Universidad Autónoma de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Informática

Aplicación de Perceptrones Paralelos y AdaBoost a Problemas de Clasificación de Muestra Extrema

Trabajo de investigación presentado para
optar al Diploma de Estudios Avanzados

Por

Iván Cantador Gutiérrez

bajo la dirección de

José R. Dorronsoro Ibero

Madrid, junio de 2005

Contenido

Índice de figuras

Índice de tablas

Resumen

En los últimos años los métodos empleados en Aprendizaje Automático y Minería de Datos, así como la aceptación de los mismos, han avanzado hasta un punto en el que están siendo aplicados en "problemas del mundo real" de forma muy habitual. En este avance se ha ido prestando progresivamente más atención e investigación a un tipo de problemas muy concreto: los problemas de clasificación desequilibrados o problemas de Muestra Extrema. En ellos el comportamiento de una clase de interés se encuentra oscurecido por el de una gran mayoría de datos pertenecientes a otras, mucho más dominantes estadísticamente, que en ocasiones pueden provocar que la distribución de la primera resulte invisible.

Para abordarlos se desarrollaron estrategias que alteran y equilibran la distribución de clases: *submuestreo* o eliminación de ejemplos de la clase mayoritaria, y *sobremuestreo* o replicación de ejemplos de la clase minoritaria. Sin embargo, estos métodos tienen inconvenientes. Mientras que el submuestreo puede dar lugar a la pérdida de datos potencialmente útiles para el aprendizaje, el sobremuestreo incrementa el tamaño del conjunto de entrenamiento, y por tanto el tiempo necesario para construir el clasificador, a la vez que puede provocar sobreajuste en el entrenamiento por emplear copias exactas de los patrones minoritarios. Investigaciones más recientes se han centrado en desarrollar estrategias que inteligentemente seleccionan para el entrenamiento sólo aquellos ejemplos que están cerca de la frontera de la clasificación. De este modo, buscan eliminar aquellos patrones "redundantes", fáciles de clasificar, que no contribuyen a la construcción del clasificador, y los patrones "ruidosos", de etiquetado erróneo, que estorban en el aprendizaje.

En este trabajo se usarán los novedosos Perceptrones Paralelos, y el concepto de margen sobre la frontera de clasificación que surge en las activaciones de los perceptrones durante el entrenamiento, para proponer una técnica alternativa que identifique los patrones "redundantes" y "ruidosos" a ser eliminados de los conjuntos de datos. Además, teniendo como objetivo la mejora del método anterior, éste se acoplará al algoritmo AdaBoost, uno de los mecanismos de combinación de clasificadores más empleado en la actualidad. Los resultados empíricos obtenidos con ambas estrategias mostrarán un alcance, y mejora en los casos más desequilibrados, de los proporcionados por los robustos Perceptrones Multicapa.

Abstract

In the last years the methods developed in Machine Learning and Data Mining, as well as their acceptance, have advanced until a point where they are being widely applied for "problems of the real world". In this setting a very specific type of problems has received progressively more attention from the scientific community: the so-called unbalanced classification problems. For them, the behaviour of a class of interest is statistically overwhelmed by others and its distribution may sometimes seem invisible to the learning algorithms.

In order to address these problems a number of strategies that attempt to balance the class distributions have been applied: *subsampling* or elimination of majority class examples, and *oversampling* or replication of minority class examples. However, these techniques have several disadvantages. While subsampling methods could produce the loss of very useful data for learning, oversampling strategies augment the dataset sizes, and therefore the needed training time, and could present overfitting, due to the use of similar copies of the minority patterns. More recent researches have focused their effort on strategies that intelligently select for training only those patterns close to the classification boundaries. Thus, they attempt to detect and remove the *redundant* examples which are safe, in the sense of being well represented in the training sample by many other similar patterns, and also the *noisy* examples that may have an incorrect label and obstruct the construction of the models.

In this work we propose the use of Parallel Perceptrons, a novel approach to the well known classical committe machines, and the concept of activation margins that naturally arises during their learning processes. We shall use these margins to detect redundant and noisy examples, and delete them from the training samples. Moreover, to further improve on the previous approach, we shall combine it with AdaBoost, one of the most used and studied ensemble methods at present. The empirical results obtained with both techniques are comparable and even better in some cases, with those provided by the stronger Multilayer Perceptrons.

Capítulo 1

Introducción

De gran interés son los problemas de clasificación difíciles originados en el "mundo real" que pueden agruparse bajo el epígrafe de Muestras Extremas (ME), y que se caracterizan por tener clases de interés cuyo número de datos es mucho menor que el de otras. La comunidad científica que trabaja en Aprendizaje Automático (del inglés *Machine Learning*, ML) asumía que la distribución de clases natural era la mejor para el aprendizaje. Sin embargo, esta asunción fue descartada al comprobarse (e.g. [43]) el efecto negativo que el desequilibrio existente entre el número de ejemplos minoritarios y mayoritarios puede provocar en la eficacia de los clasificadores (*véase sección 2.1*). Este hecho fue abordado entonces mediante dos tipos de enfoques diferentes. El primero consiste en el remuestreo de los conjuntos de entrenamiento para equilibrar el número de representantes de cada clase (*véanse secciones 2.2 y 2.3*), y el segundo se basa en la asignación de costes a los patrones, de tal modo que en el proceso de entrenamiento se premie la correcta clasificación de aquellos con mayores costes (*véase sección 2.4*).

El estudio las estrategias anteriores y la propuesta de otras nuevas para el tratamiento de ME son los temas centrales de este trabajo de investigación. A continuación se concretarán su motivación y objetivos generales, se resumirán sus contribuciones y se dará una visión general del presente documento.

1.1 Motivación

Las técnicas básicas de equilibrado de clases (*véase sección 2.2*) consistentes en la eliminación de ejemplos mayoritarios y en la replicación de ejemplos minoritarios del conjunto de entrenamiento presentan inconvenientes. Mientras que las primeras tienen el riesgo de que se pierda información relevante para el aprendizaje, las segundas incrementan el tamaño de los conjuntos de datos, y por tanto el tiempo necesario para el entrenamiento, además de

poder dar lugar a sobreajuste, por hacer uso de replicas de ejemplos minoritarios. Como alternativa, se propone una nueva estrategia basada en seleccionar para el entrenamiento sólo aquellos patrones que están cerca de la frontera de clasificación (*véase sección 2.3*), y en eliminar del conjunto de datos (a) los ejemplos "redundantes", fáciles de aprender, que no contribuyen en la construcción del clasificador, y (b) los ejemplos "ruidosos", de etiquetado erróneo, que estorban en el aprendizaje.

El problema que surge entonces es el de establecer una técnica que permita definir ambas categorías de patrones, y es esta cuestión la que motiva el trabajo a desarrollar.

1.2 Objetivos generales de la investigación

Para determinar si un ejemplo dado es redundate, ruidoso o está cerca de la frontera de clasificación, se propone como enfoque el uso del nuevo método de clasificación introducido por Auer et al. [1, 2] denominado Perceptrón Paralelo (*véase capítulo 3*). Consistente en un conjunto de perceptrones estándar, este modelo calcula en el entrenamiento un margen para la activación de los perceptrones usados.

El estudio de la manera en la que emplear ese margen para medir la relevancia de los patrones durante la construcción del clasificador, y para la asignación de estos a una de las categorías arriba citadas, con el fin de establecer una estrategia de selección de patrones de entrenamiento (PPTSS), supone el primer objetivo de la investigación (*véase sección 5.1*).

Por otra parte, teniendo como motivación la mejora de los resultados obtenidos con el mecanismo de selección de patrones de entrenamiento propuesto, se establece como segundo objetivo su adaptación (PPBoost) a técnicas de agregación de clasificadores, en concreto al bien conocido algoritmo AdaBoost de Freund y Schapire [23, 24, 39, 25, 33] (*véanse capítulo 4 y sección 5.2*).

1.3 Contribuciones de la investigación

Los resultados obtenidos (*véase capítulo 6*) demuestran empíricamente que las dos técnicas desarrolladas son comparables, y mejores en los casos más desequilibrados, a métodos de aprendizaje tan robustos como los Perceptrones Multicapa [5, 20]. Además, suponiendo novedosas ideas para estrategias de equilibrado de clases en los conjuntos de entrenamiento, aseguran una buena generalización en la clasificación, al proporcionar no sólo un buen porcentaje de aciertos global (*accuracy*) en los conjuntos de test, sino también en los obtenidos individualmente para las clases minoritaria y mayoritaria, hecho fundamental para resolver satisfactoriamente problemas de clasificación desequilibrados.

Estas propiedades, junto con el muy rápido entrenamiento de los Perceptrones Paralelos, podrían hacer de ellas útiles para abordar problemas de Muestras Extremas con alta dimensionalidad, un área de considerable interés en la actualidad (*véase capítulo 7*).

1.4 Estructura del documento

El esquema que se ha seguido en este documento coincide con las fases del trabajo realizado. Inicialmente se plantea el problema a tratar, la **clasificación de Muestras Extremas**, y se describe el estado del arte de las propuestas que se han dado para abordarlo. En una de ellas, la selección de patrones cercanos a la frontera de clasificación como conjunto de entrenamiento, se plantea si el uso de los márgenes que surgen en el entrenamiento del **Percpetrón Paralelo** es viable para el filtrado de ejemplos. Una vez estudiada, la alternativa se intenta acoplar a técnicas de agregación de clasificadores, en concreto al algoritmo **AdaBoost**, con el fin de mejorar los resultados obtenidos.

El texto se ha dividido en siete capítulos, incluyendo éste introductorio. Cada uno de ellos comienza con una motivación e introducción de los puntos que se tratarán y un pequeño párrafo que describe su estructura interna.

- El **capítulo 2** describe el problema de la clasificación de Muestras Extremas, revisa los tipos de técnicas más importantes que se han aplicado para tratarlo y plantea las medidas de eficacia adecuadas para la comparación de estas últimas.
- El **capítulo 3** describe el Perceptrón Paralelo: su arquitectura y su regla de aprendizaje, así como aspectos prácticos de su implementación.
- El **capítulo 4** motiva el uso de métodos de agregación de clasificadores como mejora de la eficacia, expone las principales estrategias de combinación de modelos y de ellas explica más en profundidad el algoritmo AdaBoost.
- El **capítulo 5** describe las dos propuestas del trabajo para abordar problemas de clasificación de Muestra Extrema. La primera (PPTSS), basada en la selección de conjuntos de entrenamiento a través del Perceptrón Paralelo, y la segunda (PPBoost), establecida como una adaptación de esa selección a AdaBoost.
- El **capítulo 6** expone los resultados obtenidos con las propuestas, comparándolos con los del Perceptrón Multicapa y su agregación mediante varios clasificadores.
- El **capítulo 7** contiene finalmente las conclusiones del trabajo: las contribuciones, una comparación con investigaciones previas realizadas en el área de la clasificación de Muestras Extremas, y las líneas abiertas para trabajo futuro.

Capítulo 2

Muestras Extremas

El Aprendizaje Automático es el área de la Inteligencia Artificial que se ocupa de desarrollar algoritmos capaces de aprender a partir de ejemplos, y constituye, junto con la Estadística, el corazón del análisis inteligente de la información. En este análisis se pueden llevar a cabo dos tipos de tareas: *tareas predictivas* (clasificación, regresión, categorización, etc.), en las que se predice uno o más valores asociados a uno o más ejemplos no considerados en el aprendizaje, y *tareas descriptivas*, en las que se describen de alguna manera (agrupamiento o *clustering* de ejemplos, correlaciones de atributos, reglas de asociación, etc.) los datos existentes.

De todas ellas la clasificación de patrones es el centro de atención de este trabajo. Las muestras se presentan como un conjunto de pares $\{(\mathbf{x}, y_{\mathbf{x}}) : \mathbf{x} \in \mathcal{X}, y_{\mathbf{x}} \in \mathcal{Y}\}$, siendo $\mathbf{x}$ el vector con los valores de los atributos de un ejemplo e $y_{\mathbf{x}}$ la clase, categoría o etiqueta del mismo. El objetivo es aprender la función $y : \mathcal{X} \rightarrow \mathcal{Y}$, denominada clasificador, que represente la correspondencia existente en los ejemplos, es decir, que para cada vector de $\mathcal{X}$ proporcione un único valor de $\mathcal{Y}$. Por simplicidad, a partir de ahora sólo se considerarán problemas en los que la etiqueta $y_{\mathbf{x}}$ toma uno de dos valores posibles $\{-1, +1\}$. Como escenario típico se asume la existencia de un conjunto de patrones de entrenamiento empleado para inducir el clasificador, y un conjunto de test independiente con el que se determina su eficacia.

En este contexto, una complicación interesante surge cuando el conjunto de entrenamiento está desequilibrado, en el sentido de que la clase de interés (minoritaria o *positiva*) posee muchos menos representantes que la otra clase (mayoritaria o *negativa*), oscureciendo o dificultando la clasificación de la primera. Esta situación de **Muestras Extremas** se da en muchas aplicaciones reales, como por ejemplo la detección de casos fraudulentos en transacciones con tarjetas de crédito [17] y llamadas telefónicas [22], los diagnósticos médicos de enfermedades poco usuales [28, 45], el reconocimiento de derramamientos de aceite en

imágenes por satélite de la superficie del mar [30], la gestión de las telecomunicaciones [21] o la categorización de textos [27, 41].

Para abordar este tipo de problemas se desarrollaron inicialmente técnicas orientadas a equilibrar el número de muestras de entrenamiento de cada clase, realizando *sobremuestreos* de la clase minoritaria [37] o *submuestreos* de la clase mayoritaria [44, 3]. Sin embargo, como se explicará más adelante, estos métodos poseen inconvenientes, y por ello se han ido planteado nuevas estrategias de muestreo [29, 19], inspiradas en las Máquinas de Vectores Soporte (del inglés *Support Vector Machines*, SVM) [8], que progresivamente sólo emplean para el entrenamiento aquellos patrones que definen la frontera de clasificación o están cerca de ella.

En este capítulo se describirá brevemente el efecto que un conjunto de entrenamiento desequilibrado puede tener sobre la clasificación. Se comentarán las técnicas clásicas de equilibrado de clases, sus inconvenientes y algunas modificaciones que se han realizado en ellas para mejorarlas, así como las nuevas estrategias de muestreo basadas en la selección de patrones de entrenamiento cercanos a la frontera de clasificación, y las técnicas de clasificación sensibles a costes. Finalmente, se discutirá qué medidas de eficacia son adecuadas cuando se trabaja con muestras extremas.

2.1 El efecto de la distribución de clases en el aprendizaje

La creación de un conjunto de datos adecuado y el posterior aprendizaje que se haga a partir de él supone diversos costes. Por una parte, en la creación del conjunto de datos se tienen que asumir costes debidos a la propia obtención de los datos, a su limpieza y filtrado, a su transporte y almacenaje, y a su etiquetado y transformación en una forma adecuada para el aprendizaje. Por su parte, el proceso de aprendizaje incluye, entre otros, costes debidos a los recursos computacionales y al tiempo empleados.

Dados estos costes, a menudo se hace necesario limitar el tamaño del conjunto de entrenamiento. El problema por supuesto radica en cómo llevarlo a cabo. Se hace esencial seleccionar los patrones cuidadosamente con el fin de minimizar el impacto que la limitación de información puede tener sobre la eficacia del clasificador. Para ello una importante elección es la distribución de clases apropiada que debe emplearse. En el campo del Aprendizaje Automático se asumía que la distribución de clases natural es la mejor para el aprendizaje. Sin embargo, esta asunción ha ido progresivamente decayendo, debido al aumento del uso de conjuntos de datos con alto grado de desequilibrio en las clases, y la comprobación del efecto negativo en la clasificación que puede provocar el desequilibrio entre el número de representantes minoritarios y mayoritarios.

Persiguiendo esta idea, Weiss y Provost en [43] muestran que la distribución de clases original no es a menudo la mejor para el aprendizaje y que se puede alcanzar un rendimiento sustancialmente mejor usando distribuciones diferentes. A través de sus experimentos, el hecho de que los clasificadores tienen peores resultados sobre la clase minoritaria quedaría justificado por dos observaciones:

- *Las "reglas" de clasificación que predicen la clase minoritaria tienden a dar un error mucho mayor que aquellas que predicen la clase mayoritaria.* En primer lugar porque las reglas que predicen la clase minoritaria, al haber sido construidas con muchos menos ejemplos, son menos precisas, y en segundo lugar porque la propia distribución de clases del conjunto de test, con muchos más ejemplos negativos, hace que haya más casos en los que poder clasificar un ejemplo como positivo de forma incorrecta.

- *Los ejemplos de test que pertenecen a la clase minoritaria son clasificados incorrectamente más frecuentemente que los ejemplos de test que pertenecen a la clase mayoritaria.* Una primera razón es que las probabilidades marginales de las clases en las distribuciones naturales están sesgadas fuertemente a favor de la clase mayoritaria. Aquellos algoritmos de aprendizaje que las emplean tienden a predecir la clase mayoritaria más de lo que harían en condiciones de equiprobabilidad. Una segunda razón se debe a que un clasificador es menos propenso a definir adecuadamente los límites de la clase minoritaria en el espacio de atributos al haber menos representantes de ésta.

Además de estas justificaciones, en el mismo trabajo, los autores determinan las mejores distribuciones de entrenamiento para un gran número de conjuntos de datos. Empleando C4.5 como algoritmo de aprendizaje, y el área bajo las curvas ROC (*Area Under the Curve*, AUC) como medida de eficacia, llegan a la conclusión de que las distribuciones óptimas para los problemas abordados debían contener en general entre un 50% y un 90% de ejemplos minoritarios, proporciones muy diferentes a las existentes en las muestras iniciales.

2.2 Técnicas básicas de equilibrado de clases

Mientras que ha habido poca investigación sobre el efecto que la distribución de clases provoca en la eficacia de un método de clasificación, existe un número considerable de trabajos sobre la manera en la que construir un "buen" clasificador cuando la distribución de los datos está altamente desequilibrada y es costoso clasificar los ejemplos de la clase minoritaria.

Una primera aproximación es la modificación de la distribución de clases en el conjunto de entrenamiento con el fin de equilibrar el número de patrones de cada clase, pudiendo distinguir dos tipos de estrategias:

- **Submuestreo** (*undersampling*) o eliminación de ejemplos mayoritarios. En [44] Wilson propone usar la regla de vecinos más cercanos (del inglés *Nearest Neighbors*, NN) como mecanismo de submuestreo. En pocas palabras, su estrategia consiste en aplicar el clasificador k–NN para estimar la etiqueta de todos los patrones de entrenamiento y descartar aquellos cuya clase no se corresponde con la clase asociada al mayor número de los k vecinos más cercanos.

  A pesar de los importantes resultados obtenidos, esta técnica no produce reducciones significativas de la clase mayoritaria. El procedimiento trabaja aplicando un clasificador k–NN que, por supuesto, también se ve afectado por el desequilibrio de clases existente, hecho que induce una tendencia a encontrar vecinos mayoritarios. Para abordar esta situación, en [3] se añade como complemento de la medida de la distancia euclídea entre patrones el uso un factor que dé mayor peso a las distancias asociadas a ejemplos minoritarios.

  Esta estrategia obtiene mejores resultados, pero no acaba con el inconveniente de que este tipo de técnicas eliminan datos potencialmente útiles para el aprendizaje. Kubat y Matwin [29] desarrollaron entonces un mecanismo de submuestreo que inteligentemente elimina sólo aquellos ejemplos mayoritarios que son "redundantes" o que "bordean" ejemplos minoritarios, suponiendo que estos últimos son el resultado de ruido en las muestras.

- **Sobremuestreo** (*oversampling*) o replicación de ejemplos minoritarios. El problema que plantean este tipo de técnicas es el incremento del tamaño del conjunto de entrenamiento y por tanto del tiempo necesario para construir el clasificador. Además, debido a que el sobremuestreo típicamente realiza copias exactas de los ejemplos de la clase minoritaria, el sobreajuste (*overfitting*) es más propenso a ocurrir.

  Para reducirlo, Chawla et al. [13] combinan métodos de submuestreo y sobremuestreo, y en vez de sobremuestrear replicando ejemplos minoritarios, forman nuevos ejemplos positivos mediante una interpolación de ejemplos minoritarios cercanos. Esencialmente, para cada vector de atributos de la clase minoritaria se elige aleatoriamente otro entre los k ejemplos positivos más cercanos. Se toma la diferencia entre ellos y se multiplica esta última por un número aleatorio entre 0 y 1. El vector resultante se suma entonces al vector considerado obteniendo una nueva muestra a incluir en el conjunto de entrenamiento.

En un estudio reciente, Barandela et al. [4] concluyeron que cuando el desequilibrio no es muy severo, las técnicas de submuestreo son la mejor opción, y que sólo cuando el *ratio* mayoritarios/minoritarios es muy grande es apropiado sobremuestrear los ejemplos positivos.

Como aproximación alternativa, Chan y Stolfo [12] proponen realizar experimentos preliminares con el fin de determinar la distribución de clases óptima para el entrenamiento (con respecto a una función de coste específica). Una vez obtenida, se generan con ella múltiples conjuntos de entrenamiento, lo cual se complementa, en la mayoría de los casos, con incluir todos los ejemplos minoritarios y sólo algunos mayoritarios en cada conjunto de entrenamiento. A continuación, se ejecuta el algoritmo de aprendizaje en cada conjunto de datos y se combinan los clasificadores generados para formar un modelo compuesto.

2.3 Técnicas de equilibrado de clases basadas en patrones frontera

Una curva del error de entrenamiento típica en una red neuronal tiene una forma exponencial, mostrando un rápido descenso del error en las épocas iniciales, seguido de una cola suave y larga en la que el error desciende muy despacio. En la parte final del entrenamiento el hecho de presentar a la red vectores (patrones) que están alejados de la frontera de clasificación no tiene a menudo influencia sobre los parámetros de la red. En Perceptrones Multicapa (PMC) [5, 20] estos vectores pertenecen a la región donde las salidas de las neuronas están en las zonas saturadas de la función de activación, dando lugar de este modo a gradientes que tienden a desaparecer. Sólo aquellos vectores cercanos a la frontera de clasificación tienen influencia significativa, con grandes gradientes que inducen la activiación de algunas neuronas cerca de sus valores umbrales o de salidas dentro de la parte lineal de las funciones sigmoidales.

En contraste, el aprendizaje realizado por las SVM tiene en cuenta inicialmente todos los vectores de entrenamiento, pero progresivamente la influencia de aquellos vectores que están lejos de la frontera de clasificación va disminuyendo, y cuando el entrenamiento está finalizando sólo un pequeño porcentaje de vectores (soporte) que están cerca del hiperplano de decisión se mantiene. Por otra parte, debido a que las SVM seleccionan los vectores que maximizan el margen existente con el hiperplano de decisión, se consigue un mayor grado de generalización en la clasificación. De este modo, esta aproximación no sólo contribuye a un incremento de la velocidad de entrenamiento, sino también a un aumento de la precisión (*accuracy*) obtenida.

Bajo el epígrafe de "técnicas de equilibrado de clases basadas en patrones frontera" se engloban aquí aquellos métodos de reducción de conjuntos de entrenamiento que buscan quedarse con aquellos patrones que están cerca de la frontera de decisión para la creación de los clasificadores. Estos métodos de reciente aparición no son muchos y han motivado el presente trabajo de investigación.

En [18] Duch selecciona patrones frontera usando las distancias entre vectores de diferentes clases. En problemas de dos clases, para cada vector de la primera clase se selecciona el vector más cercano de la segunda clase. El proceso se realiza entonces con los ejemplos de la segunda clase. Repitiendo lo anterior varias veces, al final sólo se mantendrán aquellos vectores que tengan vecinos cercanos de la clase opuesta. Para grandes conjuntos de datos este método es muy costoso, y no está claro el momento en el que se ha de realizar el cambio de entrenar con todos los vectores restantes a hacerlo sólo con los vectores frontera seleccionados.

El análisis sensible a perturbaciones en las entradas también se ha usado para visualizar y analizar las fronteras de clasificación. Éste es una forma de aprendizaje activo en el que el algoritmo de entrenamiento tiene influencia de la parte del espacio de entradas de la que proviene la información. Zhang [46] ha desarrollado un algoritmo de aprendizaje selectivo incremental que comienza con un subconjunto de entrenamiento aleatorio. Después del entrenamiento se evalúan los ejemplos restantes disponibles, y aquellos que dan un mayor error se añaden al conjunto de entrenamiento actual. Röbel [38] también ha descrito un "algoritmo de selección dinámico" incremental en el que los patrones disponibles con mayor error son incrementalmente añadidos al conjunto de entrenamiento, a la vez que se realiza una evaluación de la generalización usando un conjunto de validación.

Duch propone en [19] un método más sencillo en el que, de forma similar a la aproximación de las SVM, se parte de un conjunto con todos los patrones de entrenamiento y después de pocas épocas los vectores que no contribuyen mucho en el proceso de aprendizaje son eliminados. Los vectores que se mantienen son aquellos que dan un error dentro de un rango establecido. De este modo, se eliminan los patrones que son *redundantes* en el entrenamiento, al clasificarse siempre bien y tener un error muy bajo, y los que son *ruidosos*, pues siempre se clasifican incorrectamente y poseen un error muy alto. Cantador y Dorronsoro [9, 10] emplean el margen asociado al entrenamiento de los Perceptrones Paralelos para definir los patrones redundantes y ruidosos, y aplicar su algoritmo de filtrado. Éste tiene en cuenta el desequilibrio de clases dando más relevancia a los patrones frontera positivos y es explicado en profundidad en la *sección 5.1*. Su adaptación [11] a estrategias de combinación de clasificadores para considerar el voto mayoritario de varios componentes mejorará los resultados del mecanismo anterior y se describirá en la *sección 5.2*.

2.4 Técnicas de clasificación sensibles a costes

Cambiar la distribución de clases no es la única estrategia para mejorar la eficacia de un clasificador cuando se tratan conjuntos de datos desequilibrados. En problemas de ME el coste de clasificar incorrectamente un ejemplo de la clase minoritaria es típicamente mucho mayor que el coste de clasificar incorrectamente uno de la clase mayoritaria. Consecuentemente, métodos de aprendizaje sensibles a los costes [34], que construyan clasificadores minimizando el coste en vez del error cometido podrían ser usados para abordar problemas desequilibrados.

De hecho, para algunos autores como Weiss y Provost [43] los métodos de aprendizaje sensibles a costes son una propuesta más directa y apropiada que modificar artificialmente la distribución de clases vía submuestro o sobremuestreo, siempre que, por supuesto, el coste de adquirir y aprender del conjunto de datos disponible no sea determinante.

2.5 Medidas de eficacia adecuadas para Muestras Extremas

La manera en la que se determinan las ventajas de las diferentes variantes de los sistemas de aprendizaje es un aspecto muy importante a tener en cuenta. Para definir el criterio de eficacia se emplea la matriz de confusión o contingencia (*confusion matrix*), mostrada en la tabla 2.5.1. En ella TP (*true positive*) y TN (*true negative*) son respectivamente el número de ejemplos positivos y negativos clasificados correctamente, mientras que FP (*false positive*) es el número de ejemplos negativos clasificados como positivos y FN (*false negative*) el de ejemplos positivos clasificados como negativos.

	predichos +	predichos −
reales +	TP	FN
reales −	FP	TN

Tabla 2.5.1: Matriz de confusión/contingencia para problemas de clasificación de dos clases.

La medida de eficacia estándar en ML es el porcentaje de ejemplos correctamente clasificados (*accuracy* a partir de ahora) calculado como

$$acc = \frac{TP + TN}{TP + FN + FP + TN} \tag{2.5.1}$$

Sin embargo, esta medida no es apropiada en aplicaciones donde las clases no están igualmente representadas en el conjunto de entrenamiento. Para verlo, considérese un problema de detección de fraude en tarjetas de crédito en el que la frecuencia de transacciones no fraudulentas es del 96%. Un clasificador que etiquetase todas las transacciones como no

fraudulentas alcanzaría un *accuracy* del 96%. Aunque este valor pareciese alto, el clasificador sería inútil, pues fracasaría totalmente en el objetivo principal de detectar operaciones fraudulentas. Por otro lado, un clasificador que alcanzase un 94% de aciertos tanto al clasificar transacciones no fraudulentas como fraudulentas tendría un *accuracy* menor, pero sería considerado como altamente exitoso, pues pocas serían las operaciones fraudulentas no detectadas y las falsas alarmas de fraude provocadas.

Informalmente y siguiendo el ejemplo anterior, se quiere (a) presentar al usuario tantos casos de fraude como sean posibles, siempre que (b) el número de falsas alarmas no sea demasiado grande. Las curvas ROC (*Receiver Operating Characteristic*) explicadas por Swets en [40] se usan para medir el compromiso entre ambos requisitos. Muestran el porcentaje de ejemplos positivos reconocidos correctamente $\frac{TP}{TP+FN}$ en función del porcentaje de ejemplos negativos mal clasificados $\frac{FP}{TN+FP}$. El número de predicciones positivas puede incrementarse a costa de aumentar el número de falsas alarmas (y viceversa), con el fin de obtener un clasificador óptimo para el usuario. El area bajo la curva ROC (*Area Under the Curve*, AUC) es de hecho una de las métricas de eficacia mejor aceptadas. La figura 2.5.1 muestra la representación de 3 curvas ROC. El área bajo ellas es la medida que las caracteriza y que permite definir sus diferentes grados de bondad. Es fácil entender que la curva superior (con más AUC) es la mejor, y que la curva inferior (con menos AUC) es la peor de las tres.

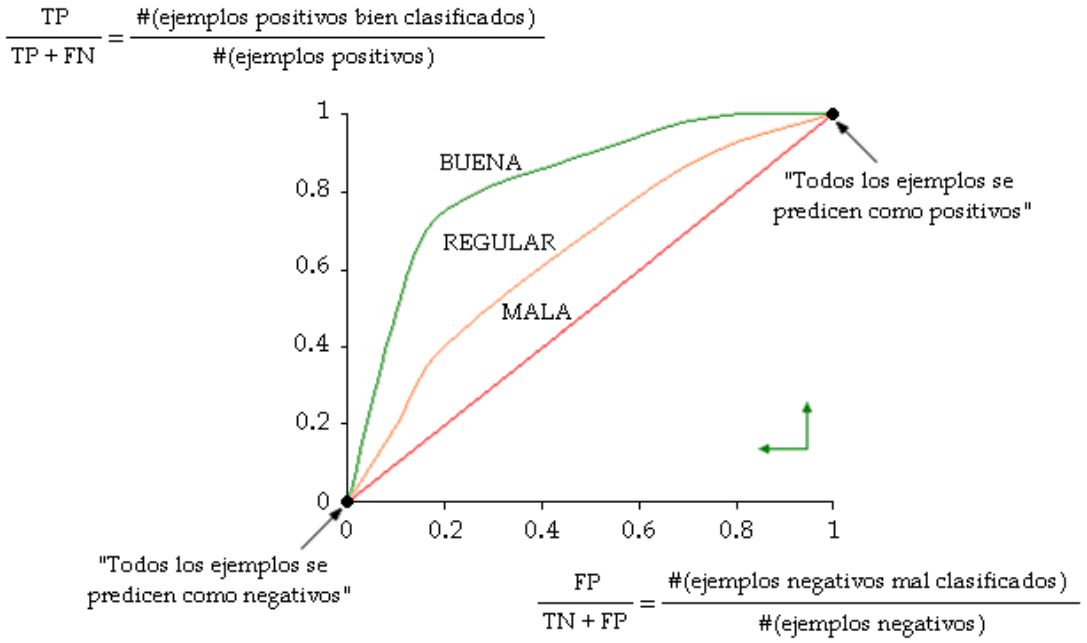

Figura 2.5.1: Ejemplo de 3 curvas ROC con diferentes grados de bondad según la métrica AUC.

Por otra parte, para medir la eficacia en entornos con clases desequilibradas, la comunidad científica que trabaja en temas de Recuperación de Información emplea la precisión (*precision*, $p = \frac{TP}{TP+FP}$) y el alcance (*recall*, $r = \frac{TP}{TP+FN}$)). La precisión mide la probabilidad de que si un sistema clasifica un ejemplo en cierta categoría, el ejemplo realmente pertenezca a la categoría, mientras que el alcance mide la probabilidad de que si un ejemplo pertenece a cierta categoría, el sistema lo asigne a la categoría. Para combinarlas, se emplean normalmente la media geométrica

$$\sqrt{p \cdot r}$$

o la más sofisticada métrica F_β de Lewis y Gale [31] definida como

$$F_\beta = \frac{(1+\beta^2) \cdot p \cdot r}{\beta^2 \cdot p + r}$$

donde el parámetro β controla la importancia relativa entre las dos medidas. Si $\beta = 1$, el valor más común para esta medida, F_β es la media armónica de ambas.

Como inconveniente de las curvas ROC, Kubat et al. [30] explican que los usuarios finales no tienen porqué disponer de una manera preprogramada de condensar la curva en una medida de eficacia única. En vez de ello, tendrían que estar capacitados para elegir el punto de la curva que mejor se ajustase a sus necesidades teniendo en cuenta el compromiso entre ejemplos positivos clasificados correctamente y ejemplos negativos clasificados incorrectamente.

Aunque en general el reto es construir un sistema que produzca clasificadores sobre el rango de amplitud máxima de la curva ROC, en ocasiones se hace necesaria una medida de eficacia que proporcione un *feedback* inmediato en términos de un único valor. Para este fin, en [30] se propone el uso de la media geométrica (*g-mean*)

$$g = \sqrt{acc^+ \cdot acc^-} \tag{2.5.2}$$

donde $acc^+ = \frac{TP}{TP+FN}$ y $acc^- = \frac{TN}{TN+FP}$ son respectivamente los porcentajes de ejemplos positivos y negativos bien clasificados. Esta medida representa un único punto de la curva ROC y tiene la propiedad distintiva de ser independiente de la distribución de los ejemplos entre las clases, siendo de este modo robusta en circunstancias donde la distribución puede cambiar con el tiempo o puede ser diferente en los conjuntos de entrenamiento y test, razón por la cual será la medida de referencia empleada en capítulos posteriores cuando se comparen los diferentes métodos de aprendizaje estudiados.

Capítulo 3

Perceptrones Paralelos

A pesar de sus éxitos tempranos, el Perceptrón, junto con su algoritmo de aprendizaje, la regla delta, fueron abandonados debido a su limitado poder expresivo. En su lugar, se recurrió a los Perceptrones Multicapa (PMC) [5, 20]: circuitos consistentes en al menos dos capas consecutivas de perceptrones, que son modificados con una función de activación suave (derivable), y que son entrenados mediante un algoritmo basado en descenso por gradiente de la función de error; calculado a su vez a través de la retropropagación (del inglés *Backpropagation*, BP) del error por las diversas capas. Sin embargo, la realización de redes PMC en hardware plantea serios problemas: requiere separar circuitos bidireccionales, necesita gran precisión al derivar la función de activación e involucra un cálculo costoso de multiplicaciones en cascada durante la fase BP. Adicionalmente, pero por razones análogas, es algo dudoso si en los sistemas neuronales biológicos se aplica realmente un proceso de retropropagación para adaptar sus comportamientos entrada/salida.

En este tercer capítulo se muestra entonces un modelo computacional alternativo: el **Perceptrón Paralelo** (PP) [1, 2]. Compuesto por una única capa de perceptrones en paralelo, cada uno con una única salida binaria, el PP se puede ver como un grupo de votantes, en el que cada voto (con valor -1 ó 1) tiene asociado el mismo peso, y en el que el voto mayoritario se considera la salida binaria del sistema. Con este esquema es posible además aplicar una función de *squashing* sobre el porcentaje de votos con valor 1, y obtener un aproximador universal de funciones continuas con rango $[-1, 1]$. A pesar de ello, esta última opción, orientada a problemas de regresión, no va a considerarse en este trabajo, que se centrará en la aplicación del PP a tareas de clasificación binaria.

En este cometido dos son las principales ventajas del PP: requiere menos de dos bits de comunicación global, y su algoritmo de aprendizaje consiste en una extensión sencilla de la regla delta para perceptrones simples. Denominada por los autores **regla p-delta**, esta nueva estrategia, en contraste con el algoritmo BP donde se transmiten valores con

alta precisión entre las unidades para actualizar los pesos de puertas sigmoidales, requerirá únicamente la difusión de una de tres posibles señales a los diversos perceptrones simples: clasificación correcta o incorrecta, y clasificación dentro de margen. Se proporcionan entonces nuevas ideas para el diseño de dispositivos adaptativos e hipótesis respecto a la organización del aprendizaje en redes neuronales biológicas, que vencen deficiencias de aproximaciones anteriores basadas en retropropagación.

En las próximas secciones se describirá la arquitectura del Perceptrón Paralelo y se explicará la regla de aprendizaje p-delta aplicada a problemas de clasificación. Se darán además la función de error a partir de la cual se deriva la regla y algunos aspectos prácticos para la implementación del modelo.

3.1 Arquitectura

Un perceptrón (también referenciado como neurona de McCulloch-Pitts) con D entradas computa la siguiente función f de $\mathbb{R}^D$ en $\{-1, 1\}$:

$$f(\mathbf{x}) = \begin{cases} 1, & \text{si } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ -1, & \text{en otro caso,} \end{cases}$$

donde $\mathbf{w} \in \mathbb{R}^D$ es el vector de pesos del perceptrón, y $\mathbf{w} \cdot \mathbf{x}$ denota el producto escalar usual $\mathbf{w} \cdot \mathbf{x} := \sum_{d=1}^{D} w^{(d)} x^{(d)}$ para $\mathbf{w} = \langle w^{(1)}, \ldots, w^{(D)} \rangle$ y $\mathbf{x} = \langle x^{(1)}, \ldots, x^{(D)} \rangle$ [†].

Un perceptrón paralelo consiste en una única capa de un número finito H de perceptrones. Sean $f_1, \ldots, f_H$ las funciones de $\mathbb{R}^D$ en $\{-1, 1\}$ que son computadas por estos perceptrones. Para una entrada dada $\mathbf{x}$, el perceptrón paralelo calcula el valor de $\sum_{h=1}^{H} f_h(\mathbf{x}) \in \{-H, \ldots, H\}$ y genera como salida $s(\sum_{h=1}^{H} f_h(\mathbf{x}))$, donde $s : \mathbb{Z} \to \mathbb{R}$ es una función de *squashing* que escala la salida a un rango deseado.

Auer et al. en [1] centran su atención en la función de squashing lineal definida a trozos s_ρ, representada en la figura 3.1 y cuya expresión analítica es la siguiente:

$$s_\rho(z) = \begin{cases} -1, & \text{si } z < -\rho \\ z/\rho, & \text{si } -\rho \leq z \leq \rho \\ +1, & \text{si } z > \rho. \end{cases}$$

[†] Por simplicidad en la notación, se asume que el umbral de cada perceptrón está normalizado a 0. Esto queda justificado si a la vez se asume que la entrada actual $\langle x^{(1)}, \ldots, x^{(D-1)} \rangle \in \mathbb{R}^{D-1}$ se extiende a una entrada D–dimensional $\langle x^{(1)}, \ldots, x^{(D)} \rangle \in \mathbb{R}^D$ con $x^{(D)} := -1$. En este caso la D–ésima componente $w^{(D)}$ del vector de pesos $\mathbf{w}$ toma el papel de umbral, pues $\sum_{d=1}^{D} w^{(d)} x^{(d)} \geq 0 \Leftrightarrow \sum_{d=1}^{D-1} w^{(d)} x^{(d)} \geq w^{(D)}$.

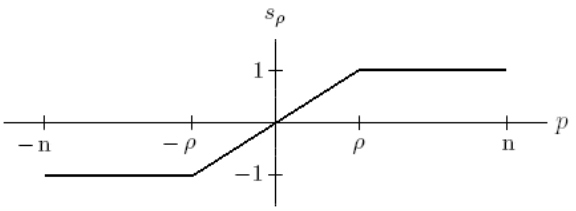

Figura 3.1.1: Función de *squashing* s_ρ para la salida de un perceptrón paralelo.

Para $\rho = 1$ esta función transforma la entrada en el rango binario $\{-1, 1\}$ y simplemente implementa el voto mayoritario de los H perceptrones (asumiendo que H es impar). De hecho, no es difícil probar que toda función booleana de $\{-1, 1\}^D$ en $\{-1, 1\}$ puede ser computada por un perceptrón paralelo (con $\rho = 1$). Lo que es más, los autores han demostrado que toda función continua $g : \mathbb{R}^D \to [-1, 1]$ puede ser aproximada por un perceptrón paralelo dentro de un error acotado ε sobre cualquier subconjunto de $\mathbb{R}^D$ cerrado y acotado, y han mostrado empíricamente que diversos problemas de clasificación reales pueden ser resueltos por perceptrones paralelos consistentes en un número muy pequeño de perceptrones. Como se mencionó anteriormente, en este texto sólo se tratará el PP para clasificación binaria y por ello, a partir de ahora se asume que $\rho = 1$.

3.2 La regla de aprendizaje p-delta

La regla de aprendizaje para perceptrones paralelos, denominada regla p–delta, consiste en dos ingredientes. El primero de ellos es la regla delta clásica, que es aplicada sobre los perceptrones individuales del PP que clasifican incorrectamente un ejemplo dado. El segundo es un procedimiento que decide si la regla delta clásica debe ser aplicada sobre alguno de los perceptrones restantes y está inspirado en las Máquinas de Vectores Soporte. En estas últimas se crean modelos que maximizan el *margen* de clasificación: una entrada $\mathbf{x} \in \mathbb{R}^D$ se clasifica por el $sign(\mathbf{w} \cdot \mathbf{x})$, y $\mathbf{w}$ se calcula de tal manera que para todos los ejemplos de entrenamiento $\mathbf{x}_n$ el producto escalar $|\mathbf{w} \cdot \mathbf{x}_n|$ sea grande. Con este principio, la regla p–delta se aplicará también sobre aquellos perceptrones individuales que dan una salida correcta, pero con un margen demasiado pequeño. Esto asegurará que las salidas de los perceptrones individuales lleguen a ser robustas contra perturbaciones de las entradas, mejorando en el proceso de aprendizaje tanto la estabilidad como la eficacia (en términos de generalización) del PP entrenado.

Obteniendo las salidas correctas

La regla p–delta puede ser aplicada en modo *batch* o de manera incremental. En modo *batch* los vectores de pesos son actualizados una única vez en cada época, después de haber sido presentado todos los ejemplos al clasificador. En el modo incremental los vectores de pesos son actualizados después de cada presentación de un ejemplo de entrenamiento. Por simplicidad en la notación se asumirá que la actualización se realiza en modo incremental. La modificación para modo *batch* es directa.

Sea $(\mathbf{x}, y_{\mathbf{x}}) \in \mathbb{R}^D \times \{-1, 1\}$ el ejemplo de entrenamiento actual y sean $\mathbf{w}_1, \ldots, \mathbf{w}_H \in \mathbb{R}^D$ los vectores de pesos de los H perceptrones individuales que forman un PP. La salida actual del PP se calcula como

$$\hat{y}_{\mathbf{x}} = \begin{cases} +1, & \text{si } \#\{h : \mathbf{w}_h \cdot \mathbf{x} \geq 0\} \geq \#\{h : \mathbf{w}_h \cdot \mathbf{x} < 0\} \\ -1, & \text{si } \#\{h : \mathbf{w}_h \cdot \mathbf{x} \geq 0\} < \#\{h : \mathbf{w}_h \cdot \mathbf{x} < 0\} \end{cases}$$

Si $\hat{y}_{\mathbf{x}} = y_{\mathbf{x}}$ entonces la salida del PP es correcta y sus pesos no necesitan ser modificados. Si por el contrario, $\hat{y}_{\mathbf{x}} = +1$ e $y_{\mathbf{x}} = -1$ (se procede análogamente para el caso $\hat{y}_{\mathbf{x}} = -1$ e $y_{\mathbf{x}} = +1$), entonces se hace necesario reducir el número de vectores de pesos $\mathbf{w}_h$ tales que $\mathbf{w}_h \cdot \mathbf{x} \geq 0$. Aplicando la regla delta clásica para tales vectores se establece la actualización

$$\mathbf{w}_h \leftarrow \mathbf{w}_h - \eta \mathbf{x}$$

donde $\eta > 0$ es la tasa de aprendizaje del PP.

Sin embargo, no es obvio cuales de los vectores con $\mathbf{w}_h \cdot \mathbf{x} \geq 0$ deberían ser modificados con esta regla. Hay varias opciones plausibles (entre las cuales las dos primeras requieren sustancialmente más sobrecarga computacional y comunicación entre los agentes locales):

1. Actualizar sólo uno de los vectores de pesos con $\mathbf{w}_h \cdot \mathbf{x} \geq 0$. Por ejemplo, el que tiene el mínimo valor de $\mathbf{w}_h \cdot \mathbf{x}$.

2. Actualizar N de los vectores de pesos con $\mathbf{w}_h \cdot \mathbf{x} \geq 0$, donde N es el número mínimo de cambios de signo en los perceptrones individuales que son necesarios para obtener la salida $\hat{y}_{\mathbf{x}}$ correcta.

3. Actualizar todos los vectores de pesos con $\mathbf{w}_h \cdot \mathbf{x} \geq 0$.

El problema de las dos primeras opciones es que éstas podrían actualizar vectores de pesos de manera inadecuada para la construcción del clasificador. Actualizando todos los vectores de pesos se asegura que al menos se realice cierto progreso en el proceso de aprendizaje, y aunque podrían modificarse demasiados vectores de pesos, este efecto negativo puede ser contrarestado mediante la aproximación de "estabilización mediante *margen*" explicada más adelante.

Nótese además que la tercera opción es la que requiere la menor comunicación entre los agentes que controlan los pesos individuales $\mathbf{w}_h$: cada perceptrón puede determinar por si mismo si $\mathbf{w}_h \cdot \mathbf{x} \geq 0$, y así, una vez que se conoce la clase a la que pertenece el error global de la salida $\hat{y}_\mathbf{x}$ del PP, no se necesita comunicación adicional para determinar qué agentes tienen que actualizar sus vectores de pesos.

De este modo, eligiendo la tercera opción y procediendo de forma análoga para el caso en el que $\hat{y}_\mathbf{x} = -1$ e $y_\mathbf{x} = +1$, se obtiene la regla de actualización

$$\mathbf{w}_h \leftarrow \mathbf{w}_h + \eta \begin{cases} -\mathbf{x}, & \text{si } \hat{y}_\mathbf{x} = +1 \text{ e } y_\mathbf{x} = -1 \text{ y } \mathbf{w}_h \cdot \mathbf{x} \geq 0 \\ +\mathbf{x}, & \text{si } \hat{y}_\mathbf{x} = -1 \text{ e } y_\mathbf{x} = +1 \text{ y } \mathbf{w}_h \cdot \mathbf{x} < 0 \\ 0, & \text{en otro caso.} \end{cases}$$

que puede comprimirse en esta expresión equivalente

$$\mathbf{w}_h \leftarrow \mathbf{w}_h + \eta \begin{cases} y_\mathbf{x} \cdot \mathbf{x}, & \text{si } \hat{y}_\mathbf{x} \neq y_\mathbf{x} \text{ e } y_\mathbf{x} \cdot \mathbf{w}_h \cdot \mathbf{x} < 0 \\ 0, & \text{en otro caso.} \end{cases} \tag{3.2.1}$$

Estabilizando las salidas

Para cualquiera de las tres opciones citadas anteriormente los vectores de pesos son actualizados sólo si la salida del PP es incorrecta. Por esta razón, los vectores de pesos permanecerían sin modificarse tan pronto como $\mathbf{w}_h \cdot \mathbf{x}$ fuese "correcto" con respecto a la salida objetivo $y_\mathbf{x}$ y la salida del PP $\hat{y}_\mathbf{x}$. De este modo, al final del entrenamiento habría a menudo unos cuantos vectores para los cuales $\mathbf{w}_h \cdot \mathbf{x}$ estaría cercano a 0 (para algún ejemplo de entrenamiento $\mathbf{x}$). Así, una pequeña perturbación en la entrada $\mathbf{x}$ podría cambiar el signo de $\mathbf{w}_h \cdot \mathbf{x}$ y la salida del perceptrón sería algo inestable, reduciendo las capacidades de generalización del PP.

Para estabilizar la salida del PP, se amplia entonces la regla de actualización 3.2.1 de modo que se mantengan los productos escalares $\mathbf{w}_h \cdot \mathbf{x}$ lejos de 0. De hecho, se intenta mantener un *margen* γ en el valor de estos productos, que, siendo un parámetro del algoritmo de aprendizaje, será establecido de manera automática (*véase sección 3.4*) a lo largo del entrenamiento. En el caso en el que $\mathbf{w}_h \cdot \mathbf{x} \geq 0$ tenga el signo correcto, pero que $\mathbf{w}_h \cdot \mathbf{x} < \gamma$, el producto $\mathbf{w}_h \cdot \mathbf{x}$ se incrementa modificando el vector de pesos $\mathbf{w}_h$ como sigue

$$\mathbf{w}_h \leftarrow \mathbf{w}_h + \eta\mu\mathbf{x}$$

para un parámetro apropiado $\mu > 0$ que medirá la importancia que se le da al *margen* γ: si $\mu \approx 0$ entonces la actualización anterior tiene poco influencia, y si μ es mayor se establece con más fuerza el *margen*.

Procediendo de forma análoga con el caso en el que $\mathbf{w}_h \cdot \mathbf{x} < 0$ tenga el signo correcto, pero que $\mathbf{w}_h \cdot \mathbf{x} > -\gamma$, se obtiene una segunda actualización de los vectores de pesos

$$\mathbf{w}_h \leftarrow \mathbf{w}_h + \eta\mu \begin{cases} +\mathbf{x}, & \text{si } \hat{y}_\mathbf{x} = y_\mathbf{x} \text{ y } 0 \leq \mathbf{w}_h \cdot \mathbf{x} < \gamma \\ -\mathbf{x}, & \text{si } \hat{y}_\mathbf{x} = y_\mathbf{x} \text{ y } -\gamma < \mathbf{w}_h \cdot \mathbf{x} < 0 \end{cases} \tag{3.2.2}$$

En este contexto obsérvese que el *margen* γ es efectivo sólo si los vectores de pesos $\mathbf{w}_h$ permanecen acotados: para satisfacer la condición de que $|\mathbf{w}_h \cdot \mathbf{x}| \geq \gamma$, se puede escalar $\mathbf{w}_h$ por un factor $C > 1$ o reducir γ por un factor equivalente $1/C$. De este modo, los pesos sufrirán una tercera actualización

$$\mathbf{w}_h \leftarrow \mathbf{w}_h - \eta \left(||\mathbf{w}_h||^2 - 1 \right) \mathbf{w}_h \tag{3.2.3}$$

que hace tender la norma $||\mathbf{w}_h||$ a 1.

Concluyendo, y reuniendo las tres actualizaciones 3.2.1, 3.2.2 y 3.2.3, la regla p-delta se muestra en la figura 3.2.1:

Para todo $h = 1, \ldots, H$:

$$\mathbf{w}_h \leftarrow \mathbf{w}_h - \eta \left(||\mathbf{w}_h||^2 - 1 \right) \mathbf{w}_h + \eta \begin{cases} y_\mathbf{x} \cdot \mathbf{x}, & \text{si } \hat{y}_\mathbf{x} \neq y_\mathbf{x} \text{ e } y_\mathbf{x} \cdot \mathbf{w}_h \cdot \mathbf{x} < 0 \\ \mu \cdot \mathbf{x}, & \text{si } \hat{y}_\mathbf{x} = y_\mathbf{x} \text{ y } 0 \leq \mathbf{w}_h \cdot \mathbf{x} < \gamma \\ -\mu \cdot \mathbf{x}, & \text{si } \hat{y}_\mathbf{x} = y_\mathbf{x} \text{ y } -\gamma < \mathbf{w}_h \cdot \mathbf{x} < 0 \\ 0, & \text{en otro caso} \end{cases}$$

Figura 3.2.1: Regla p-delta para aprendizaje incremental del PP.

3.3 Función de error de la regla p-delta

La regla p–delta descrita puede considerarse como un tipo de descenso por gradiente de la función criterio

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2} \sum_{h=1}^{H} \left(||\mathbf{w}_h||^2 - 1 \right)^2 \\ &\quad - \sum_{\mathbf{x}: y_\mathbf{x} \hat{y}_\mathbf{x} = -1} \left(\sum_{h: y_\mathbf{x} \mathbf{w}_h \mathbf{x} < 0} y_\mathbf{x} \mathbf{w}_h \mathbf{x} \right) + \mu \sum_{\mathbf{x}: y_\mathbf{x} \hat{y}_\mathbf{x} = 1} \left(\sum_{h: 0 < y_\mathbf{x} \mathbf{w}_h \mathbf{x} < \gamma} (\gamma - y_\mathbf{x} \mathbf{w}_h \mathbf{x}) \right) \\ &= J_1(\mathbf{w}) + J_2(\mathbf{w}) + \mu J_3(\mathbf{w}) \end{aligned}$$

con $\mathbf{w} = (\mathbf{w}_1, \ldots, \mathbf{w}_H)$. En ella, J_1 penaliza la desviación de $||\mathbf{w}_h||$ respecto a 1 y por lo tanto hace que los vectores de pesos tiendan a tener norma 1, J_2 es básicamente una medida

del error de clasificación; es 0 en los ejemplos bien clasificados, e inversamente proporcional a las activaciones normalizadas en los ejemplos mal clasificados (i.e., a la distancia entre las proyecciones de los patrones y los hiperplanos de separación), y J_3 puede verse como un término de regularización con peso μ que penaliza la cercanía a 0 de las proyecciones de los ejemplos bien clasificados, pero con margen inferior a γ.

En [1], sin embargo, se da una versión alternativa de la función de error:

$$
\begin{aligned}
& J(\mathbf{w}_1, \ldots, \mathbf{w}_H) \\
= \; & \frac{1}{2} \sum_{h=1}^{H} \left(||\mathbf{w}_h||^2 - 1 \right)^2 \\
& + \begin{cases}
\sum_{h:\mathbf{w}_h\mathbf{x} \geq 0} (\mu\gamma + \mathbf{w}_h\mathbf{x}) + \sum_{h:-\gamma < \mathbf{w}_h\mathbf{x} < 0} (\mu\gamma + \mu\mathbf{w}_h\mathbf{x}), & \text{si } \hat{y}_\mathbf{x} = +1, y_\mathbf{x} = -1 \\
\sum_{h:\mathbf{w}_h\mathbf{x} < 0} (\mu\gamma - \mathbf{w}_h\mathbf{x}) + \sum_{h:0 \leq \mathbf{w}_h\mathbf{x} < +\gamma} (\mu\gamma - \mu\mathbf{w}_h\mathbf{x}), & \text{si } \hat{y}_\mathbf{x} = -1, y_\mathbf{x} = +1 \\
\sum_{h:-\gamma < \mathbf{w}_h\mathbf{x} < 0} (\mu\gamma + \mu\mathbf{w}_h\mathbf{x}) + \sum_{h:0 \leq \mathbf{w}_h\mathbf{x} < +\gamma} (\mu\gamma - \mu\mathbf{w}_h\mathbf{x}), & \text{si } \hat{y}_\mathbf{x} = y_\mathbf{x}
\end{cases}
\end{aligned}
$$

Al igual que antes, el primer término penaliza la desviación de $||\mathbf{w}_h||$ respecto a 1, haciendo que los vectores de pesos tiendan a estar normalizados. La segunda y tercera líneas cubren los casos en los que la salida del PP es incorrecta. Las sumas $\sum_{h:\mathbf{w}_h\mathbf{x} \geq 0} (\mu\gamma + \mathbf{w}_h\mathbf{x})$ y $\sum_{h:\mathbf{w}_h\mathbf{x} < 0} (\mu\gamma - \mathbf{w}_h\mathbf{x})$ miden cómo de lejos los productos escalares $\mathbf{w}_h\mathbf{x}$ están en el lado incorrecto (desde 0). Por ejemplo, si $\hat{y}_\mathbf{x} = +1, y_\mathbf{x} = -1$, hay demasiados pesos $\mathbf{w}_h$ con $\mathbf{w}_h\mathbf{x} \geq 0$ y se los penaliza con $\mu\gamma + \mathbf{w}_h\mathbf{x}$. El término $\mu\gamma$ asegura compatibilidad con las otras sumas, $\sum_{h:-\gamma < \mathbf{w}_h\mathbf{x} < 0} (\mu\gamma + \mu\mathbf{w}_h\mathbf{x})$ y $\sum_{h:0 \leq \mathbf{w}_h\mathbf{x} < +\gamma} (\mu\gamma - \mu\mathbf{w}_h\mathbf{x})$, que afectan a los pesos con $\mathbf{w}_h\mathbf{x}$ dentro del margen γ alrededor de 0. Considérese por ejemplo el caso $\sum_{h:0 \leq \mathbf{w}_h\mathbf{x} < +\gamma} (\mu\gamma - \mu\mathbf{w}_h\mathbf{x})$. Si $0 \leq \mathbf{w}_h\mathbf{x} < \gamma$, entonces la distancia al margen γ es $\gamma - \mathbf{w}_h\mathbf{x}$, que ponderada por μ da el término de error $\mu\gamma - \mu\mathbf{w}_h\mathbf{x}$. Finalmente, la cuarta y última línea está asociada a aquellos casos en los que la salida del perceptrón es correcta, pero con pesos $\mathbf{w}_h$ tales que $\mathbf{w}_h\mathbf{x}$ está cerca de 0. A estos también se les penaliza con la distancia al margen γ ponderada por el parámetro μ.

Se puede comprobar que esta función de error es continua, no negativa, e igual a 0 si y sólo si $\hat{y}_\mathbf{x} = y_\mathbf{x}$, todos los vectores de pesos $\mathbf{w}_h$ satisfacen que $|\mathbf{w}_h \cdot \mathbf{x}| \geq \gamma$ y $||\mathbf{w}_h|| = 1$. La diferencia con respecto a la primera función criterio radica en el hecho de que esta segunda versión también aplica la regla delta tradicional, ponderada con el parámetro μ, a aquellos perceptrones que predicen la clase de $\mathbf{x}$ correctamente dentro del margen γ, aún cuando el comité PP lo ha hecho incorretamente (expresiones $\sum_{h:-\gamma < \mathbf{w}_h\mathbf{x} < 0} (\mu\gamma + \mu\mathbf{w}_h\mathbf{x})$ y $\sum_{h:0 \leq \mathbf{w}_h\mathbf{x} < +\gamma} (\mu\gamma - \mu\mathbf{w}_h\mathbf{x})$). Con este añadido, la regla p-delta pasaría a tener la expresión de la figura 3.3.1.

Para todo $h = 1, \ldots, H$:

$$\mathbf{w}_h \leftarrow \mathbf{w}_h - \eta \left(||\mathbf{w}_h||^2 - 1 \right) \mathbf{w}_h + \eta \begin{cases} y_{\mathbf{x}} \cdot \mathbf{x}, & \text{si } \hat{y}_{\mathbf{x}} \neq y_{\mathbf{x}} \text{ e } y_{\mathbf{x}} \cdot \mathbf{w}_h \cdot \mathbf{x} < 0 \\ \mu \cdot y_{\mathbf{x}} \cdot \mathbf{x}, & \text{si } \hat{y}_{\mathbf{x}} \neq y_{\mathbf{x}} \text{ y } 0 \leq y_{\mathbf{x}} \cdot \mathbf{w}_h \cdot \mathbf{x} < \gamma \\ \mu \cdot \mathbf{x}, & \text{si } \hat{y}_{\mathbf{x}} = y_{\mathbf{x}} \text{ y } 0 \leq \mathbf{w}_h \cdot \mathbf{x} < \gamma \\ -\mu \cdot \mathbf{x}, & \text{si } \hat{y}_{\mathbf{x}} = y_{\mathbf{x}} \text{ y } -\gamma < \mathbf{w}_h \cdot \mathbf{x} < 0 \\ 0, & \text{en otro caso} \end{cases}$$

Figura 3.3.1: Versión alternativa de la regla p-delta para aprendizaje incremental del PP.

Esta alternativa no se ha tratado en la presente investigación por no tener resultados contrastables de ella en la literatura, y su estudio podría ser una línea de trabajo futuro. La implementación realizada ha sido la expuesta en la figura 3.2.1, pero en ella se han considerado ciertas modificaciones y aspectos adicionales, que se detallan a continuación en la siguiente sección.

3.4 Consideraciones prácticas para la implementación

En las secciones anteriores se ha descrito la regla p–delta incremental. Sin embargo, como se hace en [1], los experimentos de este trabajo se obtuvieron con Perceptrones Paralelos entrenados en modo *batch*. Por esta razón, antes de acabar el capítulo, se hace necesario explicar varios aspectos prácticos sobre la implementación desarrollada.

El primero está asociado a la actualización de los pesos $\mathbf{w}_h$. En vez de modificarlos por cada ejemplo de entrenamiento presentado, se acumulan las actualizaciones de todos los ejemplos de entrenamiento, y sólo se modifican los vectores al final de la época. Con ello la regla de actualización se modifica: ya no se cambian los pesos de manera implícita con la expresión

$$\mathbf{w}_h \leftarrow \mathbf{w}_h - \eta \left(||\mathbf{w}_h||^2 - 1 \right) \mathbf{w}_h$$

sino que son normalizados explícitamente según

$$\mathbf{w}_h \leftarrow \mathbf{w}_h / ||\mathbf{w}_h|| \tag{3.4.1}$$

tras cada época del aprendizaje.

El segundo se corresponde con la elección de los parámetros η y γ. El establecimiento del valor de γ es importante y para él se ha empleado el mecanismo de ajuste automático propuesto en [1]: aumentar γ si para un ejemplo de entrenamiento $(\mathbf{x}, y_{\mathbf{x}})$ sólo unos pocos productos escalares $\mathbf{w}_h \cdot \mathbf{x}$ están dentro del margen $(-\gamma, \gamma)$, y disminuirlo si hay muchos

productos dentro de él. De este modo, si el número de productos en margen (con signo correcto respecto a $y_{\mathbf{x}}$) se calcula con las expresiones

$$\begin{aligned} M_+ &= \#\{h : 0 \leq \mathbf{w}_h \cdot \mathbf{x} < +\gamma \text{ e } \hat{y}_{\mathbf{x}} = y_{\mathbf{x}}\} \\ M_- &= \#\{h : -\gamma \leq \mathbf{w}_h \cdot \mathbf{x} < 0 \text{ e } \hat{y}_{\mathbf{x}} = y_{\mathbf{x}}\} \end{aligned}$$

el parámetro γ se actualizará según la regla

$$\begin{aligned} \gamma &\leftarrow \gamma + 0.25\eta, \quad \text{si } M_+ + M_- = 0 \\ \gamma &\leftarrow \gamma - 0.75\eta, \quad \text{si } M_+ + M_- \neq 0 \end{aligned} \tag{3.4.2}$$

En modo *batch*, como ocurre con los vectores de pesos, esta actualización se va acumulando para todos los patrones de entrenamiento, pero sólo se lleva a cabo al finalizar la época de entrenamiento actual.

A partir de ella, la convergencia hacia un valor estable de γ se consigue haciendo que la tasa de aprendizaje η tienda progresivamente a 0. Para ello, en [1, 2] se presentan dos procedimientos. Uno considera que si tras una actualización de los pesos, el error de entrenamiento ha disminuido, el valor del parámetro η ha de incrementarse, y si por el contrario el error de entrenamiento ha aumentado, el valor de η ha de disminuirse; en las referencias, multiplicando η por factores 1.1 y 0.5 respectivamente; mientras que el otro propone la actualización

$$\eta_t \leftarrow \eta_0/\sqrt{t}, \tag{3.4.3}$$

en la que η_0 es el valor inicial de η y t es la época de entrenamiento actual. Esta opción evita tener que definir los factores anteriores y será la que se emplee en las pruebas experimentales.

Capítulo 4

AdaBoost

El problema de seleccionar una teoría entre un conjunto de teorías plausibles ha sido centro de discusión filosófica a lo largo de la historia. Mientras que el filósofo medieval William de Occam defendió en su bien conocido principio de la "navaja de Occam" (*Occam's razor*) que, en igualdad de condiciones, se debería elegir la hipótesis más simple; fue el pensador griego Epicuro quien propuso una tesis prácticamente inversa: "si más de una hipótesis es consistente con los datos, debemos mantenerlas todas". En otras palabras, se deben tener en cuenta todas las hipótesis plausibles, ya que cualquiera de ellas podría extraer conclusiones válidas para resolver el problema en cuestión. Esta idea es bien conocida en los sistemas de decisión, donde es preferible consultar a varios asesores en vez de a uno sólo, y es la base de los llamados **Métodos de Agregación o Conjuntos de Modelos** (del inglés *Ensemble Methods*), que se van a tratar en este capítulo.

Intuitivamente, estos métodos utilizan un conjunto de modelos diferentes para combinarlos y crear un nuevo modelo que generalmente tendrá una precisión mejor a la obtenida por cada componente individual. En este cometido, dos son los factores que caracterizan a los distintos mecanismos de agregación. En primer lugar, la manera en la que se construyen los modelos o hipótesis del conjunto. La calidad del conjunto dependerá en alto grado de la precisión y diversidad de los componentes del mismo, ya que si los modelos son similares, los errores cometidos por ellos estarán muy correlacionados y por tanto su combinación no estaría motivada. En segundo lugar, la técnica empleada para la combinación de las predicciones de los modelos, que en general consistirá en algún mecanismo de votación sobre las hipótesis obtenidas.

En las próximas secciones se introducirán las principales alternativas en ambos factores, y se pondrá especial énfasis en unas de las más estudiadas: las técnicas de **Boosting**. De ellas, a su vez, se elegirá y explicará en detalle el bien conocido algoritmo **AdaBoost** de Freund y Schapire, debido a los buenos resultados y fundamentos teóricos que posee.

4.1 Conjuntos de modelos

Teniendo como objetivo la mejora de la precisión de las predicciones, ha surgido un interés creciente en la definición de métodos que combinan hipótesis. Estos métodos construyen un conjunto (*ensemble*) de hipótesis h_t, y combinan las predicciones de estas últimas de alguna manera (normalmente por votación) para clasificar o hacer regresión sobre ejemplos. A estas técnicas de combinación de hipótesis se les conoce como métodos multiclasificadores, de modelos combinados o de conjunto de modelos (*Ensemble Methods*). En la figura 4.1.1 se puede apreciar el esquema general que se sigue en la combinación de modelos obtenidos con un algoritmo de aprendizaje dado. Existen otras técnicas híbridas consistentes en la combinación de modelos generados por diferentes algoritmos de aprendizaje (*stacking* y *cascading*), pero no van a ser tratadas en este texto.

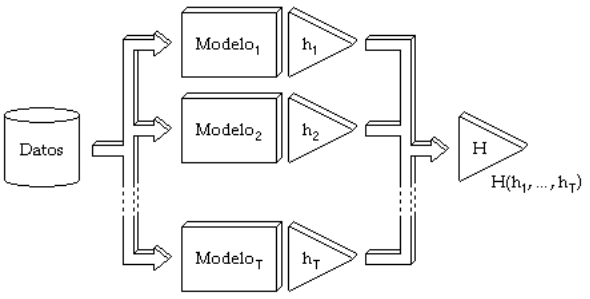

Figura 4.1.1: Esquema general de combinación no híbrida de modelos.

La calidad del modelo combinado depende en gran medida de la precisión y diversidad de los componentes del conjunto. Considérese el siguiente ejemplo [15]: dado un conjunto formado por tres clasificadores $\{h_1, h_2, h_3\}$, sea $\mathbf{x}$ un nuevo patrón a ser clasificado. Si los tres clasificadores son similares, entonces cuando $h_1(\mathbf{x})$ sea erróneo, probablemente $h_2(\mathbf{x})$ y $h_3(\mathbf{x})$ también lo serán. Sin embargo, si los clasificadores son lo bastante diversos, los errores que cometan estarán poco correlacionados, y por tanto, cuando $h_1(\mathbf{x})$ sea erróneo, $h_2(\mathbf{x})$ y $h_3(\mathbf{x})$ podrían ser correctos, y entonces, si la combinación se realizase por votación mayoritaria, el conjunto combinado clasificaría correctamente el dato $\mathbf{x}$. De manera más precisa, si los porcentaje de error de T hipótesis h_t son todos iguales a $p_e < 0.5$, y los errores ocurren de manera independiente, entonces la probabilidad de que el voto mayoritario sea incorrecto será el área bajo la distribución binomial de probabilidad p_e cuando más de $T/2$ hipótesis son incorrectas. La figura 4.1.2 muestra este hecho para un conjunto de 21 hipótesis, cada una con un porcentaje de error de 0.3. El área bajo la curva para 11 o más hipótesis simultáneamente incorrectas es 0.026, lo cual es mucho más pequeño que el porcentaje de error de una hipótesis individual. Por supuesto, si las hipótesis in-

dividuales producen errores no correlacionados mayores a 0.5, entonces el porcentaje de error del modelo combinado crecerá como resultado de la votación. Así, un aspecto clave para construir conjuntos de modelos exitosos es combinar clasificadores individuales con porcentajes de error por debajo de 0.5 y que de algún modo sean no correlacionados.

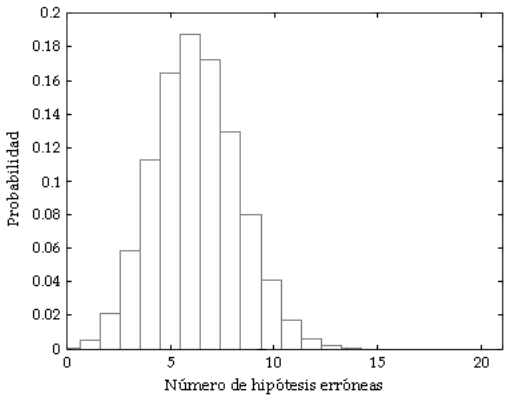

Figura 4.1.2: Probabilidad de que exactamente t de 21 hipótesis no correlacionadas con porcentaje de error $p_e = 0.3$ sean erróneas en un conjunto de clasificadores con votación mayoritaria.

Esta caracterización del problema es fascinante, pero no aborda la pregunta de si es posible en la práctica construir buenos modelos combinados. Afortunadamente, sí es a menudo posible. Hay tres razones fundamentales para ello [15].

- La primera razón es *estadística*. Un algoritmo de aprendizaje puede considerarse como una búsqueda en un espacio $\mathcal{H}$ de hipótesis. El problema estadístico surge cuando la cantidad de datos de entrenamiento disponible es demasiado pequeña comparada con el tamaño de $\mathcal{H}$. Sin los suficientes datos, el algoritmo de aprendizaje puede encontrar muchas hipótesis diferentes en $\mathcal{H}$ tales que todas dan el mismo porcentaje de aciertos sobre los datos de entrenamiento. Construyendo un modelo que combine todos los clasificadores acertados, el algoritmo puede "promediar" sus votos y reducir el riesgo de elegir clasificadores incorrectos. La figura 4.1.3 (imagen de la izquierda) representa esta situación. La curva exterior denota el espacio de hipótesis $\mathcal{H}$, la curva interior denota el conjunto de hipótesis que dan buenos porcentajes de acierto sobre el conjunto de entrenamiento, y el punto etiquetado como f es la hipótesis objetivo. Se puede ver que promediando las hipótesis anteriores es posible encontrar una buena aproximación de f.

- La segunda razón es *computacional*. Muchos algoritmos de aprendizaje trabajan desarrollando alguna forma de búsqueda local que podría quedarse atrapada en un mínimo local. Por ejemplo, las redes neuronales hacen uso de descenso por gradi-

ente para minimizar la función de error sobre el conjunto de entrenamiento, y los árboles de decisión emplean una regla de separación avariciosa para construir los clasificadores. En casos donde hay suficientes datos de entrenamiento (por lo que el problema estadístico está ausente), para el algoritmo de aprendizaje podría ser todavía muy difícil computacionalmente encontrar la mejor hipótesis. Un conjunto de clasificadores obtenidos mediante búsquedas locales en diferentes puntos iniciales podría proporcionar, como muestra la figura 4.1.3 (imagen central), una mejor aproximación a la función f deseada.

- La tercera razón es *figurativa*. En muchas aplicaciones de ML, la función objetivo f no puede ser representada por ninguna de las hipótesis de $\mathcal{H}$. Sin embargo, mediante sumas ponderadas de las hipótesis extraídas podría ser posible expandir el espacio de funciones representables. De nuevo, la figura 4.1.3 (imagen de la derecha) representa esta situación. Este aspecto es de alguna manera sutil, porque hay muchos algoritmos de aprendizaje para los cuales, siempre que se tengan suficientes datos de entrenamiento, $\mathcal{H}$ es, en principio, el espacio de todos los posibles clasificadores. No obstante, con una muestra finita de entrenamiento, estos algoritmo sólo pueden explorar un conjunto finito de hipótesis y paran la búsqueda cuando encuentran una hipótesis que se ajusta a los datos de entrenamiento. Así, en la figura 4.1.3, se debe considerar $\mathcal{H}$ como el espacio efectivo de hipótesis buscadas por el algoritmo de aprendizaje para un conjunto de entrenamiento dado.

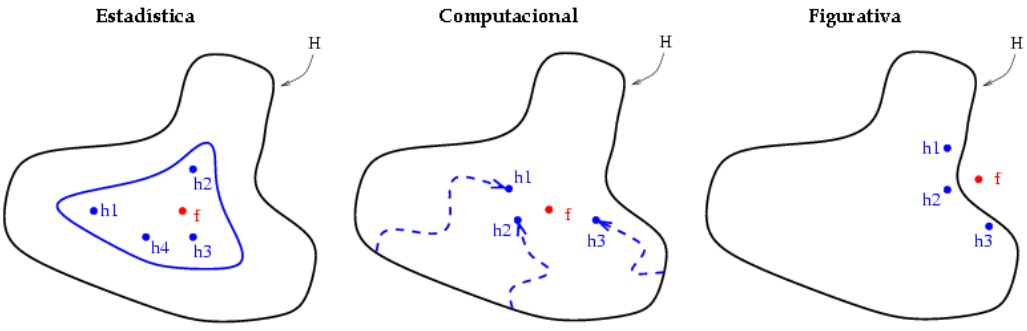

Figura 4.1.3: Tres razones fundamentales de porqué un conjunto de modelos puede dar mejores resultados que un modelo único.

Muchos métodos han sido definidos para la construcción y combinación de modelos utilizando un único algoritmo de aprendizaje. Dietterich, en [15], establece la siguiente taxonomía para las técnicas de construcción de los modelos individuales de un multiclasificador, en busca de la diversidad en las hipótesis generadas:

- **Manipulación de las muestras de entrenamiento**. El conjunto de modelos se genera mediante la ejecución repetidas veces del mismo algoritmo de aprendizaje. En cada iteración se parte de un conjunto diferente de datos de entrenamiento, y de esta manera se genera un modelo distinto. La clave en este tipo de métodos es la definición de un mecanismo adecuado para la selección de los subconjuntos a partir del conjunto de entrenamiento, y el empleo de algoritmos de aprendizaje que sean "inestables", i.e., algoritmos cuyas clasificaciones experimentan cambios importantes como resultado de pequeñas perturbaciones en las muestras de entrenamiento. Los árboles de decisión y las redes neuronales son inestables, mientras que algoritmos como la regresión lineal o la clasificación mediante vecinos próximos resultan en general muy estables. Ejemplos de métodos multiclasificadores de esta familia son *Bagging* [7, 36] y *Boosting* [23, 24, 36].

- **Manipulación de los atributos de entrada**. Los modelos son entrenados alterando el conjunto de atributos empleado para los ejemplos de entrada. La técnica *Forest* [26] pertenece a esta familia y consiste en la combinación de árboles de decisión obtenidos mediante el algoritmo C4.5 y empleando subespacios de atributos aleatorios.

- **Manipulación de las etiquetas de salida**. La construcción de cada modelo se lleva a cabo con ejemplos en los que se cambia la etiqueta de la clase a la que pertenecen. Por ejemplo, la técnica ECOC (*Error-Correcting Output Coding* [14] construye nuevos problemas de aprendizaje particionando las K clases del problema original en dos subconjuntos A_t y B_t. Los ejemplos de entrada se etiquetan como 0 si sus clases originales están en A_t, y como 1 si pertenecen a B_t. Con este nuevo etiquetado se construye el clasificador h_t. Repitiendo este proceso T veces (generando diferentes subconjuntos A_t y B_t), se obtiene un conjunto de T clasificadores $h_1, \ldots, h_T$. A continuación, dado un ejemplo $\mathbf{x}$, si $h_t(\mathbf{x}) = 0$, cada clase de A_t recibe un voto, y si $h_t(\mathbf{x}) = 1$, son las de B_t las que lo obtienen. Cuando los T clasificadores han votado, la clase con el mayor número de votos será la predicción del conjunto.

- **Manipulaciones aleatorias del proceso de aprendizaje**. Los modelos son generados introduciendo componentes aleatorias en los procesos de aprendizaje. Por ejemplo, entrenando redes neuronales con diferentes pesos iniciales.

De estas opciones, la próxima sección va a abordar con más detalle la que involucra la creación de modelos alterando los conjuntos de entrenamiento. Concretamente, se describirán los algoritmos de *Bagging* y *Boosting*, para poder posteriormente, en el apartado 4.3, introducir el algoritmo *AdaBoost*, uno de los temas centrales del presente trabajo.

4.2 Conjuntos de modelos mediante manipulación de datos de entrenamiento

Para crear conjuntos de clasificadores mediante el uso de distintas muestras de entrenamiento es fundamental que el algoritmo de aprendizaje sea inestable, produciendo hipótesis considerablemente diferentes al introducir pequeños cambios en los datos de entrada. Si se quiere definir este concepto de una manera formal es necesario introducir antes el concepto de aprendizaje "Probablemente y Aproximadamente Correcto" (PAC).

Suponiendo que los ejemplos son aleatorios, independientes e idénticamente distribuidos, y que sus etiquetas vienen dadas por una función $y \in \mathcal{Y}$, el objetivo de un algoritmo de aprendizaje es encontrar una hipótesis $h \in \mathcal{H}$ que se aproxime a y. En otras palabras, su objetivo es minimizar el error de generalización $err_D(h) = Pr_{\mathcal{X}\sim D}[h(\mathbf{x}) \neq y_{\mathbf{x}}]$, para todo ejemplo $\mathbf{x} \in \mathcal{X}$, $y_{\mathbf{x}} = y(\mathbf{x})$ y distribución de probabilidad D sobre $\mathcal{X}$. En este contexto considérense las siguientes definiciones:

Definición 4.2.1. *Una clase de funciones $\mathcal{Y}$ es **aprendible PAC fuertemente** si existe un algoritmo A tal que $\forall y \in \mathcal{Y}$, $\forall \epsilon > 0$, $\forall \delta > 0, \forall D$, y un conjunto de $N = polinomio(\frac{1}{\epsilon}, \frac{1}{\delta})$ ejemplos $\{(\boldsymbol{x}_n, y_n) : \boldsymbol{x}_n \in \mathcal{X}, y_n = y(\boldsymbol{x}_n), \mathcal{X} \sim D\}$, es capaz de computar una hipótesis $h \in \mathcal{H}$ satisfaciendo que*

$$Pr[err_D(h) > \epsilon] \leq \delta$$

Definición 4.2.2. *Una clase de funciones $\mathcal{Y}$ es **aprendible PAC débilmente** si existe un algoritmo A y un $\gamma > 0$ tal que $\forall y \in \mathcal{Y}$, $\forall \delta > 0, \forall D$, y un conjunto de $N = polinomio(\frac{1}{\delta})$ ejemplos $\{(\boldsymbol{x}_n, y_n) : \boldsymbol{x}_n \in \mathcal{X}, y_n = y(\boldsymbol{x}_n), \mathcal{X} \sim D\}$, es capaz de computar una hipótesis $h \in \mathcal{H}$ satisfaciendo que*

$$Pr[err_D(h) > \tfrac{1}{2} - \gamma] \leq \delta$$

Un algoritmo débil puede ser trivial. Por ejemplo, uno que predice la clase de un ejemplo siempre como positiva, y las muestras tienen más del 75% de ejemplos positivos. Sin embargo, éste no es un aprendiz PAC débil, pues no da un error ligeramente menor al 50% para *cualquier* distribución de ejemplos.

Los métodos de agregación buscan combinar clasificadores PAC débiles, que poseen una precisión ligeramente menor a la de una predicción aleatoria (del 50%) para conseguir un aprendiz PAC fuerte, con una tendencia del error de entrenamiento a 0. Aquí se presentan dos de ellos, *Bagging* y *Boosting*, que en busca del propósito anterior emplean diferentes conjuntos de entrenamiento. A partir de ahora, todo algoritmo de aprendizaje se considera PAC débil y se referencia como aprendiz débil (del inglés *Weak Learner*, WL).

Bagging

La técnica de Bagging (*bootstrap aggregation*) consiste en la generación de T hipótesis h_t a partir de subconjuntos de entrenamiento de N ejemplos creados mediante selección aleatoria con reemplazamiento (puede haber ejemplos repetidos) de la muestra de entrenamiento inicial, también de N elementos. La combinación de las diferentes hipótesis h_t en una hipótesis final H se lleva a cabo mediante votación mayoritaria de las primeras.

Dados $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, *con* $\mathbf{x}_n \in \mathcal{X}$, $y_n \in \mathcal{Y} = \{-1, +1\}, n = 1, \ldots, N$
Para t *=1,. . . ,T:*

- *Seleccionar con reemplazamiento N ejemplos.*
- *Entrenar un aprendiz débil usando los ejemplos seleccionados.*
- *Obtener la hipótesis débil* $h_t : \mathcal{X} \to \{-1, +1\}$.

Devolver *como hipótesis final:*

$$H(x) := sign\left(\frac{1}{T}\sum_{t=1}^{T} h_t(x)\right)$$

Figura 4.2.1: Esquema general del algoritmo Bagging.

Boosting

Las estrategias de Boosting asignan en cada iteración t un peso $D_t(\mathbf{x}_n)$ a los ejemplos de entrenamiento. Las hipótesis h_t se crean minimizando la suma de los pesos de los ejemplos mal clasificados. De esta forma se consigue que el modelo aprendido en la siguiente iteración otorgue más relevancia a los ejemplos mal clasificados por los modelos anteriores. El error ε_t se usa para determinar los pesos $D_{t+1}(\mathbf{x}_n)$ y la relevancia de h_t en la hipótesis final H.

Dados $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, *con* $\mathbf{x}_n \in \mathcal{X}$, $y_n \in \mathcal{Y} = \{-1, +1\}, n = 1, \ldots, N$
Para t *=1,. . . ,T:*

- *Determinar la distribución* D_t *sobre los ejemplos.*
- *Entrenar un aprendiz débil usando distribución* D_t.
- *Obtener la hipótesis débil* $h_t : \mathcal{X} \to \{-1, +1\}$.
- $\varepsilon_t := Pr_{D_t}[h_t(\mathbf{x}_n) \neq y_n] = \frac{1}{2} - \gamma_t$.

Devolver *hipótesis final H, combinación de las* h_t *en función de los* ε_t.

Figura 4.2.2: Esquema general de un algoritmo de Boosting.

Para incorporar la distribución D_t en el aprendizaje existen en la práctica dos aproximaciones: (a) el algoritmo selecciona un conjunto de ejemplos mediante selección con reemplazamiento de acuerdo a D_t (*boosting por remuestreo*), (b) el algoritmo intenta minimizar directamente el error de entrenamiento ponderado $\sum_{n:h_t(\mathbf{x}_n)\neq y_n} D_t(\mathbf{x}_n)$ (*boosting por reponderación*). La manera en la que se determine esta selección y en la que se combinen las hipótesis h_t caracteriza a las diversas técnicas de Boosting.

4.3 AdaBoost

AdaBoost (*Adaptative Boosting*, AB) emplea un parámetro $\alpha_t = \frac{1}{2}\ln(\frac{1-\varepsilon_t}{\varepsilon_t})$ al combinar las hipótesis h_t y determinar las distribuciones D_t. Por una parte, α_t, definido en función del error ε_t, mide la relevancia de h_t en el clasificador final $H(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$. Nótese que $\alpha_t \geq 0$ si $\varepsilon_t \leq 0.5$, y que α_t es mayor cuanto menor es ε_t, dando más importancia a aquellas hipótesis que comenten menos errores. Por otra parte, α_t sirve para establecer el peso $D_{t+1}(\mathbf{x}_n)$ de cada ejemplo. AB escala $D_t(\mathbf{x}_n)$ mediante la expresión $\exp(-\alpha_t y_n h_t(\mathbf{x}_n))$, aumentando el peso de los ejemplos mal clasificados ($y_n \neq h_t(\mathbf{x}_n)$).

Dados $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, *con* $\mathbf{x}_n \in \mathcal{X}$, $y_n \in \mathcal{Y} = \{-1, +1\}, n = 1, \ldots, N$
Inicializar $D_1(n) := \frac{1}{N}$.
Para t = 1,...,T*:*

- *Entrenar un aprendiz débil usando la distribución* D_t.
- *Obtener la hipótesis débil* $h_t : \mathcal{X} \to \{-1, +1\}$.
- $\varepsilon_t := Pr_{D_t}[h_t(\mathbf{x}_n) \neq y_n] = \frac{1}{2} - \gamma_t$.
- ***Si*** $\varepsilon_t = 0$ *ó* $\varepsilon_t \geq 0.5$*:*

  Salir *del bucle (y considerar* $T = t - 1$*).*
- $\alpha_t := \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$.
- *Actualizar:*

$$D_{t+1}(\mathbf{x}_n) := \frac{D_t(\mathbf{x}_n)\exp\left(-\alpha_t y_n h_t(\mathbf{x}_n)\right)}{Z_t}$$

  donde Z_t *es un factor de normalización tal que* D_{t+1} *sea una distribución de probabilidad.*

Devolver *como hipótesis final:*

$$H(x) := sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Figura 4.3.1: Esquema general del algoritmo AdaBoost.

Las ideas anteriores se muestran en la figura 4.3.1, esquema general del algoritmo AdaBoost para problemas de clasificación binarios. Para profundizar y razonar sobre el mismo, en el resto de la sección se dan resultados y fundamentos teóricos del algoritmo, empezando con una cota del error de entrenamiento.

Sea:

$$f_T(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{4.3.1}$$

tal que $H(x) = sign(f_T(x))$. Definiendo para cualquier predicado π la función $\mathbb{I}(\pi)$, que vale 1 si se cumple π y 0 en caso contrario, se puede probar la siguiente cota del error de entrenamiento para H.

Teorema 4.3.1. *Asumiendo la notación de la figura 4.3.1, en AdaBoost se cumple la siguiente cota del error de entrenamiento de H:*

$$\widehat{err}(H) = \frac{1}{N}\sum_{n=1}^{N} \mathbb{I}(H(\boldsymbol{x}_n) \neq y_n) \leq \prod_{t=1}^{T} Z_t$$

Demostración: Si $H(x) \neq y_n$ entonces $y_n f_T(\mathbf{x}_n) \leq 0$, que es equivalente a decir que $\exp(-y_n f_T(\mathbf{x}_n)) \geq 1$. De este modo,

$$\mathbb{I}(H(\mathbf{x}_n) \neq y_n) \leq \exp(-y_n f_T(\mathbf{x}_n)) \tag{4.3.2}$$

Por otra parte, desarrollando de forma recursiva la regla de actualización de probabilidades y expresándola en función de $f_T(x)$ se tiene:

$$\begin{aligned}
D_{T+1}(\mathbf{x}_n) &= \frac{D_T(\mathbf{x}_n)\exp\left(-\alpha_T y_n h_T(\mathbf{x}_n)\right)}{Z_T} \\
&= \frac{D_{T-1}(\mathbf{x}_n)\exp\left(-\sum_{t=T-1}^{T}\alpha_t y_n h_t(\mathbf{x}_n)\right)}{\prod_{t=T-1}^{T} Z_t} \\
&= \dots \\
&= \frac{D_1(\mathbf{x}_n)\exp\left(-\sum_{t=1}^{T}\alpha_t y_n h_t(\mathbf{x}_n)\right)}{\prod_{t=1}^{T} Z_t} \\
&= \frac{\exp\left(-\sum_{t=1}^{T}\alpha_t y_n h_t(\mathbf{x}_n)\right)}{N\prod_{t=1}^{T} Z_t} \\
&= \frac{\exp\left(-y_n f_T(\mathbf{x}_n)\right)}{N\prod_{t=1}^{T} Z_t}
\end{aligned} \tag{4.3.3}$$

Finalmente, teniendo en cuenta que $\sum_{n=1}^{N} D_t(\mathbf{x}_n) = 1$ para cualquier t y combinando las ecuaciones 4.3.2 y 4.3.3 se obtiene la cota buscada:

$$\begin{aligned}\frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(H(\mathbf{x}_n)\neq y_n) &\leq \frac{1}{N}\sum_{n=1}^{N}\exp(-y_n f_T(\mathbf{x}_n)) \\ &= \sum_{n=1}^{N}\left(\prod_{t=1}^{T} Z_t\right) D_{T+1}(\mathbf{x}_n) \\ &= \prod_{t=1}^{T} Z_t\end{aligned}$$

■

La consecuencia importante del teorema anterior es que para minimizar el error de entrenamiento una aproximación razonable podría ser minimizar avariciosamente la cota obtenida, es decir minimizar Z_t en cada iteración del algoritmo. De hecho, como se explica a continuación, esta idea es la que se aplica en AdaBoost para la elección de los coeficientes α_t que determinan la influencia de cada hipótesis débil h_t en la hipótesis combinada final.

AdaBoost como minimizador de una cota superior del error de entrenamiento

Para simplificar la notación, fijando t, sean $u_n = y_n h_t(\mathbf{x}_n)$, $Z = Z_t$, $D = D_t$, $h = h_t$ y $\alpha = \alpha_t$. En la siguiente discusión se asume sin perdida de generalización que $D(\mathbf{x}_n) \neq 0$ para todo $\mathbf{x}_n$. El objetivo es encontrar α que minimice o aproximadamente minimice Z como función de α.

Suponiendo hipótesis débiles h en el rango $[-1, +1]$, la elección de α se puede obtener aproximando Z como sigue:

$$\begin{aligned}Z &= \sum_{n=1}^{N} D(\mathbf{x}_n)e^{-\alpha u_n} \\ &\leq \sum_{n=1}^{N} D(\mathbf{x}_n)\left(\frac{1+u_n}{2}e^{-\alpha} + \frac{1-u_n}{2}e^{\alpha}\right) \qquad (4.3.4)\end{aligned}$$

Esta cota es válida siempre que $u_n \in [-1, +1]$, y esto es cierto pues, de hecho, debido a que el rango de h e y_n es $\{-1, +1\}$, se cumple que $u_n \in \{-1, +1\}$. La aproximación es esencialmente una cota lineal superior de la función $e^{-\alpha x}$ en el rango $x \in [-1, +1]$ (una prueba de ello se consigue de forma inmediata al estudiar la convexidad de $e^{-\alpha x}$ para cualquier constante $\alpha \in \mathbb{R}$). Como se comenta en [33], resaltar que otras cotas superiores, como por ejemplo cuadráticas o lineales definidas a trozos, podrían dar una mejor aproximación.

Definiendo $r = \sum_{n=1}^{N} D(\mathbf{x}_n)u_n$ y teniendo en cuenta que $\sum_{n=1}^{N} D(\mathbf{x}_n) = 1$, la derivada de la ecuación 4.3.4 respecto a α es la que se muestra a continuación:

$$\begin{aligned}
\frac{\partial Z}{\partial \alpha} &\simeq \frac{\partial}{\partial \alpha} \sum_{n=1}^{N} D(\mathbf{x}_n) \left(\frac{1+u_n}{2} e^{-\alpha} + \frac{1-u_n}{2} e^{\alpha} \right) \\
&= -\frac{1}{2} e^{-\alpha} \sum_{n=1}^{N} D(\mathbf{x}_n)(1+u_n) + \frac{1}{2} e^{\alpha} \sum_{n=1}^{N} D(\mathbf{x}_n)(1-u_n) \\
&= \frac{1}{2} \left(-e^{-\alpha} \sum_{n=1}^{N} D(\mathbf{x}_n) - e^{-\alpha} \sum_{n=1}^{N} D(\mathbf{x}_n)u_n + e^{\alpha} \sum_{n=1}^{N} D(\mathbf{x}_n) - e^{\alpha} \sum_{n=1}^{N} D(\mathbf{x}_n)u_n \right) \\
&= \frac{1}{2} \left(-e^{-\alpha} - e^{-\alpha} r + e^{\alpha} - e^{\alpha} r \right) \\
&= \frac{1}{2} \left(-e^{-\alpha}(1+r) + e^{\alpha}(1-r) \right)
\end{aligned}$$

Igualando a 0 la expresión anterior y despejando α:

$$\begin{aligned}
e^{\alpha}(1-r) &= e^{-\alpha}(1+r) \\
e^{2\alpha} &= \frac{1+r}{1-r} \\
\alpha &= \frac{1}{2} \ln \frac{1+r}{1-r}
\end{aligned} \tag{4.3.5}$$

Por otra parte, considerando problemas de dos clases en los que $u_n = y_n h(\mathbf{x}_n) \in \{-1, +1\}$, se define el error ponderado en la iteración t como:

$$\varepsilon = \sum_{n=1}^{N} D(\mathbf{x}_n) \mathbb{I}(y_n \neq h(\mathbf{x}_n)) = \sum_{n:u_n=-1} D(\mathbf{x}_n) \tag{4.3.6}$$

El valor de r se puede expresar en función de ε_t así:

$$\begin{aligned}
r &= \sum_{n=1}^{N} D(\mathbf{x}_n)u_n \\
&= \sum_{n:u_n=+1} D(\mathbf{x}_n) - \sum_{n:u_n=-1} D(\mathbf{x}_n) \\
&= \left(1 - \sum_{n:u_n=-1} D(\mathbf{x}_n) \right) - \sum_{n:u_n=-1} D(\mathbf{x}_n) \\
&= 1 - 2 \sum_{n:u_n=-1} D(\mathbf{x}_n) \\
&= 1 - 2\varepsilon_t
\end{aligned} \tag{4.3.7}$$

Sustituyendo finalmente 4.3.7 en 4.3.5 se consigue el valor de α_t empleado por AdaBoost en problemas binarios:

$$\alpha_t = \frac{1}{2} \ln \frac{1-\varepsilon_t}{\varepsilon_t} \tag{4.3.8}$$

Convergencia a 0 del error de entrenamiento de AdaBoost

Motivado el procedimiento que sigue AdaBoost para calcular los factores α_t, y con él la combinación de las hipótesis h_t y la actualización de las distribuciones D_t, se puede comprobar que este mecanismo realmente hace converger el error de entrenamiento a 0, y mantener la suposición de que este hecho haga obtener fronteras de separación de clases adecuadas para clasificar conjuntos de test independientes.

Teorema 4.3.2. *Suponiendo que se aplica el algoritmo AdaBoost definido en la figura 4.3.1 y que se cumple que* $\varepsilon_t(h_t, D_t) \leq \frac{1}{2} - \gamma_t$*, entonces el error de entrenamiento cumple:*

$$\widehat{err}(H) \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right)$$

Demostración: En el teorema 4.3.1 se estableció la siguiente cota del error de entrenamiento:

$$\widehat{err}(H) \leq \prod_{t=1}^{T} Z_t \tag{4.3.9}$$

donde $Z_t = \sum_{n=1}^{N} D_t(\mathbf{x}_n)e^{-\alpha_t y_n h_t(\mathbf{x}_n)}$.

Para problemas binarios en los que $y_n \in \{-1, +1\}$, se puede escribir Z_t en función de ε_t (*véase 4.3.6*) como sigue:

$$\begin{aligned} Z_t &= \sum_{n=1}^{N} D_t(\mathbf{x}_n)e^{-\alpha_t y_n h_t(\mathbf{x}_n)} \\ &= \sum_{n:y_n=h_t(\mathbf{x}_n)} D_t(\mathbf{x}_n)e^{-\alpha_t} + \sum_{n:y_n\neq h_t(\mathbf{x}_n)} D_t(\mathbf{x}_n)e^{\alpha_t} \\ &= (1-\varepsilon_t)e^{-\alpha_t} + \varepsilon_t e^{\alpha_t} \end{aligned} \tag{4.3.10}$$

Tomando $\alpha_t = \frac{1}{2}\ln\frac{1-\varepsilon_t}{\varepsilon_t}$ (*véase 4.3.8*) en la expresión anterior:

$$\begin{aligned} Z_t &= (1-\varepsilon_t)e^{-\alpha_t} + \varepsilon_t e^{\alpha_t} \\ &= 2\sqrt{\varepsilon_t(1-\varepsilon_t)} \end{aligned} \tag{4.3.11}$$

Sustituyendo este último valor de Z_t en la ecuación 4.3.9:

$$\begin{aligned} \widehat{err}(H) &\leq \prod_{t=1}^{T} Z_t \\ &= \prod_{t=1}^{T} (4\varepsilon_t(1-\varepsilon_t))^{\frac{1}{2}} \end{aligned} \tag{4.3.12}$$

Finalmente, sólo queda renombrar $\varepsilon_t = \frac{1}{2} - \gamma_t$ y acotar la expresión para demostrar el teorema:

$$\begin{aligned} \widehat{err}(H) &\leq \prod_{t=1}^{T} (4\varepsilon_t(1-\varepsilon_t))^{\frac{1}{2}} \\ &= \prod_{t=1}^{T} (1-4\gamma_t^2)^{\frac{1}{2}} \\ &= \exp\left(\frac{1}{2}\sum_{t=1}^{T} \ln(1-4\gamma_t^2)\right) \\ &\leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right) \end{aligned} \tag{4.3.13}$$

■

A partir esta cota se infiere que la condición $\sum_{t=1}^{T}\gamma_t^2 \to \infty$ es suficiente para garantizar que $\widehat{err}(H) \to 0$. Recordando que el error de la hipótesis h_t es $\varepsilon_t = \frac{1}{2} - \gamma_t$, y teniendo en cuenta que una hipótesis que clasifique cada instancia de forma aleatoria tiene un error de $\frac{1}{2}$ (en problemas binarios), γ_t mide cuánto mejores son las predicciones de h_t respecto a predicciones aleatorias. De este modo, si una hipótesis débil es ligeramente mejor que un clasificador aleatorio de tal modo que $\gamma_t > \gamma$ para algún $\gamma > 0$, entonces el error de entrenamiento cae exponencialmente rápido.

Sobreajuste en AdaBoost

Demostrada la convergencia del error de entrenamiento a 0, se plantea la necesidad de definir cotas para el error de generalización $err(H)$. Suponiendo que $h_1, \ldots, h_T \in \mathcal{H}$, y que $|\mathcal{H}|$ es finito[†], se puede obtener la siguiente cota de AdaBoost para $err(H)$ que involucra el error empírico $\widehat{err}(H)$:

$$err(H) \leq \widehat{err}(H) + \mathcal{O}\left(\sqrt{\frac{T\ln|\mathcal{H}| + T\ln\frac{N}{T} + \ln\frac{1}{\delta}}{N}}\right) \tag{4.3.14}$$

Es sencillo observar que este resultado predice sobreajuste. Al incrementar T, el error $\widehat{err}(H)$ disminuye, pero el segundo término de la ecuación aumenta, y eventualmente supera a $\widehat{err}(H)$. El comportamiento esperado sería entonces el mostrado en la figura 4.3.2.

[†] Existen cotas para el caso en el que $|\mathcal{H}|$ es infinito [39] y se definen en función de la dimensión-VC (de Vapnik y Chervonenkis). Con ellas las discusiones de esta sección se realizarían de manera análoga, pero no se dan aquí por complicar la notación y escapar al propósito de este texto.

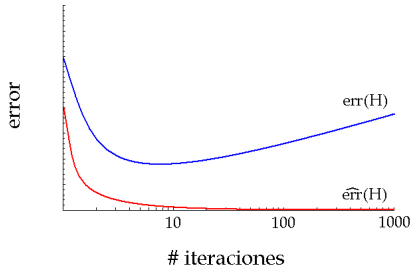

Figura 4.3.2: Comportamiento del error esperado en AdaBoost debido al sobreajuste.

Sin embargo, diversos estudios empíricos demostraron que a menudo no se observa sobreajuste en AdaBoost. Por ejemplo, la figura 4.3.3, extraída de [39], muestra la evolución de los errores de entrenamiento y generalización de AdaBoost cuando se aplica sobre árboles de decisión C4.5 como algoritmos de aprendizaje del problema *letters* [6]. En ella, el error de test no aumenta incluso después de 1000 iteraciones, y de hecho continua bajando cuando el error de entrenamiento es 0. C4.5 es un aprendiz débil y las predicciones obtenidas son más complejas al incrementar el número de iteraciones T. En este sentido, el principio de la navaja de Occam no se cumple, pues las reglas simples no predicen mejor que los modelos complejos generados. La explicación a esta controversia hay que buscarla en los márgenes y la generalización de clasificación de AdaBoost.

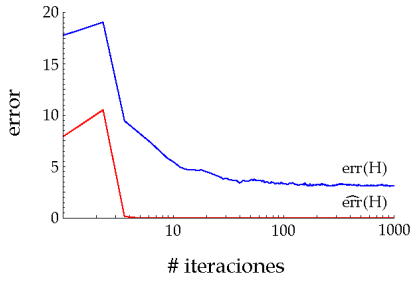

Figura 4.3.3: Curvas de error [39] para C4.5–AdaBoost aplicado en el conjunto de datos *letters*.

Márgenes de clasificación y error de generalización de AdaBoost

La razón de que no se observe sobreajuste en diversos problemas se debe a que al continuar haciendo boosting, las predicciones construidas son más "certeras", y proporcionan mayores *márgenes* respecto a las fronteras de separación en la clasificación de ejemplos. Cuanto mayor es el margen, mejor es el error de generalización. Diversos autores [39] observaron este hecho y lo fundamentaron teóricamente con nuevas cotas, basadas esta vez en márgenes de clasificación.

Para definir formalmente el concepto de margen, por conveniencia, considérense los factores α_t normalizados, hecho que no cambia las predicciones de $H(\mathbf{x})$.

$$\begin{aligned} H(\mathbf{x}) &= sign\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right) \\ \alpha_t &\geq 0 \\ \sum_{t=1}^{T} \alpha_t &= 1 \end{aligned}$$

Definición 4.3.1. *El **margen de clasificación** de un ejemplo* $(\boldsymbol{x}, y_{\boldsymbol{x}}) \in \mathcal{X} \times \{-1, +1\}$ *se define como sigue*

$$\begin{aligned} margen(\boldsymbol{x}, y_{\boldsymbol{x}}) &= y_{\boldsymbol{x}} f_T(\boldsymbol{x}) \\ &= \sum_{t=1}^{T} y_{\boldsymbol{x}} \alpha_t h_t(\boldsymbol{x}) \\ &= \sum_{t: y_{\boldsymbol{x}} = h_t(\boldsymbol{x})} \alpha_t - \sum_{t: y_{\boldsymbol{x}} \neq h_t(\boldsymbol{x})} \alpha_t \end{aligned}$$

El primer término de la expresión anterior se corresponde con la influencia de las hipótesis con predicciones correctas, mientras que el segundo equivale a la influencia de las hipótesis incorrectas. El margen es positivo ($y_{\mathbf{x}} f_T(\mathbf{x}) > 0$) si y sólo si $y_{\mathbf{x}} = H(\mathbf{x})$. El valor $|y_{\mathbf{x}} f_T(\mathbf{x})|$ mide la "fuerza" o certeza del voto. Si $|y_{\mathbf{x}} f_T(\mathbf{x})| \simeq 0$, la predicción del ejemplo $\mathbf{x}$, correcta o incorrecta, se encuentra cerca de la frontera de separación, y un cambio mínimo en la entrada podría provocar el cambio en la clase asignada.

Para AdaBoost, la figura 4.3.4 [39] muestra la distribución acumulativa del margen en los ejemplos de entrenamiento después de 5, 100 y 1000 iteraciones de boosting sobre C4.5 en el problema *letters*. Se puede observar como se incrementan los márgenes incluso después de que el error de entrenamiento haya alcanzado el valor de 0 (después de 5 iteraciones).

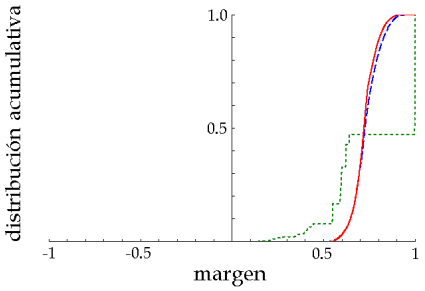

Figura 4.3.4: Distribuciones acumulativas de los márgenes [39] de C4.5–AdaBoost en el conjunto de datos *letters* para 5, 100 y 1000 iteraciones; indicadas respectivamente por las curvas punteada, rayada y continua.

Sea $\mathcal{C}(\mathcal{H})$ la envolvente convexa del conjunto finito de hipótesis $\mathcal{H}$

$$\mathcal{C}(\mathcal{H}) = \left\{ f \text{ de la forma } f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(x), \text{ donde } \alpha_t \geq 0, \sum_{t=1}^{T} \alpha_t = 1, h_t \in \mathcal{H} \right\}$$

Si D es la distribución $\mathcal{X} \times \{-1, +1\}$ que siguen los ejemplos del problema a tratar, S es la muestra con N ejemplos de entrenamiento, $Pr_D[\pi]$ es la probabilidad de que se cumpla π cuando los ejemplos $(\mathbf{x}, y_\mathbf{x})$ se escogen de acuerdo a D (e.g. $Pr_D[y_\mathbf{x} \neq H(\mathbf{x})] = err(H)$), y $Pr_S[\pi]$ es la probabilidad de que se cumpla π cuando los ejemplos $(\mathbf{x}, y_\mathbf{x})$ se escogen de la muestra S (e.g. $Pr_S[y_\mathbf{x} \neq H(\mathbf{x})] = \widehat{err}(H)$), se puede demostrar [39] la siguiente cota del error de generalización de AdaBoost:

Teorema 4.3.3. *Con probabilidad de al menos* $1 - \delta$, $\forall f \in \mathcal{C}(\mathcal{H})$, $\forall \theta > 0$,

$$err(H) = Pr_D[y_{\boldsymbol{x}} f(\boldsymbol{x}) \leq 0] \leq Pr_S[y_{\boldsymbol{x}} f(\boldsymbol{x}) \leq \theta] + \mathcal{O}\left(\frac{1}{\sqrt{N}} \sqrt{\frac{\ln N \ln |\mathcal{H}|}{\theta^2} + \ln \frac{1}{\delta}} \right)$$

Nótese que esta vez el segundo término es independiente del número de iteraciones y que depende del margen θ considerado. Para el primer término, correspondiente al error de entrenamiento basado en margen, es sencillo [39] dar una nueva cota equivalente (considerando $\theta = 0$) a la expresión 4.3.12.

$$\widehat{err}_\theta(H) = Pr_S[y_\mathbf{x} f(\mathbf{x}) \leq \theta] \leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} \tag{4.3.15}$$

En ella, si se considera que $\epsilon_t = \frac{1}{2} - \gamma_t \leq \frac{1}{2} - \gamma$ y $\theta > 0$, i.e., las predicciones de los clasificadores base son ligeramente mejores que predicciones aleatorias, entonces:

$$\begin{aligned} \widehat{err}_\theta(H) = Pr_S[y_\mathbf{x} f(\mathbf{x}) \leq \theta] &\leq 2^T \prod_{t=1}^{T} \sqrt{\epsilon_t^{1-\theta}(1-\epsilon_t)^{1+\theta}} \\ &\leq \left(\sqrt{(1-2\gamma)^{1-\theta}(1+2\gamma)^{1+\theta}} \right)^T \end{aligned} \tag{4.3.16}$$

La base de la potencia (encerrada entre paréntesis) en esta última expresión 4.3.16 es estrictamente menor que 1 cuando $\theta < \gamma$. Por esta razón, $Pr_S[y_\mathbf{x} f(\mathbf{x}) \leq \theta]$ va a 0 cuando T tiende a ∞ y $\theta < \gamma$. De este modo, se conserva la convergencia a 0 del error de entrenamiento, pero se evita ahora (gracias al segundo término de la cota del teorema 4.3.3) la implicación de mayor sobreajuste a medida que el número de iteraciones T aumenta.

Capítulo 5

Muestras Extremas, Perceptrones Paralelos y AdaBoost

Hasta este capítulo tres son los temas que se han tratado. Inicialmente se introdujeron los problemas de clasificación de Muestra Extrema, en los que existe un considerable desequilibrio entre el número de representantes de la clase de interés y del resto de clases. Se explicaron las principales técnicas de equilibrado que han sido propuestas para abordarlos, y se eligió una idea de ellas, la **selección de patrones cercanos a las fronteras de clasificación** para definir las muestras de entrenamiento, como la mejor alternativa a estudiar. A continuación, se describieron los novedosos **Perceptrones Paralelos**, su arquitectura y su algoritmo de aprendizaje para problemas de clasificación binarios; haciendo especial hincapié en su rápido entrenamiento con mínima comunicación entre las unidades, y en el concepto de margen que utilizan para estabilizar sus salidas. Finalmente, se motivaron los Métodos de Conjuntos de Modelos como mecanismo de mejora en la eficacia de los clasificadores individuales, se centró la atención en los métodos basados en la selección de diferentes conjuntos de entrenamiento para la construcción de los modelos (bagging y boosting), y se explicaron más detenidamente los fundamentos teóricos del algoritmo **AdaBoost**.

Las próximas dos secciones enlazan estos temas y exponen las dos propuestas de esta investigación: PPTSS y PPBoost. En la primera se da un algoritmo de selección de patrones de entrenamiento frontera, que está orientado a mejorar la clasificación global en Muestras Extremas y que usa el margen surgido en el entrenamiento de los Perceptrones Paralelos. En la segunda, a partir de las ideas desarrolladas en el algoritmo anterior, se propone una modificación de AdaBoost con Perceptrones Paralelos como aprendices débiles, que da más relevancia a los patrones frontera durante el proceso de boosting y que supone una estrategia de selección de patrones de entrenamiento más progresiva que la primera.

El objetivo de ambas estrategias será emplear el voto mayoritario de los componentes de un Perceptrón Paralelo para poder identificar y reducir la influencia en el aprendizaje de los patrones de entrenamiento redundantes y ruidosos, bien sea mediante su eliminación de los conjuntos de datos (PPTSS), bien sea mediante la reducción de sus pesos en el proceso de boosting (PPBoost). En el presente capítulo se motivarán y describirán en detalle, mientras que será en el capítulo siguiente donde se mostrarán los resultados empíricos obtenidos con ambas técnicas, y donde se comprobará un considerable equilibrado entre los ejemplos de cada clase, así como un alcance y mejora en algunos casos de los resultados ofrecidos por los robustos Perceptrones Multicapa.

5.1 Selección de Conjuntos de Entrenamiento en Muestras Extremas usando Perceptrones Paralelos: PPTSS

Cuando se tratan conjuntos de datos desequilibrados, uno podría esperar que ciertos ejemplos positivos y negativos pudiesen ser descartados independientemente del procedimiento de construcción del clasificador empleado. Entre ellos, por una parte, estarían los que se caracterizan por estar representados por otros patrones que se pueden considerar seguros a la hora de ser clasificados. Denominados "redundantes" en [29], serían buenos candidatos a ser eliminados del proceso de entrenamiento al no aportar información relevante en el aprendizaje. Por otra parte, estarían aquellos patrones, menos esperanzadores, que no pudiesen ser clasificados correctamente bajo cualquier discriminante, debido a que sus atributos claramente los estableciesen en una clase que no es a la que pertenecen. Estos patrones "mal etiquetados" ("ruidosos" a partir de ahora), serían adecuados para ser excluidos también del entrenamiento, al ser propensos a confundir al algoritmo de aprendizaje.

Al no ser adecuados para definir las fronteras de separación de clases, ambos tipos de patrones no son interesantes para el entrenamiento, y su eliminación del mismo sugiere un procedimiento general de tratamiento de las muestras de datos utilizadas. El problema por supuesto está en cómo definir este mecanismo de filtrado. En la literatura se han dado diversas alternativas, algunas de ellas [29, 19] basadas en el concepto de *margen* de clasificación. La idea es encontrar para el aprendizaje aquellos patrones más cercanos (en un margen de separación) a la frontera de clasificación y descartar el resto, donde se encontrarían los ejemplos redundantes y ruidosos. En el contexto de PP, como se explica en el *capítulo 3*, los márgenes surgen de manera natural durante el entrenamiento, y por ello su uso para definir y eliminar patrones de entrenamiento redundantes y ruidosos es la propuesta que se expone en esta sección, denominada a partir de ahora como PPTSS (del inglés *Parallel Perceptron Training Set Selection*) [9, 10].

Recordando algunas de las ideas expuestas en el *capítulo 3*, sea un PP con H perceptrones estándar, cada uno de los cuales con un vector de pesos $\mathbf{w} \in \mathbb{R}^D$, en el que la componente $w^{(D)}$ representa su umbral. Para una entrada $(\mathbf{x}, y_\mathbf{x}) \in \mathbb{R}^D \times \{-1, +1\}$, con $x^{(D)} = -1$, el perceptrón h computa la función $sign(\mathbf{w}_h \cdot \mathbf{x}) = sign(act_h(\mathbf{x}))$ donde $act_h(\mathbf{x})$ es la activación del perceptrón h debida a $\mathbf{x}$. De este modo, para clasificación binaria la salida del PP es entonces:

$$\begin{aligned} \hat{y}_\mathbf{x} &= sign\left(\sum_{h=1}^{H} sign(act_h(\mathbf{x}))\right) \\ &= sign\left(\#\{h : act_h(\mathbf{x}) > 0\} - \#\{h : act_h(\mathbf{x}) \leq 0\}\right) \end{aligned}$$

Suponiendo H impar, la salida es correcta si $\#\{h : act_h(\mathbf{x}) > 0\} > \#\{h : act_h(\mathbf{x}) \leq 0\}$ cuando $y_\mathbf{x} = +1$, si $\#\{h : act_h(\mathbf{x}) > 0\} < \#\{h : act_h(\mathbf{x}) \leq 0\}$ cuando $y_\mathbf{x} = -1$, o bien, de manera unificada, si $\#\{h : y_\mathbf{x} act_h(\mathbf{x}) > 0\} > \#\{h : y_\mathbf{x} act_h(\mathbf{x}) \leq 0\}$ para cualquier valor de $y_\mathbf{x} \in \{-1, +1\}$.

El valor de $y_\mathbf{x} act_h(\mathbf{x})$ es lo que se define a partir de ahora como la *activación normalizada* del perceptrón h debida a $(\mathbf{x}, y_\mathbf{x})$, y corresponde a lo que se introdujo en el *capítulo 3* como el "margen" de $(\mathbf{x}, y_\mathbf{x})$ para el perceptrón h. Nótese que el perceptrón clasifica bien al ejemplo siempre que tenga margen positivo, i.e., $y_\mathbf{x} act_h(\mathbf{x}) > 0$, y que el valor de $|y_\mathbf{x} act_h(\mathbf{x})|$ da idea de la fuerza de la predicción. Cuanto mayor sea este valor, más lejos estará la proyección del patrón del hiperplano de separación, bien sea por la parte correcta (cuando $y_\mathbf{x} act_h(\mathbf{x}) > 0$) o por la incorrecta (cuando $y_\mathbf{x} act_h(\mathbf{x}) \leq 0$), y más estable será la predicción del perceptrón. Por el contrario, si $|y_\mathbf{x} act_h(\mathbf{x})| \simeq 0$, una modificación mínima de los pesos $\mathbf{w}_h$ del perceptrón podría provocar un cambio en el signo de su activación $act_h(\mathbf{x})$ y por tanto en su predicción del patrón. Este hecho, junto con la consideración del margen de estabilización γ de un PP, son los aspectos clave del mecanismo de filtrado PPTSS.

Contrariamente a lo que sucede en, por ejemplo, las SVM, donde los márgenes afectan a salidas unidimensionales, un PP proporciona un único margen γ para sus H perceptrones. En cualquier caso, el valor de γ obtenido tras el entrenamiento refleja cómo de apartados están la mayoría de los patrones de los hiperplanos de separación. Así, se establecen cuatro "zonas" principales para cada perceptrón h y en las que se incluirán los patrones $(\mathbf{x}, y_\mathbf{x})$: altamente incorrectos, incorrectos cercanos a la frontera, correctos cercanos a la frontera y altamente correctos, de acuerdo a si la activación normalizada $y_\mathbf{x} act_h(\mathbf{x})$ pertenece respectivamente a los intervalos $(-\infty, -\gamma), (-\gamma, 0), (0, \gamma), (\gamma, \infty)$. De este modo, podría ser tentador denominar a los patrones frontera como los "vectores soporte" que definen de manera inequívoca y única el hiperplano de separación del PP. Sin embargo, obsérvese que un patrón podría proporcionar un margen positivo sobre un perceptrón y darlo negativo sobre los otros. Teniendo este hecho presente, y denotando $\mathcal{T}_R$ al conjunto de patrones

redundantes (*redundant*), $\mathcal{T}_N$ al conjunto de patrones ruidosos (*noisy*) y $\mathcal{T}_B$ al conjunto de patrones frontera (*borderline*), se sugiere la siguiente categorización de patrones:

Definición 5.1.1. *Dados un patrón* $(\boldsymbol{x}, y_{\boldsymbol{x}})$ *y un PP con perceptrones estándar* $h = 1, \ldots, H$, *se establece que*

- $(\boldsymbol{x}, y_{\boldsymbol{x}}) \in \mathcal{T}_R$ *si la mayoría de los* H *perceptrones dan un margen* $y_{\boldsymbol{x}} act_h(\boldsymbol{x}) > \gamma$,
- $(\boldsymbol{x}, y_{\boldsymbol{x}}) \in \mathcal{T}_N$ *si la mayoría de los* H *perceptrones dan un margen* $y_{\boldsymbol{x}} act_h(\boldsymbol{x}) < -\gamma$,
- $(\boldsymbol{x}, y_{\boldsymbol{x}}) \in \mathcal{T}_B$ *en otro caso.*

Estas definiciones consideran el voto mayoritario de los perceptrones de un PP, y establecen como patrones frontera a aquellos ejemplos para los cuales al menos un perceptrón proporciona una activación entre $-\gamma$ y γ, siempre que la mayoría de los perceptrones no coincidan en clasificarlos correcta o incorrectamente fuera de margen. Ahora bien, entre estos ejemplos se encuentran los patrones *quasi-ruidosos* que están mal clasificados por todos los perceptrones, aunque sea con un margen pequeño en $(-\gamma, 0)$. Empíricamente se ha comprobado que la eliminación de los patrones negativos que cumplen esta condición (a partir de ahora $\mathcal{W}_{B-}$, $all\text{–}Wrong_{B-}$), adicionalmente a la de los redundantes y ruidosos, mejora los resultados obtenidos (*véase capítulo 6*). Sin embargo, también se ha observado que no conviene quitar los ejemplos positivos que la cumplen. Debido a que los problemas tratados son de Muestra Extrema, se corre el peligro de que se eliminen todos o prácticamente todos los representantes de la clase minoritaria.

Como ejemplo ilustrativo, la figura 5.1.1 muestra la identificación de patrones redundantes, ruidosos y quasi–ruidosos en dos problemas de clasificación binaria sencillos, en los que las clases se distribuyen según 2 y 3 nubes Gaussianas en un espacio de atributos de dimensión 2. Empleando un PP de 3 perceptrones, se representan (en línea continua) las fronteras de separación alcanzadas por estos últimos, así como (en línea discontinua) sus hiperplanos paralelos a distancia γ. Se puede comprobar que (1) los ejemplos marcados como redundantes (mediante cruces), con proyecciones alejadas a más distancia que γ de la mayoría de las 3 fronteras de separación, están bien representados por otros más cercanos a las fronteras, y que (2) los ejemplos marcados como ruidosos (mediante cuadrados) están en general en zonas de solapamiento de las dos clases, y por tanto dificultan la definición de las fronteras. Los patrones marcados como quasi–ruidosos (mediante triángulos) son muy escasos en los ejemplos representados. En el caso del problema de las 2 distribuciones Gaussianas sólo se detecta uno, pero es un buen candidato a ser eliminado, pues puede considerarse dentro de la región asociada a la clase opuesta.

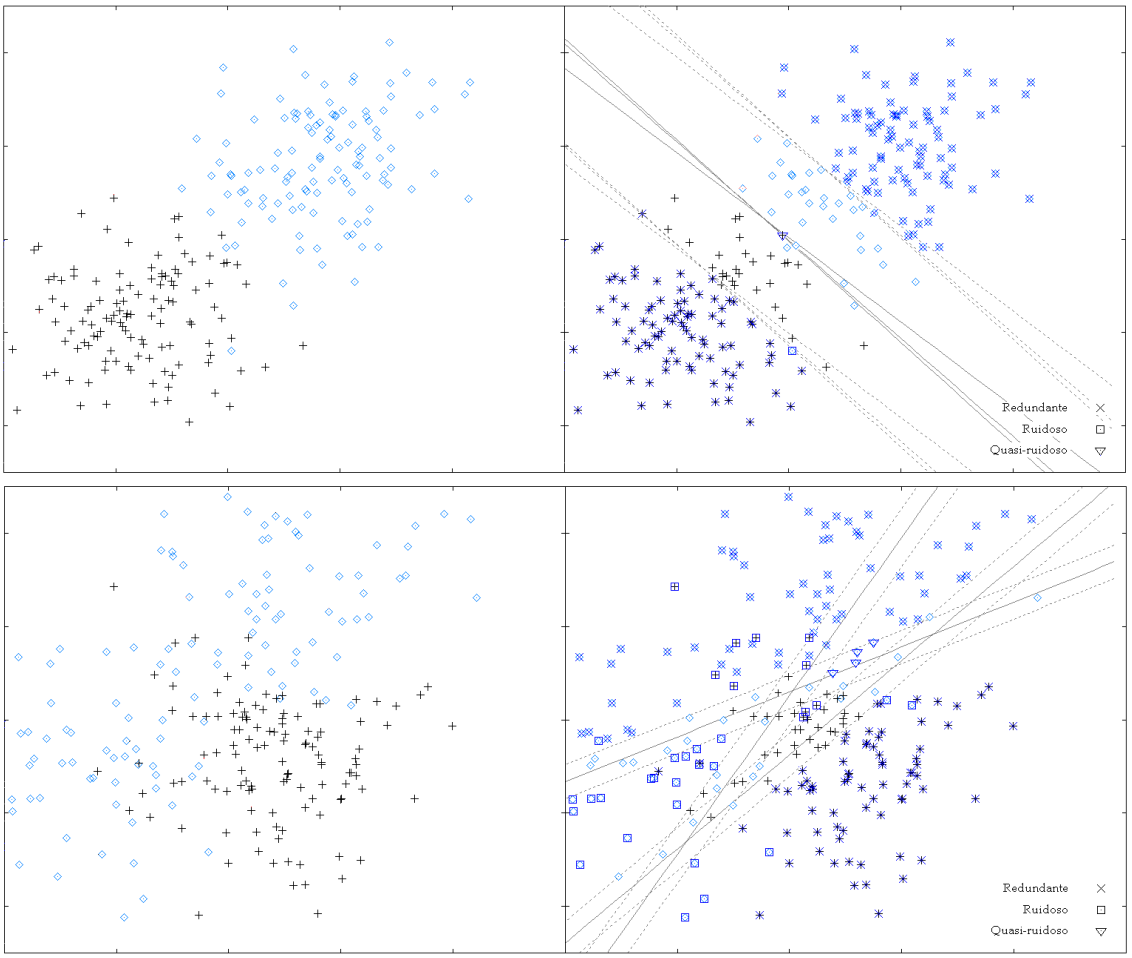

Figura 5.1.1: Identificación de patrones redundantes, ruidosos y quasi-ruidosos en 2 problemas de clasificación binaria donde las clases siguen 2 y 3 distribuciones Gaussianas.

Con todo lo explicado, el procedimiento de selección de patrones propuesto es el que sigue. De manera iterativa se entrena al PP, se define el margen γ y se eliminan los patrones redundantes, ruidosos y frontera negativos mal clasificados por todos los perceptrones: después de la iteración de entrenamiento t se excluyen los subconjuntos $\mathcal{T}_R^{(t)}$, $\mathcal{T}_N^{(t)}$ y $\mathcal{W}_{B-}^{(t)}$ del conjunto de entrenamiento $\mathcal{T}^{(t)}$ obtenido en la iteración previa. Por otra parte, como se desea que esta selección se oriente a problemas de Muestra Extrema, se establece como condición de parada que los valores de acc^+ y g (*véase sección 2.5*) hayan empeorado respecto a iteraciones previas. Con esta consideración se persigue mejorar la clasificación de los ejemplos positivos (aumentando el valor de acc^+), pero intentando mantener una buena clasificación de los ejemplos negativos (manteniendo el valor g). Si sólo se considerase la mejora de acc^+, el deterioro de acc^-, y por tanto el de acc, podría ser desastroso. La figura 5.1.2 reúne las ideas expuestas.

Dados: conjunto de entrenamiento $\mathcal{T}$*, vector de pesos* $\mathbf{w}$ *y margen* γ *del PP*
Inicializar $t := 0$, $g_{opt} := 0$, $acc^+_{opt} := 0$
$[g, acc^+, \mathbf{w}^{(t)}, \gamma^{(t)}] \leftarrow$ *EntrenarPP*$(\mathcal{T}^{(t)}, \mathbf{w}, \gamma)$
Mientras $g \geq g_{opt}$ *y* $acc^+ \geq acc^+_{opt}$*:*

- $g_{opt} := g$, $acc^+_{opt} := acc^+$, $\mathbf{w}_{opt} := \mathbf{w}^{(t)}$
- $[\mathcal{T}_R^{(t)}, \mathcal{T}_N^{(t)}, \mathcal{W}_{B-}^{(t)}] \leftarrow$ *SeleccionarPatrones*$(\mathcal{T}^{(t)}, \mathbf{w}^{(t)}, \gamma^{(t)})$
- $\mathcal{T}^{(t+1)} \leftarrow$ *EliminarPatrones*$(\mathcal{T}^{(t)}, \mathcal{T}_R^{(t)}, \mathcal{T}_N^{(t)}, \mathcal{W}_{B-}^{(t)})$
- $[g, acc^+, \mathbf{w}^{(t+1)}, \gamma^{(t+1)}] \leftarrow$ *EntrenarPP*$(\mathcal{T}^{(t+1)}, \mathbf{w}, \gamma)$
- $t := t + 1$

Figura 5.1.2: Esquema general del algoritmo PPTSS.

En el *capítulo 6* se exponen los resultados empíricos obtenidos [10] con el algoritmo, pero antes, en la próxima sección, y siguiendo la categorización de patrones 5.1.1 para PP se explica una segunda aproximación de filtrado basada en el algoritmo AdaBoost.

5.2 Adaptación de AdaBoost a Muestras Extremas usando el Perceptrón Paralelo como aprendiz débil: PPBoost

Como ya se explicó en el *capítulo 4*, AdaBoost construye un clasificador fuerte combinando un conjunto de aprendices débiles y concentrándose sucesivamente en aquellos patrones más difíciles de clasificar. Más concretamente, en cada iteración t guarda una distribución $D_t(\mathbf{x})$ sobre los patrones de entrenamiento $\mathbf{x}$, y después de construir una nueva hipótesis h_t, actualiza la distribución $D_t(\mathbf{x})$ según la expresión

$$D_{t+1}(\mathbf{x}) = \frac{1}{Z_t} D_t(\mathbf{x}) e^{-\alpha_t y_{\mathbf{x}} h_t(\mathbf{x})} \tag{5.2.1}$$

donde $y_{\mathbf{x}} = \pm 1$ es la etiqueta de clase asociada a $\mathbf{x}$, Z_t es una constante de normalización de las probabilidades $D_{t+1}(\mathbf{x})$, y α_t está relacionada con el error de entrenamiento ε_t y el peso sobre el clasificador final de la hipótesis h_t (*véase ecuación 4.3.8*). De este modo, después de cada iteración, AdaBoost pone mayor énfasis sobre aquellos patrones mal clasificados, ya que $e^{-\alpha_t y_{\mathbf{x}} h_t(\mathbf{x})} > 1$ si $y_{\mathbf{x}} h_t(\mathbf{x}) < 0$, obteniendo con ello, como se demostró en la *sección 4.3*, un descenso a 0 del error de entrenamiento ε_t.

AdaBoost se ha empleado con gran éxito en muchas aplicaciones y sobre diversos conjuntos de datos [24, 32, 16]. Sin embargo, también se ha comprobado que no alcanza tan buenos resultados cuando se aplica sobre problemas con ruido [24]. De hecho, considérese

un patrón con una etiqueta ruidosa, esto es, aunque claramente pertenezca a una clase, su etiqueta se corresponde con la otra clase. Este patrón tenderá a ser clasificado incorrectamente por las sucesivas hipótesis, y por tanto verá incrementada su probabilidad, causando de este modo que el proceso de boosting se concentre sobre él esperanzado en poder clasificarlo bien con posterioridad. Aunque este hecho podría ser útil en determinadas situaciones (como la propia detección de patrones ruidosos [42]), su consecuencia más frecuente es el deterioro progresivo de la hipótesis final.

La situación expuesta puede darse con facilidad cuando se tratan problemas desequilibrados, donde la presencia de los patrones minoritarios se ve oscurecida por el resto de patrones mayoritarios. Como se explicó en la sección anterior, en este tipo de problemas es fundamental detectar tanto los ejemplos redundantes como los ruidosos para no considerarlos en el aprendizaje. AdaBoost actúa adecuadamente con los redundantes, disminuyendo sus probabilidades de selección al estar bien clasificados, y obligando con ello a reducir su impacto en la construcción del clasificador. Sin embargo, no lo hace con los ruidosos, sobre los que incrementa la probabilidad de selección al estar mal clasificados, y aumenta de este modo su influencia en la construcción del clasificador.

La segunda propuesta de esta investigación, PPBoost [11], se basa en la detección mediante PP de los patrones ruidosos y en la modificación de la regla de actualización de probabilidades en AdaBoost para reducir su relevancia en el proceso de boosting. En concreto, empleando PP y con la definición 5.1.1 se conoce la categoría (redundante, ruidoso o frontera) a la que pertenece un patron $\mathbf{x}$ dado. A partir de ella, se establece un factor $R(\mathbf{x})$ que refleja la naturaleza del patrón y que modifica la ecuación 5.2.1 a esta otra:

$$D_{t+1}(\mathbf{x}) = \frac{1}{Z'_t} D_t(\mathbf{x}) e^{-\alpha_t R_t(\mathbf{x}) y_{\mathbf{x}} h_t(\mathbf{x})} \tag{5.2.2}$$

donde Z'_t es de nuevo un factor de normalización de las probabilidades $D_{t+1}(\mathbf{x})$. En este contexto, si $R_t(\mathbf{x}) = 1$ se recupera el procedimiento normal de boosting. Sin embargo, debido a que se desea disminuir la influencia de los patrones ruidosos $\mathbf{x} \in \mathcal{T}_N$, para estos últimos se establece $R_t(\mathbf{x}) = -1$, ya que al estar mal clasificados ($y_{\mathbf{x}} h_t(\mathbf{x}) = -1$) se obtiene que $\alpha_t R_t(\mathbf{x}) y_{\mathbf{x}} h_t(\mathbf{x}) > 0$ y así $D_{t+1}(\mathbf{x}) < D_t(\mathbf{x})$. Para el caso de los patrones redundantes $\mathbf{x} \in \mathcal{T}_R$, también candidatos a ser eliminados del entrenamiento, el valor de $R_t(\mathbf{x})$ se mantiene a 1. Debido a que están bien clasificados ($y_{\mathbf{x}} h_t(\mathbf{x}) = 1$), se tiene que $\alpha_t R_t(\mathbf{x}) y_{\mathbf{x}} h_t(\mathbf{x}) > 0$ y por tanto $D_{t+1}(\mathbf{x}) < D_t(\mathbf{x})$.

Con esta modificación del procedimiento de actualización de probabilidades, manteniendo $R_t(\mathbf{x}) = 1$ para todos los patrones frontera $\mathbf{x} \in \mathcal{T}_B$, se obliga a que AdaBoost se concentre en ejemplos mal clasificados, pero cercanos al hiperplano de clasificación, obteniendo (*véase capítulo 6*) sustanciales mejoras en el *accuracy* global. Sin embargo, como

ya se explicó en el *capítulo 2*, esta medida no es significativa en problemas de muestra desequilibrada, donde es fundamental llegar a un compromiso entre el porcentaje de aciertos en las clases mayoritaria y minoritaria. Por ello, al igual que ocurría en la selección de conjuntos de entrenamiento mediante PP, cuyo esquema se expuso en la figura 5.1.2, se hace interesante estudiar el efecto que tiene tratar de manera independiente aquellos patrones mayoritarios *quasi-ruidosos* $\mathbf{x} \in \mathcal{W}_{B-}$, clasificados incorrectamente por todos los perceptrones aunque sea dentro del margen γ del PP. Para ellos, cuando se trabaje con problemas desequilibrados, si $R_t(\mathbf{x}) = 1$, AdaBoost se centrará en un mayor porcentaje de ejemplos *quasi-ruidosos* negativos y se espera que *acc* mejore a costa de hacerlo acc^-, mientras que, por el contrario, si $R_t(\mathbf{x}) = -1$, serán los ejemplos *quasi-ruidosos* positivos los que tomarán más relevancia en el proceso de boosting, proporcionando un aumento del acc^+ que, en principio, debido al desequilibrio de clases, no tiene porqué implicar necesariamente una mejora del *acc* global. Con estas ideas, el compromiso entre ambas situaciones extremas se alcanza con una tercera alternativa, consistente en emplear para los patrones *quasi-ruidosos* negativos el valor $R(\mathbf{x}) = 0$.

Respectivamente, se denominan a las 3 opciones anteriores: PPBoost negativo, positivo y equilibrado. Sus correspondientes valores de $R(\mathbf{x})$ para los distintas categorías de patrones se resumen en la tabla 5.2.1. Los resultados empíricos de todas ellas se dan en el capítulo siguiente.

Categoría $\mathbf{x}$	PP AdaBoost	PPBoost negativo	PPBoost positivo	PPBoost equilibrado
$\mathcal{T}_R$	1	1	1	1
$\mathcal{T}_N$	1	-1	-1	-1
$\mathcal{W}_{B-}$	1	1	-1	0
Otros $\mathcal{T}_B$	1	1	1	1

Tabla 5.2.1: Resumen de los valores $R(\mathbf{x})$ de PPBoost para patrones de entrenamiento $\mathbf{x}$.

Capítulo 6

Experimentos

Descritos el escenario y las dificultades que plantean los problemas de clasificación desequilibrados, y expuestas las dos propuestas de esta investigación para abordarlos, **PPTSS** y **PPBoost**, ambas basadas en el uso de Perceptrones Paralelos, llega el momento, en este capítulo, de evaluarlas y mostrar los experimentos empíricos realizados.

En este contexto, no sólo se darán los resultados numéricos de sendos algoritmos, sino que se contrastarán con los obtenidos por los bien conocidos y potentes Perceptrones Multicapa (PMC). De este modo, el objetivo no es sólo mostrar la mejora conseguida por las dos propuestas en las capacidades predictivas del PP. Se persigue además que estas últimas sean superiores (o al menos comparables) a las ofrecidas por métodos de clasificación tan robustos como son los PMC. Sin entrar en temas de rendimiento, donde el PP posee notables ventajas respecto al PMC al desarrollar un entrenamiento mucho más rápido, la comparación entre ambos, como ya se introdujo en la *sección 2.5*, se hará únicamente a través de la media geométrica $g = \sqrt{acc^+ \cdot acc^-}$. Con ella se busca un compromiso entre los porcentajes de aciertos en las dos clases, minoritaria y mayoritaria, y no una mejora del porcentaje de aciertos de clasificación global, que pudiese ocultar el hecho de estar omitiendo los ejemplos positivos durante el aprendizaje y la posterior clasificación.

En las primeras dos secciones se describen los conjuntos de datos y la metodología empleados en los experimentos, para que en las dos posteriores directamente se muestren y se expliquen en forma de tablas los resultados obtenidos. La elección de las arquitecturas y de los parámetros iniciales de los PP no se tratan en profundidad en el capítulo, pues escapan al propósito de este texto. Sin embargo, comentar que en la investigación se llevó a cabo una fase exhaustiva de pruebas previa a los experimentos finales en busca de las configuraciones utilizadas. Como se comprobará más adelante, y coincidiendo con las observaciones de [1], el uso de arquitecturas relativamente pequeñas proporcionará un poder descriptivo de los conjuntos de datos muy alto.

6.1 Conjuntos de datos

En la evaluación empírica de los algoritmos PPTSS y PPBoost se han empleado 7 conjuntos de datos del repositorio para aprendizaje automático de la Universidad de California, Irvine (UCI) [6]. La elección de estos está motivada por el uso de problemas de clasificación binaria con diferentes grados de desequilibrio entre clases, y de los que se puedan comparar resultados presentados en la literatura [1, 2, 29]. A continuación se da una breve descripción de todos ellos y se incluye la tabla 6.1.1 que resume sus características.

- **ionosphere**: conjunto de datos que contiene 351 ejemplos de señales radar provenientes de la Ionosfera, recopiladas en Labrador (Canadá) en 1989, cada una ellas con 34 atributos continuos e indicando la existencia o ausencia de evidencias sobre cierto tipo de estructuras en la citada capa atmosférica. El 35.9% son ejemplos minoritarios.

- **diabetes**: problema "Pima Indian Diabetes" de 768 patrones con 8 atributos reales correspondientes a criterios definidos por la Organización Mundial de la Salud para describir pacientes diabéticos. La muestra, obtenida en 1988 en Arizona (USA) sobre un conjunto de mujeres de origen indio, posee el 34.9% de ejemplos minoritarios.

- **cancer**: conjunto "Winconsin breast-cancer" compuesto de 699 ejemplos de datos médicos sobre casos de cancer de pecho recopilados en hospitales de Winconsin (USA) desde 1989 a 1992. Cada uno de ellos con 9 atributos de valores numéricos discretos entre 1 y 10. El 34.5% pertenecen a la clase minoritaria.

- **vehicle**: colección de 846 patrones, recopilados en Glasgow (Escocia) en 1986, con 18 atributos continuos descriptivos de la silueta de vehículos pertenecientes a 4 marcas diferentes. Seleccionando la segunda de ellas para su clasificación contra el resto [29], el problema contiene el 25.7% de ejemplos minoritarios.

- **glass**: conjunto de 214 muestras con 9 atributos continuos sobre 7 tipos de cristal, motivadas por investigaciones criminológicas y recopiladas en Berkshire (Gran Bretaña) en 1987. Seleccionando para clasificación el último tipo de cristal [29], el porcentaje de elementos en la clase minoritaria es del 13.6%.

- **vowel**: problema de reconocimiento de vocales mayúsculas y minúsculas, con 990 patrones de 10 atributos reales recopilados en 1993. De las 10 clases existentes, se considera para clasificación la primera [29], obteniendo un 9.1% de ejemplos positivos.

- **thyroid**: recopilación de 6480 ejemplos de 8 atributos continuos con información médica sobre casos de hipotiroides. Las 3 clases iniciales se reúnen en sólo 2: pacientes hipotiroideos y sanos, obteniendo para los primeros un 7.4% del total.

Problema	# ejemplos	% positivos	# atr. continuos	# atr. discretos
ionosphere	351	35.9	34	0
diabetes	768	34.9	8	0
cancer	699	34.5	0	9
vehicle	846	25.7	18	0
glass	214	13.6	9	0
vowel	990	9.1	10	0
thyroid	6480	7.4	8	0

Tabla 6.1.1: Descripción de los conjuntos de datos empleados.

6.2 Metodología

A lo largo de las secciones restantes y en todos los casos, los resultados numéricos mostrados se han obtenido mediante un proceso de validación cruzada (*cross validation*); en concreto *10-times 10-fold cross validation*. Esto es, para cada conjunto de datos, utilizando todas los patrones del mismo, se crean de manera aleatoria 10 subconjuntos disjuntos de tamaño similar. De ellos, 9 se emplean para entrenar y 1 para evaluar un clasificador. En este esquema, se promedian los resultados alcanzados tras haber tomado cada uno de los 10 subconjuntos como conjunto de test (y el resto como entrenamiento), para dar finalmente los resultados de una iteración de evaluación. Todo el procedimiento anterior se repite y se promedia a su vez 10 veces. Además, como se ha trabajado con muestras desequilibradas, se lleva a cabo una selección estratificada de los subconjuntos, obligando a que en cada uno de ellos exista la misma proporción no nula de ejemplos minoritarios. Las tablas de resultados, por tanto, incluyen los valores promediados de las 10 iteraciones de validación cruzada y las desviaciones típicas (entre paréntesis) correspondientes a cada uno de ellos.

Los PP construidos poseían 3 perceptrones estándar, margen inicial $\gamma_0 = 0.05$ y tasa de aprendizaje $\eta_0 = 10^{-2}$ (10^{-3} para el problema thyroid). Fueron entrenados en modo *batch* (*véase sección 3.4*) durante 250 épocas, sin considerar el error de entrenamiento como criterio de parada del aprendizaje, pero observando un comportamiento de descenso del mismo suficiente para todos los casos. Las estrategias de actualización dinámica de γ y η son las descritas en la *sección 3.4*. Por su parte, los PMC, de una única capa oculta con 3 unidades y con tasa de aprendizaje $\eta = 10^{-3}$, fueron entrenados mediante descenso del error por gradiente aleatorio durante 2000 épocas *batch*. Aunque con estas características se pretendía equiparar las arquitecturas y prestaciones de ambos tipos de máquinas, se debe destacar la gran diferencia existente en los tiempos necesarios para el entrenamiento, en los

que el PP, de manera aplastante, aventaja al PMC. En aquellos experimentos que involucran procesos de boosting, tanto con PP como con PMC, el número de clasificadores generados por conjunto se estableció en 10, valor que se comprobó suficiente para la convergencia de error de entrenamiento del multiclasificador.

6.3 Resultados numéricos de PPTSS

La tabla 6.3.1 reúne los valores finales de g para el PMC, el PP y las versiones de PPTSS manteniendo y eliminando los patrones $\mathcal{W}_{B-}$. De ella se deben destacar dos hechos:

- El algoritmo PPTSS supone una mejora notable en los resultados del PP para todos los casos, excepto para el problema *cancer*, en el que empeora muy ligeramente.
- Los resultados obtenidos pot PPTSS sin $\mathcal{W}_{B-}$ son comparables a los del PMC, especialmente cuando las muestras están muy desequilibradas (*glass*, *thyroid*). En *vowel* PPTSS no alcanza a PMC, pero nótese que PP obtiene una g muy baja para ese problema.

Problema	PMC	PP	PPTSS con $\mathcal{W}_{B-}$	PPTSS sin $\mathcal{W}_{B-}$
ionosphere	80.8 (0.46)	76.9 (0.87)	77.6 (0.91)	**82.3** (0.68)
diabetes	**71.7** (0.89)	69.9 (1.39)	68.7 (0.74)	71.2 (0.83)
cancer	96.1 (0.43)	**96.8** (0.29)	96.5 (0.25)	96.6 (0.26)
vehicle	**75.7** (1.59)	64.1 (1.97)	69.6 (1.32)	72.5 (1.36)
glass	91.8 (0.45)	90.4 (1.65)	91.2 (1.13)	**91.6** (0.91)
vowel	**97.1** (1.32)	89.3 (1.06)	92.9 (0.94)	93.3 (1.69)
thyroid	95.8 (0.25)	68.1 (2.28)	73.9 (0.33)	**97.2** (0.25)

Tabla 6.3.1: Valores finales de g para PPTSS manteniendo (cuarta columna) y eliminando (quinta columna) los ejemplos $\mathcal{W}_{B-}$. Mejores resultados en negrita.

La mejora obtenida en los valores de g por PPTSS eliminando los $\mathcal{W}_{B-}$ respecto a PP se ve acompañada de una importante reducción de los conjuntos de entrenamiento, así como de un número necesario de iteraciones de filtrado muy pequeño. La tabla 6.3.2 resume estos hechos. El caso más destacable es el del problema *thyroid* donde se aumenta de una g inicial de 68.1 a una final de 97.2, reduciendo en promedio el conjunto de entrenamiento de 6480 patrones a tan sólo 64, y empleando unas 5 iteraciones.

Problema	g inicial	g final	# ej. inicial	# ej. final	# iters.
ionosphere	76.9	82.3	315	241	2.13
diabetes	69.9	71.2	691	631	0.88
cancer	96.8	96.6	629	174	1.99
vehicle	64.1	72.5	762	284	2.89
glass	90.4	91.6	193	163	0.59
vowel	89.3	93.3	891	418	1.62
thyroid	68.1	97.2	6480	64	4.81

Tabla 6.3.2: Comparación entre los valores iniciales y finales de g, el número de patrones de entrenamiento, y el número de iteraciones para PPTSS eliminando los ejemplos $\mathcal{W}_{B-}$.

Como observación adicional, se puede comprobar con la tabla 6.3.3 que PPTSS mejora los valores de g acosta de centrarse en los ejemplos minoritarios y aumentar acc^+, pero que no se olvida de los ejemplos mayoritarios, en el sentido de que los valores de acc^- se mantienen o bajan relativamente poco. De ahí que los valores de acc global se mantengan o incluso mejoren en general.

Problema	acc inicial	acc^+ inicial	acc^- inicial	acc final	acc^+ final	acc^- final
ionosphere	82.143	64.051	92.367	86.171	71.803	94.255
diabetes	75.039	57.926	84.369	75.158	61.373	82.665
cancer	96.870	96.630	97.009	96.420	97.155	96.062
vehicle	78.119	51.414	87.437	79.512	61.139	85.879
glass	95.048	84.500	96.795	95.842	86.000	97.459
vowel	97.273	80.667	98.933	97.838	88.111	98.811
thyroid	95.499	46.708	99.405	98.493	95.625	98.722

Tabla 6.3.3: Comparación entre los valores iniciales y finales de acc, acc^- y acc^+ para PPTSS eliminando los ejemplos $\mathcal{W}_{B-}$.

6.4 Resultados numéricos de PPBoost

Para destacar el valor de los resultados obtenidos con las diferentes versiones de PPBoost, se hace necesario en primer lugar poner de manifiesto la desventaja con la que se parte al aplicar AdaBoost con PP como aprendiz débil con respecto a hacerlo con PMC. La tabla 6.4.1 muestra la supremacía tanto en los valores de *acc* como de g en la combinación PMC-AdaBoost.

	PMC-AdaBoost				PP-AdaBoost			
Problema	acc	acc^+	acc^-	g	acc	acc^+	acc^-	g
ionosphere	88.40 (0.32)	73.53	96.71	84.3 (1.89)	85.49 (1.51)	68.91	94.85	80.8 (2.02)
diabetes	73.80 (0.25)	60.63	80.84	70.0 (1.75)	73.16 (0.97)	57.99	81.42	68.7 (1.06)
cancer	95.82 (0.13)	94.19	96.68	95.4 (0.55)	95.64 (0.25)	93.32	96.86	95.1 (0.52)
vehicle	83.16 (0.10)	65.35	89.30	76.4 (0.60)	79.04 (1.15)	55.33	87.30	69.5 (1.05)
glass	95.82 (0.22)	88.00	97.01	92.4 (1.52)	95.71 (1.06)	86.83	97.13	91.8 (1.84)
vowel	99.66 (0.01)	98.22	99.80	99.0 (0.12)	99.46 (0.28)	96.56	99.76	98.1 (0.80)
thyroid	98.73 (0.05)	91.30	99.32	95.2 (0.32)	97.33 (0.19)	80.14	98.85	89.0 (1.20)

Tabla 6.4.1: Valores finales de acc, acc^+, acc^- y g para AdaBoost aplicado sobre PMC y PP.

A partir de los datos presentados, las mejoras en los valores de *acc* alcanzadas por las estrategias PPBoost negativo, positivo y equilibrado quedan reflejadas en la tabla 6.4.2. Se puede observar que en todos los casos, excepto en el problema *ionosphere*, que es el menos desequilibrado, se obtienen incrementos de los valores de *acc* en relación a los ofrecidos por PP-AdaBoost. Además, y de manera individual para cada tipo de boosting, se pueden destacar los siguientes tres aspectos:

- La reducción de los pesos de los patrones redundantes y ruidosos hace que **PPBoost negativo** tenga los mejores resultados en el porcentaje de aciertos de clasificación global (debido a los altos valores de acc^-), pero no le proporcionará tan buenos valores para g, ya que los acc^+ son muy bajos.

- El hecho de reducir también los pesos de los patrones *quasiruidosos* negativos en **PPBoost positivo**, proporciona una mejora muy importante en el acc^+, que se ve contrastada con la bajada considerable de los valores de acc^-, y por tanto de los de acc y g.

- Al no aumentar o reducir (sin considerar normalización) el peso de los patrones *quasiruidosos* negativos, **PPBoost equilibrado** llega a un compromiso entre buenos valores de acc^+ y acc^-, provocando, como se verá a continuación, resultados óptimos de g en los problemas más desequilibrados.

	PPBoost neg.			PPBoost pos.			PPBoost equ.		
Problema	acc	acc^+	acc^-	acc	acc^+	acc^-	acc	acc^+	acc^-
ionosphere	**85.09** (1.21)	67.45	95.09	65.97 (1.40)	89.85	52.50	84.97 (1.20)	71.44	92.65
diabetes	**73.29** (0.88)	59.07	81.04	57.76 (0.96)	94.85	37.56	72.08 (0.88)	73.18	71.49
cancer	96.03 (0.18)	94.27	97.02	**96.39** (0.22)	99.15	94.91	96.32 (0.16)	96.07	96.50
vehicle	**79.68** (0.97)	55.04	88.27	72.09 (0.46)	95.52	63.93	78.61 (0.75)	72.41	80.78
glass	96.09 (0.78)	87.33	97.51	94.76 (0.39)	90.17	95.53	**96.33** (0.88)	89.33	97.46
vowel	**99.52** (0.35)	97.44	99.73	95.56 (0.13)	99.89	95.12	99.39 (0.24)	98.33	99.50
thyroid	**97.73** (0.32)	83.42	98.88	94.66 (0.21)	99.56	94.27	97.66 (0.27)	96.60	97.74

Tabla 6.4.2: Valores finales de acc, acc^+ y acc^- obtenidos por PPBoost negativo, positivo y equilibrado. Mejores resultados de acc en negrita.

La tabla 6.4.3 compara finalmente los valores de g en AdaBoost y en las tres propuestas PPBoost. En los problemas más desequilibrados (*glass*, *vowel*, *thyroid*), PPBoost equilibrado proporciona los mejores resultados. Es superado por PPBoost positivo en *cancer* y *vehicle*, en los que se obtenía un acc^+ muy alto, y por AdaBoost-PMC en *ionosphere*, el problema menos desequilibrado.

Problema	PMC AdaBoost	PP AdaBoost	PPBoost negativo	PPBoost positivo	PPBoost equilibrado
ionosphere	**84.3**	80.8	80.1	68.7	81.4
	(1.89)	(2.02)	(1.52)	(1.85)	(2.01)
diabetes	70.0	68.7	69.2	59.7	**72.3**
	(1.75)	(1.06)	(0.92)	(0.92)	(1.15)
cancer	95.4	95.1	95.6	**97.0**	96.3
	(0.55)	(0.52)	(0.49)	(0.25)	(0.38)
vehicle	76.4	69.5	69.7	**78.1**	76.5
	(0.60)	(1.05)	(0.99)	(0.67)	(0.88)
glass	92.4	91.8	92.3	92.8	**93.3**
	(1.52)	(1.84)	(1.38)	(1.32)	(1.96)
vowel	99.0	98.1	98.1	97.5	**98.9**
	(0.12)	(0.80)	(0.66)	(0.25)	(0.48)
thyroid	95.2	89.0	90.8	96.9	**97.2**
	(0.32)	(1.20)	(0.50)	(0.18)	(0.35)

Tabla 6.4.3: Valores finales de g obtenidos por PPBoost negativo, positivo y equilibrado, comparados con los de AdaBoost sobre PMC y PP. Mejores resultados en negrita.

En cualquier caso, PPBoost equilibrado ofrece mejores resultados que PP-AdaBoost en términos de un compromiso entre los porcentajes de aciertos alcanzados en las clases minoritaria y mayoritaria; demostrando de este modo la validez de la selección de patrones de entrenamiento propuesta, y proporcionando nuevas ideas para la identificación de patrones cercanos a la fronteras de separación a través del uso de márgenes de activación.

Por otra parte, resaltar que los resultados de PPBoost son comparables y en algunos casos mejores a los de PMC-AdaBoost. A pesar de la simplicidad de los PP, al haber elegido un número relativamente pequeño ($H = 3$) de perceptrones, la estrategia de selección ha mostrado una importante capacidad descriptiva de los conjuntos de datos empleados. Si a esto se le añade el hecho de que el entrenamiento del PP es mucho más rápido que el del PMC, se concluye que el primero puede ser un buen candidato para problemas desequilibrados de alta dimensionalidad o con un gran número de patrones.

Junto a esta última aplicación, otras posibles líneas de investigación futura derivadas del presente trabajo podrían ser: la combinación de las propuestas con técnicas de sobremuestreo, el estudio de otros algoritmos de combinación de modelos para la selección de patrones de entrenamiento representativos, o la propia detección y eliminación de ejemplos espurios (*outliers*) en problemas con ruido malicioso.

Capítulo 7

Conclusiones

En este trabajo se han introducido los problemas de clasificación desequilibrados y se han descrito las dos principales estrategias empleadas para abordarlos: submuestreo de ejemplos mayoritarios y sobremuestreo de ejemplos minoritarios. Teniendo presentes los inconvenientes de ambos enfoques, se ha propuesto el uso de estrategias de selección de patrones cercanos a las fronteras de separación de clases como técnica alternativa. Para ello, se ha planteado la eliminación, parcial o total, de la influencia de: (1) aquellos ejemplos *redundantes*, que están bien representados por otros en los conjuntos de entrenamiento, y que por ello no aportan información relevante al aprendizaje, y (2) aquellos ejemplos *ruidosos*, con etiquetas claramente opuestas (y tal vez incorrectas) a lo que los discriminantes predicen, que pueden entorpecer el proceso de construcción de los clasificadores.

En este contexto, el problema surge entonces en la definición de dichas categorías de patrones. Para resolverlo se ha sugerido de manera innovadora el uso de Perceptrones Paralelos y de sus márgenes respecto a la frontera de separación, y se han estudiado dos propuestas de selección de conjuntos de entrenamiento para muestras desequilibradas: (1) PPTSS, que elimina iterativamente los ejemplos identificados como redundantes y ruidosos, y (2) PPBoost, adaptación del bien conocido algoritmo AdaBoost, que en vez de eliminarlos, les disminuye progresivamente la probabilidad de ser elegidos para el aprendizaje.

Los experimentos realizados muestran muy buenos resultados no sólo en la precisión global de la clasificación, sino también en las correspondientes a las clases minoritaria y mayoritaria, hecho fundamental al trabajar con muestras desequilibradas. El alcance, y mejora en los casos considerados más extremos, de los resultados obtenidos por los robustos Perceptrones Multicapa, junto con la mayor rapidez de aprendizaje, hacen de las propuestas expuestas muy buenas candidatas para tratar problemas de clasificación desequilibrados de alta dimensionalidad o para motivar estrategias más generales de selección de patrones de entrenamiento.

Bibliografía

[1] Auer, P., Burgsteiner, H. M. and Maass, W. *The p-Delta Rule for Parallel Perceptrons*. Submitted for publication, 2001. 1.2, 3, 3.1, 3.3, 3.4, 3.4, 3.4, 6, 6.1

[2] Auer, P., Burgsteiner, H. M. and Maass, W. *Reducing Communication for Distributed Learning in Neural Networks*. Proceedings of ICANN'2002, Lecture Notes in Computer Science 2415: pages 123–128, 2002. 1.2, 3, 3.4, 6.1

[3] Barandela, R., Sánchez, J. S., García, V. and Ferri, F. J. *Learning from Imbalanced sets through resampling and weighting*. Lecture Notes in Computer Science 2652: pages 80–88, 2003. 2, 2.2

[4] Barandela, R., Valdovinos, R. M., Salvador Sánchez, J. and Ferri, F. J. *The Imbalanced Training Sample Problem: Under or over Sampling?*. In the 5th International Workshop on Statistical Techniques in Pattern Recognition, Lisbon, Portugal, 2004. 2.2

[5] Bishop, C. M. *Neural Networks for Pattern Recognition*. Oxford University Press, Oxford, England, 1995. 1.3, 2.3, 3

[6] Blake, C. L. and Merz, C. J. *UCI repository of machine learning databases*. http://www.ics.uci.edu/˜mlearn/MLRepository.html. Irvine, CA: University of California, Department of Information and Computer Science, 1998. 4.3, 6.1

[7] Breiman, L. *Bagging Predictors*. Machine Learning, vol. 24, pages 123-146, 1996. 4.1

[8] Burges, C. *A Tutorial on Support Vector Machines for Pattern Recognition*. Data Mining and Knlowledge Discovery, vol. 2, no. 2, 1998. 2

[9] Cantador, I. and Dorronsoro, J. R. *Parallel Perceptrons and Training Set Selection for Imbalanced Classification Problems*. Proceedings of the Learning'04 International Conference, Elche, Spain, 2004. 2.3, 5.1

[10] Cantador, I. and Dorronsoro, J. R. *Parallel Perceptrons, Activation Margins and Imbalanced Training Set Pruning*. 2nd Iberian Conference on Pattern Recognition and Image Analysis (IbPRIA'2005), Estoril, Portugal, 2005. 2.3, 5.1, 5.1

[11] Cantador, I. and Dorronsoro, J. R. *Balanced Boosting with Parallel Perceptrons*. 8th International Work-Conference on Artificial Neural Networks (IWANN'2005), Vilanova i la Geltrú (Barcelona), Spain, 2005. 2.3, 5.2

[12] Chan, P. and Stolfo, S. *Toward scalable learning with non-uniform class and cost distributions: a case study in credit card fraud detection*. In Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining, pages 164–168, Menlo Park, CA: AAAI Press, 1998. 2.2

[13] Chawla, N., Bowyer, K., Hall, L. and Kegelmeyer, W. P. *SMOTE: Synthetic Minority Over-sampling TEchnique*. In International Conference on Knowledge Based Computer Systems, 2000. 2.2

[14] Dietterich, T. G and Bakiri, G. *Solving multiclass learning problems via error-correcting output codes*. Journal of Artificial Intelligence Research, vol. 2, pages 263–286, 1995. 4.1

[15] Dietterich, T. G. *Ensemble Methods in Machine Learning*. In Proceedings of the 1st International Workshop on Multiple Classifier Systems, pages 1–15. Lectures Notes in Computer Science, 2000. 4.1, 4.1, 4.1

[16] Dietterich, T. G. *An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting and Randomization*. Machine Learning, vol. 40(2): pages 139–158, 2000. 5.2

[17] Dorronsoro, J. R., Ginel, F., Sánchez, C. and Santa Cruz, C. *Neural Fraud Detection in Credit Card Operations*. IEEE Transactions on Neural Networks, vol. 8, pages 827–834, 1997. 2

[18] Duch, W. *Similarity based methods: a general framework for classification, approximation and association*. Control and Cybernetics, vol. 29(4): pages 937–968, 2000. 2.3

[19] Duch, W. *Support Vector Neural Training*. Submitted for publication, 2004. 2, 2.3, 5.1

[20] Duda, R. O., Hart, P. E. and Stork, D. G. *Pattern classification*. 2nd ed. New York: Wiley, 2001. 1.3, 2.3, 3

[21] Ezawa, K. J., Singh, M. and Norton, S. W. *Learning Goal Oriented Bayesian Networks for Telecommunications Management*. Proceedings of the International Conference on Machine Learning, ICML'96 (pages 139–147), Bari, Italy, 1996. 2

[22] Fawcett, T. and Provost, F. *Adaptative fraud detection*. Data Mining and Knowledge Discovery, 1: pages 291–316, 1996. 2

[23] Freund, Y. *Boosting a weak learning algorithm by majority*. Information and Computation, 121: pages 256–285, 1995. 1.2, 4.1

[24] Freund, Y. and Schapire, R. E. *Experiments with a new Boosting algorithm*. In Proceedings of the 13th International Conference on Machine Learning, ICML'96, pages 148–156, 1996. 1.2, 4.1, 5.2

[25] Freund, Y. and Schapire, R. E. *A Short Introduction to Boosting*. Journal of Japanese Society for Artificial Intelligence, 14(5): 771–780, 1999. 1.2

[26] Ho, T. K. *C4.5 Decision Forests*. In Proceedings of the International Conference on Pattern Recognition, pages 545–549, 1998. 4.1

[27] Joachims, T. *Learning to Classify Text using Support Vector Machines: Methods, Theory and Algorithims*. Kluwer Academic Publisher, 2002. 2

[28] Khan, J., Wei, J. S., Ringner, M., Saal, L. H., Ladanyi, M., Westermann, F., Berthold, F., Schwab, M., Antonescu, C. R., Peterson, C. and Meltzer, P. S. *Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural netwoks*. Nature Medicine, vol. 7, no. 6, pages 673–679, 2001. 2

[29] Kubat, M. and Matwin, S. *Addressing the Curse of Imbalanced Training Sets: One-Sided Selection*. In Proceedings of the 14th International Conference on Machine Learning, ICML'97, pages 179–186, Nashville, TN, U.S.A., 1997. 2, 2.2, 5.1, 6.1

[30] Kubat, M., Holte, R. and Matwin, S. *Machine Learning for the Detection of Oil Spills in Satellite Radar Images*. Machine Learning vol. 30(2-3): pages 195-215, 1998. 2, 2.5

[31] Lewis, D. D. and Gale, W. A. *A sequential algorithm for training text classifiers*. In Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, pages 3-12, 1994. 2.5

[32] Maclin, R. and Opitz, D. *An Empirical Evaluation of Bagging and Boosting*. In Proceedings of the 14th National Conference on Artificial Intelligence, pages 546–551, 1997. 5.2

[33] Meir, R. and Rätsch, G. *An Introduction to Boosting and Leveraging.* Advanced Lectures on Machine Learning, pages 119–184, 2003. 1.2, 4.3

[34] Pazzani, M., Merz, C., Murphy, P., Ali, K., Hume, T. and Brunk, C. *Reducing misclassification costs.* In Proceedings of the 11th International Conference on Machine Learning, pages 217–225, 1994. 2.4

[35] Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. *Numerical Recipies in C: The art of Scientific Programming.* Cambridge University Press, Cambridge, England, 1988.

[36] Quinlan, J. R. *Bagging, Boosting and C4.5.* In Proceedings of 30th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence, pages 725–730, 1996. 4.1

[37] Ritter, G. I., Woodruff, H. B., Lowry, S. R. and Isenhour, T. L. *An Algorithm for Selective Nearest Neighbor Decision Rule.* IEEE Transactions on Information Theory, 21(6), pages 665–669, 1975. 2

[38] Röbel, A. *The Dynamic Pattern Selection Algorithm: Effective Training and Controlled Generalization of Backpropagation Neural Networks.* Technical Report, Institute für Angewandte Informatik, Technische Universität Berlin, pages 497–500, 1994. 2.3

[39] Schapire, R. E., Freund, Y., Bartlett, P. and Lee, W. S. *Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods.* The Annals of Statistics, 26(5): 1651–1686, 1998. (document), 1.2, , 4.3, 4.3.3, 4.3, 4.3, 4.3.4, 4.3, 4.3

[40] Swets, J. *Measuring the Accuracy of Diagnostic Systems.* Science, 240: pages 1285–1293, 1988. 2.5

[41] Tong, S. and Koller, D. *Support Vector Machine Active Learning with Applications to Text Classification.* Proceedings of ICML'00, 17th International Conference on Machine Learning, 2000. 2

[42] Verbaeten, S. and Assche, A. V. *Ensemble methods for noise elimination in classification problems.* In 4th International Workshop on Multiple Classifier Systems: pages 317-325, 2003. 5.2

[43] Weiss, G. and Provost, F. *The Effect of Class Distribution on Classifier Learning: An Empirical Study.* Technical Report ML-TR-44, Department of Computer Science, Rutgers University, 2001. 1, 2.1, 2.4

[44] Wilson, D. L. *Asymptotic properties of nearest neighbor rules using editing data sets.* IEEE Transactions on Systems, Man and Cybernetics, 2: pages 408–421, 1972. 2, 2.2

[45] Woods, K., Doss, C., Bowyer, K. W., Solka, J., Priebe, C. and Kegelmeyer, W. P. *Comparative evaluation of pattern recognition techniques for detection of micro–classification in mammography.* International Journal of Pattern Recognition and Artificial Intelligence, vol. 7: pages 1417–1436, 1993. 2

[46] Zhang B. *Accelerated Learning by Active Example Selection.* International Journal of Neural Systems, vol. 5(1): pages 67–75, 1994. 2.3