Universidad de Valladolid
Facultad de Ciencias

TRABAJO FIN DE GRADO

Grado en Matemáticas

SOBREAJUSTE Y CONTROL DEL ERROR DE GENERALIZACIÓN EN REDES NEURONALES

Autor: Alejandra Pisano Castro

Tutor/es: Eustasio del Barrio Tellado

Índice general

Introducción

Actualmente vivimos en un mundo en el que la cantidad de información y datos generados crecen a diario de manera exponencial. En el pasado, la gestión y el tratamiento de toda esta información se llevaba a cabo únicamente por los humanos. Sin embargo, hoy por hoy el reto de sacar provecho de grandes bases de datos requiere de la ayuda de la tecnología. Por tanto, se hace necesario utilizar las máquinas como herramientas para la interpretación y análisis de la información a gran escala. El aprendizaje estadístico o automático (también se suele emplear el término *machine learning*) se basa precisamente en esta idea.

Llamamos *aprendizaje estadístico* al conjunto de técnicas computacionales cuyo objetivo es conseguir que las máquinas "aprendan". Aprender en este contexto quiere decir identificar patrones observando un conjunto de datos con el fin de predecir con precisión comportamientos futuros. El propósito central es el de obtener un método capaz de asignar a cada individuo su correcta etiqueta o categoría. Esta rama de la estadística fue iniciada a mediados del siglo pasado y su rápida evolución fue motivada tanto por el vertiginoso desarrollo de los ordenadores (ordenadores cada vez más y más potentes hicieron que las tareas de tratamiento y almacenamiento de datos fueran menos costosas) como por la búsqueda de inteligencia artificial. De hecho, a día de hoy el aprendizaje automático es considerado una rama de la inteligencia artificial. Además, en muchas ocasiones el campo de actuación del aprendizaje automático se solapa con el de la inferencia estadística, ya que las dos disciplinas se basan en el análisis de datos.

Existe una división evidente en dos grandes tipos de aprendizaje: supervisado y no supervisado. El *aprendizaje supervisado* tiene como objetivo deducir una regla general de asignación a partir de la observación de un conjunto de datos previos ya clasificados. Ahora bien, la mayoría de las veces, las observaciones no vienen con etiquetas predefinidas, así que vamos a querer construir modelos que puedan clasificar correctamente estos datos, encontrando por sí mismos puntos en común, características, agrupaciones... De esta tarea se encarga el *aprendizaje no supervisado* cuyo propósito es el de construir un diseño que se ajuste a un conjunto de datos sin que estos estén previamente clasificados y dicho modelo establezca la regla para futuras etiquetas. En esta memoria nos centraremos en el aprendizaje automático supervisado.

El marco teórico fue iniciado en los años 70 por Vapnik con la motivación de respon-

der a la siguiente cuestión: ¿Qué significa que una máquina aprenda?. Con este motivo, Vapnik presenta un paradigma de aprendizaje (paradigma de aprendizaje probablemente aproximadamente correcto (PAC)) totalmente nuevo estrechamente relacionado con la estadística matemática. En el segundo capítulo, presentaremos el concepto de aprendizaje PAC y demostraremos un resultado fundamental de esta teoría: la caracterización de las clases de reglas que son PAC en términos de la dimensión Vapnik-Chervonenkis. Todas estas nociones serán detalladas en la memoria.

En el tercer capítulo, nos centraremos en una de las técnicas más innovadoras empleadas en aprendizaje estadístico: *las redes neuronales artificiales.* Desde hace algunos años se han extendido con fuerza a los más diversos ámbitos: robótica, procesado del lenguaje, reconocimiento de imágenes... Aunque la base teórica de cómo trabajan está todavía en proceso de construcción, se están convirtiendo en un campo cada vez más y más estudiado. Abordaremos en esta memoria resultados que proporcionan garantías probabilísticas para su correcto funcionamiento.

El diseño más básico de red neuronal fue el modelo de perceptrón simple que surgió en la década de los 50. Sin embargo, esta técnica inicial presentaba diversas limitaciones. Por este motivo, a lo largo de los 60 se decide "ampliar" el modelo de perceptrón y nacen los perceptrones multicapas, más flexibles y útiles. Durante los años los posteriores, las redes neuronales se dejan de lado y caen en desuso y, finalmente en los años 80, tras dos décadas de olvido, surge de nuevo el interés por el estudio de esta técnica de aprendizaje debido a la aparición del algoritmo de retropropagación o backpropagation. Dicho algoritmo supuso la posibilidad de llevar a cabo el proceso de aprendizaje de una red de manera eficaz. Más adelante, explicaremos detalladamente en qué consiste el procedimiento de retropropagación y presentaremos algunos de los más habituales optimizadores empleados en el entrenamiento de redes neuronales.

En el último capítulo, plantearemos un modelo de red neuronal en R y se hará una discusión sobre el error cometido y la precisión del diseño planteado. Gracias a este experimento quedará de manifiesto uno de los grandes inconvenientes que poseen las redes neuronales: el problema del sobreajuste.

Capítulo 1

Conceptos previos.

Comenzaremos con un ejemplo sencillo de un problema que puede ser resuelto utilizando machine learning. Imagínese que deseamos clasificar una gran cantidad de textos en diferentes tipos: científicos, literarios, políticos, artículos de opinión... En primer lugar, tenemos acceso a una muestra finita de textos ya etiquetados y deseamos predecir de manera precisa la clase a la que pertenecerán nuevos documentos. El aprendizaje estadístico tendrá como fin diseñar una regla eficiente y rigurosa capaz de etiquetar correctamente nuevos textos basándose en la experiencia adquirida observando la muestra de documentos ya clasificado. Intuitivamente, cuanto más grande sea el conjunto inicial, la capacidad de generalización de la regla será mejor. Por tanto, como veremos más adelante, el tamaño de la muestra está íntimamente relacionado con la capacidad de aprendizaje.

El ejemplo anterior es solo una prueba de lo útil que puede llegar a ser el aprendizaje estadístico. Por esta razón, supone la base de herramientas que usamos a diario como el reconocimiento de voz de asistentes inteligentes como Siri o Cortana, el filtro de spam capaz de eliminar de nuestras bandejas de entrada correos no deseados, los sistemas de recomendación de contenido utilizados por Netflix, Youtube, Spotify...

El proceso de aprendizaje de cualquier técnica de machine learning es el siguiente. Inicialmente, disponemos de un conjunto de datos ya etiquetados que dividiremos en dos: *conjunto de entrenamiento* y *conjunto de validación*. Con el primero conseguimos que la máquina "aprenda" y se ajuste, de tal manera que ante una nueva situación es capaz de actuar de manera correcta. Posteriormente, se emplea el *conjunto de validación* para medir el error cometido por el diseño generado anteriormente y poder así comprobar si el modelo que hemos construido a partir de los datos de entrenamiento "funciona". Es habitual que al establecer el modelo este se amolde al conjunto de entrenamiento y ante nuevos datos su comportamiento sea pésimo. Este fenómeno se llama *sobreajuste* y conseguir que esto no suceda es uno de los principales retos del aprendizaje estadístico.

1.1. Marco del aprendizaje estadístico.

Introduciremos los conceptos y notación necesarios para abordar el problema desde un punto de vista matemático.

En el problema de aprendizaje supervisado admitimos que se observan atributos y etiquetas. Se considera $\mathcal{X}$ el conjunto de atributos que deseamos clasificar. Nos referiremos a él como *espacio de atributos*. Denotaremos por $\mathcal{Y}$ al conjunto de posibles etiquetas o grupos que se pueden asignar a cada elemento de $\mathcal{X}$. Es habitual referirse a él como *espacio de etiquetas*. El par atributo-etiqueta (X_i, Y_i) con $X_i \in \mathcal{X} = \mathbb{R}^p$ e $Y_i \in \mathcal{Y}$ es un ejemplo de un atributo al que se le ha asignado la etiqueta Y_i . El conjunto $S = ((X_1, Y_1), \ldots, (X_n, Y_n))$, al que llamaremos *conjunto de entrenamiento*, estará formado por realizaciones de vectores aleatorios independientes idénticamente distribuidos. En esencia, S contendrá una cantidad finita de atributos ya clasificados. Cuando $\mathcal{Y} = \{0, 1\}$ o equivalentemente $\{-1, 1\}$, hablaremos de problemas de *clasificación binaria*, si $\mathcal{Y} = \{1, \ldots, K\}$ los denominaremos problemas de *clasificación multiclase*. Por último, si $\mathcal{Y} = \mathbb{R}$, hablaríamos de un problema de *regresión*.

Suponemos que tanto los vectores aleatorios $(X_1, Y_1), \ldots, (X_n, Y_n)$ de la muestra de entrenamiento como el vector aleatorio (X, Y) siguen una misma distribución que denotaremos por $\mathcal{D}$. Cabe destacar que la distribución $\mathcal{D}$ es desconocida y no se realiza ninguna suposición previa sobre ella, únicamente se posee una muestra reducida, S, del comportamiento de $\mathcal{D}$.

El propósito del aprendizaje, como ya se ha comentado previamente, es obtener una buena *regla de predicción*, es decir, una función $h : \mathcal{X} \to \mathcal{Y}$ que asigne de la mejor manera posible a cada atributo de $\mathcal{X}$ su correspondiente etiqueta del conjunto $\mathcal{Y}$. Es habitual referirnos a tales reglas de predicción simplemente como *reglas*, *hipótesis* o *predictores*. El conjunto de todas las posibles hipótesis se llama *clase de hipótesis o de reglas* y lo denotaremos por $\mathcal{H}$. Cada elemento de $\mathcal{H}$ será, por tanto, una función de $\mathcal{X}$ en $\mathcal{Y}$. Además, $\mathcal{H}$ es una clase de funciones medibles (con respecto a las σ-álgebras de Borel sobre $\mathcal{X}$ y sobre $\mathcal{Y}$).

En este trabajo nos vamos a preocupar fundamentalmente de los aspectos estadísticos relacionados con distintos métodos de aprendizaje. Por lo tanto, se prestará menos atención a los aspectos computacionales y por esta razón, se usará el término *algoritmo de aprendizaje* para referirse al método que seguimos para producir una regla a partir del conjunto de entrenamiento. Si denotamos por A al algoritmo de aprendizaje, en esencia, A es una aplicación que asigna a cada conjunto de entrenamiento S una función $A(S) \in \mathcal{H}$.

Para medir el error cometido por la supuesta "buena" regla proporcionada por A, empleamos una *función de pérdida* que denotaremos por $\ell(h, (x, y))$. Esta función determina la precisión con la que la función $h \in \mathcal{H}$ asigna a $x \in \mathcal{X}$ su verdadera clase $y \in \mathcal{Y}$. La función de pérdida toma valores en $\mathbb{R}$, aunque es habitual que únicamente tome valores

positivos. Obviamente, si la predicción se desvía demasiado de la etiqueta real, la función de pérdida tomará un valor más grande.

Para problemas de clasificación binaria, la función de pérdida más utilizada es la función de pérdida 0-1 (ℓ_{0-1}) definida como

$$\ell_{0-1}(h,(x,y)) = \mathbb{I}_{[h(x)\neq y]} = \begin{cases} 0 & \text{si } h(x) = y \\ 1 & \text{si } h(x) \neq y \end{cases}$$

Otra función comúnmente usada en problemas de regresión es la *función de pérdida cuadrática*, $\ell_2(h,(x,y)) = (h(x)-y)^2$.

Para medir de manera global el error cometido por una regla, se define el *riesgo* o *error de generalización* asociado a la regla $h \in \mathcal{H}$ con respecto a $\mathcal{D}$ de la siguiente manera:

$$L_{\mathcal{D}}(h) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[\ell\,(h,(x,y))].$$

En el caso concreto de la función de pérdida cuadrática (ver apéndice A), la expresión para el riesgo es

$$\begin{aligned} L_{\mathcal{D}}(h) &= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[h(X)-Y]^2 = \\ &= \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[Y - \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[Y|X]\right]^2 + \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[Y|X] - h(X)\right]^2. \end{aligned} \tag{1.1}$$

De esta descomposición se deduce que existe una regla, que llamaremos *regla de Bayes* y denotaremos por $h_B(x) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[Y|X=x]$, tal que minimiza $L_{\mathcal{D}}(h)$. Sin embargo, dado que la distribución $\mathcal{D}$ es desconocida la obtención de h_B es imposible. Cabe destacar que no podemos esperar que un algoritmo de aprendizaje encuentre una regla cuyo riesgo sea menor que el dado por h_B. Por tanto, nos centraremos en buscar reglas que nos proporcionen un riesgo que no se desvíe en exceso del obtenido por la regla de Bayes.

Proposición 1.1. *Para el problema de clasificación binaria, dada cualquier distribución $\mathcal{D}$ sobre $\mathcal{X}\times\mathcal{Y}$ se tiene:*

1. *Si la función de pérdida considerada es la pérdida cuadrática, la regla de Bayes que minimiza el riesgo es $h_B(x) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}[Y|X=x]$.*

2. *Si la función de pérdida considerada es la pérdida 0-1, entonces la regla de Bayes que minimiza el riesgo es*

$$h_B(x) = \begin{cases} 1 & \textit{si } \mathbb{P}_{\mathcal{D}}[Y=1|x] \geq \frac{1}{2} \\ 0 & \textit{en cualquier otro caso} \end{cases}$$

Demostración. Estudiemos los dos casos.

1. Para el primer caso, es una consecuencia trivial de la descomposición del riesgo obtenida en (1.1).

2. Calculemos una expresión del riesgo de cualquier regla $h \in \mathcal{H}$ con la función de pérdida ℓ_{0-1}.

$$\begin{aligned} L_{\mathcal{D}}(h) &= \mathop{\mathbb{E}}_{(x,y)\sim D}[\ell_{0-1}(h,(x,y))] = \mathop{\mathbb{E}}_{(x,y)\sim D}[\mathbb{I}_{h(x)\neq y}] = \\ &= \mathbb{P}_{\mathcal{D}}[\{(x,y) : h(x) \neq y\}] = \\ &= \mathbb{P}_{\mathcal{D}}[Y=1]\mathbb{P}_{\mathcal{D}}[h(x)=0|Y=1] + \mathbb{P}_{\mathcal{D}}[Y=0]\mathbb{P}_{\mathcal{D}}[h(x)=1|Y=0]. \end{aligned} \tag{1.2}$$

Por su parte, $\mathbb{P}_{\mathcal{D}}[h(x)\neq y] = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\mathbb{I}_{h(x)\neq y}\right] = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\mathbb{I}_{h(x)\neq y}|X\right]\right]$.
Se define $\phi(x) = \mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}}\left[\mathbb{I}_{h(x)\neq y}|X=x\right]$.

$$\phi(X=x) = \begin{cases} \mathbb{P}[Y=1|X=x] & \text{si } h(x)=0 \\ 1-\mathbb{P}[Y=1|X=x] & \text{si } h(x)=1 \end{cases}$$

Entonces si $\mathbb{P}[Y=1|X=x] < 1-\mathbb{P}[Y=1|X=x]$ escogemos $h(x)=0$. Si $\mathbb{P}[Y=1|X=x] > 1-\mathbb{P}[Y=1|X=x]$ entonces escogemos $h(x)=1$. Por tanto, si $\mathbb{P}[Y=1|X=x] \geq 1/2$, entonces $h(x)=1$. Esto justifica la expresión de h_B en 2.

□

Dado que el objetivo es encontrar una función $h \in \mathcal{H}$ cuyo riesgo sea mínimo y desconocemos $\mathcal{D}$, tiene sentido fijarnos en la información parcial que obtenemos sobre $\mathcal{D}$ a través de S. Por consiguiente, se define el *riesgo empírico*.

Definición 1.2. *El riesgo empírico de una regla $h \in \mathcal{H}$ sobre una muestra de entrenamiento S, con $S = (X_1,Y_1),...,(X_n,Y_n)$, viene dado por*

$$L_S(h) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(h,(X_i,Y_i)\right).$$

Para cada $h \in \mathcal{H}$ fija, en virtud de la Ley de los Grandes Números, se tiene que

$$L_S(h) \to_{c.s.} L_{\mathcal{D}}(h).$$

Por tanto, puesto que el riesgo empírico se puede emplear como estimador del riesgo, parece lógico tratar de encontrar aquella regla $h \in \mathcal{H}$ que minimice el riesgo empírico. El algoritmo de aprendizaje definido a continuación se basa en esta idea.

Definición 1.3. *(Algoritmo ERM) Dada una clase de reglas $\mathcal{H}$ y un conjunto de entrenamiento S, se define el algoritmo de minimización del riesgo empírico (ERM) como el algoritmo de aprendizaje capaz de obtener $\hat{h} \in \mathcal{H}$ tal que $L_S(\hat{h}) = \min_{h \in \mathcal{H}} L_S(h)$*

Las siglas ERM responden a su nombre en inglés *Empirical Risk Minimization* y es habitual referirnos a $\hat{h}$ como la regla ERM.

Nos interesa averiguar si realmente el riesgo de la regla $\hat{h}$ se acerca al menor riesgo posible dado por $L_{\mathcal{D}}(h_B)$, siendo h_B, como hemos comentado anteriormente, la regla de Bayes. Luego, deseamos estudiar el *exceso de riesgo*: $L_{\mathcal{D}}(\hat{h}) - L_{\mathcal{D}}(h_B)$. Teniendo en cuenta $L_S(\hat{h}) \leq L_S(h)$ para todo $h \in \mathcal{H}$, se obtiene

$$\begin{aligned} L_{\mathcal{D}}(\hat{h}) - L_{\mathcal{D}}\left(h_B\right) &= L_{\mathcal{D}}(\hat{h}) - L_S(\hat{h}) + L_S(\hat{h}) - L_{\mathcal{D}}(h) + L_{\mathcal{D}}(h) - L_{\mathcal{D}}\left(h_B\right) \\ &\leq L_{\mathcal{D}}(\hat{h}) - L_S(\hat{h}) + L_S(h) - L_{\mathcal{D}}(h) + L_{\mathcal{D}}(h) - L_{\mathcal{D}}\left(h_B\right) \\ &\leq 2 \sup_{h \in \mathcal{H}} |L_S(h) - L_{\mathcal{D}}(h)| + L_{\mathcal{D}}(h) - L_{\mathcal{D}}\left(h_B\right). \end{aligned}$$

Por lo tanto,

$$L_{\mathcal{D}}(\hat{h}) - L_{\mathcal{D}}(h_B) \leq 2 \sup_{h \in \mathcal{H}} |L_S(h) - L_{\mathcal{D}}(h)| + \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) - L_{\mathcal{D}}(h_B). \tag{1.3}$$

El término $2 \sup_{h \in \mathcal{H}} |L_S(h) - L_{\mathcal{D}}(h)|$ se llama *error de estimación* y el término $\min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) - L_{\mathcal{D}}(h_B)$ es conocido como *error de aproximación*. El error de estimación aumenta cuanto mayor es la clase $\mathcal{H}$ y el error de aproximación aumenta cuanto más pequeño es el tamaño de $\mathcal{H}$. Por tanto, debemos encontrar un equilibrio entre estos dos términos. Tomar una clase de reglas $\mathcal{H}$ grande nos conducirá hacia un error de aproximación menor pero al mismo tiempo hará crecer el error de estimación. Este fenómeno se llama *overfitting* o *sobreajuste*. El caso contrario en el que $\mathcal{H}$ es más pequeña y, por tanto, el error de estimación decrece pero el error de aproximación aumenta, es denominado *subajuste*. Hemos de elegir $\mathcal{H}$ de tal manera que no caigamos ni en el sobreajuste ni en el subajuste. Una de las técnicas empleadas para evitar el sobreajuste es la *regularización*.

1.2. Descenso de gradiente.

La minimización del riesgo es el principal objetivo del problema de aprendizaje. En este punto, es necesario recurrir a los métodos de optimización de funciones. Gran parte de las técnicas de optimización empleadas en aprendizaje automático tienen como base el *método de descenso de gradiente*. Dicho procedimiento consiste en un algoritmo iterativo de primer orden cuyo objetivo es encontrar un mínimo local de una función diferenciable. El descenso de gradiente se atribuye a Cauchy (ver [7]), quien lo propuso por primera vez en 1847 para resolver sistemas de ecuaciones lineales.

Describiremos el procedimiento de descenso de gradiente para la minimización de una función $f : \mathbb{R}^d \to \mathbb{R}$ diferenciable y convexa. Si tomamos $x_1 \in \mathbb{R}^d$, para cada $t \geq 1$, la siguiente iteración es

$$x_{t+1} = x_t - \eta \cdot \nabla f(x_t), \quad \eta > 0. \tag{1.4}$$

$\nabla f(x_t)$ contiene la información de cuánto crece la función en el punto $f(x_t)$ por cada dimensión de forma independiente. El método aprovecha esta información para moverse precisamente en sentido contrario al crecimiento de la función hasta encontrar un mínimo. En cada iteración, graduada por el parámetro η que llamaremos *longitud de paso*, nos moveremos en sentido opuesto al del gradiente. La dirección $-\nabla f(x_t)$ efectivamente es de mayor descenso pues minimiza el polinomio de Taylor de orden 1 de la función. Veámoslo.

Supongamos que en cada paso el desplazamiento se realiza en la dirección de $u \in \mathbb{R}^d$ unitario, es decir, $x_{t+1} = x_t + \eta u$. Deseamos que $f(x_{t+1}) - f(x_t)$ sea mínimo, pues esto supone haber llegado al mínimo o estar próximos a él. Definimos $g : \mathbb{R} \to \mathbb{R}$ como $g(\eta) = f(x_t + \eta u)$. Recurriendo al polinomio de Taylor de orden 1 de g en 0, se obtiene

$$f(x_t + \eta u) = g(\eta) = g(0) + g'(0)\eta + O(\eta^2) = f(x_t) + g'(0)\eta + O(\eta^2). \tag{1.5}$$

Por tanto,

$$f(x_t + \eta u) - f(x_t) = g(0) + g'(0)\eta + O(\eta^2). \tag{1.6}$$

En virtud de la regla de la cadena, $g'(\eta) = \nabla f(x_t + \eta u)^T u$. Volviendo a (1.5),

$$f(x_{t+1}) - f(x_t) = \eta \nabla f(x_t)^T u + O(\eta^2). \tag{1.7}$$

El vector unitario u que minimiza la expresión anterior será $u = \frac{-\nabla f(x_t)}{\|\nabla f(x_t)\|}$.

El procedimiento se detiene cuando la diferencia entre dos iterantes sucesivos está por debajo de un determinado parámetro denominado *tolerancia*. En ocasiones, también es habitual simplemente limitar el número de iteraciones.

Conviene destacar que cuando la función de la que buscamos el mínimo es fuertemente convexa y su gradiente es de Lipschitz, entonces el procedimiento del descenso de gradiente con longitud de paso apropiada converge al mínimo global a velocidad exponencial (ver [9]). Sin embargo, estas garantías de convergencia al mínimo no son aplicables en nuestro problema pues la función no es siquiera convexa en la mayor parte de las ocasiones.

El descenso de gradiente es la base del llamado *algoritmo de retropropagación*, que desarrollaremos detalladamente en el tercer capítulo, el cual tiene un papel fundamental en el entrenamiento de redes neuronales. Además, cuando presentemos los optimizadores de redes, veremos cómo estos no son más que modificaciones del procedimiento anterior.

Capítulo 2

Aprendizaje probablemente aproximadamente correcto.

2.1. Aprendizaje PAC.

Con el objetivo de determinar cuando una clase de reglas realmente lleva a cabo una buena predicción, se introduce el concepto de *aprendizaje probablemente aproximadamente correcto (PAC)*. El propósito es escoger, con una alta probabilidad ("problamemente"), una regla cuyo riesgo sea pequeño ("aproximadamente correcto"). La noción de aprendizaje PAC se debe principalmente a Leslie Valiant (ver [3]) quien en 1984 trabajó y profundizó sobre resultados anteriormente desarrollados por Vladimir Vapnik (ver [4]).

Vapnik definió formalmente la noción de aprendizaje como se señala a continuación.

Definición 2.1. *(Aprendizaje PAC) Un clase de reglas $\mathcal{H}$ es aprendible de forma probablemente aproximadamente correcta (aprendible PAC o simplemente PAC) si existe una función $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ y un algoritmo de aprendizaje A que proporciona la regla $h^* \in \mathcal{H}$ con la siguiente propiedad: para cada $\epsilon, \delta \in (0,1)$ y para cada distribución $\mathcal{D}$ sobre $\mathcal{X} \times \mathcal{Y}$, si h^* es obtenida a partir de una muestra de entrenamiento S de tamaño $m \geq m_{\mathcal{H}}$, entonces, con probabilidad al menos $1 - \delta$,*

$$L_{\mathcal{D}}(h^*) \leq \min_{h \in \mathcal{H}} L_{\mathcal{D}}(h) + \epsilon. \tag{2.1}$$

Probaremos que lo esencial para que una clase de reglas sea aprendible PAC es que para esa clase sea posible aproximar el riesgo uniformemente mediante el riesgo empírico. El concepto clave en este punto es el de ϵ-representatividad.

Definición 2.2. *(ϵ-representatividad) Se dice que una muestra S es ϵ-representativa si satisface*

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \epsilon \text{ para todo } h \in \mathcal{H}.$$

La función $m_{\mathcal{H}} : (0,1)^2 \to \mathbb{N}$ en la definición 2.1 de aprendizaje PAC determina la denominada *complejidad muestral* de la clase $\mathcal{H}$. Dicha función establece el número mínimo de observaciones necesarias para garantizar que se cumpla la ecuación (2.1) y está asociada al error de estimación. Tomamos $\mathcal{D}$ la distribución que sigue la muestra de entrenamiento S de tamaño n y denotamos por $\mathbb{P}_{\mathcal{D}}$ a la probabilidad asociada a $\mathcal{D}$. Se supone que existe un función $\tilde{m} : (0,1)^2 \to \mathbb{N}$ tal que para toda distribución $\mathcal{D}$ y todo $\epsilon > 0$

$$\mathbb{P}_{\mathcal{D}}\left(\sup_{h\in\mathcal{H}} |L_S(h) - L_{\mathcal{D}}(h)| > \frac{\epsilon}{2}\right) \leq \delta \text{ si } n \geq \tilde{m}(\epsilon, \delta).$$

Sea $\hat{h}$ la regla ERM, entonces, con probabilidad al menos $1-\delta$, para cualquiera $h \in \mathcal{H}$

$$L_{\mathcal{D}}(\hat{h}) \leq L_S(\hat{h}) + \frac{\epsilon}{2} \leq L_S(h) + \frac{\epsilon}{2} \leq L_{\mathcal{D}}(h) + \epsilon.$$

Por tanto, con probabilidad al menos $1-\delta$,

$$L_{\mathcal{D}}(\hat{h}) \leq \min_{h\in\mathcal{H}} L_{\mathcal{D}}(h) + \epsilon.$$

Luego, $\mathcal{H}$ es aprendible PAC con complejidad muestral $m_{\mathcal{H}}(\epsilon, \delta) \leq \tilde{m}(\epsilon, \delta)$. Del razonamiento anterior, podemos además concluir que si la muestra S es $\frac{\epsilon}{2}$-representativa, la clase de reglas $\mathcal{H}$ para el algoritmo de aprendizaje ERM es aprendible PAC.

En este punto, introducimos la propiedad de *convergencia uniforme*.

Definición 2.3. *(Convergencia Uniforme) Se dice que una clase $\mathcal{H}$ tiene la propiedad de la convergencia uniforme si existe un algoritmo de aprendizaje A y una función $m_{\mathcal{H}}^{CU} : (0,1)^2 \to \mathbb{N}$ tal que para todo $\epsilon, \delta \in (0,1)$ si la muestra de entrenamiento S es de tamaño $m \geq m_{\mathcal{H}}^{CU}(\epsilon, \delta)$, entonces para cualquiera distribución $\mathcal{D}$, con probabilidad al menos $1-\delta$ sobre la elección de la muestra S, se tiene*

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \epsilon \text{ para todo } h \in \mathcal{H}.$$

Es habitual referirnos a $m_{\mathcal{H}}^{CU}$ como *complejidad muestral* de $\mathcal{H}$ para la convergencia uniforme. Dicha función ha de ser la mínima función válida para la definición anterior.

Una proposición casi inmediata y que nos relaciona el concepto de convergencia uniforme con las clases aprendibles PAC es la siguiente.

Proposición 2.4. *Si una clase $\mathcal{H}$ tiene la propiedad de la convergencia uniforme, entonces $\mathcal{H}$ es aprendible PAC. Además, $m_{\mathcal{H}}(\epsilon, \delta) \leq m_{\mathcal{H}}^{CU}(\epsilon/2, \delta)$.*

Demostración. Sea $\mathcal{H}$ una clase de reglas con la propiedad de la convergencia uniforme y S una muestra de entrenamiento. Entonces existe una función $m_{\mathcal{H}}^{CU}(\epsilon/2, \delta)$, la complejidad muestral para la convergencia uniforme de $\mathcal{H}$, y un algoritmo de aprendizaje A tal que con probabilidad al menos $1 - \delta$ se tiene $|L_{\mathcal{D}}(h) - L_S(h)| \leq \epsilon/2$ para todo $h \in \mathcal{H}$.

Tomamos $h^* = A(S)$ la regla obtenida por el algoritmo de aprendizaje A y $\hat{h}$ la regla obtenida por el algoritmo ERM. Luego

$$\begin{aligned} L_{\mathcal{D}}(h^*) - L_{\mathcal{D}}(\hat{h}) &= L_{\mathcal{D}}(h^*) - L_S(\hat{h}) + L_S(\hat{h}) - L_{\mathcal{D}}(\hat{h}) \\ &\leq L_{\mathcal{D}}(h^*) - L_S(h^*) + L_S(\hat{h}) - L_{\mathcal{D}}(\hat{h}) \\ &\leq |L_{\mathcal{D}}(h^*) - L_S(h^*)| + |L_S(\hat{h}) - L_{\mathcal{D}}(\hat{h})| \\ &\leq 2 \sup_{h \in \mathcal{H}} |L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon. \end{aligned}$$

Por tanto, $L_{\mathcal{D}}(h^*) - L_{\mathcal{D}}(\hat{h}) \leq \epsilon$ con probabilidad al menos $1 - \delta$. Se concluye que $\mathcal{H}$ es aprendible PAC y además, $m_{\mathcal{H}}(\epsilon, \delta) \leq m_{\mathcal{H}}^{CU}(\epsilon/2, \delta)$. □

La siguiente proposición nos proporciona una condición suficiente para que una clase sea aprendible PAC.

Proposición 2.5. *Se supone que la función de pérdida toma valores en el intervalo cerrado $[0, 1]$. Entonces, las clases finitas de reglas son aprendibles PAC.*

Demostración. Inicialmente fijamos $\epsilon > 0$ y $\delta \in (0, 1)$. Buscamos el tamaño m de la muestra S tal que con probabilidad al menos $1 - \delta$, $|L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon$ para todo $h \in \mathcal{H}$. Supongamos que S sigue la distribución $\mathcal{D}$ y denotemos por $\mathbb{P}_{S \sim \mathcal{D}}$ la probabilidad asociada a la distribución $\mathcal{D}$.

Deseamos que

$$\mathbb{P}_{S \sim \mathcal{D}} \left(|L_S(h) - L_{\mathcal{D}}(h)| \leq \epsilon \right) \geq 1 - \delta \quad \forall h \in \mathcal{H}.$$

Equivalentemente,

$$\mathbb{P}_{S \sim \mathcal{D}}(|L_S(h) - L_{\mathcal{D}}(h)| > \epsilon) \leq \delta \text{ para algún } h \in \mathcal{H}. \tag{2.2}$$

Lo anterior es posible reescribirlo como

$$\mathbb{P}_{S \sim \mathcal{D}} \left(\cup_{h \in \mathcal{H}} \{ S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon \} \right) \leq \delta.$$

Por la desigualdad de Bonferroni,

$$\mathbb{P}_{S\sim\mathcal{D}}\left(\cup_{h\in\mathcal{H}}\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}\right) \leq \sum_{h\in\mathcal{H}} \mathbb{P}_{S\sim\mathcal{D}}(\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}). \quad (2.3)$$

Para todo $h \in \mathcal{H}$, $\mathbb{E}[L_S(h)] = L_{\mathcal{D}}(h)$ y $\mathbb{P}\left[0 \leq L_S(h) \leq 1\right] = 1$ pues recordemos que por hipótesis la función de pérdida toma valores en $[0, 1]$. En virtud de la desigualdad de Hoeffding (ver Lema A.1 en apéndice A),

$$\mathbb{P}_{S\sim\mathcal{D}}(\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}) \leq 2\exp(-2m\epsilon^2).$$

Finalmente, se obtiene

$$\mathbb{P}_{S\sim\mathcal{D}}\left(\cup_{h\in\mathcal{H}}\{S : |L_S(h) - L_{\mathcal{D}}(h)| > \epsilon\}\right) \leq \sum_{h\in\mathcal{H}} 2\exp(-2m\epsilon^2) = 2|\mathcal{H}|\exp(-2m\epsilon^2).$$

Volviendo a la ecuación (2.2), ha de cumplirse para que la clase $\mathcal{H}$ sea aprendible PAC

$$\mathbb{P}_{S\sim\mathcal{D}}(|L_S(h) - L_{\mathcal{D}}(h)| > \epsilon) \leq 2|\mathcal{H}|\exp(-2m\epsilon^2) \leq \delta.$$

Basta con tomar una muestra de tamaño

$$m \geq \frac{\log(2|\mathcal{H}|/\delta)}{2\epsilon^2}. \quad (2.4)$$

.

De esta manera se satisface la desigualdad anterior y podemos concluir la demostración. □

En la demostración anterior, no solo se ha demostrado que toda clase de reglas finita es aprendible PAC sino que también se ha obtenido una cota para la complejidad muestral en la ecuación (2.4) en función del tamaño de la clase.

De hecho, si $\mathcal{H}$ posee la propiedad de la convergencia uniforme, entonces $m_{\mathcal{H}}^{CU}(\epsilon/2, \delta) \leq \left\lceil \frac{2\log(2|\mathcal{H}|/\delta)}{\epsilon^2} \right\rceil$ y, en virtud de la proposición 2.4,

$$m_{\mathcal{H}}(\epsilon, \delta) \leq m_{\mathcal{H}}^{CU}(\epsilon/2, \delta) \leq \left\lceil \frac{2\log(2|\mathcal{H}|/\delta)}{\epsilon^2} \right\rceil$$

Esta discusión es válida para clases finitas de reglas. Sin embargo, la mayor parte de las clases con las que trabajaremos serán infinitas o a lo sumo infinitas-numerables, luego necesitamos desarrollar alguna herramienta que nos permita caracterizar las clases aprendibles PAC con estas condiciones.

2.2. Teorema fundamental del aprendizaje estadístico.

El *Teorema fundamental del aprendizaje estadístico*, que enunciaremos posteriormente, constituirá la caracterización que buscamos para las clases de reglas aprendibles PAC. Sin embargo, previamente hemos de presentar ciertos conceptos y resultados necesarios. Además, cabe señalar que toda la teoría desarrollada a partir de este punto se centra en problemas de clasificación binaria.

El siguiente teorema nos muestra que no existe ningún algoritmo de aprendizaje universal sobre espacios de atributos $\mathcal{X}$ arbitrariamente grandes. De hecho, la regla proporcionada por el algoritmo de aprendizaje tendrá un riesgo elevado.

Teorema 2.6. *Sea A cualquier algoritmo de aprendizaje para el problema de clasificación binaria y ℓ_{0-1} la función de pérdida asociada. Sean $\mathcal{X}$ el espacio de atributos y S una muestra de entrenamiento de tamaño $m \leq |\mathcal{X}|/2$. Entonces existe una distribución $\mathcal{D}$ sobre $\mathcal{X} \times \{0,1\}$ tal que con probabilidad al menos de 1/7 sobre la elección de S se tiene $L_{\mathcal{D}}(A(S)) \geq 1/8$.*

Demostración. Sea m el tamaño de la muestra de entrenamiento y tomemos un subconjunto C de $\mathcal{X}$ de tamaño $2m$, $C = \{c_1, \ldots, c_{2m}\}$. Existen $T = 2^{2m}$ posibles funciones de C a $\{0,1\}$.

Sean $f_1, \ldots, f_T$ esas posibles funciones. Para cada una de ellas, sea $\mathcal{D}_i$ la distribución sobre $C \times \{0,1\}$ definida como

$$\mathbb{P}_{\mathcal{D}_i}(\{(x,y)\}) = \begin{cases} 1/|C| & \text{si } y = f_i(x) \\ 0 & \text{en cualquier otro caso} \end{cases}$$

Probaremos que para cualquier algoritmo de aprendizaje A que recibe una muestra de entrenamiento de tamaño m de $C \times \{0,1\}$ se cumple

$$\max_{i \in \{1,\ldots,T\}} \mathop{\mathbb{E}}_{S \sim D_i} [L_{\mathcal{D}_i}(A(S))] \geq 1/4. \tag{2.5}$$

Esto implica directamente que para cualquier algoritmo de aprendizaje A' que recibe una muestra de entrenamiento de tamaño m de $\mathcal{X} \times \{0,1\}$, existe una distribución $\mathcal{D}$ sobre $\mathcal{X} \times \{0,1\}$ tal que

$$\mathop{\mathbb{E}}_{S \sim D} \left[L_{\mathcal{D}}(A'(S))\right] \geq 1/4. \tag{2.6}$$

En virtud del Lema A.2 (ver apéndice A) se obtiene tomando $\theta = L_{\mathcal{D}}(A'(S))$

$$\mathbb{P}[\theta \geq 1/8] \geq \frac{\mathop{\mathbb{E}}_{S \sim D}[L_{\mathcal{D}}(A'(S)] - 1/8}{7/8}. \tag{2.7}$$

Gracias a la desigualdad (2.8), $\mathbb{P}[L_{\mathcal{D}}(A'(S)) \geq 1/8] \geq \frac{1}{7}$.

Por tanto, probar la ecuación (2.5) supone acabar la demostración.

Sea $k = (2m)^m$ el número de posibles muestras de m elementos de C. Fijada la distribución $\mathcal{D}_i$, sean $S_1^i, \ldots, S_k^i$ las posibles muestras. Todas las muestras de entrenamiento tienen la misma probabilidad de ser elegidas. Luego,

$$\mathop{\mathbb{E}}_{S\sim\mathcal{D}_i}[L_{\mathcal{D}_i}(A(S))] = \frac{1}{k}\sum_{j=1}^{k} L_{\mathcal{D}_i}\left(A\left(S_j^i\right)\right). \tag{2.8}$$

Tomando el máximo en la ecuación anterior se obtiene la siguiente acotación

$$\begin{aligned}\max_{i\in\{1,\ldots,T\}} \frac{1}{k}\sum_{j=1}^{k} L_{\mathcal{D}_i}\left(A\left(S_j^i\right)\right) &\geq \frac{1}{T}\sum_{i=1}^{T}\frac{1}{k}\sum_{j=1}^{k} L_{\mathcal{D}_i}\left(A\left(S_j^i\right)\right)\\ &= \frac{1}{k}\sum_{j=1}^{k}\frac{1}{T}\sum_{i=1}^{T} L_{\mathcal{D}_i}\left(A\left(S_j^i\right)\right)\\ &\geq \min_{j\in\{1,\ldots,k\}}\frac{1}{T}\sum_{i=1}^{T} L_{\mathcal{D}_i}\left(A\left(S_j^i\right)\right).\end{aligned}$$

Fijando $j \in \{1, \ldots, k\}$, si $S_j = (x_1, \ldots, x_m)$, entonces denotamos por $v_1, \ldots, v_p$ el conjunto de elementos de C que no aparecen en S_j. Claramente $p \geq m$ (en el mejor de los casos, cuando los elementos de S_j sean todos distintos entre sí, $p = m$). Se tiene para toda función $h : C \to \{0, 1\}$ y todo $i \in \{1, \ldots, T\}$

$$\begin{aligned}L_{\mathcal{D}_i}(h) = \mathop{\mathbb{E}}_{S\sim\mathcal{D}_i}[\ell_{0-1}(h, C, Y = \{0,1\}))] &\geq \frac{1}{2m}\sum_{r=1}^{p}\mathbb{I}_{[h(v_r)\neq f_i(v_r)]}\\ &\geq \frac{1}{2p}\sum_{r=1}^{p}\mathbb{I}_{[h(v_r)\neq f_i(v_r)]}.\end{aligned} \tag{2.9}$$

Por tanto,

$$\begin{aligned}\frac{1}{T}\sum_{i=1}^{T} L_{\mathcal{D}_i}\left(A\left(S_j^i\right)\right) &\geq \frac{1}{T}\sum_{i=1}^{T}\frac{1}{2p}\sum_{r=1}^{p}\mathbb{I}_{\left[A\left(S_j^i\right)(v_r)\neq f_i(v_r)\right]}\\ &= \frac{1}{2p}\sum_{r=1}^{p}\frac{1}{T}\sum_{i=1}^{T}\mathbb{I}_{\left[A\left(S_j^i\right)(v_r)\neq f_i(v_r)\right]}\\ &\geq \frac{1}{2}\cdot\min_{r\in\{1,\ldots,p\}}\frac{1}{T}\sum_{i=1}^{T}\mathbb{I}_{\left[A\left(S_j^i\right)(v_r)\neq f_i(v_r)\right]}.\end{aligned} \tag{2.10}$$

Fijando $r \in \{1, \ldots, p\}$. Podemos relacionar las funciones $f_1, \ldots, f_T$ en $T/2$ pares de tal manera que cada par $(f_i, f_{i'})$ cumple: $f_i(c) = f_{i'}(c)$ si $c \neq v_r$. Es decir, son la misma función en los puntos distintos de v_r.

Se tiene

$$\mathbb{I}_{\left[A\left(S_j^i\right)(v_r) \neq f_i(v_r)\right]} + \mathbb{I}_{\left[A\left(S_j^{i'}\right)(v_r) \neq f_{i'}(v_r)\right]} = 1$$

Finalmente

$$\frac{1}{T} \sum_{i=1}^{T} \mathbb{I}_{\left[A\left(S_j^i\right)(v_r) \neq f_i(v_r)\right]} = \frac{1}{2}$$

Combinando esta última ecuación con las anteriores (2.8), (2.9) y (2.10) obtenemos la ecuación (2.5). □

De este teorema, se deduce fácilmente el siguiente corolario.

Corolario 2.7. *Sea $\mathcal{X}$ un dominio infinito y $\mathcal{H}$ el conjunto de todas las posibles funciones de $\mathcal{X}$ a {0,1}. Entonces, $\mathcal{H}$ no es aprendible PAC.*

Demostración. Supongamos que $\mathcal{H}$ es aprendible PAC. De la definición de aprendizaje PAC, tomando $\epsilon < 1/8$ y $\delta < 1/7$, existirá un algoritmo de aprendizaje A y un entero $m = m(\epsilon, \delta)$ tales que para cada distribución $\mathcal{D}$ sobre $\mathcal{X} \times \{0, 1\}$ con probabilidad al menos $1 - \delta$ cuando A se aplica a una muestra de entrenamiento S de tamaño m siguiendo $\mathcal{D}$ se obtiene que $L_{\mathcal{D}}(A(S)) \leq \epsilon$.

Aplicando el Teorema 2.6, dado que $m \leq |\mathcal{X}|/2$ pues $|\mathcal{X}| = \infty$, para cualquier algoritmo de aprendizaje, y en particular para A, existirá una distribución $\mathcal{D}$ tal que con probabilidad al menos $1/7 > \delta$, se tiene $L_{\mathcal{D}}(A(S)) > 1/8 > \epsilon$, lo cual es absurdo. □

Este último corolario nos proporciona un ejemplo importante de un tipo de clase que no es aprendible PAC. De hecho, nos muestra claramente cómo el hecho de trabajar con una clase de reglas excesivamente grande (la clase de todas las posibles funciones de un dominio infinito a $\{0, 1\}$) no nos garantizar controlar el error.

2.2.1. Función de crecimiento, Lema de Sauer y dimensión-VC.

Hasta ahora hemos demostrado que las clases finitas de reglas son aprendibles PAC, pero ¿Qué hace que una clase sea aprendible y otra no? ¿Pueden las clases de reglas infinitas ser aprendibles PAC bajo ciertas hipótesis, y si es así, qué determina su complejidad muestral? Dicha caracterización fue presentada por Vladimir Vapnik y Alexey Chervonenkis en 1971 y se apoya en una noción combinatoria llamada la dimensión de

Vapnik-Chervonenkis (ver [5]). Además, el hecho de que una clase sea finita no es condición necesaria para que sea aprendible PAC. Previamente, introduciremos el concepto de *restricción de una clase de reglas* y *conjunto separable*.

Definición 2.8. *(Restricción de $\mathcal{H}$ a C) Sea $\mathcal{H}$ la clase de reglas de $\mathcal{X}$ a $\{0,1\}$ y $C = \{c_1, \ldots, c_m\} \subseteq \mathcal{X}$. La restricción de $\mathcal{H}$ a C es el conjunto de todas las funciones de C a $\{0,1\}$ que pueden ser obtenidas a partir de $\mathcal{H}$. Es decir,*

$$\mathcal{H}_C = \{(h(c_1), \ldots, h(c_m)) : h \in \mathcal{H}\}. \tag{2.11}$$

Representamos cada función de C a $\{0,1\}$ como un vector de $\{0,1\}^{|C|}$.

El concepto anterior puede ser tratado de manera equivalente en términos de conjuntos. Sea $\mathcal{H}$ una clase de reglas de $\mathcal{X}$ a $\{0,1\}$. Para cada $h \in \mathcal{H}$, podemos definir el subconjunto A_h de $\mathcal{X}$ como $A_h = \{a \in \mathcal{X}; h(a) = 1\}$. Identificando cada función de $\mathcal{H}$ con su correspondiente subconjunto A_h, $\mathcal{H}$ se convierte en una clase de subconjuntos, $\mathcal{H} = \{A_h;\ h \in \mathcal{H}\}$. La *restricción* de $\mathcal{H}$ a $C \subset \mathcal{X}$ denotada por $\mathcal{H}_C$ se define como

$$\mathcal{H}_C = C \cap \{A_h;\ h \in \mathcal{H}\} = C \cap \left(\bigcup_{h \in \mathcal{H}} A_h\right) = \bigcup_{h \in \mathcal{H}} (C \cap A_h). \tag{2.12}$$

Cabe destacar que en esta memoria emplearemos el término *conjunto separable* como se define a continuación. Esta definición nada tiene que ver con la habitual definición topológica de conjunto separable.

Definición 2.9. *(Conjunto separable) Sea $\mathcal{H}$ una clase de reglas y sea C un subconjunto finito de $\mathcal{X}$, diremos que el conjunto C es separable por la clase $\mathcal{H}$ cuando la restricción de $\mathcal{H}$ a C es el conjunto de todas las posibles funciones de C a $\{0,1\}$, es decir, $|\mathcal{H}_C| = 2^{|C|}$.*

En analogía con la terminología conjuntista utilizada anteriormente diremos que un conjunto $C \subseteq \mathcal{X}$ es *separable* por $\mathcal{H}$ si todos los subconjuntos $B \subseteq C$ pueden ser expresados como $B = C \cap A_h$ para una cierta función $h \in \mathcal{H}$. Por tanto, la restricción de $\mathcal{H}$ a C contiene precisamente todos los posibles subconjuntos de C y $|\mathcal{H}_C| = 2^{|C|}$.

En busca de la caracterización correcta de las clases de reglas aprendibles PAC, se introduce el concepto de dimensión Vapnik-Chervonenkis que juega un papel fundamental en la teoría del aprendizaje estadístico. Dicha noción no es una medida al uso, sino una medida combinatoria.

Definición 2.10. *(Dimensión-VC) La dimensión-VC de una clase de reglas $\mathcal{H}$ es el supremo de los cardinales de todos los conjuntos separables por $\mathcal{H}$. Denotaremos esta dimensión por $VCdim(\mathcal{H})$. Si $\mathcal{H}$ puede separar conjuntos arbitrariamente grandes, entonces diremos que $\mathcal{H}$ tiene dimensión-VC infinita.*

Ejemplo 2.11. Plantearemos dos ejemplos de cálculo de la dimensión-VC de dos clases de reglas. Para probar que una clase $\mathcal{H}$ tiene $VCdim(\mathcal{H}) = d$ se han satisfacer las siguientes proposiciones:

1. Existe un conjunto C de tamaño d tal que C es separable por $\mathcal{H}$.

2. Todo conjunto de tamaño $d+1$ no es separable por $\mathcal{H}$.

Ejemplo 1.

Sea $\mathcal{H}$ la clase de intervalos sobre $\mathbb{R}$, es decir, $\mathcal{H} = \{h_{a,b} : a, b \in \mathbb{R}, a < b\}$ donde $h_{a,b}(x) = \mathbb{I}_{[(a,b)]}(x)$. Empleando la misma notación que en 2.12, la clase $\mathcal{H}$ es el conjunto de todos los intervalos abiertos de $\mathbb{R}$ pues para cada $h_{a,b} \in \mathcal{H}, A_{h_{a,b}} = (a, b)$.

Comenzaremos razonando para un conjunto de cardinal 2. Sea, por ejemplo, $C = \{1, 2\}$ y sean

$$\begin{aligned} &A_1 = \{(a,b) :\ a < b < 1 \text{ ó } 2 < a < b, a, b \in \mathbb{R}\}, A_2 = \{(a,b) :\ a < 1 < b < 2, a, b \in \mathbb{R}\} \\ &A_3 = \{(a,b) :\ a < 1 < 2 < b, a, b \in \mathbb{R}\},\ A_4 = \{(a,b) :\ 1 < a < 2 < b, a, b \in \mathbb{R}\}. \end{aligned} \tag{2.13}$$

Se tiene $\mathcal{H} = \bigcup_{i \in \{1,\ldots,4\}} A_i$. Obviamente $C \cap A_1 = \emptyset$, $C \cap A_2 = \{1\}$, $C \cap A_3 = \{1, 2\}$ y $C \cap A_4 = \{2\}$. Por tanto, la restricción de $\mathcal{H}$ a C viene dada por:

$$\mathcal{H}_C = \emptyset \cup \{1\} \cup \{2\} \cup \{1, 2\}$$

Dado que $|\mathcal{H}_C| = 2^{|C|} = 4$, podemos concluir que C es separable por $\mathcal{H}$ y $VCdim(\mathcal{H}) \geq 2$.

Veamos que sucede con un conjunto de tres elementos, $C = \{c_1, c_2, c_3\}$. Sin pérdida de generalidad, podemos suponer $c_1 \leq c_2 \leq c_3$. Intentando llevar a cabo una división de $\mathcal{H}$ similar a la anterior, se concluye que es imposible obtener un conjunto de intervalos que contengan a su vez a c_1 y c_3 pero no a c_2. Luego, C no es separable por $\mathcal{H}$, y por tanto, la dimensión-VC de $\mathcal{H}$ es exactamente 2.

Ejemplo 2.

Definimos la siguiente clase de reglas: $\mathcal{H} = \{h_{a,b,s} : a \leq b, s \in \{-1, 1\}\}$ donde

$$h_{a,b,s}(x) = \begin{cases} s & \text{si } x \in [a, b] \\ -s & \text{si } x \notin [a, b] \end{cases} \tag{2.14}$$

Cada una de las funciones de la clase $\mathcal{H}$ viene determinada por 3 parámetros. Veamos cual es su dimensión-VC.

Tomemos inicialmente un conjunto de 3 elementos, $C = \{1, 2, 3\}$. Realicemos una tabla de valores y veamos que sucede.

a	b	s	1	2	3
$a < 1$	$b > 3$	-1	-1	-1	-1
$2 < a < 3$	$b > 3$	1	-1	-1	1
$1 < a < 2$	$2 < b < 3$	1	-1	1	-1
$1 < a < 2$	$b > 3$	1	-1	1	1
$a < 1$	$1 < b < 2$	1	1	-1	-1
$1 < a < 2$	$2 < b < 3$	-1	1	-1	1
$a < 1$	$2 < b < 3$	1	1	1	-1
$a < 1$	$b > 3$	1	1	1	1

Como podemos observar en la tabla anterior, tomando adecuadamente los valores de los parámetros a, b y s podemos obtener todas las configuraciones posibles de clasificación del conjunto C (en total son 8 configuraciones posibles). Luego podemos concluir que $VCdim(\mathcal{H}) \geq 3$.

El razonamiento para un conjunto de cuatro elementos es análogo al ejemplo anterior. Tomemos $C = \{x_1, x_2, x_3, x_4\}$. Sin pérdida de generalidad, podemos suponer $x_1 \leq x_2 \leq x_3 \leq x_4$. Por ejemplo, la asignación -1 para x_1 y x_3 y a su vez la etiqueta 1 para x_2 y x_4, no es posible obtenerla a partir de ninguna función $h_{a,b,s}$ pues no existe intervalo cerrado que contenga a x_1 y a x_3 pero no a x_2. Luego, podemos concluir que $VCdim(\mathcal{H}) = 3$.

Observación 2.12. Como sucede en los ejemplo anterior, es habitual que la dimensión-VC de una clase de reglas coincida con el número de parámetros que determinan cada regla de la clase. Sin embargo, esto no es siempre así. Por ejemplo, considerando $\mathcal{X} = \mathbb{R}$ y la clase $\mathcal{H} = \{h_\theta; \theta \in \mathbb{R}\}$ donde $h_\theta : \mathbb{R} \to \{0, 1\}$ es definida por $h_\theta(x) = \lceil 0.5 \sin(\theta x) \rceil$, se tiene que cada h_θ viene determinada por un solo parámetro, θ, y $VCdim(\mathcal{H}) = \infty$.

En este punto, extraemos una consecuencia importante del Teorema 2.6.

Corolario 2.13. *Sea $\mathcal{H}$ una clase de reglas de $\mathcal{X}$ a $\{0, 1\}$ y sea m el tamaño de la muestra de entrenamiento. Supongamos que existe un subconjunto $C \subseteq \mathcal{X}$ de tamaño $2m$ tal que C es separable por $\mathcal{H}$. Entonces, para cualquier algoritmo de aprendizaje A, existe una distribución $\mathcal{D}$ sobre $\mathcal{X} \times \{0, 1\}$ con probabilidad al menos 1/7 sobre la elección de la muestra S de que $L_{\mathcal{D}}(A(S)) \geq 1/8$.*

Intuitivamente, este corolario nos muestra como el hecho de que aunque C sea separable por la clase $\mathcal{H}$, si únicamente contamos con una muestra con la mitad de elementos de C, las etiquetas de estas observaciones no nos aportan información sobre las etiquetas del resto del conjunto C.

Se deduce el siguiente resultado fundamental.

Teorema 2.14. *Sea $\mathcal{H}$ una clase de reglas con dimensión-VC infinita. Entonces, $\mathcal{H}$ no es aprendible PAC.*

Demostración. Supongamos que $\mathcal{H}$ es una clase de reglas con $VCdim(\mathcal{H}) = \infty$. Por tanto, para cada muestra de entrenamiento de tamaño m, existe un subconjunto de tamaño $2m$ tal que es separable por $\mathcal{H}$. En virtud del corolario 2.13, se concluye que $\mathcal{H}$ no es aprendible PAC. □

El teorema anterior es esencial en la teoría desarrollada pues nos proporciona una condición suficiente para descartar aquellas clases de reglas que no serán útiles para el aprendizaje. Nuestro objetivo es probar el recíproco, el cual constituirá una parte del *Teorema fundamental del aprendizaje*. La clave para probar el recíproco es que cuando una clase tiene dimensión-VC finita, goza de la propiedad de la convergencia uniforme. Desde un punto de vista técnico, es necesario recurrir a un resultado combinatorio conocido como *Lema de Sauer*, que presentamos a continuación. En primer lugar, introducimos la noción de *función de crecimiento*.

Definición 2.15. *(Función de crecimiento) Sea $\mathcal{H}$ una clase de reglas. Entonces la función de crecimiento de $\mathcal{H}$, denotada por $\tau_{\mathcal{H}} : \mathbb{N} \to \mathbb{N}$, se define como*

$$\tau_{\mathcal{H}}(m) = \text{máx}_{C \subset \mathcal{X}:|C|=m} |\mathcal{H}_C|$$

Por tanto, el número $\tau_{\mathcal{H}}(m)$ es el número de funciones de C a $\{0, 1\}$ que pueden ser obtenidas restringiendo $\mathcal{H}$ a C. Evidentemente, $\tau_{\mathcal{H}}(m) = 2^m$ si $m \leq VCdim(\mathcal{H})$. La función de crecimiento goza de determinadas propiedades que nos serán útiles posteriormente.

Lema 2.16. *(Función de crecimiento de la composición) Sea $\mathcal{F}_1$ el conjunto de funciones de $\mathcal{X}$ a Z y $\mathcal{F}_2$ el conjunto de funciones de Z a $\mathcal{Y}$. Sea $\mathcal{H} = \mathcal{F}_1 \circ \mathcal{F}_2$ la clase de reglas cuyas funciones son las composiciones de las funciones de $\mathcal{F}_1$ con $\mathcal{F}_2$. Esto es, para cada $f_1 \in \mathcal{F}_1$ y $f_2 \in \mathcal{F}_2$, existe $h \in \mathcal{H}$ tal que $h(\boldsymbol{x}) = f_2(f_1(\boldsymbol{x}))$. Entonces $\tau_{\mathcal{H}}(m) \leq \tau_{\mathcal{F}_1}(m)\tau_{\mathcal{F}_2}(m)$.*

Demostración. Sea $C \subseteq \mathcal{X}$ subconjunto finito de $\mathcal{X}$ con $C = \{c_1, \ldots, c_m\}$. Las restricciones de $\mathcal{H}$, $\mathcal{F}_1$ y $\mathcal{F}_2$ a C vienen dadas por

$$\begin{aligned} \mathcal{H}_C &= \{(h(c_1), \ldots, h(c_m)) : h \in \mathcal{H}\} \\ \mathcal{F}_{1_C} &= \{(f_1(c_1), \ldots, f_1(c_m)) : f_1 \in \mathcal{F}_1\} \\ \mathcal{F}_{2_C} &= \{(f_2(c_1), \ldots, f_2(c_m)) : f_2 \in \mathcal{F}_2\}. \end{aligned} \tag{2.15}$$

Por tanto, siendo $\mathcal{H} = \mathcal{F}_1 \circ \mathcal{F}_2$

$$\begin{aligned}
|\mathcal{H}_C| &= |\{(h_(c_1), \ldots, h(c_m)) : h \in \mathcal{H}\}| \\
&= |\{(f_2(f_1(c_1)), \ldots, f_2(f_1(c_m))) : f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_2\}| \\
&= \left| \bigcup_{f_1 \in \mathcal{F}_{1_C}} \{(f_2(f_1(c_1)), \ldots, f_2(f_1(c_m))) : f_2 \in \mathcal{F}_2\} \right| \\
&\leq \sum_{f_1 \in \mathcal{F}_{1_C}} |\{(f_2(f_1(c_1)), \ldots, f_2(f_1(c_m))) : f_2 \in \mathcal{F}_2\}| \\
&= \sum_{f_1 \in \mathcal{F}_{1_C}} \tau_{\mathcal{F}_2}(m) \leq \tau_{\mathcal{F}_1}(m)\tau_{\mathcal{F}_2}(m).
\end{aligned} \tag{2.16}$$

Se obtiene $\tau_{\mathcal{H}}(m) \leq \tau_{\mathcal{F}_1}(m)\tau_{\mathcal{F}_2}(m)$. □

Lema 2.17. *(Función de crecimiento del producto) Sean $\mathcal{F}_1$ y $\mathcal{F}_2$ los conjuntos de funciones de $\mathcal{X}$ a $\mathcal{Y}$. Definimos $\mathcal{H} = \mathcal{F}_1 \times \mathcal{F}_2$ como la clase del producto cartesiano. Para cada $f_1 \in \mathcal{F}_1$ y $f_2 \in \mathcal{F}_2$, existe un $h \in \mathcal{H}$ tal que $h(x) = (f_1(x), f_2(x))$.Entonces, $\tau_{\mathcal{H}}(m) \leq \tau_{\mathcal{F}_1}(m)\tau_{\mathcal{F}_2}(m)$.*

Demostración. Sea $C \subseteq \mathcal{X}$ subconjunto de $\mathcal{X}$ con $C = \{c_1, \ldots, c_m\}$. Se tiene

$$\begin{aligned}
|\mathcal{H}_C| &= |\{((f_1(\mathbf{c}_1), f_2(\mathbf{c}_1)), \ldots, (f_1(\mathbf{c}_m), f_2(\mathbf{c}_m))) : f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_2\}| \\
&= |\{((f_1(\mathbf{c}_1), \ldots, f_1(\mathbf{c}_m)), (f_2(\mathbf{c}_1), \ldots, f_2(\mathbf{c}_m))) : f_1 \in \mathcal{F}_1, f_2 \in \mathcal{F}_2\}| \\
&= |\mathcal{F}_{1C} \times \mathcal{F}_{2C}| = |\mathcal{F}_{1C}| \cdot |\mathcal{F}_{2C}|.
\end{aligned} \tag{2.17}$$

Por tanto, se concluye $\tau_{\mathcal{H}}(m) \leq \tau_{\mathcal{F}_1}(m)\tau_{\mathcal{F}_2}(m)$. □

Lema 2.18. *(Lema de Sauer) Sea $\mathcal{H}$ una clase de reglas con $VCdim(\mathcal{H}) \leq d < \infty$. Entonces, para todo m, $\tau_{\mathcal{H}}(m) \leq \sum_{i=0}^{d} \binom{m}{i}$. Si $m > d+1$, entonces $\tau_{\mathcal{H}}(m) \leq (\frac{em}{d})^d$.*

Demostración. Sean $\mathcal{H}$ una clase de reglas con $VCdim(\mathcal{H}) \leq d < \infty$ y $C \subseteq \mathcal{X}$ con $C = \{c_1, \ldots, c_m\}$. Basta demostrar que se cumple

$$|\mathcal{H}_C| \leq |\{B \subseteq C : \mathcal{H} \text{ separa } B\}|. \tag{2.18}$$

La ecuación (2.18) es equivalente al Lema de Sauer pues si $VCdim(\mathcal{H}) \leq d$ entonces no existe conjunto de tamaño mayor que d tal que sea separable por $\mathcal{H}$. Por tanto,

$$|\{B \subseteq C : \mathcal{H} \text{ separa } B\}| \leq \sum_{i=0}^{d} \binom{m}{i}$$

Se nos presentan dos casos:

1. Si $m > d+1$, entonces $\tau_{\mathcal{H}}(m) \leq (\frac{em}{d})^d$. Se deduce directamente del Lema 2.19 enunciado inmediatamente después del final de la demostración.

2. Para el caso $m \leq d+1$ emplearemos un argumento inductivo.
Si $m = 1$, entonces $|\mathcal{H}_C| \leq 2$ (el número de posibles funciones de $C = \{c_1\}$ a $\{0,1\}$ será como máximo 2). En este caso, ambos miembros de la desigualdad (2.18) serán o bien iguales a 1 o a 2. Se considera que el conjunto vacío siempre es separado por $\mathcal{H}$.

   Supongamos que la ecuación (2.18) se cumple para todo conjunto de cardinal $k < m$, probémosla para conjuntos de m elementos.

   Consideramos $C' = \{c_2, \ldots, c_m\}$. Definimos los dos siguientes conjuntos:

$$Y_0 = \{(y_2, \ldots, y_m) : (0, y_2, \ldots, y_m) \in \mathcal{H}_C \text{ ó } (1, y_2, \ldots, y_m) \in \mathcal{H}_C\}$$
$$Y_1 = \{(y_2, \ldots, y_m) : (0, y_2, \ldots, y_m) \in \mathcal{H}_C \text{ y } (1, y_2, \ldots, y_m) \in \mathcal{H}_C\}$$

   Evidentemente, $|\mathcal{H}_C| = |Y_0| + |Y_1|$. Además, dado que $Y_0 = \mathcal{H}_{C'}$, en virtud de la hipótesis de inducción se tiene

$$|Y_0| = |\mathcal{H}_{C'}| \leq \left|\left\{B \subseteq C' : \mathcal{H} \text{ separa } B\right\}\right| = |\{B \subseteq C : c_1 \notin B \text{ y } \mathcal{H} \text{ separa } B\}|$$

   Definimos $\mathcal{H}' \subseteq \mathcal{H}$ como

$$\mathcal{H}' = \left\{h \in \mathcal{H} : \exists h' \in \mathcal{H} \text{ con } \left(1 - h'(c_1), h'(c_2), \ldots, h'(c_m)\right) = \left(h(c_1), h(c_2), \ldots, h(c_m)\right)\right\}$$

   $\mathcal{H}'$ contiene todos las reglas de $\mathcal{H}$ que coinciden en C' y difieren en c_1. Es claro que si $\mathcal{H}'$ separa $B \subset C'$, entonces también separa el conjunto $B \cup \{c_1\}$. Además, $Y_1 = \mathcal{H}'_C$ y aplicando la hipótesis de inducción a $\mathcal{H}'$ y C' se tiene

$$\begin{aligned} |Y_1| = \left|\mathcal{H}'_{C'}\right| &\leq \left|\left\{B \subseteq C' : \mathcal{H}' \text{ separa } B\right\}\right| = \left|\left\{B \subseteq C' : \mathcal{H}' \text{ separa } B \cup \{c_1\}\right\}\right| \\ &= \left|\left\{B \subseteq C : c_1 \in B \text{ y } \mathcal{H}' \text{ separa } B\right\}\right| \leq |\{B \subseteq C : c_1 \in B \text{ y } \mathcal{H} \text{ separa } B\}| . \end{aligned}$$

   Por tanto,

$$\begin{aligned} |\mathcal{H}_C| &= |Y_0| + |Y_1| \\ &\leq |\{B \subseteq C : c_1 \notin B \text{ y } \mathcal{H} \text{ separa } B\}| + |\{B \subseteq C : c_1 \in B \text{ y } \mathcal{H} \text{ separa } B\}| \\ &= |\{B \subseteq C : \mathcal{H} \text{ separa } B\}| \end{aligned}$$

   Esto concluye la prueba.

□

Para el caso 1, ha sido necesario usar el siguiente lema. Una demostración del mismo puede encontrase en [1].

Lema 2.19. *Sea m,d dos enteros positivos tal que $d \leq m-2$. Entonces,*

$$\sum_{k=0}^{d} \left(\begin{array}{c} m \\ k \end{array} \right) \leq \left(\frac{em}{d}\right)^d$$

Dos propiedades interesantes que nos acotan la dimensión-VC de la unión de clases son las siguientes.

Lema 2.20. *1. Sean $\mathcal{H}_1$ y $\mathcal{H}_2$ clases de reglas sobre el mismo dominio $\mathcal{X}$. Consideramos $d = \text{máx}\{VCdim(\mathcal{H}_1), VCdim(\mathcal{H}_2)\}$. Entonces,*

$$VCdim(\mathcal{H}_1 \cup \mathcal{H}_2) \leq 2d+1. \tag{2.19}$$

2. Sean $\mathcal{H}_1, \ldots, \mathcal{H}_r$ un conjunto de clases de reglas definidas sobre un dominio $\mathcal{X}$. Se considera $d = \text{máx}_i\, VCdim(\mathcal{H}_i)$. Se supone que $d \geq 3$. Entonces,

$$VCdim\left(\cup_{i=1}^{r} \mathcal{H}_i\right) \leq 4d\log(2d) + \log(r) \tag{2.20}$$

Demostración. 1. Podemos suponer sin pérdida de generalidad que $VCdim(\mathcal{H}_1) = VCdim(\mathcal{H}_2) = d$. Sea $\mathcal{H} = \mathcal{H}_1 \cup \mathcal{H}_2$. Tomamos k entero tal que $k \geq 2d+2$. Demostraremos que $\tau_{\mathcal{H}}(k) < 2^k$. Por el Lema de Sauer,

$$\begin{aligned}
\tau_{\mathcal{H}}(k) &< \tau_{\mathcal{H}_1}(k) + \tau_{\mathcal{H}_2}(k) \leq \\
&\leq \sum_{i=0}^{d} \binom{k}{i} + \sum_{i=0}^{d} \binom{k}{i} = \\
&= \sum_{i=0}^{d} \binom{k}{i} + \sum_{i=0}^{d} \binom{k}{k-i} = \\
&= \sum_{i=0}^{d} \binom{k}{i} + \sum_{i=k-d}^{k} \binom{k}{i} \leq \\
&\leq \sum_{i=0}^{d} \binom{k}{i} + \sum_{i=d+2}^{k} \binom{k}{i} \leq \\
&\leq \sum_{i=0}^{d} \binom{k}{i} + \sum_{i=d+1}^{k} \binom{k}{i} = \sum_{i=0}^{k} \binom{k}{i} = 2^k.
\end{aligned} \tag{2.21}$$

El hecho de que $\tau_{\mathcal{H}}(k) < 2^k$, supone que $VCdim(\mathcal{H}) < k$ y por tanto $VCdim(\mathcal{H}) \leq 2d+1$.

2. Asumimos inicialmente y sin pérdida de generalidad que para cada $i \in \{1, \ldots, r\}$, $VCdim(\mathcal{H}_i) \geq 3$. Sean $\mathcal{H} = \cup_{i=1}^{r} \mathcal{H}_i$ y $k \in \{1, \ldots, d\}$ tal que $\tau_{\mathcal{H}}(k) = 2^k$. Probaremos que $k \leq 4d \log(2d) + 2 \log r$

   De la definición de función de crecimiento se tiene,

$$\tau_{\mathcal{H}}(k) \leq \sum_{i=1}^{r} \tau_{\mathcal{H}_i}(k). \tag{2.22}$$

   Aplicando el Lema de Sauer a cada uno de los términos $\tau_{\mathcal{H}_i}$ se tiene

$$\tau_{\mathcal{H}}(k) < rk^d. \tag{2.23}$$

   Se sigue tomando logaritmos (en base 2) que $k < d \log(m) + \log(r)$. En virtud del Lema A.4, se deduce $k \leq 4d \log(2d) + 2 \log r$.

□

El siguiente teorema es crucial no solo en la demostración del teorema fundamental del aprendizaje sino que también nos guiará hacia una caracterización de la complejidad muestral. Este resultado me permite acotar la diferencia entre el riesgo empírico y el riesgo de manera uniforme. La técnica utilizada en la demostración se basa en desigualdades de simetrización y aleatorización.

Teorema 2.21. *Sea $\mathcal{H}$ una clase de reglas y $\tau_{\mathcal{H}}$ su función de crecimiento. Entonces, para toda distribución $\mathcal{D}$ y todo $\delta \in (0, 1)$, con probabilidad al menos $1 - \delta$ sobre la elección de la muestra S de tamaño m se tiene*

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4 + \sqrt{\log\left(\tau_{\mathcal{H}}(2m)\right)}}{\delta\sqrt{2m}} \quad \textit{para todo } h \in \mathcal{H}.$$

Demostración. Comenzaremos probando

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}} \left[\sup_{h \in \mathcal{H}} \mid L_{\mathcal{D}}(h) - L_S(h) | \right] \leq \frac{4 + \sqrt{\log\left(\tau_{\mathcal{H}}(2m)\right)}}{\sqrt{2m}}. \tag{2.24}$$

Para cada $h \in \mathcal{H}$, se puede reescribir el riesgo como $L_{\mathcal{D}}(h) = \mathop{\mathbb{E}}_{S' \sim \mathcal{D}}[L_{S'}(h)]$ donde $S' = z'_1, \ldots, z'_m$ es una muestra adicional i.i.d siguiendo $\mathcal{D}$.

Por tanto,

$$\begin{aligned}\mathop{\mathbb{E}}_{S\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}|L_{\mathcal{D}}(h)-L_S(h)|\right] &= \mathop{\mathbb{E}}_{S\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}\left|\mathop{\mathbb{E}}_{S'\sim\mathcal{D}}[L_{S'}(h)]-L_S(h)\right|\right] = \\ &= \mathop{\mathbb{E}}_{S\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}\left|\mathop{\mathbb{E}}_{S'\sim\mathcal{D}}[L_{S'}(h)-L_S(h)]\right|\right].\end{aligned} \tag{2.25}$$

De la generalización de la desigualdad triangular, se obtiene

$$\left|\mathop{\mathbb{E}}_{S'\sim\mathcal{D}}[L_{S'}(h)-L_S(h)]\right| \leq \mathop{\mathbb{E}}_{S'\sim\mathcal{D}}[\,|L_{S'}(h)-L_S(h)|\,]. \tag{2.26}$$

Dado que el supremo es mayor que la esperanza,

$$\sup_{h\in\mathcal{H}}\mathop{\mathbb{E}}_{S'\sim\mathcal{D}}[|L_{S'}(h)-L_S(h)|] \leq \mathop{\mathbb{E}}_{S'\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}|L_{S'}(h)-L_S(h)|\right]. \tag{2.27}$$

Combinando las ecuaciones (2.25), (2.26) y (2.27), se concluye

$$\begin{aligned}\mathop{\mathbb{E}}_{S\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}|L_{\mathcal{D}}(h)-L_S(h)|\right] &= \mathop{\mathbb{E}}_{S\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}\left|\mathop{\mathbb{E}}_{S'\sim\mathcal{D}}[L_{S'}(h)]-L_S(h)\right|\right] \leq \\ &\leq \mathop{\mathbb{E}}_{S\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}\mathop{\mathbb{E}}_{S'\sim\mathcal{D}}[\,|\,L_{S'}(h)-L_S(h)\,|\,]\right] \leq \\ &\leq \mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}|L_{S'}(h)-L_S(h)|\right]\end{aligned} \tag{2.28}$$

Sustituyendo los riesgos empíricos por sus definiciones en (2.28),

$$\mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}|L_{\mathcal{D}}(h)-L_S(h)|\right] \leq \mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}\frac{1}{m}\left|\sum_{i=1}^{m}(\ell(h,z_i')-\ell(h,z_i))\right|\right]. \tag{2.29}$$

Los $2m$ vectores de las muestras S y S' son aleatorios, independientes e idénticamente distribuidos. Por tanto, se pueden reemplazar los vectores z_i por los z_i'. La variable $\ell(h,z_i')$ y $\ell(h,z_i)$ tienen la misma distribución y son independientes. Luego la distribución de $\ell(h,z_i')-\ell(h,z_i)$ es simétrica, es decir, $\ell(h,z_i')-\ell(h,z_i)$ tiene la misma distribución que $\ell(h,z_i)-\ell(h,z_i')$. Se pueden introducir m variables aleatorias que denotaremos por $\sigma_i \in \{\pm 1\}$. Por tanto,

$$\mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}\frac{1}{m}\left|\sum_{i=1}^{m}(\ell(h,z_i')-\ell(h,z_i))\right|\right] = \mathop{\mathbb{E}}_{S,S'\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}}\frac{1}{m}\left|\sum_{i=1}^{m}\sigma_i(\ell(h,z_i')-\ell(h,z_i))\right|\right]. \tag{2.30}$$

La distribución que sigue σ_i se denotará por $U_\pm$ y por $\mathop{\mathbb{E}}_{\sigma_i \sim U_\pm}$ la esperanza de σ_i. La ecuación (2.30) es igual a

$$\mathop{\mathbb{E}}_{\sigma_i \sim U_\pm} \mathop{\mathbb{E}}_{S,S' \sim \mathcal{D}} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \left| \sum_{i=1}^{m} \sigma_i (\ell(h, z_i') - \ell(h, z_i)) \right| \right]. \tag{2.31}$$

Por la linealidad de la esperanza,

$$\mathop{\mathbb{E}}_{S,S' \sim \mathcal{D}} \mathop{\mathbb{E}}_{\sigma_i \sim U_\pm} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \left| \sum_{i=1}^{m} \sigma_i (\ell(h, z_i') - \ell(h, z_i)) \right| \right]. \tag{2.32}$$

Sea C el conjunto total de vectores que aparecen en S y S'. Como únicamente estamos trabajando con C, solo podemos tomar el supremos en $\mathcal{H}_C$. Por tanto,

$$\mathop{\mathbb{E}}_{\sigma_i \sim U_\pm} \left[\sup_{h \in \mathcal{H}} \frac{1}{m} \left| \sum_{i=1}^{m} \sigma_i (\ell(h, z_i') - \ell(h, z_i)) \right| \right] = \mathop{\mathbb{E}}_{\sigma_i \sim U_\pm} \left[\max_{h \in \mathcal{H}} \frac{1}{m} \left| \sum_{i=1}^{m} \sigma_i (\ell(h, z_i') - \ell(h, z_i)) \right| \right]. \tag{2.33}$$

Para cada $h \in \mathcal{H}_C$, sea $\theta_h = \frac{1}{m} \sum_{i=1}^{m} \sigma_i (\ell(h, z_i') - \ell(h, z_i))$, entonces $\mathbb{E}[\theta_h] = 0$ y cada θ_h es una media de variables independientes que toman valores en $[-1, 1]$. Por la desigualdad de Hoeffing (ver Lema A.1 en apéndice A), para cada $\rho > 0$,

$$\mathbb{P}\left[|\theta_h| > \rho \right] \leq 2 \exp\left(-2m\rho^2 \right). \tag{2.34}$$

Aplicando la desigualdad de Bonferroni,

$$\mathbb{P}\left[\max_{h \in \mathcal{H}_C} |\theta_h| > \rho \right] \leq 2 \left| \mathcal{H}_C \right| \exp\left(-2m\rho^2 \right). \tag{2.35}$$

En virtud del Lema A.3 del apéndice A,

$$\mathop{\mathbb{E}}_{\sigma_i \sim U_\pm} \left[\max_{h \in \mathcal{H}_C} |\theta_h| \right] \leq \frac{4 + \sqrt{\log\left(|\mathcal{H}_C| \right)}}{\sqrt{2m}}. \tag{2.36}$$

Finalmente, combinando todo y recurriendo a la definición de función de crecimiento

$$\mathop{\mathbb{E}}_{S \sim \mathcal{D}} \left[\sup_{h \in \mathcal{H}} |L_{\mathcal{D}}(h) - L_S(h)| \right] \leq \frac{4 + \sqrt{\log\left(\tau_{\mathcal{H}}(2m) \right)}}{\sqrt{2m}}. \tag{2.37}$$

Dado que $\sup_{h \in \mathcal{H}} |L_{\mathcal{D}}(h) - L_S(h)|$ es una variable aleatoria no negativa, tomando

$\delta \in (0,1)$, en virtud de la desigualdad de Markov,

$$\mathbb{P}\left[\sup_{h\in\mathcal{H}} |L_{\mathcal{D}}(h) - L_S(h)| > \frac{4+\sqrt{\log\left(\tau_{\mathcal{H}}(2m)\right)}}{\delta\sqrt{2m}}\right] \leq \frac{\mathop{\mathbb{E}}_{S\sim\mathcal{D}}\left[\sup_{h\in\mathcal{H}} |L_{\mathcal{D}}(h) - L_S(h)|\right]}{\frac{4+\sqrt{\log(\tau_{\mathcal{H}}(2m))}}{\delta\sqrt{2m}}}. \quad (2.38)$$

Gracias a la acotación obtenida en (2.37),

$$\mathbb{P}\left[\sup_{h\in\mathcal{H}} |L_{\mathcal{D}}(h) - L_S(h)| > \frac{4+\sqrt{\log\left(\tau_{\mathcal{H}}(2m)\right)}}{\delta\sqrt{2m}}\right] \leq \delta. \quad (2.39)$$

Entonces,

$$\mathbb{P}\left[\sup_{h\in\mathcal{H}} |L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4+\sqrt{\log\left(\tau_{\mathcal{H}}(2m)\right)}}{\delta\sqrt{2m}}\right] \geq 1-\delta. \quad (2.40)$$

Finalmente, se puede concluir que con probabilidad al menos $1-\delta$,

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4+\sqrt{\log\left(\tau_{\mathcal{H}}(2m)\right)}}{\delta\sqrt{2m}} \text{ para todo } \ h \in \mathcal{H}. \quad (2.41)$$

□

Estamos ya en condiciones de poder probar el teorema fundamental del aprendizaje estadístico.

Teorema 2.22. *(Teorema fundamental del aprendizaje estadístico.) Sea $\mathcal{H}$ una clase de reglas con funciones de $\mathcal{X}$ a $\{0,1\}$ y sea ℓ_{0-1} la función de pérdida. Entonces, los siguientes enunciados son equivalentes:*

1. *$\mathcal{H}$ tiene la propiedad de convergencia uniforme.*
2. *$\mathcal{H}$ es aprendible PAC.*
3. *$\mathcal{H}$ tiene dimensión-VC finita.*

Demostración. $1 \Rightarrow 2$ es la proposición 2.4.
$2 \Rightarrow 3$ es el Teorema 2.14
Veamos $3 \Rightarrow 1$.
Sea $\mathcal{H}$ una clase de reglas con $VCdim(\mathcal{H}) \leq d < \infty$. Fijamos $\epsilon > 0$ y $\delta \in (0,1)$. Por el

Teorema 2.21, se tiene

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4 + \sqrt{\log\left(\tau_{\mathcal{H}}(2m)\right)}}{\delta\sqrt{2m}} \quad \text{para todo } h \in \mathcal{H}. \tag{2.42}$$

En virtud del Lema de Sauer 2.18, para $m > d+1$, $\tau_{\mathcal{H}}(2m) \leq (2em/d)^d$. Empleando este hecho para acotar la desigualdad anterior

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{4 + \sqrt{d\log\left(2em/d\right)}}{\delta\sqrt{2m}}. \tag{2.43}$$

Asumiendo $4 \leq \sqrt{d\log(2em/d)}$, se obtiene

$$|L_{\mathcal{D}}(h) - L_S(h)| \leq \frac{1}{\delta}\sqrt{\frac{2d\log(2em/d)}{m}}. \tag{2.44}$$

Para conseguir que (2.44) sea menor que ϵ es necesario (llevando a cabo diversas manipulaciones algebraicas) que

$$m \geq \frac{2d\log(m)}{(\delta\epsilon)^2} + \frac{2d\log(2e/d)}{(\delta\epsilon)^2}. \tag{2.45}$$

Basta con tomar en virtud del Lema A.4,

$$m \geq 4\frac{2d}{(\delta\epsilon)^2}\log\left(\frac{4d}{(\delta\epsilon)^2}\right) + \frac{4d\log(2e/d)}{(\delta\epsilon)^2}. \tag{2.46}$$

De esta manera, $\mathcal{H}$ tiene la propiedad de la convergencia uniforme.

□

Teorema 2.23. *Sea $\mathcal{H}$ una clase de reglas con dimensión-VC finita tal que $VCdim(\mathcal{H}) = d < \infty$. Entonces $\mathcal{H}$ es aprendible PAC con complejidad muestral*

$$m_{\mathcal{H}}(\epsilon, \delta) \leq 4\frac{2d}{(\delta\epsilon)^2}\log\left(\frac{4d}{(\delta\epsilon)^2}\right) + \frac{4d\log(2e/d)}{(\delta\epsilon)^2}$$

Toda la teoría desarrollada en este capítulo será la base para los posteriores resultados que enunciaremos y demostraremos en el siguiente capítulo para el caso concreto de las redes neuronales.

Capítulo 3

Redes neuronales artificiales.

Los elementos fundamentales y más básicos para tratar el conocimiento en el cerebro son las neuronas. Su función principal es recibir, procesar y transmitir información a través de señales químicas y eléctricas. Recordemos el esquema básico de una neurona en la figura 3.1.

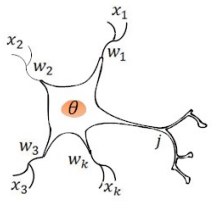

Figura 3.1: Neurona biológica.

Dichas entidades se agrupan en estructuras más complejas dando lugar a las llamadas redes neuronales. Las redes neuronales reciben estímulos que se transmiten de neurona en neurona hasta obtener una respuesta final. Esta respuesta ha sido resultado de la colaboración de todas las neuronas de la red.

En este concepto tan humano se basan las *redes neuronales artificiales*, una de las técnicas más potentes desarrolladas por el aprendizaje supervisado. Las redes neuronales artificiales son modelos de computación usados en análisis de datos. Se parecen al cerebro en dos aspectos: por una parte, la red neuronal es capaz de aprender y mejorar gracias a un proceso de aprendizaje, y por otra, las conexiones entre neuronas, llamadas cargas sinápticas, son la solidez de la transmisión del conocimiento.

Una red neuronal está formada básicamente por nodos (*neuronas*) y los enlaces entre los nodos. Cada nodo (figura 3.2) es una unidad de cálculo que puede recibir o bien datos de otras neuronas o bien información externa. Por su parte, los enlaces entre neuronas no son todos iguales. Serán los llamados *pesos* los encargados de reflejar como son las conexiones entre neuronas y como de fuertes son. Cada neurona recibirá una suma ponderada de las salidas de las neuronas que estén conectadas a ella.

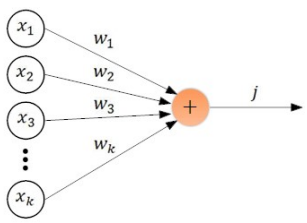

Figura 3.2: Neurona artificial.

El proceso de aprendizaje de una red se basa en la modificación de los pesos con el objetivo de que la salida final lleve a cabo la mejor clasificación posible. Como sucede con cualquier técnica de aprendizaje estadístico, puede ocurrir que durante el entrenamiento de una red esta se ajuste en exceso a las particularidades del conjunto de entrenamiento, realice una predicción pobre ante nuevas observaciones y pierda su capacidad de generalización, es decir, caigamos en el sobreajuste, fenómeno que intentaremos evitar recurriendo al *algoritmo de retropropagación.*

Formalizaremos estas ideas en la siguiente sección.

3.1. Esquema de una red neuronal.

El primer y más básico modelo de red neuronal que se diseñó fue el *perceptrón simple* y supuso el fundamento de lo que posteriormente serían las redes neuronales tal y como las conocemos hoy en día. Fue presentado por F.Rosenblatt en 1958 (ver [6]). El perceptrón constituye el diseño más sencillo de red neuronal usado para la clasificación en grupos linealmente separables. Sin embargo, a la hora de resolver problemas de clasificación más complejos, no es útil. Por este motivo, años más tarde se decide extender y generalizar el concepto de perceptrón dando lugar a las redes neuronales.

Es habitual identificar una red neuronal con un grafo (V, E) cuyos vértices corresponden con el conjunto de neuronas de la red y cuyas aristas son los enlaces entre ellas. En esta memoria, nos centraremos en las redes neuronales prealimentadas o simplemente redes neuronales conocidas como *feedforward neural networks*. Estas están constituidas por neuronas conectadas que no forman ciclos y cada conexión tiene asignado un único sentido (no son bidireccionales). Por consiguiente, los grafos que describen dichas redes son acíclicos no dirigidos.

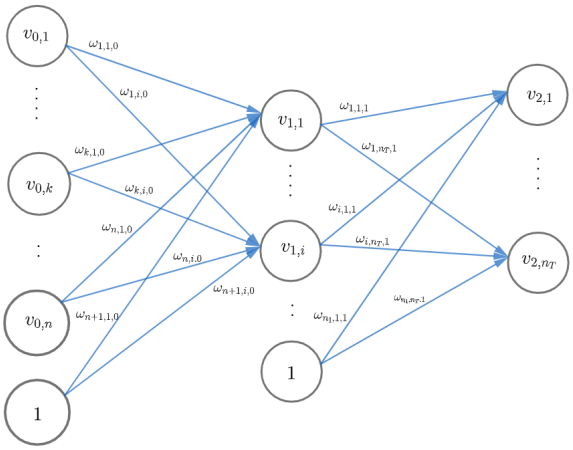

Figura 3.3: Esquema de una red neuronal.

El esquema básico de una red neuronal se muestra en la figura 3.3. La notación usada en la imagen es introducida a continuación.

Las neuronas de una red se agrupan a su vez en estructuras más pequeñas denominadas *capas*. Formalmente, suponiendo que nuestra red cuenta con $T+1$ capas, entenderemos $V = \cup_{t=0}^{T} V_t$ donde cada V_t es una capa de la red y cada arista en E conecta algún nodo de V_{t-1} con algún nodo de la capa siguiente V_t. Nos referiremos a la primera y última ,V_0 y V_T, como la *capa de entrada* y *capa de salida* respectivamente, y las intermedias, $V_1, \ldots, V_{T-1}$, suelen denominarse *capas ocultas*. V_0 se caracteriza por ser la única que recibe datos del exterior. En la figura 3.3 la red neuronal cuenta con una única capa oculta. Aquellas redes con dos o más capas intermedias se llaman *redes neuronales profundas*. El uso de capas adicionales conlleva una mayor complejidad.

Los pesos de las aristas se representan con una función $w : E \to \mathbb{R}$. Cada nodo es un elemento de cálculo, es decir, recibe ciertos datos de entrada, los modifica matemáticamente y posteriormente aplicándoles una *función de activación* $\sigma : \mathbb{R} \to \mathbb{R}$ proporciona la salida. Algunas de las funciones de activación más usadas son la función signo $\sigma(x) = sign(x)$, la función umbral $\sigma(x) = \mathbb{I}_{[x>0]}$, la sigmoide, $\sigma(x) = \frac{1}{1+exp(-x)}$ o la función lineal $\sigma(x) = x$ (la identidad). Otra de las más eficientes y populares en la actualidad es la función RELU (Rectificador Lineal Unitario) dada por

$$\sigma(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ x & \text{si} \quad x \geq 0 \end{cases} \tag{3.1}$$

Denotaremos por $v_{t,i}$ a la i-ésima neurona de la capa t-ésima. Suponiendo que la red tiene como vector de entrada $\mathbf{x}$, nos referiremos a la entrada de la neurona $v_{t,i}$ como $a_{t,i}(\mathbf{x})$ y a la salida de $v_{t,i}$ como $o_{t,i}(\mathbf{x})$. Además, todas las capas, excluyendo la capa de salida, cuentan con una última neurona que tiene como salida la constante 1, es decir, suponiendo que cada capa V_t cuenta con n_t neuronas, $o_{t,n_t}(\mathbf{x}) = 1$ para todo $1 \leq t \leq T-1$. Cabe destacar que las neuronas de la capa V_0 no realizan ningún cálculo sobre los datos de entrada así pues si $\mathbf{x} = (x_1, \ldots, x_n)$, entonces $a_{0,i}(\mathbf{x}) = x_i = o_{0,i}(\mathbf{x}),\ 1 \leq i \leq n$ y $a_{0,n+1}(\mathbf{x}) = o_{0,n+1}(\mathbf{x}) = 1$.

La entrada de cada neurona se obtiene como una suma ponderada de las salidas de todas las neuronas conectadas a ellas. Esta ponderación viene fijada por los pesos.

Sea $\omega_{i,j,t}$ el valor del peso que une la neurona i-ésima de la capa V_t con la neurona j-ésima de la capa V_{t+1}. Por tanto, la entrada de la neurona $v_{t+1,j}$ es

$$a_{t+1,j}(\mathbf{x}) = \sum_{r:(v_{t,r},v_{t+1,j})\in E} \omega_{r,j,t} \cdot o_{t,r}(\mathbf{x}) + \omega_{n_t,j,t}.$$

La salida final de la neurona $v_{t+1,j}$ no es más que la aplicación de la función de activación

escogida σ a la suma ponderada.

$$o_{t+1,j}(\mathbf{x}) = \sigma(a_{t+1,j}(\mathbf{x})). \tag{3.2}$$

Denotamos por T al número total de capas que tiene la red excluyendo V_0. T determina la *profundidad* de la red. El *tamaño* viene dado por $|V|$, es decir, el número de neuronas. La *amplitud* se define como $\max_t |V_t|$.

3.2. Aprendizaje de redes neuronales.

Nuestro objetivo es determinar cuando las redes neuronales son aprendibles PAC. Para ello, inicialmente debemos definir su clase de reglas y mostrar que sucede con su dimensión-VC.

Una red neuronal identificada con la 4-úpla (V, E, σ, w) puede ser entendida como una función $h_{V,E,\sigma,w} : \mathbb{R}^{|V_0|-1} \to \mathbb{R}^{|V_T|}$. Por tanto, podemos considerar la clase de reglas de redes neuronales de la siguiente manera:

$$\mathcal{H}_{V,E,\sigma} = \{h_{V,E,\sigma,w} : w \text{ es una función de } E \text{ en } \mathbb{R}\}$$

En esta clase, tanto el grafo (V, E) como la función de activación σ son elementos fijos. De hecho, la terna (V, E, σ) se suele llamar *arquitectura de la red*. La componente que varia es la función que proporciona los pesos $w : E \to \mathbb{R}$. Gracias a esta flexibilidad en los pesos, la red neuronal es capaz de aprender y entrenarse. Sin embargo, la arquitectura de la red también es un factor influyente en el aprendizaje del modelo.

Comencemos estudiando que tipo de funciones pueden ser expresadas empleando la clase de reglas $\mathcal{H}_{V,E,\sigma}$. El siguiente resultado nos mostrará la amplia capacidad de implementación que tienen las redes neuronales. Fijaremos la arquitectura V, E y σ, siendo σ la función signo.

Proposición 3.1. *Para cada $n \in \mathbb{N}$ existe un grafo (V, E) de profundidad 2 tal que $\mathcal{H}_{V,E,sign}$ contiene todas las posibles funciones de $\{\pm 1\}^n$ a $\{\pm 1\}$.*

Demostración. Deseamos construir una red neuronal con una única capa intermedia. Tomamos $|V_0| = n + 1$, $|V_1| = 2^n + 1$ y $V_2 = 1$. Sea E el conjunto de todos los posibles enlaces entre las neuronas de cada capa y $f : \{\pm 1\}^n \to \{\pm 1\}$ una función Booleana cualquiera. Veamos que podemos ajustar los pesos de tal manera que la red implemente correctamente la función f.

Sea $\{u_1, \ldots, u_k\}$ el conjunto de vectores de $\{\pm 1\}^n$ para los cuales la imagen por f es 1, es decir, $f(u_i) = 1$ para todo $i \in \{1, \ldots, k\}$. Sea $\mathbf{x} \in \{\pm 1\}^n$ y fijemos $u_i \in \{u_1, \ldots, u_k\}$.

Se nos presentan dos casos:

- Si $\mathbf{x} \neq u_i$, entonces siendo $\mathbf{x} = (x_1, \ldots, x_n)$ y $u_i = (u_{i_1}, \ldots, u_{i_n})$ al menos para un $j \in \{1, \ldots, n\}$ se tiene $x_j \neq u_{i_j}$ luego

$$< \mathbf{x}, u_i >= x_1 u_{i_1} + x_2 u_{i_2} + \ldots + x_n u_{i_n} \leq (n-1) - 1 = n - 2.$$

- Si $\mathbf{x} = u_i$ entonces su producto escalar será exactamente n, es decir, $< \mathbf{x}, u_i >= n$

Para cada $i \in \{1, \ldots, k\}$, se define la función g_i

$$g_i(\mathbf{x}) = sign(< \mathbf{x}, u_i > -n + 1) = \begin{cases} 1 & \text{si } \mathbf{x} = u_i \\ -1 & \text{si } \mathbf{x} \neq u_i \end{cases}$$

Luego, la función g_i asigna valores igual que la función f.

Se tiene que el conjunto de definición de f cuenta con 2^n vectores, por tanto, $k \leq 2^n$. De las $2^n + 1$ neuronas que tiene la capa intermedia, podemos adaptar los pesos para que las k primeras implementen cada una de las funciones g_i. Por tanto, podemos reescribir la función $f(\mathbf{x})$ como

$$f(\mathrm{x}) = \operatorname{sign}\left(\sum_{i=1}^{k} g_i(\mathrm{x}) + k - 1\right).$$

Veamos esta igualdad.

- Para $\mathbf{x} \notin \{u_1, \ldots, u_k\}$, se tiene:

$$sign\left(\sum_{i=1}^{k} g_i(\mathbf{x}) + k - 1\right) = \operatorname{sign}\left(\sum_{i=1}^{k} sign(< \mathbf{x}, u_i > -n + 1) + k - 1\right) =$$
$$= sign(-k + k - 1) = -1 = f(\mathbf{x})$$

- Para $\mathbf{x} \in \{u_1, \ldots, u_k\}$, se tiene:

$$sign\left(\sum_{i=1}^{k} g_i(\mathbf{x}) + k - 1\right) = sign(1 - (k-1) + k - 1) = 1 = f(\mathbf{x})$$

□

La proposición anterior pone de manifiesto cómo las redes neuronales con función de activación la función signo y con una única capa oculta pueden implementar cualquier función Booleana. Sin embargo, se nos presenta un inconveniente importante: el tamaño de la red es exponencial en n. Esto supone un obstáculo pues a medida que n aumente,

la red neuronal será excesivamente grande y costosa de manipular.

3.2.1. Complejidad muestral de las redes neuronales.

Recordemos que el *Teorema fundamental del aprendizaje estadístico* 2.22 nos caracterizaba la complejidad muestral de una clase de reglas en función de su dimensión-VC. No solo eso, sino que aseguraba que el hecho de que una clase de reglas tuviera dimensión-VC finita, hacia de dicha clase aprendible PAC. Por tanto, tiene sentido que deseemos calcular o aproximar la dimensión-VC de la clase de reglas $\mathcal{H}_{V,E,\sigma}$. De esta manera, sabremos cuando las redes neuronales llevan a cabo una buena aproximación.

Restringiremos las redes neuronales de $\mathcal{H}_{V,E,\sigma}$ a aquellas que cuentan con un única neurona en la capa de salida, es decir, nos centraremos en problemas de clasificación binaria. Razonaremos para $\sigma(x) = sign(x)$. El siguiente teorema es fundamental para garantizar el aprendizaje de las redes neuronales.

Teorema 3.2. *La dimensión-VC de $\mathcal{H}_{V,E,sign}$ es $O(|E|\log(|E|))$.*

Demostración. Denotaremos por $\mathcal{H} = \mathcal{H}_{V,E,sign}$ para simplificar la notación.

Tomamos una red neuronal con $V_0, \ldots, V_T$ capas. Fijamos $t \in \{1, \ldots, T\}$. Asignando diferentes pesos a los enlaces entre V_{t-1} y V_t, obtenemos diferentes funciones de $\mathbb{R}^{|V_{t-1}|}$ en $\{\pm 1\}^{|V_t|}$.

Denotamos por $\mathcal{H}^{(t)}$ la clase de reglas de todas las posibles funciones de $\mathbb{R}^{|V_{t-1}|}$ en $\{\pm 1\}^{|V_t|}$, es decir, $\mathcal{H}^{(t)}$ contiene todas las posibles combinaciones de pesos entre la capa V_{t-1} y V_t. Obviamente, la clase $\mathcal{H}$ se puede expresar como composición de estas funciones: $\mathcal{H} = \mathcal{H}^{(T)} \circ \cdots \circ \mathcal{H}^{(1)}$.

En virtud del Lema 2.16

$$\tau_{\mathcal{H}}(m) \leq \prod_{t=1}^{T} \tau_{\mathcal{H}^{(t)}}(m). \tag{3.3}$$

Además, cada una de las clases $\mathcal{H}^{(t)}$ es posible escribirla a su vez como producto de clases de reglas, $\mathcal{H}^{(t)} = \mathcal{H}^{(t,1)} \times \cdots \times \mathcal{H}^{(t,V_t)}$.

Por otra parte, cada $\mathcal{H}^{(t,j)}$ es el conjunto de posibles funciones de $\mathbb{R}^{|V_{t-1}|}$ en $\{\pm 1\}$, es decir, posee todas las posibles combinaciones de pesos que unen las neuronas de V_{t-1} con la neurona j-ésima de V_t.

Por el Lema 2.17, se tiene

$$\tau_{\mathcal{H}^{(t)}}(m) \leq \prod_{i=1}^{|V_t|} \tau_{\mathcal{H}^{(t,i)}}(m). \tag{3.4}$$

Cada neurona representa un semiplano homogéneo y la dimensión-VC de un semiplano homogéneo coincide con la dimensión de la entrada.

Sea $d_{t,i}$ el número de enlaces conectados a la neurona i-ésima de la capa t-ésima. Luego la dimensión-VC del semiplano homogéneo que representa la neurona i-ésima de la capa t-ésima es $d_{t,i}$. Por tanto, gracias al Lema de Sauer se tiene

$$\tau_{\mathcal{H}^{(t,i)}}(m) \leq \left(\frac{em}{d_{t,i}}\right)^{d_{t,i}} \leq (em)^{d_{t,i}}. \tag{3.5}$$

La acotación resultante en 3.3 es

$$\tau_{\mathcal{H}}(m) \leq (em)^{\sum_{t,i} d_{t,i}} = (em)^{|E|}.$$

Si $\mathcal{H}$ rompe un conjunto de m puntos, necesariamente $\tau_{\mathcal{H}}(m) = 2^m$. Por tanto,

$$2^m \leq (em)^{|E|} \Rightarrow m \leq \frac{|E|\log(em)}{\log(2)}. \tag{3.6}$$

Empleando el Lema A.4 es posible concluir. □

El teorema anterior nos muestra que la clase de redes neuronales con función de activación la función signo tienen dimensión-VC finita. En virtud del Teorema fundamental del aprendizaje estadístico 2.22 ya podemos asegurar que dicha clase de funciones es aprendible PAC. No solo eso, sino que se ha obtenido una cota para la complejidad muestral, es decir, si se trabaja con una red neuronal con un número de nodos ($|V|$) y aristas ($|E|$) lo suficientemente grande, tomando una muestra de entrenamiento de tamaño

$$m \leq \frac{|E|\log(em)}{\log(2)} \tag{3.7}$$

entonces $\mathcal{H}_{V,E,sign}$ será una clase de reglas aprendible PAC, capaz de resolver el problema de clasificación minimizando de la mejor manera posible el error cometido.

Sin embargo, el hecho de que la complejidad muestral esté acotada por una función creciente en $|E|$, implica que a mayor número de aristas es necesario un mayor tamaño de la muestra. Esto puede suponer un inconveniente importante.

Podemos combinar el resultado anterior con la proposición 3.1 y obtenemos la si-

guiente caracterización del número de neuronas que ha de tener una red neuronal para implementar una función Booleana $f : \{0,1\}^n \to \{0,1\}$.

Teorema 3.3. *Para cada $n \in \mathbb{N}$, sea $s(n)$ el menor entero tal que existe un grafo (V,E) con $|V| = s(n)$ tal que la clase de reglas $\mathcal{H}_{V,E,sign}$ contiene todas las funciones de $\{0,1\}^n$ a $\{0,1\}$. Entonces $s(n)$ es exponencial en n.*

Demostración. Supongamos que para algún grafo (V,E) se tiene que $\mathcal{H}_{V,E,sign}$ contiene todas las funciones de $\{0,1\}^n$ a $\{0,1\}$. Por tanto, $\{0,1\}^n$ es separable por $\mathcal{H}_{V,E,sign}$. Esto implica inmediatamente que $VCdim(\mathcal{H}_{V,E,sign}) = 2^n$.

Por otro lado, la dimensión-VC de $\mathcal{H}_{V,E,sign}$ está acotada por $O(|E|\log(|E|)$, como se ha visto en el Teorema 3.2. Además, $2^n \leq O(|E|\log(|E|)) \leq O(|V|^3)$. Luego, para $M > 0$ se tiene $2^n \leq M|V|^3$. Se concluye $|V| \geq K2^{n/3}$ para una constante $K > 0$. Por tanto, $|V|$ será exponencial en n pues está acotada inferiormente asintóticamente por $2^{n/3}$. □

Existen resultados similares y de orden parecido al anterior cuando la función de activación es la sigmoide.

3.3. Entrenamiento de redes neuronales.

Una de las grandes ventajas que presentan las redes neuronales es la flexibilidad que poseen sus pesos. El aprendizaje de una red neuronal es equivalente a la modificación del valor de los pesos con el objetivo de minimizar el error cometido. Por tanto, escoger adecuadamente dichos valores supondrá el éxito del entrenamiento.

Con este mismo propósito, fueron David Rumelhart, Geoffrey Hinton y Ronald Williams quienes en 1986 presentaron el *algoritmo de backpropagation o retropropagación* como un método de aprendizaje eficiente para las redes neuronales. Se basaron en la técnica de *descenso de gradiente*, presentada anteriormente, uno de los métodos más utilizados para la optimización de funciones en aprendizaje estadístico. Es especialmente característico de este algoritmo que la modificación de los pesos comienza con los pesos conectados a la capa de salida, posteriormente se modifican los pesos de la penúltima capa y así sucesivamente. Luego, su corrección se lleva a cabo en el orden inverso a su actuación en la red.

3.3.1. Algoritmo de Retropropagación.

Detallaremos seguidamente el algoritmo de backpropagation para una red neuronal con $n+1$ neuronas de entradas, k neuronas de salida y una única capa intermedia con

$r+1$ neuronas.

Siguiendo con la notación anterior, para todo $1 \leq i \leq n+1$ y para todo $1 \leq j \leq r$, será $\omega_{i,j,0}$ el peso del enlace entre la neurona $v_{0,i}$ y $v_{1,j}$. Por tanto, la entrada de cada una de las neuronas de la capa V_1 cuando la red es alimentada con el vector $\mathbf{x} = (x_1, \ldots, x_n)$ es

$$a_{1,j}(\mathbf{x}) = \sum_{i=1}^{n} \omega_{i,j,0} x_i + \omega_{n+1,j,0} \text{ para } 1 \leq j \leq r.$$

La salida de cada una de las neuronas de V_1 es simplemente la aplicación de la función de activación σ, es decir, $o_{1,j}(\mathbf{x}) = \sigma(a_{1,j}(\mathbf{x}))$, $1 \leq j \leq r$ y $o_{1,r+1}(\mathbf{x}) = 1$.

Análogamente, para todo $1 \leq i \leq r+1$ y para todo $1 \leq j \leq k$, será $\omega_{i,j,1}$ el peso del enlace entre la neurona $v_{1,i}$ y $v_{2,j}$. De nuevo, la entrada de cada una de las neuronas de la capa V_2 es

$$a_{2,j}(\mathbf{x}) = \sum_{i=1}^{r} \omega_{i,j,1} o_{1,i}(\mathbf{x}) + \omega_{n+1,j,1} \text{ para } 1 \leq j \leq k$$

La salida de cada una de las neuronas de la capa de salida es simplemente $o_{2,j}(\mathbf{x}) = \sigma(a_{2,j}(\mathbf{x}))$ para $1 \leq j \leq k$.

Una manera extendida de medir el rendimiento es a partir del error cuadrático medio de las salidas actuales de la red respecto de las deseadas, es decir, deseamos minimizar la función

$$E(\mathbf{x}, \mathbf{y}, \mathbf{W}) = \frac{1}{2} \sum_{j=1}^{k} (y_j - o_{2,j}(\mathbf{x}))^2. \tag{3.8}$$

donde $\mathbf{W}$ es el vector de pesos e $\mathbf{y} = (y_1, \ldots, y_k)$ es el verdadero valor que ha de asignar la red al vector $\mathbf{x}$. Dado que el objetivo del algoritmo de backpropagation es obtener los valores de los pesos que minimicen el error (ecuación (3.18)) entenderemos el error como una función dependiente del vector de pesos.

Debemos fijar inicialmente una amplitud de paso η o también denominada *tasa de aprendizaje*. Este parámetro determina la velocidad a la que van a ir modificándose los valores de las conexiones entre las neuronas. Generalmente η se encuentra en el intervalo [0,1], siendo los valores más cercanos a cero los que hacen que los pesos cambien poco a poco, acercándose lentamente a la convergencia del proceso, y los cercanos a uno los que hacen que la red converja más rápidamente al principio, pero siendo posible que los pesos oscilen demasiado al encontrar el mínimo.

La regla para la modificación de los pesos de la última capa en la iteración m-ésima es

$$\omega_{i,j,1}^{[m]} = \omega_{i,j,1}^{[m-1]} - \eta \frac{\partial E}{\partial \omega_{i,j,1}^{[m-1]}}, \quad 1 \leq i \leq r+1, \ 1 \leq j \leq k. \tag{3.9}$$

En virtud de la regla de la cadena, la derivada parcial del error con respecto a los pesos viene dada por

$$\frac{\partial E}{\partial \omega_{i,j,1}^{[m-1]}} = \frac{\partial E}{\partial o_{2,j}} \cdot \frac{\partial o_{2,j}}{\partial a_{2,j}} \cdot \frac{\partial a_{2,j}}{\partial \omega_{i,j,1}^{[m-1]}} = \delta_j \cdot o_{1,i}$$

donde

$$\begin{aligned}\delta_j &= \frac{\partial E}{\partial o_{2,j}} \cdot \frac{\partial o_{2,j}}{\partial a_{2,j}} = \frac{\partial E}{\partial o_{2,j}} \cdot \sigma'(a_{2,j}) \cdot = \\ &= -\sum_{i=1}^{k} (y_i - o_{2,j}(\mathbf{x})) \sigma' \left(\sum_{i=1}^{r} \omega_{i,j,1}^{[m-1]} o_{1,i} + \omega_{n+1,j,1}^{[m-1]} \right) = -(\mathbf{y} - \mathbf{o}_2) \sigma' \left(\sum_{i=1}^{r} \omega_{i,j,1}^{[m-1]} o_{1,i} + \omega_{n+1,j,1}^{[m-1]} \right).\end{aligned}$$

siendo $\mathbf{o}_2 = (o_{2,1}(\mathbf{x}), \ldots, o_{2,k}(\mathbf{x}))$

Por tanto, sustituyendo en la ecuación (3.9) se obtiene

$$\begin{aligned}\omega_{i,j,1}^{[m]} &= \omega_{i,j,1}^{[m-1]} + \eta \cdot \delta_j \cdot \sigma(a_{1,i}) = \\ &= \omega_{i,j,1}^{[m-1]} + \eta (\mathbf{y} - \mathbf{o}_2) \sigma' \left(\sum_{i=1}^{r} \omega_{i,j,1}^{[m-1]} x_i + \omega_{n+1,j,1}^{[m-1]} \right) \sigma(a_{1,i}).\end{aligned} \tag{3.10}$$

Con la expresión anterior han sido calculados los pesos de las neuronas de la última capa. Podemos realizar un proceso similar para modificar los de unión de la capa de entrada con la capa intermedia. Análogamente,

$$\omega_{i,j,0}^{[m]} = \omega_{i,j,0}^{[m-1]} - \eta \frac{\partial E}{\partial \omega_{i,j,0}^{[m-1]}}, \quad 1 \leq i \leq n+1, \ 1 \leq j \leq r. \tag{3.11}$$

De nuevo, desarrollando la derivada parcial del error con respecto de los pesos, gracias a la regla de la cadena

$$\frac{\partial E}{\partial \omega_{i,j,0}^{[m-1]}} = \delta_j \cdot \frac{\partial a_{2,j}}{\partial \omega_{i,j,0}^{[m-1]}} = \delta_j \cdot \beta_{i,j}. \tag{3.12}$$

donde

$$\beta_{i,j} = \frac{\partial a_{2,j}}{\partial \omega_{i,j,0}^{[m-1]}} = \frac{\partial a_{2,j}}{\partial o_{1,i}} \cdot \frac{\partial o_{1,i}}{\partial \omega_{i,j,0}^{[m-1]}} = \omega_{i,j,1}^{[m-1]} \cdot \sigma'(a_{1,j}) \cdot x_i =$$
$$= \omega_{i,j,1}^{[m-1]} \cdot \sigma' \left(\sum_{i=1}^{n} \omega_{i,j,0}^{[m-1]} x_i + \omega_{n+1,j,0}^{[m-1]} \right) \cdot x_i.$$

Finalmente, sustituyendo en (3.11) se concluye que los pesos de la primera capa vienen determinados por

$$\omega_{i,j,0}^{[m]} = \omega_{i,j,0}^{[m-1]} + \eta \cdot \delta_j \cdot \beta_{i,j} =$$
$$= \omega_{i,j,0}^{[m-1]} + \eta \cdot (\mathbf{y} - \mathbf{o}_2) \omega_{i,j,1}^{[m-1]} \sigma'(a_{1,j}) x_i \sigma' \left(\sum_{i=1}^{r} \omega_{i,j,1}^{[m-1]} x_i + \omega_{n+1,j,1}^{[m-1]} \right). \tag{3.13}$$

Observando detenidamente las ecuaciones (3.9) y (3.11) de actualización de los pesos, queda patente como este algoritmo se basa en el método de descenso de gradiente para alcanzar el mínimo en el error. Finalizará o bien cuando se hayan llevado a cabo un número determinado de iteraciones o bien cuando la red alcance un rendimiento deseado (error máximo permitido).

Este proceso, que hemos desarrollado más detalladamente para redes neuronales con una única capa oculta, puede ser generalizado para redes con un número arbitrario de capas. Supongamos que contamos con $V_0, \ldots, V_T$ capas y cada capa V_t posee k_t neuronas.

En el algoritmo descrito a continuación el valor η es fijo. Sin embargo, para garantizar la convergencia del proceso conviene tomar una amplitud distinta en cada iteración, es decir, fijar una secuencia de amplitudes $\eta_1, \ldots, \eta_\tau$.

Algoritmo de backpropagation para redes neuronales.

- **Input**: Red neuronal con grafo asociado (V, E), $V_0, \ldots, V_T$ capas y $|V_t| = k_t$.
  $(\mathbf{x}, \mathbf{y})$ perteneciente a la muestra de entrenamiento S.
  Función de activación $\sigma : \mathbb{R} \to \mathbb{R}$.
  Número de iteraciones τ.
1. Inicializamos el vector de pesos $\mathbf{W}^{(0)} \in \mathbb{R}^{|E|}$ y calculamos $E(\mathbf{x}, \mathbf{y}, \mathbf{W}^{(0)})$.
2. Para cada $l = 1, \ldots, \tau$,
   Para cada $s = T-1, \ldots, 1$,
   Para cada $i = 1, \ldots, k_s$,
   Para cada $j = 1, \ldots, k_{s+1}$,
   $\delta_{ij}^s = \frac{\partial E}{\partial \omega_{i,j,s}^{[l-1]}}$
   $\omega_{i,j,s}^{[l]} = \omega_{i,j,s}^{[l-1]} - \eta \cdot \delta_{ij}^s$
- **Output**: El algoritmo devuelve el vector de pesos $\mathbf{W}^{(\tau)}$.

Cabe destacar que escoger inicialmente los pesos todos nulos supone que los enlaces de las capas ocultas sean iguales. Dado que deseamos ejecutar el algoritmo un número determinado de veces cada una de ellas con un vector $\mathbf{W}^{(0)}$ diferente con el objetivo de alcanzar un buen mínimo, las entradas de $\mathbf{W}^{(0)}$ han de ser escogidas de manera aleatoria y cercanas a cero, pero nunca todas nulas.

Para darle solución al problema del sobreajuste durante el entrenamiento de redes neuronales es habitual emplear el *método de parada anticipada*. Explicaremos brevemente en que consiste esta técnica.

En primer lugar, se ha de dividir el conjunto de datos disponibles en dos: entrenamiento y validación. Esto ya se ha explicado que es general para cualquier técnica de aprendizaje estadístico. El primer conjunto lo emplearemos para calcular el gradiente y actualizar los pesos de la red. El segundo nos indicará cómo es el error que estamos cometiendo a medida que avanzamos en el entrenamiento. Cuando la red comience a ajustarse en exceso a los datos, el error evaluado con el conjunto de validación comienza a incrementarse. Si este sobrepasa un determinado valor previamente fijado, el entrenamiento se interrumpe y se retorna a los pesos con los cuales se obtuvo el mínimo error. La regla de parada anticipada proporcionan orientación sobre cuántas iteraciones se han de realizar antes de que la red comience a adaptarse demasiado.

Como hemos comentado anteriormente, el algoritmo de retropropagación se sustenta en el método de descenso de gradiente. De hecho, el descenso de gradiente es uno de los optimizadores más populares para redes neuronales y supone la base de algunos otros, pero ¿Qué es un optimizador de una red neuronal y que papel juega en el entrenamiento?. Esta cuestión la trataremos en la siguiente sección.

3.3.2. Principales optimizadores para redes neuronales.

Los *optimizadores* son piezas básicas en el aprendizaje de redes pues establecen cómo modificar los pesos de la red para conseguir que el resultado se acerque, en cada paso, un poco más al objetivo deseado.

En esta sección, emplearemos $\mathbf{W}$ para referirnos al vector de pesos de la red y será $E(\mathbf{W})$ la función de pérdida, es decir, la función que mide el error. En el algoritmo de retropropagación presentado en la sección anterior, el optimizador empleado ha sido el descenso de gradiente clásico, es decir, los pesos se actualizaban según

$$\mathbf{W}^{[t+1]} = \mathbf{W}^{[t]} - \eta \cdot \nabla E\left(\mathbf{W}^{[t]}\right). \tag{3.14}$$

Una mala elección en el parámetro η nos puede conducir hacia un método que nunca converga al mínimo pues no necesariamente la función de pérdida es convexa. De hecho, en general no suele serlo. Con el objetivo de acelerar y garantizar que el aprendizaje

sea óptimo han surgido distintos optimizadores para redes neuronales. A continuación, se presentan algunos de los más usados pero no son los únicos.

Descenso del gradiente estócastico (SGD).

Supongamos que trabajamos con un conjunto de entrenamiento excesivamente grande. Entonces, el cálculo de cada una de las derivadas parciales de la función de pérdida respecto a cada uno de los pesos de la red para cada observación es lento e inviable. Para evitar esto, aparece el *descenso de gradiente estocástico* (SGD). Esta técnica reemplaza el gradiente real (calculado a partir del conjunto de datos completo) por una estimación del mismo (calculado a partir de un subconjunto de datos seleccionado al azar).

El pseudocódigo del descenso de gradiente estocástico para una muestra de datos $S = ((x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)}))$ es:

1. Se elige un vector inicial de pesos $\mathbf{W}$ y la tasa de aprendizaje η.

2. Repetir hasta obtener un mínimo aproximado

   Se toma una muestra aleatoria del conjunto de entrenamiento de tamaño $n < m$.

   Para $i = 1, 2, \ldots, n$, hacer

   $$\mathbf{W} = \mathbf{W} - \eta \cdot \nabla E(\mathbf{W}, x^{(i)}, y^{(i)}).$$

Si el tamaño de la muestra es suficientemente grande, puede suceder que alcancemos el mínimo del error sin necesidad de recorrer todo el conjunto inicial de datos.

Momentum.

Momentum es una variante del método anterior que introduce un término adicional al algoritmo, denominado *momento* ("momentum"), proporcional al incremento de la iteración anterior. La actualización de los pesos de la red viene dada por

$$\mathbf{W}^{[t+1]} = \mathbf{W}^{[t]} - \eta \cdot \nabla E(\mathbf{W}^{[t]}) - \alpha \cdot \Delta \mathbf{W}^{[t]}. \tag{3.15}$$

donde $\Delta \mathbf{W}^{[t]}$ es el incremento de los pesos con respecto al paso anterior y α corresponde al término momento/inercia. Para α suelen considerarse valores en el intervalo (0,1) generalmente próximos a 1. Este método acelera el procedimiento SGD.

Adagrad - Algoritmo de gradiente adaptativo.

El algoritmo AdaGrad introduce una variación muy interesante en la tasa de apren-

dizaje: en vez de considerar un valor uniforme para todos los pesos, se establece un tasa específica para cada uno de ellos.

Inicialmente hemos de considerar la matriz $G^{(t)}$,

$$G^{(t)} = \sum_{i=1}^{t} \nabla E\left(\mathbf{W}^{[i]}\right) \nabla E\left(\mathbf{W}^{[i]}\right)^T \tag{3.16}$$

La diagonal de $G^{(t)}$ contiene la suma de los cuadrados de los gradientes anteriores hasta la iteración t-ésima, es decir,

$$G_{j,j}^{(t)} = \sum_{i=1}^{t} \left(\nabla E\left(\mathbf{W}^{[i]}\right)\right)^2 \tag{3.17}$$

Hemos realizado un pequeño abuso de notación expresando como $\left(\nabla E\left(\mathbf{W}^{[i]}\right)\right)^2$ el hecho de que cada una de las componentes del gradiente este elevada al cuadrado. $\left(\nabla E\left(\mathbf{W}^{[i]}\right)\right)^2$ contiene el cuadrado cada una de las derivadas parciales de E con respecto a cada componente de $\mathbf{W}^{[i]}$.

Finalmente, la regla de actualización de los pesos es

$$\mathbf{W}^{[t+1]} = \mathbf{W}^{[t]} - \frac{\eta}{\sqrt{G_{j,j}^{(t)} + \epsilon}} \cdot \nabla E\left(\mathbf{W}^{[t]}\right). \tag{3.18}$$

donde ϵ es simplemente una constante positiva que evita la división por cero.

En el denominador, a cada paso, acumulamos la suma de los cuadrados de los gradientes pasados. A medida que avanzamos en iteraciones, el denominador en (3.18) crece hasta hacer la tasa de aprendizaje η infinitesimalmente pequeña y llegaremos a un punto en el que algoritmo ya no pueda aprender. Adagrad puede mejorar significativamente la solidez del SGD y ser usardo para el entrenamiento de grandes redes neuronales.

RMSProp (Root Mean Square Propagation).

RMSProp es una extensión de AdaGard que busca solucionar el problema de acumulación de gradientes. Para eso lo que se hace es calcular una media móvil exponencial, definida de manera recursiva. Esta media móvil únicamente tiene en cuenta los pesos y los gradientes de las etapas más recientes. Si denotamos por M_t a dicha media en el paso t, la formula sería la siguiente:

$$M_t(\mathbf{W}) = \gamma M_{t-1}(\mathbf{W}) + (1-\gamma) \cdot \nabla E(\mathbf{W}) \tag{3.19}$$

Es habitual asignar el valor 0.9 a γ.

La regla de actualización para los pesos es

$$\mathbf{W}^{[t+1]} = \mathbf{W}^{[t]} - \frac{\eta}{\sqrt{M_t(\mathbf{W}^{(t)}) + \epsilon}} \cdot \nabla E\left(\mathbf{W}^{[t]}\right). \tag{3.20}$$

Podemos observar que la regla es idéntica a AdaGard solo que sustituyendo $G^{(t)}_{(j,j)}$ por M_t. Lo que logramos con esto es que los gradientes de tiempos muy lejanos en la ejecución no tengan tanto peso al momento de actualizar los parámetros, evitando la disminución excesiva de la tasa de aprendizaje.

Capítulo 4

Error y entrenamiento de una red neuronal en R.

En este último capítulo, realizaremos una discusión sobre dos de los factores más relevantes a la hora entrenar redes neuronales: la muestra de entrenamiento y el optimizador empleado. Para ello, recurriremos al programa RStudio. Concretamente, ha sido necesario la instalación de Keras, una librería de código abierto escrita en Pyhton que está especialmente creada para posibilitar la generación y experimentación con redes neuronales.

4.1. Modificación en la muestra inicial.

Es evidente que las redes neuronales gozan de gran flexibilidad, pues en la mayoría de las ocasiones vienen determinadas por un gran número de parámetros. Esto supone un aumento en la complejidad lo que conlleva inmediatamente a un incremento en la probabilidad de caer en el sobreajuste. Hacer que la clase de redes neuronales dependa de más parámetros guiados por el objetivo de minimizar el error, puede implicar que el modelo de red se adapte en exceso a los datos de entrenamiento y ante nuevas predicciones lleve a cabo asignaciones pésimas. Por tanto, conviene realizar un estudio más detallado de cuando una red neuronal cae en el sobreajuste y mostrar cómo las redes neuronales en ciertas ocasiones son especialmente útiles mientras que en algunas situaciones no son idóneas. Con este objetivo, plantearemos dos escenarios.

En primer lugar, dejaremos claro la arquitectura de la red que vamos a entrenar. Será un red con una única capa oculta y una sola una neurona en la capa de salida pues recordemos que deseamos llevar a cabo problema de clasificación binaria. El conjunto de entrenamiento estará constituido inicialmente por 100 atributos, de los cuales 50 de ellos pertenecerán a un grupo (la etiqueta asignada es 0) y los 50 restantes formarán parte del

grupo cuya etiqueta es 1.

4.1.1. Clasificación para dos grupos nítidamente diferenciados.

Comenzaremos analizando como realiza la clasificación una red neuronal para dos grupos notablemente separados. La generación de las dos familias y su representación en dos dimensiones queda reflejado en la figura 4.1.

```
X1<-matrix(rnorm(500), ncol=10, nrow = 50)
X2<-matrix(rnorm(500), ncol=10, nrow = 50)
X2[,1:5]<-X2[,1:5]+3
```

```
plot(X1[,1:2], xlab=" ", ylab = " " , xlim=c(-3,5),ylim=c(-3,5))
points(X2[,1:2], col=2)
```

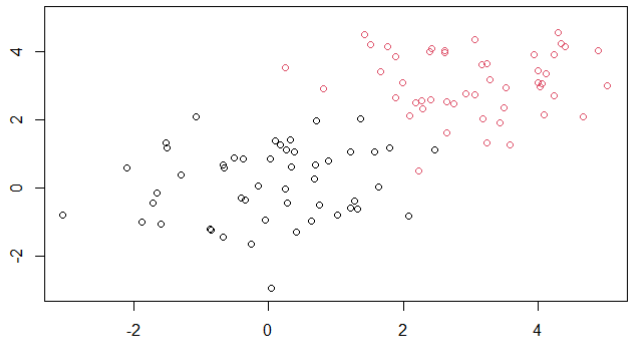

Figura 4.1: Gráfico primera población.

Cada atributo es una fila en la matriz `X1` o `X2`. La modificación de un grupo respecto al otro simplemente ha consistido en sumar 3 unidades al valor en las 5 primeras dimensiones. Procederemos posteriormente a unir ambos conjuntos y a crear el vector de etiquetas.

```
data<-rbind(X1, X2)
labels<-c(rep(0,50), rep(1,50))
```

Dado que cada atributo viene definido por 10 variables, la entrada de la red neuronal será un vector de tamaño 10. Esto supone directamente que la primera capa ha de poseer 10 neuronas. En cuanto a la segunda capa, el número de neuronas suele ser una potencia de 2. Se ha decidido que la capa intermedia cuente con 32 neuronas. Finalmente, como ya hemos mencionado, la última capa posee una sola neurona.

La función de activación empleada en la capa de entrada e intermedia ha sido la RELU. Dicha función ha ido sustituyendo con el tiempo a la sigmoide. Sin embargo, los resultados teóricos en torno a ella son escasos. Por su parte, la última neurona cuenta con la función lineal. Se ha utilizado el comando `layer_dense()` para indicar que deseamos capas de neuronas "densamente" conectadas, es decir, cada neurona de una capa recibe información sobre todas las neuronas de la capa anterior.

```
model <- keras_model_sequential() %>%
  #Capa de entrada
 layer_dense(input_shape = c(10), units=10, activation = 'relu') %>%
  #Capa intermedia
 layer_dense(units=32, activation = 'relu') %>%
  #Capa de salida
 layer_dense(units=1, activation = 'linear')
```

El resumen del modelo es el siguiente:

```
summary(model)
```

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## dense_2 (Dense)                     (None, 10)                      110
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 32)                      352
## ________________________________________________________________________________
## dense (Dense)                       (None, 1)                       33
## ================================================================================
## Total params: 495
## Trainable params: 495
## Non-trainable params: 0
## ________________________________________________________________________________
```

Podemos observar como el número de parámetros que definen el modelo de red es relativamente alto, 495. Esto hace que la red neuronal sea más flexible pues podremos modificar y ajustar más parámetros, y obtener así una mejor clasificación.

Antes de comenzar el proceso de entrenamiento, debemos fijar que función de pérdida deseamos emplear para medir el error y que optimizador queremos utilizar.

```
model %>% compile(
  optimizer = optimizer_rmsprop(),
  loss = loss_binary_crossentropy,
  metrics = c('accuracy')
)
```

La función de pérdida empleada es la de entropía cruzada. También conocida como

pérdida logarítmica. Se define como

$$L = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot \log\left(\mathbb{P}[y_i = 1]\right) + (1 - y_i) \cdot \log\left(1 - \mathbb{P}[y_i = 1]\right). \tag{4.1}$$

donde n es el número total de atributos de la muestra de entrenamiento.

En cuanto, al optimizador empleado se ha decidido usar RMSProp descrito anteriormente.

```
fit <- model %>%
  fit(
    data,labels,
    batch_size = 20,
    validation_split = 0.2,
    epochs = 30,
  )
```

El código anterior entrena el modelo para un número fijado de iteraciones (`epochs=30`), con actualizaciones de gradiente cada 20 ejemplos (`batch_size`), es decir, `batch_size` me indica el número de muestras que se propagan por la red en cada iteración: se extraen inicialmente las 20 primeras observaciones del conjunto y se entrena la red, posteriormente se toman los 20 siguientes atributos y se entrena la red, y así sucesivamente.... Es habitual referirse a este tipo de entrenamiento como entrenamiento por lotes o modo batch. Por su parte, `validation_split` nos permite decidir que porcentaje del conjunto total queremos que sea destinado a ser conjunto de validación.

Los gráficos 4.2 nos muestran tanto la pérdida en cada iteración (gráfico superior) como la precisión (gráfico inferior) para el conjunto de entrenamiento y para el de validación.

Se puede observar claramente como la pérdida, que mide el error cometido, disminuye a medida que la red neuronal avanza en el proceso de entrenamiento (linea rosa). En cuanto a la precisión, sucede lo contrario: crece con las iteraciones en el conjunto de entrenamiento. Observando el conjunto de validación (linea y puntos azules), que es el que me permite estimar el error y la precisión reales durante el proceso, su pérdida es pequeña y su precisión elevada. Luego, estamos en condiciones de afirmar que el entrenamiento está siendo satisfactorio y que este diseño de red neuronal está llevando a cabo una buena tarea de clasificación.

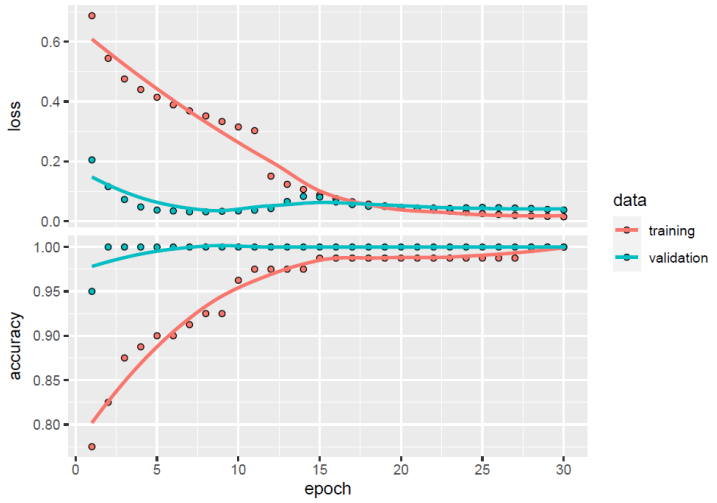

Figura 4.2: Pérdida y precisión para la primera población.

4.1.2. Clasificación para dos grupos similares.

Trabajemos con otra muestra cuya división en dos categorías no sea tan clara: el parámetro de modificación de una respecto de la otra es menor. Se ha decidido modificar únicamente en 0.5 las 5 primeras dimensiones. Podemos observar en la figura 4.3 como la división de los grupos no es nada clara.

```
X1<-matrix(rnorm(500), ncol=10, nrow = 50)
X2<-matrix(rnorm(500), ncol=10, nrow = 50)
X2[,1:5]<-X2[,1:5]+0.5
```

```
plot(X1[,1:2], xlab=" ", ylab = " " , xlim=c(-3,5),ylim=c(-3,5))
points(X2[,1:2], col=2)
```

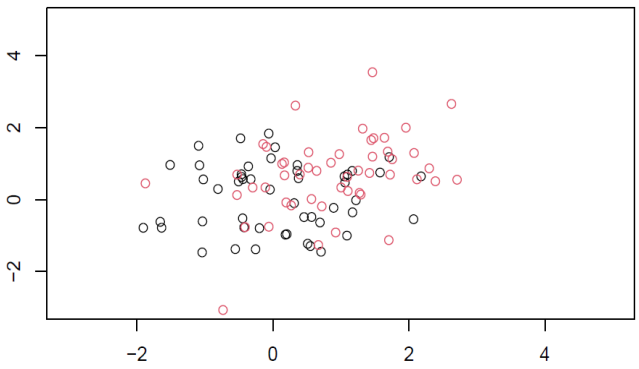

Figura 4.3: Segunda población.

La arquitectura de la red es idéntica a la anterior. En cuanto al código que compila y entrena el modelo es igual al presentado previamente. Sin embargo, los gráficos (figura 4.4) de pérdida y precisión han cambiado significativamente.

```
data<-rbind(X1, X2)
labels<-c(rep(0,50), rep(1,50))
```

```
model <- keras_model_sequential() %>%
  #Capa de entrada
 layer_dense(input_shape = c(10), units=10, activation = 'relu') %>%
  #Capa intermedia
 layer_dense(units=32, activation = 'relu') %>%
  #Capa de salida
 layer_dense(units=1, activation = 'linear')
```

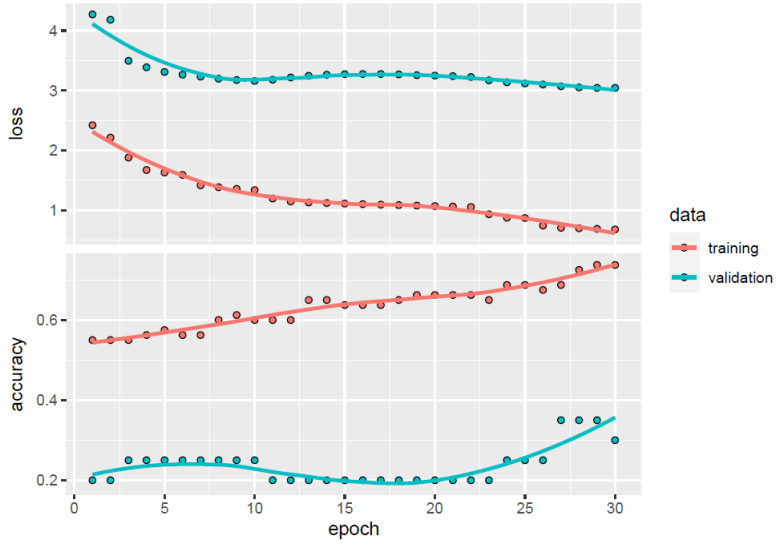

Figura 4.4: Pérdida y precisión para la segunda población.

Comentemos los resultados obtenidos. Por su parte, la precisión del conjunto de entrenamiento aumenta con las iteraciones y su pérdida disminuye aunque en general es más alta que antes. Luego, podríamos ingenuamente concluir que la clasificación que está llevando a cabo nuestra red es buena. Sin embargo, fijándonos en el conjunto de validación, este posee una pérdida muy elevada y una precisión baja a lo largo de todo el proceso. Esto quiere decir que ante nuevos datos la clasificación de la red está siendo pobre.

Recordemos que la red poseía 495 parámetros, luego su flexibilidad y adaptabilidad es alta. Debido a esto, en este segundo caso, hemos caído claramente en el sobreajuste: el diseño se ha amoldado casi a la perfección al conjunto de entrenamiento y realiza una predicción pésima ante nuevas observaciones. Esto nos muestra como aunque las redes puedan parecer herramientas perfectas para la clasificación debido a su gran flexibilidad no siempre son el modelo acertado. Por tanto, antes de enfrentarnos al problema, se ha de estudiar detenidamente cómo es el conjunto de entrenamiento y reflexionar sobre qué

tipo de red deseamos emplear.

4.2. Control del error en R para distinto optimizador.

Se nos presentarán ocasiones en las cuales será necesario reducir el número de iteraciones del procedimiento pues una elevada cantidad de pasos supone más tiempo y es más costoso. Para analizar este problema, veamos el papel que juega el optimizador y la velocidad de convergencia del error y la precisión durante el proceso dependiendo de este. Con un optimizador eficiente podremos obtener una minimización del error más rápida y por tanto, será posible disminuir el número de iteraciones.

4.2.1. Red neuronal con optimizador RMSProp.

Siguiendo con el ejemplo anterior, concretamente con el caso en el que la clasificación se ha realizado para dos grupos claramente dispersos, aumentaremos el número de iteraciones a 50.

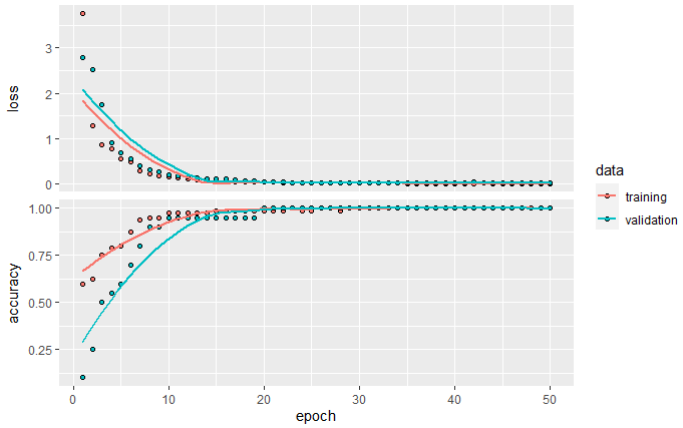

Figura 4.5: RMSProp para 50 iteraciones.

En la figura 4.5 la pérdida se estabiliza en torno a la iteración 15 y la velocidad con la que la función decrece es elevada hasta esta iteración. Aunque, el error cometido en los primeros pasos es alto, finalmente se obtiene una pérdida igual a cero. Un comportamiento similar tiene la función de precisión: crece rápidamente hasta la iteración 15 y se estabiliza a partir de esta.

Reduciendo a la mitad el número de pasos, obtenemos la figura 4.6. De nuevo, ambas funciones se estabilizan antes de la mitad de las iteraciones (en torno a la décima). En este caso, el entrenamiento de la red también es especialmente bueno: el error es bajo y su precisión alta en el conjunto de validación. Parece razonable pensar que RMSProp lleva a cabo una buena tarea de optimización incluso reduciendo el número de etapas. Comparémoslo con AdaGard.

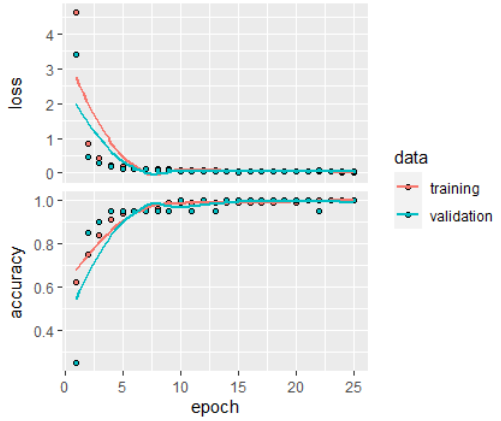

Figura 4.6: RMSProp para 25 iteraciones.

4.2.2. Red neuronal con optimizador AdaGard.

Comencemos realizando el entrenamiento para 50 iteraciones (figura 4.8).

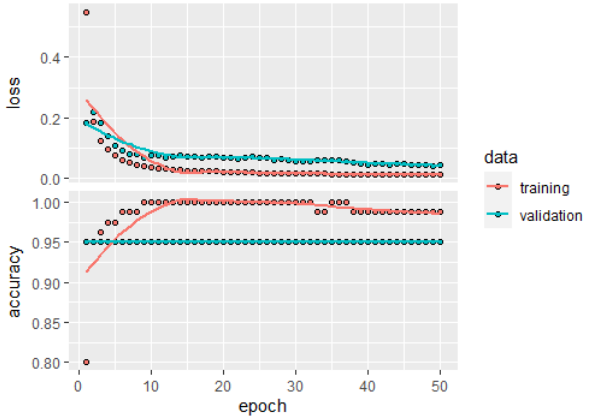

Figura 4.7: Adagard para 50 iteraciones.

Tanto el error como la precisión se estabilizan rápidamente en torno a la iteración 15. La velocidad de decrecimiento en la gráfica superior de la figura 4.7 es menor que la velocidad de decrecimiento en el caso de RMSProp.

Los gráficos obtenidos (figura 4.8) para 25 iteraciones difieren significativamente del caso anterior. Ambas funciones no se estabilizan hasta casi finalizar el entrenamiento: en torno al paso 23 ya obtenemos un error nulo y una precisión alta para el conjunto de validación. Su velocidad de convergencia está siendo menor que en el caso anterior. Luego, tiene sentido afirmar que AdaGard no es tan eficiente en el proceso de aprendizaje con un número menor de iteraciones como RMSProp.

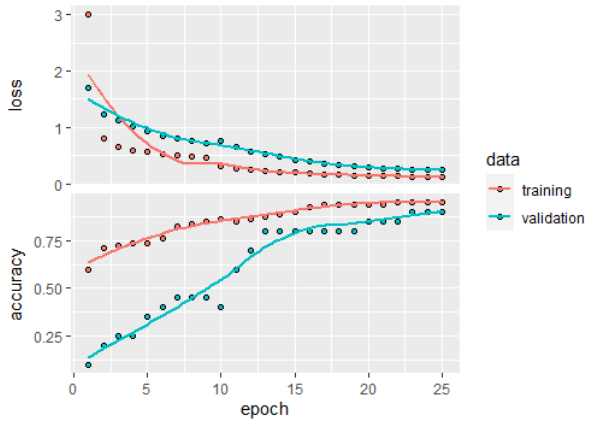

Figura 4.8: AdaGard para 25 iteraciones.

Con estos dos ejemplos, queda patente el hecho de que una buena elección en el optimizador es indispensable para realizar no solo una buena minimización del error sino también para realizar tal minimización rápidamente.

Conclusiones.

El aprendizaje estadístico se está convirtiendo en una herramienta cada vez más y más potente. Aunque su desarrollo puramente teórico no comenzó hasta hace apenas 40 años, la investigación en este ámbito ha crecido rápidamente pues su aplicabilidad es enorme. Por ejemplo, los bancos y otras empresas de la industria financiera utilizan el aprendizaje automático para fines como identificar oportunidades de inversión o prevenir fraudes. En el ámbito de la medicina también está suponiendo un instrumento clave. Gracias al machine learning se hace posible evaluar la salud de un paciente en tiempo real y ayudar a médicos a analizar datos para identificar tendencias que puedan llevar a diagnósticos y tratamientos mejorado. En el campo del marketing y las ventas, los sitios web que recomiendan artículos que podrían gustarnos en base a compras anteriores, utilizan el aprendizaje estadístico para analizar el historial de compras y promocionar otros artículos similares. Para garantizar que todos estos ejemplos de aplicaciones sean realmente eficientes se hace necesario desarrollar herramientas teóricas potentes capaces de respaldar y garantizar su efectividad. Por esto, a día de hoy el aprendizaje estadístico está en auge.

El nuevo paradigma de aprendizaje (aprendizaje PAC) presentado por el ruso Vladimir Vapnik sienta la base de toda la teoría conocida actualmente. Nos ha permitido concluir resultados muy interesantes sobre el tamaño mínimo necesario de la muestra así como garantizar la probabilidad con la cual llevaremos a cabo una buena tarea de aprendizaje. Además, y gracias al trabajo común con Alexey Chervonenkis, se han presentado nociones que nos han guiado a la demostración del Teorema fundamental del aprendizaje estadístico.

Las técnicas de aprendizaje supervisado son multitud: árboles de decisión, máquinas de vectores, K-vecinos más próximos... En esta memoria, nos hemos centrado en las redes neuronales. En primer lugar, hemos presentado algunos resultados teóricos que nos proporcionan garantías de su buen funcionamiento y su capacidad de implementación. Todo ello con ayuda de conceptos como la dimensión-VC, el Lema de Sauer, la función de crecimiento... introducidos previamente. A a pesar de estos resultados, la práctica difiere de la teoría. Gracias al R se ha podido observar el comportamiento real de una red neuronal. Ha quedado patente que si deseamos una mayor flexibilidad o capacidad de adaptación de la red para reconocer patrones que sean sumamente parecidos, se deberá invertir más tiempo en lograr que la red converja a los valores de pesos adecuados. Además, estas

simulaciones nos han mostrado el problema del sobreajuste, principal obstáculo al que se enfrenta cualquier técnica de aprendizaje estadístico, y en especial las redes neuronales. Por tanto, hay que ser catuleosos pues no siempre son el método idóneo.

Dado que se necesita una cantidad importante de recursos de un ordenador para entrenar y ejecutar una red neuronal con buenos resultados, se ha prestado atención a la posibilidad de reducir el número de iteraciones en el proceso de entrenamiento. Cambiar de optimizador puede suponer una velocidad de convergencia al mínimo error más rápida. Por lo tanto, hemos de atender no solo al modelo de red, sino también al optimizador.

Queda acreditado que las redes son una herramienta de indudable utilidad e importancia, pues son capaces de reconocer personas y objetos en imágenes, de clasificar correctamente clientes en solventes o no, de llevar a cabo traducciones instantáneas... Sin lugar a dudas, las redes neuronales se convertirán en un método cada vez más robusto a medida que la teoría en la que se fundamentan sea más amplia y sólida.

Apéndice A

El propósito del presente apéndice es citar algunos de los resultados necesarios para justificar la teoría desarrollada en esta memoria.

Supongamos que trabajamos con un vector aleatorio (X, Y) siendo $\mathbb{P}_X$ y $\mathbb{P}_Y$ las funciones de probabilidad respectivamente de X e Y y $\mathbb{P}_{X,Y}$ la función de probabilidad conjunta de ambas variables. Al valor esperado de Y conociendo $X = x$ se llama esperanza de Y condicionada por $X = x$ y se denota por $\mathbb{E}[Y \mid X = x]$.

En el caso discreto, la esperanza condicionada viene dada por la expresión

$$\mathbb{E}[Y \mid X = x] = \sum_y y\mathbb{P}(Y = y \mid X = x) = \sum_y y\frac{\mathbb{P}_{X,Y}(x, y)}{\mathbb{P}_X(x)}.$$

Para el caso continuo, se obtiene

$$\mathbb{E}[Y \mid X = x] = \int_{-\infty}^{\infty} y f_{Y|X=x}(y)dy = \int_{-\infty}^{\infty} y\frac{f_{X,Y}(x, y)}{f_{X,Y}(x)}(y)dy.$$

donde $f_{Y|X=x}$ es la función de densidad de la variable $Y|X = x$ y $f_{X,Y}$ es la función de densidad conjunta del vector aleatorio (X, Y).

$\mathbb{E}[Y|X = x]$ es una nueva variable aleatoria de la cual podemos calcular su esperanza. De hecho, si X tiene esperanza finita e Y es una variable aleatoria cualquiera, se cumple

$$\mathbb{E}[\mathbb{E}[Y|X = x]] = \mathbb{E}[Y]. \tag{A.1}$$

Otra propiedad interesante de la esperanza condicionada que nos será útil es la siguiente

$$Y \text{ y } g(X)Y \text{ tienen varianza finita} \Rightarrow \mathbb{E}[g(X)Y|X] = g(X)\mathbb{E}[Y|X]. \tag{A.2}$$

Para justificar la ecuación (1.1), en primer lugar entenderemos la esperanza condicionada de Y conociendo el valor $X = x$ como una función en la variable X, $\phi(X) = \mathbb{E}[Y|X]$. Tomando la función $h : X \to Y$,

$$\begin{aligned}\mathbb{E}[(h(X) - Y)^2] &= \mathbb{E}[(h(X) - \phi(X) + \phi(X) - Y)^2] = \\ &= \mathbb{E}[(h(X) - \phi(X))^2] + \mathbb{E}[(Y - \phi(X))^2] + 2\mathbb{E}[(h(X) - \phi(X))(\phi(X) - Y)].\end{aligned}$$

Comprobemos que el último término de la expresión anterior se anula.

$$\begin{aligned}\mathbb{E}[(h(X) - \phi(X)(\phi(X) - Y)] &= \mathbb{E}[h(X)\phi(X)] - \mathbb{E}[Yh(X)] + \mathbb{E}[Y\phi(X)] - \mathbb{E}[\phi(X)\phi(X)] = \\ &= \mathbb{E}[h(X)\mathbb{E}[Y|X]] - \mathbb{E}[Yh(X)] + \mathbb{E}[Y\mathbb{E}[Y|X]] - \mathbb{E}[\phi(X)\mathbb{E}[Y|X]].\end{aligned} \tag{A.3}$$

Haciendo uso de (A.1) y (A.2), para el primer y el último sumando se tiene

$$\begin{aligned}\mathbb{E}[h(X)\mathbb{E}[Y|X]] &= \mathbb{E}[\mathbb{E}[h(X)Y|X]] = \mathbb{E}[h(X)Y] \\ \mathbb{E}[\phi(X)\mathbb{E}[Y|X]] &= \mathbb{E}[\mathbb{E}[\phi(X)Y|X]] = \mathbb{E}[\phi(X)Y] = \mathbb{E}[Y\mathbb{E}[Y|X]].\end{aligned}$$

Combinando esto con (A.3), se obtiene $\mathbb{E}[(h(X) - \phi(X)(\phi(X) - Y)] = 0$ y por tanto,

$$\mathbb{E}[(h(X) - \phi(X)(\phi(X) - Y)] = \mathbb{E}[(h(X) - \phi(X))^2] + \mathbb{E}[(Y - \phi(X))^2]. \tag{A.4}$$

Lema A.1. *(Desigualdad de Hoeffding). Sean $\theta_1, \ldots, \theta_m$ variables aleatorias i.i.d y tales que para cada $i \in \{1, \ldots, m\}$, $\mathbb{E}[\theta_i] = \mu$ y $\mathbb{P}[a \leq \theta_i \leq b] = 1$. Entonces, para cualquiera $\epsilon \geq 0$, se tiene*

$$\mathbb{P}\left[\left|\frac{1}{m}\sum_{i=1}^{m}\theta_i - \mu\right| \leq \epsilon\right] \leq 2\exp(-2m\epsilon^2/(b-a)^2).$$

Lema A.2. *Sea Z una variable aleatoria que toma valores en $[0, 1]$. Supongamos que $\mathbb{E}[Z] = \mu$. Entonces para cualquiera $a \in (0, 1)$, se tiene:*

$$\mathbb{P}[Z > 1 - a] \leq \frac{\mu - (1 - a)}{a}. \tag{A.5}$$

Lema A.3. *Sea X una variable aleatoria y $x' \in \mathbb{R}$ un escalar. Supongamos que existen $a > 0$ y $b \geq 0$ tales que para todo $t \geq 0$ se tiene $\mathbb{P}[|X - x'| > t] \leq 2be^{-t^2/a^2}$. Entonces, $\mathbb{E}[|X - x'|] \leq a(2 + \sqrt{\log(b)})$.*

Lema A.4. *Sea* $a \geq 1$ *y* $b > 0$. *Si* $x \geq 4a\log(2a) + 2b \Rightarrow x \geq a\log(x) + b$.

Las demostraciones de los resultados precedentes pueden encontrarse en [1].

Referencias.

[1] Shalev-Shwartz, S. y Ben-David, S. (2014), *Understanding Machine Learning*, Cambridge University Press, 425-426.

[2] Mohri M., Rostamizadeh A. y Talwalkar A. (2018), *Foundations of Machine Learning* (2da ed.), The MIT Press.

[3] Valiant, L. G. (1984), *A theory of the learnable*, Communications of the ACM Vol 27, 1134-1142.

[4] Vladimir N. Vapnik,(2004), *The Nature of Statistical Learning Theory*, Springer-Verlag New York, Inc.

[5] V. Vapnik and A. Chervonenkis, (1971), On the uniform convergence of relative frequencies of events to their probabilities, *Theory of Probability and its Applications*, 16(2), 264-280.

[6] F.Rosenblatt, (1958), The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, 65(6), 386-408.

[7] A. Cauchy, (1847), Méthode générale pour la résolution des systèmes d'équations simultanées, *Comptes Rendus de l'Académie des Sciences*, 523(25), 536-538.

[8] A. Ruder, (2016), An overview of gradient descent optimization algorithms, *ArXiv, abs/1609.04747*.

[9] S. Bubeck, (2015), Convex Optimization: Algorithms and Complexity, *Foundations and Trends in Machine Learning*, Vol 8, 231–358