Universidade Vigo

Trabajo Fin de Máster

Métodos de ensamblado en Machine Learning

Ricardo Recarey Fernández

Máster en Técnicas Estadísticas

Curso 2020-2021

Propuesta de Trabajo Fin de Máster

Título en galego: Métodos de ensamblaxe en Machine Learning
Título en español: Métodos de ensamblado en Machine Learning
English title: Ensemble methods in Machine Learning
Modalidad: Modalidad A
Autor/a: Ricardo Recarey Fernández, Universidade de Santiago de Compostela
Director/a: Manuel Febrero Bande, Universidade de Santiago de Compostela
Tutor/a:
Breve resumen del trabajo: En este trabajo se hará una revisión teórico-práctica de los métodos de ensamblado de algoritmos de aprendizaje estadístico, comentando los distintos tipos que existen y su motivación (destacando *stacking*, *bagging* y *boosting*), así como sus ventajas e inconvenientes.
Recomendaciones:
Otras observaciones:

Don/doña Manuel Febrero Bande, Catedrático de la Universidade de Santiago de Compostela, informa que el Trabajo Fin de Máster titulado

Métodos de ensamblado en

Machine Learning

fue realizado bajo su dirección por don/doña Ricardo Recarey Fernández para el Máster en Técnicas Estadísticas. Estimando que el trabajo está terminado, dan su conformidad para su presentación y defensa ante un tribunal.

En Santiago, a 7 de septiembre de 2021.

El/la director/a:

Don/doña Manuel Febrero Bande

El/la director/a:

Don/doña

El/la tutor/a:

Don/doña

El/la tutor/a:

Don/doña

El/la autor/a:

Don/doña Ricardo Recarey Fernández

Índice general

Resumen IX

Prefacio XI

1. Aprendizaje estadístico supervisado **1**

1.1. Teoría estadística de la decisión 1

1.2. Terminología del aprendizaje estadístico 4

1.3. Modelos base relevantes para construir ensamblados 10

2. Técnicas de ensamblado estadístico **17**

2.1. Motivación 17

2.2. Terminología y taxonomía 19

2.3. Ejemplos de técnicas de ensamblado 21

2.3.1. Promedio de modelos Bayesiano 21

2.3.2. Votación de la mayoría del comité 22

2.3.3. Combinación ingenua Bayesiana (*Naive Bayes ensemble*) 23

2.3.4. Bagging 23

2.3.5. Random Forest 25

2.3.6. Boosting 29

2.3.7. Stacking 45

3. Ensamblados estadísticos en la práctica **47**

3.1. Introducción 47

3.2. Problemas planteados, resultados y conclusiones 51

3.2.1. **P1**: Clasificación de nivel de ingresos de población adulta 52

3.2.2. **P2**: Clasificación de dígitos escritos a mano 54

3.2.3. **P3**: Predicción del valor de un coche usado 56

A. Demostraciones **63**

A.1. Teorema 2.1 63

B. Código Python **67**

B.1. Código genérico para todos los problemas 67

B.2. Código para problema P1 68

B.3. Código para problema P2 70

B.4. Código para problema P3 71

Bibliografía **73**

Resumen

Resumen en español

En el marco del aprendizaje automático existen un amplio abanico de técnicas estadísticas para resolver problemas de carácter supervisado, tanto para tareas de regresión como de clasificación. En todas estas el objetivo principal es construir, a partir de un conjunto de datos catalogado, un modelo matemático que consiga predecir la variable respuesta para nuevos datos no observados anteriormente.

En este contexto, se denominan métodos de ensamblado estadístico al conjunto de técnicas que nos permiten combinar varios modelos a la hora de realizar predicciones en vez de utilizar uno solo. Desde las más sencillas (votación de comité con o sin pesos), hasta las más sofisticadas (*Random Forest*, *Boosting* y sus variantes), estas técnicas están siendo extensa y exitosamente utilizadas en un amplio espectro de problemas en la actualidad.

En este trabajo se hará una revisión teórico-práctica de estas técnicas, comentando los distintos tipos que existen, las motivaciones de cada uno (destacando bagging, *random forest*, boosting y *stacking*) y sus ventajas e inconvenientes frente a la utilización de un único modelo dependiendo del tipo de problema.

English abstract

Within the framework of machine learning, there is a wide range of statistical techniques to solve supervised problems, both for regression and classification tasks. In all of these, the main objective is to build, from a cataloged data set, a mathematical model that manages to predict the response variable for new data not previously observed.

In this context, ensemble methods is the set of techniques that allow us to combine several models when making predictions, instead of using only one. From the simplest (committee vote with or without weights), to the most sophisticated (Random Forest, boosting and its variants), these techniques are being widely (and successfully) used in a wide spectrum of problems today.

In this work a theoretical-practical review of these techniques will be made, commenting on the different types that exist and the motivations of each one (highlighting bagging, random forest, boosting and stacking) and its advantages and disadvantages compared to the use of a single model, depending on the type of problem.

Prefacio

> La capacidad de decisión de un comité de personas es superior a la capacidad de los miembros que lo componen individualmente, supuesto que cada uno de los miembros tiene una competencia razonable.
>
> *Teorema del jurado de Condorcet*

Se puede definir el aprendizaje estadístico como el conjunto de técnicas matemáticas que nos permiten entender y sacar partido de un conjunto de datos, ya sea para labores de interpretación de los mismos o para realizar predicciones.

Estas metodologías, junto con los grandes avances computacionales de los últimos años, son las principales responsables del actual auge de la inteligencia artificial y de sus numerosos éxitos a la hora de resolver problemas industriales que hace solo unas décadas parecían imposibles. Dentro de los casos de éxito, caben destacar los siguientes:

- **Algoritmos de recomendación**. Esta área del aprendizaje estadístico es una de las áreas de investigación más activas en la actualidad, de la mano del gran crecimiento de la industria de servicios de internet y su necesidad por personalizar qué se le ofrece y a quién. Destaca el popular *Netflix Prize* de 2009, en donde la compañía Netflix ofreció un millón de dólares a quien fuese capaz de mejorar su algoritmo de recomendación de películas un 10 %.

- **Clasificación de imágenes**. Los avances en el aprendizaje estadístico, mayoritariamente los relacionados con las redes neuronales convolucionares, fueron clave en la creación de sistemas de clasificación de imágenes ampliamente utilizados en la actualidad. Por ejemplo, los sistemas de reconocimiento facial o los sistemas de diagnóstico temprano de numerosas patologías a partir de imágenes del paciente, entre otros.

- **Conducción autónoma**. Los recientes avances en técnicas estadísticas de visión por computador y detección de objetos (destacando las redes neuronales convolucionales y las redes recurrentes) son la piedra angular del nacimiento de la conducción autónoma, una de las tecnologías que está revolucionando la industria del transporte.

- **Procesamiento de lenguaje natural (PLN)**. Herramientas como los asistentes de voz (Siri, Alexa...), sistemas de traducción e interpretación de textos, bots conversacionales, etc., son posibles gracias a los numerosos avances en la rama del PLN (relacionados la mayoría con las redes neuronales recurrentes).

- **Detección de fraude**. Los sistemas de detección de transacciones bancarias fraudulentas o de detección de *spam* en los correos electrónicos son otros ejemplos de servicios hechos posibles gracias a estas técnicas estadísticas.

Todos estos problemas encajan dentro del marco del aprendizaje estadístico antes mencionado. La mayoría de ellos, en concreto, son problemas de aprendizaje estadístico **supervisado**, en los cuales se centrará el presente proyecto. Este tipo de problemas se puede definir como:

Definición 0.1 *Dados $\boldsymbol{X} \subseteq \mathbb{R}^p$, $\boldsymbol{Y} \subseteq \mathbb{R}$ espacios vectoriales. Sea el conjunto $S = \{(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_N, y_N)\}$ formado por los N pares $(\boldsymbol{x}_i, y_i)$, denominado conjunto de entrenamiento. Sea $\boldsymbol{x}_i \in X$ el vector de características de i-ésimo par (también conocido como variables regresoras), e $y_i \in \mathbb{R}$ su variable respuesta. Un algoritmo de aprendizaje estadístico es aquel que busca una función $f(\boldsymbol{x})$ que sea capaz de predecir, dado un nuevo dato con vector de características $\boldsymbol{x}_0$, su correcta variable respuesta y_0.*

Dependiendo de si la variable que se quiere predecir es continua o categórica se estará ante un problema de regresión o clasificación, respectivamente. Concretando sobre los ejemplos anteriores, en un algoritmo de clasificación de imágenes que pretende detectar números escritos a mano, se estaría ante un problema de clasificación. En el, $\mathbf{x}_i$ sería el vector de características que definen a la i-ésima imagen, y $y_i \in \boldsymbol{Y} = \{0, 1, ..., 9\}$ sería la variable respuesta (en este caso categórica) que indica el dígito que aparece en esa imagen.

Por otro lado, un ejemplo de problema de regresión es un tasador automático de precio de coches de segunda mano, el cual a partir de las características de un vehículo nos haría una predicción de su valor de mercado. En ese caso, $\mathbf{x}_i$ sería el vector de características que definen al i-ésimo coche (año de compra, kilometraje, cv, etc.), y $y_i \in \mathbb{R}$ sería la variable respuesta continua que indica su precio.

Las técnicas analizadas a lo largo de este texto, en su gran mayoría, están capacitadas para resolver tanto problemas de regresión como de clasificación. No obstante, la mayoría de resultados aquí expuestos se centrarán en la clasificación, al ser este el problema en el que se centran la gran mayoría de trabajos en el ámbito del ensamblado estadístico.

Ensamblado estadístico: introducción y motivación

De una manera menos rigurosa, se puede decir que un algoritmo de aprendizaje supervisado intenta buscar, dentro de un espacio de hipótesis, el modelo que se ajuste mejor a los datos de entrenamiento disponibles.

Como se verá en la siguiente sección en detalle, existen una amplia gama de técnicas estadísticas para buscar y encontrar la mejor hipótesis para un problema dado, cada una con sus ventajas e inconvenientes dependiendo del escenario en cuestión. No obstante, aún existiendo modelos que se ajusten bien a los datos dentro del espacio de hipótesis disponible, el hecho de tener que escoger de entre ellas un único modelo que dé buenos resultados a lo largo de todo el espacio vectorial de características es una tarea de enorme dificultad.

Por ejemplo, imagínese la situación reflejada en la Figura 1. Se tienen un cojunto de datos S que fue generado por la hipótesis verdadera **H***, y existe un conjunto de hipótesis (entre las cuales destacamos H_1, H_2 y H_3) que dan lugar a buenos resultados de predicción en el conjunto de entrenamiento. Escogiendo una única hipótesis se podría seleccionar una que, aunque comportándose bien sobre S, no tenga el mejor error de generalización (es decir, no es la mejor prediciendo nuevos datos no incluidos en el conjunto de entrenamiento), o con una que no es la mejor hipótesis a lo largo de todo el espacio vectorial de características. Es decir, existen distintos riesgos a la hora de quedarnos con un único modelo.

Las técnicas de **ensamblado estadístico** nacen con el objetivo de paliar algunos de estos problemas. En vez de seleccionar una única hipótesis, estas técnicas permiten combinar distintos modelos para realizar la predicción final teniéndolos a todos en cuenta, potencialmente reduciendo el sesgo y la varianza que tendría un único modelo.

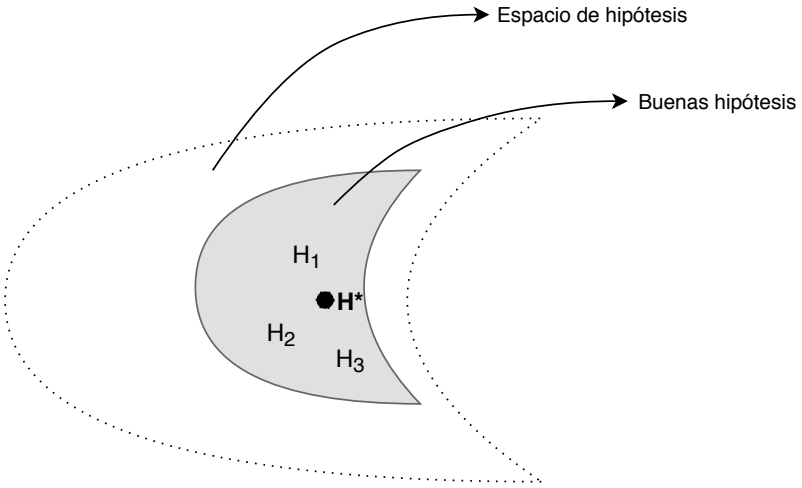

Figura 1: Representación de una de las motivaciones por la cual usar ensamblados de modelos. **H*** es la verdadera hipótesis de los datos. La zona sombreada es el área con hipótesis que se ajustan bien a los datos.

Para visualizar la justificación de por qué estas técnicas pueden ser de utilidad en ciertos casos, es de utilidad imaginarse la siguiente situación. Ante un problema de clasificación binario dado se tienen entrenados L modelos $h_1, ..., h_L$. Suponiendo que el ratio de error de clasificación de todos los modelos h_l es $p < 1/2$, y que además las predicciones de todos estos dado un nuevo dato $\mathbf{x}$ son incorreladas (ya que si todos los modelos h_l tienden a predecir lo mismo para un nuevo dato, combinarlos no va a dar lugar a ninguna mejora). Si bajo estas condiciones se utilizase como clasificador un ensamblado de las L hipótesis donde para un nuevo dato predecimos la clase que escojan la mayoría de modelos (votación de mayoría sin pesos), se tendría que la probabilidad de que la mayoría de modelos h_l cometan un error es el área debajo de la distribución binomial donde más de $L/2$ modelos se equivocan. Por ejemplo, si con estas condiciones hubiese 21 modelos a ensamblar, todos con error de clasificación $p = 0.3$, la probabilidad de que 11 o más modelos se equivoquen al mismo tiempo (equivocándose así el ensamblado antes descrito), sería de 0.026, mucho menor que el error individual de cada modelo.

Esta intuición se formaliza en Hansen y Salamon (1990), en donde los autores demuestran que una condición necesaria y suficiente para que un ensamblado de clasificadores sea mejor que sus miembros individuales es que estos sean **precisos** y **diversos**.

Se define un modelo preciso como aquel cuyas predicciones son mejores que una predicción aleatoria, y se definen dos modelos como diversos entre sí si cometen errores distintos al predecir nuevos datos $\mathbf{x}$ (es decir, que sus predicciones para un nuevo punto no estén correladas).

El objetivo principal de este proyecto es realizar un estudio teórico-práctico del estado del arte del ensamblado estadístico. Para ello, el contenido se estructurará en dos secciones: la primera tratará los distintos modelos clásicos de aprendizaje estadístico, haciendo hincapié en los que en la práctica son los más utilizados para posteriormente construir ensamblados (árboles de decisión, redes neuronales, etc.). Esta sección bebe fundamentalmente de Hastie et al. (2001). En la segunda y principal sección del proyecto se definen formalmente los ensamblados estadísticos: las razones de su existencia, su taxonomía y las técnicas más relevantes y sus propiedades (centrándose en *bagging*, *boosting* y *stacking*). Para este apartado, las referencias principales consultadas fueron Kuncheva (2014), Re y Valentini (2012) y Dietterich (2000).

Capítulo 1

Aprendizaje estadístico supervisado

Una de las partes fundamentales a la hora de construir buenos ensamblados (i.e., combinaciones) de modelos, es cerciorándose de que los modelos individuales que se van a combinar son tanto precisos como diversos. En esta sección se abordará el problema de la exactitud de los modelos y, de la mano de la teoría estadística de la decisión, se presentarán las técnicas más relevantes para construir algoritmos de aprendizaje estadístico supervisado que se ajusten de manera robusta a los datos de entrenamiento disponibles.

1.1. Teoría estadística de la decisión

Si bien el problema que se tratará de resolver es el anteriormente expuesto en la Definición 0.1, conviene antes introducir las bases teóricas sobre las que se sostiene el aprendizaje estadístico. Estas fueron fijadas mayoritariamente por Vapnik y Chervonenkis a lo largo de los años 90, y se pueden consultar sus detalles en Vapnik (1995 y 1999).

Sea $X \in \mathbb{R}^p$ un vector aleatorio (denominado habitualmente vector de características), e $Y \in \mathbb{R}$ una variable aleatoria de respuesta continua, siendo $\mathbb{P}(X, Y)$ su distribución conjunta. Supóngase que tenemos un conjunto de datos los cuales vienen generados por un modelo del tipo $Y = g(X) + \varepsilon$, con $E(\varepsilon) = 0$ y g desconocida. El objetivo será encontrar una función $f(X)$ capaz de predecir la variable respuesta Y dado un vector de características X.

Para llevar a cabo este proceso se requiere de una función de pérdida $\mathbf{L}$ que nos indique, dados Y y $f(X)$, el error que se está cometiendo en la predicción, para así intentar construir modelos que minimicen dicha cantidad. Para problemas de regresión (donde la variable respuesta es continua) las funciones de pérdida más habitual son el error cuadrático, $\mathbf{L}(Y, f(X)) = (Y - f(X))^2$, y la norma L1, $\mathbf{L}(Y, f(X)) = |Y - f(X)|$. Por otro lado, en el contexto de problemas de clasificación (i.e., donde Y es una variable categórica), un ejemplo de función de pérdida sería la función 0-1, la cual tiene la forma $\mathbf{L}(Y, f(X)) = \theta(-Y f(X))$, siendo θ la función escalonada de Heaviside:

$$\theta(x) = \begin{cases} 0 & \text{si} \quad x < 0 \\ 1 & \text{si} \quad x \geq 0 \end{cases} \tag{1.1}$$

dando así 0 cuando X se clasifica correctamente, y 1 cuando no.

En un escenario idílico en el cual conociésemos la distribución $\mathbb{P}(X, Y)$, un posible criterio para seleccionar la función f deseada sería la minimización de su error de test esperado (ETE). Este se puede definir como el error esperado que tendrá un modelo f prediciendo nuevos datos generados bajo la distribución $\mathbb{P}(X, Y)$, y se expresa matemáticamente como sigue:

$$ETE(f) = E[\mathbf{L}(Y, f(X))] = \int_{X\times Y} \mathbf{L}(Y, f(X))\mathbb{P}(x, y)dxdy.$$

Esta expresión, en el caso particular de usar la función de pérdida cuadrática, habitualmente usada para problemas de regresión por sus convenientes propiedades, se puede escribir como:

$$ETE(f) = E[Y - f(X)]^2 = \int_{X\times Y} [y - f(\mathbf{x})]^2\mathbb{P}(x, y)dxdy.$$

Así, factorizando la densidad conjunta, $\mathbb{P}(X, Y) = \mathbb{P}(Y|X)\mathbb{P}(X)$, y dividiendo la integral doble utilizando el teorema de Fubini, se obtiene:

$$ETE(f) = \int_X \left(\int_Y (y - f(\mathbf{x}))^2\mathbb{P}(y|x)dy \right) \mathbb{P}(\mathbf{x})dy$$

expresión que, por las propiedades de la esperanza condicionada, se convertiría en:

$$ETE(f) = E_X E_{Y|X}([Y - c]^2|X)$$

la cual es suficiente con minimizarla puntualmente en $\mathbf{x}$:

$$f(\mathbf{x}) = \arg\min_c E_{Y|X}([Y - c]^2|X = x)$$

y cuya solución sería:

$$f(\mathbf{x}) = E[Y|X = x]. \tag{1.2}$$

Este operador es conocido como la función de regresión, e indica que la mejor predicción posible para Y en un punto $\mathbf{x}$ concreto es su esperanza condicionada a dicho $\mathbf{x}$.

En la práctica, no obstante, la distribución conjunta $\mathbb{P}(X, Y)$ será totalmente desconocida, y lo único de lo que se dispondrá será de un conjunto de entrenamiento finito con N pares de datos $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$. Así, aunque no se puedan utilizar directamente las derivaciones hechas anteriormente, sí se podrá intentar aproximar el error de predicción esperado con métricas como la función de riesgo empírica:

$$RE(f) = \frac{1}{N}\sum_{i=1}^{N} \mathbf{L}(y_i, f(\mathbf{x}_i)) \tag{1.3}$$

la cual expresa, dado un modelo estadístico fijado $f(\mathbf{x})$ entrenado en un conjunto finito S, lo bien o mal que el modelo se ajusta a los datos de entrenamiento.

A los algoritmos de aprendizaje estadístico que intentan encontrar a partir del conjunto S una función f_S que se ajuste a los datos a traves de minimizar (1.3), se les denomina **algoritmos de minimización del riesgo empírico**.

Paralelamente, en el caso de que nuestra variable respuesta sea categórica, los conceptos anteriormente definidos son fácilmente adaptables, excepto la función de pérdida cuya naturaleza se ve modificada. Se denota como $\mathcal{G} = \{\mathcal{G}_1, ..., \mathcal{G}_K\}$ el conjunto de posibles clases de Y, cuya cardinalidad es $K = card(\mathcal{G})$. Se puede representar la función de pérdida con una matriz L de dimensión $K \times K$, en donde $\mathbf{L}(k, j)$ tendrá la "pérdida" asociada a clasificar un X que pertenece a la clase $\mathcal{G}k$ como si fuese de la clase $\mathcal{G}j$, y 0 en la diagonal. En este caso, el error de predicción esperado será:

$$ETE = E[\mathbf{L}(Y, f(X))]$$

el cual podemos reescribir de la siguiente manera:

$$ETE = E_X \sum_{k=1}^{K} L[\mathcal{G}_k, f(X)]\mathbb{P}(\mathcal{G}_k|X).$$

Análogamente al caso de regresión, se minimiza ETE puntualmente en x,

$$f(\mathbf{x}) = \arg\min_{g\in\mathcal{G}} \sum_{k=1}^{K} L[\mathcal{G}_k, g]\mathbb{P}(\mathcal{G}_k|X).$$

Esta expresión, de utilizar la función de pérdida 0-1 antes mencionada (y bastante utilizada en la práctica), se simplifica a:

$$f(\mathbf{x}) = \arg\min_{g\in\mathcal{G}}[1 - \mathbb{P}(g|X = x)].$$

Así:

$$f(\mathbf{x}) = \mathcal{G}_k \quad si \quad \mathbb{P}(\mathcal{G}|X = x) = max_{g\in\mathcal{G}}\mathbb{P}(g|X = x). \tag{1.4}$$

Esta regla de clasificación es conocida como el clasificador de Bayes, y nos indica que dado un nuevo $\mathbf{x}_0$ se debe clasificar en la clase con mayor probabilidad condicionada. Como ocurría con el caso de la regresión, al no ser conocidas las funciones de distribución anteriores, el clasificador de Bayes teórico no es posible de construir en la práctica. No obstante, puede ser de gran ayuda para hacer pruebas con datos sintéticos a partir de distribuciones conocidas y compararlo con otros algoritmos, al ser el error de este (conocido como el *Bayes rate*), el mínimo error posible para un clasificación de salida aleatoria (Hastie et al. 2001). El error de clasificación de la regla de Bayes es el siguiente:

$$1 - E[\max_j \mathbb{P}(Y = j|X)].$$

En la práctica, muchas de las técnicas de aprendizaje estadístico intentarán acercarse a dicho *Bayes rate* tratando de construir clasificadores basados en él, intentando aproximar las distribuciones de probabilidad necesarias. Algunos de estos ejemplos son los árboles de decisión, los $k-$vecinos más cercanos, la regresión logística, el clasificador de Bayes ingenuo, etc. Como se verá a continuación, la mayoría de estas técnicas asumirán que $\mathbb{P}(Y|X)$ tiene una hipotética estructura funcional (ya sea un modelo lineal, polinómico, etc.) e intentarán estimar los parámetros de dicha función a partir del conjunto de entrenamiento S. Una vez se tiene la probabilidad de que $\mathbf{x}$ pertenezca a cada una de las clases, se le asignará la más probable.

1.2. Terminología del aprendizaje estadístico

Para conocer el potencial de un modelo estadístico será crucial ser capaces de medir la exactitud del mismo. En esta sección se definen los principales conceptos necesarios para dicho propósito. Como anteriormente, se desenvolverá la teoría para el caso concreto de la regresión (utilizando en concreto la función de pérdida $\mathbf{L}(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$ del apartado anterior), para luego adaptarlo al caso de clasificación.

Una vez entrenado un modelo a través de la minimización de la función de riesgo empírica (1.3) utilizando los datos de entrenamiento $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, se dispondrá de una función $f(\mathbf{x})$ que en teoría debería, dado un nuevo vector de catacterísticas $\mathbf{x}_0$ (no antes visto por el modelo), predecir con precisión su variable respuesta y_0, i.e., $f(\mathbf{x}_0) \sim y_0$.

Así, una vez se tiene $f(\mathbf{x})$ construida, se puede usar la anteriormente comentada fórmula del **riesgo empírico** (1.3) para tener una visión de lo bien o mal que el modelo entrenado f se ajusta a los datos de entrenamiento en S. En concreto, en el caso de usar la función de pérdida del error cuadrático medio (una de las más habituales para problemas de regresión), la expresión se conoce como el error cuadrático medio del modelo:

$$ECM(f) = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))^2.$$

No obstante, como ECM fue calculado con los mismos datos con los que se entrenó el propio modelo (conjunto S), los resultados pueden no ser del todo fiables. Lo exacto que sea f prediciendo sobre los datos del conjunto de entrenamiento S no nos dará necesariamente información sobre la capacidad de generalización del modelo, es decir, la capacidad de predicción que tendrá sobre nuevos datos $\mathbf{x}_0$ no utilizados en el entrenamiento.

Esta capacidad de generalización es la que en realidad interesa medir, y se formaliza matemáticamente mediante el **error de test esperado** (ETE, también conocido como el error de generalización). Construida una función f_S a partir de un conjunto de entrenamiento S fijado, se define el error de test esperado de f_S condicionado a que se tiene el conjunto de entrenamiento S como:

$$ETE_S = E_{\mathbb{P}}[(f_S(\mathbf{x}) - y)^2] = \int_{\mathbf{x}}\int_{y}(y - f_S(\mathbf{x}))^2\mathbb{P}(\mathbf{x}, y)dyd\mathbf{x} \tag{1.5}$$

en donde se usa la función de pérdida cuadrática por sus convenientes propiedades matemáticas, aunque se podría utilizar cualquier otra. Además, la esperanza se toma sobre $(\mathbf{x}, y) \sim \mathbb{P}$ (en donde $\mathbb{P} = \mathbb{P}(X, Y)$ por simplicidad) intentando así denotar que se está calculando el error promedio sobre la distribución teórica (y desconocida) que generó nuestros datos de entrenamiento. Con respecto a la notación, a lo largo de este trabajo denotaremos como $(\mathbf{x}, y)$ a un par genérico de puntos de test generados por la distribución $\mathbb{P}$, y como $(\mathbf{x}_0, y_0)$ a un par de puntos de test concreto generados por $\mathbb{P}$ no antes visto en nuestro conjunto de entrenamiento. En el caso del conjunto S, se usará de manera ambigua para, según el caso, denotar a un conjunto de entrenamiento concreto, o a un conjunto de entrenamiento teórico generado por la distribución $\mathbb{P}(X, Y)^N$.

Así, la expresión (1.5) nos da la esperanza del error promedio que se cometerá cuando se intente predecir un nuevo vector de características $\mathbf{x}_0$ generado por la distribución $\mathbb{P}(X, Y)$. No obstante, esta expresión es verdadera para un conjunto de entrenamiento S concreto, el cual a su vez se supuso que era generado por una distribución desconocida $\mathbb{P}^N = \mathbb{P}(X, Y)^N$. Por lo tanto, como nuestra función f_S depende de dicho conjunto, será en si misma una variable aleatoria, pudiendo así ser calculada su esperanza.

Sea $\mathcal{A}$ un algoritmo de aprendizaje estadístico el cual, a partir de un conjunto de entrenamiento S, nos devuelve un modelo (o hipótesis) $f_S = \mathcal{A}(S)$. Definimos, dado un $\mathcal{A}$, el modelo esperado como:

$$\bar{f} = E_{\mathbb{P}^N}[f_S] = \int_S f_S \mathbb{P}(S) dS \tag{1.6}$$

en donde $\mathbb{P}(S)$ es la probabilidad de generar el conjunto de entrenamiento S de la distribución $\mathbb{P}^N$. De esta expresión se puede ver que $\bar{f}$ es una media ponderada de funciones, y, usando de nuevo el hecho de que f_S es una variable aleatoria, se puede calcular el error de test esperado dado únicamente el algoritmo $\mathcal{A}$, tomando así también como aleatorio el conjunto de entrenamiento S:

$$ETE = E_{\mathbb{P}}[(f_S(\mathbf{x}) - y)^2] = \int_S \int_{\mathbf{x}} \int_y (y - f_S(\mathbf{x}))^2 \mathbb{P}(\mathbf{x}, y) \mathbb{P}(S) dy d\mathbf{x} dS \tag{1.7}$$

donde S es un conjunto de entrenamiento generado por $\mathbb{P}^N$, y los pares $(\mathbf{x}, y)$ son datos de test generados por la distribución $\mathbb{P}$, dando lugar a una expresión que nos permite evaluar la capacidad de generalización que tiene un algoritmo de aprendizaje estadístico $\mathcal{A}$ con respecto a la distribución $\mathbb{P}$ de la que provienen los datos.

En la práctica, no obstante, no se podrá calcular el error de test esperado, dado que la distribución conjunta de los pares $(\mathbf{x}, y)$, $\mathbb{P}(X, Y)$, es desconocida. Sin embargo, sí existen varias técnicas para intentar aproximar dicha cantidad de una forma computacional y matemáticamente eficiente. La más utilizada, conocida como **validación cruzada** de R iteraciones (o $R-fold\ CV$ en inglés), consiste en dividir el conjunto de entrenamiento S en R subconjuntos (supóngase que de igual tamaño) para posteriormente hacer iteraciones sobre los mismos utilizando $R-1$ de ellos como conjunto de entrenamiento y el otro restante como conjunto de test sobre el cual aproximar el error de test esperado (1.5). Una vez se hace este procedimiento con todos los R subconjuntos (dando lugar a la aproximación del error de test esperado $\widehat{ETE}_i$ para cada subconjunto R_i), se puede calcular la media de todos ellos, obteniendo así el error de validación cruzada (*Cross Validation Error*):

$$CVE = \frac{1}{R} \sum_{i=1}^{R} \widehat{ETE}_i. \tag{1.8}$$

Como lo que interesa es lo bien que va a generalizar el modelo construido, el cual está entrenado con un conjunto de entrenamiento particular S, la cantidad que será más útil de estimar será el error de test de f con S fijado, ETE_S (1.5), que cuantifica lo bueno que es el modelo entrenado con ese conjunto de entrenamiento particular. No obstante, las técnicas de aproximación conocidas, entre las cuales se incluye el error de validación cruzada (CVE), únicamente estiman de manera efectiva el error de test esperado ETE (1.7), que es de utilidad para analizar el poder de predicción de un algoritmo de forma general, pero no tanto para analizar un algoritmo ya entrenado con un conjunto de datos concreto. No parece posible estimar de manera exacta el error condicional de un modelo, ETE_S, dado que solo se tiene información de un único conjunto de entrenamiento (Hastie et al 2001).

Como se comentó anteriormente, el error de entrenamiento de un modelo f entrenado sobre un conjunto particular S (ECM) y la aproximación del error test esperado (riesgo empírico, o RE) no se comportan de manera similar. En la Figura 1.1 podemos ver la relación entre estas dos cantidades.

En esta figura se puede ver la relación entre los dos tipos de errores antes mencionados en función de la complejidad del modelo a utilizar (en donde con complejidad nos referímos a la flexibilidad del modelo, es decir, número de parámetros, grados de libertad, etc.). En la gráfica de la izquierda se pueden observar distintos modelos de aprendizaje estadístico que se intentaron aproximar a partir de la muestra de datos de puntos negros (los cuales fueron generados por la línea negra continua, la cual muestra su esperanza dado X para cada $\mathbf{x}$). Como se puede observar, el modelo lineal simple se ajusta

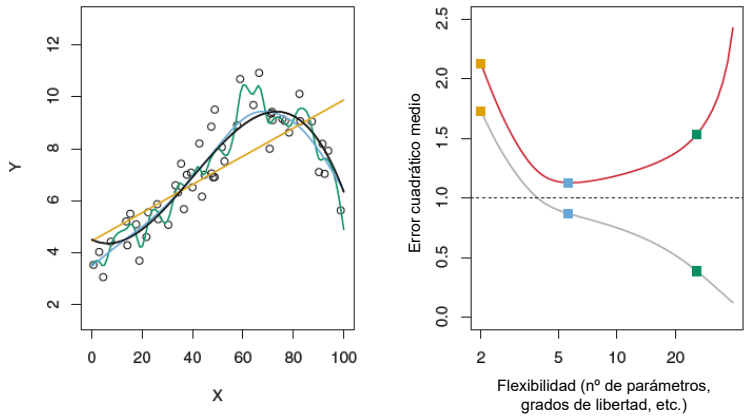

Figura 1.1: Representación del comportamiento de los errores conforme se permite más o menos flexibilidad (i.e., grados de libertad) al modelo para intentar ajustar la curva que simuló los datos (negro). En la gráfica de la derecha se puede ver como, según se aumenta la flexibilidad del modelo, por ejemplo permitiendo regresiones polinómicas, el error de entrenamiento (gris) se reduce monótamente, mientras que el error de test (en rojo) lo hace en forma de U. En el punto donde este último es mínimo se considera un buen ajuste, mientras que en los puntos en los cuales este vuelve a aumentar se considera que el modelo está sobreajustando los datos.

de manera pobre a los datos (dado que es una hipótesis más simple de lo que se necesita), dando lugar a un error alto tanto en el conjunto de test (línea roja), como en el conjunto de entrenamiento (línea gris). Conforme se aumenta la flexibilidad del modelo a utilizar (modelos no lineales, modelos no paramétricos, etc.) se ve como las funciones se ajustan cada vez mejor a los puntos negros, bajando así tanto el error de test como de entrenamiento como se puede ver en la parte central de la gráfica de la derecha. No obstante, esta flexibilidad llevada al extremo, como por ejemplo cuando aumentamos demasiado los grados de libertad de una regresión polinómica (línea de ajuste verde en la gráfica), puede ser contraproducente. En este caso, el modelo se intenta ajustar demasiado a los datos de entrenamiento, dando lugar a un **sobreajuste**. Si bien, como se puede ver en la gráfica de la derecha, el error de entrenamiento baja mucho (ya que la curva tiene la flexibilidad suficiente como para pasar casi por todos los puntos del entrenamiento), el error de test sube considerablemente (es decir, baja el poder de generalización para predecir bien datos no vistos anteriormente por el modelo).

A través de regular el equilibrio entre la flexibilidad permitida al modelo y el error de test que este comete, se dará lugar a métodos para evitar este sobreajuste. La búsqueda de este equilibrio se conoce matemáticamente como la **compensación sesgo-varianza**, y viene dada por la siguiente manera de descomponer del error de test esperado, ETE (1.7):

$$\begin{aligned} E_{\mathbf{x},y,S}[(f_S(\mathbf{x}) - y)^2] &= E_{\mathbf{x},y,S}[((f_S(\mathbf{x}) - \bar{f}(\mathbf{x})) + (\bar{f}(\mathbf{x}) - y)^2] \\ &= E_{\mathbf{x},S}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))^2] + 2E_{\mathbf{x},y,S}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x}) - y)] + E_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - y)^2]. \end{aligned} \tag{1.9}$$

En esta expresión, el término del medio evalúa a 0, dado que:

$$\begin{aligned} E_{\mathbf{x},y,S}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x}) - y)] &= E_{\mathbf{x},y}[E_S[(f_S(\mathbf{x})] - \bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x}) - y)] \\ &= E_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - \bar{f}(\mathbf{x}))(\bar{f}(\mathbf{x}) - y)] \\ &= E_{\mathbf{x},y}[0] \\ &= 0. \end{aligned} \tag{1.10}$$

Así, volviendo a la expresión anterior, quedaría:

$$E_{\mathbf{x},y,S}[(f_S(\mathbf{x}) - y)^2] = \underbrace{E_{\mathbf{x},S}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))^2]}_{\text{Varianza}} + E_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - y)^2]$$

donde el primer término es la **varianza**, y el segundo se puede continuar descomponiendo de la siguiente forma:

$$\begin{aligned} E_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - y)^2] &= E_{\mathbf{x},y}[\big((\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x})) + (\bar{y}(\mathbf{x}) - y)\big)^2] \\ &= \underbrace{E_{\mathbf{x},y}[\big(\bar{y}(\mathbf{x}) - y)\big)^2]}_{\text{Ruido}} + \underbrace{E_{\mathbf{x},y}[\big(\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x})\big)^2]}_{\text{Sesgo}^2} + 2E_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x}))(\bar{y}(\mathbf{x}) - y)] \end{aligned} \tag{1.11}$$

en donde definimos $\bar{y}(\mathbf{x})$ como la respuesta esperada para un vector de características dado $\mathbf{x}$:

$$\bar{y}(\mathbf{x}) = E_{y|\mathbf{x}}[Y] = \int_y y\mathbb{P}(y|\mathbf{x}). \tag{1.12}$$

De forma similar a lo realizado en (1.10), se puede probar que el último término es igual a 0:

$$\begin{aligned} E_{\mathbf{x},y}[(\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x}))(\bar{y}(\mathbf{x}) - y)] &= E_{\mathbf{x}}\big[E_{y|x}[\bar{y}(\mathbf{x}) - y](\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x})\big] \\ &= E_{\mathbf{x}}\big[(\bar{y}(\mathbf{x}) - E_{y|x}[y])(\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x}))\big] \\ &= E_{\mathbf{x}}\big[(\bar{y}(\mathbf{x}) - \bar{y}(\mathbf{x}))(\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x}))\big] \\ &= E_{\mathbf{x}}[0] \\ &= 0. \end{aligned}$$

Así, juntando todo lo anterior, obtendríamos la conocida como **descomposición sesgo varianza**:

$$E_{\mathbf{x},y,S}[(f_S(\mathbf{x}) - y)^2] = \underbrace{E_{\mathbf{x},S}[(f_S(\mathbf{x}) - \bar{f}(\mathbf{x}))^2]}_{\text{Varianza}} + \underbrace{E_{\mathbf{x},y}[\big(\bar{f}(\mathbf{x}) - \bar{y}(\mathbf{x})\big)^2]}_{\text{Sesgo}^2} + \underbrace{E_{\mathbf{x},y}[\big(\bar{y}(\mathbf{x}) - y)\big)^2]}_{\text{Ruido}}. \tag{1.13}$$

Por un lado, la varianza es la magnitud en la que cambia la función f estimada si se modifica la muestra de los datos S, siendo así una cantidad que nos ayuda a conocer cuánto de especializado está el modelo a un conjunto de entrenamiento particular (sobreajuste). Para algunos modelos, como pueden ser los $K-$vecinos más cercanos o los árboles de decisión, pequeños cambios en la muestra de datos tienen un impacto notorio en la estructura de f, siendo estos así generalmente métodos con gran varianza.

Por otro lado, se define el sesgo como el error sistemático que se va a cometer debido a las carencias que tiene el algoritmo seleccionado para los datos disponibles. Por ejemplo, en escenarios con datos cuyas relaciones sean complejas y no lineales, modelos poco flexibles como la regresión lineal suelen acarrear un sesgo alto, dadas las fuertes suposiciones que hacen sobre la estructura de los datos.

Por último, el ruido es una característica intrínseca de los datos disponibles, en el cual se engloban problemas en la obtención del propio dato (errores de medición), variables que no estamos teniendo en cuenta y son relevantes para nuestro problema, etc.. Es un error irreducible el cual no podrá ser modelado.

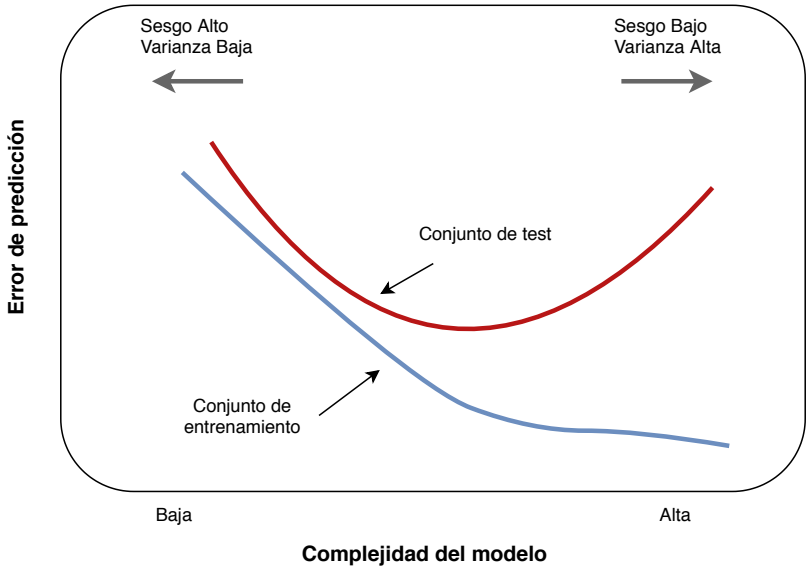

Figura 1.2: Generalización del caso particular de la Figura 1.1, en donde se representa cómo la compensación sesgo-varianza afecta al comportamiento de los errores de entrenamiento y de test a la hora de entrenar un modelo. De nuevo, con complejidad del modelo nos referimos a la flexibilidad que se le permite (es decir, a los grados de libertad, número de parámetros, etc.).

En general, este comportamiento queda representado en la Figura 1.2. Utilizar modelos con demasiada complejidad (i.e., demasiado flexibles) proporciona en la mayoría de los casos modelos con bajo sesgo a expensas de una mayor varianza, la cual se traduce mediante la expresión (1.13) en un elevado error de test esperado (ETE). Por otro lado, la utilización de modelos demasiado simples y poco flexibles conllevará a construir hipótesis con mucha menor varianza, pero con un elevado sesgo derivado de las carencias del modelo seleccionado, dando lugar de igual manera a un aumento del ETE. Así, la clave para dar con un modelo con una buena capacidad de generalización pasa por encontrar un equilibrio entre esas dos cantidades.

Regularización

Una de las técnicas más utilizadas para llegar a un buen equilibrio sesgo-varianza es la conocida como regularización. Esta se basa en utilizar determinadas restricciones sobre el modelo a entrenar para evitar que este llegue a la fase de sobreajuste. Es decir, dada una estructura supuesta para la hipótesis f (por ejemplo, un modelo polinómico), la regularización consiste en restringir la complejidad que se le permite a f a través de añadir un nuevo término a la función de pérdida (1.3) que se quiere

minimizar:

$$\min_f \sum_{i=1}^{N} \mathbf{L}(y_i, f(\mathbf{x}_i)) + \lambda R(f) \tag{1.14}$$

donde L es la función de pérdida, λ es el parámetro que regula cuanto influirá la regularización, y $R(f)$ el término de regularización en sí. Uno de los ejemplos más utilizados en la práctica es la **regularización de Tikhonov**. Suponiendo que nuestra función de aprendizaje f tiene una forma lineal caracterizada por un vector β:

$$f(\mathbf{x}) = \beta \cdot x$$

la regularización de Tikhonov sería la norma L_2 (equivalente a la conocida como regresión *ridge*):

$$\min_\beta \sum_{i=1}^{N} V(\beta \cdot \mathbf{x}_i) + \lambda \|f\|^2.$$

En la práctica, el parámetro de regularización λ será escogido por validación cruzada. Se entrenará el modelo para distintos valores de λ, estimando para cada uno de ellos el error de test esperado, escogiendo así el que lo minimice.

Clasificación

De nuevo, los conceptos definidos en esta sección son fácilmente adaptables al caso de tener una variable respuesta categórica. Por un lado, se puede definir el riesgo empírico (o **error de clasificación medio de entrenamiento**) para el caso de problemas de clasificación. Aunque se pueden usar numerosas funciones de pérdida, la función indicador I, la cual vale 0 si el punto se clasifica correctamente y 1 en otro caso, $\mathbf{L}(y, f(\mathbf{x}))) = I(y \neq f(\mathbf{x}))$, es bastante popular:

$$RE = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq \hat{y}_i) \tag{1.15}$$

donde $\hat{y}_i$ es la clase predicha para la observación i usando nuestra hipótesis f, e I representa la función indicador. Así, (1.15) se puede ver como la proporción de $\mathbf{x}_i$ que fueron clasificados incorrectamente. También es habitual usar otras funciones de pérdida más sofisticadas, que, por ejemplo, permiten asociar distintos costes dependiendo de que clase estamos clasificando erróneamente.

Análogamente al caso de regresión se puede definir el **error de test esperado** de un modelo de clasificación f entrenado en un conjunto de entrenamiento S. Este será el error medio que se cometerá al clasificar un nuevo punto de test $\mathbf{x}_0$ generado por $\mathbb{P}(X, Y)$, cuya clase correcta es y_0 y la predicha $\hat{y}_0$:

$$ETE_S = E_{\mathbb{P}}[I(y_0 \neq \hat{y}_0)] \tag{1.16}$$

en la que de nuevo usamos la función identidad I como función de pérdida.

A continuación, en la figura 1.3 se puede observar un problema de clasificación donde interesa predecir la etiqueta de cada punto (rojo o azul) dado su vector de características $x \in \mathbb{R}^2$ (donde se escoge $\mathbb{R}^2$ como espacio de características para facilitar la visualización del problema). Las líneas negras representan la frontera de decisión obtenida al entrenar sobre el mismo conjunto de datos tres modelos estadísticos para clasificación con distintas compensaciones sesgo-varianza, los cuales darán lugar a distintas cantidades de error de entrenamiento y error de test (aproximado por validación cruzada).

La motivación de esta imagen no es cuantificar ni analizar específicamente ningún algoritmo, si no visualizar gráficamente las situaciones más generales que se pueden dar cuando se entrena un modelo de clasificación: sobre-ajuste, falta de ajuste, y ajuste correcto.

En la figura de la izquierda se muestra la frontera de decisión de un modelo de clasificación lineal, cuya estructura demasiado rígida da lugar a una falta de ajuste (ocasionada por el sesgo del modelo), teniendo así un error de clasificación alto (tanto error de entrenamiento, como se puede ver en la imagen, como error de test si se aproximase por validación cruzada). En la figura de la derecha se puede onservar la situación opuesta, en donde el uso de un modelo de clasificación de 1 vecino más cercano (demasiado flexible, con sesgo nulo pero con una gran varianza), da lugar a un obvio sobreajuste de los datos. En este caso, por definición del los $k-$vecinos más cercanos cuando $k = 1$, el error de entrenamiento siempre será 0. No obstante, el error de test del mismo, aproximado por validación cruzada, será muy elevado, ya que la gran varianza del método dará lugar a fronteras de decisión muy distintas para cada subdivisión del conjunto de entrenamiento, 'memorizando' todos los puntos de entrenamiento en vez de encontrar patrones en los mismos, teniendo un mal rendimiento a la hora de predecir datos de test no vistos anteriormente. Finalmente, la figura del medio refleja un modelo con un ajuste correcto, con un buen equilibrio entre sesgo y varianza, obteniéndose así una buena capacidad de generalización.

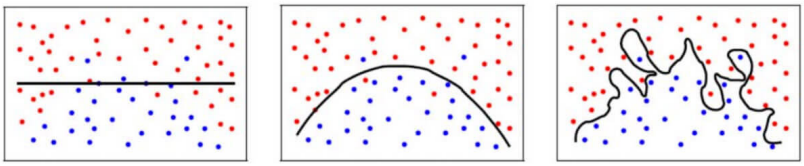

Figura 1.3: Representación de los conceptos explicados anteriormente mediante la Figura 1.1, ahora para el caso de un problema de clasificación binaria en dos dimensiones.

1.3. Modelos base relevantes para construir ensamblados

Volviendo al problema original descrito en la Definición 0.1, existen en la literatura numerosas técnicas para construir hipótesis f con una buena capacidad de generalización. En esta sección se presentarán varias de ellas, centrándose en las que son más utilizadas para la posterior construcción de ensamblados de modelos. Si bien no se entrará a demasiado detalle, se puede consultar Hastie et al. (2008) para un riguroso y extenso estudio de las mismas.

Además, este análisis se centrará mayoritariamente en el problema de clasificación binario, al ser este en donde el área de investigación de ensamblados estadísticos ha puesto más foco en las últimas décadas.

Supóngase, para todas las técnicas a continuación, que se tiene un conjunto de entrenamiento S con N pares de datos $(\mathbf{x}, y)$. Además, por conveniencia para las explicaciones y visualizaciones en esta sección, supongamos que estamos en el caso más sencillo de un problema binario en dos dimensiones. Es decir, $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2) \in \mathbb{R}^2$ e $Y \in \{0, 1\} = \mathcal{G}$, donde convenientemente se codificó una de las posibles clases de y como 0, y otra como 1. El objetivo será crear una regla de clasificación $f(\mathbf{x})$ que permita clasificar nuevos puntos $\mathbf{x}_0$ en su clase correcta. Destacar que mientras algunas técnicas intentan modelar la probabilidad de cada clase Y condicionada al punto X, i.e. $\mathbb{P}(Y = j|X = x)$, creando posteriormente una regla de clasificación a partir de ella, otras directamente intentan predecir la clase correcta sin antes modelar dicha probabilidad.

$k-$vecinos más cercanos ($k-$NN)

Los $k-$vecinos más cercanos es una popular técnica de aprendizaje estadístico no paramétrica introducida en el 1951 por Fix y Hodges. Dado un nuevo dato a clasificar $\mathbf{x}$, se le asigna la clase que predomina en los k puntos $\mathbf{x}_i$ del entrenamiento más cercanos a x (donde por cercanía se entiende cualquier distancia, usualmente la euclídea). Así, se puede definir matemáticamente la hipótesis que genera está técnica como sigue:

$$h(\mathbf{x}) = \frac{1}{k} \sum_{\mathbf{x}_i \in N_k(\mathbf{x})} y_i \tag{1.17}$$

en donde $N_k(\mathbf{x})$ es el conjunto de puntos "vecinos" de $\mathbf{x}$ (los más cercanos), y se está calculando la media de las respuestas y_i en un entorno local de $\mathbf{x}$, es decir, la proporción de puntos con clase $y_i = 1$ cerca de $\mathbf{x}$.

Una forma natural de definir una regla de clasificación a partir de la hipótesis h, en caso de que estemos ante un problema binario equilibrado, sería:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{si } h(\mathbf{x}) > 0.5 \\ 0 & \text{si } h(\mathbf{x}) \leqslant 0.5 \end{cases} \tag{1.18}$$

mientras que en el caso de un problema desbalanceado se suelen usar técnicas de aprendizaje basado en costes para modificar los umbrales de decisión dependiendo del riesgo asociado a cometer cada posible error.

Así, dado que con (1.17) se está intentando aproximar la probabilidad condicional de cada clase, $\mathbb{P}(y = 1|X = x)$ y $\mathbb{P}(y = 0|X = x)$, se puede considerar la regla de clasificación $k-$NN como una aproximación de la regla de Bayes descrita en (1.4).

Destacar que los $k-$vecinos más cercanos es una de las técnicas existentes con más flexibilidad para adaptarse a cualquier relación entre los datos, y eso puede llevar a situaciones de sobreajuste. Para evitarlas, lo que se hace es entrenar el modelo para varios valores de k, aproximando para todos ellos el error de test esperado, y escogiendo el que tenga un menor valor.

En la siguiente imagen se puede ver este fenómeno gráficamente. Para el escenario definido al principio de la sección, Hastie et al. (2001) simulan 100 datos $\mathbf{x}$ de la clase 1 (naranja) y 100 datos de la clase 0 (azul), ambas normales bidimensionales con medias distintas e igual matriz de covarianzas. En la gráfica izquierda se puede ver la frontera de decisión del clasificador $k-$NN con $k = 15$. Y en la de la derecha se puede observar como según se aumenta el grado de libertad del modelo (el valor $\frac{N}{k}$, donde N es la cardinalidad de la muestra, 200 en este caso), el error del conjunto de entrenamiento baja de forma casi monótona, mientras que el error de test aproximado tiene la forma convexa de "U" estudiada en la sección anterior. Utilizando validación cruzada, $k = 8$ daría lugar al modelo con menos error de test esperado, y por ello el que a priori tendría mayor capacidad de generalización.

Árboles de decisión

Los árboles de decisión son algoritmos de aprendizaje estadístico que tienen como objetivo segmentar el espacio de las variables predictoras ($\mathbb{R}^2$ en nuestro problema particular) en varias regiones más sencillas, denominadas nodos o ramas. Dado un nuevo dato $\mathbf{x}$ a predecir o clasificar, se utilizará la media (o en caso de clasificación, la clase más votada) que tienen los $\mathbf{x}$ del conjunto de entrena-

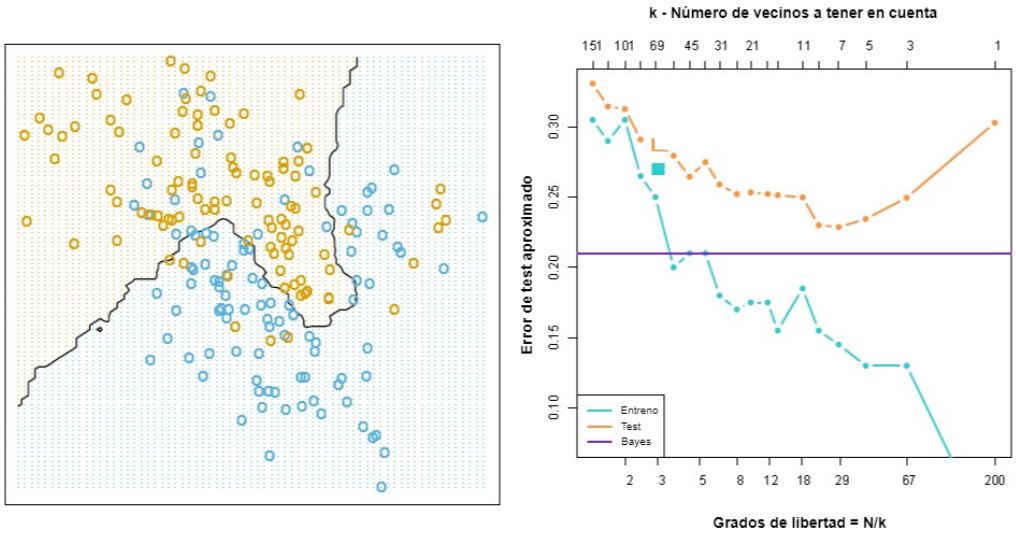

Figura 1.4: Frontera de decisión del modelo 15−NN a la izquierda, y el comportamiento de los $k-$NN según varía su valor de k, de Hastie et al. (2001), a la derecha.

miento de esa región. Como la forma en la que se van creando las reglas de segmentación pueden ser representadas en forma de árbol, estas técnicas llevan el nombre de árboles de decisión.

Los métodos basados en árboles no suelen competir en capacidad de predicción con otras técnicas más sofisticadas. No obstante, tienen la característica de ser muy simples y fáciles de interpretar. Junto con otros motivos, esta es una de las razones por las cuales son tan usados a la hora de crear ensamblados estadísticos.

Dentro de todas las variantes que hay en los árboles de decisión, aquí presentaremos su versión más sencilla. En ella, vamos a dividir el espacio de parámetros ($\mathbb{R}^2$ en nuestro caso, $\mathbb{R}^p$ en general) en M regiones distintas y no solapadas $R_1, ..., R_M$ (también denominadas nodos terminales o ramas del árbol). Una vez creadas esas regiones, para todo nuevo dato a predecir que caiga en la región R_m se le asignará la clase mayoritaria entre los datos de entrenamiento en dicha región. Es decir, si se está por ejemplo en el nodo terminal m (equivalente a la región R_m), se calcularía:

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} I(y_i = k) \tag{1.19}$$

teniendo así la proporción de datos en m para cada clase k. Con eso, se clasificarían todos los datos nuevos que caigan en m en la clase $k(m) = \arg\max_k \hat{p}_{mk}$.

Para construir esas M regiones R_m, que por conveniencia se va a suponer que serán rectángulos (o hiperrectánculos en el caso genérico), se seguirán los siguientes pasos. Dado el conjunto de entrenamiento S, lo que interesará es que los rectángulos $R_1, ..., R_M$ minimicen la función de pérdida $\mathbf{L}$ escogida:

$$\sum_{m=1}^{M} \sum_{i \in R_m} \mathbf{L}(y_i, \hat{y}_{R_m}) \tag{1.20}$$

donde $\hat{y}_{R_m}$ es la respuesta que el árbol predice para los elementos del nodo terminal m.

Mientras que en el caso de regresión la función de pérdida usada habitualmente es la de mínimos

cuadrados, en el caso de clasificación existen opciones más adecuadas para tratar respuestas categóricas. Sea el nodo terminal m representando a la región R_m en la cual hay N_m observaciones, se definen las siguientes tres funciones de pérdida para la región R_m (también conocidas como métricas de impureza de nodo terminal):

$$\begin{aligned} &\text{Error de clasificación} && \frac{1}{N_m}\sum_{\mathbf{x}_i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)} \\ &\text{Índice Gini} && \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}) \\ &\text{Entropía cruzada} && -\sum_{k=1}^{K} \hat{p}_{mk}\log \hat{p}_{mk}. \end{aligned} \tag{1.21}$$

Estas se simplifican considerablemente para nuestro caso particular de problema binario donde $K = 2$. Para este caso en la Figura 1.5 se representa la cantidad del error que cada una de esas tres métricas asigna a la incorrecta clasificación de un punto $\hat{x}$ dependiendo de la proporción de las clases en el nodo terminal. Destacar que, al contrario que el error de clasificación, tanto el índice Gini como la entropía cruzada son funciones diferenciables, lo que las hace más convenientes para ser usadas en la práctica.

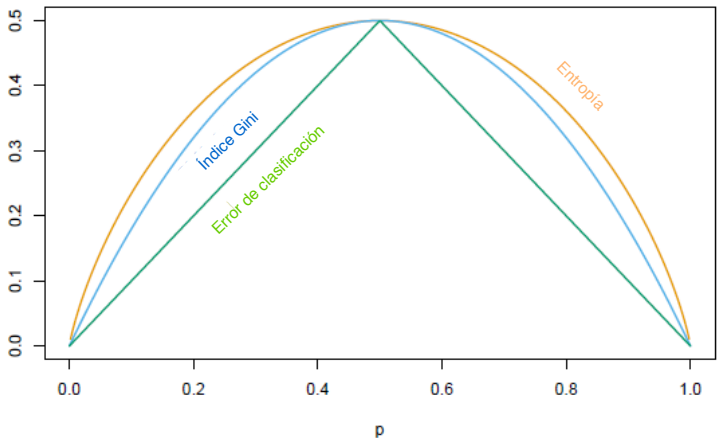

Figura 1.5: Representación del error asignado por tres métricas de impureza de nodo a la hora de clasificar incorrectamente un punto en un problema binario. Dado un nuevo punto a clasificar, las proporciones de las dos clases posibles serán p y $1-p$. En la gráfica se puede ver la impureza que tendría el nodo en función de p.

Continuando con la minimización de (1.20), considerar todas las posibles particiones del espacio de características en M rectángulos es computacionalmente costoso. Por eso, otra de las simplificaciones que se hace es construir los nodos con el método de la **segmentación binaria recursiva** de arriba abajo. Empezando desde la cima del árbol, donde todo el conjunto de entrenamiento pertenece a un único nodo, en cada paso se hace la mejor división binaria posible hasta que cada punto del entrenamiento pertenezca a su propio nodo terminal.

Formalmente, en cada paso se escogerá la variable predictora X_j y el umbral s que dividen el espacio de características en las regiones $\{X|X_j < s\}$ y $\{X|X_j > s\}$, produciendo así el mayor decrecimiento

posible del error total cometido definido en (1.20). Es decir, para cada j y s se dará lugar a las regiones:

$$R_1(j,s) = \{X|X_j < s\} \quad \text{y} \quad R_2(j,s) = \{X|X_j > s\} \tag{1.22}$$

y se buscará el valor j y s que minimice la ecuación:

$$\sum_{i:\mathbf{x}_i \in R_1(j,s)} \mathbf{L}(y_i, \hat{y}_{R_1}) + \sum_{i:\mathbf{x}_i \in R_2(j,s)} \mathbf{L}(y_i, \hat{y}_{R_2}) \tag{1.23}$$

en donde $\hat{y}_{R_1}$ es la clase predicha para los datos en la región $R_1(j,s)$ e $\hat{y}_{R_2}$ para los de $R_2(j,s)$, calculada en cada caso como la clase mayoritaria en la región que corresponda. Encontrar los j y s que resuelvan esta ecuación se puede hacer de manera rápida y eficiente (Hastie et al. 2001).

En el siguiente paso, se escogería una de las dos regiones en las que se divide el espacio de predictores y se volvería a repetir el mismo proceso, y así sucesivamente. El proceso se para cuando se tenga un único dato por nodo terminal m o cuando se llegue a un criterio de parada, como, por ejemplo, que no existan regiones R_m con menos de 5 datos de entrenamiento en ellas.

Este modelo estadístico, tal y como se ha definido, es muy probable que vaya a dar muy buenos resultados en el conjunto de entrenamiento. De hecho, si se construye un árbol sin criterio de parada (e igual que ocurre al construir los $k-$vecinos más cercanos con $k = 1$), el modelo tendrá por definición un 100 % de exactitud de clasificación en el conjunto de entrenamiento. No obstante, ese comportamiento vendrá dado por un gran sobreajuste de los datos, consecuencia de la demasiada flexibilidad del método, y producirá un rendimiento pobre al clasificar nuevos datos de test.

Para evitar este sobreajuste existe la técnica denominada poda de árboles. Esta, partiendo de un árbol grande, va fusionando según qué nodos hasta conseguir un árbol más pequeño y simple que dé lugar a una mayor capacidad de generalización. El tamaño del árbol en sí será un parámetro que gobernará la complejidad del modelo, similar al k en los $k-$vecinos más cercanos. Sea $T_{\text{máx}}$ un árbol grande creado sin criterio de parada. Se define el concepto de subárbol $T \subset T_{\text{máx}}$ como cualquier árbol obtenido de colapsar un número de nodos internos de $T_{\text{máx}}$. Similar a con los $k-$NN, una manera para estimar el tamaño de subárbol óptimo con el que quedarse podría ser estimar el error de clasificación esperado utilizando validación cruzada para varios tamaños de árbol, para posteriormente escoger el mejor. No obstante, hacer dicho cómputo para todas los posibles subárboles $T \subset T_{\text{máx}}$ es inviable en la mayoría de los casos.

Para resolver este problema surge la técnica de **poda coste - complejidad** (*cost complexity pruning* en inglés), que permite generar una secuencia de subárboles de $T_{\text{máx}}$ cumpliendo unas propiedades deseables. Se define el proceso como sigue:

Sea $|T|$ el número de nodos terminales (i.e., regiones) de un árbol T, y sea $N_m = \#\{\mathbf{x}_i \in R_m\}$ el número de datos del conjunto de entrenamiento en el nodo terminal m. A continuación se define el criterio coste-complejidad de un árbol dependiendo del parámetro $\alpha \geq 0$, en el que al cálculo del error cometido (coste), se le añade un término que tiene en cuenta cuánto de largo es el árbol construido (complejidad):

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{i:\mathbf{x}_i \in R_m} \mathbf{L}(y_i, \hat{y}_{R_m}) + \alpha|T|.$$

El objetivo será, para un secuencia finita de $\alpha \geq 0$, conseguir los subárboles $T_\alpha \subseteq T_{\text{máx}}$ que minimizan $C_\alpha(T)$. Se puede consultar Breiman et al. (1984) para la prueba de que dado un α, $C_\alpha(T)$ tiene una única solución, y también que $C_\alpha(T)$ tiene un número de valores distintos finitos para una cantidad finita de valores de α. Los detalles matemáticos de cómo conseguir la secuencia de subárboles óptimos T_α y la secuencia de valores de α se omiten en este trabajo, dado que, por su

coste computacional, los árboles usados en los ensamblados estadísticos suelen dejar los árboles base sin podar.

Para finalizar, destacar que el parámetro α regula el equilibrio entre lo bien que se ajusta el árbol a los datos y su tamaño, siendo así otro ejemplo de las técnicas de **regularización** vistas en la sección 1.2. Una vez se tiene la sucesión de árboles T_α, se puede aproximar para cada uno de ellos el error de test esperado, para finalmente escoger el que lo minimice.

Redes neuronales (NNs)

Las redes neuronales son modelos amplia y exitosamente utilizados en tareas de regresión y clasificación estadística, cuya estructura está inspirada en el funcionamiento de las neuronas del cerebro humano. Desarrolladas a lo largo de la mitad del siglo pasado, no fue hasta los avances de LeCun et al. (1998) y, posteriormente, Krizhevsky et al (2012), que recobraron importancia en una amplia gama de problemas. En concreto, las variantes más sofisticadas de estos métodos, como las redes convolucionares o recurrentes, son consideradas como el estado del arte actual a la hora de resolver problemas de procesamiento de lenguaje natural y clasificación de imágenes, entre otros.

En esta sección se presentarán las redes neuronales en su versión más sencilla, denominadas redes alimentadas hacia adelante (o *feedforward neural nets* en inglés). Estas se pueden representar como el grafo de la Figura 1.6, en donde habrá una primera capa por donde entrarán los datos, una o más capas intermedias con M neuronas donde estos sufrirán transformaciones (lineales y no lineales), y una capa de salida donde habrá tantas neuronas como variables respuesta.

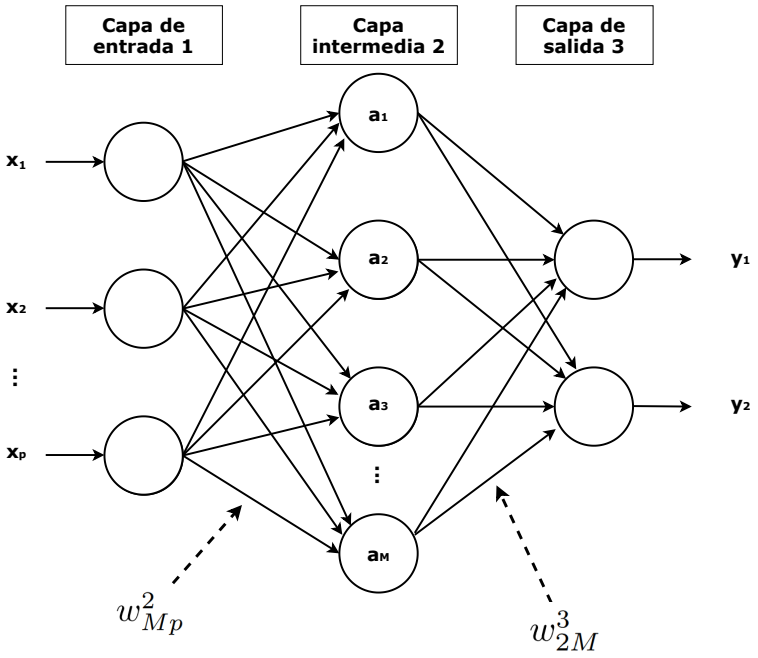

Figura 1.6: Esquema y elementos de una red neuronal totalmente conectada.

Matemáticamente, los distintos elementos y transformaciones que identifican la red neuronal de la Figura 1.6 son los siguientes. Dado un par $(x, y) \in S$ del problema **binario** propuesto a resolver, con $\mathbf{x} \in \mathbb{R}^p$ y $y \in \{-1, 1\}$, cada $\mathbf{x}$ entraría en la capa de entrada donde hay una neurona por cada una de sus p variables. Conectadas a esas neuronas de entrada, cada una identificada por un peso w, se encuentran las M neuronas de a_j. Se denomina a w^l_{jk} como el peso que va desde la neurona k de la

capa $l-1$ hasta la neurona j de la capa l. Se calcularía, de manera general, cada a_j^l de la capa l y neurona j como la siguiente combinación lineal de $\mathbf{x}$:

$$a_j^l = \sigma\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \tag{1.24}$$

donde σ se conoce como la función de activación, que intenta normalizar de alguna forma el resultado intermedio obtenido en la expresión anterior. Si bien existen distintas funciones de activación para distintos propósitos, una de las más habituales es la función sigmoide, $\sigma(z) = \frac{1}{1+\exp(-z)}$. En nuestro caso particular, no necesitaremos calcular a recursivamente como en 1.24 dado que solo se tiene una capa intermedia de activación, quedando la expresión:

$$a_j^2 = \sigma\left(\sum_{k=1}^{p} w_{jk}^2 \mathbf{x}_k + b_j^2\right)$$

en donde b sería el término del sesgo de la capa $l = 2$.

Una vez se tiene calculada la primera transformación, se podría repetir de nuevo el mismo paso para sacar las variables de salida. Si bien eso es lo que se suele hacer en un escenario de regresión utilizando una única neurona en la capa de salida, en nuestro caso lo que nos interesará será que la salida de esta red nos dé en el vector (y_1, y_2) con las probabilidades de que el dato $\mathbf{x}$ pertenezca a la clase 0 o 1 respectivamente (aunque, en el caso de la clasificación binaria, una única neurona de salida podría codificar toda esa informacion, siendo su cantidad p la probabilidad de una de las clases, y $1-p$ la de la otra). Para ello, el cálculo final será:

$$y_j = g_j\left(\sum_{k=1}^{M} w_{jk}^3 a_k^2 + b_j^3\right), j = 1, 2$$

donde la función de transformación g en el caso de clasificación suele ser $g_j(z) = \frac{\exp(z_j)}{\sum_{l=1}^{2}\exp(z_l)}$, que convenientemente hace que la suma $y_1 + y_2$ sea 1 para ser interpretado como probabilidades de pertenecer a cada una de las dos clases.

Así, se pueden ver las redes neuronales como una extensión no lineal de la regresión multivariante en donde, a través de numerosas transformaciones no lineales, se le permite al modelo aprender relaciones más complejas entre los datos.

Para entrenar este tipo de modelos se utiliza una técnica conocida como **propagación hacia atrás**. Escogida la función de coste que se usará para calcular cuánto error se comete con un conjunto de pesos w_{jk}^l dado, esta será extremadamente no convexa y no lineal, problema el cual no va a tener usualmente una solución explícita. Por lo tanto, y empezando con un conjunto de pesos aleatorios, lo que se intentará es aproximar el gradiente de dicha función para posteriormente utilizar el método iterativo del descenso del gradiente para ir actualizando los pesos w_{jk}^l según los errores cometidos en el conjunto de entrenamiento S.

Por último, escoger una arquitectura concreta de red para resolver un problema (número de capas intermedias, neuronas de cada una de ellas, etc.) es más un arte que una ciencia, y no existe una norma que funcione de manera universal en todas las situaciones.

Capítulo 2

Técnicas de ensamblado estadístico

El término ensamblado estadístico (también denominado en la literatura como fusión o combinación estadística) se refiere al conjunto de metodologías que permiten combinar varios modelos estadísticos para dar lugar a un meta-algoritmo que mejore los resultados de los modelos individuales que lo forman. Desde las técnicas más sencillas, como la votación de comité vista en la sección anterior, hasta las más sofisticadas, como bagging o boosting, esta área de investigación es una de las más populares de la última década, principalmente por los exitosos resultados empíricos cosechados en una amplia gama de campos científicos.

Aunque este ámbito es de rigurosa actualidad, cabe destacar que los primeros trabajos en esta temática fueron propuestos hace más de 50 años. Probablemente uno de los primeros fue el artículo de Dasarathy y Sheela (1979), en donde proponían un sistema de ensamblado del tipo "divide y vencerás", en el cual se divide el espacio de características $\mathbb{R}^p$ en distintas regiones, utilizando para clasificar un nuevo dato el modelo que mejor se comporte en su región correspondiente. Años después, Hansen y Salamon (1990) probarían la capacidad de reducir la varianza de los ensamblados, y la mejor generalización que una combinación de redes neuronales con distintas configuraciones tienen frente a una única. No obstante, fue Shapire R (1990) el que llevó a la popularidad esta área de la estadística al probar que se podía construir un clasificador fuerte (en el sentido del aprendizaje PAC) a partir de la combinación de varios clasificadores débiles (o clasificadores base) mediante un procedimiento que acuñó como ***boosting***.

En esta sección se hará un repaso de los hechos más relevantes de esta área: las razones de su existencia, su taxonomía, varios ejemplos de algoritmos concretos, y los resultados teóricos conocidos. En la mayoría de los casos se tratará únicamente el problema de clasificación, al ser este el más estudiado en la bibliografía consultada.

2.1. Motivación

Antes de indagar en los detalles de estas metodologías, será de utilidad entender las razones que motivaron a esta área de investigación a hacer uso de estas técnicas. Algunas de estas, de manera informal, se pueden encontrar en nuestras propias experiencias personales: desde el funcionamiento democrático de nuestro sistema político hasta las decisiones que tomamos en nuestras vidas a partir de consultar distintas fuentes de información, combinar o ensamblar distintas opiniones es algo que hacemos de manera natural en nuestro día a día. Además, existen también razones prácticas, motivadas por el gran éxito de algunas de estas técnicas en distintos campos científicos: finanzas (Leigh et al. 2002), astronomía y astrofísica (Bazell y Aha 2001), medicina (Polikar et al. 2008), y un largo etcétera.

Por otro lado, y de una manera más rigurosa, la mayoría de autores de la literatura coinciden que existen tres razones principales por las cuales combinar distintos modelos en un ensamblado puede ser beneficioso. Estas, descritas a continuación, aparecen representadas gráficamente en la Figura 2.1.

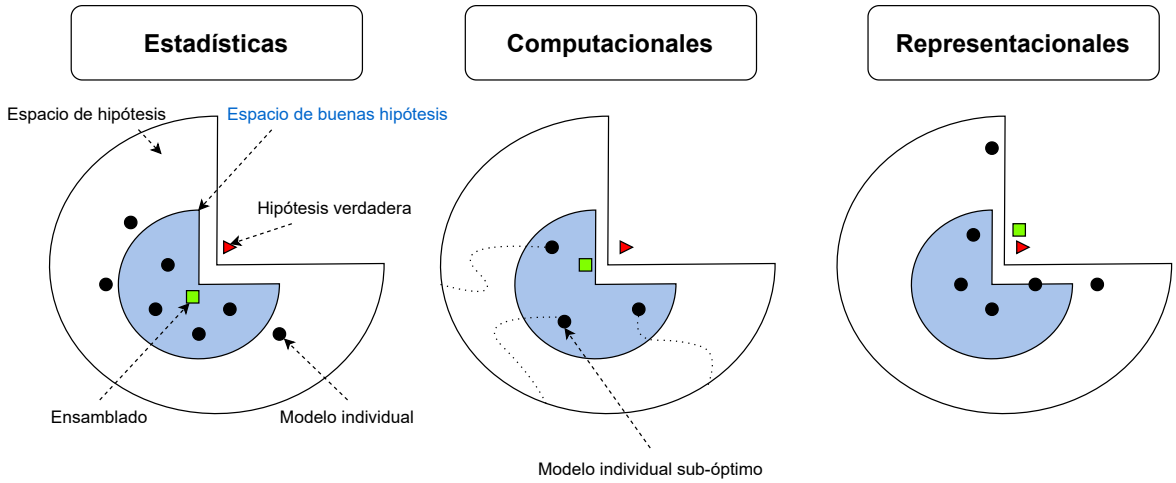

Figura 2.1: Representación gráfica de las tres razones principales por las cuales un ensamblado puede dar buenos resultados.

1. **Razones estadísticas**

   Cualquier algoritmo de aprendizaje estadístico se puede ver como la búsqueda del mejor modelo para nuestros datos en un espacio de hipótesis. Sea cual sea el procedimiento para escoger un modelo y sus parámetros (validación cruzada, etc.) siempre existirá una incertidumbre asociada con dicho proceso debido a que se dispone únicamente de un conjunto de entrenamiento S finito. Si la cantidad de datos es reducida, puede darse la situación en la cual se tengan muchos modelos en el espacio de hipótesis con un rendimiento parecido en el conjunto de test (representados como los puntos negros dentro del espacio de buenas hipótesis – área azul – en la Figura 2.1 a la izquierda). Mediante la creación de un ensamblado (representado por el cuadrado verde), se puede reducir el riesgo de escoger una hipótesis incorrecta a través de hacer una combinación de todos los modelos buenos disponibles.

2. **Razones computacionales**

   Algunos algoritmos, como las redes neuronales o los árboles de decisión, pueden dar lugar a minimizar funciones de coste altamente no convexas, pudiendo quedar los métodos utilizados para resolverlas en un óptimo local. Ensamblar distintas hipótesis, utilizando en cada una de ellas puntos de partida distintos para esa búsqueda local, puede aumentar la probabilidad de aproximar mejor la hipótesis verdadera (representada por el triángulo rojo). Otra razón es cuando el tamaño del conjunto de entrenamiento S es reducido. En estos casos, se pueden hacer remuestreos *bootstrap* de dicho conjunto, entrenando en cada uno de los remuestreos cada modelo, y combinarlos todos al final. En otros casos donde tenemos una cantidad demasiado grande de datos, una solución puede ser dividirlos en N conjuntos, entrenar un modelo en cada uno de ellos, fusionándolos después para dar lugar al modelo final.

3. **Razones representacionales**

   En algunos casos, la función verdadera que se quiere aproximar $f*$ no puede ser representada por ninguno de los modelos de nuestro espacio de hipótesis. Como se puede observar en la Figura 2.1 de la derecha, a partir de hacer combinaciones de distintos métodos que sí están en nuestro espacio de hipótesis disponible, se puede extender el mismo para aproximar mejor la hipótesis verdadera. Por ejemplo, un ensamblado de clasificadores lineales puede dar lugar a fronteras de decisión tan complejas como se quiera, dando así la posibilidad de resolver un problema no lineal no factible en caso de usar un único clasificador lineal.

2.2. Terminología y taxonomía

A lo largo de la literatura de los ensamblados estadísticos se utilizan numerosos términos para referirse a conceptos análogos. A continuación se presenta la terminología más común, comentando sus variantes más utilizadas. Sea S el conjunto de entrenamiento formado por N pares de puntos $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ (también denominados ejemplos, observaciones o vectores de características). Se define como característica o variable a cada una de las p componentes de $\mathbf{x}_i \in \mathbb{R}^p$. Cada uno de los L modelos construidos para ensamblar se denominarán clasificadores base, hipótesis, o aprendices débiles. Por último, se denomina a la combinación de los clasificadores base como ensamblado, comité, o meta-modelo.

A pesar de que el concepto de combinar modelos parezca intuitivo, las múltiples maneras de hacerlo hace que no exista a día de hoy una definición rigurosa de qué elementos definen exactamente un ensamblado estadístico. La cantidad de niveles sobre los cuales se puede caracterizar un ensamblado estadístico es muy amplia: la manera en la que se utilizan los datos de entrada; la utilización o no de todas las variables disponibles; los criterios para seleccionar los modelos o clasificadores base, o las distintas formas de combinar los resultados obtenidos, entre otras.

De este amplio abanico de posibilidades nacen las numerosas técnicas que hoy en día existen en la literatura. Con el afán de taxonomizar las mismas, y en base a la organización de Kuncheva (2014) representada en la Figura 1, se definen los cuatro niveles principales sobre los cuales se puede definir un ensamblado estadístico:

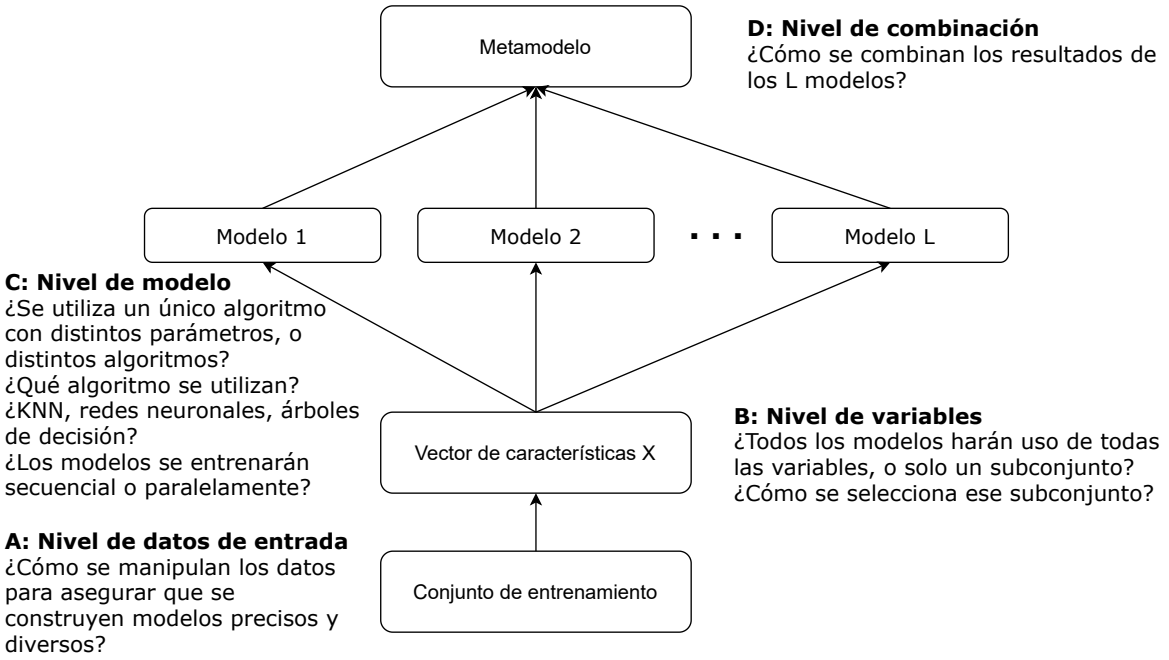

Figura 2.2: Representación gráfica de los distintos niveles sobre los cuales se pueden caracterizar a un ensamblado de modelos, basado en Kuncheva (2014).

- **Nivel de datos de entrada**

  Dado el conjunto de entrenamiento S, un ensamblado puede entrenar cada uno de sus L modelos base en el mismo conjunto S o asignarle a cada uno de ellos una perturbación del mismo denominada S^*, pudiendo ser esta un subconjunto de S, un remuestreo **bootstrap**, etc. Paralelamente, también se distingue entre los métodos que permiten entrenar los modelos base al mismo tiempo, y los que, por la naturaleza secuencial de las perturbaciones de S, no lo permiten.

  En varios de los métodos que se verán a continuación, como en bagging, cada uno de los L modelos se entrena en un conjunto S^* distinto, para crear así diversidad entre los clasificadores.

- **Nivel de variables**

  Con respecto a las distintas p variables de cada $\mathbf{x}_i \in S$, cada clasificador base puede ser entrenado utilizando todo el conjunto de variables o con solo un subconjunto del mismo.

  En algunas técnicas, como los métodos de subespacios aleatorios, cada clasificador base se entrena en un subconjunto aleatorio de variables para dar lugar a modelos menos correlados, aumentando así la diversidad.

- **Nivel de modelo**

  Dada la amplia gama de modelos de aprendizaje estadístico que se pueden utilizar como clasificadores base, las posibilidades a la hora de combinarlos son infinitas. Sin embargo, se pueden definir dos estrategias principales.

  En una de ellas, se crea un ensamblado utilizando el mismo modelo L veces, variando entre ellos únicamente los parámetros. Por ejemplo, utilizando L modelos $k-$NN modificando el parámetro k o L árboles de decisión con distintos criterios de parada y distintos niveles de podaje. En la otra, se ensamblan L métodos distintos; un $k-$NN, un modelo lineal, una red neuronal, etc.

- **Nivel de combinación**

  Una vez se dispone del resultado de los L modelos base, las posibilidades para combinarlos se dividen en dos grandes tipos: ensamblados entrenables y no entrenables.

  Dentro de los no entrenables están las técnicas que utilizan directamente la salida de los L modelos para obtener la clasificación final, por ejemplo, la votación comité sin pesos. Los entrenables, sin embargo, son técnicas de ensamblado que requieren de la construcción de un meta-modelo, utilizando las salidas de los modelos base para dotar de información extra al conjunto de entrenamiento S.

  Además, también se puede distinguir entre los métodos de combinación, que necesitan simplemente la clase predicha por los L clasificadores base, y los que necesitan a mayores el soporte que cada modelo base L le asigna a cada clase, i.e., la estimación que cada modelo hace de la probabilidad $\mathbb{P}(Y = g|X = \mathbf{x})$ con $g \in \mathcal{G}$.

Paralelamente a esta taxonomía, y centrándose en la estrategia general de la técnica, se pueden definir dos tipos principales de ensamblados: de fusión y de selección. En la fusión de clasificadores se supone que todos los L modelos base construidos tienen información de todo el espacio de características, y por lo tanto son todos utilizados a la hora de predecir la clase de un nuevo punto $\mathbf{x}$. Este es el tipo de técnicas más estudiado a día de hoy, y en el que caen la mayoría de métodos que se estudiarán en la siguiente sección, como el bagging y el boosting. En la selección de clasificadores se supone que cada modelo base solo tiene información valiosa de un subconjunto del espacio de parámetros, y, por lo tanto, a la hora de clasificar un nuevo punto solo serán utilizados los modelos base que tienen información en dicha región.

2.3. Ejemplos de técnicas de ensamblado

A lo largo de esta sección, se denotará S al conjunto de entrenamiento formado por N pares de datos (X, Y), donde $X \in \mathbb{R}^p$ e $Y \in \mathcal{G}$, siendo $\mathcal{G}$ el conjunto de posibles clases de Y con cardinalidad $card(\mathcal{G}) = K$. Se utilizará $\mathcal{G}_k$ para referirse a la k-ésima clase de $\mathcal{G}$, y g para referirse a una clase arbitraria de $\mathcal{G}$. Se denotarán h_l a cada uno de los L clasificadores base creados y $\mathbf{h} = [h_1(\mathbf{x}), ..., h_L(\mathbf{x})] \in \mathcal{G}^L$ el vector de salidas de los L clasificadores base. Se define h como el ensamblado construido a partir de los L modelos base.

2.3.1. Promedio de modelos Bayesiano

A la hora de resolver un problema de clasificación, una práctica habitual en la ciencia de datos es escoger una familia de modelos h_θ para, a partir de ajustar los datos al conjunto de entrenamiento S, encontrar el $\hat{\theta}$ óptimo que maximiza algún tipo de criterio de ajuste o precisión de clasificación (minimizar una función de coste, criterio de máxima verosimilitud...). Una vez conseguido el modelo óptimo h_0, y validado que sus parámetros son significativos, éste estaría a priori listo para utilizarse en producción para realizar predicciones.

No obstante, como se mencionó anteriormente, es posible que exista otro modelo h_1 (que puede o no ser de la misma familia h_θ) que también produzca un buen ajuste de los datos, aunque dando lugar a distintas predicciones. La posibilidad de existencia de estos modelos similares h_l es una fuente de incertidumbre, y la solución de escoger un único modelo "óptimo" la estaría ignorando, pudiendo así subestimarla.

El promedio de modelos Bayesiano intenta arreglar este problema utilizando para la predicción no solo el mejor modelo, sino un promedio de modelos, a priori, buenos candidatos. Dado un problema de clasificación, sea $\hat{g}$ la clase de un nuevo punto $\mathbf{x}$ que se quiere predecir. Por la ley de probabilidades totales, se puede escribir la probabilidad a posteriori de $\hat{g}$ condicionada a S como:

$$\mathbb{P}(\hat{g}|S) = \sum_{l=1}^{L} \mathbb{P}(\hat{g}|h_l, S)\mathbb{P}(h_l|S). \tag{2.1}$$

Haciendo la esperanza de (2.1) se tiene:

$$E\left[\hat{g}|S\right] = \sum_{l=1}^{L} E\left[\hat{g}|h_l, S\right]\mathbb{P}(h_l|S) \tag{2.2}$$

que indica que la predicción Bayesiana de la clase $\hat{g}$ viene dada por el promedio de las predicciones de los L modelos, ponderado por la probabilidad a posteriori $\mathbb{P}(h_l|S)$ de cada uno de ellos, que será mayor cuanto mejor sea el ajuste de h_l a los datos.

En la práctica, sin embargo, computar la expresión (2.2) es una tarea con numerosas complicaciones: dar con una forma de escoger una familia finita de modelos candidatos h_l; las integrales a las que dan lugar las probabilidades posteriores $\mathbb{P}(h_l|S)$; o la selección que en estas tenemos que hacer de las distribuciones previas $\mathbb{P}(h_l)$, entre otras. Se puede consultar Hoeting et al (1999) para más detalles.

Aunque en la práctica no se suele utilizar, la formulación (2.2) da lugar a populares técnicas de ensamblados estadísticos, algunas de las cuales se revisarán a continuación.

2.3.2. Votación de la mayoría del comité

La votación de comité es una técnica de ensamblado estadístico que actúa principalmente a nivel de combinación. Es mayoritariamente utilizada cuando los clasificadores base entrenados dan como salida directamente la clase predicha, sin estimar la probabilidad de cada una de las clases, $\mathbb{P}(Y = j|X = x)$.

Sean $h_l,\ l = 1, \ldots, L$ los clasificadores base entrenados, y sea $h_l(\mathbf{x}),\ l = 1, \ldots, L$ la clase que cada uno de ellos le asignan a un nuevo punto a clasificar $\mathbf{x}$. El ensamblado creado por votación de comité asignaría a $\mathbf{x}$ la clase mayoritaria entre los modelos base h_l, es decir:

$$h(\mathbf{x}) = \arg\max_{g \in \mathcal{G}} \sum_{l=1}^{L} I(h_l(\mathbf{x}) = g). \tag{2.3}$$

Como se mostró en la introducción, dadas las suposiciones de que todos los clasificadores base son precisos (ratio de acierto de clasificacion $p > 0.5$) y diversos, tenemos que el ensamblado (2.3) mejora la exactitud de los modelos individuales que lo conforman.

En concreto, en este contexto se entiende como modelos diversos aquellos que dan predicciones de manera independiente condicionados a la clase verdadera. Es decir, que para toda clase $\mathcal{G}_k \in \mathcal{G}$,

$$\mathbb{P}(h_1(\mathbf{x}), \ldots, h_L(\mathbf{x})|\mathcal{G}_k \text{ clase verdadera}) = \mathbb{P}(h_1(\mathbf{x})|\mathcal{G}_k)\mathbb{P}(h_2(\mathbf{x})|\mathcal{G}_k) \ldots \mathbb{P}(h_L(\mathbf{x})|\mathcal{G}_k). \tag{2.4}$$

En este caso, si se tiene un problema binario ($card(\mathcal{G}) = K = 2$), el ratio de acierto del ensamblado será:

$$p_{\text{ens}} = \sum_{m=\lfloor L/2 \rfloor + 1}^{L} \binom{L}{m} p^m (1-p)^{L-m} \tag{2.5}$$

donde el operador $\lfloor z \rfloor$ es el menor número entero más cercano a z. A partir de esta expresión se pueden derivar los siguientes resultados:

1. Si $p > 0.5$ se tiene que p_{ens} es monótona creciente conforme $L \to \infty$, tendiendo $p_{\text{ens}} \to 1$.
2. Si $p < 0.5$ se tiene que p_{ens} es monótona decreciente conforme $L \to \infty$, y $p_{\text{ens}} \to 0$.
3. Si $p = 0.5$ se tiene que $p_{\text{ens}} = 0.5$ para cualquier número de clasificadores base L.

Esto confirma la intuición de que un ensamblado cumpliendo estas condiciones mejorará los resultados de los modelos individuales si es lo suficientemente preciso, siendo contraproducente de otro modo.

Dado que la expresión (2.5) trata el caso particular de un problema binario, se supone que $\lfloor L/2 \rfloor + 1$ clasificadores tienen que votar una clase para salir escogida. Si bien esa condición es necesaria y suficiente con $K = 2$, esta es suficiente pero no necesaria cuando $K > 2$. Por esa razón, la precisión de un ensamblado usando la regla de votación de mayoría cuando $K > 2$ puede ser incluso mayor. Se puede probar, además, que la regla de votación por mayoría es el método de ensamblado óptimo para combinar los L clasificadores base cuando la precisión de clasificación p es igual para todos ellos (Kuncheva 2014).

Como la mayoría de métodos de ensamblado pretenden crear diversidad entre los modelos base, la suposición de que todos ellos vayan a tener la misma precisión de clasificación casi nunca se va a dar en la práctica. Así, otra de las técnicas de combinación ampliamente utilizada es la votación por mayoría con pesos. Intuitivamente, esta se basa en darle más o menos poder a los votos de un clasificador base h_l según sea su poder de predicción individual p_l.

Extendiendo la expresión (2.3) al caso con pesos, el ensamblado creado tendría la siguiente forma:

$$h(\mathbf{x}) = \arg\max_{g\in\mathcal{G}} \sum_{l=1}^{L} w_l I(h_l(\mathbf{x}) = g) \tag{2.6}$$

donde los w_l serían los pesos asociados a cada uno de los clasificadores h_l, y cuyos valores óptimos que maximizan la precisión del ensamblado serían:

$$w_l = \log\left(\frac{p_l}{1-p_l}\right),\ 0 < p_l < 1 \tag{2.7}$$

siendo p_l el ratio de acierto de clasificación del modelo h_l (se pueden consultar los detalles de la prueba en Pierce 1961).

2.3.3. Combinación ingenua Bayesiana (*Naive Bayes ensemble*)

Dado el vector de salidas de los clasificadores $\mathbf{h} = [h_1(\mathbf{x}), ..., h_L(\mathbf{x})] \in \mathcal{G}^L$, sería interesante conocer la probabilidad que tiene cada clase $\mathcal{G}_k$ de ser la correcta, condicionado al vector $\mathbf{h}$. Es decir:

$$\mathbb{P}(\mathcal{G}_k \text{ es la clase verdadera } |\mathbf{h}),\ k = 1, \ldots, K.$$

Dada la suposición de independencia definida en (2.4) y el teorema de Bayes, se puede reescribir esta expresión como:

$$\mathbb{P}(\mathcal{G}_k|\mathbf{h}) = \frac{\mathbb{P}(\mathcal{G}_k)}{\mathbb{P}(\mathbf{h})} \prod_{l=1}^{L} \mathbb{P}(h_l(\mathbf{x})|\mathcal{G}_k),\ k = 1, \ldots, K.$$

A partir de esta, la regla de Bayes ingenua se basa directamente en escoger la clase $\mathcal{G}_k$ para la cual la probabilidad $\mathbb{P}(\mathcal{G}_k|\mathbf{h})$ sea mayor. Es decir:

$$\arg\max_{\mathcal{G}_k\in\mathcal{G}} \left\{ \mathbb{P}(\mathcal{G}_k) \prod_{l=1}^{L} \mathbb{P}(h_l(\mathbf{x})|\mathcal{G}_k) \right\}$$

en donde se elimina $\mathbb{P}(\mathbf{h})$ ya que no influye al ser constante para todas las clases.

Se puede probar que la combinación Bayesiana ingenua es, bajo las suposiciones antes mencionadas, la regla óptima de ensamblaje (consultar Kuncheva 2014). La técnica lleva el nombre "ingenuo" por hacer la suposición (2.4) de que todos los clasificadores base asignan entre ellos etiquetas de manera independiente a un punto $\mathbf{x}$ si se condiciona a su clase verdadera, y "Bayesiana" por hacer uso del teorema de Bayes para su derivación.

2.3.4. Bagging

Introducido por Leo Breiman en 1994, bagging (o ***b**ootstrap **agg**regation*) es una de las técnicas más populares de ensamblado estadístico. Actuando principalmente a nivel de datos de entrada, bagging hace uso de remuestreos con reemplazamiento bootstrap para dar lugar a perturbaciones del conjunto de entrenamiento S, entrenando los clasificadores base en cada una de ellas para obtener así a modelos más diversos.

Dado un conjunto de observaciones $Z_1, \ldots, Z_N$ independientes e identicamente distribuidas, cada una de ellas con una varianza σ^2, se tiene que la varianza de la media $\bar{Z}$ es σ^2/N. Es decir, hacer

la media de un conjunto de observaciones independientes conlleva a reducir la varianza. Basándose en esta propiedad, dado un modelo h y L conjuntos de entrenamiento independientes, si se entrenase cada modelo h_l en su conjunto S_l obteniendo las predicciones $h_1(\mathbf{x}), \ldots, h_l(\mathbf{x})$, hacer la media de la siguiente manera:

$$h_{\texttt{avg}}(\mathbf{x}) = \frac{1}{L}\sum_{l=1}^{L} h_l(\mathbf{x}) \tag{2.8}$$

daría lugar a un único modelo ensamblado en donde se conseguiría reducir la varianza de sus modelos base.

En la mayoría de problemas prácticos, no obstante, se dispondrá de un único conjunto de entrenamiento S. Bagging intenta resolver este problema construyendo L conjuntos de entrenamiento S_{*l} a través de hacer remuestros bootstrap con reemplazamiento sobre S. Así, entrenando cada modelo base $h_{*l}(\mathbf{x})$ en su conjunto de entrenamiento bootstrap S_{*l}, el ensamblado bagging para el caso de regresión tendría la siguiente forma:

$$h^L_{\texttt{bag}}(\mathbf{x}) = \frac{1}{L}\sum_{l=1}^{L} h_{*l}(\mathbf{x}) \tag{2.9}$$

en donde L indica el número de modelos base a entrenar.

Mientras que se podría tratar L como un hiper-parámetro más a ajustar, existen distintos trabajos en la literatura que prueban (teórica y empíricamente) que, fijados todos los parámetros de los modelos base h_{*l}, aumentar el número de modelos L por sí solo no produce un sobreajuste. Sin embargo, eso no significa que bagging no pueda dar lugar a un ensamblado que sobreajuste los datos, simplemente ese sobreajuste no puede ser provocado por el parámetro L. Se puede consultar Probst y Boulesteix (2017) para más detalles, en donde los autores prueban que la esperanza del error cometido por el ensamblado bagging es una función monótona de L para algunas funciones de coste, la cual tenderá a un valor particular de predicción conforme $L \to \infty$. Esto parece indicar que, en vez de gastar recursos computacionales en ajustar L, se puede simplemente escoger el mayor L posible que sea computacionalmente asumible.

Para el caso de clasificación la construcción $h^L_{\texttt{bag}}(\mathbf{x})$ en (2.9) es fácilmente adaptable. Una vez entrenados los clasificadores base $h_{*l}(\mathbf{x})$, se combinarían sus predicciones de alguna de las formas existentes (votación por mayoría, votación ponderada por el poder de predicción de cada clasificador base, etc.).

Propiedades del ensamblado bagging

Si bien la intención principal del método bagging es la reducción de la varianza de los clasificadores base, el hecho de entrenarse los L modelos base en remuestreos bootstrap de S hace que las clasificaciones de los mismos no sean del todo independientes, haciendo así que la reducción de la varianza sea menor. En concreto, sean L variables identicamente distribuidas pero no independientes, cuya correlación positiva entre cada par de ellas es $\rho > 0$, se tiene que la varianza de la media será:

$$\rho\sigma^2 + \frac{1-\rho}{L}\sigma^2 \tag{2.10}$$

en donde el segundo término desaparece conforme se aumenta el número de modelos base L, mientras que el primero permanece y limita los beneficios de agregar clasificadores base dependiendo de la correlación entre las predicciones de los mismos.

Aparte de potencialmente mejorar la varianza, bagging también puede producir mejoras en la exactitud de predicción. Dado el conjunto de observaciones independientes $(\mathbf{x}_i, y_i) \in S$, supóngase que estas fueron generadas por una distribución $\mathbb{P}(X, Y)$ que denotaremos $\mathbb{P}$ por simplicidad. A partir de esta, se define $h_{\mathtt{ag}}(\mathbf{x}) = E_{\mathbb{P}}[h_*(\mathbf{x})]$ como el estimador bagging verdadero del que $h^L_{\mathtt{bag}}(\mathbf{x})$ es una estimación. En este estimador verdadero, cada modelo $h_*(\mathbf{x})$ se entrena con el conjunto bootstrap S_* generado a partir de la distribución $\mathbb{P}$. Si bien este estimador no será de utilidad en la práctica al no conocer $\mathbb{P}$, sí será conveniente para derivar algunas propiedades teóricas sobre esta técnica. Bajo estas suposiciones, dado un $\mathbf{x}$ fijo y sea Y la variable respuesta continua que se quiere predecir, se tiene:

$$\begin{aligned} E_{\mathbb{P}}[Y - h_*(\mathbf{x})]^2 &= E_{\mathbb{P}}[Y - h_{\mathtt{ag}}(\mathbf{x}) + h_{\mathtt{ag}}(\mathbf{x}) - h_*(\mathbf{x})]^2 \\ &= E_{\mathbb{P}}[Y - h_{\mathtt{ag}}(\mathbf{x})]^2 + E_{\mathbb{P}}[h_*(\mathbf{x}) - h_{\mathtt{ag}}(\mathbf{x})]^2 \\ &\geq E_{\mathbb{P}}[Y - h_{\mathtt{ag}}(\mathbf{x})]^2 \end{aligned} \tag{2.11}$$

donde el error extra que aparece en la parte derecha de la ecuación ($E_{\mathbb{P}}[h_*(\mathbf{x}) - h_{\mathtt{ag}}(\mathbf{x})]^2$) es la varianza que tiene $h_*(\mathbf{x})$ alrededor de su media $h_{\mathtt{ag}}(\mathbf{x})$. Esto sugiere que, igual que ocurre con el estimador teórico bagging verdadero, bagging basado en remuestro bootstrap puede mejorar el error cuadrático medio. Además, reescribiendo $E_{\mathbb{P}}[h_*(\mathbf{x}) - h_{\mathtt{ag}}(\mathbf{x})]^2 > 0$, se obtiene:

$$E_{\mathbb{P}}[h_*(\mathbf{x})]^2 \leq E_{\mathbb{P}}[h^2_*(\mathbf{x})] \tag{2.12}$$

lo cual indica que la potencial mejora en el error cuadrático medio que se tendrá por hacer el agregado bagging viene dado por cuánto de distintos son los dos lados de la inecuación (2.12). Esta inecuación está relacionada con la estabilidad del método h_*: cuanto más cambie h_* al ser entrenado con distintos remuestros bootstrap de S_* generados por $\mathbb{P}$, mayor será esa diferencia, indicando así que son los métodos con más varianza los que más se benefician de ser agregados con técnicas bagging. Es por eso que se suelen utilizar como modelos base en bagging árboles de decisión o redes neuronales, y no tanto modelos más estables como pueden ser los k vecinos más cercanos (se puede consultar Breiman 1994 para un análisis con más detalle sobre el impacto que la estabilidad de los modelos base tiene en el ensamblado final).

No obstante, destacar que esta mejora en la exactitud no se da siempre en el caso de clasificación. En este, e igual que en otros casos vistos anteriormente, mientras que hacer bagging con clasificadores base buenos puede ser positivo, hacerlo con clasificadores base con peor exactitud que predecir aleatoriamente puede ser contraproducente.

Dado que bagging mejora la varianza de los clasificadores base escogidos, lo más habitual en la práctica es que se utilicen como modelos base árboles de decisión no podados, debido al poco sesgo y gran varianza que estos poseen.

Por último, destacar que una vez que se hace bagging a un conjunto de árboles de decisión, el ensamblado resultante ya no será un árbol, perdiendo así la facilidad de interpretación que estos poseen a cambio de una potencial mejora en su capacidad predictiva. Sin embargo, sí existen métricas para identificar la importancia de cada una de las variables regresoras, lo que puede ayudar en la tarea de interpretación del modelo. Una vez se obtiene una colección de árboles bagging entrenados, se puede calcular cuánto se reduce el valor de la función de coste (por ejemplo el índice Gini en el caso de la clasificación) al hacer una división para cada una de las variables predictoras, y hacer la media a lo largo de los L árboles. Variables con un valor elevado de esa métrica serán consideradas importantes para el problema.

2.3.5. Random Forest

Como se señalaba en la expresión (2.10), la reducción de varianza que se consigue mediante el ensamblado bagging dependía de cuánto de correlados estaban los clasificadores base creados. El al-

goritmo *Random Forest*, también introducido por Leo Breiman (2001), tiene como objetivo mejorar esa reducción de varianza a través de construir árboles de clasificación menos correlados entre ellos. A diferencia de los métodos vistos hasta ahora, donde los modelos base h podían pertenecer a cualquier familia, Random Forest está diseñado para ser utilizado únicamente con árboles de decisión.

Para conseguir el objetivo de reducción de varianza, antes de entrenar cada árbol h_l en su conjunto de entrenamiento bootstrap S_{*l} se selecciona aleatoriamente un subconjunto de $m \leq p$ variables para ser usadas, entrenando así el árbol únicamente con un subconjunto del total de variables regresoras. A esta selección de variables a utilizar se las conoce como bagging de características.

Sea $h_{l*}(x;\theta_l)$ el $l-$ésimo árbol de decisión entrenado en el conjunto de entrenamiento bootstrap S_{*l}, en donde θ_l representa el subconjunto de m variables usado para su entrenamiento. Se define el ensamblado Random Forest para el problema de regresión como:

$$h^L_{\mathtt{rf}}(\mathbf{x}) = \frac{1}{L}\sum_{l=1}^{L} h_{*l}(x;\theta_l) \tag{2.13}$$

donde el valor θ_l caracteriza por completo el $l-$ésimo árbol base en términos de las variables que se segmentan, los umbrales en cada uno de ellas, los valores de cada nodo terminal, y las m variables que se utilizan para entrenarlo. En general, en la práctica se suele utilizar $m = \sqrt{p}$, aunque se debe tratar como un meta-parámetro más a ajustar.

Análogamente al caso de bagging, el ensamblado para predicción en (2.13) es fácilmente adaptable al caso de clasificación utilizando algunas de las técnicas de combinación de clases predichas conocidas (votación comité por mayoría, Bayes ingenuo, etc.).

Error *Out-of-bag*

Una característica interesante que comparten tanto bagging como Random Forest es la posibilidad de aproximar el error de predicción esperado (o error de generalización) sin tener que hacerlo mediante la computacionalmente costosa validación cruzada, a partir del conocido como error *out-of-bag*. En el proceso de entrenamiento de un árbol con bagging se puede probar que cada modelo base en promedio hace uso de dos tercios del conjunto de entrenamiento S. El tercio de observaciones restantes son conocidas como observaciones *out-of-bag*, y dado que no son utilizadas para el entrenamiento de ese árbol, se pueden utilizar para estimar el error de test del mismo. Es decir, para cada uno de los datos de la muestra de entrenamiento $\mathbf{x}_i$, se puede predecir la respuesta con todos los árboles h_l en los cuales $\mathbf{x}_i$ no se usó en el entreno, y calcular el error promedio que se comete al predecir la clase de $\mathbf{x}_i$. Si se hace la media para todos los $\mathbf{x} \in S$, se tendrá una válida estimación del error de generalización del algoritmo, pudiéndose probar que si el número de árboles L es lo suficiente grande el error *out-of-bag* es equivalente a la validación cruzada con $R = N$ (es decir, subdividiendo el conjunto de entrenamiento S en subconjuntos de cardinalidad 1).

Al igual que con bagging, del ensamblado Random Forest también se puede sacar la métrica de importancia de cada variable midiendo cuánto mejora el error de clasificación la división al segmentar el árbol por esa variable. No obstante, a mayores con Random Forest se puede utilizar el error *out-of-bag* para definir una nueva métrica de importancia de variables. Para cada árbol h_l creado, se calcula el error de predicción con las observaciones *out-of-bag*. A posteriori, se selecciona la $j-$ésima variable de $\mathbf{x}$ y se le añade ruido. Se calcula la diferencia promedio de precisión de clasificación dada la perturbación en la variable j, y haciéndose para todas las variables se tendría una medida de importancia de cada variable utilizada.

Propiedades del ensamblado Random Forest

Análogamente a lo comentado en el caso de bagging, Random Forest puede construir un ensamblado de árboles que sobreajusta los datos, pero este no puede ser provocado únicamente por entrenar un número L de árboles base demasiado grande (véase Breiman 2001). Así, lo habitual en la práctica es seleccionar un L tan grande como capacidad computacional se tenga.

El sobreajuste, no obstante, sí podría ser provocado por la utilización de árboles base de tamaño máximo (construidos sin criterio de parada), originando un modelo demasiado flexible. En distintos trabajos de investigación se estudió empíricamente que el uso de árboles base menos largos puede dar lugar a ligeras mejoras de rendimiento del clasificador, aunque estas pueden no valer la pena el coste computacional de estimar el hiper-parámetro de profundidad de árbol base óptimo (véase Hastie et al 2008).

Varianza, correlación entre árboles base y sesgo

Sea el estimador Random Forest $h^L_{\mathtt{rf}}(\mathbf{x})$ para un problema de regresión, se puede escribir su forma límite cuando $L \to \infty$ como:

$$h_{\mathtt{rf}}(\mathbf{x}) = E_{\theta|S}[h_{*l}(x;\theta_l)]$$

donde se hace explícita su dependencia con el conjunto de entrenamiento S. Por la expresión (2.10) se tiene que:

$$Var[h_{\mathtt{rf}}(\mathbf{x})] = \rho(\mathbf{x})\sigma^2(\mathbf{x}) \tag{2.14}$$

en donde $\rho(\mathbf{x})$ es la correlación entre cualquier par de árboles usados en el ensamblado:

$$\rho(\mathbf{x}) = corr[h_{*l}(x;\theta_l), h_{*j}(x;\theta_j)] \tag{2.15}$$

representando θ_l y θ_j a cualquier par de árboles entrenados en un remuestro bootstrap de S.

Por otro lado, $\sigma^2(\mathbf{x})$ es la varianza de cualquiera árbol base:

$$\sigma^2(\mathbf{x}) = Var[h_{*l}(x;\theta_l)].$$

Hastie et al (2008) llevan a cabo la simulación representada en la Figura 2.3, donde calculan la correlación (2.15) entre pares de árboles pertenecientes a un ensamblado Random Forest en distintos puntos $\mathbf{x}$, en función del valor m escogido. Como se puede observar, esta correlación disminuye conforme disminuye m, al ser más probable que dos árboles base sean menos parecidos si para entrenarlos se utilizan distintos conjuntos de variables. En la práctica, la correlación entre árboles base siempre será baja. Esto, junto a la expresión (2.14) da lugar a que el ensamblado Random Forest disminuya mucho la varianza individual de los árboles base.

Al igual que ocurre con bagging, el sesgo del ensamblado Random Forest al predecir un punto $\mathbf{x}$ será el mismo que el sesgo de cualquiera de los modelos individuales $h_{*l}(x;\theta_l)$:

$$\begin{aligned} \text{Sesgo}(\mathbf{x}) &= E[Y|X=\mathbf{x}] - E_S[h_{\mathtt{rf}(\mathbf{x})}] \\ &= E[Y|X=\mathbf{x}] - E_S\left[E_{\theta|S}[h_{*l}(\mathbf{x};\theta_l)]\right]. \end{aligned} \tag{2.16}$$

Este sesgo será generalmente más alto que el que se obtendría con un árbol no podado entrenado a partir de S, dadas las restricciones que supone entrenar los árboles Random Forest con remuestreos de S y con un número de variables $m < p$. Así, las mejoras en predicción que se pueden obtener mediante bagging o Random Forest son únicamente resultado de reducir la varianza.

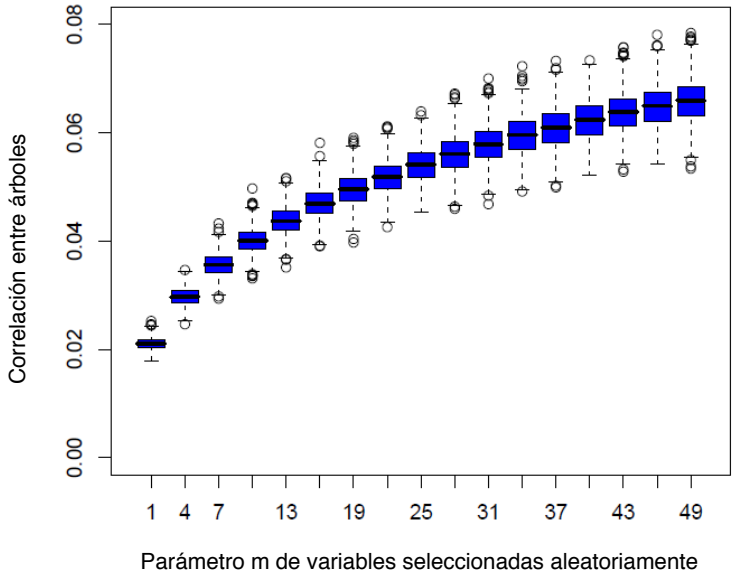

Figura 2.3: Se simulan datos bajo la expresión $Y = \frac{1}{\sqrt{50}} \sum_{j=1}^{50} X_j + \epsilon$, siendo X_j y ϵ gausianas e iid. Para cada parámetro m se crea un ensamblado Random Forest y se calcula la correlación media entre pares de árboles base.

Relación de Random Forest con $k-$NN

Tanto Random Forest como el método $k-$NN revisado en la sección 1.3 se pueden ver como métodos de vecindarios ponderados, en donde para realizar una predicción de un punto $\mathbf{x}$ se busca información en un "vecindario" de $\mathbf{x}$. Matemáticamente se pueden representar ambos métodos como:

$$h(\mathbf{x}) = \sum_{i=1}^{N} W(\mathbf{x}_i, x) y_i \tag{2.17}$$

en donde para predecir la cantidad (o clase) y de un nuevo punto $\mathbf{x}$, se promedian las cantidades de los N puntos y_i del conjunto de entrenamiento S, ponderándolas por el peso $W(\mathbf{x}_i, x)$ que será mayor para los $\mathbf{x}_i$ más cercanos a $\mathbf{x}$.

- Para el caso de los $k-$NN, los pesos son $W(\mathbf{x}_i, x) = \frac{1}{k}$ si $\mathbf{x}_i$ es uno de los k puntos más cercanos a $\mathbf{x}$, y 0 en otro caso.
- Para el caso de un árbol, $W(\mathbf{x}_i, x) = \frac{1}{k'}$ si $\mathbf{x}_i$ es uno de los k' puntos en el mismo nodo terminal que $\mathbf{x}$, y 0 en otro caso.

Como Random Forest hace un promedio de todos los árboles base, su predicción se puede escribir de la siguiente forma:

$$h_{\mathbf{rf}}^{L}(\mathbf{x}) = \frac{1}{L} \sum_{l=1}^{L} \sum_{i=1}^{N} W(\mathbf{x}_i, x) y_i = \sum_{i=1}^{N} \left(\frac{1}{L} \sum_{l=1}^{L} W(\mathbf{x}_i, x) \right) y_i \tag{2.18}$$

lo cual indica que el "bosque" de árboles se puede interpretar como una ponderación de los puntos del vecindario, donde el peso de cada punto es a su vez un promedio de los pesos de cada árbol individual. La diferencia principal radica en que en el contexto de los árboles el vecindario de $\mathbf{x}$ serán los puntos $\mathbf{x}_i$ que comparten nodo terminal en alguno de los l árboles base, dependiendo de forma compleja así el vecindario de la estructura de los árboles construidos.

2.3.6. Boosting

Las técnicas de boosting son probablemente uno de los avances más importantes en el área del aprendizaje estadístico de las últimas dos décadas. Estas, al igual que bagging y Random Forest, son una familia de algoritmos que tienen como objetivo crear un clasificador con un error de entrenamiento arbitrariamente pequeño partiendo de un conjunto de clasificadores base.

Aprendizaje fuerte y débil

En el núcleo de las técnicas de boosting están los conceptos fundamentales de aprendizaje débil y aprendizaje fuerte. Estos tienen sus raíces en la teoría PAC (*probably approximately correct*), y los definimos a continuación basándonos en el trabajo de Kearns (1994).

Supóngase una hipótesis $f \in \mathcal{H}$ en un contexto de clasificación binaria, i.e., $f : \mathbb{R}^p \to \{-1, 1\}$, en donde $\mathcal{H}$ es un espacio de funciones de $\mathbb{R}^p$ a $\{-1, 1\}$. Imaginemos, como hasta ahora, que disponemos de un conjunto de ejemplos $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ que vienen generados por una distribución no conocida $\mathbb{P}(X, Y)$, tal que $f(\mathbf{x}_i) = y_i$.

Se define como un algoritmo de aprendizaje fuerte (*strong learner*) aquel algoritmo $\mathcal{A}$ que, dado un S suficientemente grande, es capaz de generar un modelo "bueno" con alta probabilidad. Es decir, de manera formal, dada la distribución $\mathbb{P}(X, Y)$ (que denotaremos $\mathbb{P}$) y una función $f \in \mathcal{H}$, para todos $\epsilon \in (0, 1/2)$ y $\delta \in (0, 1/2)$, el algoritmo dará lugar, con probabilidad mayor o igual que $1 - \delta$, a un clasificador $h : \mathbb{R}^p \to \{-1, 1\}$ cuyo error satisfaría que $\mathbb{P}_{\mathbb{P}}[h(\mathbf{x}) \neq f(\mathbf{x})] \leq \epsilon$. Además, la complejidad computacional del algoritmo en cuestión sería como mucho polinomial dependiendo de $1/\epsilon$, $1/\delta$, N y la dimensión del espacio de características (p).

Por otro lado, mientras que el aprendizaje fuerte puede dar lugar a clasificadores con un error tan próximo a 0 como queramos (dado que se cumple para todo $\epsilon \in (0, 1/2)$ y $\delta \in (0, 1/2)$), el aprendizaje débil solo requiere que el clasificador sea ligeramente mejor que la estimación aleatoria. Formalmente, dada cualquier distribución $\mathbb{P}$, una función $f \in \mathcal{H}$ y un $\delta \in (0, 1/2)$, se dice que un algoritmo $\mathcal{A}$ de aprendizaje es débil si existe algún $\gamma > 0$ tal que el clasificador devuelto tiene, con probabilidad mayor o igual que $1 - \delta$, un error que cumple $\mathbb{P}_{\mathcal{P}}[h(\mathbf{x}) \neq f(\mathbf{x})] \leq 1/2 - \gamma$. En la literatura a este γ se le denomina como borde o ventaja del algoritmo $\mathcal{A}$, denotando la "ventaja" que este tiene sobre un modelo que simplemente haga una clasificación al azar.

El concepto de aprendizaje débil fue introducido por Kearns y Valiant (1988, 1989), y con él la pregunta de si el aprendizaje débil y fuerte eran equivalentes. Esta pregunta se acuñó como *hypothesis boosting problem*, dado que si fuera cierta, sería equivalente a decir que existe un método que nos permitiría mejorar (*boost*) uno o varios algoritmos débiles para dar lugar a uno fuerte con un error de clasificación arbitrariamente pequeño.

Esta equivalencia fue probada por Robert Shapire (1990) en su artículo *The Strength of Weak Learnability*, uno de los estudios más relevantes en el ámbito de la teoría detrás del ensamblado estadístico. La prueba es una construcción en donde se describe un algoritmo capaz de convertir un modelo débil en uno fuerte. Esta técnica, descrita en el Algoritmo 1, se acuñó como boosting, y fue la base de la mayoría de las variantes de boosting que se conocen hoy en día. La motivación del Algoritmo 1, tal y como se explica en dicho artículo, se describe a continuación.

Algoritmo 1 Procedimiento de boosting original para problema de clasificación binario

Entrada: Un conjunto de entrenamiento $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, en donde $\mathbf{x}_i \in \mathbb{R}^p$ y $y_i \in \{-1, 1\}$
Salida: Un clasificador $h_{\texttt{boost}} : \mathbb{R}^p \to \{-1, 1\}$

1: Seleccionamos, sin reemplazamiento, $N_1 < n$ elementos de S para dar lugar a S_1^*.
2: Entrenamos un algoritmo débil sobre S_1^* para obtener un clasificador h_1.
3: Seleccionamos $N_2 < n$ elementos de S, donde la mitad de ellos fueron clasificados erróneamente por h_1, dando lugar al conjunto de entrenamiento S_2^*.
4: Entrenamos de nuevo el algoritmo débil sobre S_2^* para dar lugar al clasificador h_2.
5: Seleccionamos todos los elementos de S en donde la respuesta de h_1 y h_2 no coincide, dando lugar al tercer conjunto de entrenamiento S_3^*
6: Entrenamos finalmente el algoritmo débil sobre S_3^* para dar llugar al modelo h_3.
7: Juntamos los 3 modelos entrenados, ensamblándolos en un único clásificador con la regla de la votación por mayoría: $h_{\texttt{boost}}(\mathbf{x}) = \text{signo}\big(\sum_{i=1}^{3} h_i(\mathbf{x})\big)$.

Supóngase que ante un problema de clasificación binario se tiene un algoritmo débil $\mathcal{A}$ el cual, por definición, es capaz de producir un clasificador ligeramente mejor que la predicción aleatoria. En este caso, supongamos que el clasificador está $\epsilon-$ cerca de la función objetivo, es decir, que el error de clasificación es $\epsilon \in (0, 0.5)$. El Algoritmo 1 intentará crear un algoritmo $\mathcal{A}'$, que, en base a entrenarse en distintas variaciones del conjunto de entrenamiento S disponible, de lugar a una mejora en el error de clasificación.

Para comenzar, el algoritmo $\mathcal{A}$ se entrena en un subconjunto de elementos de S (S_1^*), obteniendo un clasificador inicial h_1 con un error de clasificación ligeramente inferior al 50 %. Intuitivamente, se podría decir que h_1, mediante su entrenamiento, ha aprendido a generalizar ligeramente sobre el conjunto de datos, i.e., ha aprendido ligeramente la distribución inicial de los datos. Para mejorar el rendimiento de $\mathcal{A}$, se tendrá que forzar a que aprenda más en las partes "difíciles" del espacio de características dado. Para ello, creamos un nuevo conjunto de datos de entrenamiento S_2^* (i.e., una nueva distribución) cogiendo otro subconjunto de elementos S con la peculiaridad de que la mitad tienen que haber sido clasificados erróneamente por el modelo inicial h_1. A raíz de entrenar $\mathcal{A}$ sobre S_2^* tendremos el segundo modelo h_2, que se podría decir que se ha forzado a que aprendiera sobre un conjunto de entrenamiento "difícil". Por último, cogemos todos los elementos de S para los cuales $h_1(\mathbf{x}) \neq h_2(\mathbf{x})$, y entrenamos sobre ellos el último modelo h_3. Finalmente se construye el ensamblado $h(\mathbf{x})$ usando el criterio de la votación por mayoría.

Schapire probó, por un lado, que si el error del clasificador débil inicial es ϵ, entonces el error del ensamblado $h(\mathbf{x})$ estará acotado superiormente por $g(\epsilon) \equiv 3\epsilon^2 - 2\epsilon^3$, mejorando así el error del clasificador original. Por otro lado, probó que el error se puede continuar reduciendo aplicando recursivamente el Algoritmo 1 para dar lugar a los sucesivos $h(\mathbf{x})$. Además, también probó que dicha técnica se puede ejecutar en tiempo polinomial.

AdaBoost

De forma simililar a los métodos de bagging, las técnicas boosting también entrenarán un número L de modelos base $h_1, \ldots, h_L$. No obstante, en vez de entrenar los L modelos paralelamente en perturbaciones bootstrap del conjunto S para posteriormente agregarlos, el entrenamiento se hace de manera secuencial, donde cada modelo base se va entrenando repetidamente en versiones modificadas de S, cuyas modificaciones dependieron de los resultados obtenidos por el modelo anterior.

A continuación se presenta el algoritmo AdaBoost, una de las implementaciones de boosting más populares. Introducido en 1997 por Freund y Schapire, fue el primer algoritmo de boosting adaptativo, resolviendo así muchas de las limitaciones que poseían los métodos presentados hasta ese momento. Creado con el objetivo de mejorar la capacidad predictiva de los clasificadores binarios, AdaBoost

está diseñado para que en cada paso del entreno la distribución de los datos S se vaya adaptando a los resultados del clasificador actual, para así poner más peso en los puntos $\mathbf{x} \in S$ que el modelo clasifica de forma incorrecta. Finalmente, se utilizará un promedio ponderado de todos estos modelos secuenciales para dar lugar a la predicción final ensamblada.

En el Algoritmo 2 se define el funcionamiento de AdaBoost, en donde suponemos un problema de clasificación binario en donde disponemos de un conjunto de entrenamiento S con N pares de datos $(\mathbf{x}_i, y_i)$ donde $y_i \in \mathcal{G} = \{-1, 1\}$, y donde L el número de clasificadores base a entrenar (en concreto árboles de decisión en el caso de AdaBoost).

Aparte de AdaBoost, dentro de la familia de técnicas de boosting existen una gran cantidad de implementaciones distintas, diferenciándose la mayoría de ellas en la forma de calcular los pesos α_l asignados a cada clasificador base y los pesos w_i^l que se asignan a cada $\mathbf{x}_i \in S$ en cada iteración. Sin embargo, todos ellos siguen una filosofía parecida. En vez de entrenar, como se vio en las técnicas bagging, árboles base sin criterio de parada que pueden dar lugar a sobreajuste, las técnicas boosting intentan aprender más despacio generando sucesivamente árboles más cortos y adaptándose de forma secuencial a los errores que se van cometiendo en las iteraciones pasadas.

Los principales hiperparámetros a ajustar en un modelo boosting son los siguientes:

1. **El número de árboles base** L. Si bien en bagging y Random Forest el hecho de aumentar el número de modelos base a entrenar no suele producir un sobreajuste por sí mismo, boosting sí puede dar lugar a un sobreajuste a los datos cuando L es muy grande (aunque no es habitual, y de ocurrir lo hace de manera muy lenta). Este parámetro se suele seleccionar utilizando validación cruzada o haciendo uso del error *out-of-bag* conforme se va entrenando el modelo.

2. **El peso** α_l **de cada modelo en el ensamblado final**. También denominado parámetro de contracción, este controlará la velocidad de aprendizaje del método de boosting. En algunos casos se selecciona un valor constante, típicamente $\alpha_l = \alpha = 0.01$ ó 0.001 dependiendo del problema, y también ajustado por validación cruzada. En otros casos, como el ya mencionado AdaBoost, el parámetro de constracción α_l será adaptativo, recalculándose en cada iteración l.

3. **Los parámetros a ajustar de la familia de modelos base** h_l **escogida**. Si bien se puede *boostear* cualquier tipo de modelo base, lo más habitual es hacerlo con árboles de decisión, por su bajo sesgo y su facilidad computacional. Así, el parámetro principal a ajustar en este caso sería el número de segmentaciones d que se le permiten a cada árbol h_l creado, que controlará la complejidad (i.e., el tamaño) del mismo. Una selección popular en la práctica es $d = 1$, haciendo así que cada árbol esté compuesto por una única segmentación sobre una única variable (dando así a árboles con $|T| = 2$ nodos terminales). Estos árboles también se conocen como tocones (o *stump trees* en inglés).

Propiedades teóricas de AdaBoost

La principal propiedad teórica del algoritmo AdaBoost prueba su capacidad de reducir el error de entrenamiento conforme añadimos modelos base al ensamblado. Esta, demostrada por Shapire y Singer (1999), acota el error de entrenamiento del algoritmo según el número de modelos base entrenados.

Teorema 2.1 *Sea* h_{ada} *el clasificador ensamblado devuelto por el algoritmo AdaBoost con parámetro fijado* L*, y* γ_l *la ventaja del modelo base* h_l *tal y como se definió al inicio de la sección. Entonces:*

$$RE(h_{ada}) = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq h_{ada}(\boldsymbol{x}_i)) \leq e^{-2 \sum_{l=1}^{L} \gamma_l^2}. \tag{2.22}$$

Algoritmo 2 Algoritmo AdaBoost para problemas de clasificación binarios

Entrada: Conjunto de entrenamiento $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, con $\mathbf{x}_i \in \mathbb{R}^p$ y $y_i \in \{-1, 1\}$. Número de clasificadores base L.

Salida: Un clasificador $h_{\texttt{ada}} : \mathbb{R}^p \to \{-1, 1\}$

1. Se inicializan los pesos asignados a cada dato $\mathbf{x}_i \in S$: $w_i^1 = \dfrac{1}{N},\ i = 1, \ldots, N$

2. para $l = 1, \ldots, L$, se calcula:

   a) Se ajusta un clasificador base débil $h_l(\mathbf{x})$ al conjunto de entrenamiento S utilizando los pesos actuales w_i^l. Este modelo base tendrá como error $\mathbb{P}_{i\sim w_i^l}[h_l(\mathbf{x}_i) \neq y_i]$, en donde cabe destacar que el cálculo se hace con respecto a la distribución w_i^l sobre la que se entrenó el modelo (i.e., teniendo en cuenta los pesos que cada $\mathbf{x}_i$ tiene en S). En la práctica, esto significa que el algoritmo base usará en la función de coste los pesos w_i^l. En caso de no ser esto posible, también se puede hacer mediante un muestreo de S con reemplazamiento acorde a la distribución de los pesos, utilizando el conjunto de datos resultante para entrenar el modelo.

   b) Se calcula:

   $$\epsilon_l = \sum_{i=1}^{N} w_i^l I(y_i \neq h_l(\mathbf{x}_i)) \tag{2.19}$$

   obteniendo el error de clasificación cometido por el modelo h_l ponderado por los pesos de cada $\mathbf{x}_i$.

   c) Se calcula $\alpha_l = \dfrac{1}{2} \log\left(\dfrac{1-\epsilon_l}{\epsilon_l}\right)$, que será el peso que tendrá modelo h_l en la suma ponderada final del ensamblado. Destacar que esta forma de calcular α_l es la presentada por Schapire y Freund en el artículo original, existiendo en la literatura actual numerosas formas de calcularla dependiendo del problema en cuestión.

   d) Se recalculan los pesos a utilizar en la siguiente iteración para cada punto del conjunto de entrenamiento S:

   $$w_i^{l+1} = \frac{w_i^l \cdot \exp(\alpha_l y_i h_l(\mathbf{x}_i))}{Z_l},\ i = 1, 2, \ldots, N \tag{2.20}$$

   en donde $Z_l = \sum_{i=1}^{N} w_i^l \cdot e^{\alpha_l y_i h_l(\mathbf{x}_i)}$ es un factor normalizador.

3. El modelo ensamblado final será:

$$h_{\texttt{ada}}(\mathbf{x}) = signo\left[\sum_{l=1}^{L} \alpha_l h_l(\mathbf{x})\right] \tag{2.21}$$

Así, si existe un γ tal que $\gamma_l \geq \gamma$ para todo l, se tiene:

$$RE(h_{ada}) = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq h_{ada}(\boldsymbol{x}_i)) \leq e^{-2\gamma^2 L}. \tag{2.23}$$

Este teorema prueba una cota superior del error de entrenamiento de AdaBoost que decrece exponencialmente conforme aumentamos el número de modelos base L. Importante comentar que, aunque la cota final del error depende de γ, el algoritmo como tal no depende de dicho parámetro, así que el resultado del teorema está garantizado sin tener que seleccionar un γ particular. La demostración de este teorema se puede ver en el apéndice (demostración A.1).

No obstante, como se comentó en secciones anteriores, la métrica del error de entrenamiento puede no ser un buen indicador sobre la capacidad de generalización de un algoritmo. La métrica que sí podría ser más interesante a la hora de conocer la capacidad de predicción es el error de test esperado del clasificador (1.16), para el cual los mismos autores en 1997 probaron que, con alta probabilidad, está acotado superiormente por:

$$ETE(h_{\texttt{ada}}) \leq RE(h_{\texttt{ada}}) + \tilde{O}\left(\sqrt{\frac{d_{VC}L}{N}}\right). \tag{2.24}$$

En donde d_{VC} es la dimensión VC [1] del conjunto de clasificadores base, N es el número de elementos del conjunto de entrenamiento, y L el número de modelos base entrenados. Se utiliza la notación $\tilde{O}$ (en vez de O) para denotar que se están obviando términos logarítmicos y constantes.

Este resultado muestra que la capacidad de generalización de $h_{\texttt{ada}}$ está controlada por un equilibrio entre la flexibilidad o complejidad de los modelos base. Cuanto más fuerte sea el modelo base, menor será $RE(h_{\texttt{ada}})$, pero mayor será la dimensión VC (d_{VC}). Y viceversa, modelos base más débiles o simples tendrán una dimensión VC menor, pero requerirán de un número L de modelos base a entrenar mucho mayor. La prueba de (2.24), fuera del alcance de este texto, se puede consultar en Freund y Schapire (1997).

Además, a partir de la cota (2.24) se puede deducir que dado un L demasiado grande AdaBoost podría dar lugar a un sobreajuste de los datos. No obstante, numerosos estudios empíricos (destacando Breiman 1998) observaron que, conforme L crecía, el error de test no solo no solía aumentar, sino que en muchos casos decrecía, contradiciendo así la interpretación de la cota mencionada. A raíz de estas observaciones, Shapire et al (1998) derivaron una explicación alternativa al comportamiento del poder de generalización de AdaBoost, interpretándolo como un clasificador basado en márgenes (de hecho, esta interpretación relaciona las técnicas de boosting con las basadas en aprendizaje discriminante basado en márgenes, como las máquinas de vectores de soporte).

Formalmente, de nuevo en el contexto de clasificación binaria sobre S en donde $f(\mathbf{x}_i) = y_i \in \{-1, 1\}$, se define el margen de un par de elementos $(\mathbf{x}_i, y_i)$ para un modelo h como $f(\mathbf{x}_i)h(\mathbf{x}_i) = y_i h(\mathbf{x}_i)$. Es decir, el margen de $\mathbf{x}_i$ sería la distancia de él al hiperplano de frontera de decisión definido por h. En concreto, para AdaBoost el margen de $(\mathbf{x}_i, y_i)$ sería $f(\mathbf{x}_i)h_{\texttt{ada}}(\mathbf{x}_i) = y_i \sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i) =$

[1] La dimensión VC (Vapnik-Chervonenkis) es una medida de la compejidad que tiene un espacio de funciones binarias. Se define un modelo h como separador de un conjunto S si, para cualquier asignación de clases y_i de todo $\mathbf{x}_i$, existen parámetros θ de h tal que h no comete errores al clasificar los puntos de S. Así, para problemas de clasificación binaria, la dimensión VC de un modelo h es el máximo d_{VC} para el cual existe un conjunto S de cardinalidad d_{VC} que es separable por h. Se puede consultar en detalle su definición y su relación con la teoría del aprendizaje estadístico en Vapnik y Chervonenkis (1971).

$\sum_{l=1}^{L} \alpha_l y_i h_l(\mathbf{x}_i)$, y el margen normalizado sería:

$$f(\mathbf{x}_i)h_{\texttt{ada}}(\mathbf{x}_i) = \frac{\sum_{l=1}^{L} \alpha_l y_i h_l(\mathbf{x}_i)}{\sum_{l=1}^{L} \alpha_l} \tag{2.25}$$

en donde α_l es el peso del modelo h_l en el ensamblado final.

Este margen definido en (2.25) será un número en $[-1, 1]$, positivo únicamente si $h_{\texttt{ada}}$ clasifica bien el ejemplo $\mathbf{x}_i$, y que se puede interpretar como una métrica de confianza de la predicción (i.e., a mayor valor, mayor la confianza que se dispone para clasificar $\mathbf{x}_i$ en la clase correspondiente).

Así, basándose en el concepto de margen, Shapire et al (1998) probaron la siguiente cota del error de generalización, mejorando la dada en (2.24):

Teorema 2.2 *Sea $\mathbb{P}(X, Y)$ una distribución sobre $\mathbb{R}^p \times \{-1, 1\}$, S el conjunto de entrenamiento en donde tenemos N pares de datos $(\boldsymbol{x}_i, y_i)$ escogidos independientemente de $\mathbb{P}(X, Y)$. Sea el espacio de clasificadores base $\mathcal{H}$ un espacio con dimensión VC d_{VC}, y una constante $\delta > 0$. Entonces, con probabilidad mayor o igual que $1-\delta$, el error de test del ensamblado h_{ada} satisface la siguiente expresión para todo $\theta > 0$:*

$$ETE(h_{ada}) \leq \mathbb{P}_S[yh_{ada}(\boldsymbol{x}) \leq \theta] + O\left(\frac{1}{\sqrt{N}}\left(\frac{d_{VC}\log^2(N/d_{VC})}{\theta^2} + \log\left(1/\delta\right)\right)^{1/2}\right). \tag{2.26}$$

Si usamos de nuevo la notación $\tilde{O}$, y obviamos términos logaritmicos y constantes, tenemos:

$$ETE(h_{\texttt{ada}}) \leq \mathbb{P}_S[yh_{\texttt{ada}}(\mathbf{x}) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d_{VC}}{N\theta^2}}\right) \tag{2.27}$$

en donde la probabilidad del primer término se toma sobre el conjunto S, cuantificando así la probabilidad de que un punto del conjunto de entrenamiento tenga un margen menor al θ definido. Como se puede observar, esta cota da a entender que cuanto mayor sea el margen sobre el conjunto de entrenamiento, menor será el error de test o generalización. Por otro lado, es de interés ver como el segundo término es totalmente independiente del número de modelos base a utilizar L, a diferencia de (2.24).

Así, esta cota se suele utilizar como explicación a porqué AdaBoost no tiende a sobreajustar los datos cuando se escogen valores de L muy grandes, ya que, aún después de llegar a un error de entrenamiento sobre S igual a 0, se interpreta que el algoritmo sigue maximizando el margen en S para así minimizar el error de test.

Variantes y extensiones de AdaBoost

En la literatura existen numerosas variantes y extensiones del algoritmo AdaBoost original. A continuación se presentan las más relevantes: la extensión a problemas de clasificación multiclase y a problemas de regresión.

- **AdaBoost para problemas de clasificación multiclase**

  Existen varias técnicas para extender AdaBoost al caso de un problema de clasificación multiclase. La primera y más sencilla, AdaBoost.M1 (definida en el Algoritmo 3), fue presentada por el propio Shapire en 1997. En esta, el error de clasificación que comete cada modelo base es usado para

crear una métrica del apoyo que recibe cada una de las clases, escogiendo la clase que maximiza dicha cantidad.

No obstante, y como se puede ver en la definición del algoritmo, uno de los principales problemas de AdaBoost.M1 es que exige a los clasificadores base a tener un error de clasificación multiclase menor a 1/2, algo que puede ser complicado si estamos ante un problema difícil con muchas clases.

Posteriormente surgieron los algorirtmos AdaBoost.M2 y AdaBoost.MR, que palían las restricciones de AdaBoost.M1 a raíz de dividir el problema multiclase en un conjunto de problemas binarios más sencillos.

Por un lado, AdaBoost.MH crea un problema binario por cada instancia $\mathbf{x}$ y su posible clase $\mathcal{G}_k \in \mathcal{G}$ (en donde $\mathcal{G}$ es el conjunto de las posibles clases con cardinalidad K), e intenta resolver la pregunta de si a dicho $\mathbf{x}$ se le debe asociar la clase $\mathcal{G}_k$ o cualquiera de las otras. Esta propuesta se conoce como "uno contra el resto" (*one-versus-rest*, u OVR), y da lugar a descomponer el problema multiclase en K problemas binarios, donde K es el número de clases posibles.

Una vez creados los K clasificadores binarios $h_{\texttt{ada}}^{\mathcal{G}_k}$, se utiliza la salida real[2] de cada uno de ellos, y a posteriori se escoge la clase más probable. Es decir, la clase $\mathcal{G}_k$ que satisface:

$$h_{\texttt{ada.m2}}(\mathbf{x}) = \arg\max_{\mathcal{G}_k \in \mathcal{G}} h_{\texttt{ada}}^{\mathcal{G}_k}. \tag{2.28}$$

Por otro lado, AdaBoost.M2 (explicado en el Algoritmo 4) crea un problema binario por cada $\mathbf{x}$ y cada par de clases $\mathcal{G}_k, \mathcal{G}_j$, preguntándose si la clase que le corresponde a $\mathbf{x}$ es una u otra. La propuesta de AdaBoost.MH se conoce como "uno contra uno" (*one-versus-one*, u OVO), y convierte el problema original en $\frac{K(K-1)}{2}$ problemas binarios.

Este algoritmo es más sofisticado y complejo que AdaBoost.M1. Como se puede observar el en paso 3.a en Algoritmo 4, se le da más libertad a los modelos base, dado que en vez de devolver la clase concreta que predice para un $\mathbf{x}$ particular, devuelve un vector en $[0,1]^K$ en donde cada componente k nos da una idea de cuánto de plausible[3] es que $\mathbf{x}$ pertenezca a la clase $\mathcal{G}_k$. Además, en vez de calcular el error con la formula clásica (usada por ejemplo en 2.b en Algoritmo 3) se utiliza el pseudo-error, el cual se calcula sobre todos los pares de datos y sus clases incorrectas. De este modo, el algoritmo hace que los modelos base se centren no solo en clasificar puntos difíciles, sino también en distinguir correctamente aquellos que son difíciles de discriminar.

Así, si por ejemplo en el modelo base se tiene $h_l(\mathbf{x}_i, y_i) = 1$ y $h_l(\mathbf{x}_i, y) = 0$, significa que el modelo h_l clasifica la clase de $\mathbf{x}_i$ correctamente. Si por el contrario $h_l(\mathbf{x}_i, y_i) = 0$ y $h_l(\mathbf{x}_i, y) = 1$, entonces h_l clasifica la clase de $\mathbf{x}_i$ incorrectamente. Es esta interpretación la que lleva a construir la función de pseudo-error del punto 3.b del algoritmo, para la cual es trivial encontrar un modelo que tenga error 1/2 (por ejemplo, cualquier modelo de respuesta constante). Es esta última propiedad la que permite a Adaboost.M2 paliar el problema de AdaBoost.M1 de, sobre todo con K grande, encontrar modelos base que tuvieran error menor a 1/2.

Por último, en la fase de clasificación AdaBoost.M2 asigna a $\mathbf{x}$ la clase $y = \mathcal{G}_k$ que maximiza la media ponderada de los modelos débiles base $h_l(\mathbf{x}, y)$. En concreto, si los modelos base tienen pseudo-errores $\epsilon_1, \ldots, \epsilon_L$, y sea $\gamma_l = 1/2 - \epsilon_l$, entonces el error de entrenamiento de $h_{\texttt{ada.m2}}$ está

[2] La salida real del algoritmo AdaBoot es el resultado de $\sum_{l=1}^{L} \alpha_l h_l(\mathbf{x})$, es decir, sin aplicar como último paso función *signo*() para obtener la clase asignada. Esta cantidad puede servir para interpretar cuanto de segura es la predicción.

[3] Denotar que ese vector no es una probabilidad, por lo que sus componentes no tienen que sumar 1. Por lo tanto, sus valores no pueden ser interpretados como probabilidad de pertenecer a una clase.

acotado por:

$$\begin{aligned} RE(h_{\texttt{ada.m2}}) = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq h_{\texttt{ada.m2}}(\mathbf{x}_i)) \leq & (K-1)\prod_{l=1}^{L}\sqrt{(1-4\gamma_l^2)} \\ \leq & (K-1)\exp\left(-2\sum_{l=1}^{L}\gamma_l^2\right). \end{aligned} \tag{2.29}$$

La prueba de (2.29) se puede ver en Freund y Shapire (1997).

- **Boosting para problemas de regresión**

  La principal extensión de boosting a problemas de regresión (donde la variable respuesta es continua, i.e., $y \in \mathbb{R}$) es el algoritmo AdaBoost.R presentado en Algoritmo 5. Esta técnica, tal y como la presentaron originalmente Freund y Shapire (1997), contempla que la variable respuesta y esté en el intervalo cerrado $[0, 1]$. Su estrategia es convertir el problema de regresión en uno de clasificación, para luego aplicarle a este último el algoritmo AdaBoost original.

  Para llevar a cabo esta conversión de problemas, por cada dato en el conjunto de entrenamiento $(\mathbf{x}_i, y_i)$ se crea un conjunto infinito y continuo de puntos indexado por los pares (i, y) para todo $y \in [0, 1]$. Así, se obtendría un conjunto en donde tendríamos las instancias $\tilde{\mathbf{x}}_{i,y} = (\mathbf{x}_i, y)$ cuya respuesta sería $\tilde{y}_{i,y} = I[y \geq y_i]$. Informalmente, se podría decir que cada instancia $(\mathbf{x}_i, y_i)$ se mapea a un conjunto infinito de problemas binarios en donde, para cada $y \in [0, 1]$ se intenta responder si la respuesta correcta y_i es mayor o menor que y.

  Al mismo tiempo, cada modelo base $h_l : \mathbb{R}^p \to [0, 1]$ también se ve reducido a una hipótesis binaria del estilo $\tilde{h}_l : \mathbb{R}^p \times [0, 1] \to \{0, 1\}$ definida como:

$$\tilde{h}_l(\mathbf{x}, y) = I[y \geq h_l(\mathbf{x})] \tag{2.33}$$

  intentando así resolver los problemas binarios previamente descritos a partir del valor de $h_l(\mathbf{x})$.

  Por último, igual que se hizo para los problemas de clasificación, también se asume una distribución D (normalmente será uniforme, i.e., $D(i) = 1/N$) sobre los puntos del conjunto de entrenamiento, la cual se va modificando según los errores que se cometan a través de los pesos w_i^l asignados a cada punto $\mathbf{x}_i$ en cada iteración l. En AdaBoost.R, esta parte también se ve modificada, mapeando esa distribución a otra densidad sobre los pares (i, y) de tal manera que reducir el error de clasificación en el problema convertido sea equivalente a reducir el error cuadrático medio en el problema original. Así, estos pesos $w^l(i, y)$ son los definidos en el punto 1 de la descripción del algoritmo.

  Si calculamos el error binario del modelo base transformado $\tilde{h}_l : \mathbb{R}^p \times [0, 1] \to \{0, 1\}$ con respecto a la densidad modificada a la que dan lugar los pesos $w^l(i, y)$, se obtiene, como mencionamos anteriormente, que es directamente proporcional a calcular el error cuadrático medio del problema de regresión original:

$$\begin{aligned} \sum_{i=1}^{N}\int_0^1 \left|\tilde{y}_{i,y} - \tilde{h}\left(\tilde{x}_{i,y}\right)\right| \tilde{D}(i,y)dy &= \frac{1}{Z}\sum_{i=1}^{N} D(i)\left|\int_{y_i}^{h(\mathbf{x}_i)}\right| y - y_i|dy| \\ &= \frac{1}{2Z}\sum_{i=1}^{N} D(i)\left(h\left(\mathbf{x}_i\right) - y_i\right)^2 \end{aligned} \tag{2.34}$$

  en donde la constante de proporcionalidad es $Z/2$, con Z definido en el punto 1 del algoritmo.

  Así, cada modelo base débil $\tilde{h}(\mathbf{x}, y)$ que usamos para resolver el problema transformado es una función no decreciente sobre y. Por lo tanto, la hipótesis ensamblada $\tilde{h}_{\texttt{ada}}$ generada por AdaBoost

Algoritmo 3 Algoritmo AdaBoost.M1 para problemas de clasificación multiclase

Entrada: Conjunto de entrenamiento $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, con $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \mathcal{G} = \{\mathcal{G}_1, ..., \mathcal{G}_K\}$. Número de clasificadores base L.

Salida: Un clasificador $h_{\texttt{ada.m1}} : \mathbb{R}^p \to \mathcal{G}$.

1. Se inicializan los pesos asignados a cada dato $\mathbf{x}_i \in S$: $w_i^1 = \dfrac{1}{N},\ i = 1, \ldots, N$.
2. Para $l = 1, \ldots, L$, se calcula:
   a) Se ajusta un clasificador base débil $h_l(\mathbf{x})$ al conjunto de entrenamiento S utilizando los pesos actuales w_i^l. Este modelo base tendrá como error $\mathbb{P}_{i\sim w_i^l}[h_l(\mathbf{x}_i) \neq y_i]$, en donde cabe destacar que el cálculo se hace con respecto a la distribución w_i^l sobre la que se entrenó el modelo (i.e., teniendo en cuenta los pesos que cada $\mathbf{x}_i$ tiene en S). En la práctica, esto significa que el algoritmo base usará en la función de coste los pesos w_i^l. En caso de no ser esto posible, también se puede hacer mediante un muestreo de S con reemplazamiento acorde a la distribución de los pesos, utilizando el conjunto de datos resultante para entrenar el modelo.
   b) Se calcula:
   $$\epsilon_l = \sum_{i=1}^{N} w_i^l I(y_i \neq h_l(\mathbf{x}_i))$$
   obteniendo el error de clasificación cometido por el modelo h_l ponderado por los pesos de cada $\mathbf{x}_i$.
   c) Si el error ϵ_l es 0 o mayor o igual que 0.5, entonces reinciamos los pesos al valor inicial ($w_i^l = \dfrac{1}{N},\ i = 1, \ldots, N$), y continuamos. En caso de que $\epsilon_l \in (0, 0.5)$, calcularíamos:
   $$\beta_l = \frac{\epsilon_l}{1 - \epsilon_l}.$$
   d) Se recalculan los pesos a utilizar en la siguiente iteración para cada punto del conjunto de entrenamiento S:
   $$w_i^{l+1} = \frac{w_i^l \cdot \beta_l^{(1-I(y_i \neq h_l(\mathbf{x}_i)))}}{Z_l},\ i = 1, 2, \ldots, N$$
   en donde $Z_l = \sum_{i=1}^{N} w_i^l \cdot \beta_l^{(1-I(y_i \neq h_l(\mathbf{x}_i)))}$ es un factor normalizador.
3. Fase de clasificación. Dado un $\mathbf{x}$ a clasificar, calculamos el soporte que AdaBoost.M1 asocia a cada clase $\mathcal{G}_k$:
   $$\mu_k(\mathbf{x}) = \sum_{h_l(\mathbf{x}) = \mathcal{G}_k} \log\left(\frac{1}{\beta_l}\right). \tag{2.30}$$
   Es decir, a la hora de calcular el soporte para la clase $\mathcal{G}_k$, se tendrán en cuenta únicamente los modelos h_l los cuales predicen dicha clase para $\mathbf{x}$.
4. Por último, se clasificará $\mathbf{x}$ en la clase $y = \mathcal{G}_k$ para la cual el soporte μ_k sea el máximo. Es decir:
   $$h_{\texttt{ada.m1}}(\mathbf{x}) = \arg\max_{y \in \mathcal{G}} \sum_{l=1}^{L} \log\left(\frac{1}{\beta_l}\right). \tag{2.31}$$

Algoritmo 4 Algoritmo AdaBoost.M2 para problemas de clasificación multiclase

Entrada: Conjunto de entrenamiento $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, con $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in \mathcal{G} = \{\mathcal{G}_1, ..., \mathcal{G}_K\}$. Número de clasificadores base L.

Salida: Un clasificador $h_{\texttt{ada.m2}} : \mathbb{R}^p \to \mathcal{G}$.

1. Se comienza definiendo el conjunto B de todos los pares de puntos $\mathbf{x}$ y clases y tal que la clase correcta de $\mathbf{x}$ no es y. Es decir, los pares (i, y) en donde el punto S con índice i ($\mathbf{x}_i$) no tiene como clase $y \in \mathcal{G}$. Así, se define $B = \{(i, y) | i \in \{1, ..., N\}, y \neq y_i\}$.
2. Se inicializan los pesos, en este caso habiendo uno para cada dato $\mathbf{x}_i \in S$ y clase $y \in \mathcal{G}$:
   $w^1(i, y) = \dfrac{1}{|B|}$, para todo $(i, y) \in B$.
3. Para $l = 1, \ldots, L$, se calcula:
   a) Se ajusta un clasificador base débil $h_l(\mathbf{x})$ al conjunto de entrenamiento S utilizando los pesos actuales $w^l(i, y)$. En este caso, a diferencia del AdaBoost original, la definición de cada modelo base es $h : \mathbb{R}^p \times \mathcal{G} \to [0, 1]$, por lo que los modelos base devolverán más información (y no solo la clase predicha). Intuitivamente, se puede interpretar cada par (i, y) como si se le estuviera pidiendo al algoritmo que escogiese entre la clase y_i (la correcta para $\mathbf{x}_i$) o y (una de las incorrectas). Así, en este contexto el peso $w^l(i, y)$ sería la importancia que tendría en nuestro problema, para $\mathbf{x}_i$, distinguir entre la clase incorrecta y y la correcta y_i.
   b) Se calcula el pseudo-error con la siguiente fórmula, cuya derivación se puede ver en Freund y Schapire (1997):
   $$\epsilon_l = \frac{1}{2} \sum_{(i,y) \in B} w^l(i, y)\big(1 - h_l(\mathbf{x}_i, y_i) + h_l(\mathbf{x}_i, y)\big).$$
   c) Se calcula $\beta_l = \dfrac{\epsilon_l}{1 - \epsilon_l}$.
   d) Se recalculan los pesos a utilizar en la siguiente iteración para cada punto del conjunto de entrenamiento S:
   $$w^{l+1}(i, y) = \frac{w^l(i, y)}{Z_l} \cdot \beta_l^{(1/2)(1 - h_l(\mathbf{x}_i, y_i) + h_l(\mathbf{x}_i, y))}, \ i = 1, 2, \ldots, N$$
   en donde $Z_l = \sum_{i=1}^{N} w_i^l(i, y) \cdot \beta_l^{(1/2)(1 - h_l(\mathbf{x}_i, y_i) + h_l(\mathbf{x}_i, y))}$ es un factor normalizador.
4. Fase de clasificación. Dado un $\mathbf{x}$ a clasificar, se asociaría la clase y (equivalentemente, $\mathcal{G}_k$) que maximice la siguiente expresión:
   $$h_{\texttt{ada.m2}}(\mathbf{x}) = \arg\max_{y \in \mathcal{G}} \sum_{l=1}^{L} \log\left(\frac{1}{\beta_l}\right) h_l(\mathbf{x}, y). \tag{2.32}$$

en el problema de clasificación transformado también será no decreciente. Como la salida de $\tilde{h}_{\texttt{ada}}$ es binaria, esto implicará que para todo $\mathbf{x}$ habrá únicamente un y para el cual $\tilde{h}_{\texttt{ada}}(\mathbf{x}, y) = 0$ para todo $y' < y$ y $\tilde{h}_{\texttt{ada}}(\mathbf{x}, y) = 1$ para todo $y' > y$. Este, como se muestra en el apartado 3 de la definición del algoritmo, será el valor y que devolverá $h_{\texttt{ada.R}}(\mathbf{x})$.

Es de interés comentar que aunque la cantidad de pesos $w^l(i, y)$ es incontable, se puede observar que, si lo vemos como función de y, es una función lineal a trozos. Si tenemos $l = 1$ tenemos dos trozos lineales, y conforme aumentamos l se subdividirá uno de los trozos en dos (en el punto $h_l(\mathbf{x}_i)$), y así sucesivamente. Así, guardar y actualizar estas funciones lineales a trozos, así como el cálculo de las integrales en las que están involucradas en la defínción del algoritmo, se vuelve plausible.

Análogamente a lo descrito para AdaBoost y AdaBoost.M2, el siguiente teorema describe la cota superior del error (en concreto, error cuadrátivo medio) para AdaBoost.R en problemas de regresión.

Teorema 2.3 *Sea $h_{ada.R}$ el modelo de regresión ensamblado devuelto por el algoritmo AdaBoost.R, y sean $\epsilon_l, ..., \epsilon_L$ los errores de los modelos base tal y como se definen en 2.c del Algoritmo 5. Entonces, el error cuadrático de entrenamiento de AdaBoost.R, $\epsilon_{ada.R}$, está acotado por*

$$\epsilon_{ada.R} = \frac{1}{N}\sum_{i=1}^{N}(y_i - h_{ada}(\boldsymbol{x}_i)^2 \leq e^{-2\sum_{l=1}^{L}\gamma_l^2}. \tag{2.35}$$

Boosting como modelo aditivo

Como se comentó al principio de la sección, es habitual en las técnicas de boosting usar como modelos base árboles de decisión, y más en concreto, árboles en donde el parámetro $d = 1$. En este caso, cada árbol únicamente utilizará una de las variables de $\mathbf{x}$, y por lo tanto se puede ver que el ensamblado AdaBoost (2.21) es equivalente a ajustar un modelo aditivo. De manera más general, los modelos de expansiones aditivas de funciones base tienen la forma:

$$h(\mathbf{x}) = \sum_{l=1}^{L}\beta_l b(\mathbf{x}; \gamma_l) \tag{2.36}$$

en donde los β_l se denominan los coeficientes de expansión, y $b(x; \gamma_l)$ son funciones base típicamente simples caracterizadas por el conjunto de parámetros γ_l.

Muchas de las técnicas de aprendizaje estadístico se pueden escribir de esta forma. Por ejemplo, las redes neuronales con una única capa intermedia. En estas, $b(x; \gamma) = \sigma(\gamma_0 + \gamma_1^T x)$, donde $\sigma(z) = \frac{1}{1+\exp(-z)}$ es la función sigmoidal, y γ parametrizan una combinación lineal del dato de entrada $\mathbf{x}$.

Usualmente estos modelos dan lugar a minimizar una función de pérdida sobre los datos de la forma:

$$\min_{\{\beta_l, \gamma_l\}_1^L} \sum_{i=1}^{N} \mathbf{L}\left(y_i, \sum_{l=1}^{L}\beta_l b(\mathbf{x}_i; \gamma_l)\right). \tag{2.37}$$

Para muchas funciones de pérdida $\mathbf{L}(y, h(\mathbf{x}))$ y funciones base $b(\mathbf{x}; \gamma)$, resolver el anterior problema se vuelve computacionalmente muy costoso. Por lo tanto, dado que resolver el subproblema de ajustar una única función base:

$$\min_{\beta, \gamma} \sum_{i=1}^{N} \mathbf{L}\left(y_i, \beta b(\mathbf{x}_i; \gamma)\right) \tag{2.38}$$

Algoritmo 5 Algoritmo AdaBoost.R para problemas de regresión.

Entrada: Conjunto de entrenamiento $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, con $\mathbf{x}_i \in \mathbb{R}^p$, $y_i \in [0, 1]$.
Número de modelos base L.

Salida: Un modelo $h_{\texttt{ada.R}} : \mathbb{R}^p \to [0, 1]$

1. Se inicializa el vector de pesos:

$$w^1(i, y) = \frac{|y - y_i|}{NZ}, \text{ para todo } i = 1, ..., N \text{ e } y \in [0, 1].$$

en donde $Z = \sum_{i=1}^{N} \int_0^1 |y - y_i| \, dy$, y $D(i) = 1/N$.

2. Para $l = 1, \ldots, L$, se calcula:

   a) Se calcula:

$$p^l(i, y) = \frac{w^l(i, y)}{\sum_{i=1}^{N} \int_0^1 w^l(i, y) dy}.$$

   b) Se entrenan los modelo base haciendo uso de la distribución (o pesos) p^l calculada en el paso anterior. Esto nos devolverá un modelo base $h_l : \mathbb{R}^p \to [0, 1]$.

   c) Calculamos el error de h_l con la siguiente expresión:

$$\epsilon_l = \sum_{i=1}^{N} \left| \int_{y_i}^{h_l(\mathbf{x}_i)} p^l(i, y) dy \right|.$$

   Si $\epsilon_l > 1/2$, entonces no se continua y se sale del bucle.

   d) Se calcula $\beta = \epsilon_l / (1 - \epsilon_l)$.

   e) Se actualizan los pesos para la siguiente iteración:

$$w^l(i, y) = \begin{cases} w^l(i, y) & \text{si } y_i \leqslant y \leqslant h_l(\mathbf{x}_i) \text{ o } h_l(\mathbf{x}_i) \leqslant y \leqslant y_i \\ w^l(i, y)\beta_t & \text{de otra forma.} \end{cases}$$

   para todo $i = 1, ..., N$, $y \in [0, 1]$.

3. Fase de predicción. Se devuelve la hipótesis ensamblada final:

$$h_{\texttt{ada.R}}(\mathbf{x}) = \text{ínf} \left\{ y \in Y : \sum_{t: h_l(\mathbf{x}) \leqslant y} \log(1/\beta_t) \geqslant \frac{1}{2} \sum_t \log(1/\beta_t) \right\}.$$

es mucho más sencillo, en la práctica la expresión (2.37) se aproxima mediante el modelado progresivo por etapas (o *forward stagewise modeling*). Este va secuencialmente añadiendo nuevas funciones base a la expansión y entrenando únicamente los parámetros de la última, no teniendo que ajustar así las que ya están añadidas. Por ejemplo, para el caso de regresión con función de pérdida $\mathbf{L}(y, h(\mathbf{x})) = (y - h(\mathbf{x}))^2$, se iniciaría el procedimiento minimizando el subproblema con una única función base, y a partir de ahí se irían incluyendo más bases, minimizando, recurrentemente, la expresión:

$$\mathbf{L}(y_i, h_{l-1}(\mathbf{x}_i) + \beta b(\mathbf{x}_i; \gamma)) = (y_i - h_{l-1}(\mathbf{x}_i) - \beta b(\mathbf{x}_i; \gamma)). \tag{2.39}$$

Es decir, que una vez obtenida h_{l-1}, se ajustaría el problema de minimización al que llevaría (2.39), añadiendo posteriormente a h_{l-1} el término $\beta_l b(\mathbf{x}_i; \gamma_l)$ resultante de la minimización.

Aunque el algoritmo AdaBoost fue desarrollado de manera independiente a los modelos de expansiones aditivas, unos años después del trabajo original de Freund y Schapire se demostró que AdaBoost es equivalente a entrenar un modelo con el método de modelado progresivo por etapas utilizando como función de pérdida:

$$\mathbf{L}(y, h(\mathbf{x})) = \exp(-yh(\mathbf{x})) \tag{2.40}$$

y cuya prueba se puede consultar en Friedman et al (2000).

Este descubrimiento, poniendo a AdaBoost en el marco de una familia de modelos mucho más general, facilitó en gran medida el estudio de sus propiedades. Además, motivó la creación de un grupo de técnicas de boosting más genéricas conocidas como *gradient boosting*, las cuales son variaciones de AdaBoost que intentan minimizar (2.37) utilizando distintas funciones de pérdida.

No obstante, las facilidades a la hora de recalcular pesos descritas en (2.19) y (2.20) vienen dadas implícitamente por la utilización de la función de pérdida exponencial, y estas no tienen una derivación análoga en el caso de utilizar otras funciones $\mathbf{L}(y, h(\mathbf{x}))$. En esos otros casos, se tendrá que resolver el problema de minimización (2.37) (que puede o no ser convexo) utilizando técnicas del descenso del gradiente, y de ahí el nombre *gradient boosting*. Para llevar a cabo esta minimización, se tiene que la pérdida asociada a usar $h(\mathbf{x})$ para predecir y en el conjunto de entrenamiento es:

$$\mathbf{L}(h) = \sum_{i=1}^{N} \mathbf{L}(y_i, h(\mathbf{x}_i))$$

donde el objetivo es minimizar $\mathbf{L}(h)$ con respecto a f, el cual se puede expresar como la siguiente optimización numérica:

$$h^* = \arg\min_h \mathbf{L}(h)$$

donde los parámetros h son la aproximación que hace $h(\mathbf{x})$ en cada punto de S, i.e., $h = [h(\mathbf{x}_1), \ldots, h(\mathbf{x}_N)]$. Esta expresión se podrá resolver numericamente con un procedimiento iterativo, en donde comenzando con un valor inicial h_0, se van añadiendo términos h_l, de la siguiente forma:

$$h_L = \sum_{l=0}^{L} h_l, \;\; h_l \in \mathbb{R}^N.$$

Los distintos métodos iterativos se diferencian en cómo dan lugar al vector incremental h_l de cada iteración, siendo uno de los más populares la técnica del descenso del gradiente, donde $h_l = -\rho_l g_l$,

siendo $g_l \in \mathbb{R}^N$ el gradiente de $\mathbf{L}(h)$ evaluado en h_{l-1}:

$$g_{il} = \left[\frac{\partial \mathbf{L}(y_i, h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)}\right]_{h(\mathbf{x}_i)=h_{l-1}(\mathbf{x}_i)} \tag{2.41}$$

y ρ_l un escalar que es solución de:

$$\rho_l = \arg\min_\rho \mathbf{L}(h_{l-1} - \rho_l g_l)$$

quedando así la actualización de f en el paso l como:

$$h_l = h_{l-1} - \rho_l g_l. \tag{2.42}$$

Como se puede ver, la expresión iterativa para resolver el problema genérico a minimizar en (2.42) es similar a la estructura de nuestro modelo aditivo presentado en (2.36). Esta similitud se refleja en el algoritmo genérico de *gradient tree boosting* (cuya implementación original se denomina MART), que se define en el Algoritmo 6.

Algoritmo 6 Algoritmo Gradient Tree Boosting

1. Se inicializa el modelo con un valor constante: $h_0(\mathbf{x}) = \arg\min_\gamma \sum_{i=1}^N \mathbf{L}(y_i, \gamma)$.
2. Para $l = 1, \ldots, L$, se calcula:
   a) Para cada $i = 1, \ldots, N$, se calcula:
   $$r_{il} = -\left[\frac{\partial \mathbf{L}(y_i, h(\mathbf{x}_i))}{\partial h(\mathbf{x}_i)}\right]_{h(\mathbf{x}_i)=h_{l-1}(\mathbf{x}_i)}. \tag{2.43}$$
   también conocidos como pseudo-residuos, ya que en el caso de que $\mathbf{L}$ sea el error cuadrático medio, coincide con el residuo $y_i - h(\mathbf{x}_i)$.
   b) Se ajusta un árbol para los pseudo-residuos r_{il}, dando lugar a las regiones terminales $R_{jl}, j = 1, \ldots, J_l$.
   c) Para cada $j = 1, \ldots, J_l$, se calcula:
   $$\gamma_{jl} = \arg\min_\gamma \sum_{\mathbf{x}_i \in R_{jl}} \mathbf{L}(y_i, h_{l-1}(\mathbf{x}_i) + \gamma). \tag{2.44}$$
   d) Se actualiza la función:
   $$h_l = h_{l-1} + \sum_{j=1}^{J_N} \gamma_{jl} I(\mathbf{x}_i \in R_{jl}). \tag{2.45}$$
3. El modelo ensamblado de *gradient tree boosting* será $h(\mathbf{x}) = h_L(\mathbf{x})$.

Para adaptar este modelo al problema de clasificación se repiten los pasos del punto 2 del Algoritmo 6 K veces, una por cada una de las clases posibles en el conjunto $\mathcal{G}$. Así, en el punto 3 tendríamos K ensamblados distintos h_{kL}, en vez de un único h_L. Las predicciones de esos K modelos se podrían convertir a probabilidades asociadas con cada clase $\mathcal{G}_k$ mediante la expresión:

$$p_k(\mathbf{x}) = \frac{\exp(h_k(\mathbf{x}))}{\sum_{c=1}^K \exp(h_c(\mathbf{x})} \tag{2.46}$$

para, posteriormente, crear la regla de clasificación f que escoja la clase $\mathcal{G}_k$ con mayor probabilidad:

$$h(\mathbf{x}) = \mathcal{G}_k \text{ donde } k = \arg\max_c p_c(\mathbf{x}). \tag{2.47}$$

AdaBoost en la práctica

En la práctica AdaBoost es una técnica popular por múltiples razones, algunas de ellas ya explicadas con más detalle en secciones anteriores:

- No requiere de hiperparámetros complicados de configurar.
- No es necesario saber cual es la ventaja γ que tiene el algoritmo débil usado para generar los clasificadores en el paso 2.a del Algoritmo 2.
- Es un algoritmo computacionalmente eficiente (Freund y Schapire, 1997).
- Tiene un buen poder de generalización y suele evitar el sobreajuste.
- Existen propiedades teóricas interesantes sobre el comportamiento del error de entrenamiento y error de generalización.

No obstante, no es una técnica perfecta para todo tipo de situaciones. El principal problema en la práctica de AdaBoost es su gran sensibilidad ante datos con ruido, los cuales son muy habituales en problemas con datos reales. Esta sensibilidad es generalmente atribuida a la función de pérdida exponencial que se está minimizando (2.40), la cual hace que si una instancia $\mathbf{x}_i$ no es clasificada en su clase correcta y_i entonces su peso para la siguiente iteración aumentará drásticamente. Por ello, cuando por motivos de ruido una instancia está asociada a una clase errónea en el conjunto de entrenamiento, AdaBoost puede que intente forzar su clasificación de forma correcta, degenerando así el poder de generalización del modelo final.

A diferencia de las técnicas bagging, boosting tiene como objetivo mejorar tanto la varianza como el sesgo de un clasificador base. Para visualizar en la práctica la capacidad que estas metodologías tienen de mejorar las predicciones de un modelo base débil, y basándose en Hastie et al. (2008), se genera un conjunto de entrenamiento S con $x = (X_1, \ldots, X_{10}) \in \mathbb{R}^{10}$ cuyas variables son normales estándar idénticamente distribuidas e independientes, y con su clase $y \in \{-1, 1\}$ generada por la siguiente expresión:

$$y = \begin{cases} 1 & \text{si } \sum_{j=1}^{10} X_j^2 > \chi_{10}^2(0.5) \\ -1 & \text{en otro caso} \end{cases} \tag{2.48}$$

en donde $\chi_{10}^2(0.5)$ es la mediana de la variable aleatoria chi-cuadrado con 10 grados de libertad a la que da lugar la suma de los cuadrados de 10 normales estándar. Se genera un conjunto de entrenamiento con $N = 2000$ y uno de test con $N = 10000$.

Los resultados de aplicar AdaBoost con cotones (árboles con $d = 1$) se pueden observar en la Figura 2.4. Asimismo, se tiene que el error de test esperado de utilizar un único cotón es alrededor del 46 %, siendo solo ligeramente mejor que un clasificador que prediga la clase binaria de manera aleatoria. Conforme aumentan el número L de árboles añadidos al modelo, se puede ver como el error baja considerablemente, siendo menor incluso que el error de test medio cometido por un árbol grande ($d = 244$).

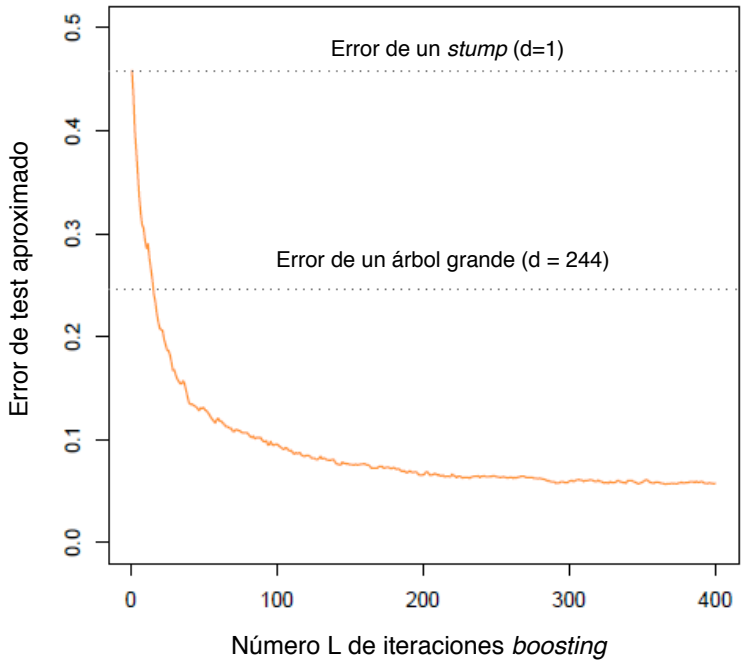

Figura 2.4: Error de test en función de L al ajustar AdaBoost a los datos generados en la expresión (2.48).

Regularización en las técnicas de boosting

Los parámetros más relevantes a la hora de controlar el posible sobreajuste de los datos en un modelo de boosting son el tamaño permitido de árbol d, y el número L de iteraciones boosting utilizadas. Con respecto a d, este es el parámetro que controla la iteración que se permite entre variables en el modelo. En el caso particular con $d = 1$, construyendo así cotones, este método da lugar a un modelo aditivo. Este parámetro se suele ajustar por validación cruzada, aunque a lo largo de la literatura se recomienda usar valores entre 4 y 8, siendo $d = 1$ insuficiente en la práctica para muchos problemas, y pudiendo ocasionar la selección de valores para d demasiado grandes un sobreajuste de los datos.

Por otro lado, el parámetro L también es de vital importancia a la hora de evitar el sobreajuste, dado que un L demasiado grande puede dar lugar a un ajuste de los datos de entrenamiento demasiado bueno, perdiendo así capacidad de generalización. Lo habitual en la práctica es monitorear el error de test aproximado conforme se va entrenando el algoritmo, escogiendo finalmente el valor que minimiza dicho error.

Controlar estos dos parámetros, sin embargo, no son las únicas maneras con las cuales se puede evitar el sobreajuste de un ensamblado de árboles tipo boosting, pudiendo utilizar también las técnicas de regularización vistas en el capítulo 1. La técnica más habitual en la práctica es añadir un coeficiente de contracción $0 < \lambda < 1$ a la expresión (2.45) del algoritmo *gradient boosting trees* que controlará el ratio de aprendizaje del procedimiento boosting (análogo a los parámetros α_l que ponderaban los modelos en AdaBoost). En Friedman (2001) se muestra empíricamente como valores pequeños de λ dan lugar, en general, a mejorar el error de test, a expensas de requerir un mayor valor L. El mismo autor recomienda seleccionar valores de $\lambda < 0.1$ al mismo tiempo que se monitorea el error de test conforme aumenta L, para detener el algoritmo antes de dar lugar a sobreajuste (técnica conocida como *early stopping*).

2.3.7. Stacking

Las técnicas de *stacking* son un procedimiento general para ensamblar modelos base (también conocidos como modelos de primer nivel) a través de entrenar un meta-modelo (o modelo de segundo nivel) que tendrá como objetivo encontrar una manera óptima de combinar la salida de los modelos individuales.

En esta técnica, presentada en el Algoritmo 7, parte de entrenar una serie de L modelos base sobre el conjunto de entrenamiento S. A priori, no hay ningún tipo de restricción sobre qué modelos se pueden usar, siendo habitual usar un conjunto de 4-10 modelos de distinta naturaleza: un regresor lineal, una máquina de vectores de soporte, una red neuronal, etc. La clave, como se vio al inicio del texto, es dar lugar a modelos variados, con predicciones lo más incorreladas posible entre ellos.

Una vez se tienen los nuevos L modelos, se crea un nuevo conjunto de entrenamiento S' con N pares de puntos $(\mathbf{z}_i, y_i)$, en donde z_i será el resultado de aplicar los L modelos de primer nivel sobre $\mathbf{x}_i$ $(z_i = [h_1(\mathbf{x}_i, ..., h_L(\mathbf{x}_i)])$, y la variable respuesta y_i queda fija.

A continuación, se entrena un meta-modelo $h_{\texttt{meta}}$ sobre el nuevo conjunto S', cuyo objetivo es encontrar patrones entre los resultados de los modelos base $\mathbf{z}$ y la respuesta y para así dar con la forma óptima de combinarlos.

Finalmente, para predecir la respuesta de un nuevo punto $\mathbf{x}$ simplemente habrá que primero obtener el vector de salidas con el resultado de los L modelos base, y pasárselo al meta-modelo para dar la respuesta final:

$$h_{\texttt{stack}}(\mathbf{x}) = h_{\texttt{meta}}\big(h_1(\mathbf{x}), ..., h_L(\mathbf{x})\big).$$

Cabe destacar que, en la fase 2 del Algoritmo 7 donde damos lugar al nuevo conjunto S', se está usando los mismos datos tanto para entrenar cada modelo h_l como para generar los z_{il}. Esto puede ocasionar problemas de sobreajuste, por lo que en la literatura se suele recomendar que las instancias que se usan para generar z_{il} no hayan sido usadas entrenando h_l.

Habitualmente esto se consigue aplicando la validación cruzada de R iteraciónes previamente definida en 1.8. Se comienza dividiendo el conjunto de entrenamiento S en R subconjuntos $S_1, ..., S_R$, donde definimos $S_{(-r)}$ como $S \backslash S_r$. Así, para cada r, obtendremos un modelo base $h_l^{(-r)}$ entrenado sobre $S_{(-r)}$, con el cual usaremos los elementos de S_r (conjunto de test) para crear los valores $z_{il} = h_l^{(-r)}(\mathbf{x}_i)$ para todo $\mathbf{x}_i \in S_r$. Al realizar el proceso de validación cruzada al completo para todo r, modelo l y instancia $\mathbf{x}_i$, obtendremos el nuevo conjunto S':

$$S' = \{((z_{i1}, ..., z_{iL}), y_i)\}_{i=1}^{N}$$

en donde la generación de cada z_{il} y el entreno de h_l fueron independientes.

Algoritmo 7 Algoritmo genérico de stacking, tanto para clasificación como regresión.

Entrada: Conjunto de entrenamiento $S = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, con $\mathbf{x}_i \in \mathbb{R}^p$.
En donde y_i variable continua o categórica.
L algoritmos $\mathcal{A}_1, ..., \mathcal{A}_L$ (por ejemplo, regresor lineal, un árbol de decisión, etc.).
Algoritmo para entrenar el meta-modelo $\mathcal{A}_{\texttt{meta}}$.

Salida: Un modelo h_{stack} que ensambla a los L modelos base haciendo uso del meta-modelo.

1. Se entrena L modelos base haciendo uso de los L algoritmos disponibles:

$$h_l = \mathcal{A}_l(S), \;\; l = 1, \ldots, L.$$

2. Se genera un nuevo conjunto de entrenamiento S' con las salidas de los modelos h_l al aplicarlos sobre S. Es decir, para cada $i = 1, ..., N$ y para cada $l = 1, ..., L$, se calcula:

$$z_{ij} = h_l(\mathbf{x}_i)$$

dando lugar así a un nuevo conjunto $S' = \{((z_{i1}, ..., z_{iL}), y_i)\}_{i=1}^{N}$ en donde tenemos N pares de datos cuya primera componente son las L salidas de los modelos h_l al aplicarles $\mathbf{x}_i$, y la segunda componente será la variable respuesta y_i inicial.

3. Se entrena el algoritmo $\mathcal{A}_{\texttt{meta}}$ sobre S' para dar lugar a un meta-modelo óptimo que ensamble los L modelo base.

$$h_{\texttt{meta}} = \mathcal{A}_{\texttt{meta}}(S').$$

4. Fase de predicción / clasificación: Dado un nuevo $\mathbf{x}$, la salida del algoritmo de stacking será:

$$h_{\texttt{stack}}(\mathbf{x}) = h_{\texttt{meta}}\big(h_1(\mathbf{x}), ..., h_L(\mathbf{x})\big).$$

Capítulo 3

Ensamblados estadísticos en la práctica

El objetivo de este capítulo es presentar varios problemas con datos reales, entrenar distintos modelos (simples y ensamblados) sobre los mismos, y comparar los resultados obtenidos. Todas las pruebas fueron realizadas con el lenguaje de programación Python, haciendo uso mayoritariamente de la librería `sklearn` (Pedregosa et al, 2011). El código utilizado se puede ver en el Apéndice B.

3.1. Introducción

Una vez tenemos un modelo h entrenado sobre un conjunto de entrenamiento S, el último paso a realizar es comprobar su rendimiento sobre un conjunto de datos de test (recomendablemente no usados en la fase de entrenamiento), para así conocer su poder de generalización. En esta sección se presentan las métricas más habituales a la hora de llevar a cabo esta fase evaluación.

		Suceso Pedicho	
	Población total	Predicción Positiva	Predicción Negativa
Suceso Real	Suceso Positivo	Positivo Real = PR	Falso Negativo = FN (Error tipo II)
	Suceso Negativo	Falso Positivo = FP (Error tipo I)	Negativo Real = NR

Tabla 3.1: Definición de matriz de confusión para un problema binario, en donde se codifica la clase positiva como $+1$ y la negativa como -1.

Métricas para evaluar problemas de clasificación binarios

En el caso de estar ante un problema de clasificación binario, uno de los métodos más habituales es usar la matriz de confusión del modelo, definida en la Tabla 3.1. A partir de sus elementos y notación se pueden definir las siguientes métricas:

- **Porcentaje de acierto de clasificación**. Equivalente al riesgo empírico de clasificación usando como pérdida la función indicador I introducida en (1.15). Es la métrica más habitual a la hora de evaluar el rendimiento de un clasificador:

$$\%\text{ de Acierto} = \frac{PR + NR}{PR + NR + FN + FP}.$$

  No obstante, destacar que un % de acierto de clasificación alto no tiene por qué indicar que el modelo tiene una buena capacidad de generalización, sobre todo cuando estamos ante un problema con clases muy desbalanceadas. Por ejemplo, de tener un problema binario en donde la clase 1 tiene una representación mucho mayor (+99 %) que la clase -1 sobre el conjunto S, un clasificador constante que siempre clasifique un nuevo punto $\mathbf{x}$ como 1 ($h_{\mathbf{x}} = 1\ \forall \mathbf{x}$) tendría un acierto mayor del 99 %, aunque su capacidad de generalización será obviamente mala sobre la clase infrarepresentada.

- **Precisión**. Se define como la proporción de observaciones positivas que predecimos correctamente sobre el total de predicciones positivas que hace el modelo. Intenta responder a la pregunta: ¿Qué proporción de observaciones predichas como positivas eran realmente positivas?:

$$\text{Precisión} = \frac{PR}{PR + FP}.$$

  Por ejemplo, en el caso anterior desbalanceado, la clase con una representación mayor tendría una precisión elevada, mientras que la infrarepresentada tendría una precisión cercana al 0 %.

- **Sensibilidad** (o *recall*): Se define como la proporción de observaciones positivas que predecimos correctamente sobre el total de observaciones positivas. Intenta responder a la pregunta: ¿Qué proporción de positivos reales fue clasificado correctamente?:

$$\text{Sensibilidad} = \frac{PR}{PR + FN}.$$

- **Valor $\mathbf{F}_1$**. Se define como la media armónica de la precisión y la sensibilidad:

$$\text{Valor } \mathbf{F_1} = 2 \cdot \frac{\text{Precisión} \cdot \text{Sensibilidad}}{\text{Precisión} + \text{Sensibilidad}} = \frac{2 \cdot \text{PR}}{2 \cdot \text{PR} + \text{FP} + \text{FN}}. \tag{3.1}$$

  El valor $\mathbf{F}_1$ suele ser una métrica recomendada cuando estamos ante problemas desbalanceados, al tener en cuenta tanto la precisión como la sensibilidad. Además, el hecho de que se use la media armónica hace que el valor de la métrica baje significativamente si la precisión o sensibilidad son muy bajas.

- **Coeficiente de correlación de Mattheus** (CCM):

$$\text{CCM} = \frac{\text{PR} \cdot \text{NR} - \text{FP} \cdot \text{FN}}{\sqrt{(\text{PR} + \text{FN})(\text{PR} + \text{FP})(\text{NR} + \text{FN})(\text{NR} + \text{FP})}}. \tag{3.2}$$

  También conocido como **coeficiente phi de Pearson**, el CCM para el caso binario es una métrica que calcula la correlación entre el vector de clases reales y predichas, dando así un valor entre -1 y 1. El CCM tiene en cuenta tanto los falsos positivos como los falsos negativos, por lo que se puede usar como métrica para problemas desbalanceados.

		Suceso Predicho		
	Población Total	Clase 1	...	Clase K
Suceso Real	Clase 1	N_{11}	...	N_{1K}
	...	N_{k1}	N_{kk}	N_{kK}
	Clase K	N_{K1}	...	N_{KK}

Tabla 3.2: Ejemplo de matriz de confusión para un problema multiclase, i.e., $K > 2$.

Así, dependiendo del problema concreto a tratar y de la importancia que en él tenga cada tipo de error, será de mayor interés optimizar nuestro modelo para maximizar unas métricas u otras. Para entender esto de manera más visual, supongamos un problema de clasificación binaria en donde queremos clasificar un tumor (en base a su vector de características $\mathbf{x}$) como maligno ($+1$) o benigno (-1). Supóngase que, entrenado un modelo h, se obtiene la siguiente matriz de confusión:

$$\begin{pmatrix} \text{PR} = 2 & \text{FN} = 15 \\ \text{FP} = 3 & \text{NR} = 180 \end{pmatrix}$$

Por un lado, si nos fijásemos en el % de acierto del algoritmo, obtendríamos $182/200 = 0.91$, lo cual nos podría dar a pensar que tenemos un buen clasificador para el problema dado. No obstante, en este problema podemos ver como las clases están desbalanceadas, y además, los dos tipos de error no tienen la misma criticidad: decirle a un paciente con un tumor maligno que el tumor que tiene es benigno sería más crítico que a la inversa.

En este caso, si calculamos la precisión obtendríamos $2/(2+3) = 0.4$, lo que nos dice que para los tumores que se han predicho como malignos por el modelo, estos son efectivamente malignos el 40 % de las veces. Por otro lado, la sensibilidad nos daría $2/(2+15) = 0.12$, diciéndonos que si un tumor es realmente maligno, nuestro modelo solo lo predice como tal el 12 % de las veces. Este es un claro ejemplo de modelo inaceptable en la practica que, de haber usado únicamente la métrica de % de acierto para evaluarlo, podríamos haber pensado que era un buen candidato para ser usado.

Métricas para evaluar problemas de clasificación multiclase

Como se puede observar, las métricas de precisión, sensibilidad y valor $\mathbf{F_1}$ fueron pensadas originalmente para problemas de clasificación binarios. Sin embargo, estas se pueden adaptar para el caso de problemas con más de dos clases ($K > 2$) haciendo uso de la extensión de la matriz de confusión multiclase definida en la Tabla 3.2.

Para la métrica de % de acierto general del modelo, la extensión al caso multiclase es trivial:

$$\%\text{ de Acierto} = \frac{\sum_{k=1}^{K} N_{kk}}{N}.$$

Para extender la métrica de la precisión, sensibilidad y valor $\mathbf{F_1}$, una posibilidad es usar el concepto

one-vs-rest (OVR) introducido en la Sección 2.3.6, calculando así las métricas para cada clase por separado, agregándolas posteriormente con algún criterio específico (media, media ponderada, etc.).

En el caso de la precisión para la clase k, se tendría:

$$\text{Precisión}_k = \frac{N_{kk}}{\sum_{i=1}^{K} N_{ik}}.$$

Calculando así la precisión que nuestro modelo tiene para distinguir entre la clase k y el resto.

Para la sensibilidad de la clase k, la métrica sería análoga a la precisión, pero iterando sobre los valores de la misma fila de la matriz de confusión en vez de por los de la columna:

$$\text{Sensibilidad}_k = \frac{N_{kk}}{\sum_{i=1}^{K} N_{ki}}.$$

En el caso del valor $\mathbf{F_1}$ para la clase k la extensión es directa, modificando en la expresión (3.1) los valores de precisión y sensibilidad general por los de una clase en particular:

$$\text{Valor } \mathbf{F_1}_k = 2 \cdot \frac{\text{Precisión}_k \cdot \text{Sensibilidad}_k}{\text{Precisión}_k + \text{Sensibilidad}_k}.$$

Una vez calculadas las métricas para cada clase, estas se pueden agrupar obteniendo así una métrica global. Es habitual utilizar la media aritmética (lo que se conoce como agregación 'macro' en la literatura) o la media ponderada por la proporción que cada clase tiene en el conjunto de entrenamiento, para así tener en cuenta desbalanceos entre las clases.

Aparte de estas adaptaciones, también existen otras métricas con definiciones específicas para el caso de problemas multiclase. En este trabajo nos centraremos en la métrica CCM, definida en (3.2), que para el caso $K > 2$ se define de la siguiente manera:

$$CCM = \frac{c \cdot s - \sum_{k=1}^{K} p_k \cdot t_k}{\sqrt{\left(s^2 - \sum_{k=1}^{K} p_k^2\right)\left(s^2 - \sum_{k=1}^{K} t_k^2\right)}}$$

en donde:

- $c = \sum_{k=1}^{K} N_{kk}$, el número total de elementos correctamente predichos.
- $s = \sum_{i=1}^{K} \sum_{j=1}^{K} N_{ij} = N$, el número total de elementos de la matriz.
- $p_k = \sum_{i=1}^{K} N_{ki}$, el número total de veces que predecimos la clase k, es decir, la suma de la columna k.
- $t_k = \sum_{j=1}^{K} N_{ik}$, el número total de veces que la clase real observada es k, es decir, la suma de la fila k.

Métricas para evaluar problemas de regresión

Para el caso de problemas de regresión (variable respuesta continua, i.e., $y \in \mathbb{R}$), supongamos que tenemos un modelo predictivo h tal que $h(\mathbf{x}) \in \mathbb{R}$. Imaginemos también que tenemos un conjunto de N pares de datos para evaluar el modelo $S' = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ no usado en la fase de entrenamiento. Las métricas de evaluación para este tipo de problemas son las siguientes:

- **Error cuadrático medio** (ECM):

$$ECM(h) = \frac{1}{N}\sum_{i=1}^{N}(y_i - h(\mathbf{x}_i))^2.$$

El ECM es una de las métricas mas usada, tanto para evaluar modelos como para entrenarlos a partir de utilizarla como función de pérdida (dado que es diferenciable). Esta nos devuelve una cantidad absoluta que indica cuánto difieren los valores predichos por nuestro modelo de los valores observados. No es interpretable por si misma, pero sí es de utilidad para comparar varios modelos.

Es habitual usar la raíz cuadrada del ECM, dando lugar a la métrica de la **raíz del error cuadrático medio** (RECM):

$$RECM(h) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - h(\mathbf{x}_i))^2}.$$

En esta expresión, al aplicar la raíz cuadrada al resultado se obtiene la métrica del error al mismo nivel que los datos originales, facilitando así su interpretación.

- **Error absoluto medio** (EAM):

$$EAM(h) = \frac{1}{N}\sum_{i=1}^{N}|y_i - h(\mathbf{x}_i)|.$$

Esta métrica es una representación más veraz de los errores cometidos por el modelo, dado que no penaliza de más los errores grandes como EMC y REMC.

- $\mathbf{R}^2$ **o coefiente de determinación**:

$$\mathbf{R}^2(h) = 1 - \frac{\sum_{i=1}^{N}\left(y_i - h(\mathbf{x}_i)\right)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \text{ en donde } \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i.$$

Esta cuantía representa la proporción de varianza de y que es explicada por las variables independientes en el modelo. Es un indicador de lo bien que ha ido el ajuste, y por lo tanto, nos da una medida de cuánto de bien vamos a predecir nuevos datos de test aún no vistos. El mayor valor que puede alcanzar es 1, y puede ser negativo.

3.2. Problemas planteados, resultados y conclusiones

Para poner en práctica los conceptos introducidos en este trabajo, en esta sección se presentan tres problemas a resolver con técnicas de aprendizaje estadístico. Se comienza con **P1**, un problema de clasificación binaria en el que se pretende predecir el nivel de ingresos de una persona adulta (codificado en dos clases: más o menos de \$50k/año) en base a distintas características de la misma (edad, nivel de educación, etc.). El segundo problema (**P2**) es de clasificación multiclase, en donde a partir de una imagen de un número del 0 al 9 escrito a mano (representada por un vector en escala de grises) se intentará predecir el dígito correspondiente. Por último, **P3** es un problema de regresión cuyo objetivo es predecir el precio de un coche usado en base a sus características (año de matriculación, potencia, etc.).

Para cada uno de los problemas se presentará el conjunto de datos y su estructura; los modelos seleccionados para resolver el problema y sus parámetros óptimos obtenidos por validación cruzada; las distintas métricas de evaluación usadas en cada uno de ellos; y el código Python utilizado para llevar a cabo el experimento.

3.2.1. P1: Clasificación de nivel de ingresos de población adulta

Para el primer problema utilizaremos el conjunto de datos *Adult Data Set*, obtenido del repositorio de Machine Learning de la UCI (Dua y Grraff, 20019). Este conjunto de datos, ampliamente utilizado en la literatura, consta de datos de cerca de 50 000 personas adultas, cada una de ellas perteneciente a una clase de nivel de ingresos: la clase >50k (que codificaremos como +1) y la clase <50k (codificada como −1).

El código utilizado para resolver este problema se puede ver en el Apéndice B.2.

Presentación del conjunto de datos

Los atributos de los que se disponen para cada persona son los siguientes:

- *age*: Variable continua con la edad de la persona.
- *workclass*: Tipo de ocupación de la persona con 8 posibles valores.
- *education*: Nivel de educación, con 15 valores posibles.
- *education-num*: Identificador de la variable de nivel de educación.
- *marital-status*: Situación sentimental, con 7 valores posibles.
- *occupation*: Ocupación de la persona.
- *relationship*: Situación sentimental, con 6 valores posibles.
- *race*: Raza de la persona, con 5 valores posibles.
- *sex*: Female, Male.
- *capital-gain*: Ganancias capitales de la persona.
- *capital-loss*: Pérdidas capitales de la persona.
- *hours-per-week*: Variable continua con las horas trabajadas por semana.
- *native-country*: País de origen.

Preprocesado de los datos

El preprocesado de los datos consta de las siguientes fases:

- Eliminamos todas las filas del conjunto de datos para las cuales algún atributo sea nulo.
- Eliminamos duplicados del conjunto de datos.
- Codificamos la variable *income* en sus dos posibles clases.

- Filtramos únicamente por el país Estados Unidos (representación mayoritaria en el conjunto de datos), para que los países con distinto nivel económico no introduzcan sesgos al problema.
- Eliminamos variables que no son de utilidad o que son redudantes: *education-num*, ya que es redudante al saber *education*, y *Native-country*, dada que nos quedamos únicamente con EEUU.
- Reducimos la cardinalidad de las variables *education* y *marital-status*, simplificando sus posibles valores.
- Codificamos las variables categóricas restantes con el método *one-hot enconding*, que convierte cada variable categórica en m variables (donde m es la cardinalidad de la variable en cuestión), obteniendo así un 1 en la variable asociada al valor correspondiente, y 0 para las demás.

El resultado final es un conjunto de datos con 42 256 registros y 50 variables.

Modelos entrenados y sus resultados

Los modelos seleccionados para este problema son los siguientes:

- Árbol de decisión para clasificación CART.
- Regresión logística.
- Perceptrón multicapa.
- Clasificador de Bayes ingenuo Gaussiano.
- $k-$vecinos más cercanos.
- Máquina de vectores de soporte para clasificación.
- Análisis discriminante lineal.
- Random Forest.
- Bagging.
- AdaBoost.
- Stacking usando como modelos base los 7 primeros modelos de esta lista, y como meta-modelo la regresión logística.

Para todos ellos el método de entrenamiento es análogo. En primer lugar, creamos una malla de posibles parámetros según el modelo en cuestión, y hacemos una búsqueda bruta en dicho espacio de parámetros con validación cruzada, quedándonos con el conjunto de parámetros que optimice el % de acierto del modelo. A continuación procedemos a entrenar de nuevo el modelo en el conjunto de entrenamiento, ya con los parámetros tuneados, para posteriormente calcular las métricas de evaluación del mismo sobre el conjunto de test.

Los resultados de cada modelo para las distintas métricas evaluadas se pueden ver en la Tabla 3.3. Para todas las métricas, con excepción del CCM, se calcula cada una de ellas a nivel de clase (>50k y <50k) y a mayores la media aritmética de ambas (bajo la variable avg). En concreto, para el % de acierto se hace la media ponderada, para tener en cuenta el desbalanceo entre clases. Destacar que, al estar ante un problema binario, el % de acierto a nivel de clase y la sensibilidad son idénticas.

Además, en la Figura 3.1 se presenta un diagrama de cajas hecho a partir del cálculo de la estimación del error de test de los distintos modelos con validación cruzada. En esta, además de ver la estimación

	Métrica												
	% Acierto			Sensibilidad			Precisión			valor $\mathbf{F}_1$			CCM
	<50k	>50k	w.avg	<50k	>50k	avg	<50k	>50k	avg	<50k	>50k	avg	valor
CART	0.88	0.62	0.81	0.88	0.62	0.756	0.88	0.64	0.76	0.88	0.63	0.75	0.51
R. Logística	0.93	0.57	0.83	0.93	0.57	0.75	0.87	0.73	0.80	0.90	0.64	0.77	0.54
LDA	0.93	0.57	0.83	0.93	0.57	0.75	0.87	0.73	0.80	0.90	0.64	0.77	0.54
MLP	0.94	0.52	0.84	0.94	0.52	0.86	0.86	0.79	0.83	0.90	0.63	0.77	0.57
$k-$NN	0.94	0.59	0.85	0.94	0.59	0.76	0.87	0.75	0.81	0.90	0.66	0.78	0.57
NB Gaussiano	0.83	**0.76**	0.81	0.83	**0.76**	**0.80**	**0.92**	0.60	0.76	0.87	0.67	0.77	0.55
SVC	0.94	0.62	0.86	0.94	0.62	0.78	0.88	0.78	0.83	0.91	0.69	0.79	0.60
Bagging	0.91	0.63	0.77	0.91	0.63	0.78	0.88	0.71	0.79	0.90	0.67	0.78	0.56
Random Forest	0.90	0.66	0.84	0.90	0.66	0.78	0.87	0.72	0.79	0.89	0.65	0.78	0.56
AdaBoost	**0.95**	0.64	**0.87**	**0.95**	0.64	**0.80**	0.89	**0.79**	**0.83**	**0.92**	**0.70**	**0.80**	**0.62**
Stacking S1	0.94	0.70	0.86	0.94	0.70	**0.80**	0.89	0.78	**0.83**	0.91	**0.70**	**0.80**	0.61

Tabla 3.3: Resultados de aplicar distintos algoritmos al problema **P1**. En negrita se destaca el mejor modelo según cada métrica.

del % de acierto de cada uno de las técnicas, se puede ver la varianza de cada una de ellas al ser entrenadas con distintos conjuntos de entrenamiento.

Como se puede observar en los resultados, todos los métodos en general tienen dificultades a la hora de predecir correctamente datos de la clase >50k por su baja representación en el conjunto de entrenamiento. En general, se aprecia que los métodos de ensamblado tienen menos varianza que los individuales, y destacan AdaBoost y S1 como mejores clasificadores (en términos de % de acierto). Teniendo en cuenta la definición del ensamblado S1 tipo stacking, parece que la combinación de los siete primeros modelos mejora, ligeramente, el rendimiento de los modelos base individuales.

3.2.2. P2: Clasificación de dígitos escritos a mano

Presentación del conjunto de datos

En este problema utilizaremos el conjunto de datos MNIST obtenido de la página web oficial de los autores (Lecun y Cortes, 2010). Este conjunto de datos consta de 60 000 imágenes de 28x28 píxeles codificadas en vectores de longitud 784, en donde cada componente tiene, en escala de grises, la intensidad de color de cada pixel. Cada una de las imágenes representa un único dígito del 0 al 9, por lo que el objetivo es clasificar cada una de ellas en la clase que le corresponda. En este caso estamos

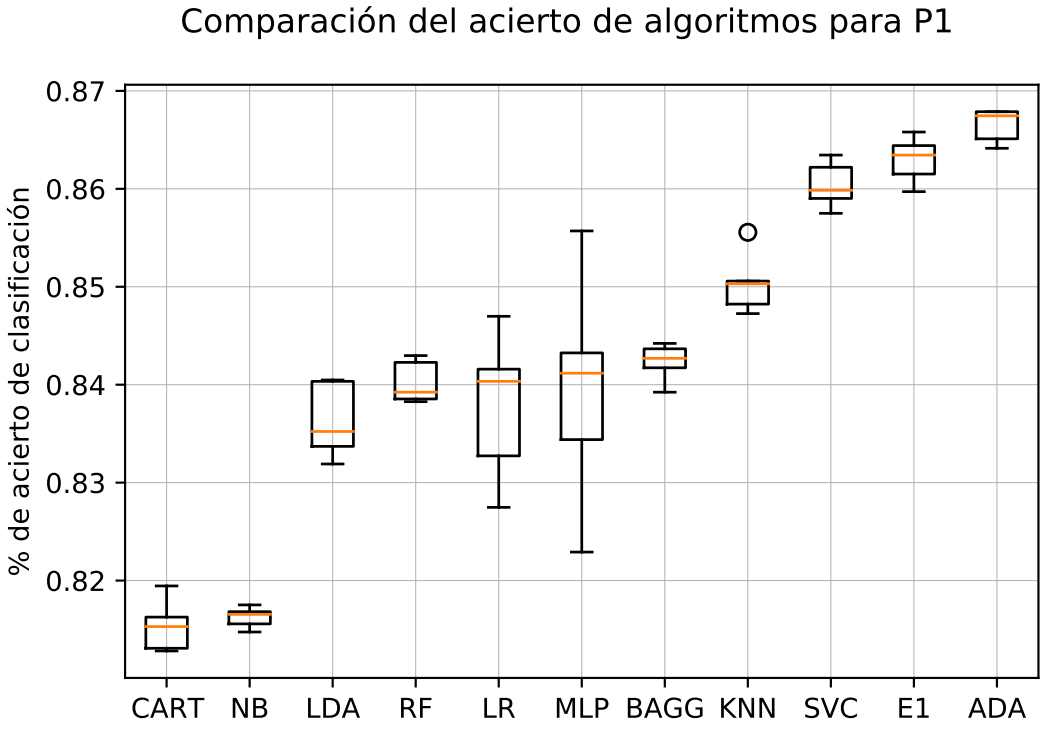

Figura 3.1: Diagrama de caja de los resultados de estimar para cada modelo su porcentaje de acierto de clasificación usando validación cruzada con $K = 5$, ordenados de izquierda a derecha de menor a mayor mediana. Por sus siglas en inglés: CART (árboles de regresión y clasificación, *Classification And Regression Trees*), NB (Bayes ingenuo, o *Naive Bayes*), LDA (Análisis discriminante lineal, o *Linear Discriminant Analysis*), LR (Regresión logística, o *logistic regression*), MLP (Perceprón multicapa, o *Multilayer Perceptron*, definido en 1.3), RF (Random Forest, definido en 2.3.5, BAGG (Bagging, definido en 2.3.4), KNN ($k-$vecinos más cercanos, o $k-$*nearest neighbors*, definido en 1.17), SVC (Máquina de vectores de soporte para clasificación, o *Support Vector Clasification*), E1 (Ensamblado tipo stacking de varios de los modelos individuales mencionados, definido en 2.3.7), ADA (AdaBoost, definido en 2.3.6).

ante un problema multiclase balanceado, dado que todas las clases tienen la misma proporción en el conjunto de entrenamiento.

De nuevo, el código utilizado para resolver este problema se puede ver en el Apéndice B.3.

Preprocesado de los datos

El único preprocesado que le hacemos a los datos es convertir las imágenes de escala de grises (valores de 0 a 255) a blanco y negro (valores 0 para blanco y 1 para negro).

Modelos entrenados y sus resultados

Los modelos a entrenar son los mismos que para el problema **P1**, de nuevo con el mismo método de entrenamiento. Para todos los modelos buscamos los mejores parámetros candidatos maximizando la métrica del porcentaje de acierto, para posteriormente entrenar el modelo final con dichos parámetros. Finalmente se estiman las distintas métricas para evaluar el rendimiento del clasificador, las cuales se resumen en la Tabla 3.4.

Teniendo en cuenta la dificultad del problema, y que las técnicas seleccionadas no están pensadas originalmente para tratar con imágenes, en general se observa que todas las técnicas obtienen resultados bastante meritorios, a excepción de AdaBoost y el clasificador de Bayes ingenuo. Destaca de nuevo el buen rendimiento de los modelos ensamblados, sobre todo Random Forest, Bagging y E1. Este último ensamblado tipo stacking, E1, es el mejor bajo todas las métricas de clasificación multiclase usadas, y mejora substancialmente el rendimiento de los siete clasificadores base que forman parte del mismo.

En el diagrama de cajas de la figura 3.2 se pueden ver, de izquierda a derecha, de menor a mayor mediana de porcentaje de acierto, todos los modelos entrenados para este problema.

3.2.3. P3: Predicción del valor de un coche usado

Para este problema utilizaremos el conjunto de datos de dominio público *Car Evaluation*, obtenido de kaggle.com, una conocida web que organiza competiciones de Machine Learning y que dispone de un amplio catálogo de conjuntos de datos. Disponemos de información de 100 000 vehículos usados, cada uno de ellos con su precio de valor de mercado. Para este caso utilizaremos únicamente un subcojunto de los mismos para hacerlo computacionalmente más asequible, usando así exclusivamente los vehículos de la marca Volkswagen (16 000 registros aproximadamente).

El código utilizado para resolver este problema se puede ver en el Apéndice B.4.

Presentación del conjunto de datos

Los atributos de los que se disponen para cada persona son los siguientes:

- *model*: Variable categórica con el modelo de coche.
- *year*: Año de fabricación del modelo.
- *price*: Precio de valor de mercado.
- *transmission*: Tipo de caja de cambio.
- *mileage*: Número de kilómetros.
- *fuelType*: Tipo de combustible usado.

	Métrica							
	% Acierto	Sensibilidad		Precisión		Valor $\mathbf{F_1}$		CCM
	avg	w.avg	avg	w.avg	avg	w.avg	avg	valor
CART	0.77	0.79	0.80	0.79	0.80	0.79	0.80	0.77
R. Logística	0.87	0.88	0.89	0.88	0.89	0.88	0.89	0.87
LDA	0.83	0.84	0.84	0.84	0.84	0.84	0.84	0.82
MLP	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.91
$k-$NN	0.91	0.92	0.93	0.92	0.93	0.92	0.93	0.92
NB Gaussiano	0.56	0.59	0.63	0.73	0.73	0.63	0.57	0.60
SVC	0.92	0.93	0.93	0.93	0.93	0.93	0.92	0.92
Bagging	0.90	0.89	0.90	0.89	0.90	0.89	0.90	0.88
Random Forest	0.93	0.95	0.95	0.95	0.95	0.95	0.94	0.90
AdaBoost	0.56	0.66	0.65	0.66	0.65	0.64	0.64	0.62
Stacking S1	**0.95**	**0.95**	**0.96**	**0.95**	**0.96**	**0.96**	**0.96**	**0.96**

Tabla 3.4: Resultados obtenidos para el problema **P2**. En negrita se destaca el mejor modelo según cada métrica. 'avg' es el resultado de hacer la media aritmética del resultado para cada clase, y 'w.avg' de la media aritmética ponderada por la proporción de elementos de cada clase.

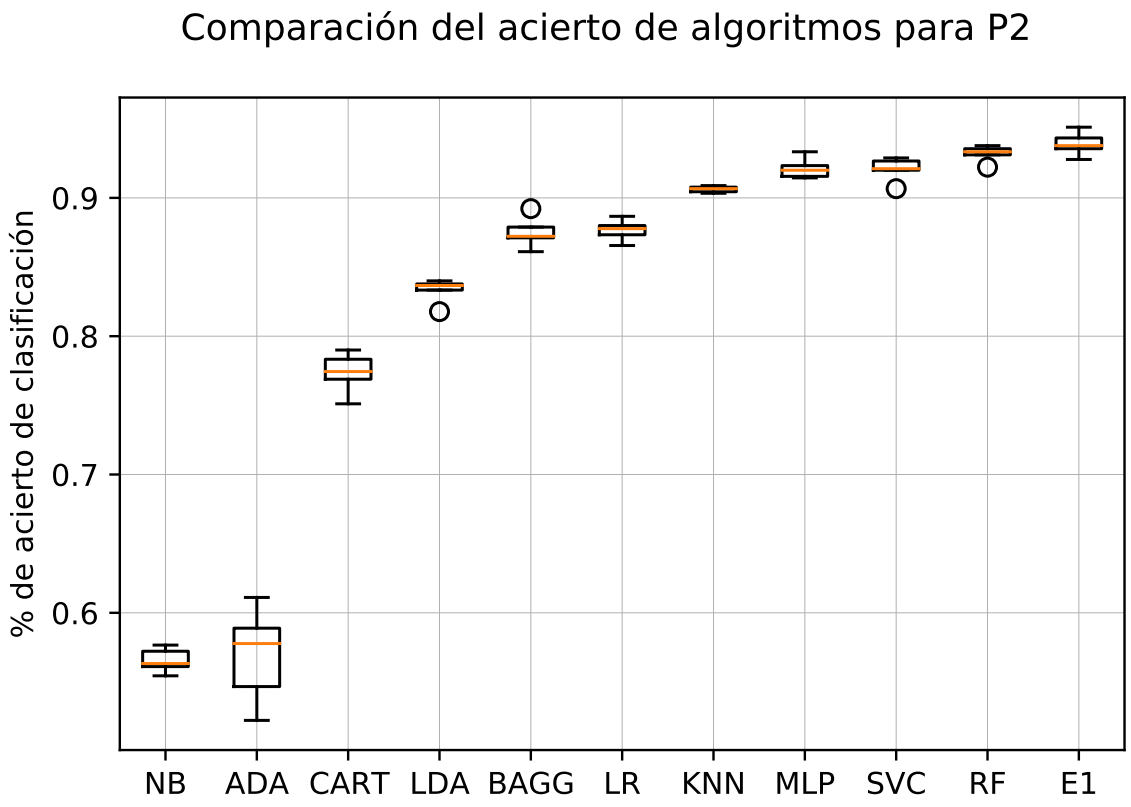

Figura 3.2: Diagrama de caja de los resultados de **P2** obtenidos tras estimar para cada modelo su porcentaje de acierto de clasificación usando validación cruzada con $K = 5$, ordenados de izquierda a derecha, de menor a mayor mediana. Se continúa con la nomenclatura de la Tabla 3.1, por sus siglas en inglés.

- *tax*: Impuesto de circulación.
- *mpg*: Consumo del vehículo, en galones por milla.
- *engineSize*: Tamaño del motor, en litros.

Preprocesado de los datos

A mayores de filtrar únicamente por los coches de la marca Volkswagen, se eliminan los registros que contengan algun valor nulo, y se codifican las variables categóricas (transmission, fuelType y model) con el mismo criterio de codificación one-hot mencionado en **P1**.

El resultado final es un conjunto de datos con 15 157 registros y 40 variables.

Modelos entrenados y sus resultados

Los modelos seleccionados para este problema son los siguientes:

- Árbol de decisión para regresión CART.
- Regresión lineal.
- Perceptrón multicapa.
- $k-$vecinos más cercanos.
- Máquina de vectores de soporte para regresión.
- Random Forest.
- Bagging.
- AdaBoost.
- Stacking usando como modelos base los cinco primeros modelos de esta lista, y como meta-modelo la regresión logística.

Para estos, aunque estemos ante un problema de regresión, el método de tuneado y entrenamiento es análogo a **P1** y **P2**. En primer lugar, creamos una malla de posibles parámetros según el modelo en cuestión, y hacemos una búsqueda bruta en dicho espacio de parámetros con validación cruzada, quedándonos con el conjunto de parámetros que optimice el valor $\mathbf{R}^2$ del modelo. A continuación procedemos a entrenar de nuevo el modelo en el conjunto de entrenamiento, ya con los parámetros tuneados, para posteriormente calcular las métricas de evaluación del mismo sobre el conjunto de test.

Los resultados de cada modelo para las cuatro métricas de regresión presentadas al inicio de esta sección se pueden ver en la Tabla 3.5. De nuevo, en la Figura 3.3 se presenta un diagrama de cajas hecho a partir del cálculo de la estimación del valor $\mathbf{R}^2$ de test de los distintos modelos con validación cruzada.

Como se puede observar en los resultados, en general todos los métodos que involucran un ensamblado tienen un mejor poder de predicción y generalización que los métodos individuales, destacando el modelo Random Forest, el cual obtiene los mejores resultados para las cuatro métricas evaluadas. Además, se puede apreciar como el ensamblado tipo stacking E1, el cual es una combinación de los cinco primeros modelos individuales entrenados, con una regresión *Ridge* como meta-modelo, mejora substancialmente los resultados de los modelos base.

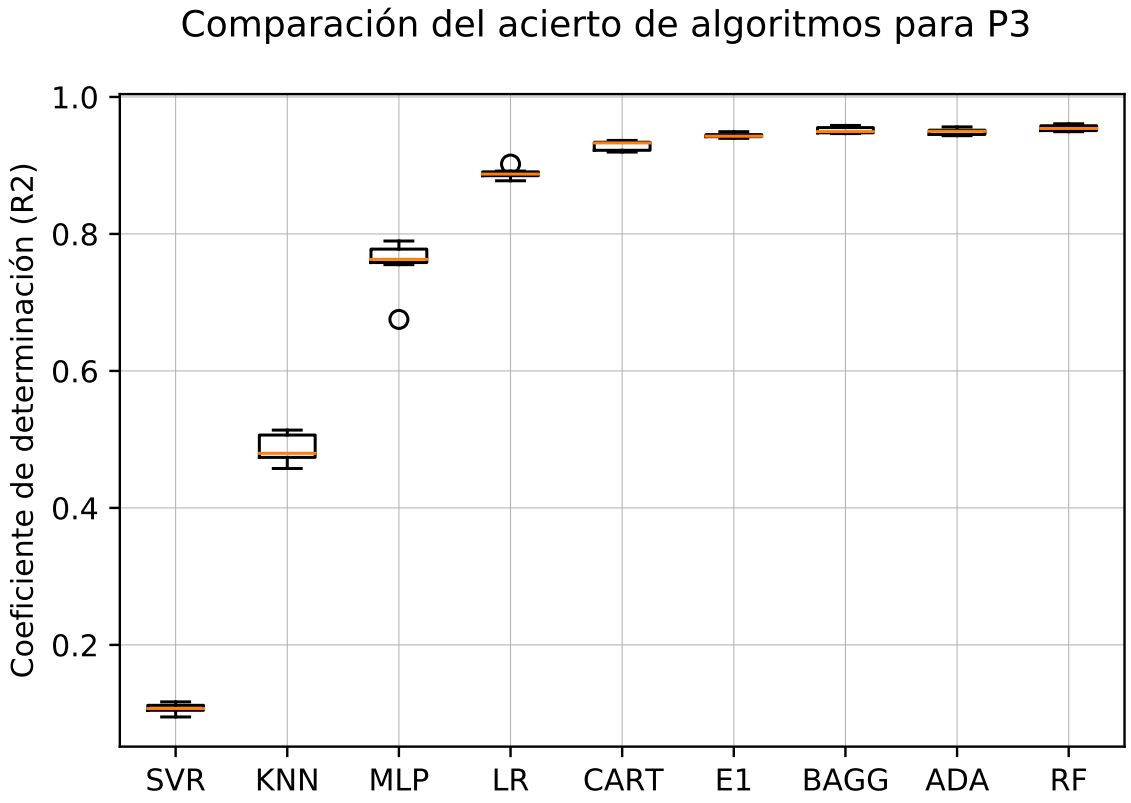

Figura 3.3: Diagrama de caja de los resultados de estimar para cada modelo su valor $\mathbf{R}^2$ usando validación cruzada con $K = 5$, ordenados de izquierda a derecha de menor a mayor mediana. Por sus siglas en inglés: SVR (Máquina de vectores de soporte para regresión, o *Support Vector Regression*), KNN ($k-$vecinos más cercanos, o $k-$*nearest neighbors*, definido en 1.17), MLP (Perceprón multicapa, o *Multilayer Perceptron*, definido en 1.3), CART (árboles de regresión y clasificación, o *Classification And Regression Trees*), LR (Regresión lineal, o *Lineal Regression*), E1 (Ensamblado tipo stacking de varios de los modelos individuales mencionados, definido en 2.3.7), BAGG (Bagging, definido en 2.3.4), definido en 1.17), ADA (AdaBoost, definido en 2.3.6), RF (Random Forest, definido en 2.3.5).

	Métrica			
	ECM	RECM	EAM	$\mathbf{R}^2$
CART	4246965.45	2059.00	1326.91	0.92
Regresión L.	6787867.59	2604.66	1852.03	0.88
$k-$NN	31190681.17	5582.78	4110.12	0.48
MLP	14276447.89	3773.71	2639.80	0.15
SVR	31190681.17	5582.78	4110.12	0.10
Random Forest	**2665615.51**	**1632.21**	**1081.43**	**0.95**
Bagging	2814938.93	1677.41	1119.33	**0.95**
Ada Boost	3085347.54	1754.60	1290.22	**0.95**
Ensamblado E1	3389747.70	1840.59	1267.80	0.94

Tabla 3.5: Resultados obtenidos para el problema **P3** tras estimar las distintas métricas utilizando validación cruzada. En negrita se destaca el mejor modelo según cada métrica. Para los errores (ECM, RECM y EAM), interesa que la métrica sea lo menor posible, mientras que para el $\mathbf{R}^2$ es a la inversa.

Apéndice A

Demostraciones

A.1. Teorema 2.1

Demostración.

La demostración del Teorema 2.1 se divide en 3 partes. En primero lugar probaremos que:

$$w_i^{L+1} = \frac{e^{-y_i \sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i)}}{N \prod_{l=1}^{L} Z_l}.$$

A partir de deshacer la recurrencia de w_i^l, tenemos:

$$\begin{aligned}
w_i^{L+1} &= \frac{w_i^L e^{-\alpha_L y_i h_L(\mathbf{x}_i)}}{Z_L} \\
&= \frac{w_i^{L-1} e^{-\alpha_{L-1} y_i h_{L-1}(\mathbf{x}_i)} e^{-\alpha_L y_i h_L(\mathbf{x}_i)}}{Z_L Z_{L-1}} \\
&\text{(continuamos recursivamente hasta llegar a } w_i^1\text{)} \\
&= w_i 1 \frac{\prod_{l=1}^{L} e^{-\alpha_l y_i h_l(\mathbf{x}_i)}}{\prod_{l=1}^{L} Z_l} \\
&= \frac{\prod_{l=1}^{L} e^{-\alpha_l y_i h_l(\mathbf{x}_i)}}{N \prod_{l=1}^{L} Z_l} \\
&= \frac{e^{\sum_{l=1}^{L} -\alpha_l y_i h_l(\mathbf{x}_i)}}{N \prod_{l=1}^{L} Z_l} \\
&= \frac{e^{-y_i \sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i)}}{N \prod_{l=1}^{L} Z_l}.
\end{aligned}$$

Por segundo lugar, se probará la siguiente cota del error intermedia:

$$RE(h_{\texttt{ada}}) \leq \prod_{l=1}^{L} Z_l.$$

Por un lado, tenemos que el clasificador final $h_{\texttt{ada}}$ comete un error si signo$\big(\sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i)\big) \neq y_i$, y

por lo tanto, $\text{signo}\big(y_i \sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i)\big) = -1$ y $-y_i \sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i)\big) \geq 0$. Así, tenemos:

$$e^{-y_i \sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i)} \geq 1 \text{ si } h_{\texttt{ada}}(\mathbf{x}_i) \neq y_i.$$

Además, dado que $e^x \geq 0 \; \forall x$,

$$e^{-y_i \sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i)} \geq 0 \text{ si } h_{\texttt{ada}}(\mathbf{x}_i) = y_i.$$

Usando estas igualdades y el resultado de la primera parte de la prueba, tenemos:

$$\begin{aligned} RE(h_{\texttt{ada}}) &= \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left(h\left(\mathbf{x}_i\right) \neq y_i\right) \leq \frac{1}{N} \sum_{i=1}^{N} e^{-y_i \sum_{l=1}^{L} \alpha_l h_l(\mathbf{x}_i)} \\ &= \frac{1}{N} \sum_{i=1}^{N} N w_i^{L+1} \prod_{t=1}^{L} Z_l \\ &= \sum_{i=1}^{N} w_i^{L+1} \prod_{t=1}^{L} Z_l \\ &= \prod_{t=1}^{L} Z_l. \end{aligned}$$

Como tercera y última parte, acotamos los valores de Z_l. Primero, segmentaremos la definición de Z_l en dos partes, una con las sumas teniendo en cuenta los puntos clasificados correctamente, y la otra con los puntos incorrectamente:

$$\begin{aligned} Z_t &= \sum_{i=1}^{m} w_i^l e^{-\alpha_l y_i h_l(\mathbf{x}_i)} \\ &= \sum_{i:y_i = h_l(\mathbf{x}_i)} w_i^l e^{-\alpha_l} + \sum_{i:y_i \neq h_l(\mathbf{x}_i)} w_i^l e^{\alpha_l} \\ &= e^{-\alpha_l} \sum_{i:y_i = h_l(\mathbf{x}_i)} w_i^l + e^{\alpha_l} \sum_{i:y_i \neq h_l(\mathbf{x}_i)} w_i^l \\ &= \left(1 - \epsilon_l\right) e^{-\alpha_l} + \epsilon_l e^{\alpha_l}. \end{aligned}$$

Como vimos en la segunda parte de la demostración, el error será menor cuanto menor sea Z_l. Por lo tanto, podemos resolver la siguiente ecuación, y sacar el valor α_l óptimo para minimizar Z_l para cada l. Derivando Z_l en función de α_l e igualando a 0:

$$0 = \frac{dZ_l}{d\alpha_l} = \left(1 - \epsilon_l\right) e^{-\alpha_l} + \epsilon_l e^{\alpha_l}$$

obtendríamos que el valor de α_l óptimo es:

$$\alpha_l = \frac{1}{2} \log\left(\frac{1 - \epsilon_l}{\epsilon_l}\right).$$

Poniendo este valor de α_l óptimo en Z_l tenemos:

$$Z_l = \left(1 - \epsilon_l\right) e^{-\alpha_l} + \epsilon_l e^{-\alpha_l} = 2\sqrt{(1 - \epsilon_l)\epsilon_l}.$$

Finalmente, si agrupamos todo lo anterior, tenemos:

$$\begin{aligned}
RE(h_{\texttt{ada}}) \leq & \prod_{l=1}^{L} Z_t = \prod_{l=1}^{L} 2\sqrt{(1-\epsilon_l)\,\epsilon_l} \\
& (\text{ usando que } \gamma_l = \frac{1}{2} - \epsilon_l) \\
= & \prod_{l=1}^{L} 2\sqrt{\left(\frac{1}{2} - \gamma_l\right)\left(\frac{1}{2} + \gamma_l\right)} \\
= & \prod_{l=1}^{L} \sqrt{1 - 4\gamma_l^2} \\
& (\text{ usando que } 1 + x \leq e^x) \\
\leq & \prod_{l=1}^{L} \sqrt{e^{-4\gamma_l^2}} = \prod_{l=1}^{L} e^{-2\gamma_l^2} \\
= & e^{-2\sum_{l=1}^{L} \gamma_l^2} \leq e^{-2T\gamma^2}
\end{aligned}$$

quedando así finalizada la prueba.

■

Apéndice B

Código Python

B.1. Código genérico para todos los problemas

```
# Imports básicos
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib

# Imports para preprocesado de datos
from sklearn.decomposition import PCA
from mpl_toolkits.mplot3d import Axes3D
from sklearn.preprocessing import LabelEncoder


# Imports de modelos
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier,
    GradientBoostingClassifier, ExtraTreesClassifier, VotingClassifier,
    BaggingClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostRegressor,
    StackingRegressor, RandomForestRegressor, BaggingRegressor
from sklearn.neural_network import MLPClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier

# Imports de métricas y selección de modelo
from sklearn.model_selection import GridSearchCV, cross_val_score, StratifiedKFold
    , learning_curve, train_test_split, KFold
from sklearn.model_selection import cross_val_score
from sklearn.metrics import silhouette_samples
from sklearn.metrics import silhouette_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report, accuracy_score
```

B.2. Código para problema P1

```
# Carga de datos
p1_data = pd.read_csv("Data/P1_DATA.csv")

# Tratado de nulos y duplicados
p1_data['workclass']=p1_data['workclass'].replace('?',np.nan)
p1_data['occupation']=p1_data['occupation'].replace('?',np.nan)
p1_data['native-country']=p1_data['native-country'].replace('?',np.nan)
p1_data.dropna(how='any',inplace=True)
p1_data=p1_data.drop_duplicates()

# Codificamos variable respuesta
p1_data.income = [1 if each=='>50K' else 0 for each in p1_data.income]

# Filtramos solo para estados unidos
p1_data = p1_data[p1_data["native-country"] == 'United-States']

# Eliminamos columnas que no son de utilidad
p1_data.drop(["fnlwgt","native-country", "educational-num"],axis=1,inplace=True)

# Simplificamos la cardinalidad de algunas columnas categóricas:
# Simplificamos alguna columna

# Educacion
p1_data['education'].replace('Preschool', 'dropout',inplace=True)
p1_data['education'].replace('10th', 'dropout',inplace=True)
p1_data['education'].replace('11th', 'dropout',inplace=True)
p1_data['education'].replace('12th', 'dropout',inplace=True)
p1_data['education'].replace('1st-4th', 'dropout',inplace=True)
p1_data['education'].replace('5th-6th', 'dropout',inplace=True)
p1_data['education'].replace('7th-8th', 'dropout',inplace=True)
p1_data['education'].replace('9th', 'dropout',inplace=True)
p1_data['education'].replace('HS-Grad', 'HighGrad',inplace=True)
p1_data['education'].replace('HS-grad', 'HighGrad',inplace=True)
p1_data['education'].replace('Some-college', 'CommunityCollege',inplace=True)
p1_data['education'].replace('Assoc-acdm', 'CommunityCollege',inplace=True)
p1_data['education'].replace('Assoc-voc', 'CommunityCollege',inplace=True)
p1_data['education'].replace('Bachelors', 'Bachelors',inplace=True)
p1_data['education'].replace('Masters', 'Masters',inplace=True)
p1_data['education'].replace('Prof-school', 'Masters',inplace=True)
p1_data['education'].replace('Doctorate', 'Doctorate',inplace=True)

# Estado sentimental
p1_data['marital-status'].replace('Never-married', 'NotMarried',inplace=True)
p1_data['marital-status'].replace(['Married-AF-spouse'], 'Married',inplace=True)
p1_data['marital-status'].replace(['Married-civ-spouse'], 'Married',inplace=True)
p1_data['marital-status'].replace(['Married-spouse-absent'], 'NotMarried',inplace=
    True)
p1_data['marital-status'].replace(['Separated'], 'Separated',inplace=True)
p1_data['marital-status'].replace(['Divorced'], 'Separated',inplace=True)
p1_data['marital-status'].replace(['Widowed'], 'Widowed',inplace=True)

# Codificamos las variables categóritas restantes
p1_data_dummies = pd.get_dummies(p1_data)

# Separamos vectores de características y respuesta
drop_elements = ['income']
y = p1_data_dummies["income"]
X = p1_data_dummies.drop(drop_elements, axis=1)

# Spliteamos conjunto de test y entrenamiento
x_train , x_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=
    1)
```

```
# Método genérico para sacar los mejores parámetros de cada uno de las técnicas
# Esto se repetiría para todos los modelos en la lista 'models'
param_grid = {
  "C" : np.array([1,2,5]),
  "gamma" : np.array(["scale","auto"])
}

kfold = StratifiedKFold(n_splits=4)

grid = GridSearchCV(estimator=SVC(), param_grid=param_grid, scoring='accuracy', cv
    =kfold)
grid_result = grid.fit(x_train, y_train)

# Obtenemos los mejores parámetros para ese modelo
grid_result.best_params_


# Creamos una lista de modelos a usar
models = []
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier(n_neighbors=20)))
models.append(('MLP', MLPClassifier()))
models.append(('SVC', SVC(C = 5, gamma = "auto")))
models.append(('RF', RandomForestClassifier(n_estimators=1000)))
models.append(('BAGG', BaggingClassifier(n_estimators=1000)))
estimators = models[0:7]
models.append(('ADA', AdaBoostClassifier(n_estimators=500, learning_rate=1)))
models.append(('E1', StackingClassifier(estimators=estimators, final_estimator=
    LogisticRegression())))

# Sacamos métricas para cada uno de ellos, con validación cruzada de 5 iteraciones
results = []
names = []
for name, model in models:
print(f"Starting to train model {name}...")
kfold = KFold(n_splits=5)
cv_results = cross_val_score(model, x_train, y_train, cv=kfold, scoring='accuracy'
    )
model.fit(x_train,y_train)
results.append(cv_results)
names.append(name)
msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(f"############## CLASIFICACION REPORT FOR {model}")
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print(msg)
print(f"SCORE a mano: {accuracy_score(y_test, y_pred)}" )
print(f"matthews_corrcoef a mano: {matthews_corrcoef(y_test, y_pred)}" )

# Plot de los resultados:
fig = plt.figure()
fig.suptitle('Comparación del acierto de algoritmos para P1')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
ax.set_ylabel("% de acierto de clasificación")
plt.savefig('Plots/P1_boxplot.pdf')
plt.show()
```

B.3. Código para problema P2

```
# Carga de datos
p2_data = pd.read_csv("Data/P2_DATA.csv")

y = p2_data.label
X = p2_data.drop(columns = ["label"])


# Split de los datos:
x_train, x_test,y_train,y_test = train_test_split(X,y,test_size=0.20)

# Preprocesado para cambiar de escala de grises a blanco y negro
x_test[x_test>0]=1
x_train[x_train>0]=1

# Tuneamos modelos
# Practicandocon GridsearchCV - KNN
param_grid = {
  "n_neighbors" : np.array([1,2,3,4,5,7,10,15,30])
}
# kfold = KFold(n_splits=4)
kfold = StratifiedKFold(n_splits=10)

grid = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=param_grid,
    scoring='accuracy', cv=kfold)
grid_result = grid.fit(x_train, y_train)
grid_result.best_score_

# Preparamos modelos

models = []
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier(n_neighbors=1)))
models.append(('MLP', MLPClassifier()))
models.append(('SVC', SVC(C = 5, gamma = "auto")))
models.append(('RF', RandomForestClassifier(n_estimators=1000)))
models.append(('BAGG', BaggingClassifier(n_estimators=1000)))
estimators = models[0:7]
models.append(('ADA', AdaBoostClassifier(n_estimators=100, learning_rate=0.5)))
models.append(('E1', StackingClassifier(estimators=estimators, final_estimator=
    LogisticRegression())))

# Obtenemos resultados:
results = []
names = []
for name, model in models:
print(f"Starting to train model {name}...")
cv_results = cross_val_score(model, x_train, y_train, cv=5, scoring='accuracy')
model.fit(x_train,y_train)
results.append(cv_results)
names.append(name)
msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
print(f"############## CLASIFICACION REPORT FOR {model}")
y_pred = model.predict(x_test)
print(classification_report(y_test, y_pred))
print(msg)
print(f"SCORE a mano: {accuracy_score(y_test, y_pred)}" )
print(f"matthews_corrcoef a mano: {matthews_corrcoef(y_test, y_pred)}" )

# Boxplot:
fig = plt.figure()
```

```
fig.suptitle('Comparación del acierto de algoritmos para P2')
ax = fig.add_subplot(111)
plt.boxplot(results1)
ax.set_xticklabels(names1)
ax.set_ylabel("% de acierto de clasificación")
plt.grid(linewidth=0.1)
plt.savefig('Plots/P2_boxplot_v7.pdf')
plt.show()
```

B.4. Código para problema P3

```
# Carga de datos
p3_data = pd.read_csv("Data/P3_DATA.csv")

# Eliminamos nulos
p3_data.dropna(how='any',inplace=True)

# Codificamos las variables categóricas con el one-hot encoding
p3_data = pd.get_dummies(p3_data)

# Variable X e y
y = p3_data.price
X = p3_data.drop(columns=["price"])


# Split de entreno y testeo
x_train, x_test,y_train,y_test = train_test_split(X,y,test_size=0.20)


# Validación cruzada para sacr los mejores parámetros de los modelos, análogo para
    todos ellos
# Practicandocon GridsearchCV - SVR
param_grid = {
  "C" : np.array([1, 0.5]),
  "kernel" : ["poly", "rgb"],
  "gamma": ["scale","auto"]
}

kfold = StratifiedKFold(n_splits=5)

grid = GridSearchCV(estimator=SVR(), param_grid=param_grid, scoring='
    neg_mean_absolute_error', cv=kfold, n_jobs=-1)
grid_result = grid.fit(x_train, y_train)
grid_result.best_params

# Modelos a entrenar
models = []
models.append(('CART', DecisionTreeRegressor()))
models.append(('LR', LinearRegression()))
models.append(('KNN', KNeighborsRegressor(n_neighbors=20)))
models.append(('MLP', MLPRegressor()))
models.append(('SVR', SVR(gamma="scale", C=1)))
models.append(('RF', RandomForestRegressor(n_estimators=700)))
models.append(('BAGG', BaggingRegressor(n_estimators=750)))
estimators = models[0:5]
models.append(('E1', StackingRegressor(estimators=estimators, n_jobs=-1)))

# Obtención de resultados y métricas
scoring = {'r2': 'r2',
  'mean_absolute_error': 'neg_mean_absolute_error',
  "mean_squared_error": 'neg_mean_squared_error',
  "neg_root_mean_squared_error":"neg_root_mean_squared_error",
```

```
    "mean_absolute_percentage_error":"neg_mean_absolute_percentage_error"
}

results = []
names=[]
for name, model in models:
print(f"Starting to train model {name}...")
cv_results = cross_validate(model,  x_train, y_train, cv=7, scoring=scoring,
    n_jobs=-1)
print(f"root_mean_squared_error: {str(-cv_results['
    test_neg_root_mean_squared_error'].mean())}")
print(f"mean_absolute_error: {str(-cv_results['test_mean_absolute_error'].mean())}
    ")
print(f"mean_squared_error: {str(-cv_results['test_mean_squared_error'].mean())}")
print(f"mean_absolute_percentage_error: {str(-cv_results['
    test_mean_absolute_percentage_error'].mean())}")
print(f"r2: {str(cv_results['test_r2'].mean())}")
results.append(cv_results)

# Boxplot:
fig = plt.figure()
# fig.set_size_inches(18.5, 10.5)
fig.suptitle('Comparación del acierto de algoritmos para P3')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
ax.set_ylabel("Coeficiente de determinación (R2)")
plt.grid(linewidth=0.1)
plt.savefig('Plots/P3_boxplot_v1.pdf')
plt.show()
```

Bibliografía

[1] Bazell D, Aha WD (2001) Ensemble of classifiers for morphological galaxy classification. The Astrophysical Journal, 548:219-223

[2] Breiman L, Friedman J, Stone C, Olshen R (1984) Classification and Regression Trees. Taylor & Francis.

[3] Breiman L (1994) Bagging Predictors. Mach Learn 24:123-140.

[4] Breiman L (1998). Arcing classifiers. The Annals of Statistics, 26(3):801?849.

[5] Dasarathy B, Sheela B (1979) Composite classifier system design: concepts and methodology. Proceedings of the IEEE, vol. 67, no. 5:708-713.

[6] Dietterich T (2000) Ensemble Methods in Machine Learning. Multiple Classifier Systems. MCS 2000. Lecture Notes in Computer Science, vol 1857:1-15.

[7] Dua D, Graff C (2019). UCI Machine Learning Repository: http://archive.ics.uci.edu/ml. Accedido 1 de Julio de 2021. Irvine, CA: University of California, School of Information and Computer Science.

[8] Freund Y, Schapire R (1997). A decision-theoretic generalization of online learning and an application to boosting, Journal of Computer and System Sciences 55:119-139.

[9] Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: A statistical view of boosting (with discussions). Annals of Statistics, 28(2):337?407.

[10] Friedman J (2001) Greedy function approximation: A gradient boosting machine. Annals of Statistics 29(5):1189?1232.

[11] Hansen L, Salamon P (1990) Neural network ensembles. IEEE Trans. Pattern Analysis and Machine Intell., 12:993-1001.

[12] Hastie T, Tibshirani R, Friedman J (2001) The Elements of Statistical Learning. Springer, Nueva York.

[13] Hoeting J, Madigan D, Raftery A, Volinski C (1999) Bayesian Model Averaging: A Tutorial. Statistical Science, vol. 14, n. 4:382-417

[14] LeCun Y, Cortes C (2010) MNIST handwritten digit database: http://yann.lecun.com/exdb/mnist/.

[15] Hunter J (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, vol. 9, n. 3. 90:95.

[16] Kearns J, Vazirani V (1994) An Introduction to Computational Learning Theory. The MIT Press.

[17] Kearns M, Valiant L (1988) Learning Boolean formulae or finite automata is as hard as factoring (Technical Report TR-14-88). Cambridge, MA: Harvard University Aiken Computation Laboratory.

[18] Kearns M, Valiant L (1989) Cryptographic limitations on learning Boolean formulae and finite automata. Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing (pp. 433-444). New York, NY: ACM Press.

[19] Krizhevsky A, Sutskever I, Hinton G (2012) ImageNet Classification with Deep Convolutional Neural Networks.

[20] Kuncheva L (2004) Combining Pattern Classifiers: Methods and Algorithms. Wiley.

[21] Kuncheva L (2014) Combining Pattern Classifiers: Methods and Algorithms, 2nd Ed. Wiley.

[22] Lecun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-Based Learning Applied to Document Recognition.

[23] Leigh W, Purvis R, Ragusa J (2002) Forecasting the NYSE composite index with technical analysis, pattern recognizer, neural networks and genetic algorithm: A case study. Decision Support Systems, 32(4):361-377.

[24] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Verplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, vol. 12: 2825-2830.

[25] Re M, Valentini G (2012) Ensemble methods: A review. Advances in Machine Learning and Data Mining for Astronomy:563-594.

[26] Schapire R (1990) The Strength of Weak Learnability. Machine Learning, vol. 5, n. 2:197-227.

[27] Schapire R, Freund Y, Bartlett P, Sun Lee W. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. The Annals of Statistics, 26(5):1651-1686.

[28] Schapire R, Singer Y (1999) Improved boosting algorithms using confidence-rated predictions. Machine Learning, 37(3):297-336.

[29] Opitz D, Maclin R (1999) Popular Ensemble Methods: An Empirical Study. Journal of Artificial Intelligence Research, vol. 11: 169-198

[30] Pierce W (1961) Improving reliability of digital systems by redundancy and adaptation. PhD thesis, Electrical Engineering, Stanford University

[31] Polikar R, Topalis A, Parikh D, Green D, Frymiare J, Kounios J, Clark C (2008) An ensemble based data fusion approach for early diagnosis of alzheimer disease. Information Fusion, 9:83-95.

[32] Probst P, Boulesteix A (2017) To tune or not to tune the number of trees in random forest? Journal of Machine Learning Research 18:1-18

[33] Van Rossum G, Drake FL (2009) Python 3 Reference Manual. Scotts Valley, CA.

[34] Vapnik V (1995) The Nature of Statistical Learning Theory. Springer.

[35] Vapnik V (1998) Statistical Learning Theory. Wiley.

[36] Vapnik V, Chervonenkis A (1971) On the uniform convergence of relative frequencies of events to their probabilities. Theory of Probability and its applications, XVI(2):264?280.