Universidad de Valladolid

Facultad de Ciencias

Trabajo Fin de Grado

Grado en Matemáticas

Gradient Boosting (Potenciación del Gradiente) en Aprendizaje Estadístico

Autor: Bernardo Martínez Celda

Tutor: Pedro C. Álvarez Esteban

Agradecimientos

A mi tutor, el Doctor Pedro César Álvarez Esteban, por compartir su tiempo conmigo ayudándome a planificar este trabajo.

A mis profesores de bachillerato, y en especial a Pedro, Roberto y María Jesús; por contagiarme su pasión por las matemáticas.

A mis compañeros del Programa de Estudios Conjunto, Mario, Pablo, Elisabet, Gabri, Elsa, Alfonso, Merino, José Antonio, y Andrea; por las momentos que hemos compartido, por su inestimable ayuda, y porque son como una familia para mí.

A mi hermana Mar, a mi pareja Laura, y a mis amigos, por haberme apoyado en todo momento y por sacar lo mejor de mí.

A mis padres, Bernardo y Marifé, por preocuparse por mí, por haber sabido aconsejarme sabiamente, y por haberme inculcado desde pequeño los valores de la constancia y la responsabilidad.

Gradient boosting (potenciación del gradiente) en aprendizaje estadístico

Bernardo Martínez Celda

Resumen

Este trabajo se centra en el estudio del boosting, un método de ensamble diseñado para combinar clasificadores débiles y formar un buen clasificador. En primer lugar se realiza una breve introducción a los problemas de aprendizaje estadístico y al método boosting. También se presenta AdaBoost, uno de los algoritmos de boosting más estudiados hasta el momento, y se prueban resultados teóricos que permiten acotar su error de generalización. Se utiliza el concepto de margen para explicar el comportamiento del error de generalización de AdaBoost, y justificar la falta de sobreajuste en ciertas ocasiones. Posteriormente se demuestra que AdaBoost es un algoritmo que optimiza una determinada función de coste mediante la búsqueda de su gradiente descendente. Por último se introducen los árboles de decisión como candidatos a ser combinados mediante algún algoritmo de boosting. La motivación de este trabajo es doble. Por un lado se pretende entender cómo funciona el boosting y a qué se deben sus cualidades, ya que es un método de ensamble que se encuentra en auge. Y por otro lado, poder realizar un estudio de caso donde se pretende predecir la configuración de pistas de un aeropuerto mediante árboles potenciados. Este estudio de caso se ha llevado a cabo en RStudio con la ayuda del paquete `xgboost`. En este trabajo se muestran las principales herramientas de este paquete y se realiza una búsqueda de parámetros óptimos para el entrenamiento de modelos combinados con `xgboost`.

Índice general

Índice de figuras

Capítulo 1

Introducción al boosting

La creación del primer ordenador a mediados del siglo XX supondría la llegada de la Revolución Digital, desde entonces tanto la ciencia como la industria han avanzado a pasos agigantados, y esta evolución trae consigo algunos desafíos que suponen un reto en el campo de la Estadística. Con la llegada de la Era de la información, montones de datos se generan en diferentes áreas, y parte del trabajo de los estadísticos consiste en darle sentido a esta vasta cantidad de información. Con este volumen de datos es posible extraer patrones y tendencias para entender "lo que dicen los datos". Esto se conoce en inglés como *learning from data*, que traducido al español es *aprendizaje a partir de los datos*.

El aprendizaje a partir de los datos tiene un papel fundamental en diversas áreas de la ciencia, finanzas y la industria. Algunos ejemplos de problemas de aprendizaje son:

- Predecir si un paciente que ha sido hospitalizado debido a un ataque al corazón sufrirá un segundo ataque. Esta predicción podría realizarse teniendo en cuenta datos demográficos, dietas, medidas clínicas, etc.
- Predecir el precio de un producto en stock dentro de seis meses, basándose en registros de venta previos.
- Identificar el número de una tarjeta bancaria a partir de una imagen digitalizada.

En un problema de aprendizaje típico se tiene una variable cuyo valor se quiere predecir según ciertos parámetros de entrada, dicha variable puede ser cuantitativa (como el precio del producto en stock) o categórica (como "padecerá un segundo ataque al corazón" / "no padecerá un segundo ataque al corazón"). Asimismo, se cuenta con una base de datos de muestras conocidas y a partir de la misma se construye un clasificador. El clasificador es una herramienta que se encarga de predecir el valor de la variable bajo estudio en muestras desconocidas. Un *buen* clasificador es aquel que predice con un alto

grado de exactitud dicha variable.

En ocasiones construir un buen clasificador puede ser una ardua tarea. Sin embargo, es posible construir reglas sencillas que realicen predicciones con menor precisión pero sin llegar a caer en la trivialidad. Ahora bien, ¿tiene algún sentido construir estos clasificadores *débiles*, moderadamente imprecisos? ¿es posible combinar estos clasificadores para formar una clasificador fuerte? En este trabajo se pretende dar respuesta a estas preguntas explicando el concepto de *boosting*. El fundamento del boosting es resolver complejos problemas de aprendizaje automático combinando estos clasificadores débiles e imprecisos, pero escogidos de forma muy cuidadosa, para formar un *comité* cuyas predicciones sean muy acertadas. Es decir, un algoritmo de boosting busca formar un comité inteligente de miembros inexpertos.

Este capítulo se ha desarrollado siguiendo la teoría presente en los capítulos 1 y 10 de Hastie *et al.* (2009), el capítulo 1 de Schapire y Freund (2013), y los capítulos 1 y 10 de Shalev-Shwartz y Ben-David (2014).

1.1. Problemas de Clasificación y Aprendizaje Automático

Este trabajo se centra principalmente en problemas de *clasificación* cuya finalidad es clasificar objetos por categorías. Por ejemplo, un sistema OCR (del inglés *Optical Character Recognition*) debe clasificar imágenes de letras en las categorías A, B, C, etc. También podemos encontrar otro ejemplo de problemas de clasificación en campos como la medicina, en los que el objetivo es determinar si el paciente de cáncer tiene un tumor maligno o benigno.

En el contexto de los problemas de clasificación, los métodos de aprendizaje automático tratan de aprender a clasificar muestras desconocidas basándose en el estudio detallado de muestras que habían sido previamente etiquetadas con su correspondiente categoría.

Los objetos a clasificar reciben el nombre de *individuos* y se denotan por x. En el ejemplo del sistema OCR, los individuos son las imágenes de las letras. En el ejemplo del diagnóstico médico los individuos son los pacientes y su sintomatología. El espacio de todos los individuos posibles se conoce como *dominio* X. Una *muestra etiquetada*, o *muestra patrón*, es un par (x, y) donde y es una *etiqueta* que indica la categoría que le corresponde al individuo x. El espacio de las etiquetas se representa mediante Y. En ocasiones, siempre y cuando el contexto no dé lugar a ambigüedades, también se emplea

el término muestra (no etiquetada) para hacer referencia a los individuos que no están etiquetados.

Durante el entrenamiento, un *algoritmo de aprendizaje* A recibe como parámetro de entrada un *conjunto de entrenamiento* S, éste consiste en un conjunto de muestras etiquetadas llamadas *muestras de entrenamiento*: $S = \{(x_1, y_1), ..., (x_m, y_m)\}$. La salida del algoritmo es una regla predictiva llamada *clasificador* o *hipótesis* $h \in \mathcal{H}$, donde $\mathcal{H}$ es el espacio de funciones $f : X \to Y$ conocido como *espacio de hipótesis*. Para hacerse una idea de en qué consiste un clasificador, uno puede imaginarlo como un programa de ordenador que toma como parámetro de entrada un individuo nuevo, cuya etiqueta es desconocida para el programa, predice la etiqueta que le corresponde a este individuo y proporciona como respuesta dicha etiqueta. En términos matemáticos, un clasificador es una función cuyo dominio es X y que toma valores en el conjunto de las etiquetas (categorías) posibles Y. Si bien los términos *clasificador* e *hipótesis* pueden utilizarse indistintamente, el primero se usará para enfatizar la aplicación de una regla predictiva para clasificar individuos desconocidos, y el segundo se empleará para enfatizar el hecho de que una regla se ha obtenido (o podría obtenerse) como resultado de un proceso de aprendizaje. Otros términos utilizados como sinónimos de clasificador e hipótesis son: *regla*, *regla predictiva*, *regla de clasificación*, *indicador*, y *modelo*.

Para evaluar la calidad de un clasificador se mide su tasa de error, esto es, la frecuencia con la que realiza predicciones incorrectas. Para ello es necesario un *conjunto de prueba* $S' = \{(x'_1, y'_1), ..., (x'_M, y'_M)\}$, este conjunto está formado por *muestras de prueba*: muestras etiquetadas pero que el ordenador no ha visto previamente, esto es, que no pertenecen al conjunto de entrenamiento. Aunque ambos son subconjuntos del conjunto total de muestras patrón disponibles, los conjuntos de entrenamiento y de prueba han de ser disjuntos. Los individuos del conjunto de prueba se pasan al clasificador como parámetro de entrada y éste clasifica cada individuo en una categoría, si esta categoría coincide con la etiqueta del individuo entonces el individuo ha sido clasificado correctamente. La fracción de individuos mal clasificados dentro del conjunto de prueba S' se llama *error de prueba* del clasificador, $\epsilon_{S'}$. De forma análoga, la fracción de individuos mal clasificados dentro del conjunto de entrenamiento S se denomina *error de entrenamiento* o *error aparente*, ϵ_S.

$$\epsilon_{S'} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{I}(y'_i \neq h(x'_i)), \qquad (x'_i, y'_i) \in S' \; \forall i = 1, 2, ..., M.$$

$$\epsilon_S = \frac{1}{m} \sum_{i=1}^{m} \mathbf{I}(y_i \neq h(x_i)), \qquad (x_i, y_i) \in S \; \forall i = 1, 2, ..., m. \tag{1.1}$$

De ahora en adelante, siempre y cuando el contexto no dé lugar a ambigüedades, se omitirá el subíndice del error de prueba y de entrenamiento.

Es obvio que si no hay relación entre el conjunto de entrenamiento S y el conjunto de prueba S' entonces el problema de aprendizaje es irresoluble. Es por este motivo que en el estudio y diseño de algoritmos de aprendizaje se parte de la hipótesis de que los conjuntos S y S'se toman aleatoriamente de la *misma* fuente. Esto es, se asume que las muestras de ambos conjuntos se toman de forma arbitraria a partir de una distribución desconocida pero fija $\mathcal{D}$, y además se supone que los conjuntos se generan a partir de la *misma* distribución. De este modo se introduce el *error de generalización* $\epsilon_{\mathcal{D}}(h)$, el cual se encarga de medir la probabilidad de que la hipótesis h clasifique incorrectamente un individuo aleatorio x de la distribución $\mathcal{D}$. El error de generalización es el valor esperado del error de prueba de cualquier conjunto generado por $\mathcal{D}$. El objetivo de los algoritmos de aprendizaje es construir clasificadores con un error de generalización pequeño.

$$\epsilon_{\mathcal{D}}(h) = \mathbf{E}_{S' \sim \mathcal{D}}[\epsilon_{S'}(h)]$$

1.2. Boosting

La motivación en el diseño del boosting era crear un procedimiento que combinase las predicciones de varios clasificadores débiles para formar un comité que diese resultados precisos.

El boosting asume la disponibilidad de un *algoritmo de aprendizaje débil* A, o *algoritmo base*, el cual produce un *clasificador débil* h perteneciente al conjunto de hipótesis $\mathcal{H}$, a partir de un conjunto de entrenamiento con muestras etiquetadas S. La meta del boosting es mejorar el desempeño del algoritmo de aprendizaje débil A. En este contexto, se supone que los clasificadores débiles no son del todo triviales en el sentido de que su tasa de error es ligeramente mejor que la de un clasificador cuyas predicciones se basan en elecciones aleatorias. Por tanto, un clasificador débil puede ser irregular y moderadamente impreciso, pero no puede ser enteramente trivial. Esta hipótesis recibe el nombre de *conjetura del aprendizaje débil* en Schapire y Freund (2013), y es clave en el estudio del boosting. A continuación se introduce una definición formal para el aprendizaje débil.

Definición 1.1. Un algoritmo de aprendizaje A se dice que es un **algoritmo débil de parámetro γ para la clase de hipótesis $\mathcal{H}$** si: $\forall \delta \in (0, 1)$ y para toda distribución $\mathcal{D}$ sobre X, existe $m_{\mathcal{H}}(\delta)$ de modo que al entrenar el algoritmo A con $m > m_{\mathcal{H}}(\delta)$ muestras independientes igualmente distribuidas (i.i.d.) generadas por $\mathcal{D}$ entonces éste devuelve una hipótesis h tal que, con probabilidad mayor que $1 - \delta$, el error de generalización satisface $\epsilon_{\mathcal{D}}(h) \leq 1/2 - \gamma$.

La conjetura del aprendizaje débil no es otra cosa que la suposición de que se trabaja con una clase de hipótesis $\mathcal{H}$ para la cual se cuenta con un algoritmo débil de parámetro γ.

Como cualquier otro algoritmo de aprendizaje, un algoritmo de boosting toma como parámetro de entrada un conjunto de muestras de entrenamiento $S = \{(x_1, y_1), ..., (x_m, y_m)\}$ donde cada x_i es un individuo de X, y cada $y_i \in Y$ es su clase o etiqueta asociada. En este trabajo se asume el caso más sencillo en el cual solo hay dos clases: $Y = \{-1, +1\}$.

El único medio que tiene un algoritmo de boosting para aprender de los datos es llamando sucesivamente al algoritmo de aprendizaje débil. Ahora bien, si se llama repetidamente a este algoritmo base sin hacer nada más, siempre con el mismo conjunto de entrenamiento S, entonces no debe esperarse que nada interesante ocurra. Al contrario, cabría esperar que siempre arrojase los mismos resultados. De este modo resulta evidente que si el algoritmo de boosting pretende mejorar la actuación del algoritmo simple entonces, de algún modo, debe manipular los datos que se proporcionan a este algoritmo débil.

Un algoritmo de aprendizaje A toma como entrada un conjunto de entrenamiento S y genera un clasificador $h : X \to Y = \{-1, +1\}$ que minimice el error de entrenamiento dado por (1.1).

La idea del boosting es aplicar sucesivamente el algoritmo base a versiones modificadas del conjunto de entrenamiento, de este modo en cada iteración se genera un nuevo clasificador débil h_t que posteriormente se combina formando un clasificador H. Generalmente los clasificadores débiles se combinan a través de una suma ponderada:

$$H(x) = \text{signo}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$

En esta ecuación, los pesos $\alpha_1, \alpha_2, ..., \alpha_T$ se calculan mediante el algoritmo de boosting y determinan la contribución del clasificador h_t dentro del *comité* H. El efecto de estos pesos es conceder mayor importancia a los clasificadores más precisos. La forma en la que se calculan los pesos α_t y el modo en que se modifica el conjunto de entrenamiento S en cada iteración $t = 1, 2, ..., T$, dan lugar a diferentes tipos de algoritmos de boosting. Algunos de estos tipos son: AdaBoost (Freund y Schapire, 1997), ARC-X4 (Breiman, 1996), ConfidenceBoost (Schapire y Singer, 1999), y LogitBoost (Friedman *et al.*, 1998).

A continuación se detalla el pseudo-código de AdaBoost (**Algoritmo 1**), éste es uno de los algoritmos de boosting más populares. AdaBoost fue descrito por primera vez en Freund y Schapire (1997). AdaBoost es un algoritmo iterativo que avanza llamando

en cada ronda al algoritmo base. En cada iteración el algoritmo modifica el conjunto de entrenamiento mediante una *distribución* de pesos sobre las muestras etiquetadas. La distribución empleada en la ronda t se denota D_t, y el peso asignado a la muestra i-ésima del conjunto de entrenamiento se denota $D_t(i)$. Este peso refleja la importancia de clasificar correctamente la muestra i-ésima en la iteración t. De este modo, se puede forzar que el algoritmo base preste más atención a muestras que en iteraciones previas no se han clasificado correctamente, a expensas de prestar menor atención a muestras que en rondas anteriores habían sido clasificadas con éxito. Inicialmente todos los pesos poseen el mismo valor y en cada iteración se aumentan los pesos de las muestras clasificadas incorrectamente, de tal forma que las muestras más difíciles de clasificar van adquiriendo un peso mayor obligando al algoritmo a centrarse en ellas.

El propósito del algoritmo base es hallar un clasificador débil $h_t : X \to \{-1, 1\}$ apropiado para la distribución D_t, la calidad del clasificador débil se mide por su error ponderado mediante la distribución D_t:

$$\epsilon_t = \mathbf{Pr}_{i\sim D_t}[h_t(x_i) \neq y_i] = \sum_{i:h_t(x_i)\neq y_i} D_t(i). \tag{1.2}$$

El algoritmo débil A trata de obtener un clasificador h_t con un bajo error ponderado ϵ_t. Sin embargo, puesto que se trata de clasificadores débiles, no se espera que este error sea especialmente pequeño. De hecho tan solo cabe esperar que h_t sea un poco mejor que un clasificador cuyas predicciones son aleatorias, y generalmente ϵ_t tendrá un valor alejado del cero.

Siguiendo el hilo del párrafo anterior, y recordando la Definición 1.1, la conjetura del aprendizaje débil consiste en que los errores de los clasificadores débiles están acotados superiormente por $\frac{1}{2}$, es decir, cada ϵ_t vale como mucho $\frac{1}{2} - \gamma$ para cierta constante positiva γ. Por lo tanto se tiene que $\epsilon_t < \frac{1}{2}$, y en consecuencia $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t} > 0$. Nótese que α_t se hace grande cuando ϵ_t disminuye, y por lo tanto, cuanto más preciso es un clasificador h_t, mayor importancia se le asigna.

La distribución D_t se actualiza en cada iteración siguiendo la regla que aparece en el **Algoritmo 1**. Esta regla indica que los pesos $D_t(i)$ de las muestras (x_i, y_i) que han sido correctamente clasificadas por h_t son multiplicados por $e^{-\alpha_t} < 1$, y aquellos que han sido erróneamente clasificados son multiplicados por $e^{\alpha_t} > 1$. Seguidamente los nuevos pesos son multiplicados por un factor de normalización Z_t. El efecto de esta regla es aumentar los pesos de las muestras mal clasificadas por h_t y disminuir los pesos de las muestras debidamente clasificadas. De este modo el peso tiende a repartirse entre las muestras más difíciles de clasificar, el algoritmo AdaBoost se encarga de escoger una nueva distribución D_{t+1} en la que el anterior clasificador débil h_t vería mermado su desempeño. En este

Algoritmo 1: AdaBoost

Entrada:

- Una clase de hipótesis base $\mathcal{H}$ conteniendo funciones $h : X \to \{-1, 1\}$.
- Un conjunto de entrenamiento: $S = \{(x_1, y_1), ..., (x_m, y_m)\}$, donde $x_i \in X$, $y_i \in Y = \{-1, 1\}$.
- Un algoritmo de aprendizaje débil A que admite como entrada un conjunto de entrenamiento S y una distribución D sobre el mismo, y devuelve una hipótesis base $h \in \mathcal{H}$ con un error ponderado pequeño $\sum_{i=1}^{m} D(i)\mathbf{I}(h(x_i) \neq y_i)$.

inicio

$D_1(i) = 1/m$ para $i = 1, ..., m$.

para $t = 1, ..., T$ **hacer**

Entrenar el algoritmo base A con el conjunto de entrenamiento y la distribución D_t, y seleccionar la hipótesis $h_t : X \to \{-1, 1\}$ que minimice el error:

$$\epsilon_t = \mathbf{Pr}_{i \sim D_t}[h_t(x_i) \neq y_i].$$

si $\epsilon_t \geq \frac{1}{2}$ **entonces**

Salida: Hipótesis final: $H(x) = \text{signo}\left(\sum_{t'=1}^{t} \alpha_{t'} h_{t'}(x)\right)$.

Tomar $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$.

para $i = 1, ..., m$ **hacer**

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$
$$= \frac{D_t(i) \exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t},$$

donde Z_t es el factor de normalización responsable de que los pesos de la distribución D_{t+1} sumen 1.

Salida: Hipótesis final: $H(x) = \text{signo}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.

sentido, AdaBoost trata de que el algoritmo base aprenda algo nuevo de los datos en cada iteración.

Después de varias llamadas al algoritmo base, AdaBoost combina los distintos clasificadores débiles en un único clasificador *combinado* H. Esto se consigue mediante una votación ponderada, en la cual dada una muestra desconocida x, el clasificador H evalúa dicha muestra en cada clasificador h_t, y cada uno de ellos proporciona un resultado, fi-

nalmente H devuelve el valor predicho por la mayoría *ponderada* de estos clasificadores débiles. El voto del clasificador t-ésimo está ponderado por el parámetro α_t.

Nótese que el algoritmo base A ha de ser un algoritmo débil de parámetro γ para la clase de hipótesis $\mathcal{H}$ para que el boosting funcione. En los siguientes capítulos probaremos varios teoremas que requieren esta condición. Sin embargo AdaBoost no necesita conocer el valor del parámetro γ, sino que se ajusta y adapta a los errores ϵ_t, los cuales pueden variar considerablemente reflejando los cambiantes niveles de desempeño entre los clasificadores débiles. Por este motivo se dice que AdaBoost es un algoritmo de *boosting adaptativo*, precisamente su nombre se debe a esta cualidad de ajustarse y adaptarse en cada iteración.

Capítulo 2

Convergencia de AdaBoost

En el capítulo anterior se presentó AdaBoost como algoritmo de boosting. Ahora bien, un algoritmo de boosting es aquel que al ser provisto de un número razonable de muestras etiquetadas y de un algoritmo de aprendizaje débil, es capaz de producir una hipótesis que realice predicciones casi perfectas en muestras que no han sido vistas durante el entrenamiento. Para probar que AdaBoost es un algoritmo de boosting debemos demostrar que el error de generalización del modelo que genera puede hacerse tan pequeño como se quiera. El análisis debe centrarse en el error de generalización, pues éste es quien mide el error cometido al clasificar muestras que son desconocidas para el modelo, que no han sido vistas durante la etapa de entrenamiento.

Uno de los objetivos de este capítulo es dar una cota para el error de generalización de un modelo obtenido mediante AdaBoost. Para lograr este objetivo es fundamental realizar un estudio previo sobre el error de entrenamiento. Como se mostrará a lo largo del capítulo, ajustar el modelo buscando minimizar el error de entrenamiento es un buen método para reducir el error de generalización.

Otro de los objetivos del capítulo es introducir al lector en la teoría de los márgenes, el desarrollo de este concepto es doblemente ventajoso. Por un lado, resulta una vía alternativa para analizar el error de generalización de AdaBoost, y por otro lado establece el marco de trabajo necesario para desarrollar el tercer capítulo de este trabajo.

Este capítulo se ha desarrollado siguiendo la teoría presente en los capítulos 2, 3, 4 y 5 de Schapire y Freund (2013).

A continuación se presentan unos resultados previos que no son exclusivos del boosting, pero que serán herramientas claves a lo largo del capítulo.

2.1. Resultados previos

En esta sección se estudia el problema básico de construir, a partir de un conjunto de entrenamiento, una regla cuyas predicciones sobre nuevos individuos sean muy precisas. Se mostrará que si es posible hallar una regla *simple* que ajuste bien las muestras de entrenamiento, y si el conjunto de entrenamiento no es demasiado pequeño, entonces esta regla generalizará bien, dando correctas predicciones sobre ejemplos de prueba que no habían sido vistos previamente.

2.1.1. Equilibrio entre simplicidad y consistencia

El algoritmo de aprendizaje encargado de generar la regla predictiva toma como parámetro de entrada un conjunto de entrenamiento $S = \{(x_1, y_1), ..., (x_m, y_m)\}$. Al igual que en el primer capítulo, se asume que solo hay dos valores posibles para las etiquetas: $y_i \in Y = \{-1, +1\} \ \ \forall i = 1, 2, ..., m$.

Para ilustrar en qué consiste el balance entre simplicidad y consistencia, se empleará el siguiente ejemplo: si uno quiere generar una regla que ajuste el siguiente conjunto de entrenamiento (Cuadro 2.1) rápidamente notará que a partir de cierto valor v todos los individuos $x_i > v$ llevan asignada la etiqueta +1, y el resto están etiquetados con -1. La regla puede escribirse del siguiente modo:

$$h(x) = \begin{cases} +1 & \text{si } x \geq v \\ -1 & \text{en otro caso} \end{cases} \tag{2.1}$$

Individuos	1,2	2,8	8,0	3,3	5,0	4,5	7,4	5,6	3,8	6,6	6,1	1,7
Etiquetas	−	−	+	−	−	−	+	+	−	+	+	−

Cuadro 2.1: Ejemplo de conjunto de entrenamiento

Una regla con la forma de (2.1) se denomina *regla umbral*. Tomando un valor $v = 5{,}3$ se obtiene una regla que es *consistente* con el conjunto de entrenamiento, ya que predice correctamente la etiqueta de todos sus individuos, y además es *simple*.

Si bien el término *simple* puede sonar un tanto vago, es algo que cualquier persona reconoce cuando lo ve. Tal y como se explica en Schapire y Freund (2013), la *simplicidad* está íntimamente ligada con nuestras expectativas a priori: por un lado esperamos que los

datos puedan ser explicados mediante una regla sencilla, y por otro lado consideramos que una regla es simple cuando coincide con nuestras expectativas.

Ahora bien, si se considera un conjunto de entrenamiento algo más complejo como el del Cuadro 2.2, nótese que solamente se trata de una ligera modificación del anterior, no resulta tan evidente hallar una regla que sea a la vez consistente y simple. Uno podría tratar de obtener una regla simple como la de (2.1), pero ningún valor de v permite obtener una regla umbral que sea consistente con el conjunto de entrenamiento, debería aceptar cometer un pequeño número de errores. De forma alternativa se puede diseñar una regla algo más compleja que sí sea consistente:

$$h(x) = \begin{cases} -1 & \text{si } x < 3{,}4 \\ +1 & \text{si } 3{,}4 \leq x < 4{,}0 \\ -1 & \text{si } 4{,}0 \leq x < 5{,}2 \\ +1 & \text{si } 5{,}2 \leq x < 6{,}3 \\ -1 & \text{si } 6{,}3 \leq x < 7{,}1 \\ +1 & \text{si } x \geq 7{,}1 \end{cases} \tag{2.2}$$

Individuos	1,2	2,8	8,0	3,3	5,0	4,5	7,4	5,6	3,8	6,6	6,1	1,7
Etiquetas	−	−	+	−	−	−	+	+	+	−	+	−

Cuadro 2.2: Ligera modificación del anterior conjunto de entrenamiento

Generalmente a la hora de construir un clasificador se busca que cumpla dos criterios básicos:

1. Que ajuste bien el conjunto de entrenamiento, es decir se busca que sea consistente.
2. Que sea simple.

Como se aprecia en el segundo ejemplo, generalmente estos dos criterios entran en conflicto: se puede ajustar muy bien los datos a costa de escoger una hipótesis compleja, y al revés, normalmente tomar una regla simple conlleva un peor ajuste de los datos. Este balance entre simplicidad y consistencia es el tema central de muchos estudios en problemas de clasificación y aprendizaje automático.

2.1.2. Sesgo del error de entrenamiento

La forma de determinar qué tan bien ajusta el conjunto de entrenamiento S una hipótesis particular h es mediante su *error de entrenamiento*, también llamado *error empírico*,

esto es la fracción de muestras mal clasificadas de las m muestras de entrenamiento. Se denota[1]:

$$\widehat{\text{err}}(h) = \frac{1}{m}\sum_{i=1}^{m} \mathbf{I}\{h(x_i) \neq y_i\}.$$

Aunque en el apartado 2.1.1 se dijo que a la hora de construir una regla predicativa h se busca que ésta sea simple y que tenga un bajo error de entrenamiento, en última instancia el objetivo del aprendizaje automático es hallar reglas que sean altamente precisas en muestras que no se han empleado durante el entrenamiento (que no formaban parte de S), muestras de un conjunto de prueba. Generalmente se asume que ambos conjuntos, el de entrenamiento y el de prueba, se generan a partir de la misma distribución $\mathcal{D}$. Con respecto a esta distribución $\mathcal{D}$, el valor esperado del error de entrenamiento de una regla h se llama *error de generalización*, o *error verdadero*,y coincide con la probabilidad de clasificar de forma incorrecta un individuo cualquiera (x, y) elegido de forma aleatoria de la distribución $\mathcal{D}$. Se denota:

$$\text{err}(h) = \mathbf{Pr}_{(x,y)\sim\mathcal{D}}[h(x) \neq y]$$

Evidentemente, un algoritmo de aprendizaje no tiene los medios para medir directamente el error de generalización $\text{err}(h)$ que precisamente trata de minimizar. En su lugar, debe hacer uso del error empírico $\widehat{\text{err}}(h)$. Si se trabajase con una sola hipótesis h, entonces $\widehat{\text{err}}(h)$ sería un estimador razonable para $\text{err}(h)$, que es su valor esperado; en este sentido, se trata de un estimador insesgado. Sin embargo, generalmente, un algoritmo de aprendizaje trabajará con un espacio de hipótesis $\mathcal{H}$ grande y elegirá la hipótesis h con menor error de entrenamiento $\widehat{\text{err}}(h)$. Desafortunadamente en este caso el error $\widehat{\text{err}}(h)$ de la regla seleccionada h no será un estimador insesgado, de hecho casi con total seguridad subestimará el error verdadero $\text{err}(h)$. La razón de esto es que al seleccionar la hipótesis con menor error de entrenamiento, el algoritmo favorece aquellas hipótesis cuyo error empírico es menor que su error verdadero. Es por este motivo que se dice que el error de entrenamiento es optimista, está sesgado.

Para entender este efecto se propone el siguiente experimento mental: imagine una clase donde el profesor pide a sus alumnos que anoten en un papel cuál creen que será el resultado de lanzar una moneda al aire diez veces, teniendo en cuenta el orden de lanzamiento y su resultado. Obviamente se espera que cualquier estudiante acierte la mitad de las tiradas. Sin embargo, quizá en una clase de unas cincuenta personas, es bastante probable que haya un estudiante muy afortunado que acierte en sus predicciones ocho o nueve lanzamientos de moneda. Aunque puede parecer que este estudiante posee

[1]En el capítulo 1 empleábamos ϵ_S para referirnos al error de entrenamiento.

poderes de clarividencia, lo que ocurre realmente es que la fortuna hizo que la verdadera precisión del estudiante del 50 % aparentase ser mucho mayor. Cuanto más grande sea la clase, mayor es el efecto: en el caso extremo, para un número muy grande de estudiantes, se esperaría que un estudiante pudiese predecir correctamente las diez tiradas.

En problemas de aprendizaje automático, por estas mismas razones, es probable que una regla que se comporta muy bien ajustando el conjunto de entrenamiento lo haga porque se fija en una serie de falsos patrones que han aparecido por completa casualidad. Inevitablemente, esto hará que el error de entrenamiento sea menor que el error verdadero de la regla escogida. Más aún, como se explica en Schapire y Freund (2013), la cantidad de sesgo depende directamente en la simplicidad o complejidad de las hipótesis consideradas: cuanto más compleja sea la forma que pueden adquirir las hipótesis, es decir cuantas menos restricciones se impongan, mayor será el espacio de hipótesis $\mathcal{H}$ y mayor será el sesgo. Afortunadamente, más adelante se verá que el sesgo se puede controlar cuando se disponga de un conjunto de entrenamiento suficientemente grande.

2.1.3. Análisis del error de generalización

En primer lugar se estudiará el error de generalización de una sola hipótesis, y posteriormente se estudiará el caso de una familia de hipótesis $\mathcal{H}$.

Una sola hipótesis

En primer lugar, consideremos una hipótesis h fija. Previamente se ha hablado de utilizar el error de entrenamiento como representante del error de generalización. Esto sirve de motivación para cuestionarse cuánto puede diferir el error de entrenamiento $\widehat{\mathrm{err}}(h)$ respecto del error verdadero $\mathrm{err}(h)$ en función del número de muestras m del conjunto de entrenamiento S. Nótese en primer lugar que existe la posibilidad de que el conjunto seleccionado S no sea en absoluto representativo y entonces el error empírico sería una estimación muy pobre del error verdadero. Esto quiere decir que es imposible dar garantías de que una acotación se sostenga con total certeza, por ello se trata de buscar una acotación que se sostenga *con mucha probabilidad* sobre la elección de un conjunto de entrenamiento aleatorio.

Una de las acotaciones más simples y más ampliamente usada es la llamada *desigualdad de Hoeffding* que se presenta a continuación en forma de teorema. La demostración del mismo puede encontrarse en Hoeffding (1963).

Teorema 2.1 (Desigualdad de Hoeffding)**.** *Sean $X_1, ..., X_m$ variables independientes tal que $X_i \in [0, 1]$. Sea $A_m = \frac{1}{m}\sum_{i=1}^{m} X_i$, entonces para cuaqluier $\epsilon > 0$ se tiene que*

$$\boldsymbol{Pr}[A_m \geq \boldsymbol{E}[A_m] + \epsilon] \leq \exp(-2m\epsilon^2) \tag{2.3}$$

$$\boldsymbol{Pr}[A_m \leq \boldsymbol{E}[A_m] - \epsilon] \leq \exp(-2m\epsilon^2) \tag{2.4}$$

Definiendo las variables $X_i = \mathbf{I}(h(x_i) = y_i)$ para cada $i = 1, 2, ..., m$, entonces $A_m = \widehat{\text{err}}(h)$ y $\mathbf{E}[A_m] = \text{err}(h)$. Aplicando el Teorema 2.1, y en particular la cota (2.4), se tiene que para $\epsilon > 0$:

$$\mathbf{Pr}[\text{err}(h) \geq \widehat{\text{err}}(h) + \epsilon] = \mathbf{Pr}[\widehat{\text{err}}(h) \leq \text{err}(h) - \epsilon] \leq \exp(-2m\epsilon^2) \tag{2.5}$$

Dado δ tal que $1 > \delta > 0$, es posible tomar $\epsilon = \sqrt{\frac{\ln 1/\delta}{2m}}$ y del párrafo anterior se deduce que:

$$\mathbf{Pr}\left[\text{err}(h) \geq \widehat{\text{err}}(h) + \sqrt{\frac{\ln 1/\delta}{2m}}\right] \leq \exp(-2m\epsilon^2)$$

Esto quiere decir que, con probabilidad al menos $1 - \delta$,

$$\text{err}(h) \leq \widehat{\text{err}}(h) + \sqrt{\frac{\ln 1/\delta}{2m}}.$$

Es decir, si h posee un bajo error empírico $\widehat{\text{err}}(h)$ sobre un conjunto de entrenamiento grande, entonces con gran certeza el error de generalización será bajo también.

Mediante un razonamiento análogo, utilizando la cota (2.3) se obtiene una cota inferior para el error $\text{err}(h)$. Es posible combinar ambas cotas del error de generalización gracias a la *desigualdad de Boole*: $\mathbf{Pr}[a \vee b] \leq \mathbf{Pr}[a] + \mathbf{Pr}[b]$ dando lugar a:

$$\mathbf{Pr}[|\text{err}(h) - \widehat{\text{err}}(h)| \geq \epsilon] \leq 2 \cdot \exp(-2m\epsilon^2)$$

Dado δ tal que $1 > \delta > 0$, es posible tomar $\epsilon = \sqrt{\frac{\ln(2/\delta)}{2m}}$ y entonces se tiene que con probabilidad al menos $1 - \delta$:

$$|\text{err}(h) - \widehat{\text{err}}(h)| \leq \sqrt{\frac{\ln 2/\delta}{2m}} \tag{2.6}$$

Queda claro que en el caso de una hipótesis h fija, con casi total seguridad, tomando un conjunto de entrenamiento grande es posible que el error de entrenamiento sea un fiel representante del error de generalización. En los siguientes apartados se verá que al considerar una familia de hipótesis, en lugar de una sola hipótesis, la cota del error verdadero dependerá de más parámetros aparte del número de muestras de entrenamiento m.

Una familia de hipótesis finita

Uno podría estar tentado de utilizar (2.6) para acotar la diferencia entre el error de entrenamiento y el error de generalización de una hipótesis h obtenida mediante un algoritmo de aprendizaje A. No obstante, los argumentos utilizados para obtener la acotación (2.6) no sirven ahora. En este caso ya no se trata de una *sola* hipótesis h, sino que el algoritmo A escoge una hipótesis dentro de un conjunto $\mathcal{H}$. El razonamiento empleado para obtener la cota (2.6) requiere que una única hipótesis h sea seleccionada *antes* de que el conjunto de entrenamiento S sea aleatoriamente escogido. En el caso que ahora nos concierne, la regla predictiva h se escoge en base al conjunto de entrenamiento, no se trata de una hipótesis fija sino que la elección de h por parte del algoritmo depende del conjunto de entrenamiento.

A pesar de este inconveniente, a continuación se verá que es posible acotar la diferencia entre el error de generalización y el error empírico de una hipótesis obtenida a partir de un algoritmo de aprendizaje.

En primer lugar se probará que el error de entrenamiento de *todas* las hipótesis $h \in \mathcal{H}$ está, con mucha probabilidad, próximo al valor de su error verdadero.

El análisis se centra en dar una cota superior del error de generalización por simplicidad y por ser el caso interesante. No es relevante que el algoritmo escoja en algún momento fortuito una hipótesis cuyo error de generalización sea significativamente menor que su error de entrenamiento. Este caso es muy atípico, como se ha mencionado previamente, los algoritmos de aprendizaje tienden a favorecer hipótesis con errores empíricos pequeños, es muy improbable que el error verdadero sea todavía menor.

Teorema 2.2. *Sea $\mathcal{H}$ un espacio finito de hipótesis, y sea $S = \{(x_1, y_1), ..., (x_m, y_m)\}$ un conjunto de entrenamiento obtenido aleatoriamente de una distribución de muestras etiquetadas $\mathcal{D}$. Sea $|\mathcal{H}|$ el tamaño del espacio de hipótesis, el número de reglas $h \in \mathcal{H}$. Entonces para cualquier $\epsilon > 0$, $\boldsymbol{Pr}[\exists h \in \mathcal{H} : err(h) \geq \widehat{err}(h) + \epsilon] \leq |\mathcal{H}| \cdot \exp(-2m\epsilon^2)$. Es decir, con probabilidad al menos $1 - \delta$, donde $1 > \delta > 0$,*

$$err(h) \leq \widehat{err}(h) + \sqrt{\frac{\ln |\mathcal{H}| + \ln 1/\delta}{2m}} \tag{2.7}$$

para todo $h \in \mathcal{H}$.

Demostración. Considérese $h \in \mathcal{H}$ fijo, entonces haciendo uso de (2.5) y de la *desigualdad de Boole* (recuérdese que $\mathcal{H}$ es finito):

$$\mathbf{Pr}[\exists h \in \mathcal{H} : \mathrm{err}(h) \geq \widehat{\mathrm{err}}(h) + \epsilon] \leq \sum_{h \in \mathcal{H}} \mathbf{Pr}[\mathrm{err}(h) \geq \widehat{\mathrm{err}}(h) + \epsilon] \leq |\mathcal{H}| \cdot \exp(-2m\epsilon^2).$$

Para obtener la expresión (2.7) basta tomar, para el δ dado, $\epsilon = \sqrt{\frac{\ln|\mathcal{H}|+\ln 1/\delta}{2m}}$ y tomar considerar el suceso complementario: $\mathbf{Pr}[\mathrm{A}] = 1 - \mathbf{Pr}[\mathrm{A}^c]$. □

Nótese que en la acotación (2.7) además del error empírico y el tamaño del conjunto de entrenamiento m ahora aparece un tercer factor que depende del tamaño del espacio de hipótesis. Este factor da cuenta, en cierto modo, de la complejidad de las hipótesis empleadas. Cuanto más complejas sean las hipótesis, más rico será el espacio $\mathcal{H}$ y por ende mayor será su dimensión.

Por último veamos que si $\hat{h}, h^* \in \mathcal{H}$, donde $\hat{h}$ minimiza el error de entrenamiento $\widehat{\mathrm{err}}(h)$, y h^* minimiza el error de generalización $\mathrm{err}(h)$, entonces $\hat{h}$ también minimiza aproximadamente el error verdadero. Basta aplicar dos veces el Teorema 2.2:

$$\begin{aligned}
\mathrm{err}(\hat{h}) &\leq \widehat{\mathrm{err}}(\hat{h}) + \epsilon \\
&= \min_{h\in\mathcal{H}} \ \widehat{\mathrm{err}}(h) + \epsilon \\
&\leq \widehat{\mathrm{err}}(h^*) + \epsilon \\
&\leq (\mathrm{err}(h^*) + \epsilon) + \epsilon \\
&= \min_{h\in\mathcal{H}} \ \mathrm{err}(h) + 2\epsilon.
\end{aligned}$$

Una familia de hipótesis infinita

Sea $\mathcal{H}$ un espacio de hipótesis infinito, $|\mathcal{H}| = \infty$, resulta evidente que la cota obtenida en el Teorema 2.2 carece de sentido ahora.

Un ejemplo de familia de hipótesis infinita es la familia de funciones umbral (2.1), donde el parámetro v puede tomar cualquier valor real y por ende existen infinitas hipótesis en esta familia. No obstante, utilizando reglas umbrales solo hay $m + 1$ formas distintas de etiquetar las m muestras (ver el ejemplo del Cuadro 2.3).

En el caso de las reglas umbrales, aunque el espacio de hipótesis es infinito, en cierto modo, solo hay $m+1$ hipótesis *efectivas* para un conjunto de entrenamiento con m ejemplos. En este caso es tentador reemplazar el valor $|\mathcal{H}|$ que aparece en (2.7) por el número de hipótesis *efectivas*, sin embargo tal argumento es totalmente falaz. En las hipótesis del Teorema 2.2, el conjunto de hipótesis (y por tanto el valor de $|\mathcal{H}|$) está fijado antes de tomar el conjunto de entrenamiento, en este caso el conjunto de hipótesis *efectivas* dependen del conjunto de entrenamiento. No obstante, estos razonamientos pueden formalizarse más y desarrollar argumentos más sofisticados que permitan utilizar esta idea de hipótesis *efectivas*.

Parámetro umbral	Individuos	1,2	2,8	3,3	5,0	8,0
$v \in (8{,}0, \infty)$	Etiquetas	−	−	−	−	−
$v \in (5, 8{,}0]$	Etiquetas	−	−	−	−	+
$v \in (3{,}3, 5{,}0]$	Etiquetas	−	−	−	+	+
$v \in (2{,}8, 3{,}3]$	Etiquetas	−	−	+	+	+
$v \in (1{,}2, 2{,}8]$	Etiquetas	−	+	+	+	+
$v \in (-\infty, 1{,}2]$	Etiquetas	+	+	+	+	+

Cuadro 2.3: Formas posibles de etiquetar los individuos de una muestra de tamaño m mediante reglas umbrales.

Definición 2.1. Dada una familia de hipótesis $\mathcal{H}$ y un conjunto de entrenamiento finito $S = \{(x_1, y_1), ..., (x_m, y_m)\}$, se define el **conjunto de dicotomías**, también llamado **conjunto de conductas**, $\prod_{\mathcal{H}}(S)$ como las distintas formas en las que es posible etiquetar los individuos de S mediante funciones de $\mathcal{H}$: $\prod_{\mathcal{H}}(S) = \{(h(x_1), ..., h(x_m)) : h \in \mathcal{H}\}$.

Definición 2.2. Dada una familia de hipótesis $\mathcal{H}$, se define la **función de crecimiento** $\prod_{\mathcal{H}}(m)$ como el máximo valor del cardinal del conjunto de dicotomías de conjuntos S con m muestras extraídas de un conjunto X: $\prod_{\mathcal{H}}(m) = \max_{S \in X^m} |\prod_{\mathcal{H}}(S)|$.

En el caso de la familia de reglas umbral se tiene que $\prod_{\mathcal{H}}(m) = m + 1$.

Ahora estamos en condiciones de enunciar un teorema más general que el Teorema 2.2, que será válido tanto en el caso de familias finitas como infinitas. La demostración de resultado puede encontrarse en Vapnik y Chervonenkis (2015).

Teorema 2.3. *Sea $\mathcal{H}$ cualquier espacio de hipótesis, se asume que se escoge aleatoriamente un conjunto de entrenamiento con m muestras. Entonces para cualquier $\epsilon > 0$, $\boldsymbol{Pr}[\exists h \in \mathcal{H} : err(h) \geq \widehat{err}(h) + \epsilon] \leq 8 \prod_{\mathcal{H}}(m) \cdot \exp(-m\epsilon^2/32)$. Es decir, con probabilidad al menos $1 - \delta$ ($1 > \delta > 0$),*

$$err(h) \leq \widehat{err}(h) + \sqrt{\frac{32(\ln \prod_{\mathcal{H}}(m) + \ln(8/\delta))}{m}} \tag{2.8}$$

para todo $h \in \mathcal{H}$.

Nótese que en este caso el factor $|\mathcal{H}|$ que en (2.7) daba cuenta de la complejidad de las hipótesis empleadas ha sido reemplazado por $\prod_{\mathcal{H}}(m)$, pero éste sigue mostrando que cuanto más simples sean las hipótesis empleadas (menor valor de $\prod_{\mathcal{H}}(m)$) mejor aproximará el error empírico al error verdadero.

Al observar la acotación (2.8) se aprecia que en casos donde la función de crecimiento sea polinómica en m, $\prod_{\mathcal{H}}(m) = O(m^d)$, entonces $\ln \prod_{\mathcal{H}}(m) \approx d \ln m$ y el segundo término a la derecha de la desigualdad (2.8), como función de m, tiende a cero a una velocidad $O\left(\sqrt{(\ln m)/m}\right)$.

Sin embargo hay casos en los que la función de crecimiento no es polinómica, como por ejemplo la función de crecimiento de la clase $\mathcal{H}$ cuyas hipótesis se definen como $+1$ en un conjunto finito de intervalos de la recta real y como -1 en el complementario. Un ejemplo de regla en este espacio $\mathcal{H}$ sería (2.2). Este espacio de hipótesis es tan rico que para *cualquier* etiquetado de un conjunto de entrenamiento, es posible construir un clasificador consistente tomando intervalos lo suficientemente pequeños alrededor de los individuos con la etiqueta $+1$. En consecuencia, el número de dicotomías para cualquier conjunto de m puntos distintos es exactamente 2^m, que es el total de formas distintas en las que se pueden agrupar m individuos en dos categorías. En este caso, $\ln \prod_{\mathcal{H}}(m) = m \ln 2$ y la cota (2.8) es inútil. Esto sirve de motivación para seguir introduciendo conceptos como la *dimensión VC* que se presenta a continuación.

Definición 2.3. Cuando una clase de hipótesis $\mathcal{H}$ es capaz de reproducir los 2^m etiquetados posibles de un conjunto S con m muestras, entonces se dice que S es **desmenuzado** por $\mathcal{H}$. Es decir, un conjunto con m muestras S es desmenuzado por $\mathcal{H}$ si y solo si $\prod_{\mathcal{H}}(S) = 2^m$.

Definición 2.4. Se define la **dimensión de Vapnik-Chervonenkis (VC)** de $\mathcal{H}$ como el tamaño de la mayor muestra S que puede ser desmenuzada por $\mathcal{H}$. Si muestras arbitrariamente grandes pueden ser desmenuzadas por $\mathcal{H}$ entonces su dimensión VC es infinito.

Por ejemplo, la clase de funciones umbral tiene dimensión VC igual a 1 ya que un punto cualquiera puede etiquetarse como $+1$ o -1 (lo que significa que la dimensión VC es al menos 1), pero ninguna pareja de puntos puede ser desmenuzada debido a que si el punto situado más a la izquierda está etiquetado con $+1$, entonces el otro punto a su derecha también debe llevar la etiqueta $+1$ (en consecuencia la dimensión VC es menor estrictamente que 2). Por otro lado, en el caso de la clase de funciones que se definen como $+1$ en un conjunto finito de intervalos de la recta real y como -1 en el complementario, se ha visto que cualquier conjunto podía desmenuzarse y por lo tanto su dimensión VC es infinito.

Cuando la dimensión VC de una clase de hipótesis $\mathcal{H}$ es infinito, $\prod_{\mathcal{H}}(m) = 2^m$ por definición. No obstante, cuando la dimensión VC de una clase tiene un valor finito d, entonces la función de crecimiento resulta ser polinomial, en concreto $O(m^d)$. Esta afirmación se se deduce del *Lema de Sauer*, cuya demostración puede encontrarse en Sauer (1972).

Lema 2.4 (Lema de Sauer)**.** *Si $\mathcal{H}$ es una clase de hipótesis con dimensión VC $d < \infty$, entonces para todo m: $\prod_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i}$.*

Nótese que si $m \leq d$, entonces esta cota es igual a 2^m y, por como se ha definido d, coincide precisamente con el valor de $\prod_{\mathcal{H}}(m)$. Para $m \geq d \geq 1$, la siguiente cota resulta útil (Kearns y Vazirani, 2018):

$$\prod_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i} \leq \left(\frac{em}{d}\right)^d \tag{2.9}$$

(e denota la base del logaritmo neperiano). La cota (2.9) se prueba mediante la fórmula del binomio de Newton. Puesto que $m \geq d \geq 1$, entonces $d/m \leq 1$ y se tiene que:

$$\left(\frac{d}{m}\right)^m \cdot \sum_{i=0}^{d} \binom{m}{i} \leq \sum_{i=0}^{d} \left(\frac{d}{m}\right)^i \cdot \binom{m}{i} = \left(1 + \frac{d}{m}\right)^m < \lim_{m\to\infty} \left(1 + \frac{d}{m}\right)^m = e^d.$$

Multiplicando ambos lados por $(m/d)^d$ se obtiene el resultado.

Haciendo uso de (2.9) y del Teorema 2.3 se deduce inmediatamente el siguiente resultado:

Teorema 2.5. *Sea $\mathcal{H}$ una clase de hipótesis con dimensión VC $d < \infty$, y sea S un conjunto de entrenamiento con m muestras obtenidas aleatoriamente, con $m \geq d \geq 1$. Entonces para cualquier $\epsilon > 0$, $\boldsymbol{Pr}[\exists h \in \mathcal{H} : err(h) \geq \widehat{err}(h) + \epsilon] \leq 8\left(\frac{em}{d}\right)^d \cdot \exp\left(-m\epsilon^2/32\right)$. Es decir, ignorando constantes, con probabilidad al menos $1 - \delta$ ($1 > \delta > 0$),*

$$err(h) \leq \widehat{err}(h) + O\left(\sqrt{\frac{d \ln(m/d) + \ln 1/\delta}{m}}\right) \tag{2.10}$$

para todo $h \in \mathcal{H}$.

En la cota (2.10) la dimensión VC del espacio de hipótesis d es un factor que depende de la complejidad de la clase de hipótesis, m es el tamaño del conjunto de entrenamiento, y el error de entrenamiento da cuenta de la consistencia del clasificador. Observando esta cota se advierte la importancia que tiene contar con un conjunto de entrenamiento grande, y se justifica la motivación de buscar hipótesis *simples* que ajusten bien las muestras de entrenamiento.

2.1.4. Hipótesis consistentes

Siempre que sea posible, el objetivo de un algoritmo de aprendizaje es obtener una hipótesis que sea *consistente* con los ejemplos de entrenamiento. Es decir, producir

una regla cuyo error de entrenamiento sea nulo. Por supuesto el análisis precedente sigue siendo válido, no obstante las cotas que se obtuvieron son mejorables en el caso de hipótesis consistentes. El siguiente resultado proporciona unas acotaciones mucho mejores para este caso (Vapnik y Chervonenkis, 1974; Blumer *et al.*, 1987, 1989).

Teorema 2.6. *Sea $\mathcal{H}$ un espacio de hipótesis y sea S un conjunto de entrenamiento con m muestras obtenidas aleatoriamente.*

1. *Si $\mathcal{H}$ es finito, entonces con probabilidad al menos $1-\delta$,*
$$err(h) \leq \frac{\ln|\mathcal{H}| + \ln 1/\delta}{m} \tag{2.11}$$
para toda hipótesis $h \in \mathcal{H}$ que sea consistente con S.
2. *Más general, para cualquier clase de hipótesis (finita o infinita) $\mathcal{H}$, entonces con probabilidad al menos $1-\delta$,*
$$err(h) \leq \frac{2\ln\prod_{\mathcal{H}}(2m) + 2\ln(2/\delta)}{m} \tag{2.12}$$
para toda hipótesis $h \in \mathcal{H}$ que sea consistente con S.
3. *Si $\mathcal{H}$ tiene dimensión VC d, con $m \geq d \geq 1$, entonces con probabilidad al menos $1-\delta$,*
$$err(h) \leq \frac{2d\ln(2em/d) + 2\ln(2/\delta)}{m} \tag{2.13}$$
para toda hipótesis $h \in \mathcal{H}$ que sea consistente con S.

Estas cotas son mejores que las obtenidas previamente porque la convergencia del error empírico al error verdadero con las anteriores acotaciones era del orden de $1/\sqrt{m}$, y en el caso de hipótesis consistentes se han obtenido cotas que dan una convergencia de orden $1/m$ (ignorando el término logarítmico), mucho más rápida.

2.2. Acotación del error de entrenamiento de AdaBoost

En la sección anterior se vio que el error de entrenamiento, el tamaño de la muestra de entrenamiento y la simplicidad del conjunto de hipótesis son clave en el análisis del error de generalización. Por ello, si uno busca garantizar la convergencia de algoritmos de boosting como AdaBoost es fundamental realizar un estudio previo sobre su error de entrenamiento.

En esta sección se proporcionará una cota del error de entrenamiento en base exclusivamente a la *conjetura del aprendizaje débil*. No se realizará ninguna suposición sobre

la distribución del conjunto de entrenamiento ni de como éste es obtenido, tampoco se especificará ningún algoritmo de aprendizaje débil concreto. De este modo se obtiene una cota de error que cuenta con la ventaja de ser flexible y general, que será aplicable a cualquier elección del algoritmo base.

2.2.1. Convergencia del error de entrenamiento de AdaBoost

El siguiente resultado proporciona una cota fundamental en el error de entrenamiento de AdaBoost, muestra que el error de entrenamiento de AdaBoost disminuye exponencialmente con el número de iteraciones del algoritmo.

Nótese que aunque en el pseudocódigo de AdaBoost (página 7) la distribución inicial de los pesos D_1 se correspondía con una distribución uniforme sobre el conjunto de entrenamiento, en las hipótesis del siguiente teorema no se realiza ninguna suposición sobre los pesos iniciales. De este modo el resultado es aplicable a cualquier distribución inicial D_1.

Teorema 2.7 (Acotación del error de entrenamiento de AdaBoost)**.** *Dada la notación del **Algoritmo 1** de la página 7, sea $\gamma_t = \frac{1}{2} - \epsilon_t$, y sea D_1 una distribución de pesos cualquiera sobre el conjunto de entrenamiento. Entonces el error de entrenamiento del clasificador resultante H, respecto la distribución D_1, está acotado del siguiente modo:*

$$\boldsymbol{Pr}_{i\sim D_1}\left[H\left(x_i\right) \neq y_i\right] \leq \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right). \tag{2.14}$$

Demostración. Sea

$$F(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

Deshaciendo la relación de recurrencia que permite obtener D_{t+1} en términos de D_t, se tiene que:

$$\begin{aligned} D_{T+1}(i) &= D_1(i) \times \frac{e^{-y_i\alpha_1 h_1(x_i)}}{Z_1} \times \cdots \times \frac{e^{-y_i\alpha_T h_T(x_i)}}{Z_T} \\ &= \frac{D_1(i)\exp\left(-y_i\sum_{t=1}^{T}\alpha_t h_t\left(x_i\right)\right)}{\prod_{t=1}^{T} Z_t} \\ &= \frac{D_1(i)\exp\left(-y_i F\left(x_i\right)\right)}{\prod_{t=1}^{T} Z_t}. \end{aligned} \tag{2.15}$$

Puesto que $H(x) = \text{signo}(F(x))$, resulta que si $H(x) \neq y$, entonces $yF(x) \leq 0$ y $\exp\left(-yF(x)\right) \geq 1$. Es decir, $\mathbf{I}(H(x) \neq y) \leq \exp\left(-yF(x)\right)$. Por lo tanto, el error de

entrenamiento del clasificador combinado es

$$\begin{aligned}\mathbf{Pr}_{i\sim D_1}\left[H\left(x_i\right) \neq y_i\right] &= \sum_{i=1}^{m} D_1(i)\mathbf{I}(H(x) \neq y) \\ &\leq \sum_{i=1}^{m} D_1(i)\exp\left(-y_i F(x_i)\right) \\ &= \sum_{i=1}^{m} D_{T+1}(i)\prod_{t=1}^{T} Z_t \\ &= \prod_{t=1}^{T} Z_t.\end{aligned} \tag{2.16}$$

donde se ha hecho uso de la Ecuación (2.15), el hecho de que los pesos $\{D_{T+1}(i)\}_{i=1}^{m}$ forman una distribución y por lo tanto su suma vale 1.

Para concluir la demostración basta con probar que $Z_t = \sqrt{1-4\gamma_t^2}$ e incorporarlo en la Ecuación (2.16). Nótese que el factor de normalización Z_t se escogía para que que la suma de los pesos de la distribución D_{T+1} sumase 1,

$$1 = \sum_{i=1}^{m} D_{t+1}(i) = \sum_{i=1}^{m} \frac{D_t(i)\exp\left(-\alpha_t y_i h_t(x_i)\right)}{Z_t},$$

a partir de esta expresión es posible escribir Z_t como:

$$Z_t = \sum_{i=1}^{m} D_t(i)\exp\left(-\alpha_t y_i h_t(x_i)\right). \tag{2.17}$$

Recordando las definiciones $\epsilon_t = \mathbf{Pr}_{i\sim D_t}[h_t(x_i) \neq y_i]$, $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ y $\gamma_t = \frac{1}{2} - \epsilon_t$, se

tiene finalmente que:

$$
\begin{aligned}
Z_t &= \sum_{i=1}^{m} D_t(i) \exp\left(-\alpha_t y_i h_t(x_i)\right) \\
&= \sum_{i:\, y_i = h(x_i)} D_t(i) \exp\left(-\alpha_t\right) + \sum_{i:\, y_i \neq h(x_i)} D_t(i) \exp\left(\alpha_t\right) \\
&= \exp\left(-\alpha_t\right) \cdot \sum_{i:\, y_i = h(x_i)} D_t(i) + \exp\left(\alpha_t\right) \cdot \sum_{i:\, y_i \neq h(x_i)} D_t(i) \\
&= \exp\left(-\alpha_t\right) \left(1 - \mathbf{Pr}_{i \sim D_t}\left[h_t(x_i) \neq y_i\right]\right) + \exp\left(\alpha_t\right) \mathbf{Pr}_{i \sim D_t}\left[h_t(x_i) \neq y_i\right] \\
&= \exp\left(-\alpha_t\right)(1 - \epsilon_t) + \exp\left(\alpha_t\right)\epsilon_t \\
&= \exp\left(-\alpha_t\right)\left(\frac{1}{2} + \gamma_t\right) + \exp\left(\alpha_t\right)\left(\frac{1}{2} - \gamma_t\right) \\
&= \exp\left(-\frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}\right)\left(\frac{1}{2} + \gamma_t\right) + \exp\left(\frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}\right)\left(\frac{1}{2} - \gamma_t\right) \\
&= \left(\sqrt{\frac{\epsilon_t}{1-\epsilon_t}}\right)\left(\frac{1}{2} + \gamma_t\right) + \left(\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}\right)\left(\frac{1}{2} - \gamma_t\right) \\
&= \left(\sqrt{\frac{\frac{1}{2} - \gamma_t}{\frac{1}{2} + \gamma_t}}\right)\left(\frac{1}{2} + \gamma_t\right) + \left(\sqrt{\frac{\frac{1}{2} + \gamma_t}{\frac{1}{2} - \gamma_t}}\right)\left(\frac{1}{2} - \gamma_t\right) \\
&= 2\sqrt{\frac{1}{4} - \gamma_t^2} = \sqrt{1 - 4\gamma_t^2}
\end{aligned}
\tag{2.18}
$$

□

El *borde* γ_t mide qué tanto mejor es el clasificador h_t comparado con un clasificador cuyas predicciones se realizan de forma aleatoria. A modo de ilustración de este resultado, supóngase que $\gamma_t \leq 0{,}1$ para todos los clasificadores h_t. Entonces el Teorema 2.7 implica que el error del clasificador combinado H es como mucho: $\mathbf{Pr}_{i \sim D_1}\left[H\left(x_i\right) \neq y_i\right] \leq \left(\sqrt{1 - 4(0{,}1)^2}\right)^T \approx (0{,}98)^T$. Es decir, el error de entrenamiento cae exponencialmente como función del número de clasificadores combinados (esto es, como número de iteraciones del algoritmo AdaBoost).

Nótese que la *conjetura del aprendizaje débil* es imprescindible para demostrar el Teorema 2.7, en concreto para que $1/2 < \epsilon_t$.

2.3. Acotación del error de generalización de AdaBoost

En el Teorema 2.7 que proporcionaba una cota para el error de entrenamiento de AdaBoost no se realizó ninguna suposición sobre los datos del conjunto de entrenamiento.

Tanto las muestras (x_i, y_i), como las hipótesis débiles h_t eran enteramente arbitrarias. No obstante, para estudiar el error de generalización es necesario realizar algunas suposiciones. Esto es porque, como se comentó en la sección 1.1, si no hay relación entre las muestras vistas durante el entrenamiento y las muestras del conjunto de prueba, entonces no puede esperarse que el clasificador se comporte bien en la etapa de prueba. Por lo tanto, como en el capítulo 1, se asume que todas las muestras (tanto las empleadas en el entrenamiento como las empleadas en la fase de prueba) se toman aleatoriamente de acuerdo a la misma distribución $\mathcal{D}$ (posiblemente desconocida) sobre $X \times \{-1, +1\}$.

Asimismo, para obtener una cota del error de generalización, se debe realizar alguna suposición sobre la complejidad de los clasificadores débiles h_t. Tal y como se hizo en la subsección 2.1.3, se supone que las hipótesis débiles h_t son seleccionadas de un espacio de hipótesis $\mathcal{H}$.

Habiendo probado en el Teorema 2.7 una cota para el error de entrenamiento de AdaBoost, a continuación se procede a obtener un cota para la diferencia entre el error de entrenamiento y el de generalización.

2.3.1. Forma y complejidad de los clasificadores débiles

Sea $\mathcal{H}$ el espacio de los clasificadores débiles, y sea C_T el espacio de clasificadores *combinados* que se podrían obtener al ejecutar el algoritmo AdaBoost durante T iteraciones. Estos clasificadores son de la forma:

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

siendo $\alpha_1, ..., \alpha_T \in \mathbb{R}$, y $h_1, ..., h_T \in \mathcal{H}$. Es posible escribir H como una función lineal $\sigma\colon \mathbb{R}^T \to \{-1, 0, 1\}$ de la forma:

$$\sigma(\mathbf{x}) = \text{signo}(\mathbf{w} \cdot \mathbf{x})$$

para cierto $\mathbf{w} \in \mathbb{R}^T$,

$$H(x) = \sigma(h_1(x), ..., h_T(x)).$$

Sea Σ_T el espacio de este tipo de funciones lineales. Entonces C_T puede escribirse como: $C_T = \{x \mapsto \sigma(h_1(x), ..., h_T(x)) : \sigma \in \Sigma_T; h_1(x), ..., h_T(x) \in \mathcal{H}\}$.

Para poder aplicar los resultados desarrollados en la sección 2.1 el clasificador H debe dar predicciones en el conjunto $\{1, -1\}$, por este motivo se redefine la función signo en el punto cero como $\text{signo}(0) = +1$.

El objetivo de este apartado es probar que el error de entrenamiento $\widehat{\text{err}}(h)$ es un buen estimador del error verdadero $\text{err}(h)$ para toda función $h \in C_T$. Como se vio en la sección 2.1, esto puede realizarse contando el número de hipótesis en C_T. Desafortunadamente, al ser Σ_T infinito (existe una función σ para cada $\mathbf{w} \in \mathbb{R}^T$), resulta que C_T es un conjunto infinito también. Sin embargo, podemos hacer uso de la *dimensión VC* para sortear esta dificultad. En el siguiente Lema se demuestra que la dimensión VC de Σ_T es finita.

Lema 2.8. *El espacio de funciones lineales Σ_n definido sobre $\mathbb{R}^n$ tiene dimensión VC n.*

Demostración. Para cada $i \in \{1, 2, ..., n\}$, sea $\mathbf{e}_i \in \mathbb{R}^n$ el vector cuya componente en la dirección i-ésima vale 1 y las componentes del resto de direcciones son 0. Consideremos el conjunto $\{\mathbf{e}_1, ..., \mathbf{e}_n\}$, entonces cualquier conjunto de etiquetas $y_1, ..., y_n$ con $y_i \in \{-1, +1\}$ puede reproducirse mediante funciones de Σ_n tomando el vector $\mathbf{w} = (y_1, ..., y_n)$ ya que $\text{signo}(\mathbf{w} \cdot \mathbf{e}_i) = y_i$. Es decir, el conjunto $\{\mathbf{e}_1, ..., \mathbf{e}_n\}$ puede ser desmenuzado por Σ_n y por lo tanto la dimensión VC de Σ_n es al menos n.

Razonando por reducción al absurdo es posible probar que la dimensión VC de Σ_n es menor estrictamente que $n + 1$ (lo que concluiría la prueba): supóngase que la dimensión VC de Σ_n es mayor o igual que $n + 1$, entonces debe existir un conjunto de $n + 1$ individuos $\{\mathbf{x}_1, ..., \mathbf{x}_{n+1}\} \subset \mathbb{R}^n$ que puede ser desmenuzado por Σ_n. Ahora, por un lado se tienen $n + 1$ vectores en un espacio de dimensión n, y como los $n+1$ vectores no pueden ser linealmente independientes, deben existir números reales $\beta_1, ..., \beta_{n+1}$, no todos nulos, de modo que

$$\sum_{i=1}^{n+1} \beta_i \mathbf{x}_i = \mathbf{0}.$$

Puede suponerse, sin pérdida de generalidad, que $\beta_{n+1} > 0$. No obstante, por otro lado, como el conjunto $\{\mathbf{x}_1, ..., \mathbf{x}_{n+1}\}$ puede ser desmenuzado por Σ_n, cualquier etiquetado puede obtenerse mediante funciones de Σ_n luego, en particular, debe existir $\mathbf{w} \in \mathbb{R}^n$ tal que

$$\text{signo}(\mathbf{w} \cdot \mathbf{x}_{n+1}) = +1, \tag{2.19}$$

y tal que

$$\text{signo}\,(\mathbf{w} \cdot \mathbf{x}_i) = \begin{cases} +1 & \text{if } \beta_i > 0 \\ -1 & \text{if } \beta_i \leq 0 \end{cases} \tag{2.20}$$

para $i = 1, ..., n$. Entonces, la Ecuación (2.19) dice que $\mathbf{w} \cdot \mathbf{x}_{n+1} > 0$, mientras que la Ecuación (2.20) implica que $\beta_i \mathbf{w} \cdot \mathbf{x}_i \geq 0$ para $i = 1, ..., n$. Esto conduce a la siguiente

contradicción:

$$\begin{aligned} 0 &= \mathbf{w} \cdot \mathbf{0} \\ &= \mathbf{w} \cdot \sum_{i=1}^{n+1} \beta_i \mathbf{x}_i \\ &= \sum_{i=1}^{n} \beta_i \left(\mathbf{w} \cdot \mathbf{x}_i\right) + \beta_{n+1} \left(\mathbf{w} \cdot \mathbf{x}_{n+1}\right) \\ &> 0 \end{aligned}$$

□

Ahora se cuenta con las herramientas necesarias para contar el número de dicotomías inducido por hipótesis de C_T en cualquier conjunto S. A continuación se estudiarán, por separado, los casos en los cuales la dimensión del espacio de hipótesis $\mathcal{H}$ es finito e infinito.

2.3.2. Espacio de hipótesis finito

Lema 2.9. *Sea $\mathcal{H}$ un espacio de hipótesis finito, y sea $m \geq T \geq 1$. Para cualquier conjunto S con m muestras, el número de dicotomías sobre S mediante funciones de C_T está acotado según:* $|\prod_{C_T}(S)| \leq \prod_{C_T}(m) \leq \left(\frac{em}{T}\right)^T |\mathcal{H}|^T$.

Demostración. Sea $S = \{x_1, ..., x_m\}$, considérese una secuencia *fija* de clasificadores $h_1, ..., h_T \in \mathcal{H}$. Respecto esta secuencia, se construye el conjunto $S' = \{\mathbf{x}'_1, ..., \mathbf{x}'_n\}$ donde sus elementos están definidos del siguiente modo: $\mathbf{x}'_i = (h_1(x_i), ..., h_t(x_i))$.

Por el Lema 2.8 se sabe que Σ_T tiene dimensión VC igual a T, y por el Lema de Sauer (Lema 2.4) y la Ecuación (2.9) aplicado a S' resulta que

$$\prod_{\Sigma_T}(S') \leq \left(\frac{em}{T}\right)^T \tag{2.21}$$

Esto es, para $h_1, ..., h_T \in \mathcal{H}$ fijos, el número de formas posibles de etiquetar los individuos de S mediante funciones del tipo $\sigma(h_1(x), ..., h_T(x))$ con $\sigma \in \Sigma_T$ está acotado superiormente por $\left(\frac{em}{T}\right)^T$.

Como el número de formas de elegir las hipótesis $h_1, ..., h_T$ es $|\mathcal{H}|^T$, entonces resulta que $|\prod_{C_T}(S)| \leq |\mathcal{H}|^T \cdot \prod_{\Sigma_T}(S')$, y por lo tanto haciendo uso de (2.21):

$$|\prod_{C_T}(S)| \leq |\mathcal{H}|^T \cdot \prod_{\Sigma_T}(S') \leq |\mathcal{H}|^T \cdot \left(\frac{em}{T}\right)^T$$

□

Ahora es posible aplicar los Teoremas 2.3 y 2.6 para obtener el siguiente resultado, el cual proporciona una acotación en el error de generalización de cualquier clasificador H obtenido mediante AdaBoost.

Teorema 2.10. *Supóngase que AdaBoost se ejecuta durante T iteraciones, utilizando un conjunto de entrenamiento con $m \geq T$ muestras aleatorias, y un espacio de hipótesis débiles $\mathcal{H}$ finito. Entonces, con probabilidad al menos $1-\delta$ (sobre la elección del conjunto de entrenamiento), el clasificador combinado H satisface*

$$err(H) \leq \widehat{err}(H) + \sqrt{\frac{32\left(T\ln\left(em|\mathcal{H}|/T\right) + \ln\left(8/\delta\right)\right)}{m}}.$$

Más aún, si H es consistente con el conjunto de entrenamiento ($\widehat{err}(H) = 0$), entonces con probabilidad al menos $1-\delta$

$$err(H) \leq \frac{2T\ln\left(2em|\mathcal{H}|/T\right) + 2\ln\left(2/\delta\right)}{m}.$$

Demostración. Basta incorporar la acotación de la ecuación del Lema 2.9 en los Teoremas 2.3 y 2.6. □

Ahora es posible probar que la *conjetura del aprendizaje débil* es suficiente para garantizar que AdaBoost generará un clasificador con un error de generalización arbitrariamente pequeño, siempre y cuando se proporcione un conjunto de entrenamiento suficientemente grande.

Corolario 2.10.1. *Supóngase que, además de las hipótesis del Teorema 2.10, también se cumple que cada clasificador débil posee un error $\epsilon_t \leq \frac{1}{2} - \gamma$ para algún $\gamma > 0$. Sea el número T igual al número entero más pequeño que es superior al valor $(\ln m)/(2\gamma^2)$. Entonces con probabilidad al menos $1-\delta$, el error de generalización del clasificador H obtenido al ejecutar el algoritmo AdaBoost durante T rondas será como mucho:*

$$O\left(\frac{1}{m}\left[\frac{(\ln m)(\ln m + \ln|\mathcal{H}|)}{\gamma^2} + \ln\left(\frac{1}{\delta}\right)\right]\right).$$

Demostración. Por el Teorema 2.7, el error de entrenamiento del clasificador combinado H vale como mucho $\exp\left(-2\gamma^2 T\right)$. Además se tiene que:

$$\frac{\ln m}{2\gamma^2} < T \iff \ln\frac{1}{m} = -\ln m < -2\gamma^2 T \iff \frac{1}{m} < \exp\left(-2\gamma^2 T\right),$$

luego el error de entrenamiento de H vale como mucho $\exp\left(-2\gamma^2 T\right) < 1/m$. Como hay m muestras en el conjunto de entrenamiento, esto quiere decir que el error de entrenamiento debe ser cero. Haciendo uso de la segunda parte del Teorema 2.10 se obtiene el resultado. □

El Corolario 2.10.1 proporciona una cota para el error de generalización cuando el algoritmo AdaBoost se detiene en la ronda T donde teóricamente el error de entrenamiento alcanza el cero. ¿Qué ocurre si el algoritmo no se detuviese en esa ronda sino en otras posteriores? Combinando los teoremas 2.7 y 2.10 se obtiene la siguiente cota del error de generalización:

$$\exp\left(-2\gamma^2 T\right) + O\left(\sqrt{\frac{T\ln\left(m|\mathcal{H}|/T\right) + \ln\left(1/\delta\right)}{m}}\right) \tag{2.22}$$

Esta función ha sido representada en la Figura 2.1, tomando $\gamma = 0{,}2$, $m = 10^6$, $|\mathcal{H}| = \exp(10)$, y $\delta = 0{,}05$. En esta representación puede observarse el clásico comportamiento de sobreajuste. Cuando T es pequeño, el término dominante es el primero y se observa una caída exponencial en la cota del error de generalización. Sin embargo, cuando T toma valores grandes, el segundo término es quien domina, dando lugar a un aumento considerable en la cota (2.22). Esta cota predice que para valores grandes de T AdaBoost sobreajusta el modelo que está entrenando, provocando una aumento del error de generalización. Sin embargo, como se explica en la sección 1.3 de Schapire y Freund (2013), a menudo AdaBoost es resistente al sobreajuste, y puede ser beneficioso ejecutar AdaBoost más rondas aunque ya se tenga un clasificador consistente. Este fenómeno no puede explicarse con la teoría que se ha visto hasta ahora, para explicar este comportamiento de AdaBoost es necesario introducir el concepto de *margen* que será el tema de la sección 2.4.

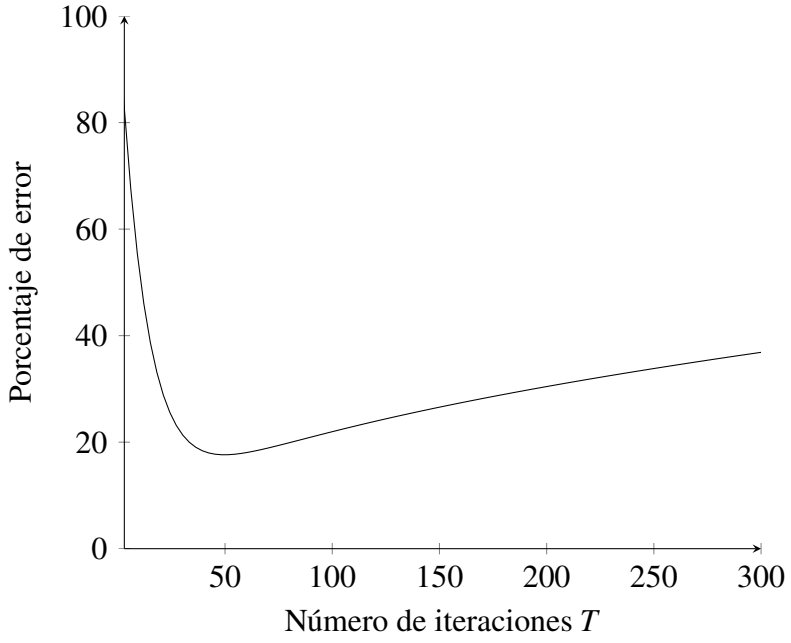


Figura 2.1: Cota del error de generalización de AdaBoost

2.3.3. Espacio de hipótesis infinito

Aunque el Teorema 2.10 se ha demostrado para un espacio $\mathcal{H}$ finito, es posible obtener un resultado análogo para el caso de un espacio $\mathcal{H}$ infinito razonando con su dimensión VC. Esencialmente se trata de adaptar el cálculo de $\prod_{C_T}(m)$ que se hizo en el Lema 2.9.

Lema 2.11. *Sea $\mathcal{H}$ un espacio de hipótesis con dimensión VC finita $d \geq 1$. Sea $m \geq$ máx$\{d, T\}$. Para cualquier conjunto S con m muestras, el número de dicotomías inducidas por funciones de C_T está acotado por:* $|\prod_{C_T}(S)| \leq \prod_{C_T}(m) \leq \left(\frac{em}{T}\right)^T \left(\frac{em}{d}\right)^{dT}$.

Demostración. Sea $S = \{x_1, ..., x_m\}$. Como $\mathcal{H}$ tiene dimensión VC finita d, entonces solo puede reproducir un conjunto finito de dicotomías de S. Es decir, solo puede etiquetar los individuos de S de un número finito de formas distintas d. Considérese un conjunto $\mathcal{H}' \subset \mathcal{H}$ que contenga exactamente un *representante* de cada una de esas dicotomías (formas de etiquetar las muestras de S). Para cada $h \in \mathcal{H}$ existe un único $h' \in \mathcal{H}'$ tal que $h(x_i) = h'(x_i)$ para cada individuo $x_i \in S$. Por como se ha definido $\mathcal{H}'$ se tiene que $|\mathcal{H}'| = |\prod_{\mathcal{H}}(S)|$, y por el Lema de Sauer 2.4 junto con la Ecuación (2.9) se tiene que: $|\mathcal{H}'| = |\prod_{\mathcal{H}}(S)| \leq \left(\frac{em}{d}\right)^d$.

Como todas las funciones en $\mathcal{H}$, sin importar como se comportan en S, poseen un *representante* en $\mathcal{H}'$, tomar una secuencia *fija* de clasificadores $h_1, ..., h_T \in \mathcal{H}$ como se hizo en el Lema 2.9, es equivalente a tomarla en $\mathcal{H}'$. El número de hipótesis que pueden escogerse es $|\mathcal{H}'|^T$. Por lo tanto, mediante un argumento análogo al que se hizo en la demostración del Lema 2.9 se tiene que $|\prod_{C_T}(S)| \leq |\mathcal{H}'|^T \cdot \left(\frac{em}{T}\right)^T \leq \left(\frac{em}{d}\right)^{dT} \cdot \left(\frac{em}{T}\right)^T$. □

La modificación del Teorema 2.10 y el Corolario 2.10.1 utilizando el Lema 2.11 es inmediata. En esencia, las cotas que ahora se obtienen reemplazan el factor $\ln|\mathcal{H}|$ por d y algunos términos logarítmicos adicionales.

Teorema 2.12. *Supóngase que AdaBoost se ejecuta durante T iteraciones, utilizando un conjunto de entrenamiento con m muestras aleatorias, y un espacio de hipótesis débiles $\mathcal{H}$ con dimensión VC $d \geq 1$ finita. Supóngase que $m \geq$ máx$\{d, T\}$. Entonces, con probabilidad al menos $1 - \delta$ (sobre la elección del conjunto de entrenamiento), el clasificador combinado H satisface*

$$err(H) \leq \widehat{err}(H) + \sqrt{\frac{32\left(T\ln\left(em/T\right) + d\ln\left(em/d\right) + \ln\left(8/\delta\right)\right)}{m}}.$$

Más aún, si H es consistente con el conjunto de entrenamiento ($\widehat{err}(H) = 0$), entonces con probabilidad al menos $1 - \delta$

$$err(H) \leq \frac{2T(\ln\left(2em/T\right) + d\ln\left(2em/d\right)) + 2\ln\left(2/\delta\right)}{m}.$$

Corolario 2.12.1. *Supóngase que, además de las hipótesis del Teorema 2.12, también se cumple que cada clasificador débil posee un error* $\epsilon_t \leq \frac{1}{2} - \gamma$ *para algún* $\gamma > 0$*. Sea el número* T *igual al número entero más pequeño que es superior al valor* $(\ln m)/(2\gamma^2)$*. Entonces con probabilidad al menos* $1 - \delta$*, el error de generalización del clasificador* H *obtenido al ejecutar el algoritmo AdaBoost durante* T *rondas será como mucho:*

$$O\left(\frac{1}{m}\left[\frac{\ln m}{\gamma^2}\left(\ln m + d\ln\left(\frac{m}{d}\right)\right) + \ln\left(\frac{1}{\delta}\right)\right]\right).$$

Por lo tanto, a modo de resumen, los Teoremas 2.10 y 2.12 muestran que (ignorando factores logarítmicos):

$$\text{err}(H) \leq \widehat{\text{err}}(H) + O\left(\sqrt{\frac{T \cdot C_{\mathcal{H}}}{m}}\right)$$

donde $C_{\mathcal{H}}$ es una medida de la complejidad del espacio de hipótesis $\mathcal{H}$, como su dimensión en el caso finito o su dimensión VC en el caso infinito. Asimismo, si H es consistente, se tiene que:

$$\text{err}(H) \leq O\left(\frac{T \cdot C_{\mathcal{H}}}{m}\right)$$

De igual modo que en los resultados que aparecen al final de la sección 2.1, estas cotas combinan tres factores clave en el diseño de los algoritmos de aprendizaje: la consistencia, la complejidad y el tamaño del conjunto de entrenamiento; aunque ahora se mide la complejidad de la combinación de los T clasificadores débiles mediante el producto $T \cdot C_{\mathcal{H}}$.

Los Corolarios 2.10.1 y 2.12.1 muestran que cuando la complejidad $C_{\mathcal{H}}$ es finita y está fijada, si se satisface la condición de aprendizaje débil, el error de generalización se acerca a cero rápidamente.

2.4. Teoría de los márgenes

En la sección 2.3 se demostraron unas cotas para el error de generalización de un clasificador obtenido por AdaBoost, estas cotas predecían que el clasificador sobreajustaba el conjunto de entrenamiento. Esta predicción, que parece razonable e intuitiva dado el aparente aumento de complejidad del clasificador combinado en cada ronda del boosting, resulta que no es algo muy habitual (Schapire y Freund, 2013). En esta sección se desarrollará una forma alternativa de estudiar el error de generalización de AdaBoost que explica la falta de sobreajuste.

El concepto que se va convertir en la pieza angular de esta sección es la *confianza*, la idea de que un clasificador puede realizar predicciones con mayor certeza que otro. La confianza fue ignorada totalmente en las secciones previas, donde solo se tenía en cuenta el número de clasificaciones incorrectas en el conjunto de entrenamiento (y no se tenía en cuenta la firmeza de las predicciones). Teniendo en cuenta de forma explícita la confianza en el análisis del error de generalización, es posible obtener mejores cotas y además predecir cuando se espera que el clasificador sobreajuste el conjunto de entrenamiento.

Para cuantificar formalmente el concepto de confianza, se introduce el concepto de *margen*. El desarrollo de esta sección consta de dos partes, en primer lugar se probará que márgenes grandes en el conjunto de entrenamiento garantizan una mejor generalización del clasificador, y en segundo lugar se demostrará que AdaBoost, con gran probabilidad, tiende a aumentar los márgenes del conjunto de entrenamiento (incluso cuando el error de entrenamiento alcanza el cero). Por lo tanto ejecutar AdaBoost durante más iteraciones de las que son necesarias para alcanzar un error de entrenamiento nulo hace que el clasificador se vuelva aún más seguro en sus predicciones. Nótese que en este análisis, el número de rondas que ejecuta AdaBoost no tiene gran impacto sobre el error de generalización, en su lugar éste está controlado por los márgenes. Este análisis predice la falta de sobreajuste bajo determinadas circunstancias.

2.4.1. Margen como medida de la confianza

Recuérdese que el clasificador combinado tiene la forma $H(x) = \text{signo}(F(x))$ donde

$$F(x) = \sum_{t=1}^{T} \alpha_t h_t(x).$$

Resulta conveniente, para el posterior análisis, normalizar los pesos de los clasificadores débiles. Sea

$$a_t = \frac{\alpha_t}{\sum_{t'=1}^{T} \alpha_{t'}}$$

y sea

$$f(x) = \sum_{t=1}^{T} a_t h_t(x) = \frac{F(x)}{\sum_{t=1}^{T} \alpha_t}. \tag{2.23}$$

Recuérdese que $\alpha_t > 0 \, \forall t = 1, ..., T$, luego $\sum_{t=1}^{T} \alpha_t > 0$ y por lo tanto $\text{signo}(f(x)) = \text{signo}(F(x))$. Entonces se puede escribir

$$H(x) = \text{signo}(f(x)).$$

Asimismo, se tiene que $\sum_{t=1}^{T} a_t = 1$.

Definición 2.5. Dada una muestra etiquetada (x, y), y un clasificador f, se define el **margen** de esta muestra como el producto $yf(x)$.

Cabe recordar que los clasificadores h_t tienen rango $\{-1, +1\}$, y que también las etiquetas $y \in \{-1, +1\}$. Como los pesos a_t están normalizados, entonces f tiene rango $[-1, +1]$, y en consecuencia el margen de (x, y), $yf(x)$, también está en $[-1, +1]$. Más aún, se tiene que $y = F(x)$ si y solo si el margen de (x, y) es positivo. Por lo tanto, el signo del margen de una muestra indica si esa muestra ha sido correctamente clasificada o no.

El clasificador combinado H puede verse como un comité de clasificadores h_t. Cuando H debe etiquetar un individuo x, convoca el comité de clasificadores y cada uno emite un voto con la predicción de la clase de x, el voto de cada clasificador h_t tiene un peso $a_t > 0$, y $H(x)$ toma el valor de la etiqueta con mayor votación. Otra forma de pensar en el margen de una muestra (x, y) es verlo como la diferencia entre el peso de los clasificadores que predicen correctamente la etiqueta (y) y el peso de los clasificadores que predicen la etiqueta incorrecta $(-y)$. Cuando la votación está muy ajustada, la predicción $H(x)$ se basa en una mayoría muy ajustada, el margen será pequeño en magnitud e intuitivamente se tiene poca confianza en la predicción. Por otro lado, cuando la predicción $H(x)$ se basa en una clara mayoría, entonces el margen correspondiente será grande e intuitivamente se tiene una mayor confianza del resultado predicho. Por lo tanto, la magnitud del margen (o equivalentemente de $f(x)$) de una muestra (x, y) es una medida razonable de la confianza de la clasificación de esa muestra por H. Las interpretaciones de estos rangos de valores del margen pueden verse en el diagrama de la Figura 2.2.

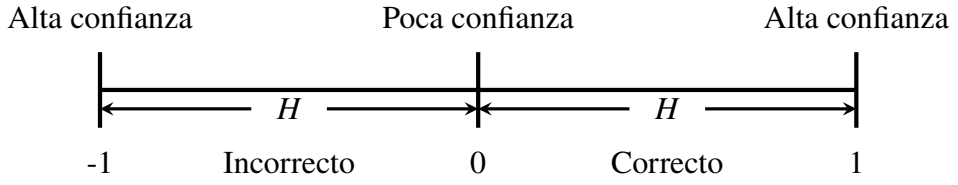


Figura 2.2: Esquema de los valores del margen según la *confianza* del clasificador H

La Figura 2.3 se ha tomado de Schapire y Freund (2013), en ella aparece representada la distribución de los márgenes de las muestras de entrenamiento después de 5, 100, y 1000 iteraciones, indicadas respectivamente mediante curvas de guión corto, guión largo (mayormente oculta bajo la curva sólida) y curva sólida. Este ejemplo muestra el efecto que tiene AdaBoost sobre los márgenes. Aunque se alcance un error de entrenamiento nulo, AdaBoost logra que con el paso de las iteraciones las muestras que tenían márgenes pequeños pasen a tener márgenes positivos más grandes.

A continuación se obtendrá una cota del error de generalización de un clasificador

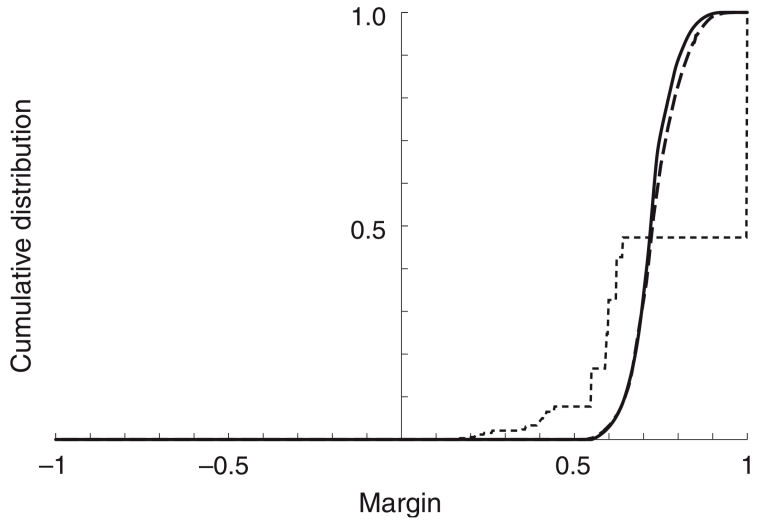


Figura 2.3: Distribución acumulada de los márgenes de las muestras de entrenamiento (Schapire y Freund, 2013)

combinado obtenido por métodos como AdaBoost. Esta cota dependerá en exclusiva de los márgenes de las muestras de entrenamiento, y no dependerá de las iteraciones del algoritmo de boosting. Esta cota predice que AdaBoost no sobreajusta los datos de entrenamiento sin importar cuantas rondas se ejecute (siempre y cuando sea posible alcanzar márgenes grandes, y se cuente con un espacio de hipótesis no muy complejo para el tamaño del conjunto de entrenamiento dado).

Posteriormente se demostrará que AdaBoost, generalmente, tiende a aumentar los márgenes de todos los ejemplos de entrenamiento en cada ronda.

2.4.2. Acotación del error del generalización de AdaBoost basado en los márgenes

Sea $\mathcal{H}$ el espacio de hipótesis, por simplicidad y sin pérdida de generalidad, se asume que es cerrado bajo el operador negación, de modo que si $h \in \mathcal{H}$ entonces $-h \in \mathcal{H}$. Esto permite evitar considerar pesos negativos para los clasificadores débiles.

Definición 2.6. Se define el conjunto $\mathrm{co}(\mathcal{H})$ como el conjunto de aplicaciones que pueden obtenerse tomando una media ponderada de los clasificadores de $\mathcal{H}$:

$$\mathrm{co}(\mathcal{H}) = \left\{ f : x \mapsto \sum_{t=1}^{T} a_t h_t(x) \colon a_1, \ldots, a_T \geq 0; \sum_{t=1}^{T} a_t = 1; h_1, \ldots, h_T \in \mathcal{H}; T \geq 1 \right\}$$

Nótese que la función f generada por AdaBoost pertenece a este espacio.

Como en las secciones previas, $\mathcal{D}$ denota la distribución de muestras de $X \times \{-1, +1\}$ de donde se toma el conjunto de entrenamiento. Puesto que H comete un error al clasificar un individuo si y solo si el margen $yf(x)$ no es positivo, es posible escribir el error de generalización de H en términos de los márgenes: $\mathbf{Pr}_{\mathcal{D}}[yF(x) \leq 0]$. Lo mismo ocurre con el error de entrenamiento: $\mathbf{Pr}_S[yF(x) \leq 0]$.

Los siguientes teoremas establecen que con gran probabilidad, el error de generalización del clasificador ponderado puede acotarse mediante el número de muestras de entrenamiento con margen inferior a cierto valor umbral θ, junto con otro término que depende del número de muestras de entrenamiento, cierta medida de la complejidad del espacio $\mathcal{H}$, y el valor umbral θ. La demostración de estos teoremas puede encontrarse en Schapire y Freund (2013) (p.98-106).

Teorema 2.13 (Caso $\mathcal{H}$ finito)**.** *Sea $\mathcal{D}$ una distribución sobre $X \times \{-1, +1\}$, y sea S un conjunto de entrenamiento con m muestras aleatoriamente escogidas según $\mathcal{D}$. Supongamos que el espacio de hipótesis base $\mathcal{H}$ es finito, y sea $\delta > 0$. Entonces con probabilidad al menos $1 - \delta$ (sobre la elección del conjunto S), todas las funciones $f \in co(\mathcal{H})$ satisfacen la siguiente cota:*

$$\boldsymbol{Pr}_{\mathcal{D}}[yf(x) \leq 0] \leq \Pr_S[yf(x) \leq \theta] + O\left(\sqrt{\frac{\ln|\mathcal{H}|}{m\theta^2} \cdot \ln\left(\frac{m\theta^2}{\ln|\mathcal{H}|}\right) + \frac{\ln(1/\delta)}{m}}\right)$$

para todo $\theta > \sqrt{(\ln|\mathcal{H}|)/(4m)}$.

Este resultado es análogo a los que se obtuvo en los teoremas 2.2 y 2.5. No obstante, esta cota no depende del número de muestras mal clasificadas en el conjunto de entrenamiento, sino de aquellas cuyo margen es inferior a cierto valor umbral θ. Además en esta cota no está presente el número de iteraciones T.

Se cuenta con un resultado análogo para el caso en el que $\mathcal{H}$ es infinito, en este caso se toma su dimensión VC como medida de perplejidad.

Teorema 2.14 (Caso $\mathcal{H}$ infinito)**.** *Sea $\mathcal{D}$ una distribución sobre $X \times \{-1, +1\}$, y sea S un conjunto de entrenamiento con m muestras aleatoriamente escogidas según $\mathcal{D}$. Supongamos que el espacio de hipótesis base $\mathcal{H}$ tiene dimensión VC finita d, y sea $\delta > 0$. Supóngase que $m \geq d \geq 1$ Entonces con probabilidad al menos $1 - \delta$ (sobre la elección del conjunto S), todas las funciones $f \in co(\mathcal{H})$ satisfacen la siguiente cota:*

$$\boldsymbol{Pr}_{\mathcal{D}}[yf(x) \leq 0] \leq \Pr_S[yf(x) \leq \theta] + O\left(\sqrt{\frac{d\ln(m/d)\ln\left(m\theta^2/d\right)}{m\theta^2} + \frac{\ln(1/\delta)}{m}}\right)$$

para todo $\theta > \sqrt{8d \ln (em/d)/(m)}$.

2.4.3. Efecto de AdaBoost sobre la distribución de los márgenes

Basta contemplar las hipótesis de los Teoremas obtenidos en la sección 2.4.2 para concluir no son exclusivos de aquellos clasificadores H obtenidos mediante boosting, sino que pueden ser aplicados también a cualquier método de votación, sin importar como se escojan los clasificadores ni sus pesos. En esta sección se darán evidencias teóricas de que AdaBoost está especialmente diseñado para maximizar el número de muestras con un margen grande. Informalmente, esto se debe a que, en cada ronda, AdaBoost pone más peso en los ejemplos con márgenes más pequeños (los ejemplos que no se clasifican bien o que no lo hacen con mucha certeza).

En el Teorema 2.7 se probó que si el error ϵ_t estaba acotado superiormente por $\frac{1}{2} - \gamma$, para algún $\gamma > 0$, entonces el error de entrenamiento disminuía exponencialmente con el número de clasificadores combinados (esto es, con el número de rondas de AdaBoost). A continuación se probará una resultado más general, se dará una cota sobre la fracción de ejemplos cuyo margen es inferior a un parámetro θ, con $\theta \geq 0$. La acotación resultante vendrá dada en términos de los bordes $\gamma_t = \frac{1}{2} - \epsilon_t$ de las hipótesis débiles, el número de rondas de AdaBoost T, y del parámetro θ. Este resultado muestra que, bajo las mismas condiciones, si θ no es muy grande, entonces la fracción de ejemplos con margen inferior a θ decrece hacia cero exponencialmente como función T.

Nótese que el Teorema 2.7 es un caso particular de este resultado tomando $\theta = 0$.

Teorema 2.15 (Acotación de los márgenes de AdaBoost)**.** *Dada la notación del* ***Algoritmo 1*** *de la página 7, sea* $\gamma_t = \frac{1}{2} - \epsilon_t$. *Entonces la fracción de muestras de entrenamiento con margen inferior o igual a* θ *es como mucho*

$$\prod_{t=1}^{T} \sqrt{(1+2\gamma_t)^{1+\theta}(1-2\gamma_t)^{1-\theta}}.$$

Demostración. Sea f definida como en la Ecuación (2.23). Se tiene que

$$\theta \geq yf(x) \Longleftrightarrow \theta \sum_{t=1}^{T} \alpha_t \geq yf(x) \sum_{t=1}^{T} \alpha_t = yF(x) = y \sum_{t=1}^{T} \alpha_t h_t(x),$$

más aún, se tiene que

$$\theta \geq yf(x) \Leftrightarrow \theta \sum_{t=1}^{T} \alpha_t \geq y \sum_{t=1}^{T} \alpha_t h_t(x) \Leftrightarrow \exp\left(-y \sum_{t=1}^{T} \alpha_t h_t(x) + \theta \sum_{t=1}^{T} \alpha_t\right) \geq 1.$$

Por lo tanto:

$$\mathbf{I}(yf(x) \leq \theta) \leq \exp\left(-y\sum_{t=1}^{T}\alpha_t h_t(x) + \theta\sum_{t=1}^{T}\alpha_t\right),$$

y en consecuencia, la fracción de muestras de entrenamiento con margen inferior o igual a θ es

$$\begin{aligned}
\mathbf{Pr}_S[yf(x) \leq \theta] &= \frac{1}{m}\sum_{i=1}^{m}\mathbf{I}\left\{y_i f\left(x_i\right) \leq \theta\right\} \\
&\leq \frac{1}{m}\sum_{i=1}^{m}\exp\left(-y_i\sum_{t=1}^{T}\alpha_t h_t\left(x_i\right) + \theta\sum_{t=1}^{T}\alpha_t\right) \\
&= \frac{\exp\left(\theta\sum_{t=1}^{T}\alpha_t\right)}{m}\sum_{i=1}^{m}\exp\left(-y_i\sum_{t=1}^{T}\alpha_t h_t\left(x_i\right)\right) \\
&= \exp\left(\theta\sum_{t=1}^{T}\alpha_t\right)\sum_{i=1}^{m}\frac{1}{m}\exp\left(-y_i\sum_{t=1}^{T}\alpha_t h_t\left(x_i\right)\right) \\
&= \exp\left(\theta\sum_{t=1}^{T}\alpha_t\right)\left(\prod_{t=1}^{T}Z_t\right).
\end{aligned}$$

para la última igualdad se ha utilizado el mismo razonamiento que en la demostración del Teorema 2.7, en concreto el argumento que se empleó para (2.16) (tomando $D_1(i) = \frac{1}{m}$).

Finalmente, recordando la expresión de Z_t obtenida en (2.18), así como las definiciones $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$, y $\gamma_t = \frac{1}{2} - \epsilon_t$:

$$\begin{aligned}
\mathbf{Pr}_S[yf(x) \leq \theta] &\leq \exp\left(\theta\sum_{t=1}^{T}\alpha_t\right)\left(\prod_{t=1}^{T}Z_t\right) \\
&= \exp\left(\theta\sum_{t=1}^{T}\frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}\right)\left(\prod_{t=1}^{T}Z_t\right) \\
&= \exp\left(\frac{\theta}{2}\ln\left(\prod_{t=1}^{T}\frac{1-\epsilon_t}{\epsilon_t}\right)\right)\left(\prod_{t=1}^{T}Z_t\right) \\
&= \left(\prod_{t=1}^{T}\frac{1-\epsilon_t}{\epsilon_t}\right)^{\frac{\theta}{2}}\left(\prod_{t=1}^{T}Z_t\right) \\
&= \left(\prod_{t=1}^{T}\frac{1/2+\gamma_t}{1/2-\gamma_t}\right)^{\frac{\theta}{2}}\left(\prod_{t=1}^{T}Z_t\right) \\
&= \left(\prod_{t=1}^{T}\frac{1+2\gamma_t}{1-2\gamma_t}\right)^{\frac{\theta}{2}}\left(\prod_{t=1}^{T}(1-2\gamma_t)(1+2\gamma_t)\right)^{\frac{1}{2}} \\
&= \prod_{t=1}^{T}\sqrt{\left((1-2\gamma_t)^{1+\theta}(1+2\gamma_t)^{1-\theta}\right)}
\end{aligned}$$

□

Para tener una idea de lo que esta cota significa, considérese que ocurre si para algún $\gamma > 0$ se cumple que $\epsilon_t \leq \frac{1}{2} - \gamma$. En ese caso, la cota del Teorema 2.15 puede simplificarse a

$$\left(\sqrt{(1+2\gamma)^{1+\theta}(1-2\gamma)^{1-\theta}}\right)^T,$$

entonces cuando

$$\sqrt{(1+2\gamma)^{1+\theta}(1-2\gamma)^{1-\theta}} < 1 \tag{2.24}$$

esta cota implica que la fracción de muestras de entrenamiento con margen $yf(x) \leq \theta$ decrece hacia cero exponencialmente con T, de hecho debe alcanzar el cero ya que esta fracción debe ser siempre un múltiplo de $1/m$. Más aún, despejando θ en la Ecuación (2.24) se tiene que:

$$\sqrt{(1+2\gamma)^{1+\theta}(1-2\gamma)^{1-\theta}} < 1 \Leftrightarrow \theta < \Upsilon(\gamma)$$

donde

$$\Upsilon(\gamma) = \frac{-\ln(1-4\gamma^2)}{\ln\left(\frac{1+2\gamma}{1-2\gamma}\right)}.$$

Esta función se ha representado en la Figura 2.4, junto con las rectas γ y 2γ. En la figura puede verse que $\gamma \leq \Upsilon(\gamma) \leq 2\gamma$ para $0 \leq \gamma < \frac{1}{2}$, y que $\Upsilon(\gamma)$ se acerca a γ cuando γ es pequeño. De este modo queda visto que si todas las hipótesis débiles tienen un borde mayor o igual que γ, como la fracción de muestras con margen $\theta < \Upsilon(\gamma)$ decrece hacia cero con el número de iteraciones T, se tiene que para valores grandes de T todas las muestras de entrenamiento tendrán margen θ mayor o igual que $\Upsilon(\gamma) > \gamma$. En este sentido, $\Upsilon(\gamma)$ acota el mínimo valor que puede tener el margen de una muestra de entrenamiento como función del menor borde γ.

Por lo tanto, esto quiere decir que cuando los clasificadores débiles sean considerablemente mejores que clasificadores cuyas predicciones son aleatorias, está garantizado que los márgenes de los ejemplos de entrenamiento serán grandes después de un número suficiente de iteraciones.

Esto sugiere que clasificadores fuertes, que realizan predicciones precisas y por lo tanto poseen bordes grandes, darán lugar a márgenes grandes y cabría esperar que funcionasen correctamente como hipótesis base en algoritmos de boosting. No obstante, estos clasificadores fuertes generalmente poseen una complejidad mayor que los clasificadores débiles, y esta complejidad, de acuerdo con los Teoremas 2.13 y 2.14, muy probablemente perjudicará la actuación del clasificador combinado resultante del proceso de boosting.

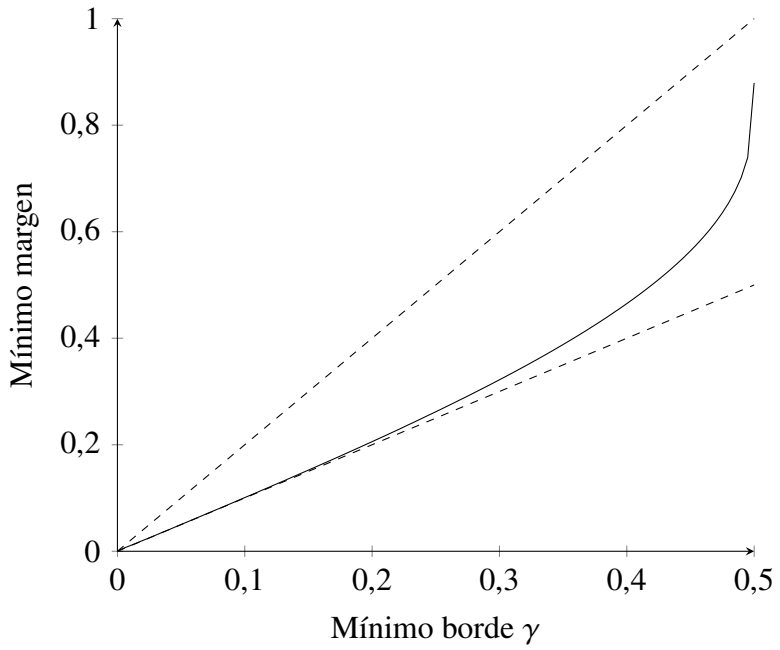


Figura 2.4: Cota inferior de los márgenes para valores grandes de T

De nuevo se observa el balance entre la complejidad (de los clasificadores base) y la consistencia (medida por los bordes de los clasificadores débiles).

Capítulo 3

Optimización de una función de coste

En este capítulo se mostrará que muchos de los algoritmos de boosting pueden verse como la optimización de una función de coste adecuada. Generalmente la función de coste dependerá del margen $yf(x)$. En relación con este tipo de funciones de coste, se presentará el algoritmo MarginBoost, se trata de un algoritmo que en cada iteración busca un clasificador que disminuya al máximo dicha función de coste. A grandes rasgos, la idea detrás de MarginBoost consiste en escoger, en cada iteración, un clasificador que apunte en la dirección descendente del gradiente de la función de coste. Esta idea de escoger los clasificadores en función del gradiente de una función de coste se debe a Leo Breiman (Breiman, 1999).

La potenciación del gradiente que se presenta en este capítulo viene implementada en los paquetes `gbm` *Gradient Boosted Models* y `xgboost` *Extreme Gradient Boosting* de R. Ambos están disponible de forma gratuita.

La teoría que se desarrolla en este capítulo se nutre del trabajo de Mason *et al.* (1999).

Manteniendo la notación que se venía empleando en los anteriores capítulos, (x, y) son muestras generadas aleatoriamente de acuerdo a cierta distribución $\mathcal{D}$ sobre $X \times Y$, donde X es el espacio de los individuos (medidas) e $Y = \{-1, 1\}$ es el espacio de las etiquetas.

Este capítulo se centra en el análisis de clasificadores combinados de la forma $H(x) = \text{signo}(F(x))$ donde

$$F(x) = \sum_{t=1}^{T} \alpha_t h_t(x),$$

$h_t \colon X \to \{-1, 1\}$ son clasificadores débiles pertenecientes a una clase $\mathcal{H}$ y $\alpha_t \in \mathbb{R}$.

Dado un conjunto de entrenamiento $S = \{(x_1, y_1), ..., (x_m, y_m)\}$ obtenido aleatoriamente según la distribución $\mathcal{D}$, se desea construir un clasificador H (como el descrito previamente) tal que la probabilidad de que clasifique incorrectamente un individuo aleatorio $\mathbf{Pr}_{\mathcal{D}}(H(x) \neq y)$ sea pequeña. En este caso, en lugar de construir el clasificador en base al error de entrenamiento $\mathbf{Pr}_S(H(x) \neq y)$, como se venía haciendo en los capítulos anteriores, se construirá H en base a una función de coste C que depende del margen $yF(x)$. Es decir, para un conjunto de entrenamiento S con m ejemplos, se busca F tal que

$$C(F) = \frac{1}{m} \sum_{i=1}^{m} C(y_i F(x_i)) \tag{3.1}$$

sea mínimo, para cierta función de coste $C : \mathbb{R} \to \mathbb{R}$. Nótese que se está empleando el símbolo C para denotar la cantidad que se busca minimizar (el coste $C(F)$), así como la propia función de coste $C : \mathbb{R} \to \mathbb{R}$. La interpretación del símbolo C debería estar siempre clara si se tiene en cuenta el contexto.

3.1. AnyBoost

Una forma de obtener un clasificador combinado que optimice (3.1) es mediante un algoritmo de gradiente descendente. Esta idea fue presentada en primer lugar en Breiman (1999), no obstante los autores de Mason *et al.* (1999) proponen un análisis más abstracto considerando el gradiente descendente dentro de un espacio con un producto interno definido. Se considera el espacio de todas las combinaciones lineales de $\mathcal{H}$, $\text{lin}(\mathcal{H})$, y el producto interno se define como

$$\langle F, G \rangle = \frac{1}{m} \sum_{i=1}^{m} F(x_i) G(x_i) \tag{3.2}$$

para todas $F, G \in \text{lin}(\mathcal{H})$. No obstante, el algoritmo AnyBoost que se define en esta sección es válido para cualquier función de coste y producto interno. Cuando se escoge como función de coste una función como (3.1), y como producto interno uno como (3.2) entonces AnyBoost da lugar a MarginBoost, que será objeto de estudio en la siguiente sección.

Ahora supóngase que se tiene una función $F \in \text{lin}(\mathcal{H})$ y que se desea hallar una nueva función $h \in \mathcal{H}$ para añadírsela a F y que de este modo el coste $C(F + \epsilon h)$ disminuya, para algún valor de ϵ pequeño. En otras palabras, se busca la "dirección" h tal que $C(F + \epsilon h)$ decrece más rápido. Viendo el coste $C(F)$ como una función sobre $\text{lin}(\mathcal{H})$, la dirección que se busca es $-\nabla C(F)(x)$, donde

$$\nabla C(F)(x) := \left. \frac{\partial C\,(F + \alpha 1_x)}{\partial \alpha} \right|_{\alpha=0},$$

y la función 1_x es la función indicadora de x. Puesto que que h debe pertenecer a $\text{lin}(\mathcal{H})$, por lo general no será posible escoger $h = -\nabla C(F)$, por ello en su lugar se busca la función $h \in \text{lin}(\mathcal{H})$ con el mayor producto interno con $-\nabla C(F)$. Esto es, se busca maximizar

$$-\langle \nabla C(F), h \rangle.$$

Esto puede comprenderse mejor si uno observa que aproximando a primer orden en ϵ,

$$C(F + \epsilon h) \approx C(F) + \epsilon \langle \nabla C(F), h \rangle$$

y por lo tanto la mayor reducción en el coste ocurrirá cuando se escoja h maximizando $-\langle \nabla C(F), h \rangle$.

Lo expuesto previamente motiva el desarrollo del **Algoritmo 2**, un algoritmo iterativo que busca combinaciones lineales F de hipótesis de $\mathcal{H}$ que minimicen el coste $C(F)$. Nótese que no se especifica el conjunto de etiquetas Y, ni se restringe la forma del producto interno ni de la función de coste, tampoco se especifica cómo se definen los pesos α_t (también conocidos como *pasos*). Nótese también que el algoritmo termina cuando $-\langle \nabla C(F_t), h_{t+1} \rangle \leq 0$, es decir, cuando el algoritmo de aprendizaje débil A devuelve una hipótesis h_{t+1} que ya no apunta en la dirección descendente de la función coste $C(F)$. Por lo tanto, el algoritmo termina cuando, aproximando a primer orden, un paso en la dirección de la hipótesis proporcionada por A aumentaría el coste.

Algoritmo 2: AnyBoost

Entrada:

- Un espacio con producto interno $(S, \langle , \rangle)$ conteniendo funciones $f : X \to Y$.
- Una clase de hipótesis base $\mathcal{H} \subset S$.
- Una función de coste diferenciable $C : \text{lin}(\mathcal{H}) \to \mathbb{R}$.
- Un algoritmo de aprendizaje débil A que tomo como entrada $F \in \text{lin}(\mathcal{H})$ y devuelva $h \in \mathcal{H}$ con un gran valor de $-\langle \nabla C(F), h \rangle$.

inicio

  $F_0(x) = 0$.

para $t = 0, ..., T$ **hacer**

  Sea $h_t = A(F_t)$.

  si $-\langle \nabla C(F_t), h_t \rangle \leq 0$ **entonces**

    Salida: F_t

  Escoge el paso α_t.

  Sea $F_{t+1} = F_t + \alpha_t h_t$

Salida: Hipótesis final: F_{T+1}.

Si bien AnyBoost puede devolver cualquier combinación lineal de hipótesis de $\mathcal{H}$, esta flexibilidad tiene el potencial de causar sobreajuste. En Mason *et al.* (1999) se propone una ligera modificación de AnyBoost llamada AntBoost.L_1, consiste en restringir el espacio $\text{lin}(\mathcal{H})$ al espacio $\text{co}(\mathcal{H})$ que son las combinaciones *convexas* de elementos de $\mathcal{H}$.

3.2. MarginBoost

Cuando en el algoritmo AnyBoost se escoge una función de coste $C\colon \text{lin}(\mathcal{H}) \to \mathbb{R}$ como (3.1), que depende del margen de las muestras, un producto interno como (3.2), y se restringe Y al conjunto $\{-1, +1\}$ AnyBoost da lugar al algoritmo MarginBoost. En este caso,

$$\nabla C(F)(x) = \begin{cases} 0 & \text{si } x \neq x_i, i = 1 \ldots m \\ \frac{1}{m} y_i C'\left(y_i F\left(x_i\right)\right) & \text{si } x = x_i \end{cases} \tag{3.3}$$

donde $C'(z)$ es la derivada respecto de z de la función de coste $C : \mathbb{R} \to \mathbb{R}$. Por lo tanto,

$$-\langle \nabla C(F), h \rangle = -\frac{1}{m^2} \sum_{i=1}^{m} y_i h\left(x_i\right) C'\left(y_i F\left(x_i\right)\right) \tag{3.4}$$

.

En la sección 2.4 se vio que obtener márgenes grandes y positivos era sinónimo de obtener predicciones que con gran certeza serían correctas. Por ello una función de coste que dependa del margen debe penalizar sobre todo a los clasificadores que produzcan márgenes negativos, y también a aquellos que produzcan márgenes pequeños (clasificadores poco fiables), favoreciendo a los clasificadores que producen márgenes grandes y positivos. En este sentido cualquier función de coste de este tipo será monótona decreciente, y entonces $-C'\left(y_i F\left(x_i\right)\right)$ será siempre positiva.

Dividiendo $-C'\left(y_i F\left(x_i\right)\right)$ entre $-\sum_{i=1}^{m} C'\left(y_i F\left(x_i\right)\right)$ se aprecia que hallar una hipótesis h que maximice $-\langle \nabla C(F), h \rangle$ es equivalente a obtener h que minimice el error *ponderado*

$$\sum_{i=1}^{m} D(i) \mathbf{I}(f(x_i) \neq y_i)$$

donde $D(1), ..., D(m)$ es la distribución

$$D(i) = \frac{C'\left(y_i F\left(x_i\right)\right)}{\sum_{i=1}^{m} C'\left(y_i F\left(x_i\right)\right)}. \tag{3.5}$$

Para comprender esto basta con notar que en (3.4), como $-C'\left(y_i F\left(x_i\right)\right) > 0$, los individuos correctamente clasificados ($y_i h(x_i) = 1$) contribuyen positivamente al valor de

$-\langle \nabla C(F), h \rangle$, mientras que los individuos clasificados incorrectamente ($y_i h(x_i) = -1$) contribuyen negativamente. Evidentemente MarginBoost debe prestar más atención a aquellas muestras (x_i, y_i) para las cuales $-C'(y_i F(x_i))$ toma valores más grandes. Por ello es razonable considerar un error ponderado donde a cada ejemplo de entrenamiento (x_i, y_i) se le asigna el peso $-C'(y_i F(x_i))$, penalizando en mayor medida los errores cometidos al clasificar un individuo con un peso elevado. La clave reside en que no todos los individuos son igual de importantes, y que esta importancia depende de la iteración. El hecho de dividir los pesos entre $\sum_{i=1}^{m} -C'(y_i F(x_i))$ es para lograr que $\sum_{i=1}^{m} D(i) = 1$.

Realizando los cambios pertinentes sobre el pseudocódigo del **Algoritmo 2** AnyBoost se obtiene el pseudo-código de MarginBoost: **Algoritmo 3**.

Algoritmo 3: MarginBoost

Entrada:

- Una función de coste diferenciable $C: \mathbb{R} \to \mathbb{R}$.
- Una clase de hipótesis base $\mathcal{H}$ conteniendo funciones $h: X \to \{1, -1\}$.
- Un conjunto de entrenamiento $S = \{(x_1, y_1), ..., (x_m, y_m)\}$ con cada $(x_i, y_i) \in X \times \{1, -1\}$.
- Un algoritmo de aprendizaje débil A que tomo como entrada un conjunto de entrenamiento S y una distribución D sobre el conjunto de entrenamiento, y devuelva hipótesis base $h \in \mathcal{H}$ con un error ponderado pequeño $\sum_{i=1}^{m} D(i)\mathbf{I}(f(x_i) \neq y_i)$.

inicio

$F_0(x) = 0$.

$D_0(i) = \frac{1}{m}$ para $i = 1, ..., m$.

para $t = 0, ..., T$ **hacer**

Sea $h_t = A(S, D_t)$.

si $\sum_{i=1}^{m} D_t(i) y_i h_t(x_i) \leq 0$ **entonces**

Salida: F_t

Escoge el paso α_t.

Sea $F_{t+1} = F_t + \alpha_t h_t$

Sea $D_{t+1}(i) = \frac{C'(y_i F_{t+1}(x_i))}{\sum_{i=1}^{m} C'(y_i F_{t+1}(x_i))}$ para $i = 1, ..., m$.

Salida: Hipótesis final: F_{T+1}.

3.3. Boosting visto como gradiente descendente

Muchos de los exitosos algoritmos de boosting son, eligiendo cuidadosamente la función de coste y la forma de los pasos α_t, casos concretos del algoritmo AnyBoost descrito anteriormente (o sus derivados).

A continuación se probará que AdaBoost, **Algoritmo 1**, es de hecho MarginBoost tomando la función de coste $C(z) = \exp(-z)$. Para ello debemos probar que las distribuciones D_t, los pesos α_t y el criterio de detención de ambos algoritmos es el mismo.

Por un lado, en (2.15) se vio que tomando $D_1(i) = \frac{1}{m}$ para $i = 1, ..., m$:

$$D_{T+1}(i) = \frac{\frac{1}{m} \exp\left(-y_i F\left(x_i\right)\right)}{\prod_{t=1}^{T} Z_t},$$

y en (2.16) se vio que también tomando $D_1(i) = \frac{1}{m}$ para $i = 1, ..., m$:

$$\prod_{t=1}^{T} Z_t = \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i F(x_i)),$$

luego

$$D_{T+1}(i) = \frac{\exp\left(-y_i F\left(x_i\right)\right)}{\sum_{i=1}^{m} \exp(-y_i F(x_i))} \tag{3.6}$$

que es precisamente la distribución de MarginBoost (3.5) para la función de coste $C(z) = \exp(-z)$.

Por otro lado, el criterio de detención de AdaBoost es

$$\sum_{i=1}^{m} D_t(i)\mathbf{I}(h_t(x_i) \neq y_i) \geq \frac{1}{2}.$$

Esto es equivalente a

$$\sum_{i=1}^{m} D_t(i)\mathbf{I}(h_t(x_i) = y_i) - \sum_{i=1}^{m} D_t(i)\mathbf{I}(h_t(x_i) \neq y_i) = \sum_{i=1}^{m} D_t(i) y_i h_t(x_i) \leq 0,$$

que es precisamente el criterio de detención de MarginBoost.

Por último, en MarginBoost una vez se ha escogido h_t se desea hallar el valor de α_t que minimice

$$\sum_{i=1}^{m} C\left(y_i F_t\left(x_i\right) + y_i \alpha_t h_t\left(x_i\right)\right) = \sum_{i=1}^{m} \exp\left(-y_i F_t\left(x_i\right) - y_i \alpha_t h_t\left(x_i\right)\right).$$

Esta forma de proceder se conoce como *búsqueda de línea*. Derivando la expresión anterior respecto de α_t e igualando a 0:

$$0 = -y_i h_t(x_i) \sum_{i=1}^{m} \exp\left(-y_i F_t(x_i) - y_i \alpha_t h_t(x_i)\right)$$

se tiene la siguiente igualdad

$$\sum_{i=1}^{m} \exp\left(-y_i F_t(x_i) - \alpha_t\right) \mathbf{I}(h_t(x_i) = y_i) = \sum_{i=1}^{m} \exp\left(-y_i F_t(x_i) + \alpha_t\right) \mathbf{I}(h_t(x_i) \neq y_i),$$

multiplicando ambos lados de la igualdad por $\exp(\alpha_t) / \sum_{i=1}^{m} \exp\left(-y_i F_t(x_i)\right)$ se llega a

$$\sum_{i=1}^{m} \frac{\exp\left(-y_i F_t(x_i)\right)}{\sum_{i=1}^{m} \exp\left(-y_i F_t(x_i)\right)} \mathbf{I}(h_t(x_i) = y_i) = \exp(2\alpha_t) \sum_{i=1}^{m} \frac{\exp\left(-y_i F_t(x_i)\right)}{\sum_{i=1}^{m} \exp\left(-y_i F_t(x_i)\right)} \mathbf{I}(h_t(x_i) \neq y_i),$$

y recordando como se definió $D_t(i)$ en **Algoritmo 3**, particularizando para la función de coste $C(z) = \exp(-z)$, esto último puede escribirse como

$$\sum_{i=1}^{m} D_t(i) \mathbf{I}(h_t(x_i) = y_i) = \exp(2\alpha_t) \sum_{i=1}^{m} D_t(i) \mathbf{I}(h_t(x_i) \neq y_i).$$

Finalmente despejando α_t se obtiene

$$\alpha_t = \frac{1}{2} \ln\left(\frac{\sum_{i=1}^{m} D_t(i) \mathbf{I}(h_t(x_i) = y_i)}{\sum_{i=1}^{m} D_t(i) \mathbf{I}(h_t(x_i) \neq y_i)}\right)$$

que, recordando la definición de ϵ_t (1.2), es precisamente como se definían los pesos en el **Algoritmo 1** AdaBoost.

Se concluye que AdaBoost está realizando un gradiente descendente sobre la función de coste

$$C(F) = \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i F(x_i))$$

con pasos α_t escogidos mediante búsqueda de línea.

A continuación se comentan otros algoritmos de boosting que también pueden ser vistos como potenciadores de gradiente descendente de determinadas funciones de coste.

En Schapire y Singer (1999) se analiza AdaBoost desde una perspectiva más amplia, considerando clasificadores base que pueden devolver valores reales en el intervalo $[-1, 1]$. Describen el algoritmo ConfidentBoost que es esencialmente Anyboost con la función de coste $C(z) = \exp(-z)$ y clasificadores base $h : X \to [-1, 1]$.

En Breiman (1996) se describe el algoritmo ARC-X4. Éste es Anyboost.L_1 con la función de coste $C(z) = (1 - z)^5$ y tomando pasos descendentes $\alpha_t = 1/t$.

En Friedman *et al.* (1998) se estudia AdaBoost como una aproximación de estimación de máxima verisimilitud. Desde esta perspectiva, desarrollan una aproximación más directa conocida como LogitBoost que exhibe un comportamiento similar a AdaBoost. LogitBoost es AnyBoost tomando como función de coste $C(z) = \log_2(1-\exp(-2z))$ y pasos obtenidos mediante el método de Newton-Raphson.

La siguiente tabla recoge las funciones de coste y el método de obtención de los pasos α_t para algunos algoritmos de boosting que pueden verse como casos particulares de AnyBoost y sus derivados (Mason *et al.*, 2000).

Algoritmo	**Función de coste**	**Paso**
AdaBoost	$e^{-yF(x)}$	Búsqueda de línea
ARC-X4	$(1 - yF(x))^5$	$1/t$
ConfidenceBoost	$e^{-yF(x)}$	Búsqueda de línea
LogitBoost	$\ln\left(1 + e^{-yF(x)}\right)$	Newton-Raphson

3.4. Resultados de convergencia

En esta sección se demuestran dos teoremas relacionados con la convergencia de AnyBoost imponiendo ciertas condiciones a la función de coste C. Estos resultados proporcionan una receta para los pasos α_t que garantiza la convergencia del algoritmo, sin embargo en la práctica normalmente se escogen pasos más pequeños de lo necesario (Mason *et al.*, 1999).

El siguiente teorema proporciona un paso α_t específico para AnyBoost y caracteriza el comportamiento límite del algoritmo para ese paso concreto.

Teorema 3.1. *Sea C: $lin(\mathcal{H}) \to \mathbb{R}$ cualquier función de coste acotada inferiormente y Lipschitz diferenciable (es decir, existe L tal que $\|\nabla C(F) - \nabla C(F')\| \leq L\|F - F'\|$ para todas $F, F' \in lin(\mathcal{H})$). Sean $F_0, F_1, \ldots$ la secuencia de hipótesis combinadas obtenidas en cada iteración del algoritmo AnyBoost utilizando pasos*

$$\alpha_{t+1} = -\frac{\langle \nabla C(F_t), h_{t+1} \rangle}{L\|h_{t+1}\|^2}.$$

Entonces, o bien AnyBoost se detiene en la ronda T con $-\langle \nabla C(F_T), h_{T+1} \rangle \leq 0$, o bien $C(F_t)$ converge a algún valor finito C^*, en cuyo caso

$$\lim_{t\to\infty} \langle \nabla C(F_t), h_{t+1} \rangle = 0.$$

Demostración. En primer lugar es necesario probar el siguiente Lema de carácter general:

Lema 3.2. *Sea* $(\mathcal{H}, \langle , \rangle)$ *un espacio con producto interno cuya norma viene dada por* $\|F\|^2 = \langle F, F\rangle$*, y sea* $C\colon \mathcal{H} \to \mathbb{R}$ *cualquier función de coste con* $\|\nabla C(F) - \nabla C(F')\| \leq L\|F - F'\|$ *para todas* $F, F' \in \mathcal{H}$*, para cierto* $L > 0$*. Entonces, para cualquier* $\alpha > 0$*, y* $F, G \in \mathcal{H}$

$$C(F + \alpha G) - C(F) \leq \alpha\langle \nabla C(F), G\rangle + \frac{L\alpha^2}{2}\|G\|^2.$$

Demostración. Se define $g : \mathbb{R} \to \mathbb{R}$ por $g(\alpha) := C(F + \alpha G)$. Su derivada se obtiene de forma inmediata aplicando la regla de la cadena, $g'(\alpha) = \langle \nabla C(F + \alpha G), G\rangle$, y por lo tanto

$$\begin{aligned} |g'(\alpha) - g'(0)| &= |\langle \nabla C(F + \alpha G) - \nabla C(F), G\rangle| \\ &\leq \|\nabla C(F + \alpha G) - \nabla C(F)\|\|G\| \\ &\leq L\|(F + \alpha G) - F\|\|G\| \\ &= L\alpha\|G\|^2 \end{aligned}$$

en la primera igualdad se ha utilizado la linealidad del producto interno, en la primera desigualdad se ha empleado la desigualdad de Cauchy-Schwartz, y en la segunda desigualdad se ha recurrido a la condición impuesta sobre la función C en el enunciado ($F + \alpha G$ juega el papel de F'). Se tiene entonces que

$$g'(\alpha) \leq g'(0) + L\alpha\|G\|^2 = \langle \nabla C(F), G\rangle + L\alpha\|G\|^2,$$

y finalmente esto implica que

$$\begin{aligned} C(F + \alpha G) - C(F) &= g(\alpha) - g(0) \\ &= \int_0^\alpha g'(w)dw \\ &\leq \int_0^\alpha \langle \nabla C(F), G\rangle + Lw\|G\|^2 dw \\ &= \alpha\langle \nabla C(F), G\rangle + L\frac{\alpha^2}{2}\|G\|^2. \end{aligned}$$

□

Ahora utilizando el resultado del Lema 3.2 se puede escribir

$$\begin{aligned} C(F_t) - C(F_{t+1}) &= C(F_t) - C(F_t + \alpha_{t+1}h_{t+1}) \\ &\geq -\alpha_{t+1}\langle \nabla C(F_t), h_{t+1}\rangle - L\frac{\alpha_{t+1}^2}{2}\|h_{t+1}\|^2. \end{aligned} \tag{3.7}$$

Si $\|h_{t+1}\| = 0$, entonces h_{t+1} es la función nula y $\langle \nabla C(F_t), h_{t+1}\rangle = 0$ por lo que el algoritmo AnyBoost terminaría en esa iteración. Supóngase que $\|h_{t+1}\| \neq 0$, sea $g : \mathbb{R} \to \mathbb{R}$ la función definida por

$$g(\alpha) = -\alpha\langle \nabla C(F_t), h_{t+1}\rangle - L\frac{\alpha^2}{2}\|h_{t+1}\|^2,$$

derivando e igualando a cero se determinan los puntos singulares de la función,

$$g'(\alpha^*) = -\langle \nabla C(F_t), h_{t+1} \rangle - L\alpha^* \|h_{t+1}\|^2 = 0 \Leftrightarrow \alpha^* = -\frac{\langle \nabla C(F_t), h_{t+1} \rangle}{L|h_{t+1}\|^2},$$

evaluando este valor de α en la segunda derivada de g,

$$g''(\alpha) = -L\|h_{t+1}\|^2 < 0 \Rightarrow g''(\alpha^*) < 0,$$

se deduce que $g(\alpha)$ alcanza su valor máximo en α^*. En consecuencia, el lado derecho de la desigualdad (3.7) es máximo para

$$\alpha_{t+1} = -\frac{\langle \nabla C(F_t), h_{t+1} \rangle}{L\|h_{t+1}\|^2},$$

que es el paso que aparece en el enunciado del teorema. Por lo tanto, para este α_{t+1} se tiene que

$$C(F_t) - C(F_{t+1}) \geq \frac{\langle \nabla C(F_t), h_{t+1} \rangle^2}{2L\|h_{t+1}\|^2}, \tag{3.8}$$

Si en alguna iteración resulta que $-\langle \nabla C(F_T), h_{T+1} \rangle \leq 0$ entonces AnyBoost se detiene y devuelve como hipótesis final F_T. En otro caso, en cada iteración se tiene que $-\langle \nabla C(F_t), h_{t+1} \rangle > 0$, luego en cada iteración se puede hallar una hipótesis $F_{t+1} = F_t + \alpha_{t+1} h_{t+1}$ que reduzca el coste: $C(F_{t+1}) < C(F_t)$. Ahora bien, puesto que C está acotada inferiormente, resulta que $C(F_t) - C(F_{t+1}) \to 0$. Esto último, por (3.8), implica que $\langle \nabla C(F_t), h_{t+1} \rangle \to 0$, como se quería demostrar. □

El siguiente resultado muestra que si el algoritmo de aprendizaje débil es capaz de hallar en cada iteración de AnyBoost la mejor hipótesis $h_t \in \mathcal{H}$, y si la función de coste C es convexa, entonces AnyBoost converge al mínimo global de la función. Por simplicidad, en el siguiente teorema se supone que en lugar de terminar cuando $-\langle \nabla C(F_T), h_{T+1} \rangle \leq 0$, AnyBoost simplemente continúa devolviendo F_T para todo $t > T$.

Teorema 3.3. *Sea $C : lin(\mathcal{H}) \to \mathbb{R}$ una función convexa con las propiedades del Teorema 3.1, sea $(F_t)_t$ la sucesión de hipótesis generadas por AnyBoost utilizando los pasos α_t que se definen en el Teorema 3.1. Se asume que el espacio $\mathcal{H}$ es cerrado bajo negación: $h \in \mathcal{H} \Rightarrow -h \in \mathcal{H}$, y también se asume que en cada ronda el algoritmo de aprendizaje débil encuentra una función h_{t+1} que maximiza $-\langle \nabla C(F_t), h_{t+1} \rangle$. Entonces, cualquier punto de acumulación F de la sucesión $(F_t)_t$ cumple*

$$\sup_{h \in \mathcal{H}} -\langle \nabla C(F), h \rangle = 0, \tag{3.9}$$

y

$$C(F) = \inf_{G \in lin(\mathcal{H})} C(G). \tag{3.10}$$

Más aún,

$$\lim_{t \to \infty} C(F_t) = \inf_{G \in lin(\mathcal{H})} C(G). \tag{3.11}$$

Demostración. Sea F un punto de acumulación de la sucesión $(F_t)_t$, razonando por reducción al absurdo, supóngase que $\sup_{h\in\mathcal{H}} -\langle \nabla C(F), h\rangle = \epsilon > 0$. De la continuidad de C, y por ser F un punto de acumulación de la sucesión, se deduce que existirán infinitas funciones F_t tal que $\sup_{h_t\in\mathcal{H}} -\langle \nabla C(F_t), h_{t+1}\rangle > \epsilon/2$. Ahora bien, aplicando (3.8) de forma recursiva se tiene que

$$\begin{aligned} C(F_{t+1}) &\leq C(F_t) - \frac{\langle \nabla C(F_t), h_{t+1}\rangle^2}{2L|h_{t+1}\|^2} \\ &\leq C(F_{t-1}) - \frac{\langle \nabla C(F_{t-1}), h_t\rangle^2}{2L|h_t\|^2} - \frac{\langle \nabla C(F_t), h_{t+1}\rangle^2}{2L|h_{t+1}\|^2} \\ &\leq \dots \end{aligned}$$

teniendo en cuenta que (por hipótesis) en cada iteración el algoritmo de aprendizaje débil encuentra la función $h_{t+1} \in \mathcal{H}$ que maximiza $-\langle \nabla C(F_t), h_{t+1}\rangle$, y que existen infinitos términos de la sucesión $(F_t)_t$ cumpliendo $\sup_{h_t\in\mathcal{H}} -\langle \nabla C(F_t), h_{t+1}\rangle > \epsilon/2$, se concluye que $C(F_t) \to -\infty$. Esto último es absurdo pues C está acotada inferiormente.

Para probar (3.10) también se razona por reducción al absurdo, se supone que existe $G \in \text{lin}(\mathcal{H})$ tal que $C(F) > C(G)$. Recordando que una función f definida sobre un dominio convexo es convexa si y solo si $f(x) - f(y) \geq \nabla f(y)\cdot(x-y)$ para todo x, y en su dominio, de la convexidad de C y su dominio, se tiene que:

$$0 > C(G) - C(F) \geq \langle G - F, \nabla C(F)\rangle. \tag{3.12}$$

Puesto que $F, G \in \text{lin}(\mathcal{H})$, $G - F = \sum_i w_i h_i$ para ciertos coeficientes w_i y elementos $h_i \in \mathcal{H}$. Por (3.12) $\langle G - F, \nabla C(F)\rangle < 0$ y al ser $\mathcal{H}$ cerrado bajo negación, implica que existe $h_i \in \mathcal{H}$ tal que $-\langle h_i, \nabla C(F)\rangle > 0$, contradiciendo (3.9). Con esto queda probado que $C(F)$ es una cota inferior del conjunto $\{C(G) : G \in \text{lin}(\mathcal{H})\}$, falta por demostrar que es la mayor de las cotas inferiores y con ello (3.10) quedaría probado. Razonando de nuevo por reducción al absurdo, supongamos que existe F^* tal que $C(F) < C(F^*)$ y $C(F^*)$ es una cota inferior del conjunto $\{C(G) : G \in \text{lin}(\mathcal{H})\}$. Sea $\epsilon = C(F^*) - C(F)$, por hipótesis $\epsilon > 0$. Como C es continua en F, y F es un punto de acumulación de $(F_t)_t$, existe un elemento de la sucesión $\widehat{F}_t$ tal que $\|C(F) - C(\widehat{F}_t)\| < \epsilon/2$. Entonces se tiene que por un lado

$$C(F) = C(F^*) + \epsilon,$$

y por otro lado

$$C(\widehat{F}_t) < C(F) + \epsilon/2$$

luego

$$C(\widehat{F}_t) - C(F^*) < -\epsilon/2 < 0 \Rightarrow C(\widehat{F}_t) < C(F^*)$$

lo cual es absurdo ya que $C(F^*)$ era por hipótesis una cota inferior de $\{C(G) : G \in \text{lin}(\mathcal{H})\}$.

Si $(F_t)_t$ tiene un punto de acumulación, entonces (3.11) se obtiene inmediatamente a partir de (3.10) y del hecho de que $(C(F_t))_t$ es una sucesión monótona decreciente. En otro caso, por el Teorema 3.1,

$$\lim_{t\to\infty} \sup_{h\in\mathcal{H}} -\langle \nabla C(F_t), h\rangle = 0,$$

lo que por convexidad de C implica (3.11). □

Estos resultados sugieren la forma que deben tener las funciones de coste para que el algoritmo AnyBoost funcione de forma satisfactoria, deben ser funciones acotadas inferiormente, Lipschitz diferenciable, y convexas.

Capítulo 4

Boosting Trees

El tema de este capítulo es la aplicación del boosting a los árboles de decisión. Los árboles de decisión son un buen ejemplo de modelos débiles, que por su sencillez los hacen muy atractivos. Uno de los algoritmos más populares diseñados para entrenar árboles de decisión se conoce como CART (Breiman *et al.*, 1984).

Con el objetivo de no alargar el trabajo en exceso, no se entrará al detalle con los árboles de clasificación, sino que se proporcionarán las bases necesarias para desarrollar la parte práctica en el capítulo 5.

Este capítulo sigue la teoría desarrollada en los capítulos 9 y 10 de Hastie *et al.* (2009), y el capítulo 18 de Shalev-Shwartz y Ben-David (2014).

4.1. Introducción a los árboles de decisión

Los algoritmos que entrenan árboles de decisión toman como parámetro de entrada un conjunto de entrenamiento con muestras etiquetadas. Cada una de estas muestras consiste en un par (x, y) donde x es un individuo con ciertos atributos $x = (x_1, ..., x_p)$, e y es una variable que puede ser tanto numérica como categórica. En el problema de clasificación binaria $y \in \{-1, +1\}$ es la etiqueta asociada al individuo x. El algoritmo escoge el atributo que mejor divide el conjunto de entrenamiento en dos subconjuntos, la elección del atributo depende de cierto criterio que se impone en el diseño del algoritmo. Normalmente cada uno de estos dos subconjuntos todavía contiene individuos de ambas clases, de modo que se escoge otro atributo que divide cada uno de estos subconjuntos en dos subconjuntos. Este proceso se repite hasta que cada subconjunto contiene elementos de una única clase. De

este modo, cada nodo interno tiene un atributo y un valor para ese atributo que determina si un individuo (cuya etiqueta se desea conocer) debe tomar la rama derecha o la izquierda que parte de ese nodo. En los nodos terminales se realiza la decisión final, dándole a ese individuo la etiqueta +1 ó −1. Así, en los árboles de decisión se parte de un nodo raíz y de forma progresiva se atraviesa el árbol a través de sus ramas hasta que se alcanza un nodo terminal donde se realiza la predicción. En la Figura 4.1 se ha realizado un esquema de un árbol de decisión.

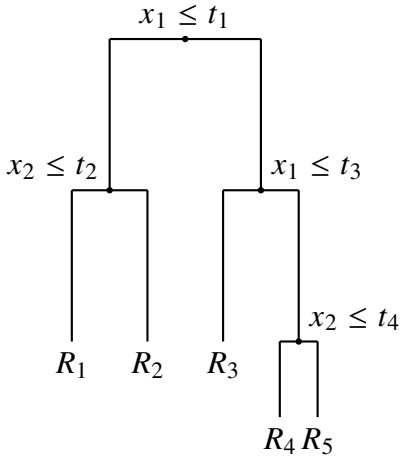


Figura 4.1: Esquema de un árbol de decisión

Sea $S = \{(x_1, y_1), ..., (x_m, y_m)\}$ un conjunto de entrenamiento, y sean R_j, $j = 1, ..., J$ los subconjuntos, o regiones, asociados a los nodos terminales de un árbol de decisión. Éstos constituyen una partición del espacio de individuos X, cada uno de ellos tiene asignada una constante γ_j, y la regla predictiva consiste en:

$$x \in R_j \Rightarrow f(x) = \gamma_j.$$

Formalmente, un árbol de parámetros $\Theta = \{R_j, \gamma_j\}_1^J$ puede expresarse como

$$T(x, \Theta) = \sum_{j=1}^{J} \gamma_j \mathbf{I}(x \in R_j), \tag{4.1}$$

donde generalmente J es un meta-parámetro, y los parámetros Θ suelen hallarse minimizando el riesgo empírico

$$\hat{\Theta} = \arg\min_{\theta} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j), \tag{4.2}$$

donde L es una función de coste análoga a la función de $C\colon \mathbb{R} \to \mathbb{R}$ que se utilizaba en el capítulo 3. Resolver (4.2) es un formidable problema de optimización combinatoria y normalmente se suele buscar soluciones subóptimas por medio de aproximaciones. Es útil dividir el problema de optimización en dos partes:

- Hallar las constantes γ_j una vez se tiene R_j. Esto suele ser trivial, generalmente en los problemas de clasificación se toma $\hat{\gamma}_j$ como la moda de la clase de los individuos que pertenecen a la región R_j.
- Hallar las regiones R_j. Esta es la parte complicada del problema, para la cual se suelen realizar aproximaciones. Nótese que determinar las regiones R_j conlleva estimar las constantes γ_j también. Una estrategia habitual es utilizar un algoritmo que de forma recursiva divida subconjuntos en otros más pequeños hasta encontrar las regiones R_j. Asimismo, a veces es necesario aproximar (4.2) por una condición más conveniente como

$$\tilde{\Theta} = \arg\min_{\theta} \sum_{i=1}^{m} \tilde{L}(y_i, T(x_i, \theta)). \tag{4.3}$$

  Una vez se han determinado los conjuntos R_j, las constantes γ_j pueden hallarse de forma más precisa utilizando el criterio original (4.2).

En la sección 9.2 del libro Hastie *et al.* (2009) se describe en detalle una estrategia para obtener árboles de clasificación siguiendo estas líneas.

4.2. Potenciación de los árboles de clasificación

Un modelo de árbol potenciado es el modelo resultante de aplicar el algoritmo de boosting a un algoritmo diseñado para entrenar árboles de decisión. Un árbol potenciado tiene la siguiente forma:

$$F_N(x) = \sum_{n=1}^{N} T(x; \Theta_n). \tag{4.4}$$

Nótese el cambio de notación que se ha llevado a cabo respecto a la notación empleada en el capítulo 3. En ese capítulo T determinaba las rondas de boosting (y por ende el número de clasificadores débiles) y h denotaba a los clasificadores débiles. En este caso, los clasificadores débiles son los árboles y se representan mediante T, y el número de éstos es N.

En cada paso de la construcción del árbol potenciado (4.4) uno debe resolver

$$\hat{\Theta}_n = \arg\min_{\Theta_n} \sum_{i=1}^{m} L(y_i, F_{n-1}(x_i) + T(x_i; \Theta_n)) \tag{4.5}$$

para obtener el conjunto de regiones y constantes $\Theta_n = \{R_{jn}, \gamma_{jn}\}_1^{J_n}$ del siguiente árbol a partir del modelo actual F_{n-1}.

A partir de las regiones R_{jn}, encontrar las constantes γ_{jn} óptimas para cada región suele ser inmediato:

$$\hat{\gamma}_{jn} = \arg\min_{\gamma_{jn}} \sum_{i=1}^{m} L(y_i, F_{n-1}(x_i) + \gamma_{jn}\mathbf{I}(x_i \in R_{jn})). \tag{4.6}$$

Hallar las regiones R_{jn} es un problema complejo, aunque este problema se simplifica para algunas funciones de pérdida como la exponencial $L(f) = \exp(-yf(x))$, o la función de pérdida cuadrática $L(f) = |y - f(x)|^2$.

Una estrategia habitual para construir los árboles consiste en recurrir a los algoritmos desarrollados en el capítulo 3: AnyBoost y MarginBoost, que trataban de potenciar el gradiente de una función de coste. En el caso de los árboles de decisión esto consiste en buscar, en cada iteración, el árbol que apunta en la dirección descendente del gradiente de una función de coste $L(F)$.

4.3. Tamaño de los árboles potenciados

El tamaño de los árboles suele determinarse del siguiente modo: en primer lugar se construye un árbol muy grande (sobredimensionado), y luego éste se va podando (se cortan ramas de decisión) de abajo hacia arriba hasta llegar al número óptimo de nodos terminales. Sin embargo este procedimiento no es recomendable en el caso de los árboles potenciados, donde su aplicación resulta en que los árboles tienden a ser demasiado largos (especialmente en las iteraciones iniciales) según puede leerse en Hastie *et al.* (2009). Esto degrada de forma sustancial el desempeño del árbol potenciado e incrementa los tiempos computacionales.

La estrategia más simple para evitar este problema en la potenciación de árboles es forzar que todos los árboles tengan el mismo tamaño, imponiendo $J_n = J \; \forall n$. De este modo en cada iteración del algoritmo de boosting se construye un árbol con J nodos terminales. Por ello J se convierte en un meta-parámetro del proceso de boosting que debe ser ajustado para aprovechar al máximo los datos disponibles.

Aunque en la mayoría de los casos $J = 2$ será insuficiente, es muy improbable que se requiera un valor superior a $J = 10$. En Hastie *et al.* (2009) se indica que por experiencia $4 \leq J \leq 8$ suele funcionar bien en el contexto del boosting. No obstante, uno puede afinar el valor de J probando diferentes valores y comprobando cual da menor error en el conjunto de validación.

Capítulo 5

Estudio de caso

Este capítulo está dedicado a resolver un problema de aprendizaje estadístico mediante la técnica de boosting aplicada a árboles de decisión. En primer lugar se introducirá brevemente el problema planteado, y posteriormente se describirán los pasos llevados a cabo para resolver el problema.

5.1. Descripción del problema

El aeropuerto Madrid-Barajas cuenta con cuatro pistas paralelas dos a dos, las 18L/36R - 18R/36L y las 14L/32R - 14R/32L. Estas pistas se utilizan tanto para aterrizar como para despegar, ahora bien su uso no es arbitrario. Si una pareja de pistas paralelas se utiliza para despegar, entonces la otra pareja se emplea para el aterrizaje. De este modo se definen dos posibles configuraciones:

- Norte: las pistas 18L/36R - 18R/36L se utilizan para despegar, y las 14L/32R - 14R/32L se utilizan para aterrizar.
- Sur: las pistas 14L/32R - 14R/32L se utilizan para despegar, y las 18L/36R - 18R/36L se utilizan para aterrizar.

La Figura[1] 5.1 permite ilustrar la configuración de pistas. A la izquierda se halla la configuración norte, y a la derecha se encuentra la configuración sur. Los colores verde y rojo denotan las aeronaves que despegan y aterrizan, respectivamente. La configuración preferente en el aeropuerto Madrid-Barajas es la norte. No obstante la configuración del

[1]Figura tomada de https://www.copac.es/wp-content/uploads/2017/08/guia-aemet-LEMD.pdf a fecha de 09/06/2021

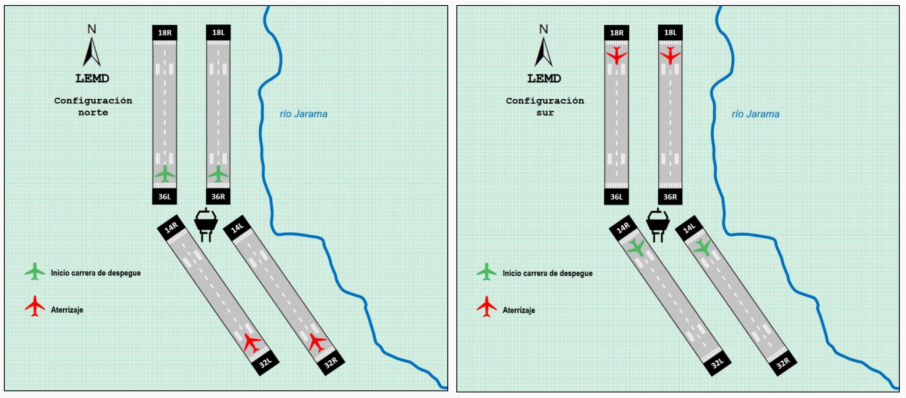


Figura 5.1: Configuración de pistas aeropuerto Madrid-Barajas

aeropuerto puede cambiar por diversos factores, algunos de ellos son meteorológicos, y otros están relacionados con la densidad de vuelos.

El problema que se plantea consiste en predecir la configuración de pistas a corto plazo, para resolverlo se cuenta con datos tomados durante dos años, en intervalos de treinta minutos, que contienen información meteorológica y sobre la densidad de vuelos.

El problema que se plantea es interesante ya que permitiría a las aerolíneas optimizar recursos en la planificación del vuelo. La trayectoria de las aeronaves se ve afectada por la configuración de pistas del aeropuerto de origen y destino, conocer esta configuración permitiría diseñar la ruta óptima ahorrando tiempo y combustible.

5.2. Resolución del problema

El caso planteado es un problema de aprendizaje estadístico de clasificación binaria, como los que se han estudiado de forma teórica en los capítulos anteriores. Se desea construir un modelo que permita predecir la configuración de las pistas del aeropuerto Madrid-Barajas.

Para estudiar este problema de clasificación binaria se trabajará con RStudio y se empleará el paquete `xgboost` de R. XGBoost es un algoritmo de boosting aplicado a árboles de decisión, puede encontrarse más información en Chen y Guestrin (2016) donde se presenta un estudio completo del mismo.

5.2.1. Exploración del conjunto de datos

Se agradece a Boeing Research and Technologies Europe (BRTE) y al proyecto- contrato "Desarrollo de un algoritmo para la predicción de la configuración de los aeropuertos y soporte en data management en la plataforma ADAPT"(03/07/2019 – 30/01/2020), entre BRTE y la UVa, el acceso a los datos que constituyen este caso de estudio.

El conjunto de datos consiste en un total de 31946 muestras etiquetadas, y cada muestra tiene 18 variables predictoras, todas ellas carácter numérico, cuyos valores se han medido en intervalos de media hora. Estas variables se agrupan en:

- Variables *meteorológicas*: temperatura (tmp), punto de rocío (dew_pt), humedad relativa (rel_hum), dirección del viento (wind_dir), velocidad del viento (wind_sp), velocidad de ráfagas de aire (wind_gust), visibilidad (visib), presión (press), nivel del manto de nubes (sky_lvl), lluvia (rain), nieve (snow), llovizna (drizzle) y niebla (fog).
- Variables de tipo *fecha*: hora del día (time_class), día de la semana (day_week) y mes del año (month).
- Variables de tipo *densidad de vuelo*: número de vuelos que aterrizan (num_land) y número de vuelos que despegan (num_tkoff).

A continuación se muestra una tabla con 11 de las 18 variables predictoras de 10 individuos aleatorios:

```
# A tibble: 10 x 18
     tmp dew_pt rel_hum wind_dir wind_sp wind_gust visib press sky_lvl  rain  snow
   <dbl>  <dbl>   <dbl>    <dbl>   <dbl>     <dbl> <dbl> <dbl>   <dbl> <int> <int>
 1     3      3   100        340   13.0          0     2   998  0.0610     0     0
 2    11      2    53.8      280   24.1          0    10  1011  1.83       0     0
 3    12      9    81.9      200   18.5          0    10  1007  0.457      0     0
 4     9     -9    27.0      200    7.41         0    10  1006 10          0     0
 5    13      9    76.7      210    3.70         0    10  1025 10          0     0
 6    15     10    72.0       NA    3.70         0    10  1019 10          0     0
 7     9      9   100        140   11.1          0     5  1006  0.0914     0     0
 8    21      9    46.2      340    7.41         0    10  1020 10          0     0
 9    20     17    82.9       NA    3.70         0    10  1020  0.549      0     0
10     9      8    93.5       NA    7.41         0    10  1009  1.52      -1     0
# ... with 7 more variables: drizzle <int>, fog <int>, time_class <int>,
#   day_week <int>, month <dbl>, num_land <int>, num_tkoff <int>
```

Este conjunto de datos se ha dividido en dos partes, el 70 % de las muestras constituyen el conjunto de entrenamiento, y el 30 % restante forma el conjunto de prueba que se utilizará

para evaluar el modelo.

5.2.2. `xgboost`

La función `xgboost` dentro del paquete `xgboost` es la encargada de generar un clasificador combinado utilizando boosting con árboles de decisión. Los parámetros básicos de esta función son:

- `data`: matriz con el formato `DMatrix`, propio del paquete en uso. Cada fila de esta matriz es un individuo del conjunto de entrenamiento, y las columnas deben contener los valores de las variables predictoras.
- `objective`: indica qué tipo de problema de aprendizaje estadístico queremos abarcar, en este caso: `binary:logistic` (clasificación binaria).
- `nrounds`: número total de árboles a combinar (número de iteraciones).
- `max.depth`: profundidad máxima de los árboles.
- `eta`: este parámetro se conoce como *tasa de aprendizaje*, escala la contribución de cada árbol por un valor $0 <$ `eta` <1. Se emplea para prevenir el comportamiento de sobreajuste.

```
xgboost(data = ,
        objective = "binary:logistic",
        nrounds = ,
        max.depth = ,
        eta = )
```

Para los problemas de clasificación binaria `xgboost` utiliza la función de pérdida de entropía cruzada (binaria)[2], también conocida como función de pérdida logarítmica, o logística. Dada una muestra etiquetada (x, y) con $y \in \{-1, +1\}$, el clasificador devuelve un vector $(1 - p, p)$ donde p es la probabilidad de que x pertenezca a la clase $y = +1$. Para esa muestra y esa predicción, el error dado por la función de pérdida logística se calcula como:

$$L(y, p) = -\left(\frac{y+1}{2}\ln(p) - \frac{y-1}{2}\ln(1-p)\right). \tag{5.1}$$

Nótese que esta función de pérdida tiene en cuenta la probabilidad asignada por el clasificador para la etiqueta correcta. De este modo si la probabilidad de la etiqueta correcta se encuentra próxima a 1 entonces la función toma valores próximos a cero. Por contra,

[2]En inglés: *(binary) cross entropy loss*

cuanto menor sea la probabilidad asignada a la categoría correcta, mayor penalización obtendrá. En la Figura 5.2 se ha representado la función $-\ln(x)$ con $0 < x \leq 1$, donde puede advertirse la importancia que tiene no sólo clasificar correctamente un individuo, sino de hacerlo asignando a la etiqueta correcta una alta probabilidad.

Para evaluar el modelo sobre un conjunto de muestras empleando esta función de pérdida, basta con evaluar esta función de forma individual sobre cada muestra y luego tomar la media aritmética.

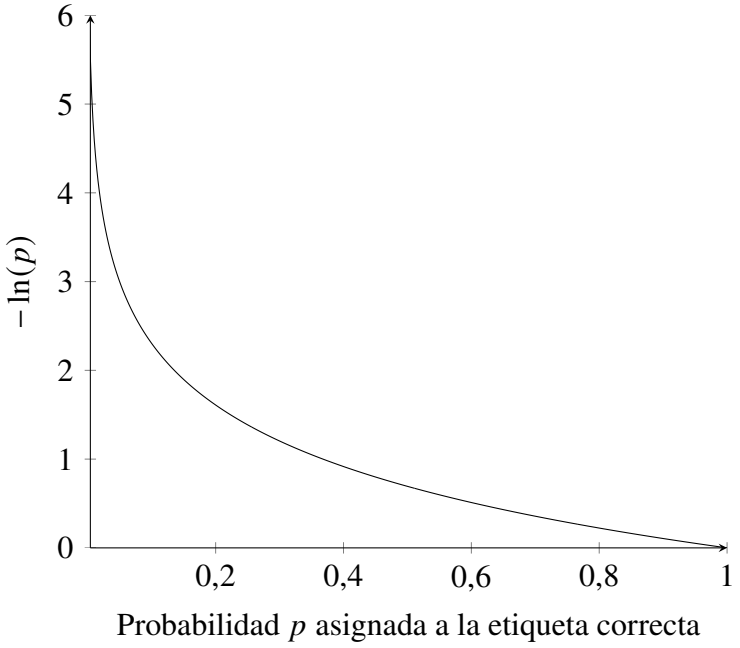


Figura 5.2: Función de pérdida de entropía cruzada

Dentro del paquete `xgboost` se encuentra, `xgboost.cv`. Esta función es de especial interés ya que genera un modelo de árboles potenciado como la función `xgboost`, pero utilizando validación cruzada[3] de n iteraciones (donde n es un parámetro que se especifica mediante `nfold`).

La validación cruzada de n iteraciones es una técnica que consiste en dividir el conjunto de entrenamiento en n subconjuntos disjuntos, en cada iteración uno de los subconjuntos se utiliza como conjunto de prueba y los $(n-1)$ restantes se toman como conjunto de entrenamiento. Generalmente en cada iteración se obtiene un error de prueba evaluando el modelo obtenido con el subconjunto que se ha quedado fuera del entrenamiento. Finalmente se realiza una media aritmética de los resultados de cada iteración y se obtiene una estimación del error de generalización. Aunque el valor de n depende del tamaño del conjunto de entrenamiento, en la práctica lo más común es tomar $n = 10$.

[3]En inglés: *cross validation*.

En este caso, para evaluar los modelos en cada iteración `xgboost.cv` utiliza la función de pérdida logarítmica. Esta evaluación se denotará mediante `cv-error` o `cv-error` (logloss), donde el término "logloss" se refiere al uso de la función de pérdida de entropía cruzada.

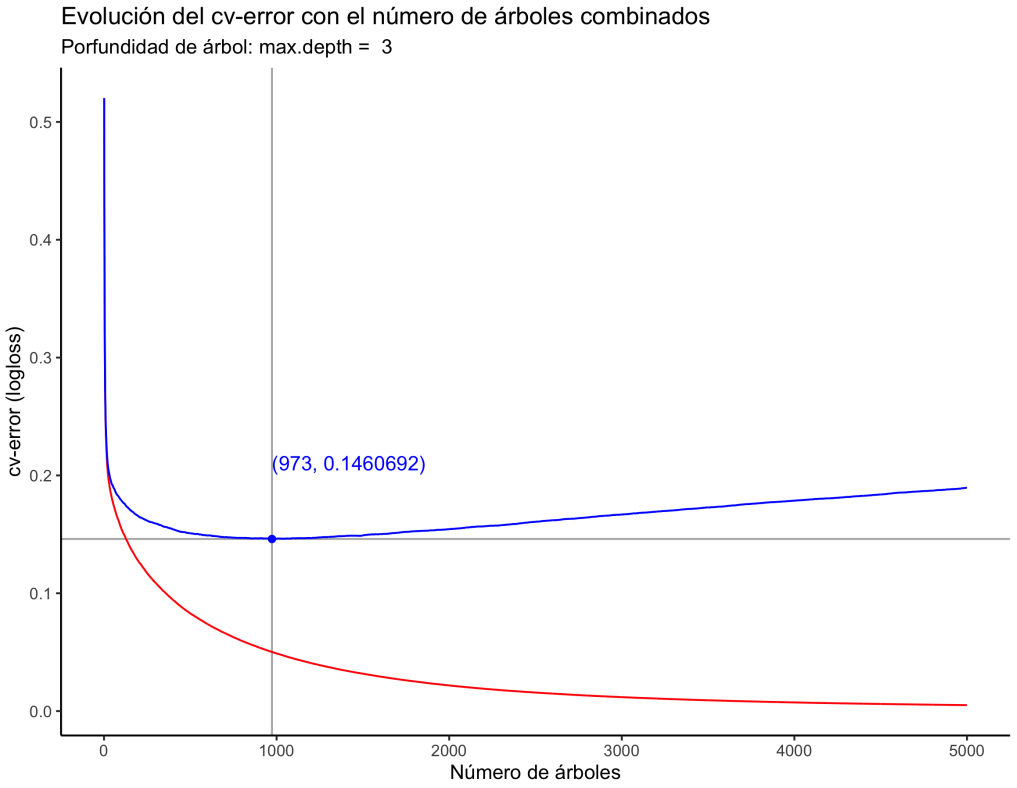
En la Figura 5.3 se ha representado de color azul el valor de la función de pérdida logarítmica sobre el conjunto que ha sido excluido del entrenamiento (`cv-error`), frente al número de árboles combinados (número de iteraciones); y en rojo aparece el valor de la función de pérdida sobre el conjunto de entrenamiento. El corte de la recta vertical gris con el eje horizontal determina el número de iteraciones donde `cv-error` alcanza su valor mínimo, y este valor viene dado por el corte de la recta horizontal gris con el eje de ordenadas.

En la Figura 5.3 aparecen dos gráficas, cada una se corresponde a dos modelos distintos entrenados mediante `xgboost.cv` durante 5000 iteraciones tomando `nfold` = 10, tasa de aprendizaje `eta` = 0,3 y profundidad de los árboles `max.depth` = 3 en el caso 5.3a, y `max.depth` = 6 en el caso 5.3b. Analizando este par de figuras se aprecia que en ambos casos aparece el efecto de sobreajuste similar al que que predecía la cota de la Ecuación (2.22) y que se representó en la Figura 2.1. Asimismo se aprecia que el valor mínimo del `cv-error` es menor en el segundo caso, y que para alcanzar este valor se necesita combinar muchos menos árboles que en el primer caso (187 árboles combinados frente a 973). La diferencia en el comportamiento del `cv-error` radica en que la familia de los árboles de profundidad `max.depth` = 6 es más compleja.

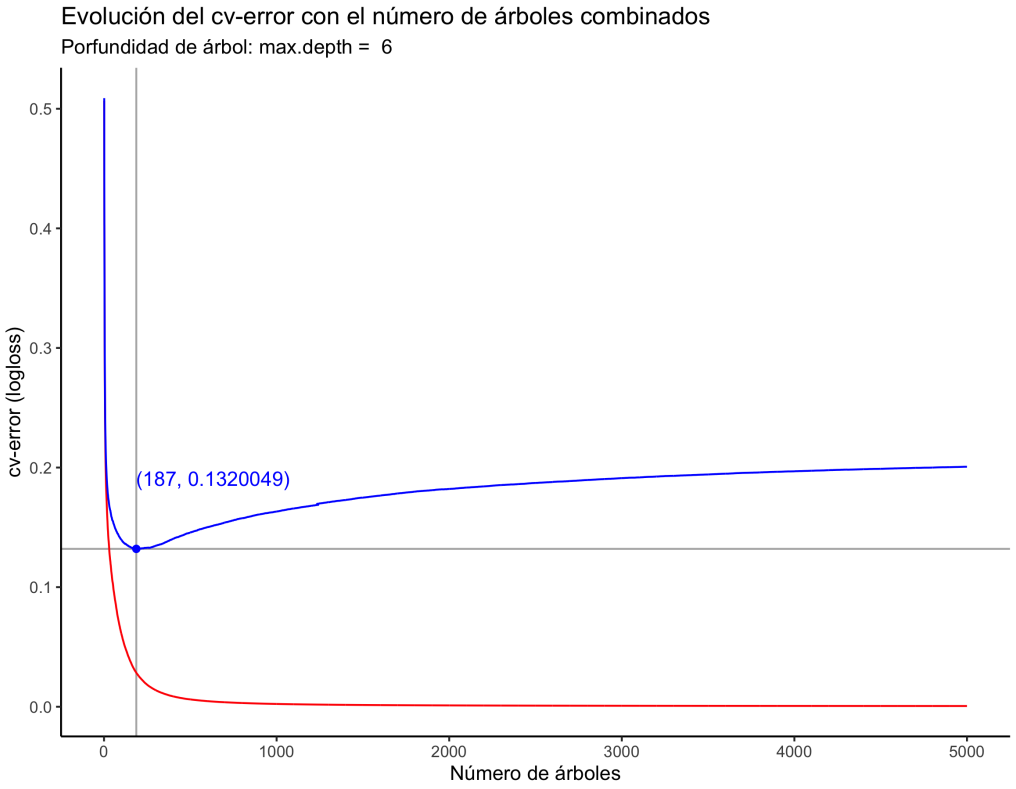
Lo expuesto en el párrafo anterior refleja claramente la necesidad de realizar una búsqueda de parámetros óptimos para construir el modelo de árboles potenciados con `xgboost`.

5.2.3. Búsqueda de parámetros óptimos

Para realizar esta búsqueda de parámetros óptimos se han entrenado 20 modelos diferentes mediante `xgboost.cv` con las posibles combinaciones de parámetros de `eta` $\in \{0{,}01, 0{,}05, 0{,}1, 0{,}3\}$ y `max.depth` $\in \{4, 5, 6, 7, 8\}$. En todos los casos el número de iteraciones es el mismo, `nrounds` = 5000. Sin embargo esta vez se ha introducido un nuevo parámetro en la función, `early_stopping_rounds` = 10, este parámetro se encarga de detener el entrenamiento si el `cv-error` no mejora durante 10 iteraciones consecutivas. Este parámetro es muy útil tanto para evitar el efecto se sobreajuste, como para ahorrar tiempo en la búsqueda.



(a) `eta` = 0,3 `max.depth` = 3



(b) `eta` = 0,3 `max.depth` = 6

Figura 5.3: Comparación del comportamiento de la función de pérdida

Para cada combinación de parámetros se ha recopilado la siguiente información:

- `min_logloss`: el mínimo valor de `cv-error` (función de pérdida logística).
- `optimal_trees`: el número de iteraciones necesarias para alcanzar el mínimo `cv-error`.
- `tiempo`: el tiempo, en segundos, requerido para completar el entrenamiento.
- La evolución del `cv-error` con el número de iteraciones.

Los resultados, ordenados de menor a mayor error, son los siguientes:

	eta	max_depth	optimal_trees	min_logloss	tiempo
1	0.05	8	631	0.1283530	194.564
2	0.05	7	808	0.1287021	208.877
3	0.05	6	1089	0.1288265	233.341
4	0.01	8	3109	0.1292332	944.251
5	0.10	8	321	0.1292809	100.859
6	0.01	7	4021	0.1297296	1029.817
7	0.10	7	439	0.1299297	114.887
8	0.10	6	575	0.1299551	124.591
9	0.01	6	5000	0.1321008	1055.545
10	0.30	6	149	0.1331814	34.195
11	0.10	5	781	0.1335203	136.793
12	0.30	8	105	0.1336123	35.311
13	0.05	5	1346	0.1351189	233.901
14	0.30	7	136	0.1357820	37.742
15	0.30	5	257	0.1376216	46.373
16	0.30	4	341	0.1404829	48.278
17	0.01	5	4999	0.1418685	865.054
18	0.10	4	814	0.1436846	112.650
19	0.05	4	1854	0.1439742	254.880
20	0.01	4	5000	0.1545247	682.728

Con estos datos se han realizado diversos análisis en los que se compara: la evolución del valor de la función de pérdida (`cv-error`), el valor del mínimo de esta función según los parámetros `eta` y `max.depth`, y el valor mínimo frente al tiempo necesario para entrenar el modelo.

La Figura 5.4 está compuesta por 5 gráficas, cada una de ella se corresponde al entrenamiento de árboles con distinta profundidad. Se encuentran ordenadas de menor a mayor profundidad, situándose en la parte superior la gráfica correspondiente a los árboles con `max.depth` = 4. Esta figura muestra de forma muy visual el diferente comportamiento que tiene el `cv-error` en cada caso. Por un lado se aprecia que, sin importar cual sea el valor de la tasa de aprendizaje `eta`, los modelos entrenados con árboles de menor

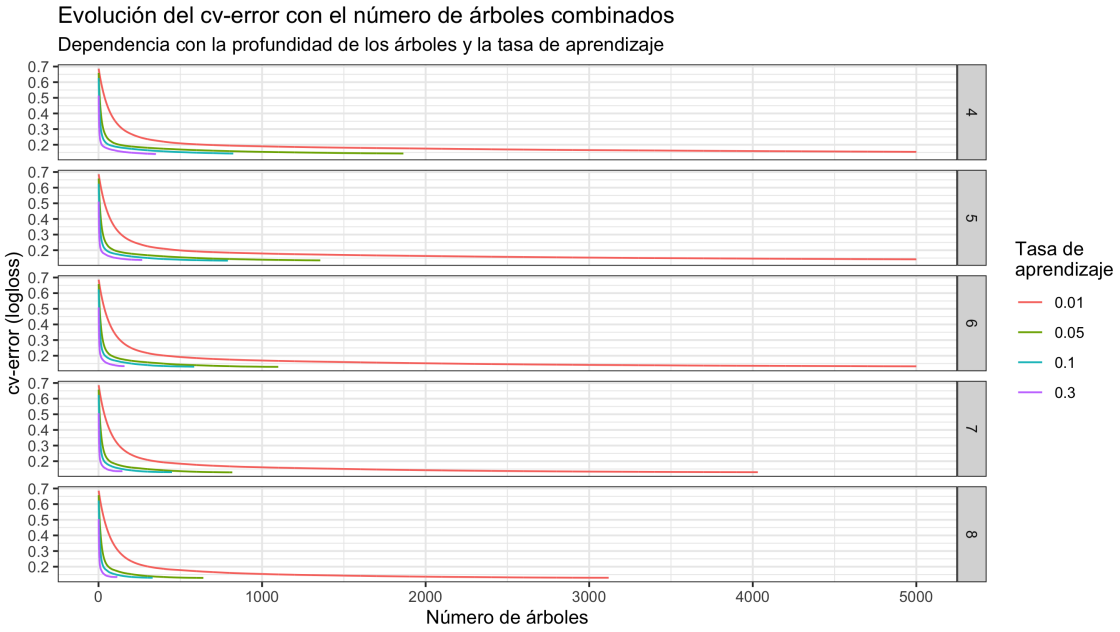


Figura 5.4: Evolución del valor de la función de pérdida

profundidad requieren un mayor número de iteraciones para alcanzar el valor mínimo de la función de pérdida. Por otro lado queda claro que, empleando árboles de la misma profundidad, tomar tasas de aprendizaje pequeñas aumenta considerablemente el número de árboles que hay que combinar para alcanzar la mínima pérdida.

De este análisis se desprende que las combinaciones con tasa de aprendizaje `eta` = 0,01 quedan totalmente descartadas puesto que requieren una cantidad de árboles muy grande para alcanzar el mínimo `cv-error`. De hecho en los tres primeros casos se observa que tras 5000 iteraciones todavía no se ha alcanzado este valor mínimo.

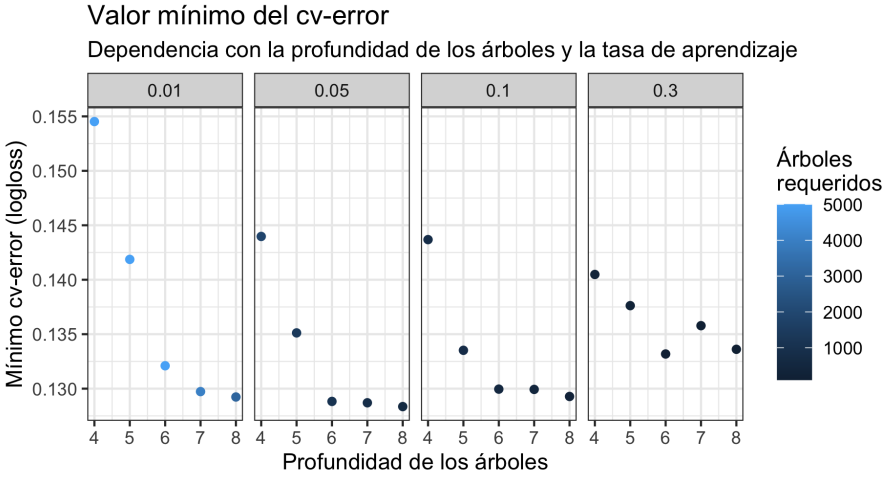


Figura 5.5: Valor mínimo de la función de pérdida

La Figura 5.5 muestra, para cada valor de `eta`, el mínimo valor de `cv-error` según la profundidad de los árboles. Rápidamente se observa que los mejores resultados se obtienen para combinaciones de `eta` $\in \{0{,}05, 0{,}1\}$ y `max.depth` $\in \{6, 7, 8\}$.

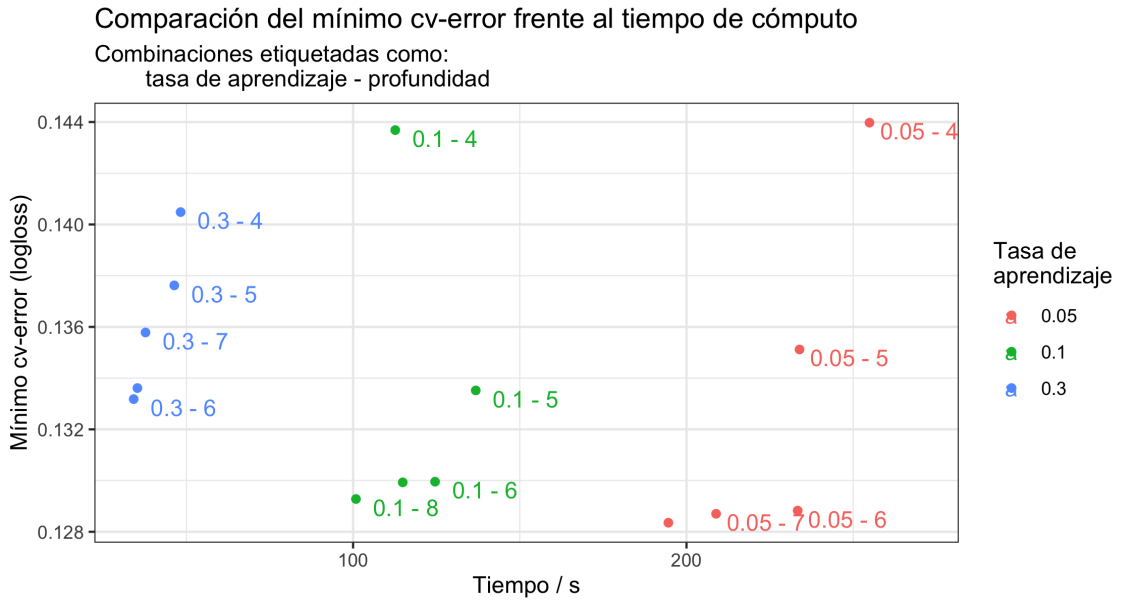


Figura 5.6: Valor mínimo de la función de pérdida frente a tiempo de entrenamiento

Por último se ha querido tener en consideración el tiempo. Según en qué situaciones puede resultar más conveniente trabajar con un modelo ligeramente menos preciso, pero que requiera un menor tiempo de entrenamiento. Por ello en la Figura 5.6 se ha querido representar el valor mínimo de `cv-error` frente al tiempo necesario para el entrenamiento[4]. Nótese que este tiempo no es el mismo que se requiere para entrenar un modelo con la función `xgboost`, pues al estar haciendo validación cruzada los tiempos que aquí aparecen son mucho mayores. No obstante, lo que es realmente interesante no es el tiempo requerido para entrenar un modelo sino las *diferencias* de tiempo utilizando las diferentes combinaciones de parámetros.

Atendiendo a la Figura 5.6 se aprecia que los entrenamientos con una tasa de aprendizaje `eta` $= 3$ se llevan a cabo en un tiempo mucho menor que el resto. Además para las combinaciones con `eta` $\in \{0{,}05, 0{,}1\}$ se observa que, excluyendo el caso de árboles con `max.depth` $= 4$, cuando la profundidad de los árboles es `max.depth` $= 8$ se obtienen modelos bastante más rápidos de entrenar.

[4]Se ha excluido del análisis los modelos con `eta` $= 0{,}01$ pues requerían un tiempo muchísimo mayor al resto.

Por lo expuesto anteriormente se concluye que las tres mejores combinaciones de parámetros son:

```
    eta max_depth optimal_trees min_logloss   tiempo
1  0.05         8           631   0.1283530  194.564
5  0.10         8           321   0.1292809  100.859
10 0.30         6           149   0.1331814   34.195
```

5.2.4. Modelo final

Una vez entrenados los tres modelos, con la función `xgboost` y los parámetros obtenidos en el apartado anterior, éstos se han evaluado con el conjunto de prueba. No obstante, antes de presentar los resultados conviene introducir los conceptos:

- Precisión: representa la probabilidad de clasificar correctamente una muestra. Se calcula como el cociente entre el número de muestras clasificadas correctamente y el número de muestras total. La precisión puede utilizarse para determinar una estimación del error de generalización mediante el cálculo: 1 − precisión.
- IC 95 %: se trata del intervalo de 95 % de confianza para la estimación de la precisión.
- Sensibilidad: representa la probabilidad de que un individuo con la etiqueta +1 se clasifique correctamente. Es el cociente entre el número de muestras clasificadas *correctamente* con +1 y el número de muestras con la etiqueta +1.
- Especificidad: representa la probabilidad de que un individuo con la etiqueta −1 se clasifique correctamente. Es el cociente entre el número de muestras clasificadas *correctamente* con −1 y el número de muestras con la etiqueta −1.
- Pérdida logística: es el resultado de evaluar la función de pérdida logística (5.1) sobre el conjunto de prueba. Es de gran utilidad para comparar modelos de clasificación.

En este caso la etiqueta +1 se corresponde con la configuración Norte de las pistas. Los resultados de los modelos finales son los siguientes:

- Modelo con `eta` = 0,05, `max.depth` = 8, `nrounds` = 631. Modelo entrenado en 21,568 segundos.

```
                Precisión : 0.9538
                   IC 95% : (0.9494, 0.9579)
             Sensibilidad : 0.9792
            Especificidad : 0.8615
      Pérdida logística : 0.1259
```

- Modelo con `eta` = 0,1, `max.depth` = 8, `nrounds` = 321. Modelo entrenado en 11,089 segundos.

```
              Precisión : 0.9525
                 IC 95% : (0.9481, 0.9567)
           Sensibilidad : 0.9792
          Especificidad : 0.8557
      Pérdida logística : 0.1262
```

- Modelo con `eta` = 0,3, `max.depth` = 6, `nrounds` = 149. Modelo entrenado en 3,596 segundos.

```
              Precisión : 0.9507
                 IC 95% : (0.9462, 0.955)
           Sensibilidad : 0.9776
          Especificidad : 0.8533
      Pérdida logística : 0.1293
```

Los tres modelos tienen una capacidad predictora similar. En estas condiciones, en la práctica es preferible escoger el modelo más sencillo, el formado por un menor número de árboles. No obstante cualquiera de estas tres combinaciones es correcta para el estudio de caso actual.

5.3. Importancia de las variables predictoras

El paquete `xgboost` cuenta con una función que determina cuanta importancia tiene cada una de las variables predictoras que se han empleado para entrenar el modelo. Esta función es `xbg.importance` y admite como parámetro un modelo entrenado con `xgboost`.

Para determinar la importancia que tiene cada variable, `xbg.importance` analiza el número de nodos en los que aparece esa variable y los beneficios, en términos de reducción del error de entrenamiento, que supone tener esa variable en los nodos.

Esta función ha sido aplicada a un modelo entrenado con los parámetros `eta` = 0,3, `max.depth` = 6, `nrounds` = 149. Estos son los resultados:

	Feature	Importance
1:	wind_dir	0.3668
2:	wind_sp	0.2334
3:	press	0.0640
4:	month	0.0587
5:	rel_hum	0.0464
6:	dew_pt	0.0434
7:	tmp	0.0434
8:	time_class	0.0335
9:	sky_lvl	0.0284
10:	day_week	0.0276
11:	num_tkoff	0.0203
12:	num_land	0.0180
13:	wind_gust	0.0123
14:	visib	0.0019
15:	rain	0.0010
16:	fog	0.0005
17:	drizzle	0.0002

Estos resultados han sido representados en la Figura 5.7.

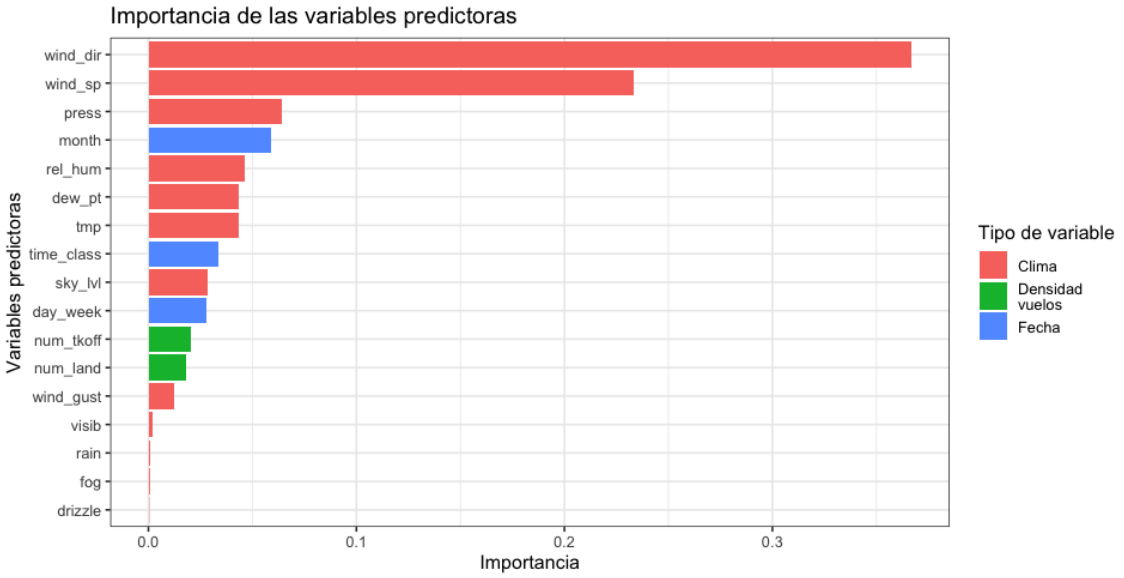


Figura 5.7: Importancia de las variables predictoras

Esta medida de la importancia de las variables arroja un poco de luz sobre los factores que determinan el cambio de configuración de las pistas en el aeropuerto Madrid-Barajas. Se aprecia que el viento es un factor clave en la configuración de las pistas, esto tiene mucho sentido si se cae en la cuenta de la dificultad que supone aterrizar una aeronave con el viento de cola.

5.4. Conclusiones

Este estudio de caso ha servido para poner en práctica los conceptos desarrollados en los capítulos previos. En particular, en la Figura 5.3 se puede apreciar tanto el sesgo del error de entrenamiento, como el efecto de sobreajuste, análogo al predicho por la cota de la Ecuación (2.22).

A modo de resumen, el análisis desarrollado nos permite concluir que en este estudio de caso entrenar árboles de profundidad `max.depth` = 6 con una tasa de aprendizaje `eta` = 0,3 permite obtener un clasificador combinado con un error de generalización estimado del 5 %.

Para terminar, cabe destacar que este estudio puede realizarse sobre otros aeropuertos. Los resultados son satisfactorios y el paquete `xgbooost` ha resultado ser una herramienta muy versátil. Queda pendiente para un futuro estudiar la configuración de pistas de un aeropuerto de mayor tamaño como el aeropuerto de Amsterdam-Schipol.

Bibliografía

Blumer, A., Ehrenfeucht, A., Haussler, D., y Warmuth, M. K. (1987). Occam's razor. *Information Processing Letters*, 24(6):377–380.

Blumer, A., Ehrenfeucht, A., Haussler, D., y Warmuth, M. K. (1989). Learnability and the vapnik-chervonenkis dimension. *Journal of the ACM (JACM)*, 36(4):929–965.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

Breiman, L. (1999). Prediction games and arcing algorithms. *Neural Computation*, 11(7):1493–1517.

Breiman, L., Friedman, J., Stone, C. J., y Olshen, R. A. (1984). *Classification and regression trees*. CRC press.

Chen, T. y Guestrin, C. (2016). Xgboost: A scalable tree boosting system. En *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794.

Freund, Y. y Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.

Friedman, J., Hastie, T., y Tibshirani, R. (1998). Additive logistic regression: A statistical view of boosting (technical report). *Stanford University Statistics Department. www. statstanford. edu/˜ tibs. Hartigan, J., & Wong, M.(1979). A k-means clustering algorithm, ALGORITHM AS*, 136:322–330.

Hastie, T., Tibshirani, R., y Friedman, J. (2009). *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

Kearns, M. J. y Vazirani, U. (2018). *An Introduction to Computational Learning Theory*.

Mason, L., Baxter, J., Bartlett, P., y Frean, M. (1999). Boosting algorithms as gradient descent in function space. Nips.

Mason, L., Baxter, J., Bartlett, P., y Frean, M. (2000). Boosting algorithms as gradient descent (advances in neural information processing systems, vol. 12).

Sauer, N. (1972). On the density of families of sets. *Journal of Combinatorial Theory, Series A*, 13(1):145–147.

Schapire, R. E. y Freund, Y. (2013). Boosting: Foundations and algorithms. *Kybernetes*.

Schapire, R. E. y Singer, Y. (1999). Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336.

Shalev-Shwartz, S. y Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge university press.

Vapnik, V. y Chervonenkis, A. (1974). Theory of pattern recognition.

Vapnik, V. y Chervonenkis, A. (2015). On the uniform convergence of relative frequencies of events to their probabilities. En *Measures of complexity*, pp. 11–30. Springer.

Apéndice A

Cógido del estudio de caso

Código empleado para desarrollar el capítulo 5.

```
# Cargamos las librerías
library(xgboost)
library(tidyverse)
library(caret)
library(tictoc)

# Cargamos los datos desde un archivo .Rds
data <- readRDS("/Users/bernardo/Desktop/TFG Mates/datos_para_Bernardo/
                data_con_rw_FR24_LEMD_2018-01-01_2019-12-31.Rds")
etiquetas <- data$global_config
atributos <- data$features_df
muestras <- left_join(etiquetas, atributos, by = c("time_ref" = "time_ref"))
rm(data, etiquetas, atributos)
muestras <- muestras %>% select(-valid_time)
# Recodificamos la configuración para dejarlo como Norte / Sur
muestras$config[str_detect(muestras$config,"18")] <- "S"
muestras$config[str_detect(muestras$config,"32")] <- "N"
# xgboost requiere matrices numéricas para funcionar
muestras$config <- muestras$config %>%
  as.factor() %>%
  as.numeric %>% {.-1} # xgboost espera que las etiquetas tengan valor en 0-1
# Separamos el conjunto de test y el de entrenamiento
set.seed(1998)
train_df <- sample_frac(muestras, size = 0.7)
test_df <- setdiff(muestras, train_df)
# Convertir a DMatrix
train_mat <- train_df %>%
  select(-config, -time_ref) %>% # Nos quedamos solo con los atributos
  as.matrix %>%
  xgb.DMatrix(data = ., label = train_df$config)
test_mat <- test_df %>%
  select(-config, -time_ref) %>%
  as.matrix %>%
  xgb.DMatrix(data = ., label = test_df$config)

#### Entrenamiento del modelo con hiperparámetros arbitrarios ####
```

```
modelo <- xgboost(data = train_mat,
                  objective = "binary:logistic",
                  nrounds = 1500,
                  max.depth = 3,
                  eta = 0.3,
                  nthread = 2)
# Evaluación del modelo
predict <- predict(modelo, test_mat)
cbind(predict > 0.5, test_df$config) %>%
  data.frame() %>%
  table() %>%
  confusionMatrix()

#### xgboost con validación cruzada CV ####
profundidad <- 3 #Probar con 3 y 6
modelo_cv <- xgb.cv(data = train_mat,
                    objective = "binary:logistic",
                    nrounds = 5000,
                    max.depth = profundidad,
                    eta = 0.3,
                    nthread = 2,
                    nfold = 10
                    )
# Hallemos el n   de iteraciones que minimice el train y test error
modelo_cv$evaluation_log %>%
  summarise(
    ntrees.train = which(train_logloss_mean == min(train_logloss_mean))[1],
    logloss.train = min(train_logloss_mean),
    ntrees.test = which(test_logloss_mean == min(test_logloss_mean))[1],
    logloss.test = min(test_logloss_mean),
  )
min_logloss.test <- min(modelo_cv$evaluation_log$test_logloss_mean)
min_tree.test <- which(modelo_cv$evaluation_log$test_logloss_mean ==
                         min_logloss.test)[1]
ggplot(modelo_cv$evaluation_log) +
  geom_vline(xintercept = min_tree.test, color = "grey70") +
  geom_hline(yintercept = min_logloss.test, color = "grey70") +
  geom_line(aes(iter, train_logloss_mean), color = "red") +
  geom_line(aes(iter, test_logloss_mean), color = "blue") +
  geom_point(
    data = slice_min(modelo_cv$evaluation_log,
                     order_by = test_logloss_mean, n = 1),
    aes(x = iter, y = min(test_logloss_mean)),
    color = "blue"
  ) +
  annotate("text", x = min_tree.test, y = min_logloss.test*1.4,
           label = paste0("(",min_tree.test,", ",min_logloss.test,")"),
           vjust = 0,
           hjust = 0,
           color = "blue") +
  labs(
    title = "Evolución del cv-error con el número de árboles combinados",
    subtitle = paste("Porfundidad de árbol: max.depth = ", profundidad),
    x     = "Número de árboles",
    y     = "cv-error (logloss)",
    color = ""
  ) +
  theme_classic() +
```

```
  theme(legend.position = "bottom")
ggsave(paste0("cvErrorDepth",profundidad,".png"),
       path = "/Users/bernardo/Desktop/TFG Mates/Figuras")
rm(min_logloss.test, min_tree.test)

#### Optimización de hiperparámetros eta y max_depth ####
hyper_grid <- expand.grid(
  eta = c(.01, .05, .1, .3),
  max_depth = c(4, 5, 6, 7, 8)
)
nrow(hyper_grid) # Número de combinaciones de hiperparámetros
tic("Complete model tuning")
tic.clearlog()
for(i in 1:nrow(hyper_grid)){
  tic(i)
  params <- list(
    eta = hyper_grid$eta[i],
    max_depth = hyper_grid$max_depth[i]
  )
  modelo_tune <- xgb.cv(data = train_mat,
                        params = params,
                        objective = "binary:logistic",
                        nrounds = 5000,
                        nthread = 2,
                        nfold = 10,
                        early_stopping_rounds = 10
  )
  toc(log = TRUE, quiet = FALSE)
  hyper_grid$optimal_trees[i] <-
    which.min(modelo_tune$evaluation_log$test_logloss_mean)
  hyper_grid$min_logloss[i] <-
    min(modelo_tune$evaluation_log$test_logloss_mean)
  hyper_grid$resultado[[i]] <-
    modelo_tune$evaluation_log
  print(paste("Ronda", i, "de", nrow(hyper_grid)))
}
log.lst <- tic.log(format = FALSE)
tic.clearlog()
toc()
# Añadimos los tiempos al hyper_grid
timings <- unlist(lapply(log.lst, function(x) x$toc - x$tic))
hyper_grid$tiempo <- timings
rm(timings, log.lst)
# Mostramos las 10 combinaciones de parámetros con menor error de test
hyper_grid %>%  select(-resultado) %>% arrange(min_logloss) %>% head(10)
# Preparamos los datos que serán representados
data2plot <- list()
for(i in 1:nrow(hyper_grid)){
  data2plot[[i]] <- hyper_grid$resultado[[i]]
  data2plot[[i]]$tiempo <- hyper_grid$tiempo[i]
  data2plot[[i]]$optimal_trees <- hyper_grid$optimal_trees[i]
  data2plot[[i]]$min_logloss <- hyper_grid$min_logloss[i]
  data2plot[[i]]$eta <- hyper_grid$eta[i]
  data2plot[[i]]$max_depth <- hyper_grid$max_depth[i]
}
data2plot <- bind_rows(data2plot)
data2plot$eta <- data2plot$eta %>% as.factor()
data2plot$max_depth <- data2plot$max_depth %>% as.factor()
```

```
# Representamos la evolución del test error frente a numero iter para cada eta
data2plot %>%
  ggplot() +
  geom_line(aes(x = iter, y = test_logloss_mean, color = eta)) +
  facet_grid(rows = vars(max_depth)) +
  labs( title = "Evolución del cv-error con el número de árboles combinados",
        subtitle = "Dependencia con la profundidad de los árboles
        y la tasa de aprendizaje",
        x = "Número de árboles",
        y = "cv-error (logloss)",
        color = "Tasa de\naprendizaje") +
  theme(legend.position = "bottom") +
  theme_bw()
ggsave("evError.png", path = "/Users/bernardo/Desktop/TFG Mates/Figuras",
       width = 18.18, height = 10.23, scale = 0.5)
# Representar min test error para cada max_depth.
ggplot(hyper_grid) +
  geom_point(aes(max_depth, min_logloss, color = optimal_trees)) +
  facet_grid( ~ eta) +
  labs( title = "Valor mínimo del cv-error",
        subtitle = "Dependencia con la profundidad de los árboles
        y la tasa de aprendizaje",
        x = "Profundidad de los árboles",
        y = "Mínimo cv-error (logloss)",
        color = "Árboles\nrequeridos") +
  theme_bw()
ggsave("minError.png", path = "/Users/bernardo/Desktop/TFG Mates/Figuras",
       width = 12.42, height = 6.698, scale = 0.5)
# Comparar error test frente a tiempo de ejecución
data2plot$label <- paste(data2plot$eta,"-",data2plot$max_depth)
data2plot %>%
  select(tiempo, min_logloss, label, eta) %>%
  filter(tiempo < 300) %>% #Filtramos las observaciones con tiempo desorbitado
  distinct() %>%
  ggplot(aes(x = tiempo, y = min_logloss, label = label, color = eta)) +
  geom_point() +
  geom_text(check_overlap = TRUE, vjust = 1, nudge_x = 15) +
  labs( title = "Comparación del mínimo cv-error frente al tiempo de cómputo",
        subtitle = "Combinaciones etiquetadas como:
        tasa de aprendizaje - profundidad",
        x = "Tiempo",
        y = "Mínimo cv-error (logloss)",
        color = "Tasa de\naprendizaje") +
  theme_bw()
ggsave("errorVStiempo.png", path = "/Users/bernardo/Desktop/TFG Mates/Figuras",
       width = 15.02, height = 8.094, scale = 0.5)

# CONCLUSIÓN: mejores combinaciones son 0.3-6-149, 0.1-8-321, 0.05-8-631.

#### Generamos un modelo con los hiperparámetros óptimos. ####
tic("Entrenamiento modelo final")
modelo_final <- xgboost(data = train_mat,
                        objective = "binary:logistic",
                        nrounds = 149,
                        max.depth = 6,
                        eta = 0.3,
                        nthread = 2,
)
```

```
toc()
# Evaluación del modelo
predict <- predict(modelo_final, test_mat)
cbind(predict > 0.5, test_df$config) %>%
  data.frame() %>%
  table() %>%
  confusionMatrix()
# Importancia de los predictores/atributos
importance_matrix <- xgb.importance(model = modelo_final)
xgb.plot.importance(importance_matrix)
```