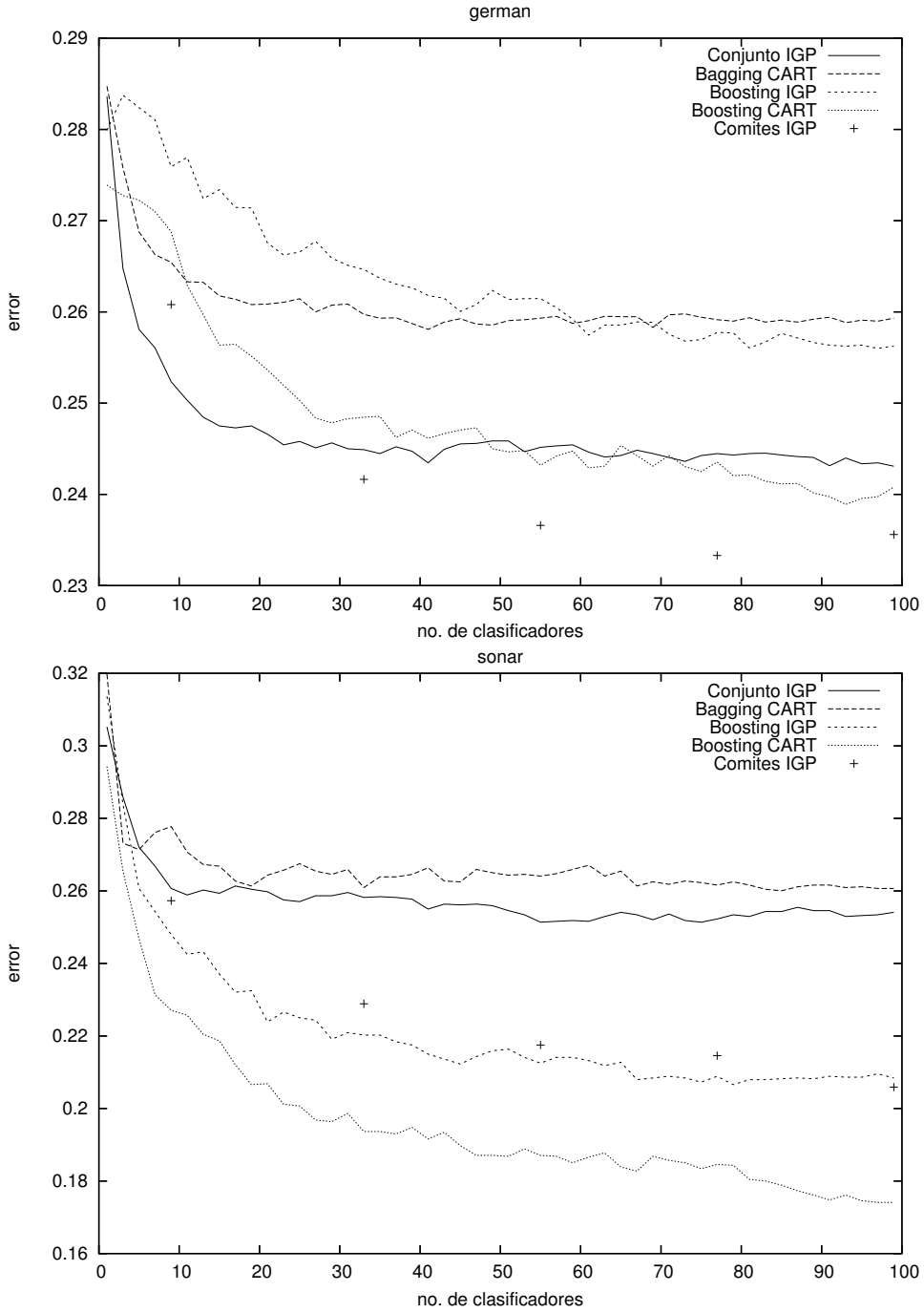



Figura 3.7: Evolución del error con respecto al número de clasificadores para los conjuntos de datos *German Credit* (*gráfico superior*) y *Sonar* (*gráfico inferior*)

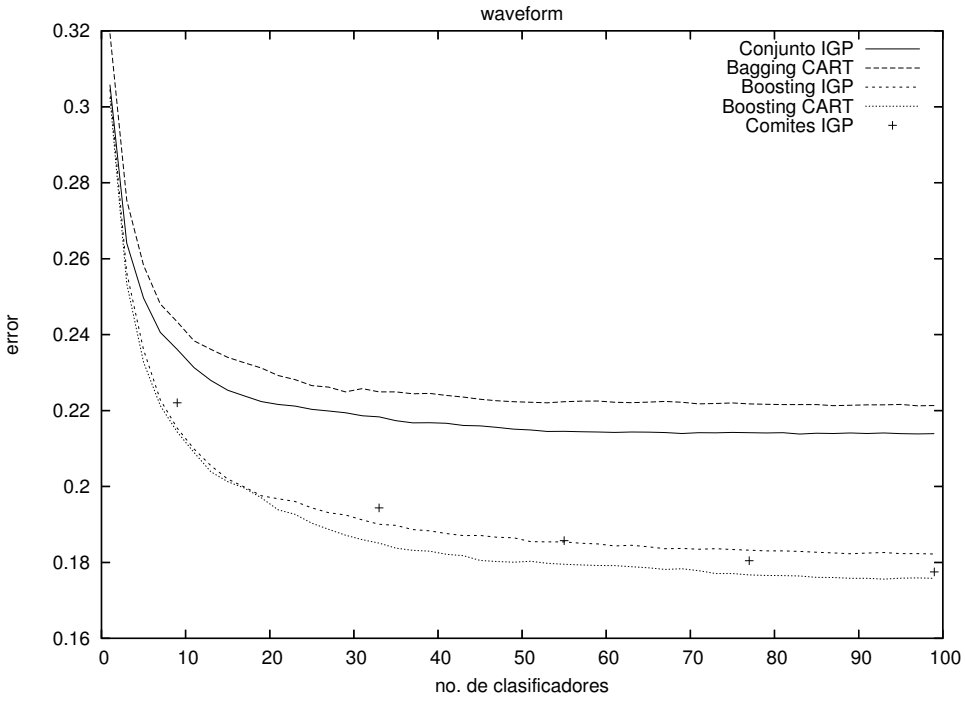



Figura 3.8: Evolución del error con respecto al número de clasificadores para el *Waveform*

para todos los conjuntos de datos el método propuesto obtiene resultados mejores o similares. De hecho, de los 5 problemas estudiados el conjunto IGP tiene menor error en *Breast Cancer Wisconsin*, *German Credit* y *Waveform*, y también en *Sonar* aunque en este último caso con menor margen. En *Pima Indian Diabetes* el resultado de ambos algoritmos es similar. Los valores-p de la prueba-t de Student pareada de dos colas entre ambos algoritmos y para distintos tamaños del conjunto de clasificadores se muestran en el cuadro 3.4 (valores con valor-p $< 0.005$ están resaltados en negrita). Se puede observar que las diferencias del conjunto IGP con respecto a *bagging* CART son estadísticamente significativas (con valores-p de hasta 2.3e-12) en la mayoría de conjuntos de datos analizados. Otro hecho a resaltar es que las diferencias entre ambos algoritmos se consiguen en las primeras iteraciones ($T = 9$) y se mantiene al añadir más clasificadores (ver figuras 3.6 (gráfico superior), 3.7 (gráfico superior) y 3.8).

Este funcionamiento generalmente mejor del método propuesto conjunto IGP sobre *bagging* CART puede ser debido al hecho de que en el conjunto IGP cada clasificador se construye utilizando todos los datos en vez de con el 62.3 % de los datos como en *bagging*.

Comparando *boosting* IGP y *boosting* CART se puede ver que el conjunto basado en CART obtiene mejores resultados en los problemas analizados. Comparando los conjuntos basados en *boosting* y los basados en *bagging* se ve que, en general, los algoritmos basados

Cuadro 3.3: Error medio para conjuntos compuestos de 1, 9 y 99 clasificadores (desviación estándar entre paréntesis)

| | | Breast | Diabetes | German | Sonar | Waveform |
|---|---|---|---|---|---|---|
| T=1 | *Bagging* CART | 0.0678 (0.022) | 0.266 (0.023) | 0.285 (0.019) | 0.320 (0.049) | 0.319 (0.017) |
| | Conjunto IGP | **0.0561** (0.016) | **0.263** (0.025) | **0.283** (0.021) | **0.305** (0.052) | **0.306** (0.017) |
| T=9 | *Bagging* CART | 0.0528 (0.016) | 0.250 (0.023) | 0.265 (0.019) | 0.278 (0.043) | 0.243 (0.021) |
| | Conjunto IGP | 0.0464 (0.017) | **0.244** (0.019) | **0.252** (0.018) | 0.261 (0.047) | 0.236 (0.016) |
| | *Boosting* CART | 0.0447 (0.011) | 0.269 (0.026) | 0.269 (0.021) | **0.227** (0.047) | **0.214** (0.012) |
| | *Boosting* IGP | **0.0421** (0.014) | 0.274 (0.023) | 0.276 (0.021) | 0.248 (0.052) | 0.215 (0.010) |
| | Comités IGP$^{(1)}$ | 0.0437 (0.013) | 0.252 (0.020) | 0.261 (0.017) | 0.257 (0.052) | 0.222 (0.0091) |
| T=99 | *Bagging* CART | 0.0467 (0.015) | 0.249 (0.019) | 0.259 (0.017) | 0.261 (0.042) | 0.222 (0.022) |
| | Conjunto IGP | 0.0423 (0.013) | 0.247 (0.023) | 0.243 (0.017) | 0.252 (0.043) | 0.214 (0.019) |
| | *Boosting* CART | 0.0364 (0.011) | 0.261 (0.018) | 0.241 (0.016) | **0.174** (0.039) | **0.176** (0.0064) |
| | *Boosting* IGP | 0.0378 (0.013) | 0.264 (0.022) | 0.256 (0.021) | 0.208 (0.050) | 0.182 (0.010) |
| | Comités IGP$^{(2)}$ | **0.0343** (0.011) | **0.242** (0.020) | **0.236** (0.014) | 0.206 (0.043) | 0.177 (0.0052) |

(1) 3 comités de 3 clasificadores (9 clasificadores en total)

(2) 11 comités de 9 clasificadores (99 clasificadores en total)

en *boosting* obtienen mejores resultados que los que usan *bagging*. Como excepción aparece de nuevo el conjunto *Pima Indian Diabetes*. Asimismo, en el conjunto *German Credit* se observa que para *boosting* el error de clasificación disminuye más lentamente con el

Cuadro 3.4: prueba-t para el conjunto IGP *vs. bagging* CART para 1, 9 y 99 clasificadores

| | $T = 1$ | $T = 9$ | $T = 99$ |
|---|---|---|---|
| Breast W. | **3.6e-4** | **2.7e-3** | **1.6e-3** |
| Diabetes | 0.50 | 0.088 | 0.33 |
| German | 0.64 | **6.6e-6** | **2.3e-12** |
| Sonar | 0.14 | 0.046 | 0.023 |
| Waveform | **0.0043** | **0.0042** | **2.8e-7** |

número de clasificadores que para *bagging*. Como consecuencia se obtienen peores resultados para pocos clasificadores y errores equivalentes o menores para un número elevado de clasificadores. Estos resultados coinciden con los obtenidos en otros estudios [Webb, 2000].

Con respecto a los comités IGP se puede ver en el cuadro 3.3 que, en general, es la mejor elección aunque su convergencia sea más lenta. Para $T = 9$ $(3 \times 3)$ los comités IGP no obtienen el mejor resultado en ninguno de los problemas estudiados mientras que para $T = 99$ $(11 \times 9)$ devuelve el mejor resultado en 3 de los 5 problemas y el segundo mejor en los otros dos conjuntos. En el cuadro 3.5 se muestran los resultados de la prueba-t de Student para $T = 99$ de los comités IGP con respecto a los otros 4 conjuntos de clasificadores. Se han resaltado los resultados con valor-p$< 0.005$ en la prueba-t de Student. En este cuadro se puede observar cómo, para $T = 99$, los comités IGP obtienen mejoras que son estadísticamente significativas (prueba-t de Student $< 0.005$) con respecto a *bagging* CART para todos los conjuntos excepto (de nuevo) para *Pima Indian Diabetes* donde obtienen errores equivalentes. Con respecto a *boosting* CART y para $T = 99$ las diferencias se reducen. Los comités IGP obtienen mejores resultados en 3 de los 5 conjuntos aunque las

Cuadro 3.5: Valores-p de la prueba-t de Student pareada para comités IGP con respecto al resto de conjuntos probados usando $T = 99$. Se ha resaltado en **negrita** los valores-p$< 0.005$. Los valores recuadrados corresponden a resultados desfavorables a comités IGP

| | *Bagging* CART | Conjunto IGP | *Boosting* CART | *Boosting* IGP |
|---|---|---|---|---|
| Breast W. | **3.5e-5** | **0.002** | 0.33 | 0.11 |
| Diabetes | 0.14 | 0.37 | **3.5e-6** | **3.1e-6** |
| German | **2e-10** | **0.022** | 0.085 | **3.6e-8** |
| Sonar | **9.6e-8** | **2.5e-6** | **2.4e-4** | 0.82 |
| Waveform | **6e-19** | **1e-18** | 0.66 | **1.2e-5** |

diferencias son sólo significativas en *Sonar*, a favor de *boosting* CART y en *Pima Indian Diabetes* a favor del algoritmo propuesto comités IGP. Además, se puede observar que de las 20 posibles comparaciones los comités IGP son más efectivos en 11, en 8 las diferencias no son estadísticamente significativas y en 1 es menos eficaz (*boosting* CART y conjunto *Sonar*).

### Variación con el número de ejemplos

En una segunda tanda de experimentos se ha medido la variación del error de clasificación para el conjunto IGP y *bagging* CART con respecto al número de datos de entrenamiento para el conjunto sintético *Waveform* [Breiman *et al.*, 1984]. El cuadro 3.6 muestra el error de test (promediado sobre 10 ejecuciones con la desviación estándar entre paréntesis) y el número medio de hojas de los árboles generados por el conjunto IGP y *bagging* CART. La última columna muestra los valores-p usando la prueba-t de Student. De nuevo, se han resaltado los valores con valor-p$< 0.005$. En cada iteración se ha generado un conjunto compuesto de 101 árboles usando los mismos conjuntos de entrenamiento para ambos algoritmos. Una representación gráfica de los resultados se muestra en la figura 3.9. Se puede observar que a medida que se incrementa el tamaño del conjunto de datos las diferencias entre ambos algoritmos también se incrementan. Esto puede ser debido a que el algoritmo IGP sólo usa una mitad de los datos para hacer crecer el árbol, lo que puede llevar a no tener datos suficientes para alcanzar el tamaño óptimo cuando se utilizan pocos datos. Sin embargo, a medida que aumenta el número de datos el conjunto IGP los aprovecha más eficientemente construyendo árboles casi del doble de tamaño que *bagging* CART.

Cuadro 3.6: Variación del error (en %) y tamaño del árbol (número de hojas) con respecto al tamaño del conjunto de entrenamiento para *Waveform* usando 101 clasificadores. La desviación estándar se indica entre paréntesis

| Tamaño | *Bagging* CART | $\|T\|$ | Conjunto IGP | $\|T\|$ | prueba-t |
|---|---|---|---|---|---|
| 50 | 26.1 (2.0) | 3.42 | 26.2 (2.8) | 3.61 | 0.8288 |
| 100 | 24.2 (3.0) | 4.64 | 24.0 (3.1) | 5.13 | 0.6856 |
| 150 | 23.9 (2.7) | 5.40 | 23.1 (2.1) | 6.59 | 0.0320 |
| 200 | 23.9 (1.9) | 6.30 | 23.0 (1.8) | 8.05 | 0.0552 |
| 250 | 24.0 (2.9) | 6.52 | 23.2 (2.6) | 9.09 | 0.0302 |
| 300 | 22.6 (3.2) | 7.90 | 21.9 (2.8) | 11.0 | 0.0203 |
| 500 | 20.4 (1.0) | 10.5 | 19.8 (0.8) | 16.2 | 0.0422 |
| 750 | 21.4 (1.9) | 12.6 | 20.1 (0.9) | 21.7 | 0.0176 |
| 1000 | 20.3 (1.9) | 15.5 | **18.6 (1.1)** | 28.3 | **0.0013** |

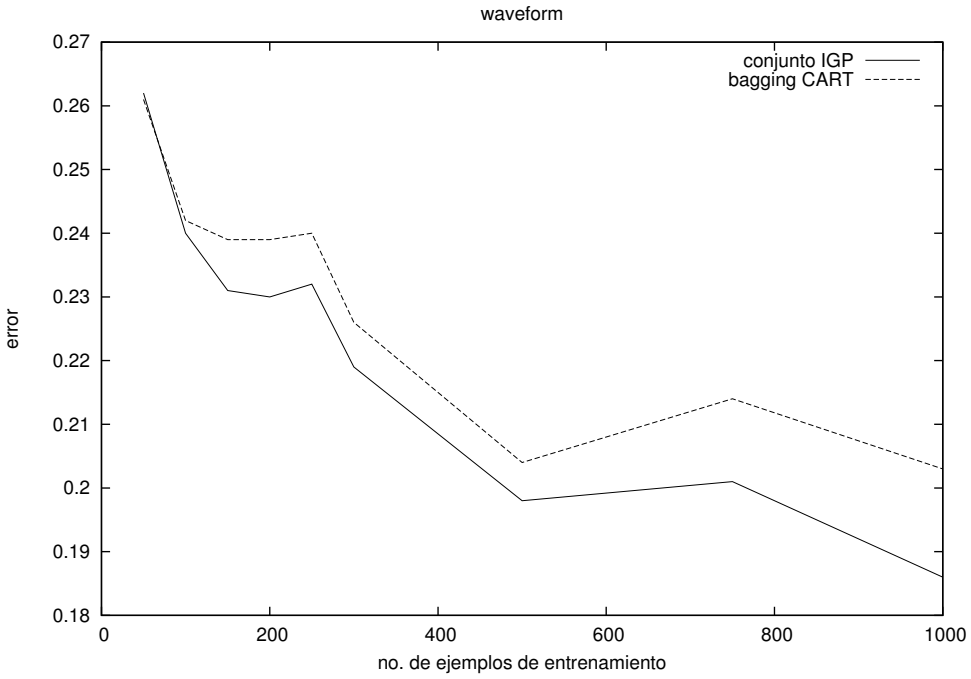



Figura 3.9: Variación del error con respecto al tamaño del conjunto de entrenamiento para *Waveform*

### Tiempos de ejecución

Las últimas medidas realizadas son los tiempos de ejecución de cada uno de los algoritmos. En el cuadro 3.7 se muestran los tiempos necesarios para construir un conjunto de clasificadores de 101 árboles usando el problema *Waveform* con 300 datos de entrenamiento. Se puede ver que los algoritmos que utilizan árboles construidos con el método IGP son mucho más rápidos que aquéllos basados en árboles CART. Esto se debe a que el algoritmo IGP no necesita construir árboles auxiliares para podar el árbol. El algoritmo IGP obtiene el tamaño final de forma iterativa tras pocas iteraciones (4 como máximo), mientras que

Cuadro 3.7: Tiempo medio (seg.) de ejecución para construir conjuntos de 101 clasificadores para *Waveform* con 300 datos de entrenamiento (usando un ordenador con procesador Celeron® a 400 MHz.)

| | *Bagging* CART | Conjunto IGP | *Boosting* CART | *Boosting* IGP | Comités IGP |
|---|---|---|---|---|---|
| tiempo (seg.) | 538 | 59 | 604 | 100 | 97 |

CART necesita construir árboles auxiliares para la poda (validación cruzada con 10 árboles). Además los árboles IGP sólo usan la mitad de los datos tanto para generar como para podar el árbol.

## 3.4. Conclusiones

Es este capítulo se han presentado y analizado tres nuevos métodos para la construcción de conjuntos de clasificadores: conjunto IGP, *boosting* IGP y Comités IGP. Todos ellos están basados en el algoritmo de generación de árboles de decisión de crecimiento y poda iterativos (IGP) [Gelfand *et al.*, 1991].

Se ha observado que el conjunto IGP genera de forma natural un conjunto de clasificadores diversos sin necesidad de añadir aleatoriedad espuria en el conjunto de datos o en el procedimiento de aprendizaje. Los experimentos realizados en problemas estándar de la colección UCI muestran cómo los conjuntos de clasificadores generados asignando de manera aleatoria los ejemplos de entrenamiento a cada uno de los dos subconjuntos utilizados en el algoritmo IGP obtienen mejoras de clasificación respecto a conjuntos de clasificadores generados con *bagging* utilizando CART como algoritmo base. Esto indica que el conjunto IGP obtiene clasificadores suficientemente diversos a pesar de que todos se construyen usando los mismos ejemplos de entrenamiento.

Además, se puede observar que cuando se incrementa el tamaño del conjunto de entrenamiento se incrementa la mejora dada por el conjunto IGP con respecto a *bagging* CART en el conjunto sintético *Waveform*. Esta mejora parece tener su explicación en el incremento de la diferencia de tamaño de los árboles generados. El algoritmo IGP ha obtenido en las pruebas realizadas árboles más grandes en promedio que CART. Las diferencias de error observadas varían en paralelo con las diferencias de tamaño en los árboles generados por los algoritmos. Estas diferencias tanto en el tamaño de los árboles generados como en el error se incrementan a medida que aumenta el número de ejemplos de entrenamiento.

La variante de *boosting* con árboles IGP propuesta obtiene, en la mayoría de los problemas analizados, un error menor que los algoritmos tipo *bagging*, pero presenta equivalentes o peores resultados que *boosting* basado en CART. Una posible explicación es que la modificación de los pesos dentro de los dos grupos de datos no consigue en la misma medida que *boosting* que cada ejecución se centre más en los datos mal clasificados por los clasificadores base previamente generados.

Asimismo, se ha mostrado que los Comités IGP obtienen resultados excelentes en los problemas explorados. En la mayoría de problemas analizados, los errores de clasificación son equivalentes a *boosting* CART y, además, no presenta los problemas de generalización que tiene *boosting* en algunos conjuntos con ruido. Parece que los comités de árboles IGP consiguen el comportamiento robusto de *bagging* para no aumentar el error del algoritmo base y, al mismo tiempo, mantienen la eficacia de *boosting* para reducir el error en

conjuntos no ruidosos.

También hay que resaltar que los métodos presentados son más eficientes desde un punto de vista computacional que los conjuntos de clasificación basados en CART. En CART se necesita construir árboles auxiliares para obtener los parámetros de poda por validación cruzada (normalmente de 10 árboles) mientras que en el algoritmo IGP sólo se genera un árbol por cada miembro del conjunto. Además, los pasos de crecimiento y poda son sólo sobre la mitad de los datos, lo que conduce a una considerable reducción del tiempo de proceso. Además el algoritmo IGP converge tras pocas iteraciones (normalmente 2 ó 3 iteraciones y no más de 4) en los conjuntos estudiados.

Finalmente, hemos observado que el algoritmo IGP obtiene resultados equivalentes o ligeramente peores que CART cuando se ejecuta individualmente. Esto contradice las conclusiones dadas en [Gelfand *et al.*, 1991]. Puede ser debido a diferencias en la implementación de los algoritmos y a que los resultados experimentales expuestos en dicho artículo no son muy extensos: sólo se muestran los errores para 5 ejecuciones de IGP y CART en el problema *Waveform* utilizando 300 ejemplos de entrenamiento.

# Capítulo 4

# Conjuntos de clasificadores generados por la alteración de las etiquetas de clase de los ejemplos

*En este capítulo se presenta un conjunto de clasificadores cuyos miembros son construidos a partir de alteraciones de las etiquetas de clase de un porcentaje de ejemplos elegidos aleatoriamente de entre los que componen el conjunto de entrenamiento. Utilizando este método se pueden obtener grandes mejoras en el error de clasificación cuando se utiliza una alta probabilidad de modificación de etiquetas de clase y se generan conjuntos con un número elevado de clasificadores. Asimismo se muestra cómo los clasificadores generados siguiendo este procedimiento cometen errores en el conjunto de entrenamiento estadísticamente no correlacionados. La dependencia del error de entrenamiento de los conjuntos generados con el tamaño del conjunto es independiente del problema de clasificación analizado. En concreto, se muestra cómo para problemas de clasificación binarios, esta dependencia se puede analizar en términos de un proceso de Bernoulli. Finalmente, se muestran los resultados de experimentos realizados en 15 bases de datos estándar que demuestran las mejoras que se pueden obtener con este procedimiento.*

## 4.1. Introducción

En este capítulo presentamos una variante de los conjuntos de clasificadores *flipping* [Breiman, 2000], que pertenece a la categoría de los bosques aleatorios (*random forests*) cuando es utilizado junto con árboles de decisión [Breiman, 2001] y a la de los conjuntos que modifican las etiquetas de clases para obtener una cierta diversidad [Dietterich, 2000a]. En el trabajo de Breiman [Breiman, 2000], cada clasificador individual del conjunto se construye usando una alteración del conjunto original en la que las etiquetas de

clase se han modificado aleatoriamente: la clase de cada ejemplo se cambia con una probabilidad que depende de una tasa de modificación global, definida como la proporción media de ejemplos cuya etiqueta de clase es modificada, y de las proporciones de las distintas clases en el conjunto de datos original. Las probabilidades de modificación se eligen de forma que se mantenga la distribución original de las clases en el conjunto perturbado. En conjuntos compuestos de 100 clasificadores se obtienen tasas de error similares o ligeramente mejores que *bagging*. En este estudio hemos observado que si se usan conjuntos más grandes ($\approx 1000$ clasificadores) y tasas de modificación de las etiquetas de clase altas, se pueden alcanzar unas tasas de error mucho mejores, comparables o mejores que *boosting*. A diferencia del método presentado en [Breiman, 2000], nuestro método no requiere que se mantengan la distribución original de clases en los conjuntos modificados. Esto hace posible, como veremos en la siguiente sección, el uso de tasas de modificación global de etiquetas mayores para conjuntos con clases desequilibradas, lo que permite a su vez alcanzar mejores errores de generalización.

En la sección 4.2 de este capítulo se describe el algoritmo de construcción de conjuntos de clasificadores mediante la modificación de las etiquetas de clase; la sección 4.3 presenta un experimento sencillo que nos servirá para analizar en detalle el funcionamiento del algoritmo propuesto; la capacidad de clasificación del algoritmo se ha medido en 15 conjuntos de datos y se ha comparado con el algoritmo *flipping* propuesto por Breiman [Breiman, 2000], además de con *bagging* y *boosting* (sección 4.4); finalmente, se resumen las conclusiones de este capítulo.

## 4.2. Modificación de las etiquetas de clase

En [Breiman, 2000] se propone la generación de conjuntos de clasificadores mediante la modificación aleatoria de las etiquetas de clase de los ejemplos de entrada de acuerdo con la siguiente matriz de probabilidades

$$\begin{array}{l} P_{j\leftarrow i} = wP_j \quad \text{para } \ i \neq j \\ P_{i\leftarrow i} = 1 - w(1 - P_i) \ , \end{array} \tag{4.1}$$

donde $P_{j\leftarrow i}$ es la probabilidad de que un ejemplo con etiqueta $i$ pase a tener etiqueta $j$, $P_i$ es la proporción de elementos de clase $i$ en el conjuntos de entrenamiento, y $w$ es proporcional a la tasa de modificación global (fracción media de ejemplos modificados), $p$,

$$w = \frac{p}{1 - \sum_j P_j^2} = \frac{p}{2\sum_j \sum_{k>j} P_j P_k}. \tag{4.2}$$

Esta matriz de probabilidades, ec. (4.1), está definida de manera que las proporciones de clase se mantengan aproximadamente constantes en el conjunto modificado.

Para conseguir que este método funcione, el valor de la tasa de modificación global

$p$ debe ser menor que un cierto valor máximo de tal forma que el error de entrenamiento tienda a cero al incrementarse el número de clasificadores individuales que integran el conjunto. Obviamente, no se pueden alterar las etiquetas de todos los ejemplos, porque se perdería toda la información de clases y por tanto del problema. El valor máximo de $p$ depende tanto del número de clases como de las distribuciones de clases. En problemas de clasificación binaria, esta condición viene dada por

$$p < P_{min}, \tag{4.3}$$

donde $P_{min}$ es la proporción de ejemplos que pertenecen a la clase minoritaria. La desigualdad (4.3) asegura que, en promedio, la fracción de ejemplos modificados dentro de la clase minoritaria es menor que $1/2$. Tasas de modificación global por encima de este límite modificarían la etiqueta de más de la mitad de los ejemplos de la clase minoritaria. Como consecuencia, las regiones del espacio de características pertenecientes a la clase minoritaria se verían inundadas por ejemplos etiquetados como de clase mayoritaria y por tanto, estas regiones serían clasificadas de forma incorrecta por el conjunto.

Nuestra propuesta consiste en generar cada clasificador del conjunto de clasificadores usando una perturbación del conjunto de entrada. En cada conjunto de datos perturbado se modifica una fracción fija $p$ de los ejemplos del conjunto original, seleccionada aleatoriamente y sin tener en cuenta la clase del ejemplo. La etiqueta de clase de estos ejemplos se cambia a su vez aleatoriamente por otra clase existente y diferente. Esto define la siguiente matriz de probabilidades fija e independiente de la distribución de clases:

$$\begin{array}{l} P_{j \leftarrow i} = p/(K-1) \quad \text{para } \; i \neq j \\ P_{i \leftarrow i} = 1 - p \; , \end{array} \tag{4.4}$$

donde $K$ es el número de clases. Este procedimiento genera conjuntos de entrenamiento en los que la distribución de clases normalmente difiere de la distribución original del conjunto de entrenamiento. De hecho, la distribución de clases para conjuntos desequilibrados tiende a equilibrarse al incrementar $p$ en los conjuntos perturbados.

Para asegurar la convergencia del conjunto en el conjunto de entrenamiento debe haber para cada clase una mayoría de ejemplos correctamente etiquetados (no modificados). Esta condición se alcanza en el conjunto de entrenamiento (en promedio) si $P_{j \leftarrow i} < P_{i \leftarrow i}$ que de acuerdo con la ecuación (4.4) se cumple para

$$p < (K-1)/K, \tag{4.5}$$

independientemente de la distribución inicial de clases. De acuerdo con esta ecuación definimos el máximo valor de $p$ para el método propuesto como

$$p_{max} = (K-1)/K. \tag{4.6}$$

También resulta conveniente definir la proporción entre la tasa de modificación, $p$, y su máximo como

$$\hat{p} = p/p_{max}. \tag{4.7}$$

Por tanto, para conjuntos desequilibrados, el método propuesto incrementa el rango de posibles valores de $p$, con respecto al método de *flipping* [Breiman, 2000]. Este es un factor determinante para las mejoras de generalización que se obtienen con el conjunto, como veremos en la sección 4.4.

Para que el algoritmo funcione de forma eficiente es necesario utilizar un clasificador base que obtenga un error en entrenamiento lo más bajo posible. Hay que tener en cuenta que un clasificador que obtenga una clasificación perfecta (error 0) en el conjunto de entrenamiento modificado tendrá un error igual a la proporción de ejemplos modificados, $p$, en el conjunto de entrenamiento original. Un árbol de decisión sin podar y que ha sido desarrollado hasta su tamaño máximo de forma que todos los ejemplos del conjunto de entrenamiento estén perfectamente clasificados, cumple este requisito. De hecho, un árbol de decisión siempre es capaz de obtener error $0$ siempre que no haya en el conjunto alterado ejemplos con los mismos valores de los atributos que pertenezcan a clases distintas.

Una característica interesante del procedimiento *class-switching* es que la selección aleatoria de los ejemplos crea clasificadores cuyos errores en el conjunto de entrenamiento son independientes estadísticamente. Los clasificadores generados tienen error cero en el conjunto de entrenamiento modificado. Por tanto, su tasa de error en el conjunto de entrenamiento original es igual a la fracción de ejemplos cuyas etiquetas de clase han sido modificadas (esto es $p$). Estos ejemplos han sido elegidos aleatoriamente y de manera independiente para cada uno de los distintos conjuntos de entrenamiento. Basándonos en esta propiedad se puede estimar el error de entrenamiento sin necesidad de tener en cuenta de qué problema de clasificación concreto se trata. Para clasificación binaria el funcionamiento del conjunto se puede analizar como un proceso de Bernoulli, donde cada clasificador tiene una probabilidad $(1 - p)$ de clasificar correctamente un ejemplo de entrenamiento seleccionado al azar. La decisión de un clasificador dado sobre un ejemplo de entrenamiento es, por construcción, independiente de la decisión de los otros clasificadores. En consecuencia, la probabilidad de que haya un número determinado de clasificadores dando la clasificación correcta viene dada por una distribución binomial. Por tanto, el error de entrenamiento se puede estimar como la probabilidad de que haya más de la mitad de los clasificadores dando una clasificación incorrecta para un ejemplo dado

$$train\_error(T) = \sum_{t=\lfloor 1+T/2 \rfloor}^{T} \left( \begin{array}{c} T \\ t \end{array} \right) p^t (1-p)^{T-t} , \tag{4.8}$$

donde $T$ es el número de clasificadores del conjunto (que asumimos que es impar para evitar los empates). Basándonos en la distribución binomial también podemos estimar las

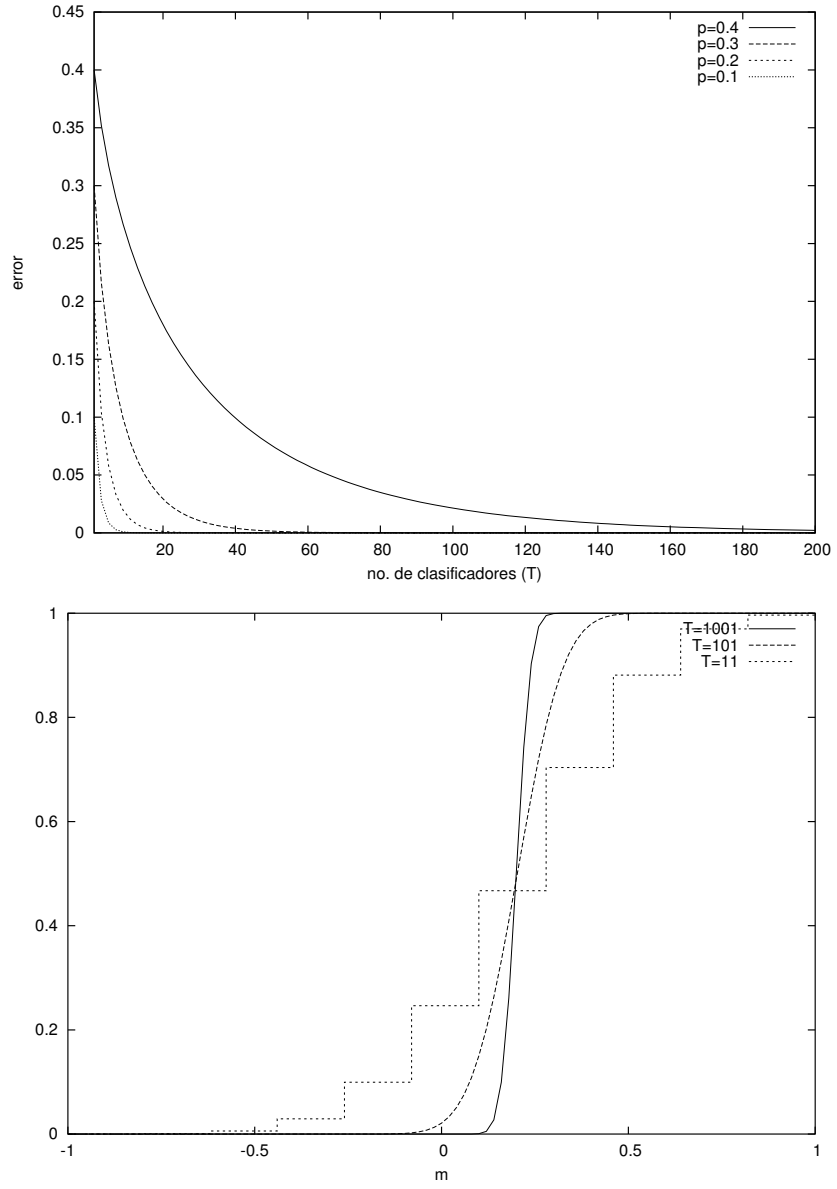



Figura 4.1: (*Gráfica superior*) Estimación del error de entrenamiento para un problema binario de clasificación con respecto al tamaño del conjunto con tasas de modificación de clases de: $p = 0.1$ (*línea punteada*), $p = 0.2$ (*línea de trazos cortos*), $p = 0.3$ (*línea de trazos largos*) y $p = 0.4$ (*línea continua*). (*Gráfica inferior*) Estimaciones de las curvas de margen para un problema binario de clasificación en conjuntos con tasa de modificación de clases de $p = 0.4$ para tamaños de conjunto de 11 (*línea de trazos cortos*), 101 (*línea de trazos largos*) y 1001 (*línea continua*) clasificadores

curvas de distribución acumulada de margen del conjunto *class-switching* en entrenamiento para un problema de dos clases:

$$train\_margin(m) = \sum_{t=0}^{\lfloor T(m+1)/2 \rfloor} \left( \begin{array}{c} T \\ t \end{array} \right) p^{T-t}(1-p)^t \ , \tag{4.9}$$

donde $m$ es el margen de clasificación, definido como la fracción de clasificadores correctos menos la fracción de erróneos para un problema de dos clases (ver [Schapire *et al.*, 1998] o sección 2.4.2). Para un ejemplo dado, el margen será igual a $-1$ cuando todos los miembros del conjunto están de acuerdo en una clasificación incorrecta y será igual a $1$ cuando todos los elementos están de acuerdo en la clase correcta.

Las curvas correspondientes a las ecs. (4.8) y (4.9) se muestran en las gráficas superior e inferior de la figura 4.1, respectivamente. En la figura 4.1 (gráfica superior) se muestra la evolución del error de entrenamiento con el número de clasificadores para diferentes valores de $p$ y para números impares de clasificadores. Hay que resaltar que todas las curvas tienden a $0$ ya que estamos considerando un problema binario de clasificación y los valores seleccionados de $p$ están por debajo de $0.5$. Estas gráficas muestran que para valores mayores de $p$ se necesitan más clasificadores para que converja el error de entrenamiento. La figura 4.1 (gráfica inferior) muestra el margen del conjunto para el conjunto de entrenamiento usando $p = 0.4$ y conjuntos de tamaños $11$, $101$ y $1001$ clasificadores, respectivamente. Se puede comprobar como todas las curvas están centradas en $m = 1 - 2p$ y que, al incrementar $T$ todos los ejemplos tienden a tener el mismo valor $1 - 2p$ del margen.

Las ecuaciones (4.8) y (4.9) y las gráficas de la figura 4.1 son válidas solamente para el conjunto de entrenamiento. Sin embargo, es de esperar que las características de estas curvas se reflejen también en los conjuntos de test. En concreto, el comportamiento del error de generalización dependerá del valor de $p$. Por un lado tenemos que el tamaño del conjunto necesario para alcanzar la convergencia será mayor con valores mayores de $p$. Sin embargo, y dado que el error en test es normalmente mayor que en entrenamiento, el umbral efectivo para $p$ deberá ser menor que el valor dado por la ec. (4.6). Por debajo de este valor de $p$ consideramos que se obtendrá una disminución del error de generalización con el número de clasificadores que será tanto más lenta cuanto más se acerque $p$ a esta umbral desde abajo.

## 4.3. Un experimento ilustrativo

Para entender mejor cómo funciona el conjunto *class-switching* en comparación con otros conjuntos de clasificadores como *bagging* y *boosting* hemos analizado en detalle el comportamiento de estos algoritmos en un problema sencillo de clasificación. El problema consiste en dos clases separables linealmente en un espacio de atributos bidimensional,

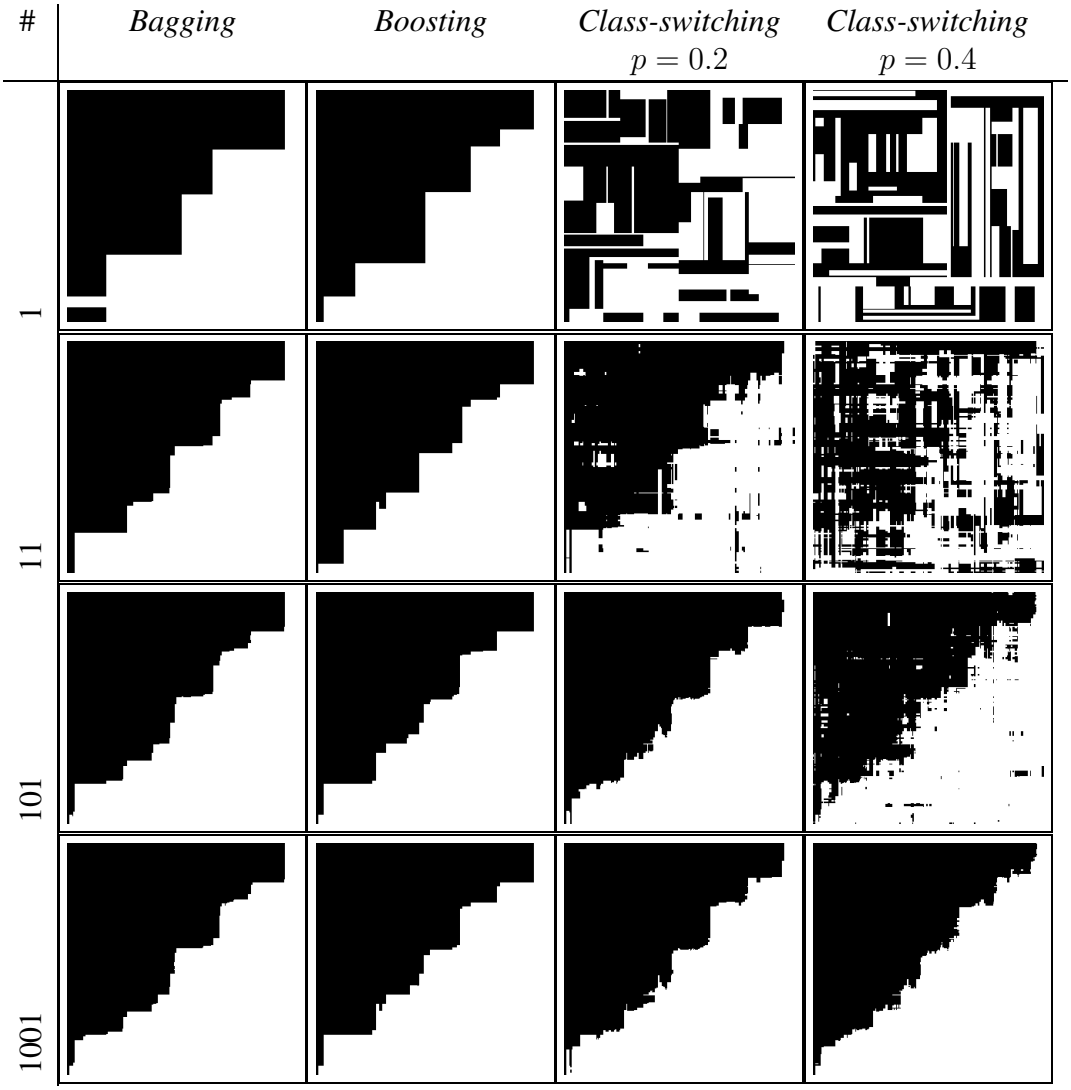



Figura 4.2: Mapa de clasificación para un problema perfectamente separable linealmente para *bagging*, *boosting* y conjuntos *class-switching* ($p = 0.2$ y $p = 0.4$). El número de árboles usados en los conjuntos se señala en la columna de la izquierda para cada línea (1, 11, 101 y 1001 árboles, de arriba a abajo)

donde las dos clases están separadas por la línea $y = x$. Esta frontera diagonal es difícil de representar para árboles de decisión como C4.5 o CART, que utilizan divisiones paralelas a los ejes del espacio de atributos. Esta limitación hace que los árboles de decisión encuentren fronteras de separación de baja calidad para este tipo de problemas.

El conjunto de entrenamiento consta de $300$ vectores aleatorios distribuidos al azar uniformemente en el cuadrado unidad ($x \sim U[0,1]$ y $y \sim U[0,1]$). Usando estos datos, se han construido los conjuntos *bagging*, *boosting* y *class-switching* ($p = 0.4$ y $p = 0.2$) de tamaño $1001$ usando árboles C4.5 como clasificador base. Los valores de los parámetros usados en la construcción de los árboles y conjuntos son los mismos que los de los experimentos descritos en la sección 4.4.

La eficacia de los distintos conjuntos se ha probado usando un conjunto de test compuesto de $300 \times 300$ puntos distribuídos en forma de rejilla regular en el cuadrado unidad. La figura 4.3 muestra los resultados de clasificación de estos puntos para varias etapas del proceso usando imágenes donde blanco y negro indican las dos clases del problema. La primera fila muestra los resultados usando un único clasificador y la última fila muestra los resultados cuando se usan los 1001 árboles generados. Se han usado siempre números impares de clasificadores para evitar los empates en los procesos de votación. Esta figura muestra que *bagging* y *boosting* convergen más rápidamente a su comportamiento de clasificación asintótico que los conjuntos *class-switching*. Inicialmente, el algoritmo *class-switching* tiene un error muy elevado. De hecho, cuando se usa un solo clasificador este conjunto muestra un patrón de clasificación que no tiene ningún parecido con el patrón buscado. Cuando se usa $p = 0.4$ como tasa de modificación y se utilizan $101$ clasificadores el espacio de características sigue sin estar correctamente separado. Se necesita un gran número de elementos ($\sim 1000$) para definir correctamente la frontera de clasificación o, al menos, para alcanzar el comportamiento asintótico del conjunto. Esto es coherente con la ecuación (4.8), y refuerza la conjetura de que es necesario utilizar muchos clasificadores para alcanzar el nivel asintótico de error. Pero a pesar de la lenta convergencia, la precisión final del conjunto es mejor que *bagging* o *boosting*. Más interesante que la precisión del conjunto es el perfil de la frontera final alcanzado. *Bagging* y *boosting* generan fronteras de clasificación que tienen un gran parecido con aquéllas obtenidas por C4.5. Los conjuntos *class-switching* generan una frontera de decisión mucho más compleja, cuya forma es muy distinta a las generadas por C4.5. Esto sugiere que el algoritmo *class-switching* puede obtener reducciones significativas del sesgo de clasificación del algoritmo base.

El origen de las diferencias en la complejidad de las fronteras puede ser explicado por el hecho de que *bagging* y *boosting* plantean diferentes problemas de clasificación al algoritmo base que los conjuntos *class-switching*. Para cada elemento del conjunto, *bagging* y *boosting* generan un problema de clasificación que tiene una relación clara con el problema original. De hecho, cada uno de los clasificadores de un conjunto *bagging* es una solución razonablemente buena del problema. *Boosting* es distinto de *bagging* en este aspecto pero aun así genera problemas que están muy relacionados con el problema original. De hecho, a medida que crece el tamaño del conjunto *boosting*, también lo hace el peso de los ejemplos más veces mal clasificados. Por tanto, el clasificador base tiende a centrarse en resolver las partes más complicadas del problema. Sin embargo, el algoritmo *class-switching* genera

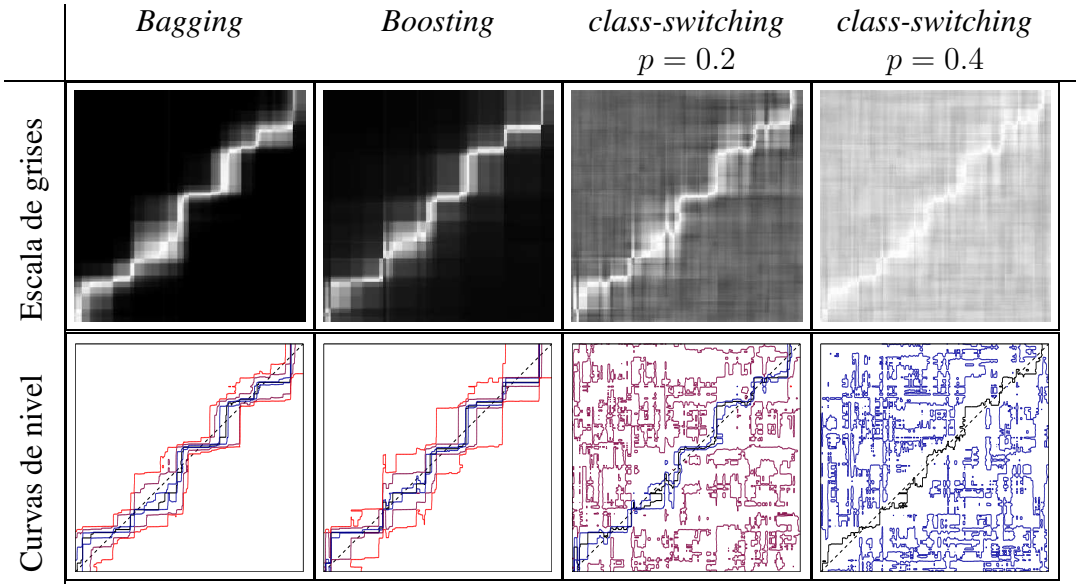



Figura 4.3: Mapa del margen para un problema separable linealmente para *bagging*, *boosting* y conjuntos *class-switching* ($p = 0.2$ y $p = 0.4$) usando $1001$ clasificadores (más detalles en el texto)

sustitutos del problema muy distintos del problema original (especialmente para valores altos de $p$), y cuyo parecido con el problema es solamente estadística: la frontera de decisión se perfila solo de forma asintótica al aumentar el número de clasificadores y sólo cuando se alcanza un número alto de los mismos.

La figura 4.3 muestra el mapa del margen de la decisión final ($T = 1001$) para los distintos conjuntos. En este análisis el margen está definido como diferencia de votos (ponderada o no) entre la clase más votada y la segunda más votada en vez de usar el margen basado en la diferencia entre la clase correcta y la incorrecta, ver [Schapire *et al.*, 1998] o sección 2.4.2 para más detalles sobre el margen. La primera fila representa el valor del margen usando una imagen con paleta invertida de grises, donde grises más claros indican valores menores de margen. La segunda fila muestra los mismos valores de margen usando un mapa de curvas de nivel, donde cada línea representa posiciones en el espacio de atributos con mismos valores de margen. La frontera real del problema (la diagonal $y = x$) también se ha marcado con una línea discontinua a trazos. Estos mapas muestran las curvas de nivel para márgenes: $0$ (la decisión del conjunto, marcado por líneas más obscuras en los mapas), y $0.2$, $0.6$ y $0.8$. En los conjuntos *bagging* y *boosting*, las curvas de nivel para los valores $0.2$, $0.6$ y $0.8$ aparecen en parejas (una línea por clase) en posiciones más alejadas de la frontera de decisión cuanto mayor es el valor del margen. En el conjunto *class-switching* con $p = 0.2$ aparecen la frontera de decisión y los pares de las curvas de

nivel con valores $0.2$ y $0.6$. La pareja de líneas para el valor de margen $0.2$ aparece muy cerca de la frontera de decisión, mientras que las curvas de nivel de valor $0.6$ tienen una estructura compleja y llenan practicamente todo el espacio de características. Para el conjunto *class-switching* con $p = 0.4$ sólo aparecen las curvas de nivel para márgenes de $0$ (frontera de decisión) y $0.2$. Para este conjunto no existen puntos donde el margen sea $0.6$ o mayor. Las diferencias de los mapas de margen entre los distintos conjuntos es clara en esta figura: *bagging* genera amplias zonas contiguas donde todos los clasificadores están de acuerdo y un borde de incertidumbre bastante estrecho. *Boosting* genera una frontera más precisa a costa de márgenes menores en las zonas adyacentes a la frontera (el borde de incertidumbre es mayor). Ambos conjuntos, *bagging* y *boosting* tienen progresivamente valores más altos del margen cuando nos desplazamos a regiones más alejadas de la frontera de decisión. El algoritmo *class-switching* genera una distribución de puntos con el mismo margen más deslocalizada. Asintóticamente, según crece el número de clasificadores del conjunto, el mapa de margen consiste en amplias mesetas con un valor constante $\approx 1 - 2p$ de margen, separado por zonas estrechas de margen a lo largo de la frontera de decisión con valores menores de margen.

## 4.4. Experimentos en conjuntos UCI

Para evaluar las mejoras de clasificación que se pueden obtener usando los conjuntos *class-switching*, hemos comparado la precisión de este método con C4.5, el método *flipping* de Breiman [Breiman, 2000], *boosting* y *bagging*. Todos los conjuntos se han generado usando C4.5 Release 8 como algoritmo base. Para C4.5, *bagging* y *boosting* basados en C4.5 se han usado las opciones por omisión del paquete C4.5. Para construir los conjuntos *class-switching* y *flipping* no hemos utilizado el término de penalización basado en el principio de longitud de descripción mínima (*Minimum Description Length*) que se aplica a los atributos cuantitativos. C4.5 con este término de penalización generalmente construye árboles mejores y más pequeños [Quinlan, 1996b]. Sin embargo, el uso de este criterio detiene la construcción del árbol C4.5 antes de que se obtengan todos los nodos hojas puros. Por tanto, no cumple el requisito para los conjuntos *class-switching*, que consiste en tener error cero (o aproximadamente cero) en el conjunto de entrenamiento. También hemos fijado el número mínimo de ejemplos por nodo hoja a 1 y se usan árboles desarrollados completamente (sin poda). Esta configuración es similar a la implementación de Breiman de los conjuntos *flipping* [Breiman, 2000] en la que usaba árboles CART sin podar [Breiman *et al.*, 1984].

La variante de *boosting* implementada está descrita en la sección 2.6.1 y es básicamente la misma utilizada en [Webb, 2000]. Se ha utilizado reponderación (en vez de remuestreo), como se sugiere en [Quinlan, 1996a]. Esto permite que todos los ejemplos estén incluidos en la construcción de cada elemento del conjunto. El peso mínimo para un ejemplo se ha

Cuadro 4.1: Características de los problemas utilizados

| Problema | Entrenamiento | Test | Atributos | Clases | Distribución de clases |
|---|---|---|---|---|---|
| Australian | 500 | 190 | 14 | 2 | 383/307 |
| Breast W. | 500 | 199 | 9 | 2 | 458/241 |
| Diabetes | 468 | 300 | 8 | 2 | 500/268 |
| German | 600 | 400 | 20 | 2 | 700/300 |
| Heart | 170 | 100 | 13 | 2 | 150/120 |
| Horse-Colic | 244 | 124 | 21 | 2 | 232/136 |
| Ionosphere | 234 | 117 | 34 | 2 | 225/126 |
| New-thyroid | 140 | 75 | 5 | 3 | 150/35/30 |
| Segment | 210 | 2100 | 19 | 7 | *uniforme* |
| Threenorm | 300 | 5000 | 20 | 2 | *uniforme* |
| Tic-tac-toe | 600 | 358 | 9 | 2 | 626/332 |
| Twonorm | 300 | 5000 | 20 | 2 | *uniforme* |
| Vowel | 600 | 390 | 10 | 11 | *uniforme* |
| Waveform | 300 | 5000 | 21 | 3 | *uniforme* |
| Wine | 100 | 78 | 13 | 3 | 71/59/48 |

marcado en $10^{-8}$ para evitar problemas numéricos de agotamiento (*underflow*). Asimismo el proceso de *boosting* no se para cuando un aprendiz alcanza un error mayor o igual que $0.5$ o igual a $0$. En estos casos el conjunto de entrenamiento se substituye por un muestreo bootstrap del conjunto original con todos los pesos asignados a $1/N$. En estos casos, el último clasificador se descarta si su error es mayor o igual a $0.5$ o se mantiene en el conjunto con un peso igual a $\ln(10^{10})$ —equivalente al de un clasificador con un error muy pequeño ($\approx 10^{-10}$)— si su error es igual a $0$. En cinco de los problemas estudiados esta última modificación produjo algunas diferencias que siempre condujeron a incrementos en promedio del error de generalización.

Se han probado los algoritmos implementados en 15 problemas de aprendizaje automático. Tres de ellos son problemas sintéticos: (*Threenorm, Twonorm* y *Waveform*) propuestos en las referencias [Breiman, 1996b; Breiman *et al.*, 1984]. El resto de problemas están incluidos en la colección de problemas de UCI [Blake y Merz, 1998]: *Australian Credit, Breast Cancer Wisconsin, Pima Indian Diabetes, German Credit, Heart, Horse Colic, Ionosphere, New-Thyroid, Image Segmentation, Tic-tac-toe, Vowel* y *Wine*. Las bases de datos han sido elegidas de forma que haya problemas de una gran variedad de campos de aplicación, así como conjuntos sintéticos, conjuntos con diferente número de clases y atributos, etc. En el cuadro 4.1 se muestra, para cada base de datos, el número de ejemplos usados para entrenamiento y test, el número de atributos, el número de clases y el número de ejemplos por clase. La proporción usada para entrenamiento es aproximadamente $2/3$

del número total de ejemplos excepto para los conjuntos sintéticos y para el conjunto *Image Segmentation*. En este último se han usado las particiones definidas en su documentación. Para más detalles sobre los distintos conjuntos ver apéndice A.

Para cada conjunto se han llevado a cabo $100$ ejecuciones. Cada ejecución incluye los siguientes pasos:

1. Generación de una partición estratificada de los datos de entrada en entrenamiento y test para los conjuntos reales y un muestreo aleatorio para los conjuntos sintéticos (ver cuadro 4.1 para ver los tamaños utilizados).

2. Construcción de un árbol C4.5, y conjuntos de $1000$ árboles usando: *class-switching* y *flipping* (con los siguientes valores de $\hat{p}$: $1/5$, $2/5$, $3/5$ y $4/5$), *boosting* y *bagging*.

3. Cálculo del error de los clasificadores en el conjunto de test para obtener una estimación del error de generalización.

En total estos experimentos han involucrado 100 ejecuciones por cada una de las 15 bases de datos. En cada base de datos se han aplicado 10 configuraciones de conjuntos de clasificadores diferentes. Cada conjunto generado está compuesto por 1000 árboles. Esto hace que se hayan generado un total de $15$ millones de árboles de decisión para este experimento.

El cuadro 4.2 presenta los resultados para el promedio del error de test obtenido por C4.5 y los distintos conjuntos de clasificadores usando $1000$ árboles. El menor error alcanzado para cada problema se ha marcado en negrita y el segundo mejor se ha subrayado. La desviación estándar se muestra solamente para C4.5. Excepto en algunos casos (marcados en cursiva en el cuadro), las desviaciones estándar de los conjuntos son menores que las mostradas para el árbol C4.5. En resumen podemos decir que: el conjunto *class-switching* obtiene 10 mejores resultados en 9 conjuntos (2 con $\hat{p} = 4/5$, 6 con $\hat{p} = 3/5$ y dos con $\hat{p} = 2/5$); *flipping* obtiene el mejor resultado en $4$ problemas ($2 \times \hat{p} = 3/5$ y $2 \times \hat{p} = 2/5$); *boosting* devuelve el mejor resultado en los conjuntos sintéticos *Threenorm* y *Twonorm* y en el *Tic-tac-toe* y *bagging* es el mejor en dos conjuntos considerados difíciles como son: *Pima Indian Diabetes* y *Heart*.

En el cuadro 4.3 se muestra un cuadro resumen del funcionamiento global de los algoritmos analizados. Esto se muestra como registros *victorias/empates/derrotas*, donde el (primer / segundo / tercer) número mostrado en cada celda corresponde al número de conjuntos en los que el algoritmo mostrado en la columna de la izquierda (gana / empata / pierde) con respecto al algoritmo mostrado en la primera fila. Para cada columna, se ha resaltado el registro con mayor número de $(victorias - derrotas)$, siempre que sea positivo. En este cuadro podemos ver que el único algoritmo que es mejor que todos los demás es *class-switching* junto con $\hat{p} = 3/5$. Además, *class-switching* con $\hat{p} = 3/5$ y $\hat{p} = 2/5$ son las dos únicas configuraciones que mejoran los resultados de *boosting*.

Cuadro 4.2: Error medio de test (en %) usando C4.5, y 1000 clasificadores para: *class-switching*, *flipping*, *boosting* y *bagging*. El mejor resultado para cada problema se ha resaltado en **negrita**. El segundo mejor se ha subrayado. Promedios con una desviación estándar mayor que la mostrada para C4.5 se muestran en *cursiva*

| | *C4.5* | *class-switching* ($\hat{p}=$) | | | | *flipping* ($\hat{p}=$) | | | | *boosting* | *bagging* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $4/5$ | $3/5$ | $2/5$ | $1/5$ | $4/5$ | $3/5$ | $2/5$ | $1/5$ | | |
| Australian | 14.3±2.2 | *14.8* | **13.0** | **13.0** | 13.5 | *20.8* | 13.6 | **13.0** | 13.5 | 13.4 | 13.3 |
| Breast W. | 5.4±1.4 | **3.0** | 3.1 | 3.1 | 3.6 | *34.4* | *7.1* | 3.8 | 3.8 | 3.2 | 3.9 |
| Diabetes | 27.0±2.6 | 25.7 | 25.6 | 25.4 | 25.8 | 34.9 | 29.2 | 26.2 | 25.7 | 26.1 | **24.6** |
| German | 28.9±2.2 | 26.7 | **25.0** | 25.1 | 26.8 | 30.0 | *29.9* | 26.7 | 26.3 | 25.5 | 25.7 |
| Heart | 23.6±3.5 | 22.4 | 21.2 | 21.7 | *22.8* | *29.0* | 22.1 | 21.8 | *23.0* | 19.5 | **19.1** |
| Horse-colic | 15.9±2.9 | 15.8 | 16.1 | 16.0 | 15.8 | 36.7 | 18.4 | **15.3** | 15.6 | 17.1 | 16.0 |
| Ionosphere | 10.9±2.8 | 8.1 | 6.9 | **6.2** | 6.3 | 35.9 | *18.7* | 7.0 | 6.3 | 6.4 | 7.5 |
| New-thyroid | 8.4±3.1 | **3.9** | 4.0 | 4.2 | 5.1 | 30.2 | 30.3 | *10.8* | 4.5 | 5.7 | *6.1* |
| Segment | 10.3±1.4 | 7.6 | **5.5** | 5.7 | 7.0 | 7.5 | **5.5** | 5.7 | 7.1 | 6.5 | 8.1 |
| Threenorm | 31.7±1.2 | 18.7 | 17.7 | 18.2 | *19.9* | 18.7 | 17.7 | 18.2 | *20.0* | **15.7** | 19.1 |
| Tic-tac-toe | 17.3±2.3 | 6.7 | 3.4 | 3.9 | 6.3 | 34.8 | 19.1 | 6.5 | 6.2 | **1.2** | 8.9 |
| Twonorm | 21.6±0.7 | 4.6 | 3.8 | 4.0 | *5.5* | 4.6 | 3.8 | 4.0 | *5.6* | **3.7** | *6.6* |
| Vowel | 26.5±2.4 | 4.9 | **4.7** | 6.1 | 8.4 | 5.0 | **4.7** | 6.0 | 8.4 | 7.5 | 13.2 |
| Waveform | 29.0±1.3 | 19.2 | **16.9** | 17.3 | 19.3 | 22.5 | 17.5 | 17.6 | 19.4 | 17.4 | 19.4 |
| Wine | 9.2±4.0 | 2.6 | **1.2** | 1.8 | 3.1 | *7.7* | 1.5 | 1.5 | 3.0 | 4.1 | 6.4 |

Cuadro 4.3: Resumen de registros victoria/empate/derrota. Para cada columna se ha resaltado en **negrita** el registros con mayor $(victorias - derrotas)$ (siempre que sea positivo)

| | | *C4.5* | *class-switching* ($\hat{p} =$) | | | | *flipping* ($\hat{p} =$) | | | | *boosting* | *bagging* |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | $4/5$ | $3/5$ | $2/5$ | $1/5$ | $4/5$ | $3/5$ | $2/5$ | $1/5$ | | |
| *C4.5* | | X | 1/0/14 | 1/0/14 | 1/0/14 | 0/0/15 | 9/0/6 | 7/0/8 | 1/0/14 | 0/0/15 | 1/0/14 | 1/0/14 |
| *switching* | $\hat{p} = 4/5$ | 14/0/1 | X | 3/0/12 | 4/0/11 | 10/1/4 | 12/2/1 | 7/0/8 | 4/1/10 | 8/1/6 | 6/0/9 | 10/0/5 |
| | $\hat{p} = 3/5$ | 14/0/1 | **12/0/3** | X | **10/2/3** | 13/0/2 | **15/0/0** | **11/4/0** | **13/1/1** | 13/0/2 | 10/0/5 | 12/0/3 |
| | $\hat{p} = 2/5$ | 14/0/1 | 11/0/4 | 3/2/10 | X | **14/0/1** | 14/0/1 | 10/0/5 | 8/4/3 | **14/0/1** | **11/0/4** | **12/1/2** |
| | $\hat{p} = 1/5$ | **15/0/0** | 4/1/10 | 2/0/13 | 1/0/14 | X | 12/0/3 | 8/0/7 | 5/0/10 | 6/3/6 | 5/0/10 | 10/0/5 |
| *flipping* | $\hat{p} = 4/5$ | 6/0/9 | 1/2/12 | 0/0/15 | 1/0/14 | 3/0/12 | X | 1/0/14 | 1/0/14 | 3/0/12 | 1/0/14 | 4/0/11 |
| | $\hat{p} = 3/5$ | 8/0/7 | 8/0/7 | 0/4/11 | 5/0/10 | 7/0/8 | 14/0/1 | X | 5/1/9 | 7/0/8 | 3/0/12 | 6/0/9 |
| | $\hat{p} = 2/5$ | 14/0/1 | 10/1/4 | 1/1/13 | 3/4/8 | 10/0/5 | 14/0/1 | 9/1/5 | X | 9/1/5 | 5/0/10 | 11/0/4 |
| | $\hat{p} = 1/5$ | **15/0/0** | 6/1/8 | 2/0/13 | 1/0/14 | 6/3/6 | 12/0/3 | 8/0/7 | 5/1/9 | X | 5/0/10 | 9/1/5 |
| *boosting* | | 14/0/1 | 9/0/6 | 5/0/10 | 4/0/11 | 10/0/5 | 14/0/1 | 12/0/3 | 10/0/5 | 10/0/5 | X | 11/0/4 |
| *bagging* | | 14/0/1 | 5/0/10 | 3/0/12 | 2/1/12 | 5/0/10 | 11/0/4 | 9/0/6 | 4/0/11 | 5/1/9 | 4/0/11 | X |

Cuadro 4.4: Prueba-t para comparar *class-switching* ($\hat{p} = 3/5$) con respecto a las otras configuraciones analizadas. Se ha resaltado en **negrita** los valores-p$< 0.005$. Los valores recuadrados corresponden a resultados desfavorables a *class-switching* ($\hat{p} = 3/5$)

| | *C4.5* | *class-switching* ($\hat{p}=$) | | | | *flipping* ($\hat{p}=$) | | | | *boosting* | *bagging* |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $4/5$ | $3/5$ | $2/5$ | $1/5$ | $4/5$ | $3/5$ | $2/5$ | $1/5$ | | |
| Australian | **3e-10** | **3e-16** | | 0.92 | **4e-4** | **1e-52** | **7e-5** | 0.52 | **4e-3** | 0.06 | 0.05 |
| Breast W. | **1e-34** | 0.83 | | 0.08 | **1e-10** | **8e-145** | **6e-42** | **1e-15** | **2e-16** | 0.07 | **7e-18** |
| Diabetes | **4e-6** | 0.22 | | 0.20 | 0.20 | **9e-67** | **2e-25** | **3e-4** | 0.48 | **6e-4** | **2e-8** |
| German | **1e-29** | **5e-21** | | 0.60 | **6e-15** | **6e-55** | **4e-54** | **3e-20** | **1e-10** | **1e-3** | **6e-4** |
| Heart | **4e-8** | **5e-5** | | 0.01 | **4e-7** | **7e-37** | **2e-3** | 0.01 | **6e-8** | **4e-7** | **2e-9** |
| Horse-Colic | 0.49 | 0.12 | | 0.54 | 0.18 | **3e-92** | **7e-10** | **6e-7** | 0.007 | **6e-5** | 0.80 |
| Ionosphere | **6e-24** | **4e-9** | | **6e-6** | **3e-4** | **3e-118** | **2e-59** | 0.31 | **2e-4** | **8e-4** | **4e-4** |
| New-thyroid | **4e-29** | 0.44 | | 0.17 | **3e-7** | **1e-7** | **2e-7** | **1e-33** | 0.02 | **3e-11** | **1e-11** |
| Segment | **6e-53** | **2e-44** | | 0.01 | **3e-29** | **5e-44** | 0.22 | **3e-3** | **1e-29** | **8e-19** | **3e-44** |
| Threenorm | **1e-99** | **2e-32** | | **6e-14** | **4e-36** | **4e-32** | 0.43 | **9e-13** | **4e-38** | **4e-51** | **6e-17** |
| Tic-tac-toe | **3e-81** | **8e-49** | | **2e-7** | **6e-30** | **1e-152** | **6e-99** | **1e-44** | **1e-30** | **1e-42** | **1e-53** |
| Twonorm | **8e-142** | **3e-65** | | **2e-6** | **3e-32** | **9e-61** | 0.98 | **3e-8** | **2e-33** | **7e-11** | **1e-34** |
| Vowel | **6e-93** | 0.007 | | **4e-28** | **9e-46** | **1e-6** | 0.79 | **3e-28** | **5e-46** | **2e-41** | **3e-65** |
| Waveform | **2e-93** | **1e-62** | | **5e-12** | **4e-40** | **4e-30** | **1e-12** | **3e-19** | **5e-40** | **2e-9** | **1e-36** |
| Wine | **2e-37** | **6e-12** | | **5e-5** | **4e-14** | **1e-27** | 0.04 | 0.04 | **1e-13** | **3e-18** | **4e-26** |

Se ha utilizado la prueba-t de Student pareada de dos colas para analizar las diferencias que obtiene el mejor algoritmo (*class-switching* $\hat{p} = 3/5$) con respecto al resto de algoritmos y configuraciones. Estos resultados se muestran en el cuadro 4.4. Se han resaltado en negrita las diferencias estadísticamente significativas (valor-p$< 0.5\,\%$). Asimismo se han recuadrado los resultados que son desfavorables a *class-switching* $\hat{p} = 3/5$ ya sean significativos o no. Se puede observar que las diferencias entre el error que obtiene *class-switching* $\hat{p} = 3/5$ y los errores de los otros métodos son significativas en la mayoría de los casos. *Class-switching* y *flipping* con $\hat{p} = 3/5$ obtienen resultados equivalentes en la mayoría de bases de datos con distribución de clases uniforme, concretamente para: *Image Segmentation, Threenorm, Twonorm* y *Vowel*. En el problema*Waveform*, que también tiene distribución de clases uniforme, los resultados son significativamente mejores para *class-switching*. En el resto de bases de datos las comparaciones de *class-switching* $\hat{p} = 3/5$ y *flipping* favorecen generalmente al primero. Con respecto a *bagging* el algoritmo propuesto obtiene diferencias significativas favorables en 11 problemas, desfavorables en 2 y no significativas en otras 2 bases de datos. Las diferencias más exiguas se obtiene con respecto a *boosting* donde *class-switching* $\hat{p} = 3/5$ obtiene diferencias significativas favorables en 8 problemas, desfavorables en 5 y en 2 problemas los resultados son equivalentes.

La figura 4.4 muestra la dependencia del error medio de entrenamiento (gráfica superior) y test (gráfica inferior) con el tamaño del conjunto de los conjuntos *class-switching* para distintos valores de $\hat{p}$ y en el problema *Breast Cancer Wisconsin*. Se puede ver cómo las curvas de error de entrenamiento (gráfica superior) son muy similares a las de la figura 4.1, y confirman el análisis del error de entrenamiento basado en la ecuación (4.8). Es necesario insistir en que la similitud no es exacta: las curvas de error para el conjunto *Breast Cancer Wisconsin* no comienzan para un clasificador en el valor de $p$ como era de esperar. Esto se debe a que el conjunto *Breast Cancer Wisconsin* tiene varios ejemplos con los mismos valores en los atributos que no se pueden separar si el algoritmo *class-switching* cambia la clase de alguno de ellos.

El cuadro 4.2 y la figura 4.4 confirman que la convergencia del error tanto de entrenamiento como de test en los conjuntos *class-switching* está relacionada con $\hat{p}$, la relación entre la probabilidad de modificación global y el máximo valor posible de modificación global, definido en la ecuación (4.7): valores más altos de $\hat{p}$ presentan una convergencia a los niveles asintóticos de error más lenta. En el conjunto *Breast Cancer Wisconsin*, por ejemplo, el conjunto *class-switching* con $\hat{p} = 4/5$ obtuvo el mejor resultado, pero necesitó $200$, $400$ y $800$ clasificadores para alcanzar tasas de error equivalentes a *bagging*, conjunto *class-switching* con $\hat{p} = 1/5$ y *boosting*, respectivamente. Alguna mejora adicional se puede obtener si se añaden más clasificadores (ver gráfica inferior de la figura 4.4). En otros conjuntos (*Threenorm, Tic-tac-toe* y *Twonorm*) el conjunto *class-switching* con $\hat{p} = 4/5$ puede alcanzar mejores precisiones si se combinan más clasificadores. En *Twonorm* se alcanza un error de $3.8$ usando $2000$ árboles (obtiene $4.6$ con $1000$ árboles) y en *Tic-tac-toe* se llega a $4.9$ de error con $5000$ árboles (obtiene $6.7$ con $1000$ árboles).

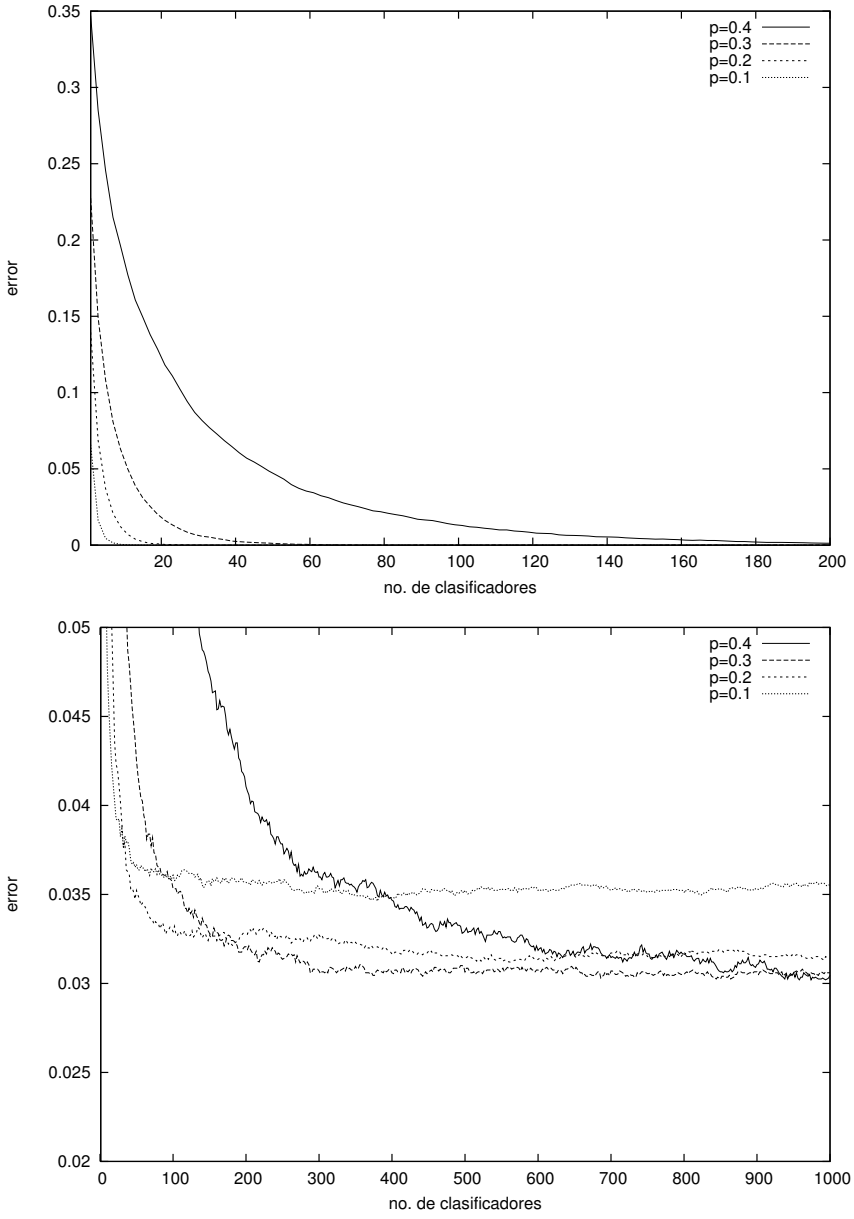



Figura 4.4: Error medio de entrenamiento (*gráfica superior*) y test (*gráfica inferior*) para el problema *Breast Cancer Wisconsin*

Asimismo se debe resaltar cómo la precisión en la clasificación está también relacionada con $\hat{p}$. Del cuadro 4.2 y la figura 4.4 vemos que valores más altos de $\hat{p}$ tienden a obtener mejores resultados de generalización. Sin embargo, cuando se usan valores de $\hat{p}$ cercanos a

Cuadro 4.5: Número medio de clasificadores base (en %) con un error en test mayor de $p_{max}$

| | *class-switching* | |
|---|---|---|
| **Problema** | $\hat{p} = 4/5$ | $\hat{p} = 3/5$ |
| Australian | 9.2 | 0.1 |
| Breast W. | 0.3 | 0.0 |
| Diabetes | 12.1 | 1.0 |
| German | 12.9 | 1.1 |
| Heart | 21.4 | 6.2 |
| Horse-Colic | 9.4 | 0.2 |
| Ionosphere | 12.1 | 0.5 |
| New-thyroid | 4.2 | 0.0 |
| Segment | 0.0 | 0.0 |
| Threenorm | 3.5 | 0.0 |
| Tic-tac-toe | 5.2 | 0.0 |
| Twonorm | 0.2 | 0.0 |
| Vowel | 0.0 | 0.0 |
| Waveform | 0.1 | 0.0 |
| Wine | 5.3 | 0.1 |

1 ($\hat{p} = 4/5$), los errores de *class-switching* son generalmente peores que cuando se utiliza $\hat{p} = 3/5$. Esto se puede explicar a partir del error de generalización de los clasificadores individuales creados. En el cuadro 4.5 se presenta el número medio de clasificadores que tienen errores de generalización mayores de $p_{max}$ (clasificadores con error mayor que un clasificador aleatorio que generalmente empeoran el funcionamiento del conjunto. Ver sección 2.4 para más detalles sobre cómo contribuyen los clasificadores individuales al conjunto) para conjuntos *class-switching* con $\hat{p} = 3/5$ y $\hat{p} = 4/5$. En este cuadro se puede observar cómo en el conjunto *German Credit*, los conjuntos *class-switching* con $\hat{p} = 4/5$ tienen en media un $12.9\,\%$ de los clasificadores con un error por encima de $0.5$ (con un error final del conjunto de $26.7$), mientras que con $\hat{p} = 3/5$ este valor se reduce a un $1.1\,\%$ (obteniendo un error de generalización de $25.0$).

Los conjuntos *flipping* y *class-switching* obtienen resultados muy similares de generalización en bases de datos donde las clases están equilibradas. Sin embargo, y tal como se esperaba de la ecuación (4.3), el uso de *flipping* con valores de $p$ que están por encima de la proporción de la clase minoritaria genera una mayoría de ejemplos etiquetados con la clase mayoritaria dentro de la región del espacio de atributos donde se sitúan los ejemplos de la clase minoritaria. Esto implica que los algoritmos individuales tienden a etiquetar estas regiones incorrectamente. En el cuadro 4.2 este efecto se puede observar para $P_{min} \approx p$

Cuadro 4.6: Error medio de test (en %) para *Threenorm* usando conjuntos desequilibrados para los algoritmos *class-switching*/*flipping*

| | $\hat{p} = 4/5$ | $\hat{p} = 3/5$ | $\hat{p} = 2/5$ | $\hat{p} = 1/5$ |
|---|---|---|---|---|
| $P_{min} = 0.5$ | 18.7/18.7 | **17.7/17.7** | 18.2/18.2 | 19.9/20.0 |
| $P_{min} = 0.4$ | 18.9/37.8 | **17.9**/23.8 | **17.9**/19.8 | 19.5/19.6 |
| $P_{min} = 0.3$ | 18.0/30.0 | **17.1**/29.6 | 17.3/22.7 | 18.0/19.1 |
| $P_{min} = 0.2$ | 15.1/20.0 | 14.6/20.0 | **14.4**/19.9 | 14.9/17.0 |
| $P_{min} = 0.1$ | 9.7/10.0 | **9.6**/10.0 | **9.6**/10.0 | **9.6**/10.0 |

y $P_{min} \leq p$. En esos casos el conjunto clasifica todo el espacio de atributos como de clase mayoritaria, obteniendo un error en test igual a la proporción de ejemplos de la clase minoritaria.

Dado que las mayores diferencias entre el método propuesto y el algoritmo de Breiman (*flipping*) se dan principalmente para conjuntos desequilibrados, hemos realizado una comparación en detalle para estos dos algoritmos en el conjunto sintético *Threenorm*. Hemos seleccionado el problema *Threenorm* porque los resultados obtenidos por ambos algoritmos son muy similares cuando se usan conjuntos con aproximadamente el mismo número de ejemplos de las dos clases. Además, el uso de un conjunto sintético nos permite modificar las probabilidades a priori de las clases al generar los conjuntos de entrenamiento y test. Se han probado ambos algoritmos usando valores de $P_{min}$ (esto es, la fracción de ejemplos que pertenecen a la clase minoritaria) de: $0.4$, $0.3$, $0.2$ y $0.1$. Para cada valor de $P_{min}$ hemos creado 10 conjuntos de entrenamiento compuestos de 300 ejemplos y un único conjunto de test de 5000 ejemplos con las mismas proporciones de clases. El error medio para ambos algoritmos en los conjuntos de test para distintos valores de $\hat{p}$ y $P_{min}$ se muestran en el cuadro 4.6, junto con los resultados para los conjuntos equilibrados ($P_{min} = 0.5$) del cuadro 4.2 como referencia. Estos resultados ponen en evidencia que al reducir $P_{min}$ el rango de posibles valores de $p$ para el algoritmo *flipping* se reduce, así como su capacidad de generalización. *Flipping* y *class-switching* obtienen resultados similares para el conjunto equilibrado. Sin embargo, *flipping* es notablemente peor que *class-switching* ($1.9$ puntos porcentuales peor) cuando se usan conjuntos ligeramente desequilibrados ($P_{min} = 0.4$), considerando el mejor resultado para cada algoritmo dentro de los valores de $\hat{p}$ utilizados. Estas diferencias se incrementan para $P_{min} = 0.3$ y $P_{min} = 0.2$ a $2.0$ y $2.4$ puntos porcentuales respectivamente. Para $P_{min} = 0.1$ la diferencia se reduce a $0.4$ puntos. Sin embargo, para esta última configuración ninguno de los dos algoritmos obtiene buenos resultados. Se trata de un problema con distribuciones muy descompensadas (sólo se usan 30 ejemplos de la clase minoritaria) que se debe abordar con otras técnicas de clasificación [Cantador y Dorronsoro, 2004].

## 4.5. Conclusiones

La modificación aleatoria de las etiquetas de clase de los ejemplos de entrenamiento es un procedimiento útil para generar conjuntos de clasificadores que: obtienen errores de generalización significativamente mejores que *bagging* y cuya eficacia es comparable o mejor que *boosting* en varios problemas de clasificación de la colección de problemas de UCI y problemas de clasificación sintéticos. Estas mejoras de clasificación se alcanzan para tasas relativamente altas de modificación de etiquetas de clases y para conjuntos con un gran número de clasificadores.

La modificación aleatoria de las salidas como método de generación de conjuntos de clasificadores fue propuesta inicialmente en [Breiman, 2000]. En esta referencia, los experimentos de clasificación fueron realizados con conjuntos de $100$ clasificadores, que son demasiado pequeños para que se ponga de manifiesto todo el potencial del método. Con los experimentos realizados se ha ilustrado que es necesario utilizar un elevado número de clasificadores (hasta $1000$ predictores) para alcanzar el comportamiento asintótico del conjunto, especialmente para tasas altas de modificación de clases. Además, el método de modificación de etiquetas propuesto, a diferencia del propuesto por Breiman, mantiene constante la probabilidad de modificación global de clase (independientemente de la etiqueta original o la distribución original de clases) para cada ejemplo de entrenamiento. Con esta modificación se pueden utilizar valores más altos de modificación de clases para conjuntos desequilibrados. Esta modificación permite alcanzar errores de generalización significativamente mejores que *flipping* en los conjuntos con distribución desequilibrada de clases. Para conjuntos con distribuciones de clases uniforme, el método desarrollado y el propuesto por Breiman obtienen resultados de clasificación equivalentes.

Otro punto importante abordado en este capítulo es la relación entre la tasa de modificación de clases $p$ con la precisión final del conjunto. Valores más altos de $p$ generan más ruido en los problemas de clasificación que tienen que resolver los algoritmos base. Esto significa que, para mayores valores de $p$, el patrón de clasificación de cada clasificador individual tiene menos similitud con el problema original. En consecuencia, es necesario incluir un mayor número de elementos en el conjunto para perfilar de manera precisa las fronteras de clasificación del problema original. No obstante, lejos de ser una desventaja, el uso de valores altos de $p$ genera fronteras de clasificación más complejas que, en los problemas analizados, conducen a mejores tasas de generalización. Existe un límite superior para el valor de $p$ que se puede utilizar. Este límite corresponde al valor por encima del cual los clasificadores individuales se acercan al funcionamiento de un clasificador aleatorio. Los experimentos realizados muestran que los conjuntos *class-switching* con valores de la tasa de modificación de clases relativa de $3/5$ alcanzan los mejores resultados en promedio para los problemas analizados.

Asimismo, el método propuesto para la generación de los conjuntos de entrenamiento perturbados permite realizar un análisis estadístico del proceso de entrenamiento para

problemas de dos clases en términos de un proceso de Bernoulli. Suponiendo que los clasificadores individuales tienen suficiente flexibilidad para alcanzar error de clasificación nulo en los conjuntos perturbados, entonces las curvas de aprendizaje que muestran la dependencia del error en función del tamaño del conjunto se pueden describir como una suma de términos de una distribución binomial. Además estas curvas de error en el conjunto de entrenamiento son independientes del problema de aprendizaje y sólo dependen de la tasa de modificación de clase $p$, siempre que se usen conjuntos de datos en los que no existan varios ejemplos caracterizados por el mismo vector de atributos.

# Parte II

# Ordenación y poda de conjuntos de clasificadores

# Capítulo 5

# Orden de agregación y poda en conjuntos *bagging*

*El orden en que los clasificadores se agregan en un conjunto puede ser una herramienta útil para la selección de subconjuntos de clasificadores más eficientes que el conjunto original completo. En general, el error de generalización de un conjunto de clasificadores ordenados aleatoriamente disminuye al incrementarse el número de clasificadores y tiende de manera asintótica a un valor constante. Si se modifica adecuadamente el orden de agregación de los clasificadores del conjunto, el error de generalización puede alcanzar un mínimo cuyo valor esté por debajo del error asintótico del conjunto completo. En este capítulo se presentan varias heurísticas que utilizan las correlaciones entre clasificadores generados mediante bagging para identificar un orden apropiado que permita seleccionar un subconjunto de clasificadores con buenas capacidades de generalización. Una vez ordenado el conjunto éste se poda para seleccionar los $\tau$ primeros clasificadores de acuerdo con un porcentaje de poda prefijado o mediante otras reglas de poda. De esta manera se pueden construir conjuntos de clasificadores de menor tamaño y con menor error de clasificación en conjuntos de test que el conjunto original completo.*

## 5.1. Introducción

Como hemos visto en los capítulos precedentes, los conjuntos de clasificadores consiguen reducir el error de clasificación mediante la combinación de las decisiones de clasificadores del mismo tipo pero que presentan cierta variabilidad. En *bagging* la diversidad entre clasificadores individuales se obtiene variando los datos de entrenamiento: cada clasificador se construye usando un conjunto de entrenamiento generado con un muestreo *bootstrap* con repetición. Cada conjunto de datos generado contiene en media un $63.2\,\%$ de los datos del conjunto original.

El comportamiento típico en *bagging* es que el error disminuye de manera monótona

a medida que aumenta el tamaño del conjunto. El error tiende asintóticamente a un valor constante que se considera el mejor resultado que *bagging* puede alcanzar. A medida que se añaden más clasificadores al conjunto, éstos tienden a compensar los errores cometidos por los clasificadores precedentes sin tener en cuenta de forma explícita la complementariedad entre los elementos del conjunto. Esto hace que sea necesario el uso de un gran número de clasificadores (50–200) para garantizar la convergencia. Asimismo, el uso de un gran número de clasificadores supone un coste añadido tanto en memoria necesaria para almacenar el conjunto, como en tiempo de ejecución para clasificar nuevas instancias. Este último aspecto es crítico en aplicaciones en las que es necesario clasificar rápidamente ejemplos.

Las preguntas que surgen en este punto y a las que daremos respuesta en este capítulo son: ¿Podemos modificar este proceso estocástico de forma que la curva de aprendizaje que describe la evolución del error con el número de clasificadores incorporados al conjunto tenga un descenso inicial más rápido? ¿Se pueden aprovechar las correlaciones entre los clasificadores del conjunto para hacer que *bagging* alcance mejores errores de generalización con un subconjunto de clasificadores de menor tamaño que el conjunto original? En este capítulo presentamos una serie de métodos que aprovechan las correlaciones entre los clasificadores individuales en *bagging* para seleccionar un subconjunto de clasificadores de tamaño menor que el conjunto original que mejore la capacidad de generalización de todo conjunto.

En la sección 5.3 describiremos brevemente investigaciones recientes relacionadas con la poda en conjuntos de clasificadores. En la sección 5.4 se presentan las reglas propuestas en esta tesis para modificar el orden de agregación de conjuntos de clasificadores generados con *bagging*: reducción de error (variante de una regla presentada en [Margineantu y Dietterich, 1997]), medida de complementariedad, minimización de distancias de margen, ordenación por ángulos y ordenación basada en *boosting*. Las heurísticas presentadas se han probado empíricamente en 18 conjuntos de datos sintéticos y de diversos campos de aplicación obtenidos de la colección de problemas de UCI [Blake y Merz, 1998]. Finalmente, se exponen las conclusiones de este capítulo.

## 5.2. Ordenación de clasificadores

Como hemos visto previamente, *bagging* genera los distintos clasificadores que forman parte del conjunto de forma independiente: la construcción de cada uno de ellos depende exclusivamente del muestreo *bootstrap* realizado. Este comportamiento contrasta con el procedimiento utilizado en *boosting*. En *boosting* la construcción de cada clasificador depende de todos los clasificadores generados con anterioridad. El proceso de aprendizaje en *bagging* es, por tanto, no determinista —dos ejecuciones sobre los mismos datos producen dos curvas de error distintas— dependiente de los muestreos aleatorios *bootstrap*.

*Boosting*, por el contrario, es determinista (siempre que el clasificador base lo sea) y produce la misma curva de aprendizaje para los mismos datos de entrenamiento.

Al añadir los primeros clasificadores al conjunto se produce generamente un descenso del error de clasificación tanto en entrenamiento como en test. A medida que se van añadiendo más modelos, la pendiente de bajada del error se va reduciendo hasta que el error de *bagging* se satura en un valor constante. El conjunto compuesto por un único clasificador cometerá un error relativamente alto (hay que recordar que para la construcción de cada clasificador el remuestreo *bootstrap* utiliza sólo en media el $63.2\,\%$ de los ejemplos de entrenamiento). El segundo clasificador que se añade al conjunto compensará una porción importante de los errores cometidos por el primero ya que su conjunto de entrenamiento contendrá en media un $63.2\,\%$ de los ejemplos no utilizados para la construcción del primer clasificador. Cada nuevo clasificador añadido al conjunto compensa cada vez menos errores porque habrá menos ejemplos mal clasificados. A medida que se añaden nuevos clasificadores los errores tienden a desaparecer o el proceso no es capaz de eliminarlos, lo que en cualquier caso resulta en la saturación de la capacidad de aprendizaje y el error del conjunto se estabiliza.

En [Breiman, 2001] se demuestra utilizando la Ley de los Grandes Números que el error de generalización de los bosques aleatorios (hay que recordar que *bagging* es un tipo de bosque aleatorio —*random forest*) siempre converge con alta probabilidad. Es decir, estos algoritmos no tienen tendencia acusada al sobreaprendizaje sino que van alcanzando un valor límite de generalización al añadir nuevos clasificadores. Otra manera de analizar este proceso es como la extracción de una variable $\boldsymbol{Y}$ aleatoria $N$ dimensional siendo $N$ el número de ejemplos de entrenamiento y donde cada elemento $Y_i$ puede tomar, con una probabilidad dada, dos posibles valores: clasificación correcta o clasificación incorrecta del ejemplo de entrenamiento $i$. La generación de cada nuevo clasificador en el proceso de construcción del conjunto se puede identificar como una extracción aleatoria de $\boldsymbol{Y}$. De esta forma, la probabilidad asociada a cada valor de $Y_i$ viene definida como la probabilidad de que un clasificador extraído al azar del conjunto clasifique el ejemplo $\boldsymbol{x}_i$ correctamente. A medida que se realizan extracciones es más probable que todos los ejemplos hayan convergido a su valor nominal de probabilidad. De forma equivalente, a medida que se generan clasificadores es más probable que el conjunto haya convergido a su error final. Desde este punto de vista se puede considerar a *bagging* como un proceso estocástico de tipo Monte Carlo para estimar las probabilidades de clasificar bien cada dato de entrenamiento [Esposito y Saitta, 2003; 2004].

Nuestro objetivo es, una vez que todas las extracciones han sido realizadas (es decir una vez generados los clasificadores del conjunto) modificar el orden de agregación del conjunto para que, aquellas extracciones (clasificadores) más favorables a nuestro propósito (clasificar bien todos los ejemplos), aparezcan antes en la secuencia. La hipótesis que

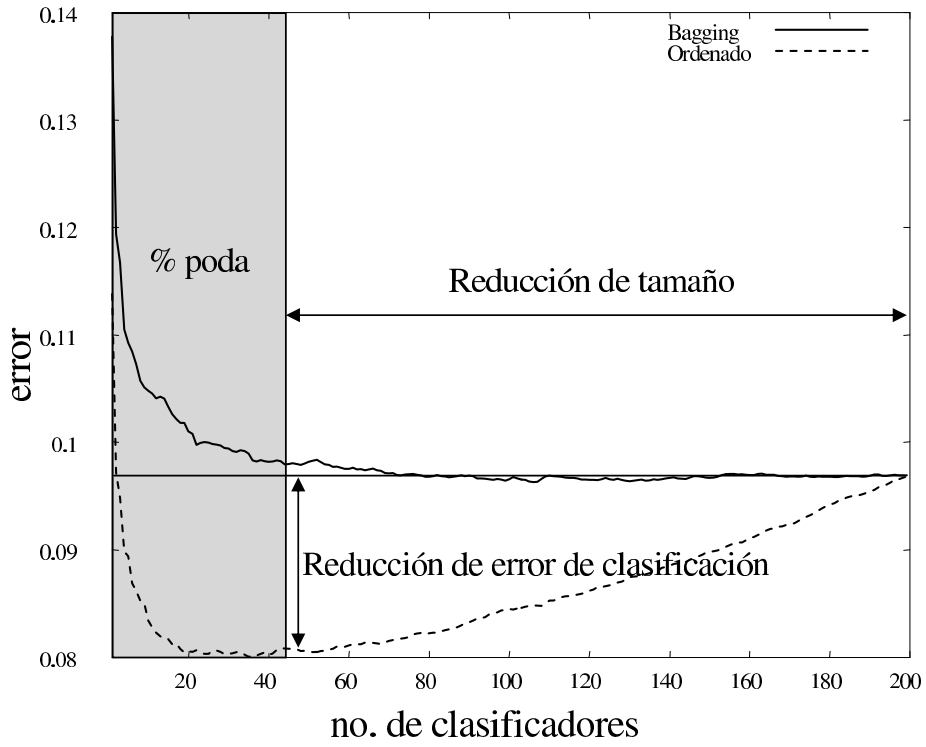



Figura 5.1: Evolución de *bagging* con el número de clasificadores (*línea continua*) y *bagging* ordenado (*línea a trazos*)

formulamos es que la curva de aprendizaje de esta nueva secuencia tendrá una bajada inicial más abrupta para alcanzar un mínimo que estará por debajo del error final de *bagging*, para finalmente ascender de manera lenta y alcanzar el error de *bagging*. El punto final de la curva para cualquier ordenación ha de ser el error final de *bagging* ya que corresponde a incluir todos los clasificadores. Esto es, el resultado de las votaciones es el mismo independientemente del orden en que se emitan los votos. Nuestra hipótesis es que una nueva ordenación de *bagging* puede reducir tanto el número de clasificadores a utilizar como la capacidad de generalización del conjunto. Esto se puede ver esquemáticamente en la figura 5.1.

No es de esperar que esta estrategia funcione en *boosting* ya que se trata de un algoritmo secuencial, en el que los distintos clasificadores se construyen con el objetivo de clasificar correctamente ejemplos en los que los clasificadores anteriores han errado. En *boosting*, ya existe un orden intrínseco de los elementos del conjunto. Esto se puede ver en la figura 5.2 donde se muestran las curvas de error (entrenamiento y test) de *boosting* (gráfico inferior) y *bagging* (gráfico superior) en negrita junto con el resultado de 20 reordenaciones aleatorias

de ambos conjuntos. Se puede observar cómo para *bagging* las distintas secuencias aparecen distribuidas a ambos lados del orden original mientras que en *boosting* todas las nuevas secuencias tienen un error peor que el orden original al menos hasta 40 clasificadores.

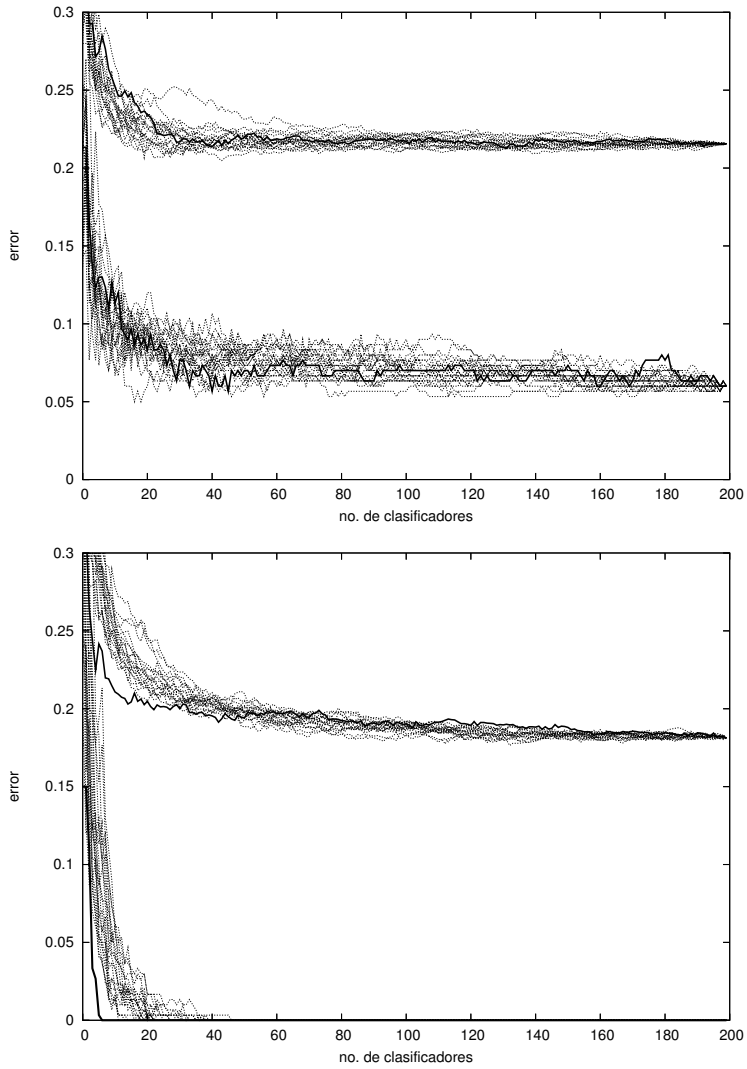



Figura 5.2: Error de entrenamiento (*líneas inferiores*) y test (*líneas superiores*) de 20 ordenaciones aleatorias de un conjunto generado con *bagging* (*gráfico superior*) y otro con *boosting* (*gráfico inferior*). Se ha resaltado el orden original con una línea más gruesa

El número de posibles secuencias u ordenaciones existentes dado que se tiene un conjunto compuesto de $T$ clasificadores es de $T!$, lo que convierte el problema de ordenación óptima en inabordable. Sin embargo, de las $T!$ secuencias muchas son equivalentes ya que

únicamente se quiere seleccionar un subconjunto de clasificadores que tengan buena capacidad de generalización independientemente del orden de los mismos dentro del subconjunto. Aun así, el número de posibles subconjuntos es de $2^T - 1$ (sin contar el conjunto vacío) algunos de los cuales se espera que tengan una capacidad de generalización mejor que la del conjunto completo. Sin embargo, este problema sigue siendo en la práctica intratable para conjuntos de clasificadores que tienen típicamente un tamaño de $\approx 100$ clasificadores. En [Tamon y Xiang, 2000] se demuestra que, suponiendo que la minimización del error de entrenamiento conduce a la minimización del error de generalización, el problema de seleccionar el mejor subconjunto es NP-completo. Por tanto es necesario seguir simplificando el problema. Para ello, hacemos la suposición de que el mejor subconjunto de tamaño $u - 1$ está incluido en el mejor subconjunto de tamaño $u$. Aunque esta suposición no tiene por qué ser correcta, sí parece probable que en general los subconjuntos de tamaños $u - 1$ y $u$ compartan la mayoría de sus clasificadores. De este modo el problema se reduce a diseñar una regla que determine el clasificador a seleccionar en cada paso de entre los clasificadores restantes. Esto reduce los algoritmos de búsqueda a orden $O(T^2)$. Como veremos esto se puede reducir aún más para reglas que calculen una cantidad por clasificador y ordenen por esa cantidad (por ejemplo mediante *quick-sort* lo que da un tiempo medio de ejecución de $O(T \log(T))$).

## 5.3. Otros Trabajos Relacionados

Tal como se afirma en [Margineantu y Dietterich, 1997], una desventaja de los conjuntos de clasificadores es la gran cantidad de memoria requerida para almacenar todos los clasificadores del conjunto. En aplicaciones prácticas se hace difícil justificar el uso de un clasificador que requiere más capacidad de almacenamiento que la base de datos de la que se ha generado, especialmente cuando otros métodos como búsqueda en diccionario o vecinos próximos también pueden dar buenos resultados. Esta observación llevó a Margineantu y Dietterich a investigar si todos los clasificadores de un conjunto generado con *AdaBoost* [Freund y Schapire, 1995] eran fundamentales para la eficiencia del mismo. También hay que tener en cuenta que la reducción de las necesidades de almacenamiento no es la única ventaja que se obtiene de la reducción del tamaño del conjunto. Además se obtienen incrementos en la velocidad de clasificación, elemento crítico en aplicaciones en línea, donde la capacidad de respuesta es proporcional a la velocidad del clasificador.

En [Margineantu y Dietterich, 1997] se propone una serie de heurísticas para seleccionar los clasificadores fundamentales de un conjunto *AdaBoost* con remuestreo para un determinado valor de poda. La mayoría de estas heurísticas están basadas en medidas de diversidad y error de clasificación. Los experimentos presentados indican que se puede reducir de forma significativa el número de clasificadores (con podas de hasta $60 - 80\,\%$

en algunas bases de datos) sin reducir mucho la capacidad de generalización del conjunto. Este estudio fue ligeramente ampliado en [Tamon y Xiang, 2000], donde se propone una modificación menor a una de las podas basadas en diversidad entre clasificadores. La mayor contribución de este último artículo es, sin embargo, y como ya hemos mencionado previamente, la demostración de que la selección del mejor subconjunto de clasificadores en intratable (NP-completo).

Un enfoque distinto consiste en reemplazar el conjunto completo por un nuevo clasificador que emule la salida del conjunto. Esta enorme simplificación permite a expertos humanos analizar el clasificador resultante y no tener así que tratar el conjunto como una caja negra. En esta línea de argumentación se enfoca el artículo [Domingos, 1997] donde se presenta el método CMM (*Combined Multiple Models*). Este método consiste en generar nuevos ejemplos aleatorios que son etiquetados por el conjunto y añadidos a los datos de entrenamiento ya existentes. Posteriormente, se genera un nuevo y único clasificador a partir del conjunto de datos extendido con la idea de que aprenda y se ajuste a las fronteras del conjunto de clasificadores original. El autor implementó CMM con *bagging* usando como clasificador base C4.5RULES [Quinlan, 1993]. En sus experimentos muestra que el conjunto de reglas obtenidas mantiene el 60 % de las mejoras de clasificación obtenidas por el conjunto de clasificadores con una complejidad reducida de reglas. Éstas no superan nunca en más de 6 veces la complejidad del conjunto de reglas que se obtiene aplicando C45RULES directamente.

En [Prodromidis y Stolfo, 2001] se describe una técnica intermedia entre la selección y la substitución de clasificadores. Estos autores proponen la construcción de un clasificador a partir de las salidas de los clasificadores del conjunto para luego hacer una selección de éstos. El método presentado se basa en la poda de coste-complejidad del algoritmo CART [Breiman *et al.*, 1984] (visto en la sección 2.2). Para ello se entrena un árbol CART a partir de un nuevo conjunto de datos cuyos atributos vienen dados por las salidas de los clasificadores del conjunto a los datos de entrenamiento. Para estos datos, la etiqueta de clase viene dada por la decisión final tomada por el conjunto. Posteriormente se poda este árbol con la poda de coste-complejidad de CART. Finalmente se eliminan aquellos clasificadores del conjunto que no se usan en el árbol podado. Los resultados obtenidos muestran que se puede mantener la precisión del conjunto completo con podas de hasta el 60–80 % de los clasificadores base. Con podas del 90 %, el subconjunto se mantiene en el 60–80 % de las tasas de mejora del error del conjunto completo. Este proceso conduce a clasificadores cuya velocidad de clasificación es $6.38$ veces mayor que los conjuntos originales.

Para resolver este problema también se han aplicado técnicas de agrupamiento (*clustering*). El objetivo es agrupar clasificadores por similitud en distintos grupos (*cluster*) para finalmente quedarse con un representante de cada grupo [Giacinto y Roli, 2001; Bakker y Heskes, 2003]. En [Giacinto y Roli, 2001] usan conjuntos pequeños de redes neuronales —hasta 23 modelos con distintas arquitecturas— de los que obtienen mejoras

de hasta $6.8$ puntos porcentuales ($88.04\,\% \rightarrow 94.83\,\%$ ) con respecto al conjunto completo usando 3 de los modelos generados. Sin embargo, el subconjunto seleccionado tiene un porcentaje mayor de patrones rechazados que pasa de $1.64\,\%$ en el conjunto completo a $4.72\,\%$ de los patrones para el subconjunto seleccionado siendo la mejora efectiva de $\approx 3.6$ puntos porcentuales. Estos resultados se obtuvieron en un dominio específico de clasificación de cultivos a partir de imágenes multiespectrales de satélite. En [Bakker y Heskes, 2003] se aplica el algoritmo propuesto a dos problemas de regresión. En los experimentos realizados consiguen reducir de 50 redes neuronales iniciales a entre 3 y 7 (dependiendo de la configuración de las redes) con resultados equivalentes o ligeramente mejores que los de todo el conjunto.

En una serie de artículos publicados por Zhou *et al.* [Zhou *et al.*, 2002; Zhou y Tang, 2003] se aplican algoritmos genéticos (AG) para buscar el subconjunto óptimo de clasificadores. En estos trabajos se aplica AG para determinar las ponderaciones óptimas para los clasificadores base de un conjunto para reducir el error de clasificación. El AG utiliza un cromosoma con un esquema de coma flotante en que cada gen representa el peso de cada red neuronal dentro del conjunto. Se hace evolucionar una población de estos cromosomas normalizando la suma de pesos dentro de cada cromosoma en cada generación. La idoneidad de cada cromosoma se evalúa con el inverso del error. El error se calcula mediante voto ponderado de los clasificadores usando los pesos dados por el cromosoma. Una vez seleccionado el esquema de pesos del conjunto se seleccionan los clasificadores que superen el peso medio [Zhou *et al.*, 2002]. Asimismo, este procedimiento lo aplican a conjuntos de árboles de decisión utilizando un esquema binario de donde se obtiene directamente qué clasificadores estarán en la selección final y cuáles no [Zhou y Tang, 2003]. Los experimentos se realizan con conjuntos pequeños (20 elementos) generados con *bagging* y presentan una mejora tanto en el error de clasificación como en la reducción del tamaño del conjunto. No están claras las ventajas de utilizar AG para este problema, ya que dado el reducido número de clasificadores del conjunto sería posible llevar a cabo una búsqueda exhaustiva para encontrar el subconjunto óptimo.

En [Demir y Alpaydin, 2005] se introduce un factor de utilidad que tiene en cuenta el coste de clasificar nuevas instancias y así seleccionar el subconjunto que maximiza la función utilidad en velocidad de clasificación y error.

## 5.4. Algoritmos de ordenación

En esta sección se proporciona una descripción detallada de las reglas desarrolladas en esta tesis para la selección del orden de agregación de los clasificadores del conjunto. El uso de estas reglas de ordenación permite mejorar el error de generalización de *bagging* mediante la selección de subconjuntos de clasificadores del conjunto original. Partiendo de un subconjunto de tamaño $u - 1$, se obtiene uno de tamaño $u$ añadiendo un nuevo

clasificador, seleccionado de acuerdo con una regla determinada. El orden aleatorio inicial $(t = 1, 2, \ldots, T)$ de los clasificadores de *bagging* se reemplaza por un orden distinto $(s_1, s_2, \ldots s_T)$, donde $s_j$ es el índice de la posición original del clasificador que ocupa la posición $j$ en la nueva ordenación del conjunto. Finalmente, se seleccionan los $\tau$ primeros clasificadores dependiendo del nivel de poda deseado.

En esta sección vamos a simplificar la notación dada en el capítulo 2. La entrada de los algoritmos de aprendizaje consiste en un conjunto de datos de entrenamiento etiquetados, ec. (2.1): $L = \{(\boldsymbol{x}_i,\, y_i), i = 1, 2, ..., N\}$ donde $\boldsymbol{x}_i$ es el vector de atributos e $y_i$ es la etiqueta de clase. Por simplicidad consideraremos sólo problemas de clasificación binarios donde $y \in \{-1, 1\}$. Los resultados se pueden extender fácilmente a problemas con múltiples clases. Como hemos visto, *bagging* genera una serie de hipótesis $\{h_t(\boldsymbol{x}) : t = 1, \ldots, T\}$, mediante diferentes remuestreos *bootstrap* de $L$. La decisión final de *bagging* se toma por mayoría de acuerdo con la ecuación de salida de la figura 2.6. Si asumimos clasificación binaria donde $h_t(\boldsymbol{x}) = \pm 1$ la hipótesis combinada se puede expresar como

$$H(\boldsymbol{x}) = signo\left(\sum_{t=1}^{T} h_t(\boldsymbol{x})\right) . \tag{5.1}$$

### 5.4.1. Ordenación basada en propiedades individuales

Se han llevado a cabo una serie de experimentos preliminares para establecer si las características de los clasificadores individuales son medidas útiles para la ordenación de los conjuntos. En concreto, se han utilizado distintas estimaciones del error individual de generalización de los clasificadores para establecer un orden dentro del conjunto:

- Ordenación usando el error de los clasificadores en el conjunto de entrenamiento.
- Ordenación usando el error en el conjunto *out-of-bag* de cada clasificador (ejemplos no utilizados por el clasificador al entrenar dejados fuera por el proceso de *bootstrap* [Breiman, 1996c]).
- Ordenación estimando el error en un conjunto independiente del conjunto de entrenamiento y de test y suficientemente grande.

Hemos comprobado cómo el error en entrenamiento no es un indicador fiable del error de generalización de cada clasificador y no conduce a ninguna ordenación útil del conjunto. Asimismo, tampoco ha llevado a ninguna ordenación válida el uso de un conjunto independiente del conjunto de entrenamiento como es el conjunto cambiante *out-of-bag* (este conjunto varía de un clasificador a otro). Los conjuntos *out-of-bag* presentan el problema añadido de que la comparación de los errores individuales de distintos clasificadores no es fiable debido a las fluctuaciones de muestreo de los distintos conjuntos *out-of-bag*.

En cualquier caso, el uso de un conjunto de validación fijo para todos los clasificadores y suficientemente grande tampoco conduce a ninguna ordenación que produzca una mejora apreciable en el error de generalización. Basándonos en estos resultados, concluimos que ordenaciones guiadas por las capacidades individuales de los clasificadores del conjunto no llevan a la identificación de un subconjunto que supere al conjunto generado por *bagging* completo. Para diseñar una regla de ordenación válida es necesario tener en cuenta la complementariedad de los clasificadores. De hecho, el combinar clasificadores muy precisos pero muy similares no se obtienen mejoras en la clasificación, mientras que si se combinan clasificadores diversos que compensan sus errores sí que se obtiene una mejora en la clasificación.

### 5.4.2. Algoritmos de ordenación codiciosos

Los métodos de ordenación efectivos han de tener en cuenta la complementariedad entre los distintos elementos del conjunto para realizar la ordenación. Un clasificador individual puede tener un error alto de clasificación pero su contribución puede ser importante al combinarlo con otros clasificadores [Esposito y Saitta, 2003; 2004]. A continuación se describen las reglas de ordenación propuestas que siguen una estrategia codiciosa y que son eficaces para la reducción del error de generalización como veremos en la sec. 5.5. Estas reglas son: reducción de error, complementariedad, minimización de distancias de margen, ordenación por ángulos y ordenación basada en *boosting*. Estas reglas usan un conjunto de selección compuesto de $N_{sel}$ ejemplos $L_{sel} = \{(\boldsymbol{x}_i,\, y_i), i = 1, 2, ..., N_{sel}\}$, que en principio puede conincidir con el conjunto de entrenamiento.

#### Reducción de error

Este método es equivalente al presentado en [Margineantu y Dietterich, 1997] sin reajuste (*backfitting*). Funciona como sigue: (i) se inicializa la secuencia eligiendo el clasificador con menor error en el conjunto de datos de selección; (ii) a continuación se añaden clasificadores uno a uno de forma que se minimice el error del conjunto parcial de clasificadores en el conjunto $L_{sel}$. Por consiguiente, el clasificador seleccionado en la iteración $u$ es el que maximiza la expresión

$$s_u = \underset{k}{argmax} \sum_{i=1}^{N_{sel}} signo \left( h_k(\boldsymbol{x}_i) + \sum_{t=1}^{u-1} h_{s_t}(\boldsymbol{x}_i) \right) y_i \ , \tag{5.2}$$

donde el índice $k$ tiene como rango las etiquetas de los clasificadores que no han sido incluídos en el subconjunto de tamaño $u - 1$.

**Medida de complementariedad**

Este procedimiento favorece la inclusión de clasificadores cuyo funcionamiento sea complementario al del subconjunto ya seleccionado. Como en la regla precedente el conjunto se inicia seleccionando el clasificador con menor error en $L_{sel}$. A continuación, se construye el subconjunto de tamaño $u$ a partir del de tamaño $u-1$ incorporando el clasificador que maximiza

$$s_u = \underset{k}{argmax} \sum_{i=1}^{N_{sel}} I\left(y_i = h_k(\boldsymbol{x}_i)\ AND\ y_i \neq signo\left(\sum_{t=1}^{u-1} h_{s_t}(\boldsymbol{x}_i)\right)\right)\ , \qquad (5.3)$$

donde $k$ recorre las etiquetas de los clasificadores que aún no han sido seleccionados y donde $I(true) = 1,\ I(false) = 0$. Esta medida se puede interpretar como la cantidad que un clasificador desplaza la decisión del conjunto hacia la clasificación correcta. Este criterio selecciona para su inclusión en el subconjunto, el clasificador que clasifican bien el mayor número de datos donde el subconjunto parcial está fallando.

**Minimización de la distancia de margen**

Considerando el conjunto de datos $L_{sel}$ compuesto de $N_{sel}$ elementos. Definimos $\boldsymbol{c}_t$, como el vector característico del clasificador $h_t$, como un vector de dimensión $N_{sel}$ cuyos componentes son

$$c_{ti} = y_i h_t(\boldsymbol{x}_i),\ i = 1, 2, \ldots, N_{sel}\ , \qquad (5.4)$$

donde $c_{ti}$ es igual a 1 si $h_t$ clasifica correctamente el ejemplo $i$ de $L_{sel}$ y $-1$ en caso contrario. La media de los vectores característicos del conjunto es

$$\boldsymbol{c}_{ens} = \frac{1}{T} \sum_{t}^{T} \boldsymbol{c}_t\ . \qquad (5.5)$$

En un problema de clasificación binario, la componente $i$ del vector característico promedio del conjunto es igual al margen del ejemplo $i$, definido en el intervalo $[-1, 1]$ como la diferencia entre los votos que recibe la clase correcta y los votos que recibe la clase incorrecta más común [Schapire *et al.*, 1998]. En general, para problemas con múltiples clases, esta cantidad es igual a $(1 - 2\, edge(i))$ del conjunto para el ejemplo $i$, donde *edge* se define como la diferencia entre los votos que recibe la clase correcta y todos los que reciben las clases incorrectas, normalizado al intervalo $[0, 1]$ [Breiman, 1997]. Se tiene por tanto que el ejemplo $i$ será correctamente clasificado por el conjunto si la componente $i$ del vector característico promedio $\boldsymbol{c}_{ens}$ es positiva. Esto es, un conjunto cuyo vector característico promedio esté en el primer cuadrante del espacio $N_{sel}$ dimensional clasificará correctamente todos los ejemplos del conjunto $L_{sel}$.

Nuestro objetivo por tanto es identificar un subconjunto de clasificadores cuyo vector característico promedio esté lo más próximo posible al primer cuadrante. Para ello seleccionamos una posición arbitraria en el primer cuadrante como un vector constante con componentes iguales como

$$o_i = p \qquad i = 1, \ldots, N_{sel} : \text{con } 0 < p < 1 \; . \tag{5.6}$$

Los clasificadores se añaden al conjunto de forma que se reduzca lo más posible la distancia de $\boldsymbol{c}_{ens}$ al punto objetivo $\boldsymbol{o}$. El clasificador seleccionado en la iteración $u$ es el que minimiza

$$s_u = \underset{k}{argmin}\, d\left(\boldsymbol{o}, \frac{1}{T}\left(\boldsymbol{c}_k + \sum_{t=1}^{u-1} \boldsymbol{c}_{s_t}\right)\right) \; , \tag{5.7}$$

donde $k$ recorre las etiquetas de los clasificadores fuera del subconjunto y donde $d(\boldsymbol{u}, \boldsymbol{v})$ es la distancia euclídea entre los vectores $\boldsymbol{u}$ y $\boldsymbol{v}$.

El valor de $p$ elegido debe ser pequeño ($p \in (0.05, 0.25)$). De este modo, los ejemplos de fácil clasificación (aquellos correctamente clasificados por la mayoría de clasificadores) pasarán a tener un valor cercano a $p$ desde las primeras iteraciones y, consecuentemente, su influencia en el proceso de selección de los siguientes clasificadores es menor, lo que incrementa la influencia de los ejemplos más difíciles de clasificar. Si se eligiera un valor de $p$ cercano a 1 habría una atracción similar para todos los ejemplos durante todo el proceso de selección, lo que genera ordenaciones menos efectivas.

En la figura 5.3 se muestra gráficamente el proceso que sigue este método para ordenar los clasificadores. En esta figura se han dibujado unos hipotéticos vectores característicos de dos dimensiones (esto es $N_{sel} = 2$) para un conjunto compuesto de 11 clasificadores. En negro se han dibujado los vectores correspondientes a los clasificadores ordenados según el proceso aleatorio de *bagging*. En gris se ven los mismos clasificadores ordenados según la minimización de distancias de margen donde el punto objetivo $\boldsymbol{o}$ se ha dibujado con un punto gris. Obviamente, ambas ordenaciones acaban en el mismo punto (punto negro en la figura).

### Ordenación por ángulos

Esta regla de ordenación está basada en criterios similares a los de la regla precedente pero utiliza los ángulos de los vectores característicos con respecto a un vector de referencia, $\boldsymbol{c}_{ref}$. Este vector de referencia se define como la proyección de la diagonal del primer cuadrante en el hiperplano definido por $\boldsymbol{c}_{ens}$. A continuación ordenamos los clasificadores de menor a mayor por los valores de los ángulos entre los vectores característicos de cada clasificador base y el vector de referencia $\boldsymbol{c}_{ref}$.

En un conjunto de clasificadores de tamaño $T$, la operación de ordenación se puede hacer usando el algoritmo *quick-sort*, que tiene un tiempo medio de ejecución de

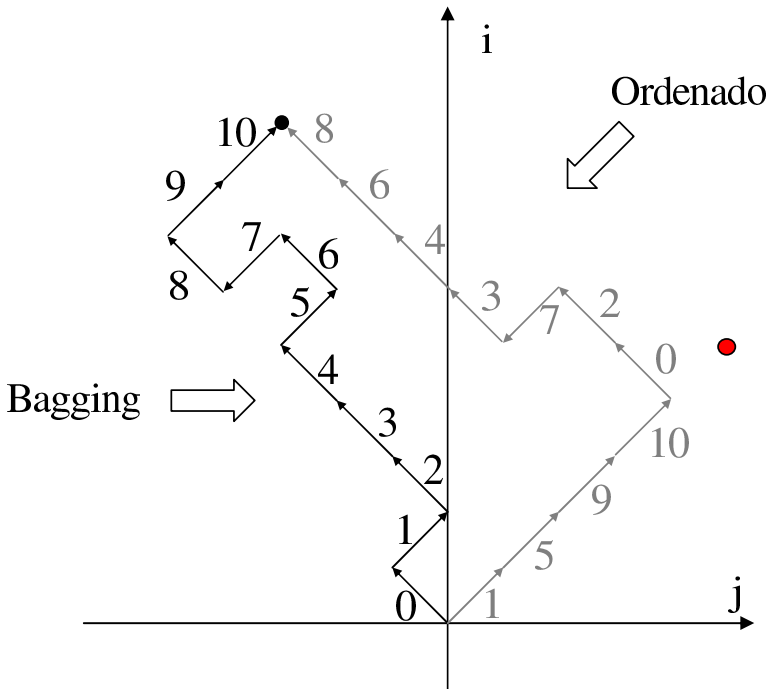



Figura 5.3: Vectores característicos de 11 clasificadores ordenados según el proceso aleatorio de *bagging* (*en negro*) y el mismo conjunto de vectores ordenado con el método de minimización de distancias de margen (*en gris*). Más detalles en el texto

$O(T\log(T))$. También se puede utilizar el *quick-select* si solamente estamos interesados en la selección de los $\tau$ mejores clasificadores. Esto da un tiempo medio de ejecución de $O(T)$. Esta técnica difiere sensiblemente de todas las demás, ya que tiene un tiempo de ejecución lineal frente a la complejidad cuadrática del resto. El resto de reglas presentadas requieren la definición de una cantidad que es evaluada en los clasificadores restantes para elegir el mejor de ellos. La evaluación se debe hacer en todos los pasos ya que las medidas definidas tienen en cuenta el subconjunto ya seleccionado, y éste es modificado en cada paso. Por el contrario esta regla define un punto de referencia fijo $\boldsymbol{c}_{ref}$ con respecto al cual se evalúan los clasificadores, lo que permite hacer una ordenación directa basada en propiedades individuales de los clasificadores en relación al conjunto.

El vector de referencia, se ha elegido de forma que se maximiza el torque sobre $\boldsymbol{c}_{ens}$ (que representa la tendencia central del conjunto completo) con respecto a la dirección que corresponde a la clasificación perfecta (primer cuadrante). Este efecto se obtiene eligiendo $\boldsymbol{c}_{ref} = \boldsymbol{o} + \lambda\boldsymbol{c}_{ens}$, donde $\boldsymbol{o}$ es un vector sobre la diagonal del primer cuadrante, y $\lambda$ es una constante tal que $\boldsymbol{c}_{ref}$ sea perpendicular a $\boldsymbol{c}_{ens}$ ($\boldsymbol{c}_{ref}\perp\boldsymbol{c}_{ens}$). Veamos un ejemplo: consideremos un conjunto de entrenamiento compuesto de tres ejemplos y un conjunto de

clasificadores con $\boldsymbol{c}_{ens} = \{1, 0.5, -0.5\}$. Este vector corresponde a un conjunto en el que el primer ejemplo es clasificado correctamente por todos los clasificadores del conjunto, el segundo por un 75 % de los clasificadores y el tercero por un 25 %. La proyección se calcula requiriendo que $\boldsymbol{c}_{ref} = \boldsymbol{o} + \lambda \boldsymbol{c}_{ens}$ y que $\boldsymbol{c}_{ref}$ sea perpendicular a $\boldsymbol{c}_{ens}$ ( $\boldsymbol{c}_{ref} \perp \boldsymbol{c}_{ens}$). Esto se cumple para el valor $\lambda = -\boldsymbol{o} \cdot \boldsymbol{c}_{ens} / |\boldsymbol{c}_{ens}|^2$. Es este caso lambda queda $\lambda = -2/3$ y en consecuencia $\boldsymbol{c}_{ref} = \{1/3, 2/3, 4/3\}$. Con esta elección para $\boldsymbol{c}_{ref}$, en la fase de ordenación, se ejercerá un torque mayor sobre las dimensiones correspondientes a los ejemplos más difíciles de clasificar por el conjunto, esto es, los ejemplos tercero y segundo. Por otro lado, hay que tener en cuenta, que el vector $\boldsymbol{c}_{ref}$ es inestable cuando los vectores que definen la proyección ($\boldsymbol{c}_{ens}$ y la diagonal del primer cuadrante) están cerca. Pequeñas variaciones de $\boldsymbol{c}_{ens}$ (por ejemplo si quitamos un número pequeño de clasificadores al conjunto) puede hacer que $\boldsymbol{c}_{ref}$ cambie su sentido. En estos casos, por tanto, la ordenación es menos fiable y el proceso menos efectivo. Este es el caso cuando se usan conjuntos que alcanzan rápidamente error cero en entrenamiento, como *boosting* o *bagging* con árboles sin podar.

En la figura 5.4 se muestra una representación gráfica del proceso de aprendizaje de *bagging* y del proceso de aprendizaje tras la ordenación. Las curvas mostradas corresponden a la proyección de los caminos aleatorios de *bagging* (línea continua) y *bagging* ordenado (línea a trazos) seguidos por la suma incremental de los vectores característicos ($\sum_{t=1}^{\tau} \boldsymbol{c}_t;\ \tau = 1, 2, \ldots, T$) en dos y tres dimensiones. En estos gráficos se puede observar la naturaleza estocástica del proceso de aprendizaje de *bagging*. A medida que se incrementa el número de clasificadores las probabilidades de clasificar correctamente cada ejemplo de entrenamiento vienen determinadas de forma más precisa [Esposito y Saitta, 2004]. Esto es, cada nuevo clasificador que se añade al conjunto modifica en menor medida que los anteriores la dirección de $\boldsymbol{c}_{ens}$. Además se puede observar como este proceso aleatorio puede ser modificado reordenando los clasificadores para dar lugar a una mejor clasificación. Estas curvas han sido calculadas para un conjunto de 200 clasificadores entrenados en el problema *Waveform* usando 300 ejemplos de entrenamiento (los vectores característicos tienen 300 dimensiones). En el gráfico superior los vectores se han proyectado en dos dimensiones en el plano definido por $\boldsymbol{c}_{ens}$ (eje z en el gráfico) y $\boldsymbol{c}_{ref}$ (eje x). El gráfico intermedio muestra la proyección sobre un plano perpendicular a $\boldsymbol{c}_{ens}$, definido por $\boldsymbol{c}_{ref}$ (eje x) y un vector perpendicular a $\boldsymbol{c}_{ens}$ y $\boldsymbol{c}_{ref}$ (eje y). En este caso las curvas mostradas son una proyección sobre un plano perpendicular al vector que define al conjunto, $\boldsymbol{c}_{ens}$. Por tanto, cualquier camino que incluya todos los clasificadores empieza y acaba en el origen de coordenadas. Finalmente, el gráfico inferior es una proyección en 3 dimensiones sobre los ejes x, y, z previamente definidos. Para *bagging* (línea continua) se puede ver como la suma incremental de los vectores característicos sigue una ruta que se puede considerar como un puente browniano que comienza en el origen y acaba en $T \times \boldsymbol{c}_{ens}$. El conjunto ordenado (línea a trazos) reordena los pasos del camino aleatorio original de forma que los primeros pasos conducen a una máxima aproximación del caminante con $\boldsymbol{c}_{ref}$. De ahí la forma característica del recorrido, alargada en la dirección de $\boldsymbol{c}_{ref}$.

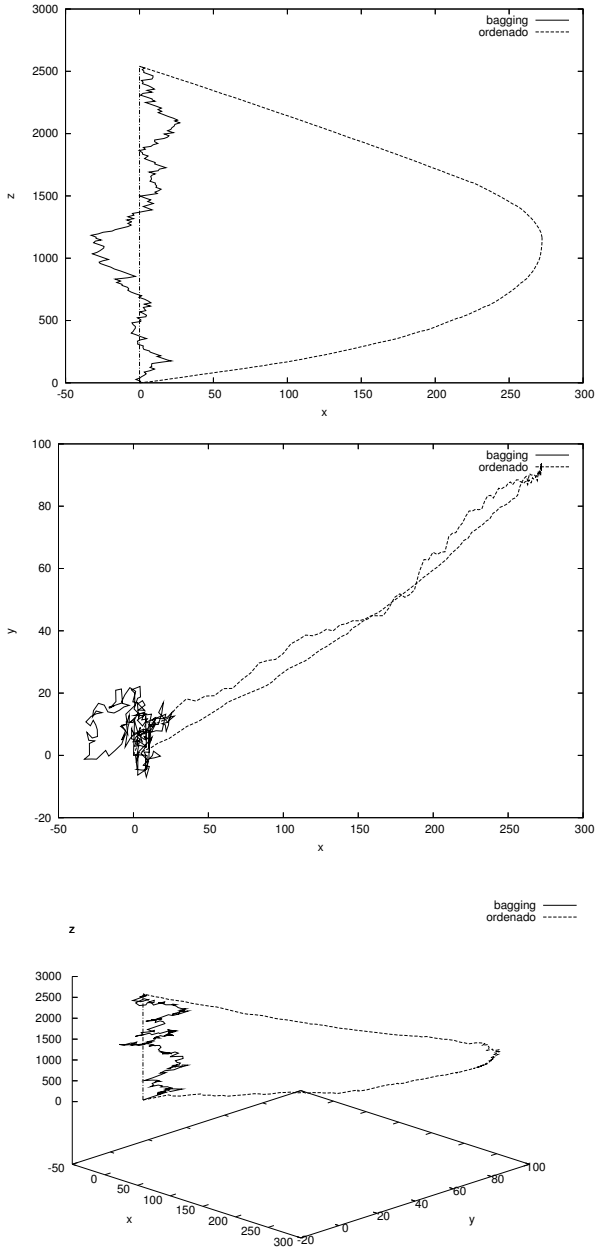



Figura 5.4: Proyección de la suma incremental de los vectores característicos de *bagging* ordenados (*línea a trazos*) y sin ordenar (*línea continua*) en: dos dimensiones $\boldsymbol{c}_{ens}$ (eje z) y $\boldsymbol{c}_{ref}$ (eje x) (*gráfico superior*), dos dimensiones $\boldsymbol{c}_{ref}$ y un eje perpendicular a $\boldsymbol{c}_{ref}$ y a $\boldsymbol{c}_{ens}$ (eje y) (*gráfico intermedio*) y en las tres dimensiones definidas previamente (*gráfico inferior*). Los gráficos son para el problema *Waveform* con 300 ejemplos y 200 clasificadores

**Entradas:**

Conjunto de entrenamiento $L_{sel}$ de tamaño $N$
Conjunto $H$ compuesto de $T$ clasificadores

**Salida:**

$\{h_{s_1}, h_{s_2}, \ldots, h_{s_T}\}$

1. Asignar $\boldsymbol{w}_1[i] = 1/N,\ i = 1, \ldots, N$
2. for u=1 to T {
   //Obtiene el aprendiz con menor error ponderado
3. $h_{s_u}$ = SeleccionaClasificador($H$, $L_{sel}$, $\boldsymbol{w}_u$)
4. $\epsilon_u$ = Error($h_{s_u}$, $L_{sel}$, $\boldsymbol{w}_u$)
5. $\beta_u = \epsilon_u/(1 - \epsilon_u)$
6. if $(\epsilon_u \geq 0.5)$ {
7. Asignar pesos $\boldsymbol{w}_{u+1}[i] = 1/N,\ i = 1, \ldots, N$
8. continue
9. }
11. Extraer $h_{s_u}$ de $H$
12. for j=1 to N {
13. if $(h_{s_u}(\boldsymbol{x}_j) \neq y_j)$ then $w_{u+1}[j] = w_u[j]/2\epsilon_u$
14. else $w_{u+1}[j] = w_u[j]/2(1 - \epsilon_u)$
15. }
16. }

Figura 5.5: Pseudocódigo de ordenación basada en *boosting*

**Poda basada en boosting**

Esta regla de poda se basa en utilizar el esquema de ponderación de los ejemplos del algoritmo *AdaBoost* [Freund y Schapire, 1995] para determinar el orden en que se agregan los clasificadores. En la figura 5.5 se presenta el pseudocódigo de la ordenación basada en *boosting*. Se parte de un conjunto de clasificadores $H$ generados con *bagging* y un conjunto de datos $L$. El núcleo del algoritmo es similar a *boosting* (figura 2.7). Sin embargo, en vez de generar en cada iteración una hipótesis a partir del conjunto de entrenamiento ponderado, se selecciona ésta de entre los clasificadores aún no seleccionados provenientes de un conjunto *bagging*. En concreto se elige el clasificador con el menor error ponderado en el conjunto de entrenamiento (paso 3). En cada iteración el algoritmo actualiza los pesos de los ejemplos: los pesos de los ejemplos clasificados correctamente (incorrectamente) por el último clasificador seleccionado se decrementan (incrementan) del mismo modo que en *AdaBoost*. Para evitar que el algoritmo pare prematuramente en caso de que no se encuentre

ningún clasificador con un error ponderado menor de 50 % se reasignan los pesos de los ejemplos a $1/N$ y el proceso continúa. Asimismo, y a diferencia de *AdaBoost*, también se continúa con el proceso de selección cuando el clasificador seleccionado tiene error cero. Sin embargo, esta situación se da raras veces, ya que en *bagging* generalmente no se generan clasificadores con error cero en entrenamiento y si existen, éstos son seleccionados en las primeras iteraciones del algoritmo. Una vez finalizado el proceso de selección se pueden usar, como en las reglas precedentes, un porcentaje de poda fijo para seleccionar un subconjunto compuesto por los $\tau$ primeros clasificadores. En lugar de utilizar un valor de poda fija a priori, también se puede usar como regla de parada la propia de *boosting*, esto es, parando la ordenación con el primer clasificador que alcance error por encima de 50 %. Como veremos este método no tiende a seleccionar el tamaño de subconjunto óptimo. Otra modificación que surge de forma natural es tomar la decisión final del conjunto con voto ponderado (usando los pesos como los define *boosting*) en vez de con voto directo. Esta modificación no conduce a mejoras con respecto al voto no ponderado.

### 5.4.3. Validación de la ordenación codiciosa por comparación con algoritmos óptimos de selección

#### Solución exacta por búsqueda exhaustiva

El análisis exhaustivo de todos los posibles subconjuntos no es una solución factible para aplicar a conjuntos de clasificadores que típicamente en la literatura tienen entre 50 y 200 elementos. ¡Habría que evaluar entre $O(2^{50})$ y $O(2^{200})$ subconjuntos! En todo caso, este enfoque lo podemos utilizar en conjuntos más pequeños para comprobar lo lejos que están los subconjuntos obtenidos por aplicación de los algoritmos de ordenación codiciosos propuestos en la sección 5.4.2 de los subconjuntos óptimos del tamaño correspondiente. Esto nos servirá como validación de las heurísticas propuestas como herramientas de optimización en sí mismas.

Esta optimización por búsqueda exhaustiva se ha aplicado al conjunto *Waveform* para obtener la mejor solución para cada posible tamaño de subconjunto. En los experimentos sólo se analizaron los subconjuntos de tamaño impar para reducir a la mitad el tiempo de computación y además reducir los empates en las votaciones.

Se han explorado conjuntos de dos tamaños con un número diferente de ejecuciones:

- Una ejecución para un conjunto compuesto de 31 clasificadores que conlleva la evaluación de $2^{31}/2 = 1\,073\,741\,824$ subconjuntos. El tiempo de ejecución de este proceso fue de 3 días, 3 horas y 50 min. en un Pentium® 4 a 3200 MHz.

- Cien ejecuciones con conjuntos de 25 clasificadores. Esto involucra la evaluación de $100 \times 2^{25}/2 = 1\,677\,721\,600$ subconjuntos. Para cada ejecución se ha requerido en

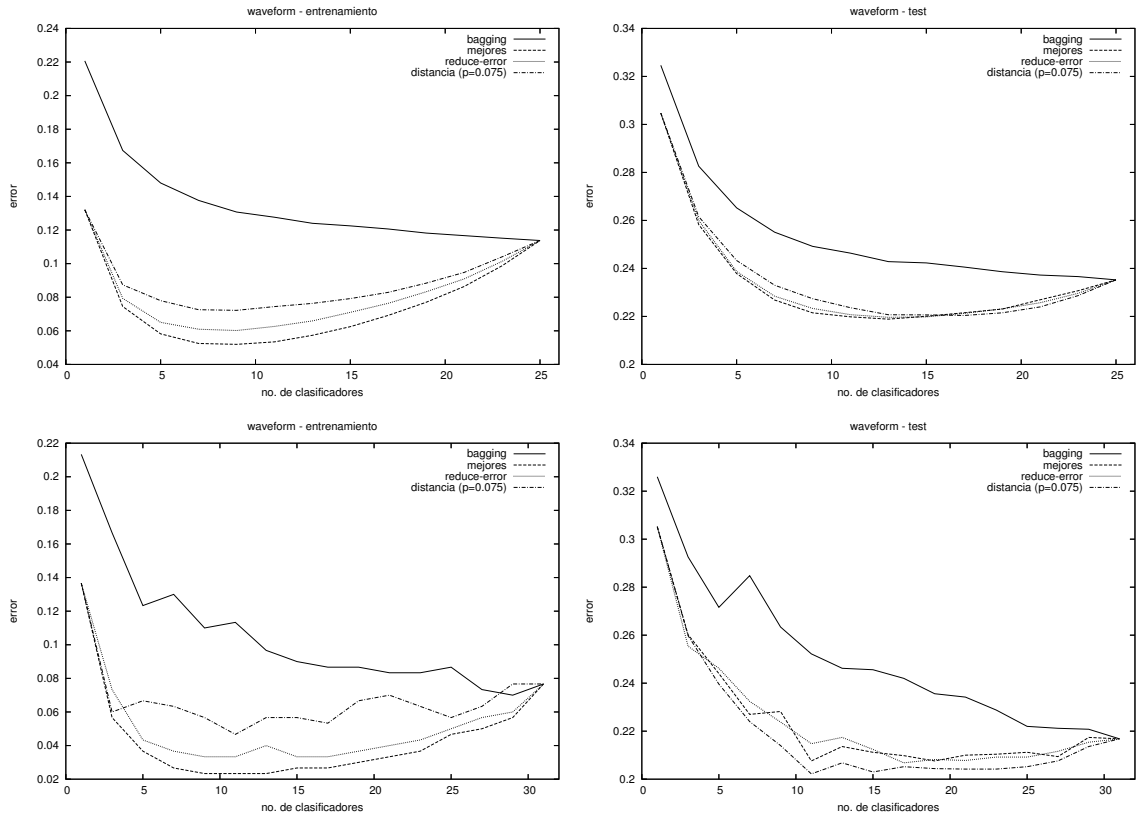



Figura 5.6: Curvas de error de entrenamiento y test para *bagging* (*línea continua*), mejores soluciones (*línea de trazos*), reducción de error (*línea trazo-punto*) y distancias de margen (p=0.075) (*línea punteada*) para *Waveform*

media 4291 seg. de tiempo de CPU en el mismo procesador y casi 5 días de tiempo de procesador en total.

Estos conjuntos fueron ordenados asimismo utilizando los métodos de reducción de error y de minimización de distancias de margen ($p = 0.075$).

En la figura 5.6 se muestran los errores de entrenamiento (gráficas a la izquierda) y test (derecha) para 31 clasificadores (abajo) y la media de las ejecuciones con 25 clasificadores (arriba) para los distintos algoritmos. Las curvas inferiores de las gráficas de entrenamiento no representan una secuencia incremental de clasificadores; cada punto de estas curvas es el error del mejor subconjunto para el tamaño correspondiente (para un número impar de clasificadores) obtenido por búsqueda exhaustiva. Por tanto estas curvas representan el límite inferior de error alcanzable en el conjunto de entrenamiento.

De las figuras de entrenamiento podemos observar que el método de reducción de error

obtiene resultados muy cercanos al mejor subconjunto posible en entrenamiento. En el conjunto de test, los resultados son equivalentes para ambos algoritmos. El algoritmo de distancias de margen ($p = 0.075$) queda más alejado de este límite óptimo en el conjunto de entrenamiento, lo que es razonable dado que este algoritmo no está diseñado para reducir directamente el error de entrenamiento. Sin embargo, el método de minimización de distancias de margen obtiene errores de generalización equivalentes (con 25 clasificadores) o mejores (para 31 clasificadores) que el algoritmo global óptimo. Esta última observación se debe tomar con precaución ya que estos experimentos se han realizado con pocos muestreos y sólo un conjunto de datos. No obstante, los resultados obtenidos confirman los resultados que presentaremos en la sección 5.5.2 donde el método de reducción de error obtiene mejores resultados en entrenamiento que el método reducción de distancias de margen mientras que el comportamiento en test se invierte (ver cuadros 5.7 y 5.8).

Una cuestión interesante que surge en este punto es: ¿Cómo de diferentes son las selecciones de clasificadores hechas por el algoritmo codicioso y el algoritmo que encuentra el óptimo global? Para responder a esta pregunta hemos calculado para cada ejecución una matriz de coincidencias $\boldsymbol{O}_{ij}$ cuyos elementos $\boldsymbol{O}_{ij}$ valen 1 si el clasificador seleccionado en la posición $j$ por el algoritmo codicioso está incluido en la solución óptima de tamaño $i$ y $\boldsymbol{O}_{ij} = 0$ en caso contrario. En el caso en que ambos algoritmos seleccionaran los mismos clasificadores para cada tamaño, la matriz de coincidencias sería una matriz triangular inferior con ceros encima de la diagonal y unos debajo y en la diagonal. La figura 5.7 muestra la matriz de coincidencias media para las 100 ejecuciones con 25 clasificadores (*gráfica superior*) y la matriz de coincidencias para 31 clasificadores (*gráfica inferior*). Los índices $i$ y $j$ se representan en las ordenadas y abscisas respectivamente. En vez de mostrar los valores numéricos se ha optado por mostrar las matrices usando una escala invertida de grises lineal donde las celdas blancas y negras puras representan $\overline{\boldsymbol{O}_{ij}} = 0$ y $\overline{\boldsymbol{O}_{ij}} = 1$ respectivamente, y tonos progresivos de grises para valores intermedios. Esta representación permite comparar rápidamente ambos algoritmos. Por otro lado, en la columna derecha se indica el número medio de soluciones que alcanzan el resultado óptimo para cada tamaño. Cuando hay más de un subconjunto, para un tamaño dado, que obtiene el resultado óptimo, se considera el más similar la solución obtenida por el algoritmo de reducción de error. Esto se hace así porque estamos interesados en determinar lo lejos que están los distintos metodos de selección y esto viene dado por la diferencias entre las soluciones más parecidas. Del mismo modo, tampoco se han tenido en cuenta las posibles soluciones que puede dar la ordenación por reducción de error. Con la ordenación por reducción de error se pueden obtener ordenaciones distintas cuando en un paso del algoritmo se encuentra más de un clasificador que reduce el error en la misma medida, la elección de uno u otro daría secuencias de ordenación distintas.

Estas figuras muestran que las matrices de coincidencias presentan un patrón muy similar a una matriz diagonal inferior con una pequeña dispersión cerca de la diagonal. También hay que hacer notar que clasificadores seleccionados al final por el algoritmo codicioso no

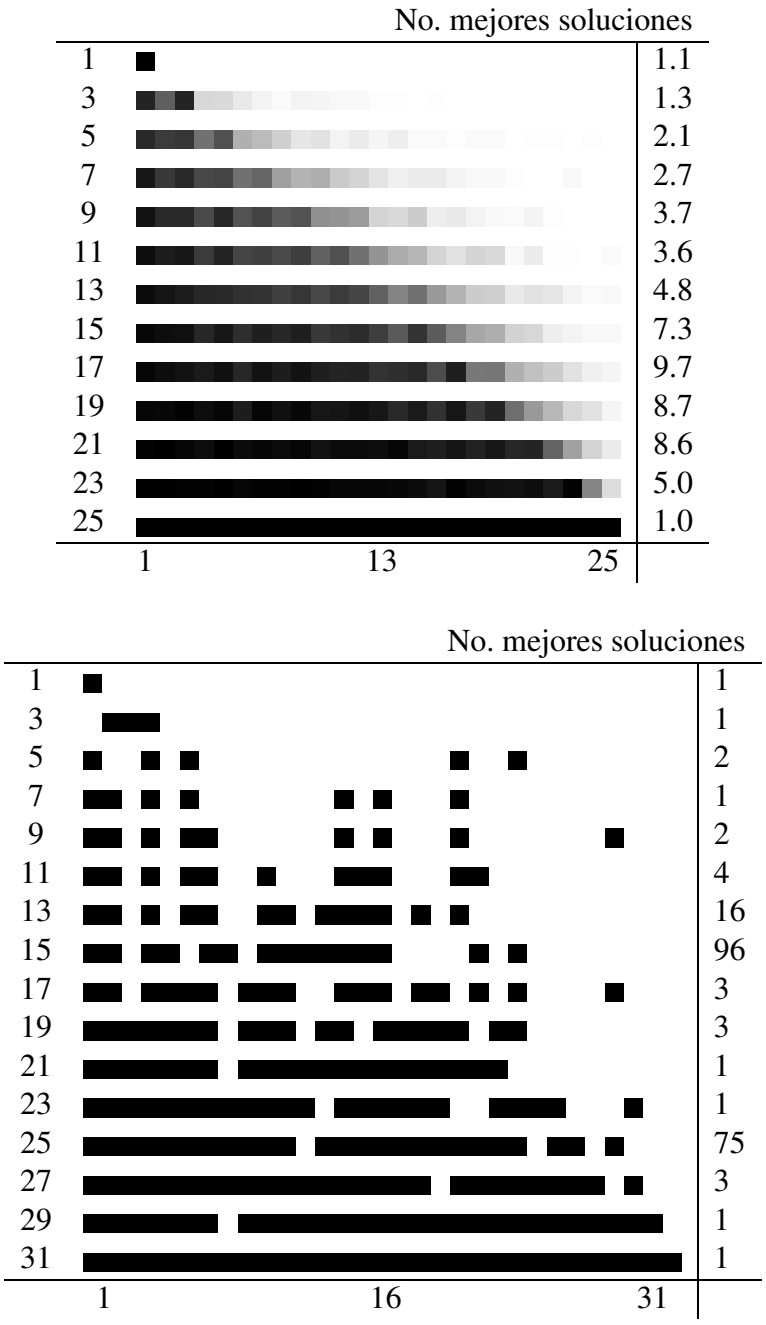



Figura 5.7: Matrices de coincidencias $\boldsymbol{O}_{ij}$ que representan la selección de cada clasificador usando la mejor solución (*ordenadas*) y reducción de error (*abscisas*). El número de mejores soluciones encontradas para cada tamaño se muestra en la columna derecha (más detalles en el texto)

se seleccionan en las soluciones óptimas de subconjuntos pequeños. Estos resultados validan la suposición inicial que se hizo al inicio del capítulo y en la que se han basado las heurísticas desarrolladas. Es decir, el mejor subconjunto de tamaño $u-1$ comparte con el mejor subconjunto de tamaño $u$ la mayoría de sus elementos.

**Selección de subconjuntos utilizando algoritmos genéticos**

Para conjuntos de tamaño superior no se puede hacer búsqueda exhaustiva, por lo que hemos recurrido a algoritmos genéticos (AG) para buscar subconjuntos de *bagging* óptimos. Al igual que en los experimentos de la sección 5.4 hemos usado conjuntos de 200 clasificadores. Para este enfoque se han utilizado representaciones y parámetros similares a los experimentos mostrados en [Zhou y Tang, 2003] y usando las recomendaciones de [Eiben y Smith, 2003].

Para la representación del cromosoma se ha considerado una cadena binaria de genes de longitud igual al número de árboles del conjunto de clasificación y donde el gen $i$ representa la presencia ($=1$) o ausencia ($=0$) del árbol $i$ en el conjunto final.

La función de idoneidad (*fitness*) usada para la identificación de los mejores cromosomas es la precisión del subconjunto (representado por el cromosoma) más un factor de tamaño que tiene en cuenta el número de clasificadores seleccionados. El conjunto de datos utilizado para medir el error de clasificación es el conjunto de entrenamiento. De este modo, tanto el proceso de entrenamiento (*bagging*) como el proceso de selección (AG) se basan en los mismos ejemplos, en el mismo modo en que se han realizado los experimentos presentados en la sección 5.5.2. El factor de tamaño se ha introducido para hacer que se prefieran conjuntos más grandes (aquéllos con más unos). Este sesgo se añade para compensar el hecho de que el mínimo en el conjunto de entrenamiento generalmente se obtiene para subconjuntos menores de clasificadores (ver figuras 5.10–5.14) que en test. No obstante el peso dado a este factor dentro de la función de idoneidad es siempre menor que $1/N$. Esto garantiza que: (i) los cromosomas con menor error siempre tengan mayor idoneidad independientemente del número de clasificadores seleccionados y (ii) si dos cromosomas obtienen el mismo error de clasificación en el conjunto de entrenamiento entonces aquél que incluya más clasificadores tendrá un valor de idoneidad mayor. Esto se ha hecho así porque en caso contrario el AG selecciona subconjuntos demasiado pequeños que no conducen a buenas cotas de error de generalización.

La idoneidad (*fitness*) de un cromosoma $chr$ la mediremos como:

$$Fitness(chr) = 1/(1 + error(chr, L) + count\_zeroes(chr)/(2\,T\,N)) \;, \qquad (5.8)$$

donde $count\_zeroes$ es una función que cuenta el número de ceros en la cadena $chr$ —esto es, el número de elementos no seleccionados— y donde $error$ devuelve el error de clasificación del subconjunto representado por $chr$ en el conjunto de entrenamiento $L$.

Los parametros de configuración del algoritmo genético fueron ajustados en experimentos preliminares y siguen las recomendaciones dadas en [Eiben y Smith, 2003]. Asimismo, la configuración del experimento es muy similar a la utilizada en [Zhou y Tang, 2003].

El número de generaciones y el tamaño de la población es 200 lo que requiere hacer $40\,000$ evaluaciones de la función de idoneidad para cada ejecución. El operador de mutación utilizado es la operación lógica "no" para cada bit (*bit-flip*) con probabilidad $0.005$. Es decir, en cada generación muta 1 bit por individuo en promedio. Para evitar el sesgo posicional se utilizó cruce uniforme (*uniform crossover*) con probabilidad $0.65$. Se aplicó reemplazamiento de la población con elitismo. Es decir, cada generación se substituye por la siguiente y se mantienen dos copias del mejor individuo de la generación actual en la siguiente. La población se inicializó diagonalmente de forma que se representan todos los posibles conjuntos de tamaño uno —con inicialización aleatoria se obtienen resultados en entrenamiento notablemente peores que con la inicialización diagonal. Más concretamente, el individuo $i$ se inicializa con todos sus bits a $0$ excepto su gen $i$ que se inicializa a $1$. Además, en una segunda tanda de experimentos se substituyeron dos cromosomas de la población inicial con la solución dada por el algoritmo reducción de error (el subconjunto con menor error en entrenamiento). Todos estos parámetros quedan recogidos en el cuadro 5.1.

Cuadro 5.1: Configuración del AG

| | |
|---|---:|
| Representación | Cadena binaria de 200-Bits |
| Recombinación | Cruce uniforme |
| Probabilidad de cruce | 0.65 |
| Mutación | Operación lógica "no"para cada bit (*Bit flip*) |
| Probabilidad de mutación | 0.005 por bit |
| Selección de progenitores | Proporcional a la idoneidad |
| Selección de supervivientes | Reemplazamiento de la población con elitismo |
| Tamaño de la población | 200 |
| Número de vástagos | 200 |
| Inicialización | Diagonal (más la solución dada por reducción de error) |
| Condición de parada | 200 épocas |

Hemos aplicado AG a dos conjuntos diferentes de la colección de problemas de UCI [Blake y Merz, 1998]: *Pima Indian Diabetes* y *Waveform*. Se han utilizado los mismos conjuntos de *bagging* generados para las experimentaciones de la sección 5.5.2 compuestos de 200 árboles CART y podados usando validación cruzada de $10$ particiones. Para cada problema se hicieron, por tanto, 100 ejecuciones usando las mismas particiones entre entrenamiento y test que las de los experimentos de la sección 5.5.2 y descritas en el cuadro 5.6. Para cada ejecución se han seguido los siguientes pasos:

1. Seleccionar el mejor subconjunto aplicando AG.
2. Aplicar el algoritmo de reducción de error.
3. Seleccionar el mejor subconjunto aplicando AG incluyendo en la población original dos cromosomas con la mejor solución obtenida en el paso 2.

Los resultados de estas pruebas se muestran en los cuadros 5.2 y 5.3 para los conjuntos de *Pima Indian Diabetes* y *Waveform* respectivamente. Estos cuadros muestran el error medio alcanzado en entrenamiento y test, y el número medio de clasificadores seleccionados para las diferentes inicializaciones: inicialización diagonal (mostrado como "AG") e inicialización con la solución dada por el método de reducción de error ("AG-RE"). La solución obtenida con el algoritmo de reducción de error, esto es, la que alcanza menor error en entrenamiento también se muestra en el cuadro en la columna "RE".

Cuadro 5.2: Resultados para *Pima Indian Diabetes* usando AG y reducción de error

| | *Bagging* | AG | RE | AG-RE |
|---|---|---|---|---|
| Entrenamiento | 20.8 | 13.0 | 10.3 | 10.3 |
| Test | 24.9 | 26.0 | 25.3 | 25.3 |
| No. árboles | 200 | 8.5 | 10.4 | 10.4 |

Cuadro 5.3: Resultados para *Waveform* usando AG y reducción de error

| | *Bagging* | AG | RE | AG-RE |
|---|---|---|---|---|
| Entrenamiento | 10.5 | 1.28 | 0.607 | 0.557 |
| Test | 22.8 | 20.0 | 20.2 | 20.0 |
| No. árboles | 200 | 47.7 | 35.0 | 39.0 |

De los cuadros 5.2 y 5.3 se puede observar que:

- El algoritmo codicioso de reducción de error obtiene menor error en entrenamiento para ambos conjuntos que el AG que comienza su proceso de optimización con una población de individuos cada uno de los cuales correspondiente a un conjunto distinto con un único clasificador (inicialización diagonal).
- Mejoras muy pequeñas o incluso ninguna mejora en absoluto se obtienen cuando se inicializa el AG con la solución dada por el método de reducción de error. En una sola de las 100 ejecuciones de *Pima Indian Diabetes* el AG fue capaz de incrementar la reducción del error de entrenamiento. Esto se logró en 13 de las 100 ejecuciones

para el conjunto *Waveform* y en 72 ocasiones se consiguió aumentar el tamaño del subconjunto.

- El error de generalización para ambas ejecuciones de AG y para la ordenación por reducción de error son muy similares. El algoritmo de reducción de error obtiene subconjuntos con menor error de generalización que AG cuando se utiliza el 20 % de clasificadores en lugar del número de clasificadores que tienen menor error en entrenamiento (ver sección 5.8).

Todas estas observaciones apoyan las conclusiones obtenidas de los experimentos realizados usando búsqueda exhaustiva y nos permiten decir que las heurísticas codiciosas propuestas (i) tienen buena capacidad de optimización —todas las heurísticas excepto ordenación por ángulos se pueden considerar como un mismo algoritmo de optimización que minimiza/maximiza distintas funciones—, dado que el algoritmo de ordenación obtiene subconjuntos con un error menor en entrenamiento y usando una fracción del tiempo necesario para ejecutar AG y (ii) los subconjuntos seleccionados por las heurísticas codiciosas tienen buena capacidad de generalización, al menos para el algoritmo reducción de error que obtiene un resultado mejor que AG en el conjunto de test en los problemas de clasificación estudiados cuando se usa el 20 % de clasificadores. Además, una ventaja adicional derivada del uso de heurísticas de ordenación es que se obtiene una secuencia de soluciones en vez de una solución única pudiendo ajustarse a potenciales límites de tamaño o velocidad de clasificación de forma directa.

En todo caso, estas conclusiones hay que tomarlas con cautela ya que la eficacia de los AG puede ser muy distinta si se usan diferentes representaciones de los individuos o valores de los parámetros utilizados en la optimización.

## 5.5. Resultados experimentales

### 5.5.1. Efecto del número de clasificadores del conjunto de partida en la ordenación

Se ha realizado un experimento para evaluar cómo el número inicial de clasificadores en el conjunto de *bagging* original afecta al funcionamiento de los conjuntos ordenados. Para este experimento se han generado conjuntos de clasificadores compuestos por 1000 clasificadores individuales que han sido ordenados teniendo en cuenta sólo los primeros 11, 25, 51, 75, 101, 151, 201, 251, 501, 751 y 1000 árboles respectivamente. Se han realizado 100 ejecuciones usando los mismos tamaños de particiones definidos en el cuadro 5.6. Se han usado los problemas *Pima Indian Diabetes* y *Waveform* y se han aplicado las heurísticas de ordenación de: reducción de error, minimización de distancias de margen y ordenación basada en *boosting*.

Los resultados se pueden ver en las figuras 5.8 y 5.9 para *Pima Indian Diabetes* y *Waveform* respectivamente. Estas figuras muestran por columnas los resultados medios de entrenamiento (*primera columna*) y test (*segunda columna*) y por filas los resultados usando los algoritmos: reducción de error (*primera fila*), minimización de distancias de margen (*segunda fila*) y ordenación basada en *boosting* (*tercera fila*). En la última fila se muestra la evolución de los errores mínimos obtenidos por cada heurística de ordenación. Los puntos se han unido con rectas que sirven como guías visuales para trazar más fácilmente la evolución de los errores mínimos en función del tamaño del conjunto de partida: el mínimo de la ordenación que ha usado 11 clasificadores se ha unido con el que ordena 25 elementos, que a su vez se ha enlazado con el de 51, etc. Por ello se observa que cuando una ordenación alcanza un mínimo que necesita menos clasificadores que el mínimo de otra ordenación que parte de un número menor de clasificadores, la línea retrocede .

Las figuras 5.8 y 5.9 muestran que en entrenamiento, inicialmente, las ordenaciones presentan una tendencia de bajada muy similar. A medida que aumenta el número de clasificadores las curvas se van separando: las correspondientes a conjuntos con un número total menor generalmente comienzan a ascender antes que las correspondientes a conjuntos iniciales mayores. Las curvas apenas se cruzan unas con otras sino que se van envolviendo. Esto es razonable (sobre todo para el método reducción de error) teniendo en cuenta que se está usando un conjunto de clasificadores incremental (todos los clasificadores de la ordenación que usa, por ejemplo, 251 están en la de 501, 751 y 1000) y que se minimiza una función basándose en una medida sobre los datos de entrenemiento.

Las curvas de error de test no son tan homogéneas como las correspondientes a error de entrenamiento (sobre todo en el conjunto *Pima Indian Diabetes*). Las curvas muestran una bajada inicial muy parecida. Tras este descenso las curvas se separan progresivamente. La separación de las distintas curvas con respecto a la línea de bajada principal es distinta para ordenación por reducción de error y ordenación basada en *boosting* que para minimización de distancias de margen. Las dos primeras heurísticas tienen un comportamiento similar al observado en entrenamiento: primero se separan por arriba aquellas curvas correspondientes a conjuntos con un número inicial menor de clasificadores. Para la heurística de reducción de distancias de margen las curvas se invierten: en las fases iniciales están por encima las curvas correspondientes a conjuntos con un número inicial mayor de elementos. Aun así el punto mínimo alcanzado tiende a ser inferior para las curvas correspondientes a conjuntos de clasificadores con más elementos.

Estos resultados muestran cómo los mínimos en el conjunto de entrenamiento aparecen para un número mucho más pequeño que en test. Como caso extremo está el algoritmo de minimización de distancias de margen para 1000 árboles en el conjunto *Pima Indian Diabetes*. En este problema el mínimo en entrenamiento está en 5 árboles mientras que en test está por encima de 170.

Los mínimos alcanzados se muestran en las gráficas inferiores de las figuras 5.8 y 5.9.

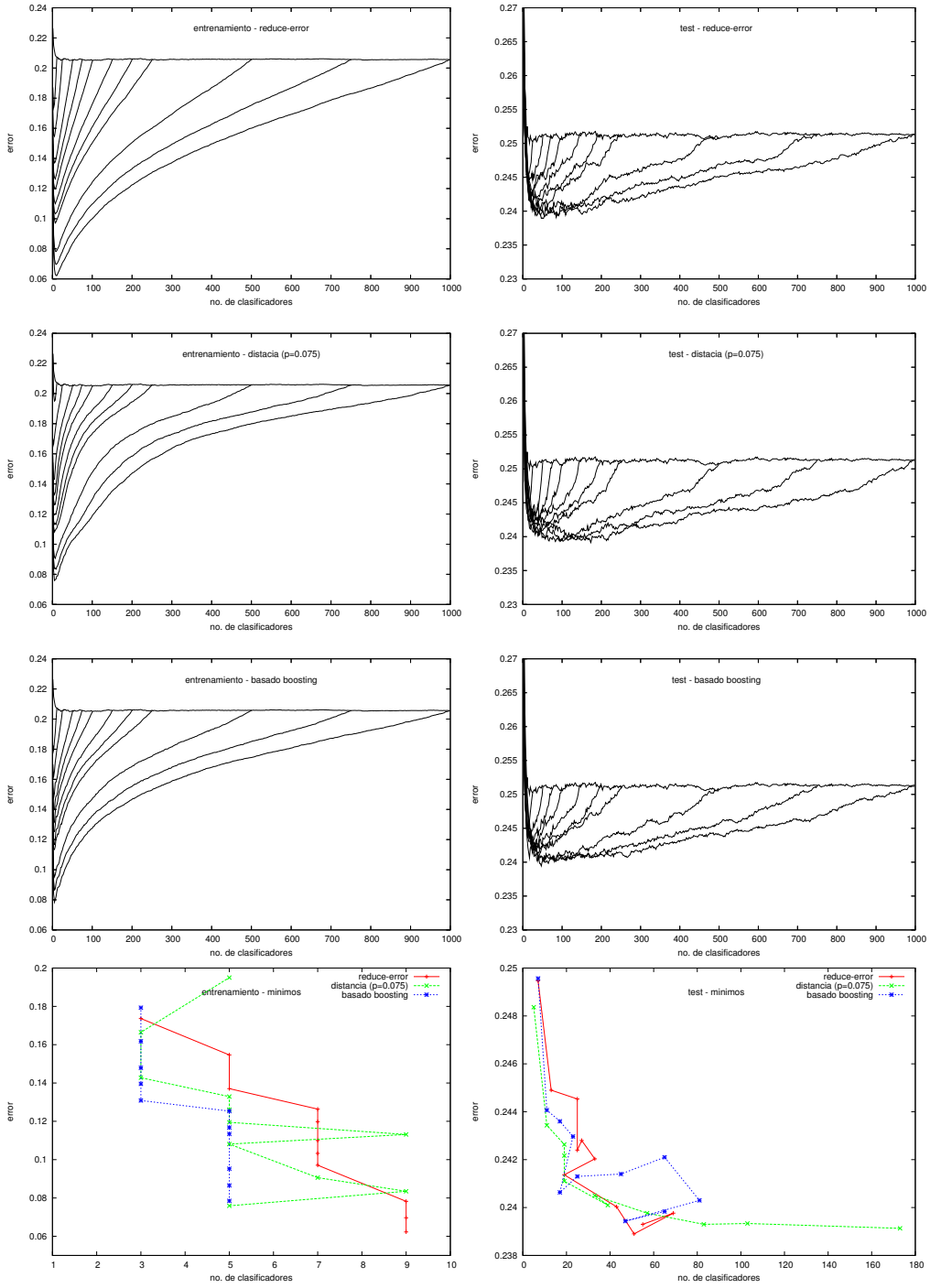



Figura 5.8: Error de entrenamiento y test para *Pima Diabetes* de *bagging* y ordenado usando: 11, 25, 51, 75, 101, 151, 201, 251, 501, 751 y 1000 árboles. (Más detalles en el texto)

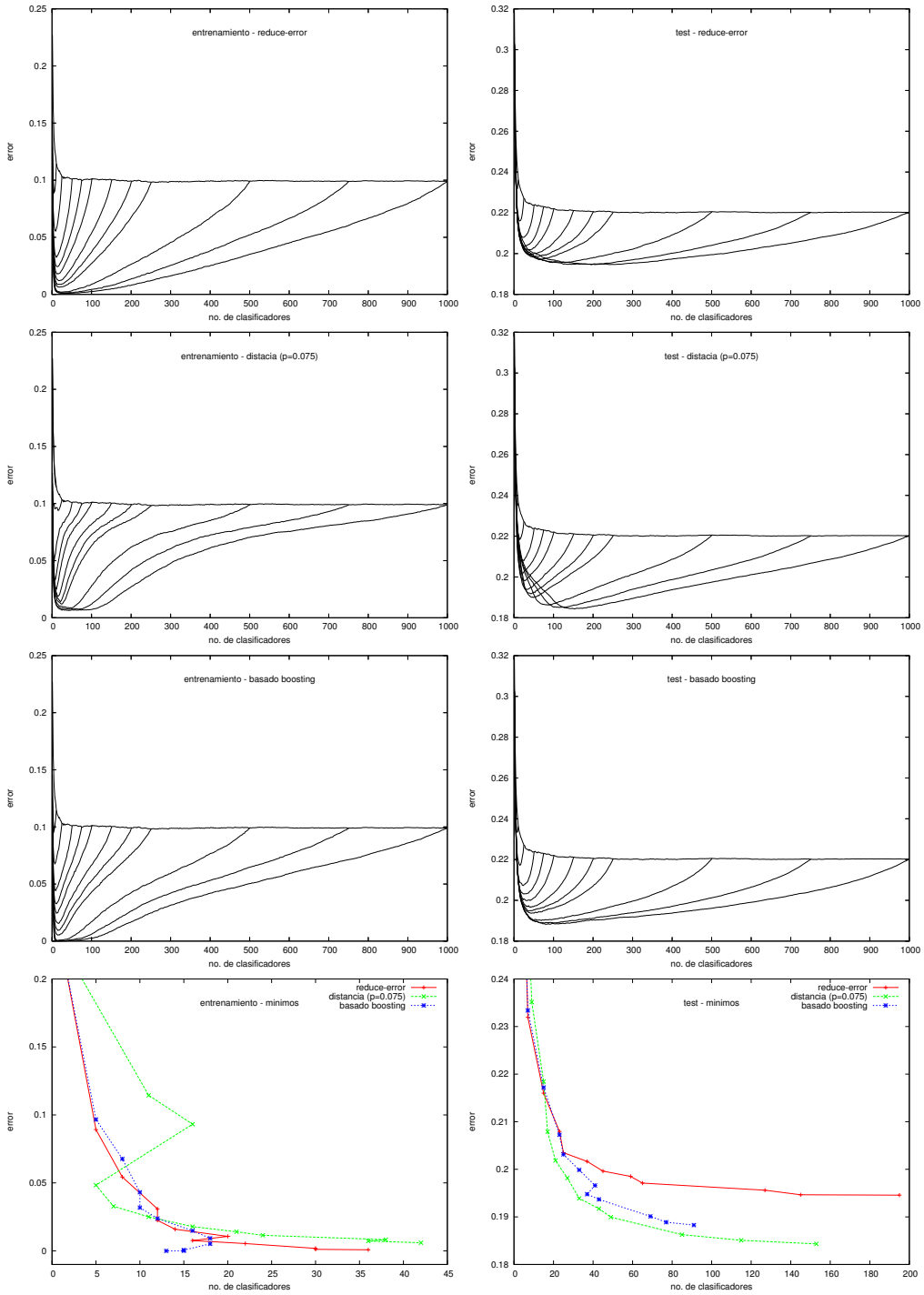



Figura 5.9: Error de entrenamiento y test para *Waveform* de *bagging* y ordenado usando: 11, 25, 51, 75, 101, 151, 201, 251, 501, 751 y 1000 árboles. (Más detalles en el texto)

Asimismo en los cuadros 5.4 y 5.5 se dan los valores de los mínimos en test. Para cada configuración se muestran los errores mínimos, el número de clasificadores utilizados y porcentaje de clasificadores con respecto al número inicial de clasificadores. Se puede observar una tendencia del error de test a saturarse. Esta tendencia es más clara para el

Cuadro 5.4: Error medio mínimo en test y número de clasificadores necesarios para alcanzar el mínimo para distintos tamaños iniciales del conjunto para *Pima Indian Diabetes*

| Tamaño | Reduc. error | | Dist. ($p = 0.075$) | | B. *boosting* | |
|---|---|---|---|---|---|---|
| inicial | error | no. clasf. | error | no. clasf. | error | no. clasf. |
| 11 | 24.9 | 7 (63.6 %) | 24.8 | 5 (45.5 %) | 25.0 | 7 (63.6 %) |
| 25 | 24.5 | 13 (52.0 %) | 24.3 | 11 (44.0 %) | 24.4 | 11 (44.0 %) |
| 51 | 24.5 | 25 (49.0 %) | 24.3 | 19 (37.3 %) | 24.4 | 17 (33.3 %) |
| 75 | 24.2 | 25 (33.3 %) | 24.2 | 19 (25.3 %) | 24.3 | 23 (30.7 %) |
| 101 | 24.3 | 27 (26.7 %) | 24.1 | 19 (18.8 %) | 24.1 | 17 (16.8 %) |
| 151 | 24.2 | 33 (21.9 %) | 24.0 | 39 (25.8 %) | 24.1 | 25 (16.6 %) |
| 201 | 24.1 | 19 (9.5 %) | 24.0 | 33 (16.4 %) | 24.1 | 45 (22.4 %) |
| 251 | 24.0 | 43 (17.1 %) | 24.0 | 57 (22.7 %) | 24.2 | 65 (25.9 %) |
| 501 | 23.9 | 51 (10.2 %) | 23.9 | 83 (16.6 %) | 24.0 | 81 (16.2 %) |
| 751 | 24.0 | 69 (9.2 %) | 23.9 | 103 (13.7 %) | 23.9 | 47 (6.3 %) |
| 1000 | 23.9 | 55 (5.5 %) | 23.9 | 173 (17.3 %) | 24.0 | 65 (6.5 %) |

Cuadro 5.5: Error medio mínimo en test y número de clasificadores necesarios para alcanzar el mínimo para distintos tamaños iniciales del conjunto para *Waveform*

| Tamaño | Reduc. error | | Dist. ($p = 0.075$) | | B. *boosting* | |
|---|---|---|---|---|---|---|
| inicial | error | no. clasf. | error | no. clasf. | error | no. clasf. |
| 11 | 23.2 | 7 (63.6 %) | 23.5 | 9 (81.8 %) | 23.3 | 7 (63.6 %) |
| 25 | 21.6 | 15 (60.0 %) | 21.8 | 15 (60.0 %) | 21.7 | 15 (60.0 %) |
| 51 | 20.8 | 23 (45.1 %) | 20.8 | 17 (33.3 %) | 20.7 | 23 (45.1 %) |
| 75 | 20.3 | 25 (33.3 %) | 20.2 | 21 (28.0 %) | 20.3 | 25 (33.3 %) |
| 101 | 20.2 | 37 (36.6 %) | 19.8 | 27 (26.7 %) | 20.0 | 33 (32.7 %) |
| 151 | 20.0 | 45 (29.8 %) | 19.4 | 33 (21.9 %) | 19.7 | 41 (27.2 %) |
| 201 | 19.8 | 59 (29.4 %) | 19.2 | 43 (21.4 %) | 19.5 | 37 (18.4 %) |
| 251 | 19.7 | 65 (25.9 %) | 19.0 | 49 (19.5 %) | 19.4 | 43 (17.1 %) |
| 501 | 19.6 | 127 (25.3 %) | 18.6 | 85 (17.0 %) | 19.0 | 69 (13.8 %) |
| 751 | 19.5 | 145 (19.3 %) | 18.5 | 115 (15.3 %) | 18.9 | 77 (10.3 %) |
| 1000 | 19.5 | 195 (19.5 %) | 18.4 | 153 (15.3 %) | 18.8 | 91 (9.1 %) |

problema *Waveform*. Asimismo se puede observar cómo el número de clasificadores necesarios para alcanzar el error mínimo en test aumenta a medida que se aumenta el tamaño inicial del conjunto. Este aumento es más lento que el aumento del tamaño del conjunto inicial como podemos observar de la tendencia a la baja del porcentaje de clasificadores necesarios. No parece que se puedan obtener reducciones del error que justifiquen la ordenación a partir de conjuntos iniciales con más clasificadores. Además, aumentar el número inicial de clasificadores hace que aumente también el tamaño de los subconjuntos que es necesario para alcanzar el mínimo del error.

### 5.5.2. Experimentos en bases de datos

Se han realizado experimentos en 18 bases de datos para mostrar la eficacia de los clasificadores obtenidos con las heurísticas de ordenación y poda propuestas. Dos de las bases de datos son conjuntos sintéticos (*Waveform* y *Twonorm* propuestos en [Breiman *et al.*, 1984; Breiman, 1996b]. Los problemas restantes están incluídos en la colección de problemas de UCI [Blake y Merz, 1998]: *Audio, Australian Credit, Breast Cancer Wisconsin, Pima Indian Diabetes, German Credit, Heart, Horse Colic, Ionosphere, Labor Negotiations, New-Thyroid, Image Segmentation, Sonar, Tic-tac-toe, Vehicle, Vowel* y *Wine*. En el cuadro 5.6 se muestra el número de ejemplos usados para entrenar y para test, así como el número de atributos y el número de clases para cada conjunto de datos. Más detalles sobre las distintas bases de datos se pueden encontrar en el apéndice A.

Para cada problema se llevaron a cabo $100$ experimentos. Cada experimento conlleva los siguientes pasos:

1. Generación de una partición aleatoria estratificada de los datos entre entrenamiento y test (ver cuadro 5.6 para los tamaños). Para los conjuntos sintéticos este paso se realizó por muestreo aleatorio a partir de las distribuciones reales que son conocidas. Las particiones utilizadas son las mismas (en los problemas comunes) que las usadas en el capítulo 4.

2. Creación de un conjunto *bagging* de $200$ árboles CART podados usando la poda de coste-complejidad con validación cruzada de 10 particiones (ver [Breiman *et al.*, 1984] para más detalles o sección 2.2).

3. Ordenación de los árboles de decisión usando los 5 procedimientos descritos en la sección anterior (reducción de error, medida de complementariedad, minimización de distancias de margen, ordenación por ángulos y ordenación basada en *boosting*), usando como conjunto de selección los mismos datos de entrenamiento usados para generar el conjunto. Para el procedimiento de Minimización por distancia de margen se ha elegido un valor de $p = 0.075$ basándonos en experimentos preliminares. Se obtienen resultados similares con $p = 0.05$ y $p = 0.25$. Aunque con $p = 0.25$ es necesario seleccionar un número mayor de clasificadores.

Cuadro 5.6: Conjuntos de datos usados en los experimentos

| Problema | Entrenamiento | Test | Atributos | Clases |
|---|---|---|---|---|
| Audio | 140 | 86 | 69 | 24 |
| Australian | 500 | 190 | 14 | 2 |
| Breast W. | 500 | 199 | 9 | 2 |
| Diabetes | 468 | 300 | 8 | 2 |
| German | 600 | 400 | 20 | 2 |
| Heart | 170 | 100 | 13 | 2 |
| Horse-Colic | 244 | 124 | 21 | 2 |
| Ionosphere | 234 | 117 | 34 | 2 |
| Labor | 37 | 20 | 16 | 2 |
| New-thyroid | 140 | 75 | 5 | 3 |
| Segment | 210 | 2100 | 19 | 7 |
| Sonar | 138 | 70 | 60 | 2 |
| Tic-tac-toe | 600 | 358 | 9 | 2 |
| Twonorm | 300 | 5000 | 20 | 2 |
| Vehicle | 564 | 282 | 18 | 4 |
| Vowel | 600 | 390 | 10 | 11 |
| Waveform | 300 | 5000 | 21 | 3 |
| Wine | 100 | 78 | 13 | 3 |

4. Evaluación de los conjuntos ordenados en el conjunto de test usando $10\,\%$, $20\,\%$ y $40\,\%$ de los clasificadores (esto es, podas del $90\,\%$, $80\,\%$ y $60\,\%$) del conjunto original.

Estos resultados se presentan gráficamente en las figuras 5.10–5.14 en las que se muestra el error medio de entrenamiento y test en función del número de clasificadores para todos los problemas estudiados. Las distintas curvas corresponden a distintos ordenes de agregación: la línea continua roja corresponde al orden inicial de *bagging*, que es aleatorio. Las líneas discontinuas (y con distinto color) corresponden a ordenaciones realizadas con: reducción de error (verde), complementariedad (azul oscuro), minimización de distancia de margen con $p = 0.075$ (rosa), ordenación por ángulos (azul claro) y ordenación basada en *boosting* (negro). Estas figuras ilustran la dependencia del error de clasificación con respecto al número de clasificadores. Tal como se esperaba, en los conjuntos ordenados aleatoriamente el error disminuye generalmente de forma monótona a medida que se incrementa el número de clasificadores, hasta que alcanza asintóticamente un valor constante de error. Por el contrario, los conjuntos ordenados presentan curvas para los errores de test con un mínimo para un número intermedio de clasificadores. Además, para todos subconjuntos, exceptuando los más pequeños, el error de generalización de las curvas ordenadas está por

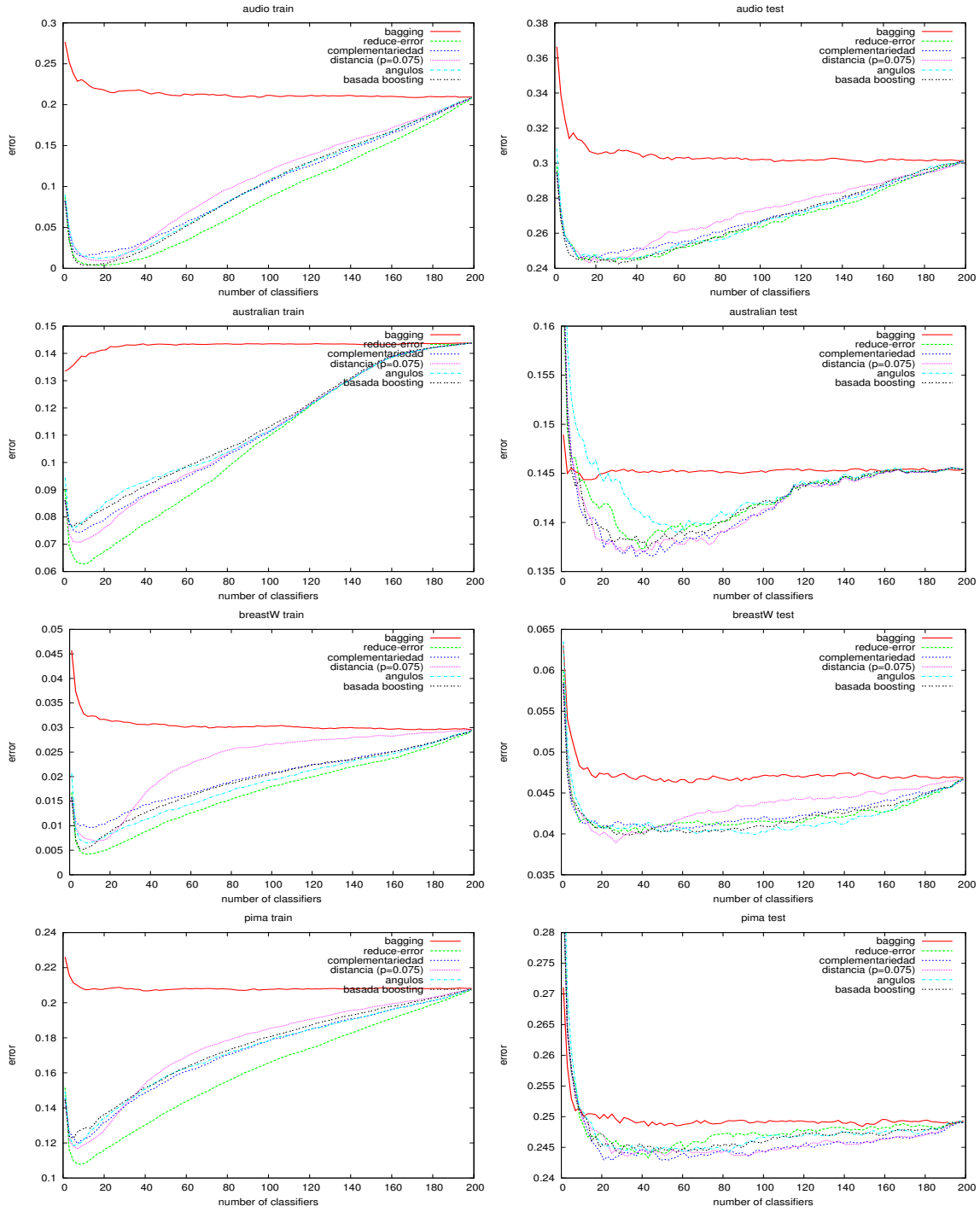



Figura 5.10: Error de entrenamiento y test para *Audio*, *Australian*, *Breast Cancer* y *Pima Indian Diabetes*

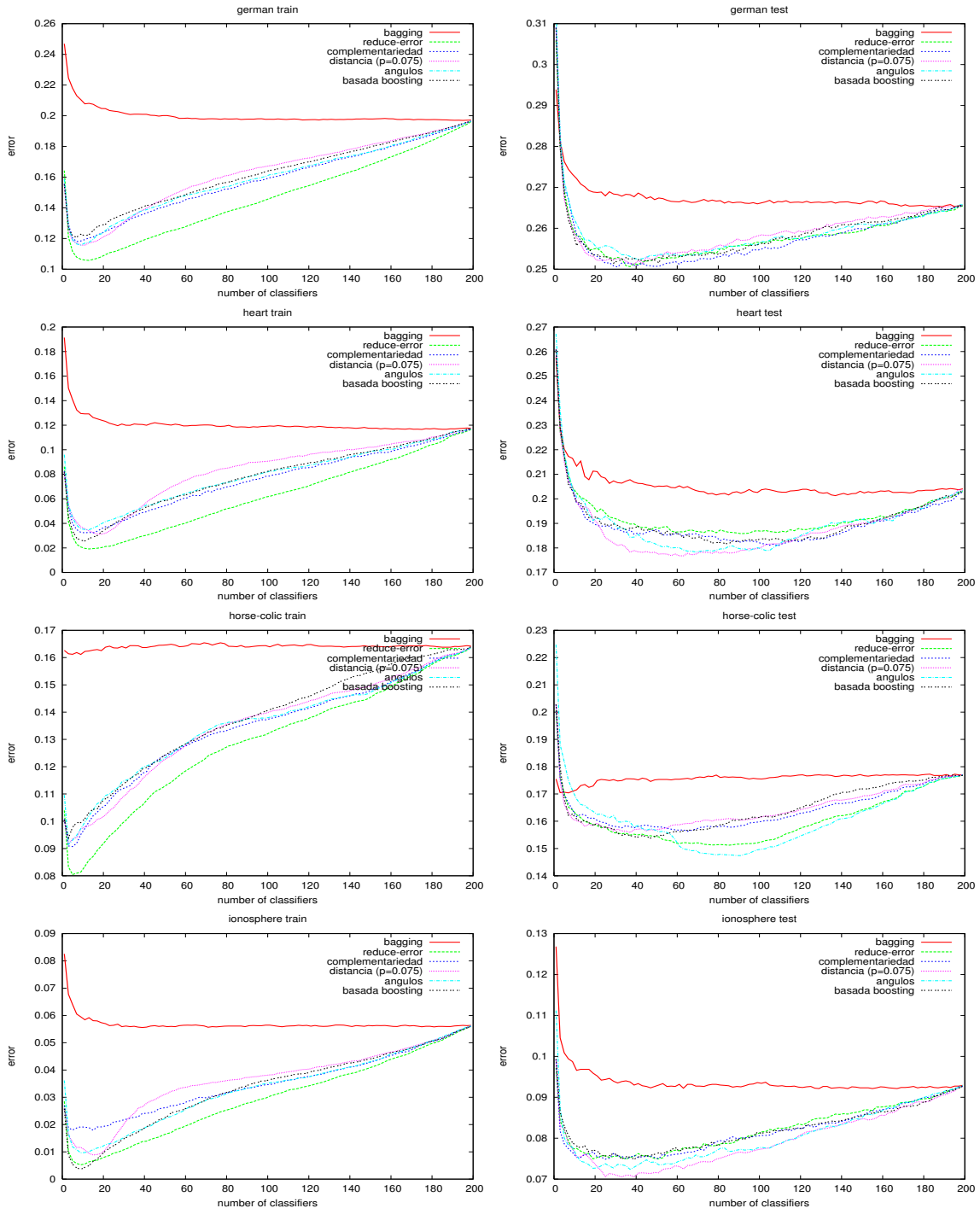



Figura 5.11: Error de entrenamiento y test para *German Credit*, *Heart*, *Horse-colic* e *Ionosphere*

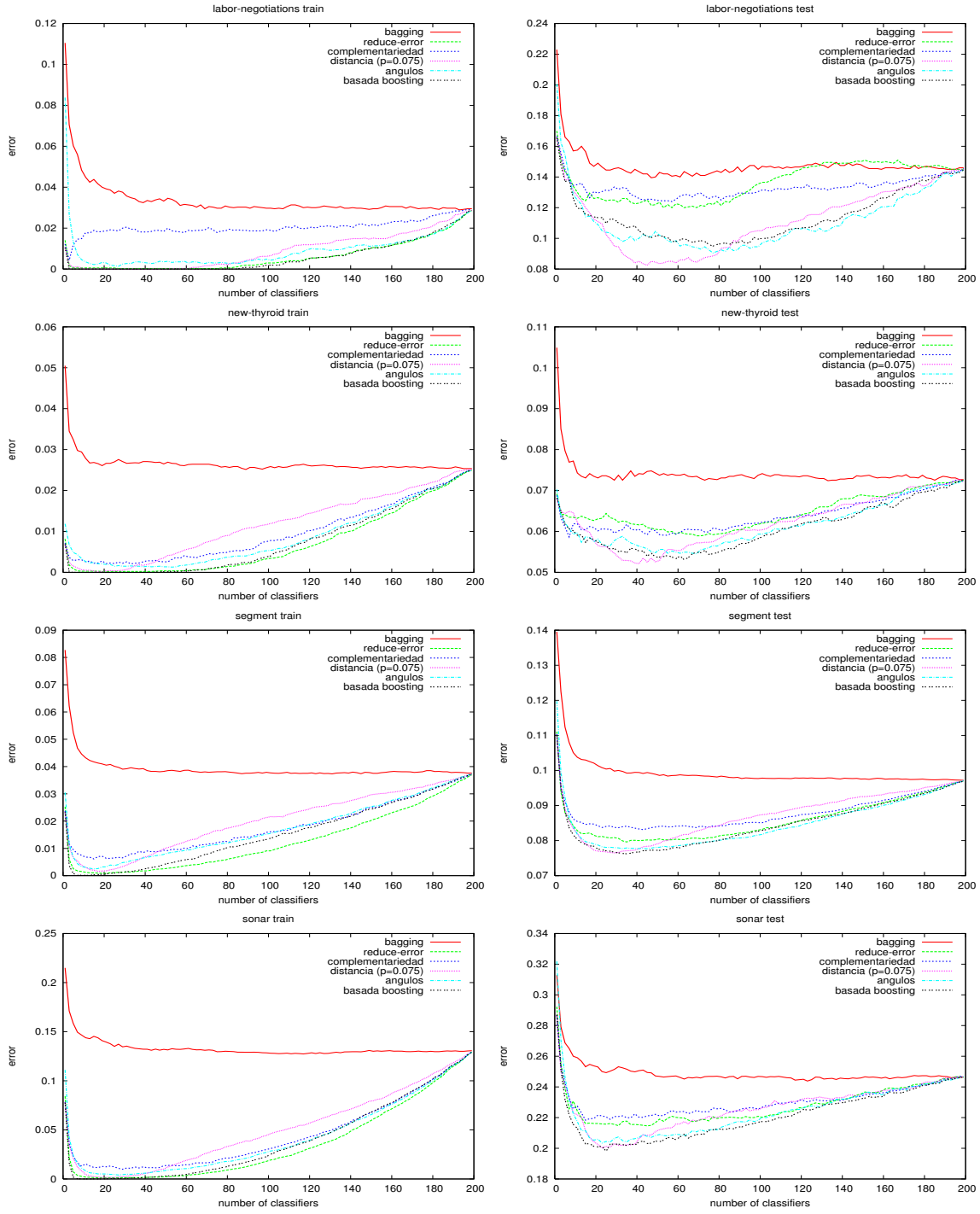



Figura 5.12: Error de entrenamiento y test para *Labor Negotiations*, *New-Thyroid*, *Image Segmentation* y *Sonar*

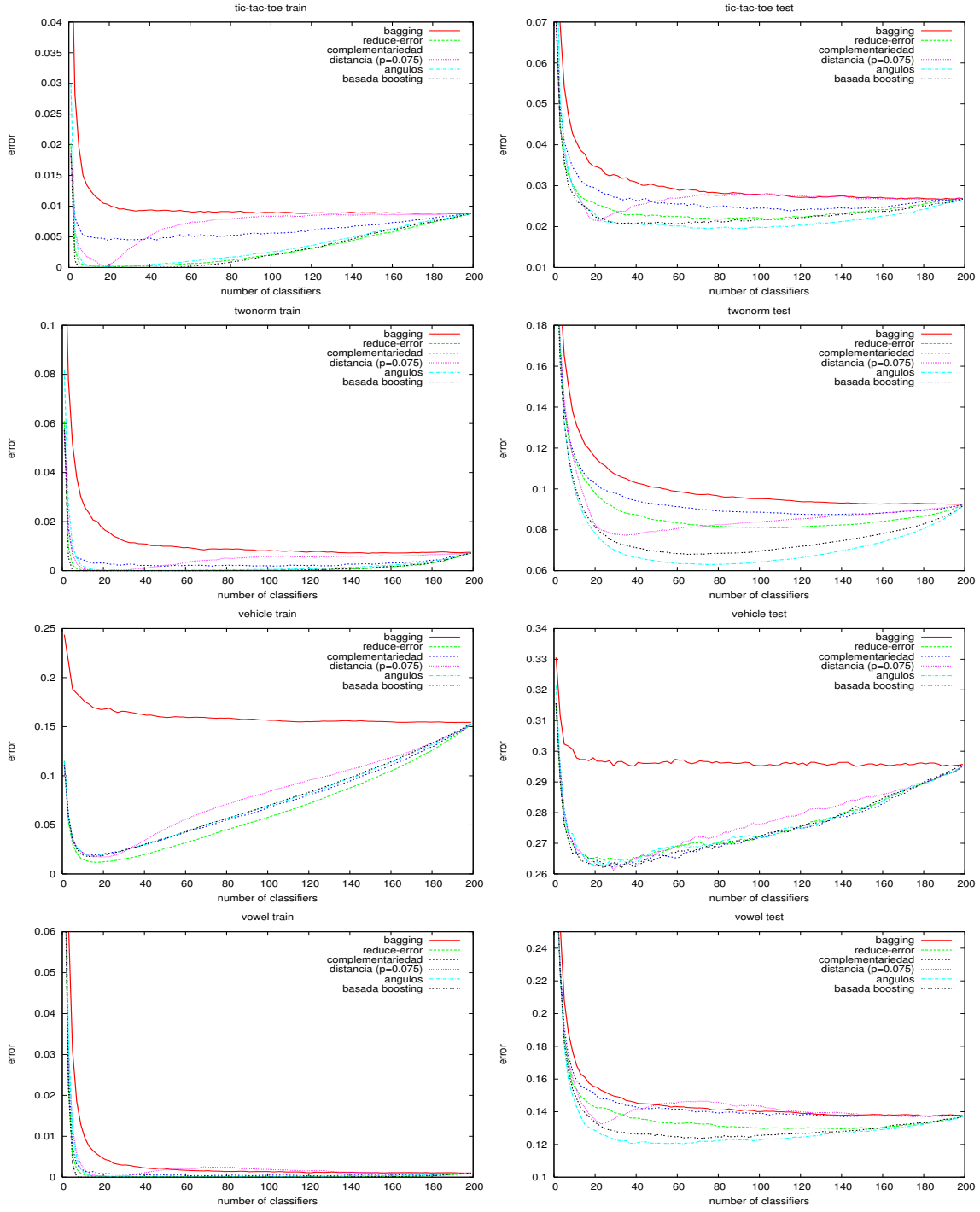



Figura 5.13: Error de entrenamiento y test para *Tic-tac-toe*, *Twonorm*, *Vehicle* y *Vowel*

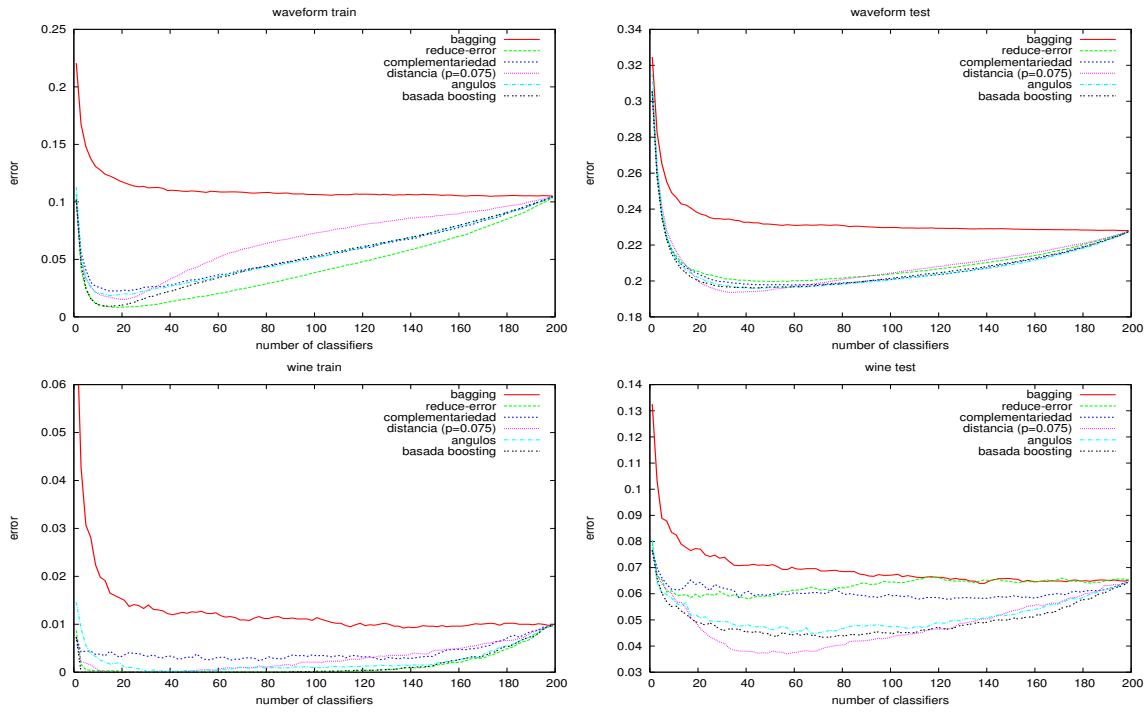



Figura 5.14: Error de entrenamiento y test para *Waveform* y *Wine*

debajo del error asintótico de *bagging* (figuras 5.10–5.14 (*columna derecha*)). Esto hace que sea fácil la selección de un subconjunto que mejore al conjunto completo. Por otro lado, el mínimo en el conjunto de entrenamiento suele obtenerse para un número menor de clasificadores que el mínimo en el conjunto de test. Esto tiene como consecuencia que sea difícil la selección, a partir de los datos de entrenamiento, de un porcentaje de poda que produzca el mejor error de generalización posible.

También es importante resaltar que en los conjuntos *Australian* (fig. 5.10 segunda fila) y *Horse-colic* (fig. 5.11 tercera fila) la curva de *bagging* es prácticamente constante (subiendo incluso en entrenamiento en *Australian*) y apenas consigue mejorar el resultado obtenido por un clasificador. A pesar de esto, la ordenación funciona en estos conjuntos y conduce a una mejoría apreciable con respecto al error de *bagging*. Por otro lado, en algunos conjuntos donde el error de *bagging* en entrenamiento es muy bajo, varias de las heurísticas no consiguen obtener mejoras importantes con respecto a *bagging*. Esto sucede principalmente en los conjuntos *Tic-tac-toe*, *Twonorm* y *Vowel* (fig. 5.13 primera, segunda y última fila respectivamente). En este último conjunto de datos la minimización de distancias de margen muestra incluso un error de test superior a *bagging* a partir de 55 clasificadores.

Los cuadros 5.7 y 5.8 presentan los errores de entrenamiento y test obtenidos por las distintas heurísticas de ordenación y para los distintos valores de poda y conjuntos de datos

seleccionados. Los valores mostrados son el promedio sobre 100 ejecuciones. La primera columna muestra el nombre del problema de clasificación. En la segunda columna se dan los errores de *bagging* usando todos los clasificadores. La desviación estándar se muestra tras el signo $\pm$. Los siguientes grupos de columnas presentan el error medio para los métodos reducción de error, medida de complementariedad, minimización de distancias de margen ($p = 0.075$), ordenación por ángulos y ordenación basada en *boosting* en conjuntos de tamaños 10 %, 20 % y 40 % del conjunto original respectivamente. Las desviaciones estándar no se muestran. En general, son menores que la obtenida por *bagging* para cada conjunto. Como excepciones están *Australian*, *Pima Indian Diabetes* y *German Credit* en los que generalmente el error para el conjunto ordenado tiene una desviación estándar mayor que los errores en *bagging* aunque las diferencias no son grandes, $0.1$–$0.3$ puntos. Asimismo los cuadros 5.7 y 5.8 muestran en negrita el mejor resultado para cada conjunto y subrayado el segundo mejor resultado (siempre que sólo haya un único mejor resultado).

En el cuadro 5.8 podemos observar que minimización de distancia de margen es el método que da mejores resultados, obteniendo el mejor resultado en 10 de los 18 problemas y el segundo mejor en otras 2 bases de datos. Le siguen ordenación basada en *boosting* (5 mejores + 3 segundos), ordenación de ángulos (4+4), complementariedad (4+0) y reducción de error (3+2). Asimismo, se puede ver cómo los métodos propuestos generalmente reducen el error de clasificación del conjunto completo (hay algunos valores mayores en *Twonorm* y *Vowel*). Además estas mejoras se logran para un gran rango de valores de poda. El error de generalización normalmente se sitúa por debajo del error asintótico de *bagging* empezando en subconjuntos pequeños, conteniendo menos de un 10 % de los clasificadores originales. Otro hecho importante es que a menudo el método que muestra mejores tasas de generalización (reducción de distancias de margen usando $20\,\%$ de los clasificadores) no coincide con el método que obtiene los mejores resultados en entrenamiento (reducción de error con $10\,\%$ de los clasificadores). Como ejemplo extremo está el conjunto de datos *Heart* donde el peor resultado en entrenamiento (minimización de distancias de margen usando $40\,\%$ de los clasificadores) corresponde al mejor error de generalización. Por contra, el método de poda que obtiene el mejor error en entrenamiento (reducción de error con $10\,\%$ de los clasificadores) presenta el peor resultado de entre los distintos métodos y podas en test. Estos resultados muestran cómo un método que se basa exclusivamente en la reducción del error de entrenamiento, como el algoritmo de ordenación por reducción de error, tiende a sobreajustar más que otros métodos de ordenación.

En el cuadro 5.9 se muestran los resultados de aplicar la prueba-t de Student pareada de dos colas para comparar *bagging* con respecto los distintos métodos de ordenación y porcentajes de poda mostrados en el cuadro 5.8. Se han resaltado en negrita diferencias con un valor-p inferior a $0.5\,\%$. Asimismo se han recuadrado los resultados favorables a *bagging*. En este cuadro se puede observar cómo los resultados son abrumadores a favor de las heurísticas de ordenación y poda propuestas. *Bagging* sólo obtiene resultados significativos a su favor con respecto a algunas heurísticas en los conjuntos *Twonorm* y *Vowel*.

Cuadro 5.7: Media del error de entrenamiento en % para conjuntos compuestos de 10 %, 20 % y 40 % clasificadores. El mejor resultado se muestra en **negrita**. El segundo mejor subrayado

| | *Bagging* | **Reduc. error** | | | **Complemen.** | | | **Dist** ($p = 0.075$) | | | **Ángulos** | | | **B. *boosting*** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tamaño | 100 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % |
| Audio | 21.0±4.6 | **0.4** | 1.4 | 6.1 | 2.1 | 3.4 | 8.2 | 0.9 | 3.3 | 9.7 | 1.3 | 2.7 | 8.2 | 0.6 | 2.5 | 8.1 |
| Australian | 14.4±0.9 | **6.8** | 7.8 | 9.9 | 8.0 | 8.9 | 10.3 | 7.7 | 8.8 | 10.4 | 8.6 | 9.3 | 10.4 | 8.3 | 9.2 | 10.5 |
| Breast W. | 3.0±0.6 | **0.6** | 0.9 | 1.6 | 1.1 | 1.5 | 1.9 | 0.8 | 1.8 | 2.6 | 0.9 | 1.2 | 1.7 | 0.9 | 1.3 | 1.9 |
| Diabetes | 20.8±1.2 | **11.7** | 13.1 | 15.6 | 13.2 | 14.9 | 17.1 | 12.9 | 15.5 | 17.9 | 13.5 | 15.1 | 17.2 | 13.7 | 15.2 | 17.3 |
| German | 19.7±1.6 | **10.9** | 12.0 | 13.8 | 12.5 | 13.7 | 15.2 | 12.2 | 14.0 | 16.1 | 12.5 | 13.9 | 15.4 | 12.9 | 14.2 | 15.7 |
| Heart | 11.8±2.9 | **2.1** | 3.0 | 5.2 | 3.7 | 4.9 | 7.0 | 3.2 | 5.7 | 8.5 | 4.1 | 5.5 | 7.3 | 3.5 | 5.3 | 7.4 |
| Horse-Colic | 16.4±2.0 | **9.3** | 10.8 | 12.8 | 10.6 | 11.8 | 13.4 | 10.2 | 11.7 | 13.6 | 10.9 | 12.0 | 13.6 | 10.8 | 12.0 | 13.5 |
| Ionosphere | 5.7±1.5 | **0.8** | 1.4 | 2.6 | 1.9 | 2.4 | 3.2 | 1.1 | 2.8 | 3.6 | 1.3 | 1.9 | 3.2 | 1.0 | 1.9 | 3.2 |
| Labor | 2.9±2.4 | 0.1 | **0.0** | 0.1 | 1.9 | 1.8 | 1.9 | **0.0** | **0.0** | 0.2 | 0.3 | 0.3 | 0.3 | **0.0** | **0.0** | **0.0** |
| New-thyroid | 2.6±1.3 | **0.0** | **0.0** | 0.2 | 0.2 | 0.3 | 0.5 | **0.0** | 0.2 | 0.9 | 0.2 | 0.2 | 0.4 | **0.0** | **0.0** | 0.2 |
| Segment | 3.8±1.6 | **0.1** | 0.2 | 0.6 | 0.6 | 0.9 | 1.2 | 0.2 | 0.7 | 1.8 | 0.3 | 0.7 | 1.2 | **0.1** | 0.3 | 1.1 |
| Sonar | 13.1±3.8 | 0.1 | 0.2 | 0.9 | 1.2 | 1.1 | 2.1 | 0.2 | 0.6 | 3.4 | 0.5 | 0.6 | 1.8 | **0.0** | 0.2 | 1.3 |
| Tic-tac-toe | 0.9±0.3 | **0.0** | **0.0** | 0.1 | 0.5 | 0.5 | 0.5 | 0.1 | 0.5 | 0.8 | **0.0** | **0.0** | 0.2 | **0.0** | **0.0** | 0.1 |
| Twonorm | 0.7±1.1 | **0.0** | **0.0** | **0.0** | 0.3 | 0.2 | 0.2 | **0.0** | 0.1 | 0.5 | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** |
| Vehicle | 15.4±2.7 | **1.3** | 2.0 | 4.6 | 2.1 | 3.0 | 5.6 | 1.7 | 3.5 | 7.2 | 2.0 | 3.1 | 5.7 | 2.0 | 3.1 | 5.7 |
| Vowel | 0.1±0.1 | **0.0** | **0.0** | **0.0** | 0.1 | 0.1 | **0.0** | **0.0** | 0.1 | 0.2 | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** |
| Waveform | 10.5±3.0 | **0.9** | 1.3 | 2.9 | 2.3 | 2.8 | 4.4 | 1.5 | 3.4 | 6.4 | 2.0 | 2.7 | 4.3 | 1.1 | 2.3 | 4.5 |
| Wine | 1.0±1.0 | **0.0** | **0.0** | **0.0** | 0.4 | 0.3 | 0.3 | **0.0** | **0.0** | 0.1 | 0.1 | **0.0** | 0.1 | **0.0** | **0.0** | **0.0** |

Cuadro 5.8: Media del error de test en % para conjuntos compuestos de 10 %, 20 % y 40 % clasificadores. El mejor resultado se muestra en **negrita**. El segundo mejor subrayado

| | *Bagging* | **Reduc. error** | | | **Complemen.** | | | **Dist** ($p = 0.075$) | | | **Ángulos** | | | **B. *boosting*** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tamaño | 100 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % |
| Audio | 30.2±4.1 | 24.5 | **24.4** | 25.7 | 24.6 | 25.2 | 26.1 | 24.5 | 25.0 | 26.6 | 24.8 | 24.6 | 25.6 | **24.4** | 24.6 | 25.8 |
| Australian | 14.5±2.1 | 14.2 | **13.7** | 14.0 | **13.7** | **13.7** | 13.9 | 13.8 | **13.7** | 13.9 | 14.4 | 14.1 | 14.1 | 13.9 | 13.8 | 14.0 |
| Breast W. | 4.7±1.5 | 4.1 | 4.1 | 4.1 | 4.1 | 4.1 | 4.2 | **4.0** | **4.0** | 4.3 | 4.1 | 4.1 | 4.1 | 4.1 | **4.0** | **4.0** |
| Diabetes | 24.9±1.8 | 24.7 | 24.4 | 24.7 | **24.3** | 24.5 | 24.5 | 24.7 | 24.4 | 24.4 | 24.5 | 24.5 | 24.4 | 24.6 | 24.5 | 24.6 |
| German | 26.6±1.6 | 25.5 | **25.1** | 25.5 | 25.3 | **25.1** | 25.3 | 25.2 | 25.2 | 25.6 | 25.5 | 25.2 | 25.5 | 25.4 | 25.2 | 25.4 |
| Heart | 20.4±4.3 | 19.5 | 18.9 | 18.7 | 18.9 | 18.6 | 18.3 | 19.0 | **17.8** | **17.8** | 19.2 | 18.6 | 18.0 | 19.2 | 18.9 | 18.2 |
| Horse-Colic | 17.7±2.9 | 15.8 | 15.5 | 15.1 | 16.0 | 15.8 | 15.8 | 15.9 | 15.7 | 16.1 | 16.3 | 15.8 | **14.8** | 15.9 | 15.4 | 15.9 |
| Ionosphere | 9.3±2.5 | 7.5 | 7.6 | 7.9 | 7.5 | 7.5 | 7.9 | 7.3 | **7.1** | 7.5 | 7.3 | 7.4 | 7.8 | 7.7 | 7.5 | 7.8 |
| Labor | 14.4±7.8 | 12.7 | 12.3 | 12.2 | 13.0 | 12.6 | 12.5 | 11.1 | **8.5** | 9.4 | 10.9 | 10.0 | 9.3 | 11.4 | 10.5 | 9.7 |
| New-thyroid | 7.3±3.1 | 6.3 | 6.2 | 5.9 | 6.0 | 6.1 | 6.0 | 5.7 | **5.2** | 5.9 | 5.8 | 5.6 | 5.7 | 5.7 | 5.5 | 5.5 |
| Segment | 9.7±1.7 | 8.1 | 8.0 | 8.1 | 8.4 | 8.3 | 8.4 | **7.7** | 7.8 | 8.4 | 7.9 | 7.8 | 8.0 | 7.8 | **7.7** | 8.0 |
| Sonar | 24.7±4.7 | 21.6 | 21.5 | 22.0 | 22.2 | 22.0 | 22.6 | 20.2 | 20.6 | 22.1 | 20.6 | 20.7 | 21.4 | **20.1** | 20.4 | 21.2 |
| Tic-tac-toe | 2.7±1.1 | 2.5 | 2.3 | 2.2 | 2.9 | 2.6 | 2.5 | 2.2 | 2.5 | 2.8 | 2.2 | 2.1 | **2.0** | 2.2 | 2.1 | 2.1 |
| Twonorm | 9.3±3.1 | 9.7 | 8.7 | 8.2 | 10.2 | 9.4 | 8.9 | 8.1 | 7.8 | 8.2 | 7.7 | 6.6 | **6.3** | 8.0 | 7.1 | 6.9 |
| Vehicle | 29.6±2.2 | 26.5 | 26.5 | 27.0 | 26.4 | **26.3** | 26.9 | **26.3** | 26.5 | 27.3 | 26.4 | 26.4 | 27.1 | **26.3** | **26.3** | 27.0 |
| Vowel | 13.7±2.2 | 14.2 | 13.6 | 13.1 | 14.9 | 14.3 | 13.9 | 13.4 | 14.2 | 14.6 | 12.8 | **12.1** | 12.3 | 13.3 | 12.6 | 12.4 |
| Waveform | 22.8±2.5 | 20.5 | 20.0 | 20.2 | 20.3 | 19.8 | 19.9 | 19.9 | **19.4** | 20.1 | 20.2 | 19.6 | 19.8 | 20.0 | 19.6 | 19.9 |
| Wine | 6.5±4.0 | 5.9 | 5.8 | 6.2 | 6.4 | 6.1 | 6.0 | 4.7 | **3.8** | 4.1 | 5.1 | 4.8 | 4.7 | 4.9 | 4.5 | 4.4 |

Cuadro 5.9: Prueba-t para comparar *bagging* con respecto a las distintas técnicas de ordenación y poda. Se ha resaltado en **negrita** los valores-p$< 0.005$. Los valores recuadrados corresponden a resultados favorables a *bagging*

| | **Reduc. error** | | | **Complemen.** | | | **Dist** ($p = 0.075$) | | | **Ángulos** | | | **B. *boosting*** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| tamaño | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % | 10 % | 20 % | 40 % |
| Audio | **2e-30** | **1e-31** | **4e-31** | **7e-31** | **5e-30** | **3e-27** | **2e-29** | **7e-31** | **7e-28** | **2e-27** | **2e-29** | **7e-27** | **2e-29** | **5e-31** | **6e-29** |
| Australian | 0.04 | **3e-6** | **3e-4** | **2e-6** | **1e-6** | **5e-6** | **9e-5** | **1e-6** | **7e-6** | 0.55 | 0.01 | **2e-3** | **4e-4** | **5e-6** | **8e-5** |
| Breast | **9e-6** | **2e-6** | **6e-8** | **4e-6** | **6e-6** | **7e-8** | **2e-6** | **4e-10** | **1e-7** | **4e-5** | **8e-6** | **1e-7** | **9e-6** | **1e-7** | **2e-9** |
| Diabetes | 0.12 | **9e-5** | 0.07 | **1e-4** | **1e-3** | **1e-4** | 0.12 | **5e-4** | **2e-4** | 0.03 | 0.007 | **4e-4** | 0.04 | **3e-3** | **2e-3** |
| German | **4e-9** | **3e-18** | **4e-14** | **6e-11** | **3e-17** | **9e-16** | **5e-14** | **4e-15** | **4e-11** | **1e-8** | **2e-11** | **2e-13** | **7e-12** | **7e-16** | **8e-17** |
| Heart | 0.01 | **5e-6** | **3e-7** | **5e-5** | **1e-6** | **6e-9** | **4e-4** | **2e-10** | **4e-13** | 0.005 | **5e-6** | **5e-10** | **2e-3** | **3e-5** | **3e-10** |
| Horse-Colic | **2e-10** | **9e-15** | **2e-22** | **4e-10** | **1e-12** | **2e-15** | **1e-9** | **2e-14** | **2e-12** | **3e-6** | **9e-10** | **2e-21** | **2e-10** | **7e-18** | **5e-17** |
| Ionosphere | **2e-12** | **2e-15** | **2e-14** | **2e-15** | **3e-17** | **5e-13** | **7e-15** | **4e-19** | **6e-14** | **8e-15** | **1e-14** | **1e-13** | **4e-11** | **2e-15** | **2e-14** |
| Labor | 0.01 | **4e-4** | **5e-6** | 0.007 | **4e-4** | **5e-5** | **5e-6** | **8e-12** | **9e-12** | **5e-5** | **3e-8** | **1e-12** | **2e-5** | **2e-7** | **4e-10** |
| New-thyroid | **2e-4** | **3e-6** | **3e-9** | **2e-10** | **5e-11** | **1e-12** | **2e-10** | **6e-15** | **1e-13** | **1e-6** | **5e-10** | **1e-11** | **7e-9** | **1e-11** | **5e-14** |
| Segment | **2e-16** | **5e-19** | **3e-21** | **1e-14** | **3e-19** | **7e-21** | **2e-20** | **8e-23** | **4e-20** | **1e-17** | **1e-20** | **1e-20** | **9e-22** | **2e-23** | **3e-23** |
| Sonar | **2e-12** | **7e-16** | **3e-14** | **2e-10** | **2e-11** | **2e-11** | **2e-16** | **4e-16** | **4e-12** | **2e-14** | **4e-16** | **2e-13** | **3e-19** | **3e-23** | **2e-15** |
| Tic-tac-toe | 0.09 | **1e-6** | **2e-12** | 0.01 | 0.61 | **7e-4** | **4e-8** | 0.06 | 0.05 | **2e-5** | **1e-10** | **4e-16** | **2e-6** | **2e-9** | **9e-14** |
| Twonorm | 0.05 | **3e-3** | **2e-10** | **1e-5** | 0.36 | 0.02 | **5e-6** | **2e-9** | **5e-7** | **4e-7** | **3e-16** | **1e-23** | **1e-6** | **9e-15** | **2e-20** |
| Vehicle | **1e-23** | **4e-26** | **8e-26** | **8e-23** | **1e-29** | **9e-27** | **4e-24** | **3e-27** | **4e-27** | **1e-21** | **2e-24** | **3e-27** | **2e-23** | **7e-26** | **2e-28** |
| Vowel | **3e-4** | 0.28 | **3e-10** | **6e-14** | **4e-5** | 0.06 | 0.01 | **6e-5** | **5e-14** | **1e-9** | **2e-27** | **1e-29** | **4e-4** | **2e-18** | **4e-25** |
| Waveform | **1e-20** | **2e-27** | **2e-28** | **2e-20** | **5e-27** | **7e-30** | **2e-25** | **3e-32** | **1e-29** | **1e-21** | **5e-28** | **1e-31** | **3e-25** | **3e-31** | **3e-33** |
| Wine | 0.08 | 0.03 | 0.30 | 0.75 | 0.06 | **1e-3** | **2e-7** | **7e-12** | **1e-13** | **2e-4** | **2e-5** | **2e-7** | **1e-6** | **3e-9** | **9e-12** |

Además, se puede observar que ordenación basada en *boosting* es el método que obtiene mayor número de resultados significativos favorables con respecto a *bagging*. Sólo en el conjunto *Pima Indian Diabetes* la ordenación basada en *boosting* con un porcentaje de poda del 10 % no obtiene una mejora significativa con respecto a *bagging*. Asimismo, la regla ordenación basada en ángulos con un 40 % de los clasificadores también obtiene resultados significativamente mejores que *bagging* en todos los conjuntos de datos analizados.

Se han aplicado otros criterios de parada sin demasiado éxito. Para la ordenación basada en *boosting* se ha utilizado el criterio de parada de *boosting* para utilizar el número de clasificadores seleccionados cuando se obtiene el primero con error mayor que $0.5$ (paso 6 del algoritmo de la figura 5.5). Se obtienen errores medio punto peores en media con respecto a la selección fija del $20\,\%$ de los clasificadores con un porcentaje de árboles seleccionados muy variable de un conjunto a otro, resultando en el uso de 5 árboles de media para *Australian* y *Horse-colic* mientras que son necesarios en torno a 130 árboles para parar detener el proceso en otros conjuntos: *Labor*, *Vowel* y *Wine*. Por otro lado, el uso de pesos en los clasificadores para hacer la clasificación tampoco aporta ninguna mejora. Un criterio de parada aplicable a la ordenación por ángulos consiste en calcular la media de los ángulos de los vectores característicos de aquellos vectores cuyos ángulos con respecto a $\boldsymbol{c}_{ref}$ sean menores que $\pi/2$. A continuación se seleccionan sólo los clasificadores cuyo ángulo del vector característico sea menor que esta media. Esta regla da estimaciones razonables del número de clasificadores (15–30 % del total dependiendo del conjunto) necesarios para obtener buenos resultados de error en test. Con este criterio de poda se obtienen resultados muy similares a los obtenidos con una tasa de poda fija e igual al $20\,\%$ de los clasificadores originales.

### Tiempos de ejecución

Como hemos visto previamente todas las heurísticas presentadas tienen un orden de ejecución cuadrático con el número de clasificadores, excepto la ordenación por ángulos que tiene un orden medio de ejecución de $O(T\log(T))$. En el cuadro 5.10 se muestran los tiempos medios de ejecución para ordenar *bagging* usando ordenación por ángulos (OA) y minimización de distancias de margen (MDM) partiendo de 50, 100, 200, 400, 800 y 1600 árboles para el conjunto *Waveform* con 300 ejemplos de entrenamiento. Los órdenes de ejecución para la ordenación tienen, aparte de una dependencia con el número de clasificadores, una dependencia lineal con el número de ejemplos usados para la ordenación del conjunto. Esta última dependencia no es el objeto de las mediciones hechas en este experimento. Los resultados mostrados son la media sobre 100 ordenaciones realizadas usando un procesador Pentium® 4 a 3.2 GHz. Estos resultados muestran claramente el comportamiento aproximadamente lineal de ordenación por ángulos, en contraste a la complejidad cuadrática de las otras ordenaciones, concretamente minimización de distancias de margen.

Cuadro 5.10: Tiempo (s) medio de ordenación para ordenación por ángulos (OA) y minimización de distancias de margen (MDM) para distintos tamaños de conjuntos de clasificadores

| Tamaño | **50** | **100** | **200** | **400** | **800** | **1600** |
|---|---|---|---|---|---|---|
| OA | 0.086 | 0.14 | 0.28 | 0.56 | 1.2 | 2.4 |
| MDM | 0.18 | 0.52 | 1.7 | 6.3 | 24.3 | 94.4 |

## 5.6. Conclusiones

En este capítulo se propone la modificación del orden de agregación en un conjunto de clasificadores para seleccionar un subconjunto óptimo de clasificadores de tamaño menor que el conjunto de partida. Si los clasificadores se ordenan con reglas que tienen en cuenta la complementariedad entre los clasificadores, las curvas de aprendizaje que muestran la dependencia del error de clasificación con el número de clasificadores tienen un mínimo para tamaños intermedios. Este mínimo corresponde a un subconjunto con menos clasificadores y, generalmente, menor error de generalización que el conjunto completo en los conjuntos estudiados. Se han mostrado resultados usando las distintas reglas de ordenación propuestas: reducción de error (variante de una regla presentada en [Margineantu y Dietterich, 1997]), complementariedad, minimización por distancias de margen, ordenación por ángulos y ordenación basada en *boosting*. Se han hecho experimentos que demuestran la utilidad de estas reglas para la ordenación de conjuntos basados en *bagging*.

Las reglas que se basan exclusivamente en características individuales de los clasificadores, como error en entrenamiento o en un conjunto de validación, no han permitido seleccionar subconjuntos más eficaces que el conjunto completo generado con *bagging*. Esto es debido a que estos procedimientos de ordenación no tienen en cuenta la complementariedad entre los clasificadores para construir el conjunto. Las reglas propuestas en esta tesis tienen en cuenta explícitamente esta complementariedad: minimización de distancias de margen, la ordenación por ángulos y ordenación basada en *boosting* intentan aumentar el margen de los ejemplos más difíciles. La regla de complementariedad tiene en cuenta la clasificación dada por el conjunto para realizar la selección de los clasificadores. Los experimentos realizados muestran que la regla de minimización de distancias de margen utilizando un $20\,\%$ de los clasificadores ($80\,\%$ de poda) obtiene los mejores resultados en la mayoría de los conjuntos seleccionados. Además, las reglas de ordenación basada en *boosting* utilizando $20 - 40\,\%$ de los clasificadores y la de ordenación por ángulos con $40\,\%$ obtienen resultados significativamente mejores que *bagging* en todos los conjuntos estudiados.

También hemos podido observar cómo los conjuntos ordenados presentan una mejora del error de generalización para un rango grande de valores de poda. Esto significa que

con los métodos propuestos es fácil seleccionar un subconjunto más pequeño y eficiente en clasificación por lo que su uso junto con *bagging* es recomendable. Sin embargo, y dado que en las curvas de error de entrenamiento la posición del mínimo se obtiene para un número menor de clasificadores, es difícil determinar con exactitud el porcentaje óptimo de poda. Se han propuesto dos reglas para resolver este problema en el algoritmo de ordenación basada en ángulos y en el algoritmo de ordenación basada en *boosting*. Estas reglas consiguen resultados equivalentes a los obtenidos con la regla que selecciona el $20\,\%$ de los clasificadores originales.

En cuanto al tamaño inicial del conjunto, no parece razonable partir de conjuntos compuestos de más de en torno a 250 clasificadores para los conjuntos explorados. El uso de conjuntos iniciales más grandes no conduce a grandes mejoras en el error de generalización. El uso de conjuntos de tamaño mayor selecciona subconjuntos con un mayor número de clasificadores, lo que hace que se pierda una de las ventajas más interesantes de estos algoritmos, que es el obtener un subconjunto pequeño de clasificadores eficaz en clasificación.

El tiempo de ejecución de todas las heurísticas de ordenación presentadas es cuadrático en el número de clasificadores $T$, excepto la ordenación por ángulos que tiene un tiempo equivalente al *quick-sort*, esto es $O(T\log(T))$ (además, y si sólo nos interesa la selección de los $\tau$ primeros clasificadores, se puede aplicar el algoritmo *quick-select* que tiene un tiempo medio de ejecución de $O(T)$). El método de ordenación por ángulos es por tanto la heurística más rápida de las presentadas para la ordenación y poda de clasificadores dentro de un conjunto.

La aplicación de búsqueda exhaustiva para la selección del subconjunto óptimo nos ha permitido validar las heurísticas presentadas como herramientas de optimización en sí mismas. Se ha podido observar cómo la búsqueda codiciosa de reducción de error obtiene resultados muy cercanos a la búsqueda exhaustiva en conjuntos de tamaño menor que 30 para el problema *Waveform*. Las heurísticas de búsqueda presentadas se basan en que el subconjunto de tamaño $u$ se obtiene añadiendo un elemento al subconjunto de tamaño $u-1$. Esto no es siempre así, pero se ha podido comprobar cómo la solución obtenida por este procedimiento codicioso conduce a soluciones próximas a la óptima. También se han aplicado algoritmos genéticos para resolver el problema de la optimización en conjuntos de tamaño superior, en los que la búsqueda exhaustiva no es posible. En la implementación realizada no se han alcanzado las cotas de error de los métodos basados en heurísticas de ordenación ni en entrenamiento ni en test. Sin embargo, hay que mostrarse prudentes ante este resultado ya que es posible que distintas codificaciones o configuraciones en el AG den lugar a mejoras.

# Capítulo 6

# Conclusiones y trabajo futuro

Como resultado de las investigaciones realizadas en el marco de esta tesis doctoral se han desarrollado una serie de herramientas de clasificación dentro del campo de la clasificación supervisada basadas en los conjuntos de clasificadores. Los distintos algoritmos presentados aportan mejoras en la capacidad de generalización y en algunos casos en el uso eficiente de recursos computacionales. Los métodos propuestos se pueden dividir en dos grupos claramente diferenciados: los de creación de conjuntos de clasificadores y los de ordenación y poda de estos conjuntos una vez generados.

Los procedimientos de creación de conjuntos que hemos desarrollado en esta tesis incluyen métodos que utilizan como base el algoritmo de crecimiento y poda iterativos IGP [Gelfand *et al.*, 1991] y el método *class-switching* [Martínez-Muñoz y Suárez, 2005b].

Dentro de los procedimientos de creación de conjuntos de clasificadores se han propuesto tres nuevos métodos que usan los árboles IGP como algoritmo base. Estos son: conjunto IGP [Martínez-Muñoz y Suárez, 2002; 2004b], *boosting* IGP y comités IGP [Martínez-Muñoz y Suárez, 2005a]. Para construir un árbol de decisión a partir de un conjunto de datos de entrenamiento, el algoritmo IGP divide dicho conjunto en dos subconjuntos de igual tamaño y distribución de clases similar a la del conjunto inicial. En cada iteración del algoritmo uno de los subconjuntos se utiliza para hacer crecer el árbol y el otro para podarlo. Los papeles de los subconjuntos son intercambiados en cada una de las iteraciones de crecimiento y poda. Partiendo de distintas divisiones iniciales de los datos el algoritmo IGP construye árboles diferentes. Los conjuntos de árboles IGP propuestos aprovechan esta variabilidad intrínseca del algoritmo de construcción de árboles IGP para generar los conjuntos de clasificadores. El primer método propuesto, conjunto IGP, genera cada árbol IGP del conjunto utilizando una división aleatoria distinta de los datos de entrenamiento. Este algoritmo genera clasificadores diversos entre sí sin necesidad de realizar remuestreos de datos o perturbaciones externas y utiliza todos los datos de entrenamiento (con el mismo peso) para construir cada uno de los clasificadores del conjunto. Asimismo, el conjunto IGP reduce el error de generalización con respecto al algoritmo base entrenado con todos

los ejemplos para los problemas analizados. Este algoritmo es robusto en conjuntos de datos difíciles como *Pima Indian Diabetes* al igual que *bagging* y a diferencia de *boosting*. Asimismo, el conjunto IGP obtiene menores errores de generalización que *bagging* en los conjuntos analizados. *Boosting* IGP, por su parte, se puede considerar como un algoritmo de tipo *boosting* en el que los clasificadores son generados de forma que se especialicen en la clasificación de datos de entrenamiento que han sido mal clasificados por los clasificadores previamente generados. Sin embargo este algoritmo no es capaz de alcanzar la capacidad de generalización del *boosting* original. El tercer algoritmo basado en árboles IGP, comités IGP, es un algoritmo híbrido entre conjunto IGP y *boosting* IGP. Esta combinación de características le confiere buenas propiedades en cuanto a capacidad de generalización (comparables con *boosting*) y buena estabilidad frente al ruido como *bagging*.

Asimismo, se ha propuesto un método de construcción de conjuntos de clasificadores basado en la modificación aleatoria de las etiquetas de clase. A este algoritmo de creación de conjuntos lo hemos denominado *class-switching* [Martínez-Muñoz y Suárez, 2005b]. Para construir cada clasificador individual, *class-switching* genera un nuevo conjunto de datos modificando aleatoriamente las etiquetas de clase de un porcentaje fijo y elegido al azar de ejemplos del conjunto de entrenamiento. Siempre que los clasificadores individuales obtengan error cero en los conjuntos modificados, este procedimiento genera clasificadores cuyos errores en el conjunto de entrenamiento original son independientes entre sí. De hecho, para problemas de dos clases, *class-switching* se puede analizar como un proceso de Bernoulli: la probabilidad de que un clasificador individual extraído al azar del conjunto clasifique bien un ejemplo cualquiera de entrenamiento es siempre igual a uno menos el porcentaje de ejemplos modificados. Como consecuencia, la evolución de las curvas de error en entrenamiento con el número de clasificadores sólo depende del porcentaje de ejemplos modificados. Es decir, estas curvas son independientes del problema de clasificación. *Class-switching* alcanza su rendimiento óptimo para porcentajes de modificación de las etiquetas de clase elevados (en torno al 30 % de los ejemplos en problemas binarios y mayores para problemas con múltiples clases) y usando un gran número de clasificadores (en torno a 1000 clasificadores). Bajo estas condiciones *class-switching* obtiene en media resultados muy superiores a *bagging* y mejores que *boosting* en los problemas estudiados.

En la segunda parte de este trabajo de tesis se han propuesto una serie de métodos basados en la reordenación de los clasificadores de un conjunto generado con *bagging* [Martínez-Muñoz y Suárez, 2004a; 2006]. Estas reordenaciones permiten reducir el número de clasificadores del conjunto que se utilizan consiguiendo tanto una disminución de requerimientos de almacenaje, como un aumento de la velocidad de clasificación, lo cual es un factor clave en aplicaciones en línea. Los conjuntos de clasificadores que se generan mediante la aplicación de las heurísticas de ordenación y poda propuestas mejoran la capacidad de generalización de *bagging* en los problemas analizados. Para que los métodos de ordenación sean efectivos han de tener en cuenta la complementariedad de los elementos dentro del conjunto. Una vez ordenado el conjunto de clasificadores se seleccionan los $\tau$

primeros elementos de acuerdo con una regla de poda. Se han desarrollado cinco métodos de ordenación basados en la complementariedad entre los clasificadores individuales: reducción de error, medida de complementariedad, minimización de distancias de margen, ordenación por ángulos y ordenación basada en *boosting*. En la mayoría de ellos (todos excepto el método de ordenación por ángulos) se aplica el siguiente procedimiento: a partir de un subconjunto de clasificadores de tamaño $u-1$ se selecciona un clasificador de entre los restantes de forma que se minimice/maximice una cantidad para el subconjunto de tamaño $u$. Para la ordenación por reducción de error esta cantidad es el error de clasificación. La medida de complementariedad se basa en contar el número de ejemplos mal clasificados por el subconjunto de tamaño $u-1$ y bien por el clasificador a seleccionar. El método de minimización de distancias de margen utiliza una medida de distancia en el espacio de clasificación. En este espacio, de dimensión igual al número de ejemplos empleados en el proceso de ordenación, se codifica el funcionamiento de cada clasificador individual por medio de un vector cuyas componentes indican la clasificación correcta/incorrecta del clasificador para cada dato. Por último, la ordenación basada en *boosting* se basa en calcular el error de clasificación ponderado con pesos que se modifican de una forma similar a *boosting*. El método de ordenación por ángulos, por su parte, ordena los clasificadores por el ángulo que forman con respecto a un eje de clasificación perpendicular al eje de clasificación del conjunto completo en el mismo espacio de clasificación de ejemplos del método de distancias de margen.

Todas las heurísticas propuestas generan un nuevo orden de agregación de los clasificadores del conjunto. Con esta nueva ordenación, la curva de dependencia del error de clasificación con el número de clasificadores presenta las siguientes características: (i) disminución inicial del error de generalización a medida que aumenta el número de clasificadores. Esta disminución es más pronunciada que la de las curvas correspondientes a *bagging* con el orden de agregación aleatorio original; (ii) se alcanza un mínimo para un número intermedio de clasificadores correspondiente a un subconjunto cuyo error está por debajo del error del conjunto completo; (iii) finalmente aumenta hasta el error final de *bagging* para el total de los clasificadores (como es de esperar). Estas características se observan tanto en las curvas de entrenamiento como en las de test. Generalmente, para casi todas las reglas y conjuntos estudiados, el conjunto ordenado obtiene resultados por debajo del error final del *bagging* a partir de un número pequeño de clasificadores. En general, en los problemas analizados, se alcanza un error por debajo del error de *bagging* en subconjuntos con tamaño mayor que el $10\,\%$ del tamaño del conjunto original para conjuntos suficientemente grandes ($\geq 100$ clasificadores). Por tanto, para obtener mejoras de clasificación basta con podar el conjunto en este amplio rango (10–$100\,\%$ de los clasificadores iniciales). Las pruebas realizadas sobre 18 conjuntos de datos tanto sintéticos como de diversos campos de aplicación han mostrado que una selección del $20\,\%$ (poda del $80\,\%$) de clasificadores produce mejoras significativas con respecto al conjunto completo, siendo minimización de distancias de margen el método que en media mejores resultados ha producido.

En cuanto a desarrollos futuros, dentro de los métodos basados en árboles IGP, sería interesante analizar su comportamiento en un rango mayor de problemas de clasificación, así como analizar la diversidad de los clasificadores que obtiene y compararla con la obtenida por *bagging* y *boosting*.

De más interés sería el análisis de la diversidad de los clasificadores generados con el método *class-switching*, ya que se debería ver una relación bastante directa entre el porcentaje de ejemplos modificados y las nubes obtenidas en los diagramas kappa-error. Asimismo, puede ser muy interesante la combinación del análisis de diversidad con una modificación del algoritmo de alteración de etiquetas de clase (como el presentado en [Kuncheva y Kountchev, 2002]) para que generara clasificadores con distintas medidas de diversidad y no estrictamente independientes como los que produce el método propuesto.

Por otro lado, los métodos de ordenación propuestos se pueden aplicar a una gran variedad de conjuntos y problemas. La extensión más inmediata del trabajo presentado sería aplicarlo a otros conjuntos compuestos por otro tipo de clasificadores como por ejemplo a conjuntos de redes neuronales. Asimismo estos métodos con pequeñas adaptaciones se podrían aplicar a regresión. La regla de reducción de error adaptada a regresión podría buscar el regresor que más reduzca el error cuadrático medio. La medida de complementariedad puede seleccionar el regresor que reduzca el error cuadrático medio del mayor número de ejemplos.

Por otro lado, y espoleados por la observación de que las heurísticas propuestas no son útiles para ordenar conjuntos formados por clasificadores que tienen una capacidad expresiva elevada, habría que analizar más en profundidad cómo varía la capacidad de generalización de los subconjuntos obtenidos por ordenación y poda con la capacidad de representación del clasificador individual. Por ejemplo variando la tasa de poda de los árboles generados en el conjunto pasando de árboles no podados a árboles con una sola pregunta (*Decision Stump*), o modificando el número de neuronas en la capa oculta de una red neuronal. Estudios preliminares han mostrado que la ordenación y poda de *class-switching*, *bagging* con árboles sin podar o con redes neuronales con muchos nodos en la capa oculta no mejoran significativamente la capacidad de generalización de los conjuntos.

Finalmente es necesario profundizar en el análisis de la dependencia de las curvas de error de generalización con el número de clasificadores para los conjuntos ordenados (y sin ordenar). Esto permitiría dar una estimación más precisa de la posición del mínimo.

# Apéndice A

# Descripción de los conjuntos de datos utilizados

### A.1.1. Audio

| Audiology-standarized | | | Repositorio UCI (Professor Jergen at Baylor College of Medicine) |
|---|---|---|---|
| Datos: | 226 | Atributos: | 69 categóricos de los cuales 60 binarios |
| Clases: | 24 | Distribución: | 1 (x5), 2 (x7), 3, 4 (x3), 6, 8, 9, 20, 22 (x2), 48 y 57 |
| Tipo: | Real | Ausentes: | Sí (317 valores: 2 %) |
| Descripción: | Identificación de afecciones del oído. | | |
| Observaciones: | Conjunto con muchas clases con muy pocos ejemplos: 16 clases con menos de 5 ejemplos y 5 clases con un solo dato. Esto hace que sea prácticamente imposible de predecir correctamente (siempre habrá clases que o aparecen en entrenamiento o en test pero no en los dos). | | |

### A.1.2. Australian Credit

| Australian Credit | | | Repositorio UCI (Confidencial, enviado por Ross Quinlan) |
|---|---|---|---|
| Datos: | 690 | Atributos: | 14 (6 cuantitativos, 8 categóricos) |
| Clases: | 2 | Distribución: | 307 y 383 |
| Tipo: | Real | Ausentes: | Sí (37 ejemplos cuyos valores están substituidos por la moda/media (atribs cat./cuan.)) |
| Descripción: | Conjunto sobre aplicaciones de tarjetas de crédito. Todos los atributos y valores de clase están cambiados (por confidencialidad) y no se sabe a qué hacen referencia. | | |
| Observaciones: | El problema original tenía valores ausentes que fueron substituidos por la media/moda (cuantitativo/categórico). Conjunto utilizado en el proyecto Statlog [Michie *et al.*, 1994]. | | |

### A.1.3. Breast Cancer Wisconsin

| Breast Cancer Wisconsin | | | Repositorio UCI (Dr. William H. Wolberg - University of Wisconsin Hospitals) |
|---|---|---|---|
| Datos: | 699 | Atributos: | 9 cuantitativos |
| Clases: | 2 | Distribución: | 458 (benigno) y 241 (maligno) |
| Tipo: | Real | Ausentes: | Sí (16 valores: $<1\,\%$) |
| Descripción: | Consiste en distinguir entre cancer de pecho maligno o benigno. La base de datos contiene información obtenida a partir de muestras como: uniformidad en el tamaño y forma de las células, mitosis, etc. | | |
| Observaciones: | Conjunto relativamente sencillo donde un discriminante lineal obtiene precisiones por encima del $90\,\%$. | | |

### A.1.4. Pima Indian Diabetes

| Pima Indian Diabetes | | | Repositorio UCI (National Institute of Diabetes and Digestive and Kidney Diseases) |
|---|---|---|---|
| Datos: | 768 | Atributos: | 8 cuantitativos |
| Clases: | 2 | Distribución: | 500 (no diabética) y 268 (diabética) |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Se debe identificar si las pacientes son diabéticas o no de acuerdo con los criterios de la Organización Mundial de la Salud. Los atributos incluyen: edad, índice de masa corporal, concentración de glucosa en el plasma con un test oral, presión, etc. | | |
| Observaciones: | Base de datos obtenida a partir de una mayor (no pública) de donde se extrajeron una serie de pacientes mujeres de al menos 21 años con herencia de los indios Pima. Se trata de un problema difícil de clasificar donde incluso la clase no tiene correspondencia directa con el hecho de ser diabético, se obtuvo a partir de otro atributo muy indicativo pero no definitivo para la diagnosis de la enfermendad. Conjunto utilizado en el proyecto Statlog [Michie *et al.*, 1994]. | | |

### A.1.5. German Credit

| German Credit | | | Repositorio UCI (Professor Dr. Hans Hofmann - Universität Hamburg) |
|---|---|---|---|
| Datos: | 1000 | Atributos: | 20 (7 cuantitativos, 13 categóricos) |
| Clases: | 2 | Distribución: | 700 (bueno) y 300 (malo) |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Identificación de un cliente como bueno o malo a partir de la cantidad del crédito, ahorros, trabajo, edad, etc. | | |
| Observaciones: | Existe otra versión con 24 atributos numéricos que se usó en el proyecto Statlog [Michie *et al.*, 1994] donde además se la utilizaron con una matriz de coste que penalizaba clasificar un cliente como bueno siendo malo. Se trata de un problema complejo donde es difícil bajar de 30 % deerror (porcentaje de la clase más probable a priori). | | |

### A.1.6. Heart

| Heart | | | Repositorio UCI (Robert Detrano - Cleveland Clinic Foundation) |
|---|---|---|---|
| Datos: | 270 | Atributos: | 13 (10 cuantitativos, 3 categóricos) |
| Clases: | 2 | Distribución: | 150 (ausencia) y 120 (presencia) |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Consiste en la identificación de ausencia o presencia de enfermedad coronaria en pacientes a partir de: edad, sexo, tipo de dolor de pecho, pruebas médicas, etc. | | |
| Observaciones: | Esta base de datos fue creada en el proyecto Statlog [Michie *et al.*, 1994] a partir de la base de datos *Heart-Cleveland*. La base de datos original contenía 75 atributos y 5 grados de enfermedad coronaria que fueron simplificados a 13 atributos y dos clases (ausencia o presencia de enfermedad). Asimismo, se eliminaron una serie de instancias por tener valores ausentes y otras causas. En el proyecto Statlog esta base de datos se utilizo con una matriz de coste que penalizaba clasificar un paciente como sano estando enfermo. | | |

### A.1.7. Horse Colic

| Horse Colic | | | Repositorio UCI (Mary McLeish y Matt Cecile - University of Guelph) |
|---|---|---|---|
| Datos: | 368 | Atributos: | 21 (7 cuantitativos, 14 categóricos) |
| Clases: | 2 | Distribución: | 232 (Sí) y 136 (No) |
| Tipo: | Real | Ausentes: | Sí (30 % de los valores) |
| Descripción: | A partir del estado de los caballos (pulse, temperatura de distintas partes del cuerpo, frecuencia respiratoria, etc) determinar si la lesion era retrospectivamente para operar o no | | |
| Observaciones: | Existes 5 posibles campos sobre los que clasificar. Cuando se utiliza este conjunto se eliminan los 4 campos con clases que no se usen además de un identificador de hospital y otra variable más con todos sus valores ausentes. Se trata de un problema complicado en parte por la gran cantidad de valores ausente. | | |

### A.1.8. Ionosphere

| Ionosphere | | | Repositorio UCI<br>Vince Sigillito - Johns Hopkins University |
|---|---|---|---|
| Datos: | 351 | Atributos: | 34 cuantitativos |
| Clases: | 2 | Distribución: | 225 (bueno) y 126 (malo) |
| Tipo: | Real | Ausentes: | No |
| Descripción: | El objetivo es identificar electrones libres en la ionosfera resultando en mediciones que identifican alguna estructura en la ionosfera (mediciones buenas) y aquéllas que no (mediciones malas). | | |
| Observaciones: | Información de radar proveniente de 16 antenas de alta frecuencia situadas en la bahía Goose, Labrador (Canadá). La señal está compuesta de 17 pulsos que se procesan para obtener los dos valores de una señal electromagnética compleja resultando en los 34 atributos del conjunto. | | |

### A.1.9. Labor Negotiations

| Labor Negotiations | | | Repositorio UCI<br>(Collective Barganing Review, montly publication, Labour Canada) |
|---|---|---|---|
| Datos: | 57 | Atributos: | 16 (8 cuantitativos, 8 categóricos) |
| Clases: | 2 | Distribución: | 37 (buen acuerdo de convenio) y 20 (malo) |
| Tipo: | Real | Ausentes: | Sí (326 - 36 % de los valores) |
| Descripción: | El problema consiste en identificar buenos y malos acuerdos de trabajo. La base de datos incluye información de: duración del acuerdo, incremento salarial en los primeros años, horas de trabajo semanales, número de días de vacaciones pagados, etc. | | |
| Observaciones: | Los datos resumen los acuerdos finales alcanzados en negociaciones de trabajo en Canadá durante un periodo comprendido entre 1988 y 1989. Incluyen convenios colectivos de diversos sectores con plantillas de al menos 500 trabajadores (profesores, enfermeras, personal universitario, policia, etc). | | |

### A.1.10. New-Thyroid

| New-thyroid | | | Repositorio UCI (Danny Coomans - James Cook University) |
|---|---|---|---|
| Datos: | 215 | Atributos: | 5 cuantitativos |
| Clases: | 3 | Distribución: | 30 (hipo), 35 (hiper) y 150 (normal) |
| Tipo: | Real | Ausentes: | No |
| Descripción: | A partir de 5 pruebas de laboratorio identificar si el paciente sufre de hipotiroidismo, hipertiroidismo o está normal. | | |
| Observaciones: | La clase se obtuvo a partir de una diagnosis basada en más información que la de la base de datos (anamnesis, scáner, etc.). | | |

### A.1.11. Image Segmentation

| Image Segmentation | | | Repositorio UCI (Vision Group, University of Massachusetts) |
|---|---|---|---|
| Datos: | 2310 | Atributos: | 19 cuantitativos |
| Clases: | 7 | Distribución: | Aprox. equilibrada |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Consiste en la identificación de distintas texturas (ladrillo, cielo, hojas, cemento, ventana, camino o hierba) dentro de imágenes. Cada instancia define una serie de características de una región de 3x3 píxeles como: el valor medio de rojo, verde y azul, contrastes, intensidades, etc. Las instancias fueros obtenidas aleatoriamente a partir de una base de datos de 7 imágenes en exteriores. | | |
| Observaciones: | Conjunto utilizado en el proyecto Statlog [Michie *et al.*, 1994]. | | |

### A.1.12. Sonar

| Sonar | | | Repositorio UCI (Terry Sejnowski - University of California) |
|---|---|---|---|
| Datos: | 208 | Atributos: | 60 cuantitativos |
| Clases: | 2 | Distribución: | 111 (minas) y 97 (rocas) |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Se trata de discernir entre señales de sónar rebotadas de rocas de las rebotadas de cilindros metálicos (ambas obtenidas desde distintos ángulos). Cada uno de los atributos representa la energía para una banda de frecuencia integrada durante un determinado lapso de tiempo y codificada en el rango $[0, 1]$. | | |
| Observaciones: | | | |

### A.1.13. Threenorm

| Threenorm | | | (Leo Breiman) |
|---|---|---|---|
| Datos: | - | Atributos: | 20 cuantitativos |
| Clases: | 2 | Distribución: | En general se usa equilibrada |
| Tipo: | Sint. | Ausentes: | En general se usa sin valores ausentes |
| Descripción: | Las dos clases se generan a partir de tres normales en 20 dimensiones. La clase 1 se extrae de dos normales con matrices de covarianza unidad y con media $(a, a, \ldots, a)$ y $(-a, -a, \ldots, -a)$. La clase 2 se extrae de la tercera normal, unitaria y con media $(a, -a, a, -a \ldots, a, -a)$. Donde $a = 2/\sqrt{20}$. | | |
| Observaciones: | La frontera de Bayes viene definida por la unión continua de dos hiperplanos oblicuos. El error de Bayes es aproximadamente 10.5 % [Breiman, 1996b]. | | |

### A.1.14. Tic-tac-toe

| Tic-tac-toe | | | Repositorio UCI (David W. Aha) |
|---|---|---|---|
| Datos: | 958 | Atributos: | 9 categóricos |
| Clases: | 2 | Distribución: | 626 (gana x) y 332 (pierde 'x') |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Define todas las posibles posiciones finales del juego del tic-tac-toe (similar al 3-en-raya). Consiste en determinar si ganan (tienen 3 fichas en raya) las 'x' (que son los que empizan) o no. Cada uno de los atributos indica el contenido de una de las 9 casillas de tablero 3x3 de entre: ficha 'x', ficha 'o' o vacío. | | |
| Observaciones: | El concepto detrás de este problema es conocido por lo que se puede usar igual que un conjunto sintético para estudiar el efecto pueden tener distintas modificaciones en los datos. El error mínimo alcanzable es 0 % (no tiene error de Bayes). | | |

### A.1.15. Twonorm

| Twonorm | | | (Leo Breiman) |
|---|---|---|---|
| Datos: | - | Atributos: | 20 cuantitativos |
| Clases: | 2 | Distribución: | En general se usa equilibrada |
| Tipo: | Sint. | Ausentes: | En general se usa sin valores ausentes |
| Descripción: | Cada clase se extrae de una distribución normal de 20 dimensiones con matriz de covarianza unidad y con media $(a, a, \ldots, a)$ para la clase 1 y con media $(-a, -a, \ldots, -a)$ para la clase 2. Donde $a = 2/\sqrt{20}$. | | |
| Observaciones: | La frontera de Bayes es un hiperplano oblicuo que pasa por el origen y que está definido por el vector $(a, a, \ldots, a)$. El error de Bayes es aproximadamente 2.3 % [Breiman, 1996b]. | | |

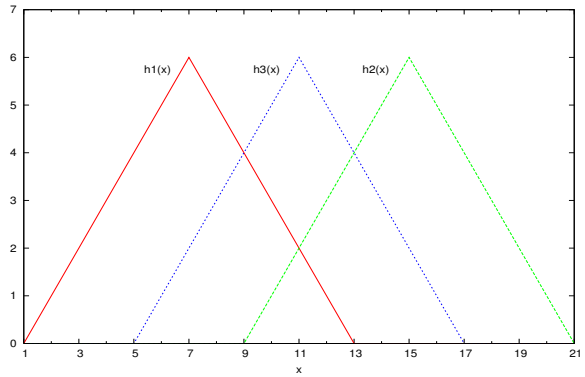
### A.1.16. Vehicle

| Vehicle silhouettes | | | Repositorio UCI (Drs.Pete Mowforth and Barry Shepherd - Turing Institute Glasgow) |
|---|---|---|---|
| Datos: | 846 | Atributos: | 18 cuantitativos |
| Clases: | 4 | Distribución: | 240 (opel), 240 (saab), 240 (bus) y 226 (van) |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Se trata de identificar un tipo de vehículo a partir de ciertas características de su silueta. Los posibles vehículos son: Autobus de dos pisos, Opel Manta400, Saab 9000 y furgoneta Chevrolet. Se extrajeron imágenes 128x128 que a continuación se pasaron a blanco y negro para obtener la silueta. Posteriormente, de la forma de la silueta se obtuvieron 18 atributos como: compactación (radio medio$^2$/área), circularidad (perímetro$^2$/área), relación entre el eje mayor y menor,etc. que son los que se utilizan para clasificar. | | |
| Observaciones: | Conjunto utilizado en el proyecto Statlog [Michie *et al.*, 1994]. | | |

### A.1.17. Vowel

| Vowel | | | Repositorio UCI (David Deterding) |
|---|---|---|---|
| Datos: | 990 | Atributos: | 10 cuantitativos |
| Clases: | 11 | Distribución: | Equilibrada |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Este problema consiste en la distinción entre los 11 fonemas vocales del inglés. Los datos contienen información de la pronunciación de 15 locutores (8 hombres y 7 mujeres) pronunciando seis veces cada fonema lo que hace un total de $11 \times 15 \times 6 = 990$ ejemplos. | | |
| Observaciones: | La señal de voz se procesó mediante un filtro de paso bajo y se digitalizarón a 12 bits con una frecuencia de muestroe de 10kHz. Un análisis posterior dio los 10 atributos a partir de unos coeficientes de reflexión. | | |

### A.1.18. Waveform

| Waveform (Leo Breiman) | | | |
|---|---|---|---|
| Datos: | - | Atributos: | 21 cuantitativos |
| Clases: | 3 | Distribución: | En general se utiliza equilibrada |
| Tipo: | Sint. | Ausentes: | En general se usa sin valores ausentes |
| Descripción: | Consiste en distinguir entre tres señales provenientes de distintas mezclas de señales triangulares. | | |
| Observaciones: | Las tres clases del problema se generan mezclando las tres siguientes ondas triangulares ($h_1(x)$, $h_2(x)$ y $h_3(x)$):   del siguiente modo: ▪ Clase 1 = $uh_1(x) + (1-u)h_2(x) + \epsilon_x \qquad x = 1, 2, \ldots, 21$ ▪ Clase 2 = $uh_1(x) + (1-u)h_3(x) + \epsilon_x \qquad x = 1, 2, \ldots, 21$ ▪ Clase 3 = $uh_2(x) + (1-u)h_3(x) + \epsilon_x \qquad x = 1, 2, \ldots, 21$ donde $u$ en un número aleatorio uniforme en el rango $[0, 1]$ y $\epsilon_1, \epsilon_2, \ldots, \epsilon_{21}$ es ruido gausiano proveniente de una normal N(0, 1). Se puede obtener una expresión para la regla de Bayes con la que se puede estimar el error de Bayes de este conjunto. En [Breiman, 1996b] estiman este error en 13.2 %. | | |

### A.1.19. Wine

| Wine | | | Repositorio UCI<br>(Forina, M. - Istituto di Analisi e Tecnologie Farmaceutiche ed Alimentari Genova) |
|---|---|---|---|
| Datos: | 178 | Atributos: | 13: cuantitativos y categóricos |
| Clases: | 3 | Distribución: | 71, 59 y 48 |
| Tipo: | Real | Ausentes: | No |
| Descripción: | Continene datos del resultado del análisis químico de vinos italianos de una misma región pero de distitos tipos de uva. Los análisis determinaron la cantidad de 13 constituyentes en cada uno de los tres tipos de vino como: alcohol, ácido málico, intensidad de color, etc. | | |
| Observaciones: | Conjunto relativamente fácil con tres clases separables linealmente por dos hiperplanos. | | |

# Bibliografía


[Aha *et al.*, 1991] David W. Aha, Dennis Kibler, y Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.

[Bakker y Heskes, 2003] Bart Bakker y Tom Heskes. Clustering ensembles of neural network models. *Neural Networks*, 16(2):261–269, marzo 2003.

[Bauer y Kohavi, 1999] Eric Bauer y Ron Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1-2):105–139, 1999.

[Blake y Merz, 1998] C. L. Blake y C. J. Merz. UCI repository of machine learning databases, 1998.

[Blumer *et al.*, 1990] A. Blumer, E. Ehrenfeucht, D Haussler, y M. K. Warmuth. Occam's razor. En Jude Shavlik y Thomas G. Dietterich, editors, *Readings in Machine Learning*, The Morgan Kaufmann Series in Machine Learning, páginas 201–204. Morgan Kaufmann, 1990.

[Breiman *et al.*, 1984] Leo Breiman, J. H. Friedman, R. A. Olshen, y C. J. Stone. *Classification and Regression Trees*. Chapman & Hall, New York, 1984.

[Breiman, 1996a] Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Breiman, 1996b] Leo Breiman. Bias, variance, and arcing classifiers. Technical Report 460, Statistics Department, University of California, 1996.

[Breiman, 1996c] Leo Breiman. Out-of-bag estimation. Technical report, Statistics Department, University of California, 1996.

[Breiman, 1997] Leo Breiman. Arcing the edge. Technical report, University of California, Berkeley, CA, 1997.

[Breiman, 1998] Leo Breiman. Arcing classifiers. *The Annals of Statistics*, 26(3):801–849, 1998.

[Breiman, 1999] Leo Breiman. Pasting small votes for classification in large databases and on-line. *Machine Learning*, 36(1-2):85–103, 1999.

[Breiman, 2000] Leo Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):229–242, 2000.

[Breiman, 2001] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[Bryll *et al.*, 2003] Robert Bryll, Ricardo Gutierrez-Osuna, y Francis Quek. Attribute bagging: improving accuracy of classifier ensembles by using random feature subsets. *Pattern Recognition*, 36(6):1291–1302, junio 2003.

[Burges, 1998] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[Cantador y Dorronsoro, 2004] I. Cantador y J. R. Dorronsoro. Parallel perceptrons and training set selection for imbalanced classification problems. En *Proceedings of the Learning 04 International Conference*, 2004.

[Chan *et al.*, 1999] P. K. Chan, W. Fan, Andreas L. Prodromidis, y Salvatore J. Stolfo. Distributed data mining in credit card fraud detection. *IEEE Intelligent Systems and their Applications*, 14(6):67–74, 1999.

[Chang y Pavlidis, 1977] R. Chang y T. Pavlidis. Fuzzy decision tree algorithms. *IEEE Transactions on Systems, Man and Cybernetics*, 7(1):28–35, 1977.

[Chawla *et al.*, 2004] Nitesh V. Chawla, Lawrence O. Hall, Kevin W. Bowyer, y W. Philip Kegelmeyer. Learning ensembles from bites: A scalable and accurate approach. *Journal of Machine Learning Research*, 5:421–451, 2004.

[Christensen *et al.*, 2003] Stefan W. Christensen, Ian Sinclair, y Philippa A. S. Reed. Designing committees of models through deliberate weighting of data points. *Journal of Machine Learning Research*, 4:39–66, 2003.

[De Stefano y Montesinos, 2000] L. De Stefano y S. Montesinos. *Monitoring of Groundwater Extraction. En Application of space Techniques to the Integrated Management of river basin Water Resources*. Montesinos & Castaño (Eds.), 2000.

[Demir y Alpaydin, 2005] Cigdem Demir y Ethem Alpaydin. Cost-conscious classifier ensembles. *Pattern Recognition Letters*, 26(14):2206–2214, 2005.

[Dietterich y Bakiri, 1995] Thomas G. Dietterich y Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, 2:263–286, 1995.

[Dietterich y Kong, 1995] Thomas G. Dietterich y E.B. Kong. Machine learning bias, statistical bias, and statistical variance of decision tree algorithms. Technical report, Oregon State University, Covallis, OR, 1995.

[Dietterich, 1998a] Thomas G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1923, 1998.

[Dietterich, 1998b] Thomas G. Dietterich. Machine-learning research: Four current directions. *The AI Magazine*, 18(4):97–136, 1998.

[Dietterich, 2000a] Thomas G. Dietterich. Ensemble methods in machine learning. En *Multiple Classifier Systems: First International Workshop*, páginas 1–15, 2000.

[Dietterich, 2000b] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

[Domingos, 1997] Pedro Domingos. Knowledge acquisition from examples via multiple models. En *Proc. 14th International Conference on Machine Learning*, páginas 98–106. Morgan Kaufmann, 1997.

[Dorronsoro *et al.*, 1997] J. R. Dorronsoro, Francisco Ginel, Carmen Sánchez, y Carlos Santa Cruz. Neural fraud detection in credit card operations. *IEEE Transactions on Neural Networks*, 8(4):827–834, 1997.

[Duda *et al.*, 2001] R. O. Duda, P. E. Hart, y D. G. Stork. *Pattern Classification*. John Wiley and Sons, New York, 2ª edición, 2001.

[Efron y Tibshirani, 1994] Bradley Efron y Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC, 1994.

[Eiben y Smith, 2003] A. E. Eiben y J. E. Smith. *Introduction to evolutionary computing*. Springer-Verlag, Berlin, 2003.

[Esposito *et al.*, 1997] F. Esposito, D. Malerba, G. Semeraro, y J. Kay. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.

[Esposito y Saitta, 2003] Roberto Esposito y Lorenza Saitta. Monte carlo theory as an explanation of bagging and boosting. En *Proceeding of the Eighteenth International Joint Conference on Artificial Intelligence*, páginas 499–504. Morgan Kaufmann, 2003.

[Esposito y Saitta, 2004] Roberto Esposito y Lorenza Saitta. A monte carlo analysis of ensemble classification. En *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, páginas 265–272, New York, NY, USA, 2004. ACM Press.

[Fan *et al.*, 2003] W. Fan, H. Wang andP. S. Yu, y S. Ma. Is random model better? on its accuracy and efficiency. En *Third IEEE International Conference on Data Mining, 2003. ICDM 2003*, páginas 51–58, 2003.

[Fawcett y Provost, 1997] Tom Fawcett y Foster Provost. Adaptive fraud detection. *Data Mining and Knowledge Discovery*, 1:291–316, 1997.

[Fawcett, 2003] Tom Fawcett. "In vivo" spam filtering: A challenge problem for data mining. *KDD Explorations*, 5(2), 2003.

[Freund y Schapire, 1995] Yoav Freund y Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. En *Proc. 2nd European Conference on Computational Learning Theory*, páginas 23–37, 1995.

[Friedman, 1997] J. H. Friedman. On bias, variance, 0/1-loss, and the curse-of-dimensionality. *Data Mining and Knowledge Discovery*, 1(1):55–77, 1997.

[Fürnkranz, 2002] Johannes Fürnkranz. Round robin classification. *Journal of Machine Learning Research*, 2:721–747, 2002.

[Gama y Brazdil, 2000] João Gama y Pavel Brazdil. Cascade generalization. *Machine Learning*, 41(3):315–343, 2000.

[Gelfand *et al.*, 1991] S.B. Gelfand, C.S. Ravishankar, y E.J. Delp. An iterative growing and pruning algorithm for classification tree design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(2):138–150, 1991.

[Giacinto y Roli, 2001] Giorgio Giacinto y Fabio Roli. An approach to the automatic design of multiple classifier systems. *Pattern Recognition Letters*, 22(1):25–33, 2001.

[Grandvalet, 2004] Yves Grandvalet. Bagging equalizes influence. *Machine Learning*, 55(3):251–270, 2004.

[Grove y Schuurmans, 1998] A. Grove y D. Schuurmans. Boosting in the limit: Maximizing the margin of learned ensembles. En *Proceedings of the Fifteenth National Conference on Artifical Intelligence*, páginas 692–699, 1998.

[Hall y Samworth, 2005] Peter Hall y Richard J. Samworth. Properties of bagged nearest neighbour classifiers. *Journal of the Royal Statistical Society Series B*, 67(3):363–379, 2005.

[Haskell *et al.*, 2004] Richard E. Haskell, Charles Lee, y Darrin M. Hanna. Geno-fuzzy classification trees. *Pattern Recognition*, 37(8):1653–1659, 2004.

[Haykin, 1999] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[Ho, 1998] Tin Kam Ho. C4.5 decision forests. En *Proceedings of Fourteenth International Conference on Pattern Recognition*, volumen 1, páginas 545–549, 1998.

[Hothorn y Lausen, 2003] Torsten Hothorn y Berthold Lausen. Double-bagging: combining classifiers by bootstrap aggregation. *Pattern Recognition*, 36(6):1303–1309, junio 2003.

[Ittner y Schlosser, 1996] Andreas Ittner y Michael Schlosser. Non-linear decision trees - NDT. En *International Conference on Machine Learning*, páginas 252–257, 1996.

[Jacobs *et al.*, 1991] R. A. Jacobs, M. I. Jordan, S.J. Nowlan, y G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[Jain *et al.*, 2000] A. K. Jain, R. P. W. Duin, y Mao Jianchang. Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[Jain *et al.*, 2002] Anil K. Jain, Friederike D. Griess, y Scott D. Connell. On-line signature verification. *Pattern Recognition*, 35(12):2963–2972, 2002.

[Janikow, 1998] C. Z. Janikow. Fuzzy decision trees: issues and methods. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 28(1):1–15, 1998.

[Jensen, 1996] F. V. Jensen. *An introduction to Bayesian networks*. Taylor and Francis, London, 1996.

[Jordan y Jacobs, 1994] Michael I. Jordan y Robert A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural Computation*, 6(2):181–214, 1994.

[Kim *et al.*, 2003] Hyun-Chul Kim, Shaoning Pang, Hong-Mo Je, Daijin Kim, y Sung Yang Bang. Constructing support vector machine ensemble. *Pattern Recognition*, 36(12):2757–2767, 2003.

[Kittler *et al.*, 1998] J. Kittler, M. Hatef, R.P.W. Duin, y J. Matas. On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(3):226–239, 1998.

[Kohavi y Wolpert, 1996] Ron Kohavi y David H. Wolpert. Bias plus variance decomposition for zero-one loss functions. En *Proceedings of the 13th International Conference on Machine Learning*, páginas 275–283, 1996.

[Kong y Dietterich, 1995] E. B. Kong y Thomas G. Dietterich. Error-correcting output coding corrects bias and variance. En *Proceedings of the Twelfth International Conference on Machine Learning*, páginas 313–321, 1995.

[Kononenko, 2001] Igor Kononenko. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine*, 23(1):89–109, 2001.

[Kuncheva *et al.*, 2001] Ludmila I. Kuncheva, James C. Bezdek, y Robert P. W. Duin. Decision templates for multiple classifier fusion: an experimental comparison. *Pattern Recognition*, 34(2):299–314, 2001.

[Kuncheva y Kountchev, 2002] Ludmila I. Kuncheva y Roumen K. Kountchev. Generating classifier outputs of fixed accuracy and diversity. *Pattern Recognition Letters*, 23:593–600, 2002.

[Kuncheva y Whitaker, 2003] Ludmila I. Kuncheva y Christopher J. Whitaker. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Machine Learning*, 51(2):181–207, mayo 2003.

[Mao, 1998] Jianchang Mao. A case study on bagging, boosting and basic ensembles of neural networks for OCR. En *The 1998 IEEE International Joint Conference on Neural Networks*, volumen 3, páginas 1828–1833, 1998.

[Margineantu y Dietterich, 1997] Dragos D. Margineantu y Thomas G. Dietterich. Pruning adaptive boosting. En *Proc. 14th International Conference on Machine Learning*, páginas 211–218. Morgan Kaufmann, 1997.

[Martínez-Muñoz y Suárez, 2002] Gonzalo Martínez-Muñoz y Alberto Suárez. Using all data to generate decision tree ensembles. En *Proc. of Learning'02*, páginas 181–186, 2002.

[Martínez-Muñoz y Suárez, 2004a] Gonzalo Martínez-Muñoz y Alberto Suárez. Aggregation ordering in bagging. En *Proc. of the IASTED International Conference on Artificial Intelligence and Applications*, páginas 258–263. Acta Press, 2004.

[Martínez-Muñoz y Suárez, 2004b] Gonzalo Martínez-Muñoz y Alberto Suárez. Using all data to generate decision tree ensembles. *IEEE Transactions on Systems, Man and Cybernetics part C*, 34(4):393–397, 2004.

[Martínez-Muñoz y Suárez, 2005a] Gonzalo Martínez-Muñoz y Alberto Suárez. Comités de árboles IGP. En *Actas del I simposio de inteligencia computacional*, páginas 277–283. Thomson Press, 2005.

[Martínez-Muñoz y Suárez, 2005b] Gonzalo Martínez-Muñoz y Alberto Suárez. Switching class labels to generate classification ensembles. *Pattern Recognition*, 38(10):1483–1494, 2005.

[Martínez-Muñoz y Suárez, 2006] Gonzalo Martínez-Muñoz y Alberto Suárez. Using boosting to prune bagging ensembles. *Pattern Recognition Letters*, En revisión, 2006.

[Mason *et al.*, 2000] Llew Mason, Peter L. Bartlett, y Jonathan Baxter. Improved generalization through explicit optimization of margins. *Machine Learning*, 38(3):243–255, 2000.

[Michie *et al.*, 1994] D. Michie, D. J. Spiegelhalter, y C. C. Taylor. *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, New York, 1994.

[Mingers, 1989a] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227–243, 1989.

[Mingers, 1989b] John Mingers. An empirical comparison of selection measures for decision-tree induction. *Machine Learning*, 3(4):319–342, 1989.

[Mitchell, 1980] T. M. Mitchell. The need for biases in learning generalizations. Technical report, Rutgers University, New Brunswick, New Jersey, 1980.

[Mitchell, 1990] T. M. Mitchell. The need for biases in learning generalizations. En Jude Shavlik y Thomas G. Dietterich, editors, *Readings in Machine Learning*, The Morgan Kaufmann Series in Machine Learning, páginas 184–191. Morgan Kaufmann, 1990.

[Mitchell, 1997] T. M. Mitchell. *Machine Learning*. McGraw Hill, New York, 1997.

[Mori *et al.*, 1992] S. Mori, C. Y. Suen, y K. Yamamoto. Historical review of OCR research and development. *Proceedings of the IEEE*, 80(7):1029–1058, 1992.

[Murray *et al.*, 2005] Joseph F. Murray, Gordon F. Hughes, y Kenneth Kreutz-Delgado. Machine learning methods for predicting failures in hard drives: A multiple-instance application. *Journal of Machine Learning Research*, 6:783–816, 2005.

[Nadeau y Bengio, 2003] Claude Nadeau y Yoshua Bengio. Inference for the generalization error. *Machine Learning*, 52(3):239–281, 2003.

[Opitz y Maclin, 1999] D. Opitz y R. Maclin. Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research*, 11:169–198, 1999.

[Ortega *et al.*, 2001] Julio Ortega, Moshe Koppel, y Shlomo Argamon. Arbitrating among competing classifiers using learned referees. *Knowledge and Information Systems*, 3(4):470–490, 2001.

[Pearl, 1988] Judea Pearl. *Probabilistic reasoning in intelligent systems networks of plausible inference*. Morgan Kaufmann, 1988.

[Prodromidis y Stolfo, 2001] Andreas L. Prodromidis y Salvatore J. Stolfo. Cost complexity-based pruning of ensemble classifiers. *Knowledge and Information Systems*, 3(4):449–469, 2001.

[Pudil *et al.*, 1992] P. Pudil, J. Novovicova, S. Blaha, y J. Kittler. Multistage pattern recognition with reject option. En *Proc. 11th IAPR Int. Conf. Pattern Recognition*, volumen 2, páginas 92–95, 1992.

[Quinlan, 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Quinlan, 1993] J. R. Quinlan. *C4.5 programs for machine learning*. Morgan Kaufmann, 1993.

[Quinlan, 1996a] J. R. Quinlan. Bagging, boosting, and C4.5. En *Proc. 13th National Conference on Artificial Intelligence*, páginas 725–730, Cambridge, MA, 1996.

[Quinlan, 1996b] J. R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.

[Quinlan, 1998] J. R. Quinlan. Miniboosting decision trees. En *Proceedings of Fifteenth National Conference on Artificial Intelligence*. AAAI Press, 1998.

[Rätsch *et al.*, 2001] G. Rätsch, T. Onoda, y K.-R. Müller. Soft margins for AdaBoost. *Machine Learning*, 42(3):287–320, marzo 2001.

[Rätsch *et al.*, 2002] G. Rätsch, S. Mika, B. Scholkopf, y K.-R. Muller. Constructing boosting algorithms from svms: an application to one-class classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1184–1199, 2002.

[Ross, 1987] S. M. Ross. *Introduction to probability and statistics for engineers and scientists*. John Wiley & Sons, 1987.

[Salzberg, 1997] S. L. Salzberg. On comparing classifiers: pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1:317–328, 1997.

[Schapire *et al.*, 1998] Robert E. Schapire, Yoav Freund, Peter L. Bartlett, y W. S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, 12(5):1651–1686, 1998.

[Schapire y Singer, 2000] Robert E. Schapire y Yoram Singer. Boostexter: A boosting-based system for text categorization. *Machine Learning*, boosting(2-3):135–168, 2000.

[Schapire, 1990] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.

[Schapire, 1997] Robert E. Schapire. Using output codes to boost multiclass learning problems. En *Proc. 14th International Conference on Machine Learning*, páginas 313–321. Morgan Kaufmann, 1997.

[Sharkey, 1999] A. J. C. Sharkey. *Combining Artificial Neural Nets: Ensemble and Modular Multi-Net Systems*. Springer-Verlag, London, 1999.

[Skurichina y Duin, 1998] Marina Skurichina y Robert P. W. Duin. Bagging for linear classifiers. *Pattern Recognition*, 31(7):909–930, julio 1998.

[Skurichina y Duin, 2002] Marina Skurichina y Robert P. W. Duin. Bagging, boosting and the random subspace method for linear classifiers. *Pattern Analysis & Applications*, 5(2):121–135, 2002.

[Stamatatos y Widmer, 2005] Efstathios Stamatatos y Gerhard Widmer. Automatic identification of music performers with learning ensembles. *Artificial Intelligence*, 165(1):37–56, 2005.

[Stroustrup, 1997] Bjarne Stroustrup. *The C++ programming language*. Addison-Wesley, 1997.

[Suárez y Lutsko, 1999] Alberto Suárez y J.F. Lutsko. Globally optimal fuzzy decision trees for classification and regression. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1297–1311, 1999.

[Tamon y Xiang, 2000] Christino Tamon y Jie Xiang. On the boosting pruning problem. En *Proc. 11th European Conference on Machine Learning*, volumen 1810, páginas 404–412. Springer, Berlin, 2000.

[Tapiador Mateos *et al.*, 2005] Marino Tapiador Mateos, Juan A. Siguenza Pizarro, y otros autores. *Tecnologías biométricas aplicadas a la seguridad*. Ra-ma, 2005.

[Theodoridis, 2003] S. Theodoridis. *Pattern recognition*. Academic Press, 2003.

[Todorovski y Džeroski, 2003] Ljupčo Todorovski y Sašo Džeroski. Combining classifiers with meta decision trees. *Machine Learning*, 50(3):223–249, 2003.

[Tumer y Ghosh, 1996] Kagan Tumer y Joydeep Ghosh. Error correlation and error reduction in ensemble classifiers. *Connection Science*, 8(3-4):385–403, 1996.

[Valentini y Dietterich, 2004] Giorgio Valentini y Thomas G. Dietterich. Bias-variance analysis of support vector machines for the development of svm-based ensemble methods. *Journal of Machine Learning Research*, 5:725–775, 2004.

[Vapnik, 1995] Vladimir Vapnik. *The nature of statistical learning theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1995.

[Webb, 2000] Geoffrey I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, 40(2):159–196, agosto 2000.

[Wolpert y Macready, 1999] David H. Wolpert y William G. Macready. An efficient method to estimate bagging's generalization error. *Machine Learning*, 35(1):41–55, 1999.

[Wolpert, 1990] David H. Wolpert. Stacked generalization. Technical Report LA-UR-90-3460, Los Alamos, NM, 1990.

[Wolpert, 1995] David H. Wolpert. The relationship between PAC, the statistical physics framework, the bayesian framework and the vc framework. En *The Mathematics of Generalization*, páginas 117–214. Addison-Wesley, 1995.

[Zhou *et al.*, 2002] Z.-H. Zhou, J. Wu, y W. Tang. Ensembling neural networks: Many could be better than all. *Artificial Intelligence*, 137(1-2):239–263, 2002.

[Zhou y Tang, 2003] Z.-H. Zhou y W. Tang. Selective ensemble of decision trees. En *Lecture Notes in Artificial Intelligence*, páginas 476–483, Berlin: Springer, 2003.