Universidad Nacional Autónoma de México

Facultad de Ciencias

Boosting con árboles de decisión y Random Forest

T E S I S

que para obtener el título de:

Actuario

presenta:

Luis Carlos Cortés Ruiz

Asesora:

Dra. Guillermina Eslava Gómez

Ciudad Universitaria, CD. MX., 2022

Datos del jurado

1. Datos del alumno
Cortés
Ruiz
Luis Carlos
Universidad Nacional Autónoma de México
Facultad de Ciencias
Actuaría

2. Datos de la asesora
Dra.
Guillermina
Eslava
Gómez

3. Datos de la sinodal 1
Mat.
Margarita Elvira
Chávez
Cano

4. Datos de la sinodal 2
Dra.
Lizbeth
Naranjo
Albarrán

5. Datos del sinodal 3
Dr.
Arturo
Reding
Bernal

6. Datos de la sinodal 4
Act.
Yadira
Rivas
Godoy

7. Datos del trabajo escrito
Bosting con árboles de decisión y Random Forest
153 p.
2021

A mi mamá, quien me enseñó a leer,
y a mi hermano, con quien puedo contar.

Agradecimientos

Hoy, 8 de diciembre de 2021, me dispongo a escribir los agradecimientos de esta tesis, el paso que siempre supe sería el último, el momento antes de entregarla a la Biblioteca Central. Contemplo todo lo que ha pasado desde el 16 de junio de 2020, el día cuando le pedí a la doctora Guillermina Eslava que me asesorara. Mis últimas clases de la carrera por videollamada, el servicio social en el Hospital General aunque nunca he puesto pie en aquel lugar, las diferentes etapas de esta tesis. Cuando le preguntaba a Monse cuál gráfica sería mejor, cuando Sebastián reprodujo parte de mi código en su computadora para ayudarme, cuando esperaba a que mis experimentos en `R` terminaran de correr mientras veía *Queen's Gambit* con mi mamá, cuando mi hermano imprimió un borrador a las cinco de la mañana y Angel me acompañó entregarlo. Las entrevistas de trabajo que tuve y la oferta de trabajo que acepté. Los comentarios de mis sinodales y las correcciones que hoy terminé. No puedo creer que he llegado al día en el que puedo agradecer por el último año y medio.

Agradezco a mi mamá, quien me ha dedicado su vida. Una vida siempre sincera, valerosa e incansable. Me enseñó así que la vida no es perfecta, que los errores se perdonan, y que siempre estaremos juntos, sin importar la distancia. Por mantener viva la memoria de mi papá y enseñarme que no hay amor más verdadero, gracias.

Agradezco a mi hermano Alan, mi amigo desde hace 19 años, y a Luneta que nos acompaña desde hace 10. También a quien ha sido como mi hermana, Aby. A mi abuelito, Marisol, Lupita, Javier, Michel, Mariana y Alexia, a quienes siempre admiraré y querré de aquí a la Luna. A mi abuelita y Estela, quienes me han visto crecer.

Agradezco a mi asesora de tesis, la doctora Guillermina Eslava, por todo el tiempo que me dedicó. No pude haber pedido mejor acompañamiento, y esta tesis es solo una muestra más de su laudable trayectoria académica. A mis sinodales: la matemática Margarita Chávez, la doctora Lizbeth Naranjo (quien aprobó la tesis en mi cumpleaños), el doctor Arturo Reding y la actuaria Yadira Rivas, por sus invaluables observaciones. Les tengo admiración y respeto. A mis profesores y profesoras, desde el CENDI hasta la Facultad de Ciencias, con una mención especial a la maestra Karen Lanzguerrero.

Agradezco a una larga lista de amigos y amigas, que cada día sé que soy increíblemente afortunado de tener, en especial a Kenya, Fernando, Helena, Sofía, Fernanda y Jonathan. Y porque han marcados estos últimos tres años, agradezco a Monse, Angel y Sebastián: les quiero cerca, pero incluso si tomamos caminos diferentes, sus nombres estarán en este papel para siempre, y en mi corazón mientras palpite.

Espero seguir teniendo mucho que agradecer el resto de mi vida.

Índice general

Índice de tablas

Índice de figuras

Boosting con árboles de decisión y *Random Forest*

por

Luis Carlos Cortés Ruiz

Resumen

La primera parte de este trabajo se dedica a los diferentes propósitos que tiene la estadística (estimación, atribución y predicción) y cómo se desarrolló el aprendizaje estadístico o *machine learning*; se incluyen discusiones de expertos sobre el tema. Se introduce la notación usada, así como las funciones de pérdida a utilizar en los ajustes, tanto en casos de regresión como de clasificación. Se toma la oportunidad para presentar la descomposición en varianza y sesgo. Después, se presentan el error de predicción y diferentes maneras de estimarlo: error aparente, error de entrenamiento, error de prueba, *repeated training-test* y validación cruzada. Para ello, también se introducen los conjuntos *train*, *test* y *validation*, que surgen de la base de datos que desea estudiarse.

En la segunda parte se presentan los algoritmos centrales del trabajo, empezando por árboles de decisión. Se habla de su construcción usando funciones de pérdida en los casos de regresión y clasificación, los parámetros necesarios y características distintivas. Posteriormente, se introduce *Random Forest*, así como su predecesor *bagging*. Se presentan resultados para entender su eficacia, además del mecanismo para hacer las predicciones y sus parámetros. Se menciona lo que hace el remuestreo con *bootstrap* y cómo funciona dentro de estos algoritmos. Asimismo, se habla de la muestra *out-of-bag* para estimar el error de prueba y unas características de interpretación: importancia de variables y dependencia parcial por diversos métodos. El último algoritmo es *boosting* por gradiente: se dan a conocer las optimizaciones numéricas que se logran por medio de modelos aditivos y *steepest descent*. Se presentan tres diferentes versiones del algoritmo según las funciones de pérdida usadas, y se muestran aspectos adicionales como la regularización, la profundidad de los árboles y el submuestreo.

La última parte está dedicada a las aplicaciones con la interfaz `h2o` a través de `R`, con tres bases de datos; la primera de ellas es un caso de regresión, las otras dos se tratan de clasificación. Se muestran ajustes selectos y su análisis con información como parámetros y diferentes tipos de error seleccionados. La primera es `California Housing`, donde el propósito principal es predecir los precios de casas por grupos. Se complementa con aspectos como importancia de variables y dependencia parcial. La segunda base de datos es `MNIST`, donde se predicen cuáles dígitos del 0 al 9 escribieron las personas. En la tercera base, `Ancestry`, se predice la ascendencia de un grupo de personas con base en su genética. Se discute la importancia de variables con una comparación con V de Crámer. En cada ejemplo se ajustaron modelos de regresión, ya sea lineal (para el caso de una variable respuesta continua) o logística (para el caso de una variable respuesta categórica), con fines comparativos, además con un cotejo con los resultados de otras publicaciones donde fuese posible.

En los apéndices se incluyen el código de `R` y comentarios sobre el paquete `randomForest`.

Introducción

El paso veloz de los avances del internet y poder computacional en el siglo XXI han traído a la puerta de la estadística nuevos retos y oportunidades, particularmente en el área de la exploración, análisis y predicción de datos. Las herramientas y conocimientos que obtenemos en la licenciatura en actuaría en la Facultad de Ciencias de la UNAM nos hacen los candidatos ideales para enfrentar esta nueva era de información, y uno de los campos con perspectivas muy prometedoras es el aprendizaje estadístico automatizado o *statistical machine learning*. En esta tesis se abordarán dos algoritmos eficaces para la predicción que, aunque surgieron a finales del siglo XX, son usados ampliamente en la industria y la investigación actuales. Estos son *Random Forest*, propuesto por Leo Breiman, y *boosting* por gradiente, desarrollado por Jerome Friedman.

Ambos métodos se basan en un algoritmo antecesor, los árboles de decisión `CART` (*Classificacion and Regression Trees*), introducidos por los dos autores arriba mencionados, junto con Richard Olsen y Charles Stone. La decisión de presentar este tema surgió de una fascinación personal con el mecanismo de los árboles de decisión de llegar a predicciones, por medio de la división de una base de datos heterogénea en subconjuntos homogéneos. Dicha idea también permite una interpretación simple e intuitiva de la base de datos a la mano, a través de su representación gráfica. Esta fascinación continuó con el conocimiento de *Random Forest* y *boosting*: el primero es un conjunto de árboles, todos idénticamente distribuidos, donde se hace la predicción siguiendo un principio de democracia; el segundo construye árboles sucesivamente, cada uno con base en los que lo precedieron, para que, al terminar, la suma de todos sea la predicción final.

Aunque el enfoque es únicamente en estos dos algoritmos, las técnicas mostradas de preparación de los datos, aplicación de los algoritmos y selección y evaluación de los ajustes son estándares en la comunidad de *machine learning*, por lo que los mismos pasos puestos en práctica pueden ser usados en otros algoritmos, aquellos donde se busque predecir una nueva observación y a partir de una o más variables explicativas x, con base en un conjunto de datos observados.

En este sentido, el primer capítulo tocará una breve historia de la evolución del aprendizaje estadístico, empezando con un ejemplo familiar para los estudiantes y enseñantes de estadística: el modelo de regresión lineal. Esto da entrada a las diferentes metas de esta disciplina que, inspirado en el trabajo de Efron (2020), se discuten como estimación, atribución y predicción. La regresión lineal ajusta un modelo que *estima* coeficientes que expliquen un fenómeno a través de las variables explicativas, *atribuye* una significancia a dichos coeficientes y sirve para *predecir* información nueva. Con un enfoque en la predicción, se hablará de las funciones de pérdida, útiles para construir y evaluar algoritmos de predicción, además de su aplicación en la estimación del error de predicción. Serán explicadas las diferentes versiones de esa estimación: error aparente, errores de conjuntos de entrenamiento y de prueba, error por *repeated training-test* (referida en James *et al.* (2016) como *validation set approach*) y error por validación cruzada. Además, se mencionarán componentes importantes en los ajustes: el sesgo y la varianza.

El segundo capítulo es donde se presentan los algoritmos centrales de la tesis. Empezando por los árboles de decisión, tanto en el caso de clasificación (cuando la variable a predecir es categórica) como en el de regresión (cuando pertenece a los números reales). Se hablará de su construcción y técnicas para controlar su tamaño, y se complementará con imágenes para ilustrar la mecánica de los árboles. Aunque los árboles son altamente interpretables y con bajo sesgo (es decir, el valor esperado de la estimación no es muy diferente al valor real), pueden ser muy inestables, afectando la predicción datos nuevos.

Por ello, el capítulo continuará con la propuesta para tomar el bajo sesgo de los árboles y reducir su varianza: *Random Forest*, además de su antecesor *bagging*. Ambos se basan en *boostrap*, una herramienta de remuestreo poderosa, y se basan en una colección de árboles, cuyo promedio reduce dramáticamente la varianza. A pesar de ser un algoritmo totalmente

predictivo, ha habido esfuerzos para tener maneras de interpretarlo, como lo son la importancia de variables y la dependencia parcial, que también serán revisadas.

Estas últimas dos herramientas también pueden ser usadas en *boosting* por gradiente, que usa un modelo aditivo para alcanzar la predicción, así como la optimización por el gradiente de la función de pérdida para definir sus parámetros. Aquí se presentarán tres versiones del mismo algoritmo, cada una construida con una función de pérdida diferente.

En el capítulo 3 se aplicarán los dos algoritmos de interés en tres ejemplos distintos, cada uno con distintas características y tamaños. El primero es la base de datos `California Housing`, un caso de regresión en el que se estimará valores de casas agrupadas en conjuntos con base en información como número de habitantes y ubicación. La segunda base de datos es `MNIST`, un problema de clasificación, en el que se usarán imágenes de números manuscritos para predecir qué dígitos son, entre el 0 y el 9. La tercera base de datos es `Ancestry`, en la que con información genética se clasificarán personas en cuatro ascendencias posibles.

En cada uno de los casos anteriores se presentarán en tablas los ajustes elegidos, además del respectivo proceso de selección y evaluación. Se harán comparaciones con los modelos de regresión para mostrar las diferencias que existen. Cuando es posible, como en `California Housing` y `Ancestry`, se usarán los métodos de interpretación. A lo largo de este capítulo, se hará uso de gráficas en una variedad de situaciones para facilitar la lectura.

El lenguaje de programación usado será `R`. Se ejemplificará con la base de datos de `California Housing` el uso del paquete `Random Forest`, no obstante, la interfaz en `R` de `h2o`, una plataforma de *machine learning*, será usada a través de todos los ejemplos. En un anexo se encontrará el código escrito a lo largo de los tres ejemplos, así como aquél de las ilustraciones de los árboles de decisión.

Pensando en las dificultades que surgieron en la lectura e investigación de los temas, esta tesis fue escrita de la manera más clara posible, como me hubiese gustado que a mí me lo explicaran. Espero haber tenido éxito en esa tarea.

Capítulo 1

Aprendizaje estadístico

> *But the cleverest algorithms are no substitute for human intelligence [...] Take the output not as absolute truth, but as smart computer generated guesses that may be helpful in leading to a deeper understanding of the problem.*
>
> — Leo Breiman y Adele Cutler

El objetivo de este capítulo es establecer lo que se logra con los algoritmos de aprendizaje estadístico y una muy breve historia de la evolución de esta disciplina. Para ello se introducirá la notación necesaria, así como los casos de análisis que se revisarán en este trabajo: regresión y clasificación. Más adelante, se hablará de tres propósitos que tiene la estadística y las diferencias entre ellos; estos son la inferencia, la atribución y la predicción, siendo el principal interés de este trabajo la última de ellas. La distinción de estos tres enfoques está directamente inspirado en el trabajo de Efron (2020).

La segunda parte del capítulo se enfoca en los elementos necesarios para la construcción de algoritmos y la evaluación de ellos. Por una parte, se encuentran las funciones de pérdida, donde se presentan las más utilizadas; adicionalmente se presentan el sesgo y la varianza, aspectos importantes del aprendizaje estadístico. Por otro lado, se habla de los diferentes tipos de error que pueden tener los algoritmos, así como de diversas técnicas para estimarlos, una

de las cuales es validación cruzada. El capítulo cierra con comentarios adicionales sobre aquel método.

1.1. Estimación, atribución y predicción

"La estadística es la ciencia de aprender de la experiencia"[1] es una de las maneras posibles de referirse a esta disciplina de las matemáticas. En términos generales, se basa en la recolección, organización, análisis y presentación de datos, éstos pudiendo surgir de cualquier ámbito imaginable: información genética de un grupo de personas padeciendo la misma enfermedad, censos nacionales de población y vivienda, o la relación entre las estaturas de padres e hijos. De hecho, este último punto motivó el modelo de regresión lineal de Francis Galton en 1887, aunque ya existían antecedentes del inicio de aquel siglo por parte de Carl Friedrich Gauss y Adrien-Marie Legendre.

Un ajuste de este modelo comienza con un conjunto de datos $\boldsymbol{d}$:

$$\boldsymbol{d} = \{(x_i, y_i), i = 1, 2, \ldots, N\},$$

donde x_i es una sola observación de la variable aleatoria $X \in \mathcal{X}^N$, representando, en este caso, la estatura del padre (referida como variable independiente o explicativa). Cada x_i está pareada con un y_i, proveniente de $Y \in \mathcal{Y}^N$, la estatura del hijo (conocida como variable dependiente o respuesta). Los puntos de datos (x_i, y_i) son independientes e idénticamente distribuidos (i.i.d.) de alguna distribución P.

Se puede suponer que existe una relación $Y = f(X) + \varepsilon$. El segundo sumando es un error, o "ruido", con media cero y varianza σ^2. Se desea aprender de una verdad descrita por $f(X)$, pero que está oscurecida por ε en los datos que se tienen disponibles.

El modelo de regresión formulado para predecir la estatura y se expresa como:

$$Y = \beta_0 + \beta_1 X + \varepsilon, \tag{1-1}$$

[1] Efron y Hastie (2016)

que, en un ejemplo simplificado del caso de Galton, significa que la estatura y del hijo está determinada por $\hat{\beta}_1$ veces la estatura x del padre, más $\hat{\beta}_0$ y un error ε.

Este modelo puede ser usado cuando se tiene una variable Y cuantitativa, es decir, continua: los valores tienen un orden que refleja su naturaleza. Además, se generaliza al modelo de regresión lineal múltiple, en el que se considera más de una variable explicativa.

Pensemos, entonces, que se tienen p variables explicativas o predictoras. Estas pueden ser continuas, categóricas o binarias; es posible que las p variables sean todas de un solo tipo, o una combinación de ellas (es decir, algunas continuas y otras categóricas, por ejemplo). La representación matricial de los datos $\boldsymbol{d}$ con las características mencionadas es:

$$\boldsymbol{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N1} & x_{N2} & \cdots & x_{Np} \end{pmatrix}, \qquad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix},$$

donde x_{ij} representa el valor de la variable explicativa j en la observación i. Cuando un vector o matriz de variables explicativas tenga un tamaño diferente de N, entonces se usará un tipo de letra normal. En cambio, si tiene N componentes, se representará en letras negritas. Esto significa que esta tipografía se usará solo cuando se trate del conjunto de las N observaciones totales.

Una observación x_i con todos sus componentes, es decir, variables explicativas, se representa con un vector:

$$x_i = \begin{pmatrix} x_{i1}, & x_{i2}, & \ldots, & x_{ip} \end{pmatrix},$$

mientras que las N observaciones de una de las p variables X_j se escriben como:

$$\boldsymbol{x}_j = \begin{pmatrix} x_{1j} \\ x_{2j} \\ \vdots \\ x_{Nj} \end{pmatrix}$$

El siguiente es un modelo de regresión lineal múltiple con p variables explicativas:

$$Y = f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p, \tag{1-2}$$

que podría representar el ingreso económico de una persona en función de su educación, su puesto y antigüedad en el trabajo, entre otras variables.

Para enfrentar el problema de una variable cualitativa (por ejemplo, si una institución aseguradora caerá en insolvencia o no, que se puede codificar como $Y = \{0, 1\}$), Ronald Fisher propuso en 1936 el análisis discriminante lineal, para clasificar los datos según una respuesta dicotómica. Una década más tarde, la regresión logística surgiría como una alternativa más ampliamente aplicada, donde es posible estimar probabilidades $p(X) = P(Y = 1|X)$ a partir de

$$\log\left(\frac{p(X)}{1 - p(X)}\right) = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p. \tag{1-3}$$

En ambos métodos se han desarrollado generalizaciones que permiten estimar una respuesta de más de dos opciones, preferentemente llamados niveles, clases o factores (por ejemplo, el color de ojos de una persona). La notación descrita y los modelos presentados se inspiran en James *et al.* (2017) y Hastie *et al.* (2009).

Estimación y atribución

En el aprendizaje estadístico, los modelos mencionados se llaman de *regresión* cuando la variable respuesta es continua, o de *clasificación* cuando es categórica. Una primera tarea se da en la *inferencia*, es decir, a partir de una muestra de la población, se busca hacer conclusiones generales acerca de las características desconocidas de la misma, de acuerdo con Vázquez Alamilla, *et al.* (2019). La inferencia puede dividirse en dos propósitos: la estimación y la atribución.

El primer propósito es la *estimación*: los modelos estiman una función que refleje un fenómeno de interés, esto es, se pretende llegar a una verdad subyacente con una estructura científica lo suficientemente confiable, según Efron (2020). Esta verdad subyacente puede pensarse como una superficie recreada a partir de los datos disponibles $\boldsymbol{d}$ (ver figura 1-1).

En este sentido, el segundo propósito es la *atribución*, que se encarga de identificar cuáles variables explicativas tienen una relación significativa con la respuesta. Usualmente se desea encontrar un subconjunto pequeño de ellas que modele la relación satisfactoriamente.

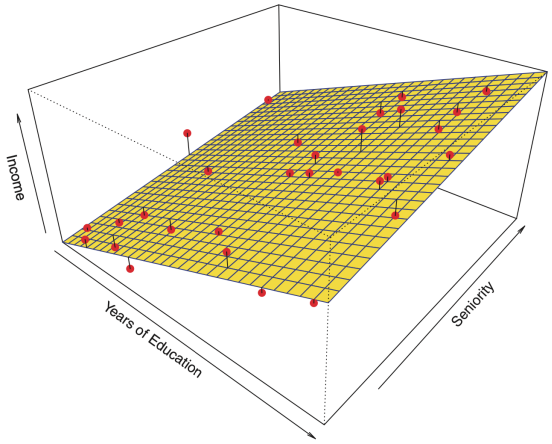


Figura 1-1: Superficie que modela el ingreso (*income*) en función de años de educación y antigüedad (*years of education* y *seniority*, respectivamente). Fuente: James, *et al.* (2017).

Predicción

El tercer propósito que deben cumplir estos modelos es el de predecir puntos inéditos de Y a partir de nueva información de X. Las grandes cantidades de datos generadas gracias al internet, además del avance acelerado del poder computacional desde fines del siglo pasado, han dado paso a algoritmos que, por decirlo de una manera, viven y mueren por esta tarea. Esto no quiere decir que los modelos tradicionales no sean aptos para ello: además de inferir características de una población a partir de un conjunto de datos, y de atribuir significancia a las variables explicativas, pueden realizar predicciones. En el modelo de regresión lineal (1-2), por ejemplo, pueden introducirse nuevos valores conocidos de las variables explicativas para conocer el valor de la respuesta. De la misma manera, en el caso de una variable respuesta categórica, el modelo de regresión logística (1-3) devuelve probabilidades. Éstas pueden ser usadas para predecir si se elige un umbral: por ejemplo, si $p(X) > 0.5$, entonces $X = 1$, o si $p(X) \leq 0.5$, entonces $X = 0$, para el caso binario mencionado anteriormente.

Sin embargo, han surgido propuestas dedicadas completamente a la predicción, como *Ran-*

dom Forest, *support vector machine* y *neural networks*. Estas ideas nacieron de la necesidad de trabajar con enormes bases de datos, mucho mayores que aquellas que se tenían cuando se desarrolló la teoría estadística clásica. El tamaño de dichas bases de datos puede definirse tanto por el número de observaciones, como por la cantidad de variables explicativas. En general, se busca una regla

$$\hat{Y} = \hat{f}(X),$$

es decir, una función que prediga a Y a partir de X.

Estas estimaciones $\hat{f}(X)$ no tienen como objetivo principal explicar la relación entre X y Y. Breiman (2001b) argumenta que la tarea de encontrar un modelo paramétrico como (1-2) que refleje un mecanismo complejo, originado en la naturaleza, alejó a la estadística de una amplia variedad de problemas, tanto científicos como comerciales. Es por ello que recomendó impulsar un enfoque logarítmico dedicado a la predicción, cuyo único supuesto es que las observaciones sean i.i.d. de una misma distribución. No desdeña los modelos paramétricos, ya que expone que tanto los datos como el problema a resolver guiarán la elección del modelo o algoritmo a implementar.

No obstante, el desarrollo de estos algoritmos ha descuidado los aspectos de estimación y atribución a cambio de su poderosa capacidad de predicción. Una razón es que no tienen las restricciones que los modelos tradicionales necesitan, como supuestos de linealidad, por ejemplo. Así, las aproximaciones algorítmicas poseen mayor libertad, sus funciones adoptando formas que se ajustan a una gran cantidad de fenómenos.

Estas relaciones predictivas son a menudo efímeras, haciendo a un lado la búsqueda de una verdad subyacente que trascienda el tiempo. En cambio, se ha desarrollado teoría que estudia las propiedades de los algoritmos, parte de la cual se presentará en este trabajo. Ante esta perspectiva, Efron (2020) ha declarado que "el uso de algoritmos de predicción para descubrimientos científicos dependerá en la demostración de su validez a largo plazo". D. R. Cox, el prominente estadístico británico, expresó que las soluciones algorítmicas pueden ser una alternativa útil y eficaz en ciertos contextos; la estabilidad de los predictores a través del tiempo y su necesidad eventual de recalibración son temas de importancia mencionados. Estas últimas afirmaciones

se encuentran en los comentarios a Breiman (2001b).

A pesar de las diferencias entre los dos enfoques, es posible afirmar que el término *aprendizaje estadístico*, aunque discutiblemente reciente, tiene una historia inherente a la de la teoría estadística. Con los avances presentados en el área algorítmica, esta disciplina ha adquirido un segundo adjetivo, estableciéndose actualmente como *aprendizaje estadístico automatizado* o *aprendizaje de máquina*, más publicitado como *(statistical) machine learning*.

En Efron (2020) se mencionan algunos esfuerzos para evaluar estos avances logarítmicos con métodos estadísticos tradicionales, así como los intentos de adaptar los modelos clásicos a las escalas de datos contemporáneas. En aquel artículo también se afirma que la estadística tiene el deber de desarrollar la teoría faltante para analizar los nuevos métodos, aunque resalta que los avances han sido escasos.

En la figura 1-2 se muestra una superficie de predicción generada por un árbol de decisión basándose en dos variables predictoras. En contraste, la figura 1-1 exhibió un plano generado por una regresión lineal. Las diferencias entre algoritmos de predicción y la regresión son comparados en la figura 1-3, donde se observa que los algoritmos siguen la tendencia de la regresión, pero de una manera discontinua e irregular. La construcción de la superficie según el árbol de decisión se revisará en el siguiente capítulo.

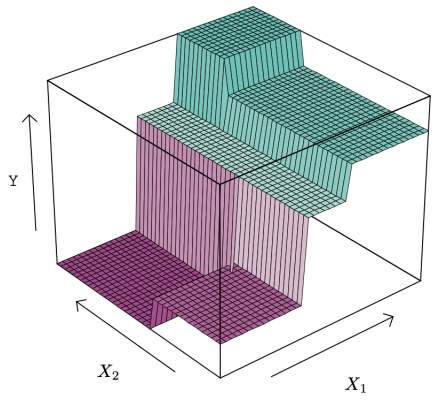


Figura 1-2: Superficie generada por un árbol de regresión. Fuente: James, *et al.* (2017).

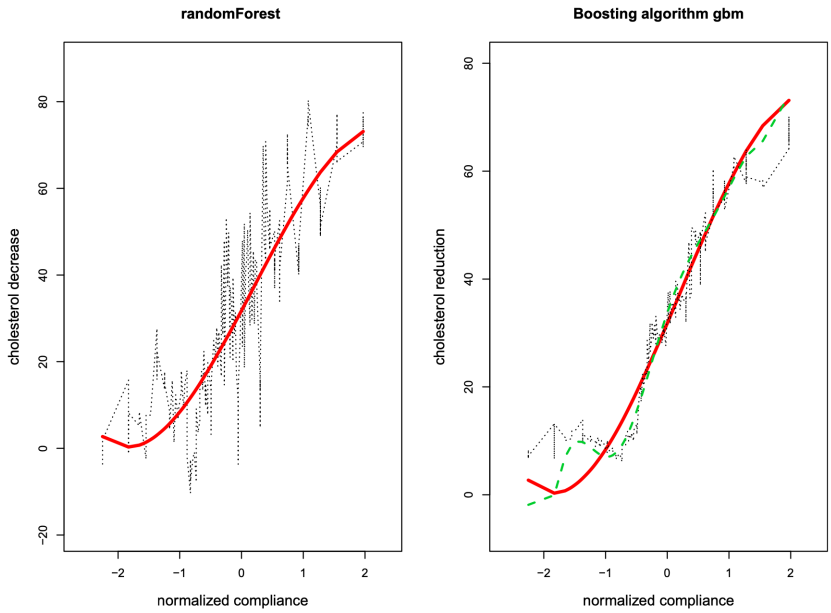


Figura 1-3: Estimación del decremento del colesterol (*cholesterol decrease*) en función del cumplimiento normalizado (*normalized compliance*; la proporción de medicamento tomada por el paciente en relación a la dosis recetada). La curva roja es la regresión cúbica por mínimos cuadrados (R^2 ajustada=0.48), comparada contra la estimación de los algoritmos *Random Forest* (izquierda) y *boosting*. La curva punteada verde es la estimación de una regresión polinomial de octavo grado. Fuente: Efron (2020).

1.1.1. Aprendizaje supervisado

De acuerdo con James *et al.* (2017), subsección 2.1.4, en el aprendizaje estadístico se hace una distinción entre aprendizaje supervisado y no supervisado. Los problemas discutidos hasta ahora, donde el conjunto de datos se compone de variable(s) explicativa(s) X y una variable respuesta Y, corresponden al dominio del aprendizaje supervisado: el objetivo es analizar la relación entre X y Y, ya sea para inferencia o para predicción. El análisis, por lo tanto, puede ser supervisado por medio de los resultados obtenidos, a través de comparaciones con las observaciones de Y.

Por otra parte, en el aprendizaje no supervisado no se ha identificado a una variable como la respuesta, por lo que las tareas se tratan de entender la estructura de las observaciones y encontrar asociaciones entre ellas. Los algoritmos aquí presentados son correspondientes al aprendizaje supervisado.

1.2. Funciones de pérdida

Un elemento imprescindible en el análisis supervisado es una medida que permita evaluar un ajuste dado. Para ello se tienen las funciones de pérdida (también conocidas como funciones de costo) $L(Y, \hat{f}(X))$, que cuantifican la diferencia entre los valores observados de Y y las predicciones $\hat{f}(X)$:

$$L(Y, \hat{f}(X)) : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_{\geq 0}, \tag{1-4}$$

de acuerdo con Bates *et al.* (2021).

El objetivo es minimizar su valor y son usadas tanto para el desarrollo de los algoritmos, como para hacer comparaciones entre distintos ajustes. En este último propósito, es de vital importancia entender cómo los ajustes se generalizan a datos no vistos anteriormente.

En esta sección se presentarán tres funciones para tareas de regresión: el cuadrado de los residuos, la pérdida absoluta y la pérdida de Huber. En tareas de clasificación, por otro lado, se encuentran el error de clasificación y la devianza. Se hablará de algunas variaciones y consecuencias. A menos que se indique lo contrario, estas expresiones son explicadas en las secciones 7.2 y 7.3 de Hastie *et al.* (2009).

1.2.1. Funciones de pérdida para regresión

Cuadrado de los residuos (y el error cuadrático medio)

La función de pérdida del cuadrado de los residuos,

$$L(Y, \hat{f}(X)) = (Y - \hat{f}(X))^2, \tag{1-5}$$

es ampliamente utilizada por sus ventajas analíticas en tareas de regresión. Su valor es pequeño en i cuando los valores Y y X son cercanos, pero crece rápidamente cuando comienzan a separarse.

En algunos casos (por ejemplo, cuando es necesario derivar la función de pérdida) es prefe-

rible trabajar con una ligera variación:

$$L(Y, \hat{f}(X)) = \frac{1}{2}(Y - \hat{f}(X))^2. \tag{1-6}$$

Este caso se revisará más adelante, cuando el algoritmo de *boosting* sea presentado. La fórmula (1-6) puede ser consultada en la sección 10.10 de Hastie *et al.* (2009).

Del cuadrado de los residuos se obtiene el error cuadrático medio, que es usado para comparaciones entre ajustes:

$$ECM = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{f}(x_i))^2, \tag{1-7}$$

que es un estimador de la precisión, considerando todas las observaciones del conjunto de datos. De esta expresión se obtiene la **descomposición en sesgo y varianza**, usando el valor esperado del cuadrado de los residuos para un valor dado $X = x_0$:

$$E\left[(Y - \hat{f}(X))^2 | X = x_0\right] = \text{Var}\left(\hat{f}(x_0)\right) + \text{Sesgo}^2\left(\hat{f}(x_0)\right) + \text{Var}(\varepsilon), \tag{1-8}$$

donde

$$\text{Var}\left(\hat{f}(x_0)\right) = E\left[(\hat{f}(x_0) - E[\hat{f}(x_0)])^2\right], \quad \text{Sesgo}\left(\hat{f}(x_0)\right) = E[\hat{f}(x_0)] - f(x_0),$$

y ε es el término de error proveniente de $y_0 = f(x_0) + \varepsilon$, con media cero, de acuerdo con la subsección 7.3 de Hastie *et al.* (2009).

Esto significa que, desde una perspectiva del cuadrado de los residuos, el error de una función estimadora se deberá a:

- Varianza: la magnitud de los cambios en sus resultados si se calcula con un conjunto de datos distinto. Un estimador tiene varianza grande si pequeños cambios en el conjunto de datos implica resultados muy distintos. Generalmente, los modelos de regresión lineales son estables. Los árboles de decisión, tópico del siguiente capítulo, son altamente variables.

- Sesgo al cuadrado: es la diferencia entre la media del verdadero $f(x_0)$ y el valor espe-

rado del estimador $\hat{f}(x_0)$. Dado que una regresión lineal supone que la relación entre la respuesta y las variables explicativas tiene un comportamiento determinado, entonces es propensa al sesgo. En contraste, los árboles de regresión no dependen de supuestos sobre los datos.

- La varianza de ε es la variación aleatoria inherente al nuevo punto x_0. A diferencia de los dos componentes anteriores, no puede ser mitigado por el ajuste, por lo que se considera irreducible.

Una consecuencia importante de esta descomposición es el *bias-variance trade-off* (intercambio o balance entre sesgo y varianza). Mientras más flexible sea un método, es decir, mientras mayor sea su adaptabilidad a un conjunto de datos, presentará mayor varianza. Simultáneamente, esa adaptabilidad típicamente reducirá el sesgo (de forma inversa, menor flexibilidad implicará una reducción de varianza y un incremento en el sesgo). Una ilustración de este concepto se encuentra más adelante en este capítulo (figura 1-6), una vez que se hayan introducido los conjuntos de entrenamiento y prueba.
La descomposición se deriva a continuación, donde se abreviarán $f = f(x_0)$ y $\hat{f} = \hat{f}(x_0)$.

$$\begin{aligned} E\left[(y_0 - \hat{f})^2\right] &= E\left[(y_0 - f + f - \hat{f})^2\right] \\ &= E\Big[\underbrace{(y_0 - f)^2}_{\text{a)}} + \underbrace{(f - \hat{f})^2}_{\text{b)}} + \underbrace{2(y_0 - f)(f - \hat{f})}_{\text{c)}}\Big]. \end{aligned}$$

a) $E\left[(y_0 - f)^2\right] = E\left[(f + \varepsilon - f)^2\right] = E[\varepsilon^2] = \mathrm{Var}(\varepsilon) + E^2[\varepsilon] = \mathrm{Var}(\varepsilon)$.

b) Recordando que $f(x_0)$ es determinista, a diferencia de $\hat{f}(x_0)$, ya que que este último

depende del conjunto de datos con el que se estimó:

$$\begin{aligned} E\left[(f-\hat{f})^2\right] &= E\left[(\hat{f}-E[\hat{f}]+E[\hat{f}]-f)^2\right] \\ &= E\left[(\hat{f}-E[\hat{f}])^2\right]+(E[\hat{f}]-f)^2+2E\left[(\hat{f}-E[\hat{f}])(E[\hat{f}]-f)\right] \\ &= \mathrm{Var}(\hat{f})+\mathrm{Sesgo}^2(\hat{f})+2E\left[\hat{f}E[\hat{f}]-E[\hat{f}]E[\hat{f}]-\hat{f}f+E[\hat{f}]f\right] \\ &= \mathrm{Var}(\hat{f})+\mathrm{Sesgo}^2(\hat{f})+2\left(E[\hat{f}]E[\hat{f}]-E[\hat{f}]E[\hat{f}]-E[\hat{f}]f+E[\hat{f}]f\right) \\ &= \mathrm{Var}(\hat{f})+\mathrm{Sesgo}^2(\hat{f}). \end{aligned}$$

c) Teniendo en mente que $E[y_0] = E[f+\varepsilon] = E[f] = f$,

$$\begin{aligned} E[(y_0-f)(f-\hat{f})] = E[y_0 f - f^2 - y_0\hat{f} + f\hat{f}] &= f^2 - f^2 - E[y_0\hat{f}] + fE[\hat{f}] \\ &= -E\left[(f+\varepsilon)\hat{f}\right] + fE[\hat{f}] \\ &= -E[f\hat{f}] - E[\varepsilon\hat{f}] + fE[\hat{f}] \\ &= -fE[\hat{f}] - E[\varepsilon]E[\hat{f}] + fE[\hat{f}] \\ &= 0. \end{aligned}$$

Pérdida absoluta y pérdida de Huber

La pérdida absoluta,

$$L(Y,\hat{f}(X)) = |Y-\hat{f}(X)|, \tag{1-9}$$

es una alternativa más robusta que el cuadrado de los residuos, en el sentido de que los valores atípicos (también llamados por su nombre en inglés, *outliers*) tienen menos influencia. Es decir, la pérdida del cuadrado de los residuos enfatiza grandes valores de $|Y-\hat{f}(X)|$, haciendo que aquellos valores tengan mayor representación en la estimación de f. Este aspecto cobra especial relevancia cuando la distribución de los residuos tiene colas pesadas.

También puede usarse el promedio de los errores absolutos sobre todos los puntos del ajuste, es decir, el error absoluto medio. Análogamente, es menos severo que el ECM para aquellos valores grandes de $|Y-\hat{f}(X)|$. Esta fórmula es la versión muestral de la expresión encontrada

en la sección 10.14.1 de Hastie *et al.* (2009):

$$EAM = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{f}(x_i)|. \tag{1-10}$$

La desventaja de los residuos absolutos es que no es una función diferenciable, por lo que su uso no es tan extensivo. Una posible solución a este problema es la función de pérdida de Huber, introducida en 1964 por Peter J. Huber, estadístico de la Escuela Politécnica Federal de Zúrich, y encontrada en la sección 10.6 de Hastie *et al.* (2009):

$$L(Y, \hat{f}(X)) = \begin{cases} (Y - \hat{f}(X))^2 & \text{si } |Y - \hat{f}(X)| \leq \delta \\ 2\delta|Y - \hat{f}(X)| - \delta^2 & \text{si } |Y - \hat{f}(X)| > \delta, \end{cases} \tag{1-11}$$

donde $\delta = \text{cuantil}_\alpha\{|y - \hat{f}(X)|\}$. Aquí debe elegirse un valor $\alpha \in (0, 1)$.

La pérdida de Huber tiene una versión alternativa, que es equivalente a aquella del cuadrado de los residuos (expresión 1-6). Similarmente, es usada en el algoritmo *boosting*. Se encuentra en Friedman (2001), aunque se basa en el trabajo de Huber ya referido.

$$L(Y, \hat{f}(X)) = \begin{cases} \frac{1}{2}(Y - \hat{f}(X))^2 & \text{si } |Y - \hat{f}(X)| \leq \delta \\ \delta(|Y - \hat{f}(X)| - \frac{\delta}{2}) & \text{si } |Y - \hat{f}(X)| > \delta, \end{cases} \tag{1-12}$$

Esta función, en ambas versiones, cumple con las ventajas analíticas del cuadrado de los residuos y con la robustez de la pérdida absoluta, pues emula a la primera función en los valores $|Y - \hat{f}(X)|$ cercanos a cero y a la segunda función en todos los demás. Este comportamiento puede visualizarse en la figura 1-4 más adelante. Según Friedman (2001), el punto de transición δ define los valores residuales que pueden considerarse *outliers*. Estas son las observaciones que pueden ser modificadas arbitrariamente sin degradar la calidad del resultado.

La pérdida de Huber usualmente es utilizada en los ajustes con el algoritmo *boosting*, y no como una medida de comparaciones entre ajustes como en los promedios del cuadrado de los residuos (expresión 1-7) y la pérdida absoluta (expresión 1-10).

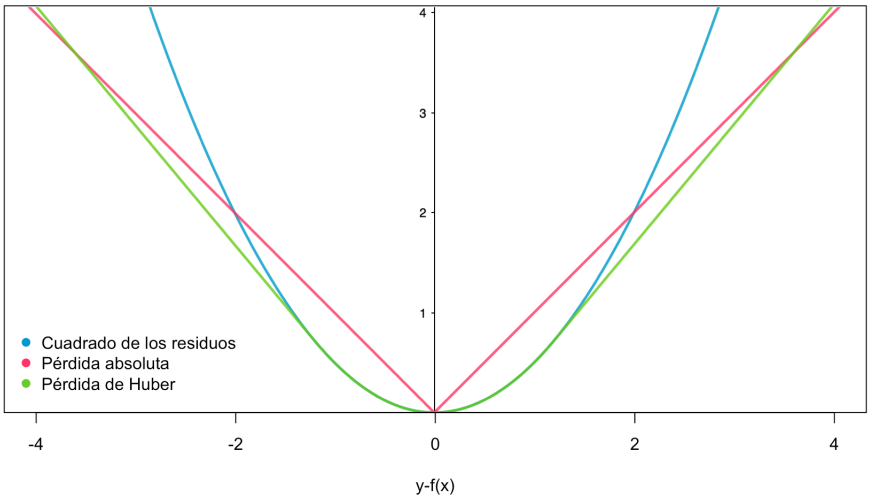


Figura 1-4: Comparación de las tres funciones de pérdida revisadas para el caso de regresión. $\delta = 1.2$ en la función de Huber. Tanto en esta última como en el cuadrado de los residuos, se presenta la versión multiplicada por $\frac{1}{2}$ (expresiones 1-6 y 1-12).

1.2.2. Funciones de pérdida para clasificación

Error de predicción y la matriz de confusión

En el contexto de clasificación, la función de pérdida más usada para evaluar un ajuste es el error de predicción:

$$L(Y, \hat{f}(X)) = \mathbb{1}_{\{Y \neq \hat{f}(X)\}}. \tag{1-13}$$

Su versión muestral, como es vista en la sección 10.1 de Hastie *et al.* (2009), es:

$$\text{Error} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_{\{y_i \neq \hat{f}(x_i)\}}. \tag{1-14}$$

A veces es llamada tasa de error: naturalmente, su valor siempre estará entre 0 y 1, por lo que puede ser representada como porcentaje.

Una herramienta útil que permite conocer el error de predicción por clase, y no de manera general, es la matriz de confusión. Ello permitirá observar si existen clases en los que el predictor cometió más errores; dependiendo del contexto, puede haber interés en disminuir el error en una clase en particular (en la detección de una enfermedad, es más grave un falso negativo que

un falso positivo, por ejemplo). Un ejemplo tomado de James *et al.* (2017) se muestra en la tabla 1-1, donde la tarea era clasificar correos electrónicos entre `spam` y `no spam`.

Reales	**Predicciones**	
	`spam`	`no spam`
`spam`	58.3 %	2.5 %
`no spam`	3.0 %	36.3 %

Tabla 1-1: Matriz de confusión para clasificación de correos electrónicos en dos posibles clases. La diagonal es la tasa de predicción correcta, mientras que las otras entradas son las tasas de error para cada clase. La tasa de error general es de 5.5 %.

Devianza

En las tareas de clasificación, los posibles valores de Y estarán en el conjunto de clases $1, 2, \ldots, K$. Es posible estimar las probabilidades $\hat{p}_k(X) = P(Y = k|X)$.

En la sección 10.5 de Hastie *et al.* (2009), se presenta el caso binomial ($Y \in \{0, 1\}$) de la devianza, también conocida como entropía y logverosimilitud negativa:

$$L(Y, \hat{p}(X)) = -[Y \log(\hat{p}(X)) + (1 - Y)\log(1 - \hat{p}(X))]. \tag{1-15}$$

El caso general para K clases, encontrado en la sección 10.6 de la anterior referencia, que será usada en los algoritmos a presentar:

$$L(Y, \hat{p}(X)) = -\sum_{k=1}^{K} \mathbb{1}_{\{Y=k\}} \log(\hat{p}_k(X)). \tag{1-16}$$

Esta expresión proviene del desarrollo de la regresión logística, donde se maximiza la logverosimilitud de la distribición multinomial (equivalente a minimizar la logverosimilitud negativa):

$$l(\theta) = \sum_{i=1}^{N} \log(p_k(x_i; \theta)). \tag{1-17}$$

Y donde posteriormente se puede calcular su devianza nula, es decir, menos dos veces la logverosimilitud, según lo visto en la sección 4.4.1 de Hastie *et al.* (2009). De forma más general, la devianza entre dos densidades $f_1(y)$ y $f_2(y)$ de la familia exponencial se ve en Efron (2010),

apéndice A, como:

$$D(f_1, f_2) = 2 \int_{\mathcal{Y}} f_1(y) \log \left(\frac{f_1(y)}{f_2(y)} \right) dy,$$

donde se integra sobre el espacio común $\mathcal{Y}$.

Aunque el error de predicción es una medida sencilla de entender, que da una imagen muy clara del rendimiento del ajuste, complementar esa cantidad con la devianza puede indicar la calidad de las predicciones. Por ejemplo, considere un ajuste de un caso binomial, codificado como 0 o 1, donde se consideran las probabilidades $\hat{p}(x)$ mayores a 0.5 (el umbral típico) como correspondientes a la clase 1. La devianza correspondiente a $\hat{p}(x) = 0.51$, es de $-\log(0.51) \approx 0.67$, mientras que $\hat{p}(x) = 0.9$, que es más decisivamente de la clase 1, es de $-\log(0.9) \approx 0.10$. Recordando que las funciones de pérdida deben minimizarse, es claramente preferible elegir un ajuste que asigne a una observación dada una probabilidad mayor de pertenecer a su verdadera clase, y no solo clasificarla correctamente como tal.

1.3. Selección y evaluación del ajuste

Se ha revisado en la sección anterior que cada punto de los datos observados trae consigo *ruido* irreducible, o un pequeño error ε. Esto implica que todo ajuste que realice una predicción sobre información nueva no podrá ser perfecto, por más que el algoritmo elegido sea flexible y se adapte a cualquier caso de análisis. Si un ajuste describe estrechamente a un primer conjunto de datos, es probable que tenga problemas para generalizarse a otras muestras de la misma distribución y que, por tanto, no sea útil para predecir nuevos conjuntos de datos. Esto representa un riesgo común en el aprendizaje estadístico: el **sobrejauste**.

Para evitar este problema, se han desarrollado metodologías que permitan **seleccionar** ajustes y **evaluar** su poder de generalización. En este sentido, es de interés especial conocer el error de predicción

$$Err = E[L(Y, \hat{f}(X))], \tag{1-18}$$

como se presenta en la sección 12.2 de Efron y Hastie (2016). Un primer intento por estimar

esa cantidad es el error aparente

$$\overline{Err} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i)), \tag{1-19}$$

es decir, la media de la función de pérdida sobre el conjunto de datos con el que fue entrenado. Esta cantidad usualmente es una estimación optimista del error de predicción, puesto que se evalúa el desempeño de la regla de predicción sobre datos que "ya conoce".

Por ello, cuando se tiene acceso a grandes cantidades de datos, es ideal tener tres subconjuntos, elegidos aleatoriamente de la base de datos disponible: el de entrenamiento (*training*) para realizar los ajustes, el de validación (*validation*) para estimar sus errores de predicción en información nueva y elegir a uno de ellos, y el de prueba (*test*) para evaluar el error de generalización únicamente del ajuste elegido. Esta partición se ilustra en la figura 1-5.

Figura 1-5: Partición de un conjunto d en los tres subconjuntos. Una sugerencia, como aquí se muestra, es 50 %-25 %-25 %. Gráfico inspirado en Hastie *et al.* (2009).

El error calculado con el primer conjunto es usualmente referido como *training set error* y *training error* en Efron (2020) y James *et al.* (2017), respectivamente. El error aparente (1-19) es, entonces, un error de entrenamiento, solo que calculado con el total de las N observaciones.

Cuando el error se estima con un conjunto de datos diferente al que se usó para entrenar al algoritmo, entonces se le llama *test set error* o *test error* (error de prueba):

$$Err_{\mathcal{T}} = E[L(Y, \hat{f}(X))|\mathcal{T}], \tag{1-20}$$

donde X y Y son elegidos independiente y aleatoriamente de la misma distribución subyacente P que el conjunto de entrenamiento $\mathcal{T}$, en el cual se basa el ajuste en cuestión. $Err_{\mathcal{T}}$ es una

cantidad aleatoria que depende del conjunto de entrenamiento, y el error de prueba se refiere específicamente a ese conjunto.

De acuerdo a la sección 12.2 de Efron y Hastie (2016), este es un estimador insesgado de (1-18). Además,

$$Err = E[L((Y, \hat{f}(X))] = E[Err_{\mathcal{T}}] \tag{1-21}$$

es la esperanza a través de posibles conjuntos de entrenamiento provenientes de P. Las últimas tres fórmulas fueron encontradas en la sección 7.2 de Hastie *et al.* (2009).

El objetivo es encontrar ajustes que satisfactoriamente disminuyan el error de prueba (1-20), no solo el error de entrenamiento. La figura 1-6 ilustra el *bias-variance trade-off*, que explica lo necesario para encontrar un error de prueba mínimo.

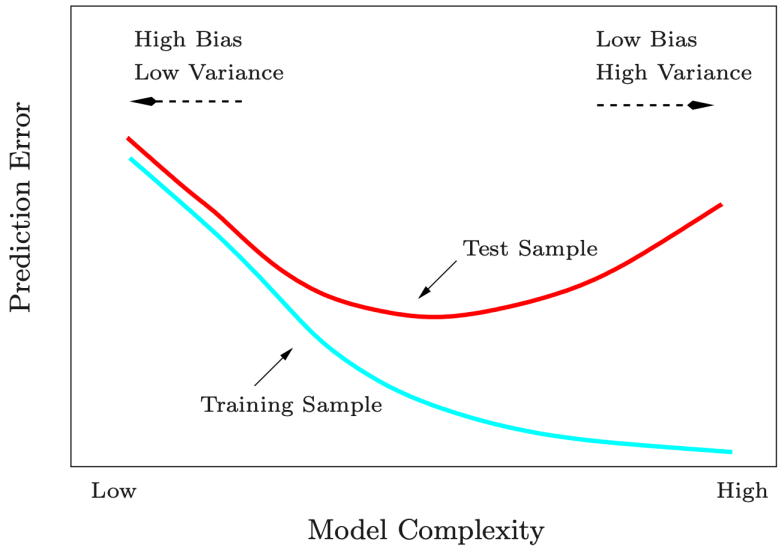


Figura 1-6: Ilustración de *bias-varinace trade-off* en función de la complejidad del modelo y su reflejo en los errores de los conjuntos de entrenamiento y prueba. Fuente: Hastie *et al.* (2009).

Un modelo complejo (con mayor flexibilidad y número de variables explicativas) se basará fuertemente en las estructuras particulares del conjunto de entrenamiento, disminuyendo el sesgo, pero aumentando la varianza y por ende, el error de prueba. Por otro lado, un modelo de baja complejidad tendrá baja varianza, resultando en predicciones muy similares en diferentes observaciones. Esto ocasionará un alto sesgo, ya que el valor esperado de la predicción estará alejado del valor real que se pretende estimar. Entonces, existe un modelo de complejidad media

que devolverá el error de prueba mínimo.

Aunque elegir aleatoriamente los conjuntos de entrenamiento, validación y prueba resulta en una estimación honesta del error de prueba, Efron (2020) señaló un potencial problema de este acercamiento: *The Training/Test Paradigm*. En un experimento de datos clínicos con *Random Forest* y *boosting*, los dos algoritmos que trata este trabajo, probó a ir en contra de la aleatoriedad que se requiere para elegir el subconjunto de entrenamiento. Al utilizar el orden de los datos del personal médico que los recolectó, sus ajustes tuvieron un error de prueba más alto que cuando eligió al azar el subconjunto de entrenamiento. Ante ello, especuló que diferencias metodológicas a través del tiempo pudieron haber influido en la recolección de los datos (resultando en que el conjunto de las primeras observaciones fue medido de forma distinta a otros subconjuntos), por lo que información del pasado puede no ser relevante para predecir información del futuro. Esto puede ser entendido como un cambio en el mecanismo de generación de datos: un algoritmo entrenado con datos de la distribución P puede no ser eficiente con datos de una distribución $\tilde{P}$.

Teniendo en cuenta este problema, resaltó que el punto fuerte de los algoritmos se encuentra en el trabajo con fenómenos efímeros, en donde surgen vastas cantidades de datos con alta frecuencia: detección de fraude en grandes bancos, recomendaciones de videos y películas, reconocimiento facial, etcétera. En estos escenarios, la selección aleatoria del conjunto de entrenamiento tiene sentido, puesto que se acepta un alcance limitado en la validez temporal de los datos. Los algoritmos, entonces, pueden ser actualizados con el flujo constante de nueva información.

Sin embargo, no siempre se tendrán conjuntos de datos suficientemente grandes para elegir conjuntos de entrenamiento, validación y prueba sin comprometer el alcance del ajuste ni la habilidad para evaluar su calidad. Debido a ello, se han desarrollado métodos de evaluación que permiten trabajar con conjuntos de tamaños más modestos. Al generar múltiples conjuntos de prueba, estos métodos también atienden el riesgo de que el único conjunto de prueba disponible resulte, por suerte de la selección, en una estimación optimista del error verdadero. Los métodos de remuestreo explicados a continuación usualmente no consideran tres particiones de los datos;

dos son útiles: entrenamiento y prueba, es decir, obtención de $\hat{f}(X)$ y cálculo de $L(Y, \hat{f}(X))$ para estimar el rendimiento en datos nuevos, sin un tercer subconjunto de observaciones para validar por segunda ocasión el error de prueba.

1.3.1. *Repeated training-test*

Este primer método divide aleatoriamente al conjunto de datos en los subconjuntos de entrenamiento y prueba. Elecciones típicas son una división en 70 %-30 % u 80 %-20 %, pero la decisión del investigador o investigadora depende del tamaño del conjunto completo particular. De este modo, se particionan los datos en los conjuntos ajenos $\mathcal{T}^{(train)} \cup \mathcal{T}^{(test)} = \{1, \ldots, N\}$. La estimación del error de predicción es:

$$\widehat{Err}^{(split)} = \frac{1}{|\mathcal{T}^{(test)}|} \sum_{i \in \mathcal{T}^{(test)}} L(y_i, \hat{f}(x_i)). \tag{1-22}$$

La estimación anterior es referida como *data splitting* en Bates *et al.* (2021). Puede tener alta varianza, puesto que depende de cómo resulte la división en entrenamiento y prueba. Además, no garantiza que todas las observaciones sean usadas para ajustar el modelo, lo que puede ser una limitante: en el aprendizaje estadístico, tener una mayor cantidad de observaciones lleva a los algoritmos a una mejor toma de decisiones. En la sección 5.1.1 de James *et al.* (2017) se habla de la práctica común de repetir este proceso (llamado *validation set approach* en esa fuente) un número elevado de veces, y promediar el resultado para obtener el error de prueba estimado. Andreas Buja, doctor en matemáticas y estadística por la Escuela Politécnica Federal de Zúrich y profesor en la Universidad de Pennsylvania, recomienda dividir a los datos aleatoriamente en proporciones de $\frac{2}{3}$ y $\frac{1}{3}$ y repetir el proceso de 100 a 500 veces, con la intención de estabilizar el promedio. Esta información fue encontrada en el capítulo 10 de su curso de estadística de 2019.[2]

[2] `http://www-stat.wharton.upenn.edu/~buja/STAT-961/`

1.3.2. Validación cruzada

Validación cruzada, o *cross-validation* en inglés, es un método que mejora en las limitaciones de *repeated train-test*. Aquí se divide el conjunto de datos en K partes iguales (llamándose, en este caso, "validación cruzada en K partes"). Si $K = 5$, una ilustración de este escenario puede encontrarse en la figura 1-7.

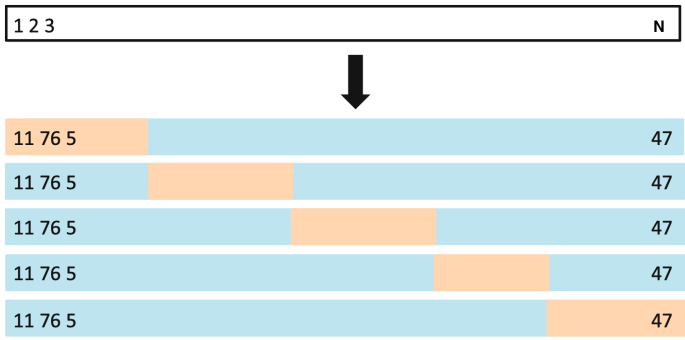


Figura 1-7: Se reordenan aleatoriamente las N observaciones (aquí, el orden cambia de $1, 2, 3, \ldots, N$ a $11, 76, 5, \ldots, N, \ldots, 47$), y se hacen cinco divisiones disjuntas del mismo tamaño. Cada división se usará una vez como conjunto de prueba (naranja), mientras que el resto se usará como conjunto de entrenamiento (azul). Fuente: James, *et al.* (2017).

En la k-ésima iteración se calcula el error de predicción de la k-ésima parte de la función estimada con las $K - 1$ partes restantes. Al repetirse para todas las partes, se promedian los K errores calculados. De esta manera, todas las observaciones son usadas, tanto para el ajuste, como para la evaluación. Si este proceso es repetido varias veces y de nuevo es promediado, entonces también se tendrán conjuntos de entrenamiento y prueba más diversos, lo que puede disminuir la varianza del estimador del error de prueba calculado.

Si $\kappa : \{1, \ldots, N\} \to \{1, \ldots, K\}$ es una función que indica a qué partición es asignada la observación i, y $\hat{f}^{-k}(x)$ es la función con la k-ésima partición removida, entonces el estimador del error de prueba es:

$$\widehat{Err}^{(CV)} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{-\kappa(i)}(x_i)). \tag{1-23}$$

Esta explicación se encuentra en la sección 7.10 de Hastie *et al.* (2009).

Típicamente se usa $K = 5$ o $K = 10$. El caso de $K = N$ tiene un nombre especial: *LOO-CV*,

leave-one-out cross-validation. El nombre lo explica correctamente, pues en cada iteración se entrena al algoritmo con $N-1$ observaciones para ser evaluado con la N-ésima que no fue usada, repitiéndose el proceso N veces.

Además de ser computacionalmente extenuante, *LOO-CV* es un estimador con mayor varianza que las alternativas como $K = 5$ o $K = 10$. Esto se debe a que los conjuntos de entrenamiento están altamente correlacionados, compartiendo $N-2$ de las observaciones: las estimaciones serán muy parecidas entre sí, con la posible implicación de que los errores de conjuntos de datos distintos serán diferentes al obtenido con *LOO-CV*. Por otro lado, validación cruzada de 5 partes tiene subconjuntos con intersecciones más pequeñas entre sí y, por ende, generará un estimador de menor varianza.

Tanto en *repeated training-test* como en validación cruzada, el ajuste final es el entrenado con el conjunto completo de datos. La repetición de estos procesos para su posterior promediado es preferible, mas no indispensable. A menudo, a las estimaciones del error de predicción con estos dos métodos se les llamará "error de prueba por validación cruzada" o "error de prueba por *repeated training-test*", debido a que se evalúan en conjuntos de prueba.

Comentarios sobre validación cruzada

En la sección 7.12 de Hastie *et al.* (2009) y más extensivamente en Bates *et al.* (2021) se discute, teórica y experimentalmente, si validación cruzada funciona para estimar apropiadamente el error de prueba condicional en el conjunto $\mathcal{T}$ (expresión 1-20) y el error de prueba esperado (expresión 1-21). Se encontró que es difícil hallar una estimación del error de prueba para un conjunto de entrenamiento en particular, dado solo ese conjunto. En cambio, la validación cruzada y métodos similares dan estimados razonables del error de prueba esperado: el resultado es un estimador del error de predicción promedio a través de conjuntos de datos hipotéticos de la misma distribución.

Ahora se comentará sobre aspectos a tomar en cuenta al realizar validación cruzada. Su objetivo es obtener una estimación insesgada del error de generalización, en el sentido de que los diferentes ajustes generados nunca son evaluados con observaciones con las que se haya

entrenado. Por ello, es de vital importancia evitar un error común en su aplicación.

Una metodología incorrecta es la siguiente:

1. En un problema con un número muy grande de variables explicativas, se hace una selección de variables previo a usar validación cruzada (por ejemplo, que muestran una correlación fuerte con la variable respuesta).

2. Implementar un algoritmo de predicción con el subconjunto elegido de variables explicativas.

3. Usar validación cruzada para la selección de parámetros y estimar el error de predicción.

El error yace en que todos los subconjuntos de validación cruzada pueden contener tantos datos que fueron usados como entrenamiento, como otros que fueron usados como prueba. No hay garantía de que los datos con los que se estimó el error de prueba en validación cruzada no hayan formado parte del entrenamiento del paso 2: en este caso se dice que se "filtró información" de ese paso al siguiente. Por lo tanto, no se ha imitado el proceso de estimar el error de prueba en un conjunto completamente nuevo de datos.

Una metodología correcta es la siguiente:

1. Dividir las observaciones en K subconjuntos.

2. Para cada subconjunto $k = 1, \ldots, K$:

    a) Elegir las variables explicativas con mayor correlación, usando todas las observaciones menos la del subconjunto k.

    b) Con este subconjunto de predictores, implementar los algoritmos de predicción, usando todas las observaciones menos la del subconjunto k.

    c) Usar la regla de predicción para clasificar a aquellas observaciones del subconjunto k.

Habiendo terminado el proceso para las K partes, puede estimarse el error de predicción con validación cruzada. Estos ejemplos fueron tomados de 7.10.2 de Hastie *et al.* (2009), y son

discutidos en una de las lecciones en video de James *et al.* (2017), impartidas por Trevor Hastie y Robert Tibshirani.[3]

[3] www.youtube.com/watch?v=r64tRyHFAJ8&list=PLOg0ngHtcqbPTlZzRHA2ocQZqB1D_qZ5V&index=23

Capítulo 2

Algoritmos de árboles de decisión

And I remember thinking "are we out of the woods yet?"

— Taylor Swift y Jack Antonoff

En 1984, Leo Breiman, Jerome Friedman, Charles Stone y R.A. Olsen publicaron *Classification and Regression Trees*, donde dieron a conocer el algoritmo `CART`, homónimo al título del libro. Cuando los datos son heterogéneos y no cumplen supuestos como linealidad, esta herramienta es una alternativa a métodos tradicionales como modelos de regresión, puesto que divide a los datos en subconjuntos ajenos más homogéneos, resultando en un ajuste interpretable y preciso. En la primera subsección de este capítulo se presentará la construcción de estos árboles, tanto en la presentación de regresión como en la de clasificación. Se incluirán gráficas que ilustren los conceptos, así como métodos de podado, en los que se reduce su tamaño, y las funciones de pérdida ideales para su uso.

En las secciones subsecuentes se hablará de poderosos algoritmos de predicción que surgieron a partir de los árboles de decisión, tomando sus características únicas para mejorar su poder de estimar nuevos datos, provenientes de la misma distribución de donde fue generada la información de entrenamiento. Estos son *Random Forest* y *boosting*.

2.1. Árboles de decisión

Dado el conjunto de datos $\boldsymbol{d} = \{(x_i, y_i), i = 1, 2, \ldots, N\}$, uno de los métodos para obtener $\hat{Y} = \hat{f}(X)$ son los árboles de decisión, que funcionan tanto con variables respuesta categóricas como continuas (numéricas). Un árbol de decisión `CART` genera una función por intervalos mediante los pasos siguientes:

1. Divide a los posibles valores de los predictores X en M regiones disjuntas $R_1, R_2, \ldots, R_M$.

2. Cada región R_m deberá tener asociado un valor constante γ_m, que es la predicción correspondiente a las observaciones asignadas a R_m según sus valores de X.

De esta manera se obtiene la función:

$$T(X; \Theta) = \sum_{m=1}^{M} \gamma_m \mathbb{1}_{\{X \in R_m\}}; \tag{2-1}$$

cada árbol $T(X; \Theta)$ está caracterizado por R_m y γ_m, $m = 1, \ldots, M$, identificados de manera conjunta por Θ. Esta expresión y serie de pasos pueden ser consultados en la subsección 9.2.2 de Hastie *et al.* (2009), y significa que si una observación de X está en la región R_m, entonces el valor estimado de Y para aquella observación es γ_m.

Los árboles de decisión son reconocibles por su representación gráfica y su facilidad para ser interpretados. Para entender de dónde proviene la representación de árbol y explicar sus componentes, se presenta un ejemplo de la base de datos `California Housing`, que se trabajará más a fondo en el próximo capítulo. El árbol de la figura 2-1 fue ajustado para predecir la variable de la mediana del precio de grupos de casas (en unidades de 10,000 dólares), y solo se usa la variable explicativa `MedInc`, la mediana del ingreso de los habitantes de dichos grupos de casas.

Las $n = 20,640$ observaciones de la base de datos pasan por el *nodo* u *hoja inicial*, en el que se evaluará su valor de `MedInc`. Es aquí donde se crea la primera partición, donde el punto de corte es 5. Las observaciones que tienen un valor menor a ello, pasan a un *nodo intermedio*, donde vuelven a ser particionadas, con la misma variable explicativa (no siempre es el caso, pudo haber sido usada cualquier otra variable), pero con 3.1 como el punto de corte. De esta

manera, se crearon tres regiones finales: de izquierda a derecha R_1, donde terminaron el 38 % de los registros y se predice un valor de `MedianHouseValue` de $\hat{\gamma}_1 = 1.4$; R_2, con el 41 % de los datos y una predicción de $\hat{\gamma}_2 = 2.1$. Por último, en la región R_3 está el 21 % de las observaciones y se estima que $\hat{\gamma}_3 = 3.3$.

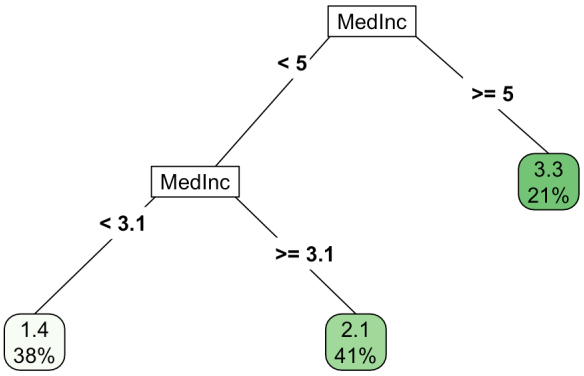


Figura 2-1: Árbol de regresión para predecir la variable `MedianHouseValue`, la mediana del precio de las casas en un grupo. Recursivamente, el árbol particionó a las observaciones según el valor de `MedInc`, la mediana del ingreso de los habitantes.

Como se habrá notado, todas las particiones deben resultar en dos subconjuntos, y éstas pueden hacerse tanto con variables categóricas como con continuas (como en el ejemplo mostrado). En este caso, $\hat{\gamma}$ es un valor continuo, pero los árboles de decisión también lidian con valores categóricos. Aprovechando que solo se usó una variable explicativa para particionar al conjunto de observaciones, puede visualizarse el resultado de éste en la figura 2-2.

El concepto es simple, pero es necesario atender cuestiones importantes:

- ¿Cómo se diferencian las variables respuesta continuas de las categóricas?
- ¿Cómo encontrar las mejores particiones?
- Dadas las regiones, ¿cuál es la mejor estimación de γ?
- ¿Qué tamaño tendrá el árbol?, es decir, ¿cuál es el número óptimo de observaciones en cada región?

Las preguntas anteriores son problemas de optimización y algunas de ellas se resuelven de

manera ligeramente diferente según la variable respuesta sea continua o categórica. Cuando es continua, se debe construir un *árbol de regresión*.

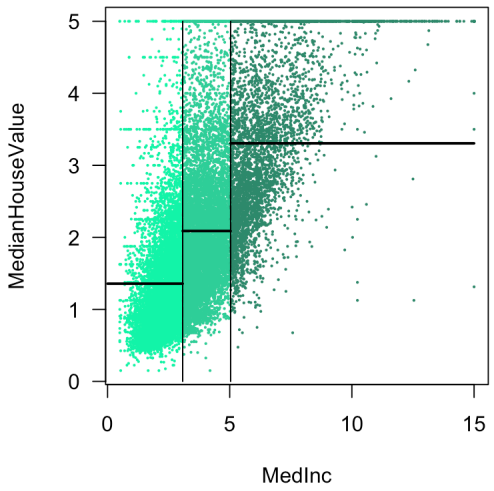


Figura 2-2: Las tres regiones finales del árbol de regresión de la figura 2-1. Las líneas horizontales son el valor de cada $\hat{\gamma}_i$, $i = \{1, 2, 3\}$. Estos valores son el promedio de la variable respuesta, `MedianHouseValue`, en cada una de las regiones. Gráfica inspirada en el curso de estadística de Andreas Buja de 2019, capítulo 11.

2.1.1. Árboles de regresión

Desde el comienzo, en el nodo inicial, donde se tienen todas las observaciones, y hasta llegar a los nodos finales, las biparticiones se hacen eligiendo un punto de corte t y un predictor X_j, resultando en las regiones izquierda y derecha:

$$R_{izq}(j,t) = \{X|X_j < t\} \text{ y } R_{der}(j,t) = \{X|X_j \geq t\} \text{ si } X_j \text{ es continua.} \tag{2-2}$$

Es posible que en el conjunto de variables explicativas existan del tipo continuo o categórico. En este segundo caso, entonces el punto de corte es uno de los k niveles, manejándose entonces como k variables *dummies* o binarias:

$$R_{izq}(j,t) = \{X|X_j = k\} \text{ y } R_{der}(j,t) = \{X|X_j \neq k\} \text{ si } X_j \text{ es categórica.} \tag{2-3}$$

El valor $\hat{\gamma}_m$ que se asigna a R_m es el que minimiza la diferencia con respecto a la variable

respuesta observada de los datos que se encuentran en la región (que son en total N_m), es decir, el promedio de las respuestas de cada observación:

$$\hat{\gamma}_m = \sum_{x_i \in R_m} \frac{y_i}{N_m} = \text{promedio}(y_i | x_i \in R_m), \tag{2-4}$$

Entonces, el problema de optimización a resolver es el siguiente, involucrando sumas de errores al cuadrado:

$$\min_{j,t} \left\{ \sum_{x_i \in R_{izq}(j,t)} (y_i - \hat{\gamma}_{izq})^2 + \sum_{x_i \in R_{der}(j,t)} (y_i - \hat{\gamma}_{der})^2 \right\}. \tag{2-5}$$

Estas expresiones pueden ser consultadas en la subsección 9.2.2 de Hastie *et al.* (2009).

En otras palabras, para cada predictor X_j se busca el punto de corte t tal que se minimice las sumas del cuadrado de los residuos que resulta de las diferencias entre la predicción $\hat{\gamma}$ y los valores observados de Y en la región, considerando R_{izq} y R_{der} simultáneamente. Para encontrar t, se toma el punto medio entre cada par de observaciones de X_j (o si la variable es categórica, se toma cada nivel k para manejarlo como variable *dummy*), y se calcula la expresión (2-5). En la figura 2-3 se ejemplifica este proceso, del cual resultó la primera partición del árbol de regresión presentado previamente en la figura 2-1.

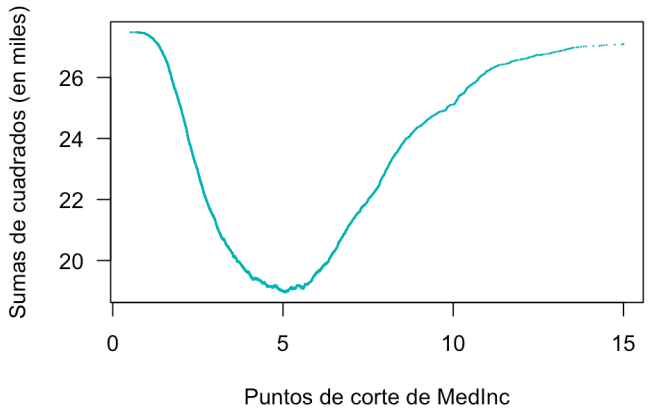


Figura 2-3: Sumas de residuos al cuadrado como se presenta en la expresión (2-5) para la variable `MedInc`. El punto mínimo es el valor 5.04 de dicha variable, y representa la suma de cuadrados más pequeña de entre todas las variables explicativas. Gráfica inspirada en el curso de estadística de Andreas Buja de 2019, capítulo 11.

Se hace el mismo ejercicio para cada variable explicativa, sin olvidar que primero se toma una variable predictora y un punto de corte para que posteriormente exista la $\hat{\gamma}$ correspondiente. Los árboles de decisión, sin importar si son de regresión o clasificación (a revisarse más adelante), funcionan con conjuntos de variables explicativas que incluyan ambos tipos, continuos y categóricos.

El mismo proceso puede repetirse con las observaciones resultantes en cada región, creando subregiones, y así iterativamente hasta que se defina un punto de paro. A este proceso se le llama **partición binaria iterativa**.

Es posible seguir creando subregiones hasta que cada observación de **d** tenga la suya, única para sí. Alternativamente, puede definirse un mínimo de observaciones que deben tener las regiones finales, como más adelante en la figura 2-4, donde se definió que se tuvieran 800. Un número alto de ellas implicará árboles menos profundos.

Otra forma de reducir el tamaño es dejar crecer un árbol T_0 hasta el tamaño máximo, y a través del *costo de complejidad* se "poda", es decir, se reduce su tamaño, hasta encontrar el mejor posible. Este método se explica a continuación.

Primero se crea un árbol T_0, que podría ser de tamaño máximo o de un número predeterminado de nodos finales. La tarea es encontrar un árbol $T \subset T_0$, y en lugar de revisar todos aquellos posibles T (pues sería un proceso largo y costoso computacionalmente), se generará una sucesión particular de ellos. Para ello, se colapsan, uno a uno, aquellos nodos internos que se encuentran inmediatamente antes de los nodos finales, y para cada subárbol resultante se calcula la suma del cuadrado de los residuos total:

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{\gamma}_m)^2, \tag{2-6}$$

donde $|T|$ es el número de nodos finales del árbol T. Esta suma de residuos al cuadrado aumentará cada que se colapse uno de los nodos, ya que perderá precisión sobre el conjunto de datos en el que se basa. Se tiene que seleccionar el subárbol que genere el menor incremento en esta medida de error, repitiendo este proceso sucesivamente hasta tener una sucesión de árboles que termine al llegar al del nodo inicial. Cada uno es únicamente representado por el tamaño $|T|$.

El siguiente paso se hace definiendo el criterio de costo de complejidad y calculándolo para cada subárbol T:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{\gamma}_m)^2 + \alpha|T|, \tag{2-7}$$

donde $\alpha \in \mathbb{R}^+$ es parámetro de costo que, al ser multiplicado por el número de nodos finales, hace que $C_\alpha(T)$ crezca. Para cada α se elegirá el árbol que minimice $C_\alpha(T)$. En este sentido, α se usa para regular el tamaño del árbol: si $\alpha = 0$, el árbol que minimiza el criterio es T_0, y en tanto α crezca, se elegirán subárboles T_α cada vez más chicos.

Lo último por definir es el mejor valor α. Un método es validación cruzada: elegir cierto número de $\alpha's$ distintos con sus correspondientes árboles T_α, provenientes de la sucesión decreciente de árboles podados con el criterio de costo de complejidad.

i) Se divide el conjunto de datos en K partes. Se separa cada parte k, y con el resto de las observaciones se realiza el procedimiento del párrafo anterior. Es decir, se aplica validación cruzada.

ii) Una vez conocidos los diferentes T_α, se calcula el error de cada T_α con la parte k no usada de los datos.

iii) Ya que se ha repetido esto en todas las partes k, se promedia el error (2-6) en función de cada α. Se supone a $\hat{\alpha}$ como el que produjo menor error (2-6) promediado.

Finalmente, se tiene $T_{\hat{\alpha}}$ como el mejor árbol podado. Este proceso y sus expresiones correspondientes fueron tomadas de la subsección 8.1.1 de James *et al.* (2017). El árbol que se presentó al inicio del capítulo es una versión podada del de la figura 2-1, con $\alpha = 0.06$.

Entre más grande sea el árbol, más ajustado estará al conjunto de datos en particular; mientras más pequeño sea, mayor será su posibilidad de generalizarse a otros conjuntos de datos provenientes de la misma distribución subyacente. Por lo tanto, con el parámetro α se busca un buen balance entre el ajuste y el error que pueden tener los árboles.

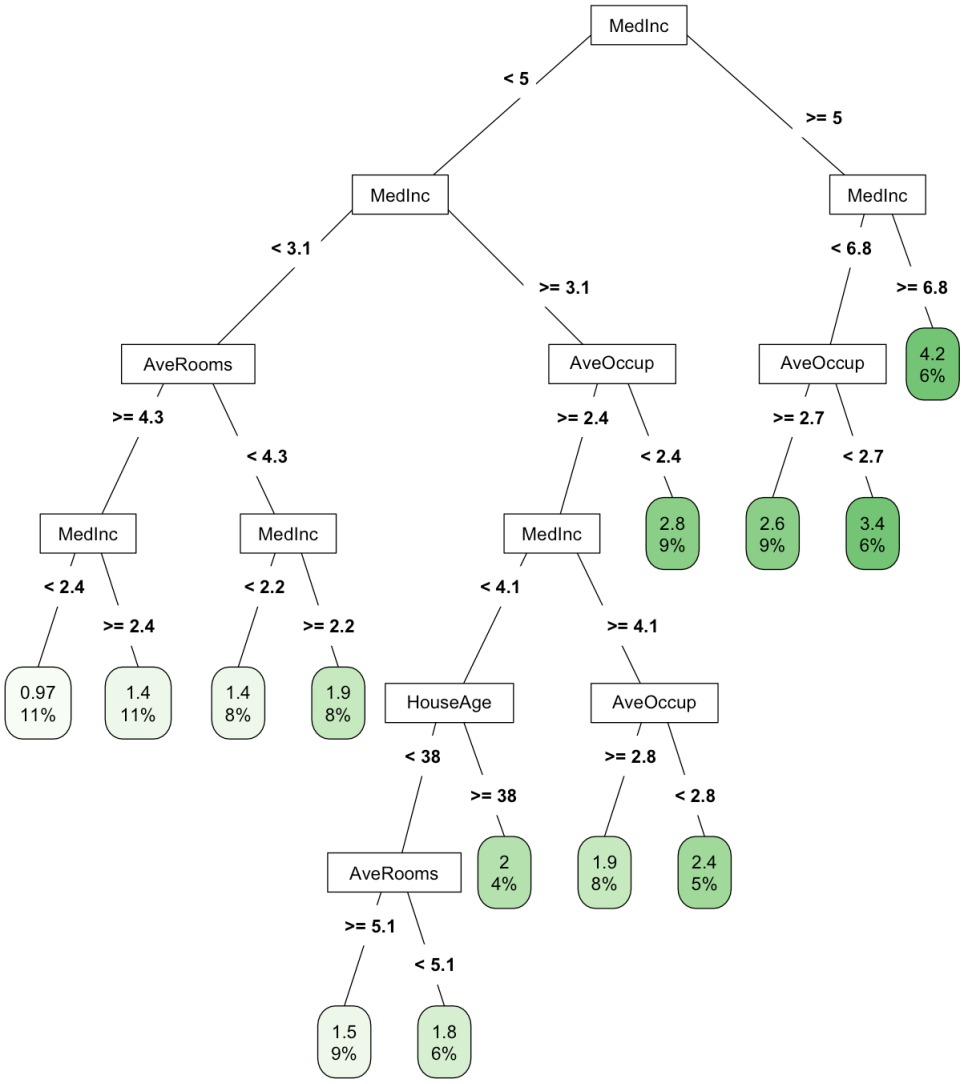


Figura 2-4: Árbol de regresión de la base de datos `California Housing`, donde se definió que las regiones finales tuvieran un mínimo de 800 observaciones. Se predice `MedianHouseValue` a partir de `MedInc` (mediana del ingreso), `AveRooms` (promedio de cuartos), `AveOccup` (promedio de personas), `HouseAge` (mediana de la edad de las casas). La mediana del ingreso es la variable más útil para particionar la base de datos. Otra característica es que entre menos personas en promedio tenga el grupo de casas, más elevado es la mediana del precio de ellas.

2.1.2. Árboles de clasificación

Cuando la respuesta es categórica, entonces la tarea se vuelve una de clasificación y deben hacerse ligeras modificaciones a la construcción de los árboles. La más evidente es la elección de la función a optimizar, pues el cuadrado de los residuos $(Y - \hat{f}(X))^2$ no es apto para esta situación.

Al ser creadas las regiones, el valor predicho no es el promedio de las observaciones asignadas a ella, sino que es la clase (i.e. categoría) más frecuente entre las que se encuentran ahí. Para ello, se calculan las proporciones en las que aparece cada clase k en el nodo final m, asociado con la región R_m. Estas pueden ser tomadas como probabilidades de clase $P(Y = k|X)$. Según Hastie *et al.* (2009):

$$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}_{\{y_i = k\}}, \tag{2-8}$$

donde N_m es el número de observaciones en la región R_m. Entonces la clasificación predicha es simplemente la más frecuente en el nodo:

$$k(m) = \arg\max_k \hat{p}_{m,k}. \tag{2-9}$$

Para crear el árbol y posteriormente podarlo, se tienen tres funciones como alternativa a la suma de residuos al cuadrado. La primera de ellas es el error de clasificación:

$$\frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}_{\{y_i \neq k(m)\}} = 1 - \hat{p}_{m,k(m)}. \tag{2-10}$$

Esta función de pérdida también es la preferida para evaluar el error del árbol.

Las dos funciones siguientes son más frecuentemente usadas por su derivabilidad, y por ende, en propiedades de optimización. Siguiendo lo presentado en Hastie *et al.* (2009):

- Índice de Gini:

$$\sum_{k=1}^{K} \hat{p}_{m,k}(1 - \hat{p}_{m,k}). \tag{2-11}$$

Es conocido como un índice de impureza de nodos porque, si las probabilidades de clase $\hat{p}_{m,k}$ son cercanas a los extremos 1 o 0, significa que el nodo contundentemente se inclina por una clase en particular. En ese caso, el índice será pequeño, denotando un nodo puro.

- Entropía:

$$-\sum_{k=1}^{K} \hat{p}_{j,k} log(\hat{p}_{j,k}). \tag{2-12}$$

También tiene la propiedad de medir la pureza de nodo, pues valores de $\hat{p}_{m,k}$ cercanos a 0 o 1 regresarán índices próximos a 0.

La elección de función a optimizar debe elegirse de acuerdo a su rendimiento con el conjunto de datos a la mano, y puede ser usada también para el podado. El índice de Gini es la función predeterminada en las implementaciones utilizadas en este trabajo.

Veamos más a detalle cómo se haría la partición binaria iterativa con el índice de Gini. Para cada partición posible se calculará G_m, el índice de Gini para el nodo m, considerando a m_{izq} y m_{der} como los nodos izquierdo y derecho resultantes de la partición, respectivamente, y a N_{izq}, N_{der} como su cardinalidad (el número de observaciones que cae en ellas):

$$G_m = N_{izq} \left[\sum_{k=1}^{K} (\hat{p}_{m_{izq},k})(1 - \hat{p}_{m_{izq},k}) \right] + N_{der} \left[\sum_{k=1}^{K} (\hat{p}_{m_{der},k})(1 - \hat{p}_{m_{der},k}) \right]. \tag{2-13}$$

La expresión anterior se deriva de lo encontrado en Efron (2020). Se entiende a $\sum_{k=1}^{K}$ como la suma sobre cada k clase de las K correspondientes a la variable respuesta categórica.

Para finalizar, se elige la variable y su respectiva partición que tengan el menor G_m, indicando los nodos más puros posibles, es decir, los que dividen a las observaciones en dos subconjuntos tan diferentes como se pueda. Como fue explicado, la clase predicha para el nodo es la más frecuente de las observaciones encontradas ahí.

En la figura 2-5 se presenta un árbol de clasificación usando el índice de Gini para la base de datos `Ancestry`, que será trabajada en el capítulo siguiente. La variable respuesta es `race`, con cuatro clases. Cada variable explicativa es una categórica de dos o tres niveles (i.e. `0`, `1` y `2`).

La ventaja principal de los árboles de decisión es su interpretabilidad, y que funcionan sin transformaciones de variables. Cuando el conjunto de datos es heterogéneo, una función suave como una regresión puede fallar en capturar la información subyacente. La forma de trabajo de los árboles de dividir a la información por partes homogéneas puede dar resultados más precisos en esos casos, adaptándose a una amplia variedad de conjuntos de datos.

Además, cuando el propósito no es la predicción, sino la explicación y estimación de una base de datos en particular, como suele ocurrir en la bioestadística, los árboles son útiles para seleccionar variables y descifrar las interacciones que tienen entre sí, aspectos muy útiles que consecuentemente pueden ser usados en modelos de regresión.

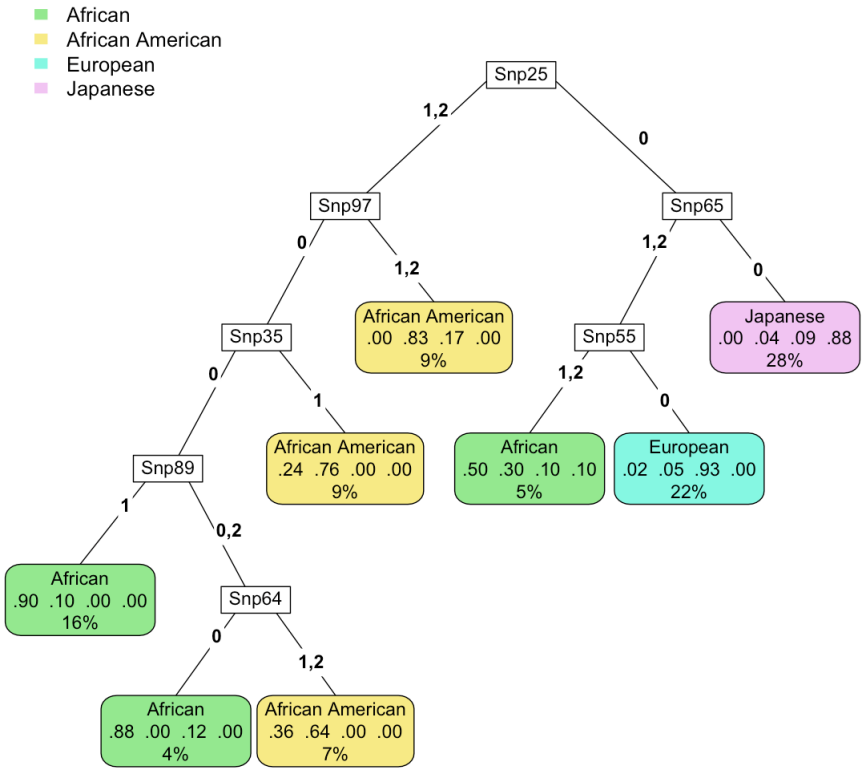


Figura 2-5: Árbol de clasificación para la base de datos `Ancestry`. La variable respuesta es de cuatro clases. Las variables explicativas son categóricas de hasta tres niveles. Cada nodo final tiene la proporción en que miembros de una clase están presentes. Por ejemplo, la región R_6, al lado izquierdo del nodo azul, está compuesta en 0.50 por `African`, 0.30 por `African American`, 0.10 por `European` y 0.10 por `Japanese`. El valor porcentual indica la proporción de observaciones de la base de datos contenidas en el nodo, independientemente de la clase. Se usó el índice de Gini en su construcción.

Sin embargo, su poder de predicción es menor a aquellos de otros métodos, debido principalmente a la alta varianza que tienen, producto del bajo sesgo que pueden alcanzar. Según Hastie *et al.* (2009), en la subsección 9.2.4: "La razón principal de esta inestabilidad es la naturaleza jerárquica del proceso: el efecto de un error en la parte de arriba es propagado a todas las particiones debajo". Una gráfica ilustrativa se encuentra en la figura 2-6, donde se observa el intercambio entre sesgo y varianza en función de la profundidad de los árboles. Se ajustaron los árboles a la base de datos `California Housing`.

En la búsqueda por mejorar esta característica, se han desarrollado métodos basados en los árboles de decisión, que son competitivos contra otros modelos y algoritmos de predicción, que mantienen el bajo sesgo de los árboles pero reducen su varianza.

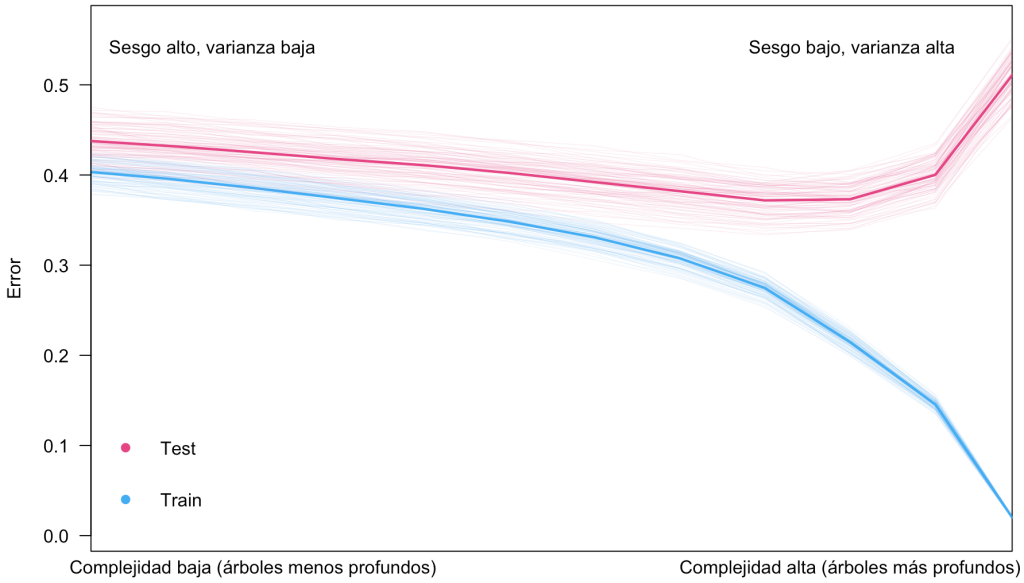


Figura 2-6: Evolución de los errores `train` y `test` en árboles de regresión para la base de datos `California Housing`. Se aumentó la complejidad del modelo, empezando con árboles con un tamaño mínimo de nodos finales de 100 observaciones (menos profundidad) hasta un tamaño mínimo de una observación (la mayor profundidad posible).
El error fue medido como la media del cuadrado de los residuos y fue calculado con `repeated training-test` (ver subsección 1.3.1), dividiendo 100 veces la base de datos en $\frac{2}{3}$ para entrenar y $\frac{1}{3}$ para evaluar. Las curvas tenues representan cada repetición, la línea sólida es el promedio de ellas. El menor error de prueba fue de 0.372, con el tamaño mínimo de 20 observaciones por nodo final. La gráfica es de creación propia pero inspirada en Hastie *et al.* (2009), sección 7.2.

2.2. *Random Forest*

Los árboles de decisión y la necesidad de conseguir una mayor precisión a través de ellos trajeron consigo *bagging* y *Random Forest*, los cuales se presentarán en esta sección. Para ello, se introducirá *bootstrap*, un tipo de remuestreo fundamental para mejorar el poder predictivo de conjuntos árboles de decisión, y posteriormente se explicará la manera en que estos algoritmos toman decisiones en materia de predicción, así como sus parámetros fundamentales. Una vez establecida esta base, se mostrará el algoritmo de *Random Forest*. Para cerrar la primera parte, se hablará brevemente de la resistencia al sobreajuste característica de este método de predicción.

La segunda parte de la sección trata de las características especiales de *Random Forest*: la muestra *out-of-bag* y su estimación del error de prueba, la importancia de variables y la dependencia parcial. Cuando sea conveniente, se ilustrará con ejemplos pequeños a lo largo de la sección.

Random Forest fue desarrollado por Adele Cutler y Leo Breiman en 2001, y explicado por primera vez en Breiman (2001a) posterior a sus ideas precursoras en *Bagging predictors* (1996) y *Arcing classifiers* (1998), donde se presentó *bagging*.

2.2.1. *Bagging*

Los árboles de decisión son sensibles a alteraciones en el conjunto de datos en el que se entrena, lo que significa que dos árboles pueden generar resultados significativamente diferentes entre sí si se les asignan dos conjuntos de datos distintos, aunque provengan de la misma distribución e incluso compartan un número de observaciones. En otras palabras, tienen alta varianza.

Entonces surge la idea de que, mientras los datos vengan de una misma distribución, se puede crear una serie de árboles, cada uno sobre un conjunto de datos distinto, y que la decisión tomada al final sea la indicada por el grupo, pudiendo ser la media en el caso de regresión, o la moda en el de clasificación.

Sin embargo, frecuentemente no se tiene un conjunto de datos lo suficientemente grande para poder asignar diferentes subconjuntos a cada árbol de la colección, por lo que se recurre a un

muestreo con reemplazo: el *bootstrap*. Teniendo un conjunto de datos d, la muestra *bootstrap* $\boldsymbol{d}^* = \{(x_i^*, y_i^*),\, i \in 1, \ldots, N\}$ es aquella donde cada observación puede ser elegida con la misma probabilidad, i.e. N^{-1}, en todos los ensayos. Por lo tanto, es posible que una fracción de las observaciones esté presente más de una vez en una sola muestra, implicando que otras no aparezcan.

El *bootstrap* es una herramienta estadística poderosa, y su uso en los árboles de decisión ha demostrado tener gran potencial predictivo. Puede consultarse más sobre *bootstrap* en Efron y Hastie (2016). A partir de esto, un primer acercamiento es ***bagging***: ***b****ootstrap* ***agg****regat****ing***. Aquí se tiene una serie de B árboles

$$\{T^{*b}(X),\, b = 1, \ldots, B\}, \tag{2-14}$$

cada uno usando una muestra *bootstrap* $\boldsymbol{d}^*$ distinta, construido como se vio en la subsección anterior. Podrían ser podados, pero es preferible dejarlos crecer al tamaño máximo como una estrategia para reducir el sesgo, pues árboles muy profundos crean nodos finales muy cercanos a los valores observados. Teniendo el sesgo bajo, el siguiente paso para mejorar la capacidad de predicción es reduciendo la varianza, que se puede conseguir promediando $T^{*b}(X)$.

Según la subsección 8.2.1 de James *et al.* (2017):

$$\frac{1}{B}\sum_{b=1}^{B} T^{*b}(x_i) = f_B(x_i), \tag{2-15}$$

donde $f_B(x_i)$ es el conjunto final de B árboles para la observación i.

Como los árboles son idénticamente distribuidos, el valor esperado de (2-15) es el mismo que el valor esperado de cualquier único árbol de la colección. Entonces, la reducción de sesgo logra mantenerse, y la varianza se reduce, pues la varianza de variables idénticamente distribuidas, pero no independientes necesariamente, es:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2, \tag{2-16}$$

donde ρ es la correlación entre pares de variables. Mientras B crece, el segundo término desaparece. La expresión (2-16) debe mantenerse en mente, pues será útil cuando se hable de *Random Forest*. Este párrafo sigue la explicación de la subsección 15.2 de Hastie *et al.* (2009).

En el caso de que Y sea una variable categórica con K clases, se selecciona la más votada, como se presenta en la sección 8.7 de Hastie *et al.* (2009):

i. Se calcula la proporción en que la clase k fue predicha:

$$\hat{p}_k^* = \frac{1}{B}\sum_{b=1}^{B} \mathbb{1}_{\{T^{*b}(x_i)=k\}}, \tag{2-17}$$

y con ello, un vector de proporciones $\hat{p}^* = \{\hat{p}_1^*, \ldots, \hat{p}_K^*\}$.

ii. La clase predicha es

$$f_B(x_i) = \arg\max_k \hat{p}^*, \tag{2-18}$$

Si el interés es conseguir las probabilidades de clase $\hat{p}_k$ de la observación x, entonces se deben promediar las probabilidades generadas en cada árbol $T^{*b}(x)$ como en la expresión (2-8) de la subsección anterior, en lugar de la proporción calculada en (2-18). Un ejemplo de clasificación binaria sobre por qué no funciona tomar la proporción (2-18) como la probabilidad de clase se da en Hastie *et al.* (2009): si la clase 1 tiene una probabilidad conocida de 0.75, y todos los árboles en *bagging* lo clasifican correctamente, entonces $\hat{p}_1^*(x) = 1$, que no es una muy buena estimación de 0.75.

Bagging no solo funciona en árboles, también puede ser aplicado en otros métodos de predicción. Sin embargo, su habilidad estabilizadora de varianza es ventajosa cuando se tienen métodos inestables, como se ejemplifica en Breiman (1996). Al aplicarse con métodos más estables, los resultados no mejorarán sustancialmente, ya que no hay varianza significativa que disminuir.

Algunas iteraciones de *bagging* con árboles de clasificación se encuentran en la figura 2-7.

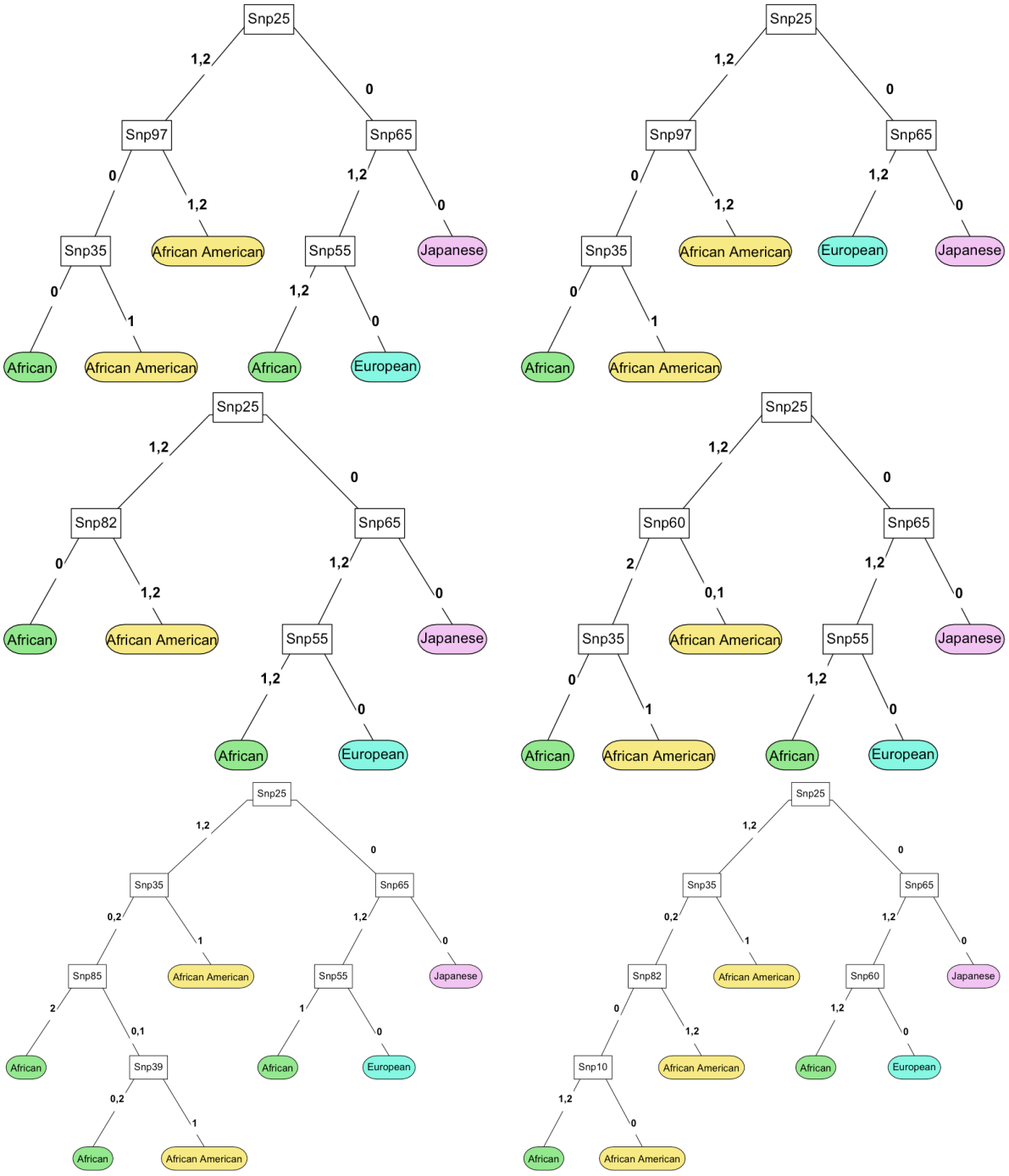


Figura 2-7: Árboles de clasificación para la base `Ancestry`, trabajada en el capítulo siguiente. Las variables predictoras son categóricas de hasta tres niveles. El primer árbol (arriba a la izquierda) se construyó con todos los datos ($N = 197$); el resto, con diferentes muestras *bootstrap*. Todos tienen un mínimo de 10 observaciones en los nodos finales y fue podado con $\alpha = 0.01$ (ver subsección 2.1.1).

Es posible ir un paso más allá para aprovechar las propiedades del promedio de árboles frondosos: disminuir la correlación entre ellos. El muestreo *bootstrap* ya logra parte de esta tarea, pero es aún más efectivo si, además, a cada nodo solo se le permite revisar cierto subconjunto de variables predictoras para elegir aquella con la que se hará una partición. Recordando la expresión (2-16):

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2,$$

lo que queda por hacer una vez que el segundo sumando desaparece, es reducir la correlación ρ entre cada par de árboles. Puede encontrarse un análisis de este tema en la subsección 15.4 de Hastie *et al.* (2009).

Con esta modificación se tiene ***Random Forest***: por medio del remuestreo *bootstrap* se eligen las observaciones x en cada árbol, y en cada nodo se usa cierto número predeterminado de variables $X_F \subset X$, cada vez distintas y elegidas al azar, con el que se va a realizar la partición.

Random Forest es conocido por mejorar la capacidad de predicción de los árboles de decisión, aunque deja de lado su interpretabilidad y visualización. Estos parámetros son usualmente:

- B: el número de árboles. Usualmente se experimenta con él hasta que haya evidencia de la convergencia del error, como explican James *et al.* (2017), subsección 8.2.2, por lo que depende del conjunto de observaciones. B debe ser por lo menos 100 para asegurar el error de predicción converja, pero 1000, o incluso más, es preferible.

- `mtry`: el número de variables a probar en los nodos de los árboles; estos son elegidos en cada nodo de forma aleatoria. Efron y Hastie (2016), sección 17.1, sugieren que si `mtry` es muy pequeño, puede ser que las variables con un efecto importante en la respuesta sean elegidas muy pocas veces, mientras que un número grande de variables puede pasar que un reducido grupo de ellas sea seleccionado la mayor parte del tiempo, sin aportar mucha información. El número recomendado de variables candidatas en cada nodo es $\frac{P}{3}$ en clasificación y $\sqrt{P}$ en regresión, de acuerdo con James *et al.* (2017).

- La profundidad de los árboles puede regularse mediante la selección de uno de los siguientes parámetros:

- `nodesize`: El número mínimo de observaciones en cada nodo, donde mientras más grande sea un número, menos profundo será el árbol. Las particiones continuarán mientras se respete el mínimo establecido, que es de uno en clasificación y cinco en regresión.
- M: El número de nodos terminales. No suele ser controlada en *Random Forest*, ya que el objetivo es reducir el sesgo particionando tanto como el número mínimo de observaciones lo permita; sin embargo, se vuelve una elección importante cuando se trata del otro algoritmo a revisar, *boosting* (se hablará sobre esto en la subsección 2.3.3).
- J: Número de veces a particionar. La implementación de `h2o`, que se usa en el presente trabajo, determina la profundidad como el número de veces que se dividirán a los nodos existentes. Así, un valor de 1 es una sola partición de la base de datos (dos nodos finales), un valor de 2 hace una partición en cada nodo de la primera división, (cuatro nodos finales), 2^J es el número de nodos finales.

Lo descrito queda condensado en el algoritmo siguiente, obtenido de Hastie *et al.*, subsección 15.2 (2009):

Algoritmo 2.2.1 *Random Forest*

1. *Para $b = 1$ y hasta B:*

   a) *Hacer un muestreo bootstrap d^* de tamaño N del conjunto de datos d.*

   b) *Crear un árbol de decisión T^{*b} con la muestra generada, y para cada nodo:*

      i. *seleccionar `mtry` variables al azar de las p posibles,*

      ii. *seleccionar la mejor variable entre las `mtry` con su respectivo punto de corte,*

      iii. *dividir en dos regiones hasta que se alcance la profundidad deseada.*

2. *Tener la colección de árboles $\{T^{*b}, b = 1, \ldots, B\}$.*

3. *Para obtener la predicción $f_B(x_i)$ de la observación x_i, calcular el promedio de los árboles sobre ese punto, o si es el caso, la clase más votada, como en las expresiones (2-15) y (2-18), respectivamente.*

Aunque en esta sección por comodidad se usó la notación $T^{*b}(x_i)$ para referirse al árbol de la muestra *bootstrap* b, dicho árbol b está caracterizado por Θ_b, es decir, por sus regiones finales R y constantes $\hat{\gamma}$, como fue explicado en la sección anterior, por lo que una notación más exacta es $T(x_i; \Theta_b)$. Por la ley fuerte de los grandes números, Breiman (2001a) demostró que *Random Forest* no sobreajusta mientras más árboles son agregados a la sucesión. Entonces, si $f_B(X)$ es el ajuste de *Random Forest* resultante de la sucesión de árboles $\{T(X; \Theta_b)\}_1^B$ (donde Θ dependerá del conjunto de entrenamiento en cada $T(X; \Theta)$), ya sea en en el caso de clasificación o regresión, entonces

$$\lim_{B \to \infty} f_B(X) = E_\Theta[T(X; \Theta)]. \tag{2-19}$$

Random Forest toma su decisión sobre B árboles diferentes, por lo que más iteraciones harán que se acerque más al error "verdadero" que $E_\Theta[T(X; \Theta)]$ puede alcanzar, como indica Hastie *et al.* (2009) en la subsección 8.7. Sin embargo, en la subsección 15.3.4 de aquel trabajo se advierte que añadir más árboles no sobreajustará siempre y cuando el límite $E_\Theta[T(X; \Theta)]$ no sobreajuste por sí mismo. Es decir, el conjunto de árboles muy profundos puede resultar en un modelo muy complejo que llevará al sobrejuaste. En este trabajo no se encontró aquel problema.

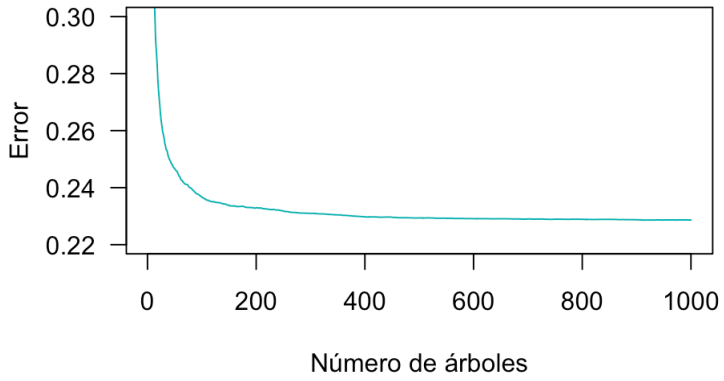


Figura 2-8: Evolución del error de prueba mientras $B \to \infty$ en la base de datos `California Housing`, que será trabajada en el capítulo siguiente.

Un aspecto más es que Breiman (2001a) define a *Random Forest* como la colección de árboles caracterizados por vectores aleatorios $\{\Theta_b\}_{b=1}^{B}$, idénticamente distribuidos, con la selección aleatoria de variables para probar en el nodo siendo una posibilidad, pero no una característica necesaria. No obstante, aquí se trata como obligatoria para que la colección de árboles sea llamada *Random Forest*, y si no es así, simplemente es *bagging*, al igual que es explicado en la literatura posterior al trabajo de Leo Breiman, como en Hastie *et al.* (2009) y Efron y Hastie (2016), por ejemplo.

2.2.2. Otras características

Bagging y *Random Forest* comparten las mismas características explotables, como tener disponible una muestra *Out-of-bag* ("fuera de bolsa") para estimar el error de predicción, así como una manera de conocer las variables más importantes y métodos para describir la dependencia de la variable respuesta respecto a variables predictores.

Estimación del error por muestra *Out-of-bag (OOB)*

En un ajuste de *bagging* o *Random Forest* de B árboles, una observación dada no aparece en aproximadamente $e^{-1}B \approx 0.37B$ árboles: la probabilidad de que una observación no sea elegida de entre las N observaciones es de

$$1 - \frac{1}{N},$$

entonces, la probabilidad de que no aparezca ni una vez en N ensayos (muestreo con reemplazo, i.e. *bootstrap*) es

$$\left(1 - \frac{1}{N}\right)^N.$$

La aproximación $e^{-1}B$ se consigue con una N grande, ya que

$$\lim_{N\to\infty} \left(1 - \frac{1}{N}\right)^N = \frac{1}{e}. \tag{2-20}$$

En el apéndice B se muestra experimentalmente que la aproximación se cumple. Implementaciones como `h2o`, en las cuales no se hace muestreo con *bootstrap*, este resultado es usado

para seleccionar una fracción de la base de datos al construir cada árbol.

Para ilustrar este concepto, se considera un conjunto de datos

$$\boldsymbol{d} = \{(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5)\}.$$

Al realizar muestreo con reemplazo tres veces sobre él, se obtuvieron los siguientes conjuntos *bootstrap*. Se muestran los conjuntos *OOB* asociados:

$$\boldsymbol{d}^{*1} = \{(x_2, y_2), (x_3, y_3), (x_1, y_1), (x_3, y_3), (x_1, y_1)\}, \quad d^1_{OOB} = \{(x_4, y_4), (x_5, y_5)\};$$
$$\boldsymbol{d}^{*2} = \{(x_4, y_4), (x_4, y_4), (x_1, y_1), (x_2, y_2), (x_4, y_4)\}, \quad d^2_{OOB} = \{(x_3, y_3), (x_5, y_5)\};$$
$$\boldsymbol{d}^{*3} = \{(x_3, y_3), (x_1, y_1), (x_2, y_2), (x_1, y_1), (x_5, y_5)\}, \quad d^3_{OOB} = \{(x_4, y_4)\}.$$

Los conjuntos *OOB* pueden convertirse en un conjunto para calcular el error del grupo de árboles, recordando que para estimar el error de prueba se necesita que las observaciones con las que se hacen las predicciones no sean las mismas con las que se entrenó al ajuste. Para ello, cada observación es predicha con los árboles en los que **no** fue seleccionada. De acuerdo con la subsección 17.1 de Efron y Hastie (2016), el error *OOB* es:

$$f^{*(i)}(x_i) = \frac{1}{B_i} \sum_{b:\, i \notin d^{*b}} T^{*b}(x_i), \tag{2-21}$$

donde B_i es el número de árboles donde no aparece la observación (x_i, y_i). El caso de clasificación es análogo, usando las expresiones (2-17) y (2-18).

A continuación se obtiene el error de la muestra *OOB*, usando la función de pérdida L adecuada, como el cuadrado de los residuos o la devianza:

$$\widehat{Err}^{(OOB)} = \frac{1}{N} \sum_{i=1}^{N} L\left(y_i, f^{*(i)}(x_i)\right). \tag{2-22}$$

Con ello se obtiene un estimador del error de prueba, por lo que no hay necesidad de separar la base de datos para calcularlo como con otros métodos. Esto cobra especial relevancia cuando

se tienen bases de datos donde una división de ellas resultaría en conjuntos de entrenamiento y prueba demasiado pequeños, o cuando el número total de observaciones es de tal tamaño que hacer validación cruzada tomaría gran trabajo computacional.

No obstante, dado que cada árbol se basa en alrededor de dos tercios de las observaciones, el estimador *OOB* cuenta con solo un tercio de la información que tiene el total de los árboles construidos. Por consiguiente, el error obtenido con la muestra *OOB* sobreestimará el verdadero error, ya que con *Random Forest* éste decrece mientras se añaden árboles a la colección, y en consecuencia el error *OOB* tendrá un retraso. Es por ello que se necesita crear árboles más allá de la aparente convergencia del error del conjunto de prueba si se requiere un estimador del error de prueba fiel a lo que se tendría con dicho conjunto, según Breiman (2001a).

Efron y Hastie (2016), sección 17.1, menciona que esta estimación, calculada con tres veces el número de árboles necesarios para que se estabilice *Random Forest*/*bagging*, es equivalente a *leave-one-out cross-validation (LOOCV)*. Esta aseveración tiene sentido: en *LOOCV* se entrena a un algoritmo con $N-1$ datos y se calcula el error con la única observación que no fue usada para entrenar, repitiendo el proceso para cada una de las observaciones (ver subsección 1.3 de este trabajo). Con la estimación *OOB* se calcula el error para la observación x en aquellos árboles donde no fue usada, como en la expresión (2-21). Mientras el número de árboles $B \to \infty$, el resto de las observaciones han sido usadas en el acumulado de árboles que no incluyeron a x. Ambos métodos, entonces, estiman el error para todas y cada una de las observaciones, cuando el ajuste se entrena con el subconjunto que no contiene a x.

Importancia de variables

La interpretabilidad de un mecanismo de predicción es de gran importancia para conocer las interacciones que las variables tienen entre sí, así como el valor que cada una ofrece al poder predictivo del modelo. La muestra *OOB* puede ser usada para estimar la importancia que cada variable tiene.

Un método es que en cada árbol se haga la predicción usando las observaciones *OOB*, para después ser usadas de una manera distinta: en cada variable $\boldsymbol{x}_p$, una a la vez, se permutarán

aleatoriamente las observaciones dentro de esa columna, se ajusta un *Random Forest* a los datos modificados, se tiene una medida de error de la muestra *OOB*, y es comparado con el error *OOB* de la base sin modificar. El cambio en esa medida de error es un indicador de la importancia de la variable. Un cambio pequeño indica que los cambios de esa variable fueron poco relevantes en el resultado final, mientras que cambios grandes denotan una influencia significativa de la variable, provocado por alteraciones en ella. Esta primera propuesta es de Breiman (2001a).

Para ilustrar lo anterior, se ajustará *bagging* de un árbol, es decir, un solo árbol de decisión con muestreo *bootstrap*, sin selección aleatoria de variables en cada nodo. La base de datos es una muestra de ocho observaciones de `California Housing`, que se trabajará a fondo en el capítulo próximo. Cada observación muestra datos agrupados de conjuntos de casas, como el promedio de habitantes o de alcobas por vivienda. La variable respuesta es `MedianHouseValue`, la mediana del valor de las casas.

`MedianHouseValue`	...	`MedInc`	`AveBedrms`	...
3.585		8.301	0.972	
2.697		4.037	1.104	
3.413		5.643	1.073	
3.521	...	7.257	1.073	...
4.526		8.325	1.024	
2.992		3.659	0.951	
2.414		3.120	1.062	
3.422		3.846	1.081	

Tabla 2-1: Muestra de la base de datos `California Housing`, con $p = 8$ variables predictoras y `MedianHouseValue` como variable respuesta. Con un ajuste de *bagging* de un solo árbol, el error *OOB* es de 0.219 (media del cuadrado de los residuos). Los colores en la variable `MedInc` son auxiliares para los siguientes pasos.

Los colores de la tabla anterior ayudarán a entender cómo funciona el mecanismo. Para conocer la importancia de la variable `MedInc`, la mediana del ingreso, se permutarán sus valores, mientras todas las demás variables permanecen estáticas en la tabla 2-2.

MedianHouseValue	...	MedInc	AveBedrms	...
3.585		7.257	0.972	
2.697		8.301	1.104	
3.413		4.037	1.073	
3.521	...	8.325	1.073	...
4.526		5.643	1.024	
2.992		3.659	0.951	
2.414		3.846	1.062	
3.422		3.120	1.081	

Tabla 2-2: Muestra de la base de datos `California Housing`, con los valores de `MedInc` permutados. De esta forma, el error (media del cuadrado de los residuos) es de 0.394, un incremento relativo de 80.1 % respecto al ajuste de la base sin permutar.

Al repetir el proceso de permutar los valores en cada una de las variables, se registra el respectivo error *OOB* y se calcula el aumento relativo respecto a la base sin modificar (figura 2-9).

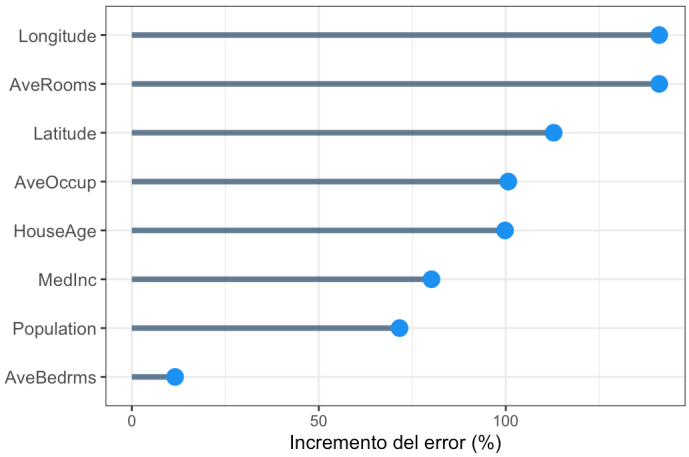


Figura 2-9: Ejemplo de importancia de variables según el aumento del error (media del cuadrado de los residuos) para la base `California Housing`. El ajuste entrenado es uno de *bagging* de un solo árbol.

El ejemplo recién visto tiene como propósito explicar una manera de encontrar la importancia de variables. Al tener una muestra tan pequeña, no representativa de la base de datos, y al haber ajustado un solo árbol, los resultados no reflejan fielmente la influencia que tienen las variables. En el capítulo siguiente se analizarán adecuadamente ejemplos de la importancia de variables, aplicados a `California Housing` y a otras bases de datos, con ajustes de más

iteraciones. En esos casos, se promedia el incremento del error de los B árboles.

Una segunda manera de conocer la importancia de variables es midiendo el aumento de precisión que aporta una variable explicativa cuando es usada en una partición de los árboles. El origen del siguiente método se da en Breiman *et al.* (1984), ideado para árboles de decisión individuales, y fue expandido especialmente para *boosting* (a ser revisado más adelante) en Friedman (2001), aunque también puede ser usado en *Random Forest*. Este es el método que se empleará en el siguiente capítulo.

En cada nodo dentro se busca dividir al conjunto de datos para conseguir la mejor partición. Un criterio para medir la mejora empírica generada por la partición de un nodo t, con base en el cuadrado de los residuos, es:

$$i^2(t_{izq}, t_{der}) = \frac{t_{izq} t_{der}}{t_{izq} + t_{der}} (\hat{\gamma}_{izq} - \hat{\gamma}_{der})^2, \tag{2-23}$$

donde t_{izq} y t_{der} son los nodos resultantes izquierdo y derecho, con cardinalidades respectivas de N_{izq} y N_{der}, así como las constantes de predicción $\hat{\gamma}_{izq}$ y $\hat{\gamma}_{der}$. A mayor i^2, mejor es la partición: mientras más grande es la diferencia entre $\hat{\gamma}_{izq}$ y $\hat{\gamma}_{der}$, más diferentes entre sí son los dos nodos creados.

Entonces, la importancia de la variable X_j en un árbol T con M nodos finales está dada por

$$\mathcal{I}_j^2(T) = \sum_{t=1}^{M-1} i_t^2 \, \mathbb{1}_{\{v(t)=j\}}, \tag{2-24}$$

donde i_t^2 representa la mejora en el nodo t, la indicadora $\mathbb{1}_{\{v(t)=j\}}$ significa que se sumarán las i_t^2 correspondientes a cuando la partición en el nodo t se da en la variable X_j (uno de los $M-1$ nodos internos).

Para tener la importancia de X_j en los B árboles que componen el *Random Forest* se calcula:

$$\mathcal{I}_j^2 = \frac{1}{B} \sum_{b=1}^{B} \mathcal{I}_j^2(T^b). \tag{2-25}$$

Finalmente, a la variable con mayor importancia se le asigna un valor de 100 y las demás son

ajustadas acorde a esa escala.

Como se verá al final del presente capítulo, *boosting* para clasificación genera K conjuntos de árboles, uno para cada clase de la variable respuesta. En este caso:

$$\mathcal{I}_{jk}^2 = \frac{1}{B}\sum_{b=1}^{B}\mathcal{I}_j^2(T^{kb}), \tag{2-26}$$

donde T_{kb} es el árbol inducido para la clase k en la iteración b. Se interpreta como la relevancia que tiene X_j para distinguir a la clase k de las otras $K-1$. Este resultado puede ser muy útil, puesto que hay distribuciones donde diferentes variables explicativas son relevantes para diferentes clases, información que no se puede conseguir de una medida de importancia general. Ésta última se consigue promediando sobre la importancia de las clases:

$$\mathcal{I}_j^2 = \frac{1}{K}\sum_{k=1}^{K}\mathcal{I}_{jk}^2. \tag{2-27}$$

Dependencia parcial

Una vez identificadas las variables más influyentes en el modelo, el siguiente paso es tratar de reconocer cambios en la variable respuesta en función de las predictoras, tratándose de un esfuerzo más de otorgar intrepretabilidad a los ajustes de conjuntos de árboles. Se ha propuesto un método gráfico para esta tarea, en el que se presenta el efecto que una o dos variables tienen en la respuesta (siendo gráficas de dos o tres dimensiones, respectivamente). Las funciones de más de tres variables predictoras son imposibles de visualizar, por lo que la aplicación de este método se ha limitado a los casos mencionados.

La dependencia de un ajuste de *Random Forest* o de *boosting* $\hat{f}(X)$ de un subconjunto de variables X_S se puede estimar si se considera primero el efecto promedio que tiene el resto de las variables (el complemento), X_C. Nótese que $X = X_S \cup X_C$ y, por tanto, $f(X) = f(X_S, X_C)$. El principal interés es la visualización de estos resultados, por lo que el número de variables X_S suele estar limitado a uno o dos. Por ello, se puede tener una colección de gráficas de dependencia parcial de una sola variable explicativa.

Una manera de definir la dependencia, de acuerdo con la subsección 10.13.2 de Hastie *et al.*

(2009) es:

$$f_S(X_S) = E_{X_C} f(X_S, X_C). \tag{2-28}$$

En *Random Forest* puede estimarse la expresión anterior como:

$$\bar{f}_S(X_S) = \frac{1}{N} \sum_{i=1}^{N} f(X_S, x_{iC}), \tag{2-29}$$

con $\{x_{1C}, x_{2C}, \ldots, x_{NC}\}$ como los valores de X_C en el conjunto de entrenamiento. Estos valores deben evaluarse para cada conjunto de valores de X_S que deseen graficarse, por lo que puede volverse computacionalmente extenuante. Afortunadamente, existe una manera alternativa de estimar la dependencia parcial cuando se usan árboles de decisión.

En la construcción de los árboles, cada nodo tiene almacenada la proporción de observaciones que pasó a través de él. En lugar de hacer predicciones para todos las observaciones, se utilizará esa información. En este método, solo se requieren valores de interés de X_S, que pasarán por el árbol creado con el conjunto de datos. Friedman (2001) explica:

1. Las observaciones pasarán por el nodo inicial y, en este punto, cada una tendrá un peso asociado de 1.

2. En nodos intermedios, existen dos posibilidades:

    i. Si la variable que define la partición del nodo pertenece al subconjunto X_S, entonces la observación sigue su camino, a la subregión izquierda o derecha, dependiendo de la regla de partición definida en el nodo. El peso asociado a la observación permanece sin cambios.

    ii. En caso contrario, la variable seguirá *ambos* caminos a las dos subregiones distintas, izquierda y derecha, y en cada una se multiplicará su peso por la proporción de observaciones que pasaron a aquellos nodos. Por ejemplo, si había 10 observaciones en el nodo principal y, después de la partición, el nodo izquierdo recibe cuatro observaciones y el derecho seis, entonces se multiplicará el peso por 0.4 en el izquierda y 0.6 en el derecho.

3. Cada observación seguirá su camino por los nodos según las reglas descritas (pudiendo ser que cada observación termine en más de un nodo final por la posibilidad ii). En el nodo final, se multiplica el peso resultante por el valor de la predicción correspondiente en esa región.

4. $\bar{f}_S(X_S)$ para el árbol será la suma de aquellos productos, es decir, un promedio ponderado; para el conjunto de árboles, es el promedio de los resultados obtenidos en los árboles individuales.

Los dos métodos son maneras válidas de estimar $\bar{f}_S$, pero la segunda es considerablemente más rápida al no tener que realizar las predicciones para cada observación del conjunto de datos.

Hastie *et al.* (2009) resaltan que la dependencia parcial $\bar{f}_S(X_S)$ representa el efecto de las variables X_S en $f(X)$ después de considerar el efecto promedio de las otras variables X_C. **No** son el efecto de X_S ignorando X_C.

2.3. *Boosting* por gradiente

Durante la misma época en la que *Random Forest* fue propuesto y tomado en serio en la comunidad creciente de *machine learning*, *boosting*, otro método basado en árboles de decisión, también empezó a ser reconocido y estudiado. Esta sección se basa, paso a paso, en la investigación de la comunidad estadística de la época, en particular, el trabajo de Jerome Friedman en *Greedy function approximation: A gradient boosting machine* (1999), *Stochastic Gradient Boosting* (2001) y en *Additive Logistic Regression: A statistical view of boosting* (2000), este último con Trevor Hastie y Robert Tibshirani.

La similitud con *Random Forest* es que se basa en la combinación de un conjunto de árboles de decisión, pero la forma de conseguir la predicción es a través de la optimización de una función de pérdida L, haciendo que los nuevos árboles enmienden los errores de los anteriores, y paso a paso se alcance una solución. Se revisará a continuación el uso de modelos aditivos y de métodos numéricos para este propósito, con los que se construirá el algoritmo de *boosting*. Se presentarán las tres versiones necesarias del algoritmo (dos de regresión y una de clasificación)

para las aplicaciones del último capítulo, y al finalizar se hablará de dos parámetros interesantes de este algoritmo: la profundidad de los árboles y el submuestreo.

2.3.1. Modelos aditivos

Los modelos aditivos son expansiones aditivas de funciones. Estas funciones deben seguir una misma estructura, por ejemplo, pueden ser árboles de decisión. En general:

$$f(X) = \sum_{b=1}^{B} \beta_b \, h(X; \Theta_b), \tag{2-30}$$

donde β_b, $b = 1, \ldots, B$, son coeficientes y $h(X; \Theta_b)$ son funciones de varias variables X y caracterizadas por Θ_b, un conjunto de parámetros distinto para cada función.

Se busca el conjunto de parámetros $\{\beta_b, \Theta_b\}_{b=1}^{B}$ que minimicen la función de pérdida para el modelo aditivo completo y todas las observaciones del conjunto de datos:

$$\min_{\{\beta_b, \Theta_b\}_{b=1}^{B}} \sum_{i=i}^{N} L\left(y_i, \sum_{b=1}^{B} \beta_b \, h(x_i; \Theta_b)\right). \tag{2-31}$$

La solución se complica dependiendo de la función $h(X; \Theta_b)$ y la función de pérdida. Optimizar toda la expansión implicaría modificar los parámetros y coeficientes de las funciones anteriores mientras los sumandos $\beta_b \, h(x; \Theta_b)$ se acumulan, por lo que se prefiere encontrar solución etapa a etapa, optimizando una función a la vez, sin modificar las anteriores:

$$\min_{\beta, \Theta} \sum_{i=1}^{N} L(y_i, \beta h(x_i; \Theta)). \tag{2-32}$$

Lo anterior queda expresado en el siguiente algoritmo del **modelado aditivo hacia adelante por etapas** (en inglés, como se ve en Hastie *et al.* (2009), subsección 10.3, *Forward stagewise additive modeling*), a partir de ahora referido como MAHAE:

Algoritmo 2.3.1 *Modelado aditivo hacia adelante por etapas (MAHAE)*

1. *Inicializar con* $f_0(X) = 0$
2. *Para* $b = 1$ *y hasta* B*:*

i. Calcular:

$$(\beta_b, \Theta_b) = \arg\min_{\beta,\Theta} \sum_{i=1}^{N} L(y_i, f_{b-i}(x_i) + \beta h(x_i; \Theta)).$$

ii. Actualizar $f_b(X) = f_{b-1}(X) + \beta_b h(X; \Theta_b)$.

De esta manera, en cada iteración solo hay que preocuparse por encontrar los parámetros $\{\beta_b, \Theta_b\}$ óptimos, sin modificar los de las iteraciones anteriores, haciendo que la adición de $\beta_b\, h(X; \Theta_b)$ ("salto" o *boost*) al modelo f_{b-1} se acerque lo más posible a la variable respuesta Y, por medio de la minimización de la función de pérdida.

Es de interés particular cuando la función $h(X; \Theta_b)$ es un árbol de decisión y Θ_b lo caracteriza a través de las reglas de partición en los nodos y las predicciones de las regiones finales (i.e. $\Theta_b = \{R_{mb}, \gamma_{mb}\}_{m=1}^{M_b}$: los parámetros del árbol de la iteración b con M_b regiones finales). Cuando se usan los árboles en *boosting*, el modelo es el siguiente:

$$f_B(X) = \sum_{b=1}^{B} T(X; \Theta_b). \tag{2-33}$$

Los primeros algoritmos de *boosting*, como *Adaboost*, que puede ser consultado en la subsección 17.4 de Efron y Hastie (2016), consideraban coeficientes β; no obstante, *boosting* por gradiente no los ocupa. Por lo tanto, el paso 2 del algoritmo MAHAE se reduce a encontrar las regiones y las constantes del próximo árbol basándose en la suma de los anteriores, como se presenta en Hastie *et al.* (2009), subsección 10.9:

$$\hat{\Theta}_b = \arg\min_{\Theta_b} = \sum_{i=1}^{N} L(y_i,\ f_{b-1}(x_i) + T(x_i; \Theta_b)). \tag{2-34}$$

El objetivo, dado el modelo f_{b-1}, es encontrar $\Theta_b = \{R_{mb}, \gamma_{mb}\}_{m=1}^{M_b}$: cada uno de los M_b nodos finales y sus respectivas constantes de predicción γ_{mb} correspondientes al árbol b.

Recordando que en la construcción de los árboles primero se definen las regiones y después las predicciones dentro de ellas (ver subsección 2.1.1 de este trabajo), dada una región R_{mb} se

busca la constante $\hat{\gamma}_{mb}$:

$$\hat{\gamma}_{mb} = \arg\min_{\gamma_{mb}} \sum_{x_i \in R_{mb}} L(y_i, f_{b-1}(x_i) + \gamma_{mb}). \tag{2-35}$$

Encontrar $\hat{\Theta}_b$, en particular R_{mb}, es una tarea complicada generalmente, pero si se usa el cuadrado de los residuos $(Y - f(X))^2$ como función de pérdida, la solución a (2-34) es:

$$L(y_i, f_{b-1}(x_i) + T(x_i; \Theta_b)) = (y_i - f_{b-1}(x_i) - T(x_i; \Theta_b))^2 = (r_{ib} - T(x_i; \Theta_b))^2, \tag{2-36}$$

con $r_{ib} = y_i - f_{b-1}(x_i)$ como el residual de la i-ésima observación en la iteración b. Aunque sigue siendo difícil de resolver en el caso particular de $T(x_i; \Theta_b)$, se puede aproximar con la construcción de árboles hacia los mismos residuales. Es decir, la solución es el árbol de regresión que mejor predice a los residuales, con $\hat{\gamma}_{mb}$ como el promedio de ellos en la región m. La suma sucesiva de las predicciones de residuales se acercará cada vez más al valor observado.

El uso de otras funciones de pérdida, como la función de Huber (una alternativa más robusta para el caso de regresión), o la devianza para el caso de clasificación, no tienen soluciones sencillas como la anterior, por lo que se requiere de otro método de aproximación.

2.3.2. Optimización numérica: *boosting* por gradiente

Se pueden conseguir algoritmos con cualquier función de pérdida L para resolver (2-34). La función de pérdida L a optimizar en los modelos aditivos está en función de $f(X)$ y la respuesta Y que busca predecir:

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)). \tag{2-37}$$

La minimización de $L(f)$ con respecto a f es:

$$\hat{\boldsymbol{f}} = \arg\min_{f} L(\boldsymbol{f}). \tag{2-38}$$

Se puede suponer a los componentes de $\boldsymbol{f}$ como parámetros, siendo números reales $f(x_i)$ evaluados en cada punto de X: $\boldsymbol{f} = \{f(x_1), \ldots, f(x_N)\}$.

La optimización suele ser a través de la suma de vectores:

$$\boldsymbol{f}_B = \sum_{b=0}^{B} \boldsymbol{h}_b, \quad \boldsymbol{h}_b \in \mathbb{R}^N, \tag{2-39}$$

en la que se empieza con una estimación inicial $\boldsymbol{f}_0 = \boldsymbol{h}_0$ y los sucesivos $\{\boldsymbol{f}_b\}_{b=1}^{B}$ son una sucesión de pasos, cada uno de ellos basando en los anteriores. Lo anterior es acorde a lo expuesto en Hastie *et al.* (2009), subsección 10.10. Se encontrará cada $\boldsymbol{h}_b$ con el gradiente.

Steepest descent

El *steepest descent* ("descenso más inclinado"), elige los pasos mencionados como $\boldsymbol{h}_b = -\rho_b \boldsymbol{g}_b$, donde ρ_b es un escalar que rige el *tamaño* del paso y $\boldsymbol{g}_b$ es un vector que indica la *dirección*, ya que es el gradiente de $L(\boldsymbol{f})$ en $\boldsymbol{f} = \boldsymbol{f}_{b-1}$ para cada observación i:

$$\boldsymbol{g}_b = \{g_{ib}\} = \left\{ \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x_i)=f_{b-1}(x_i)} \right\}. \tag{2-40}$$

ρ_b busca el tamaño del paso en el camino indicado por el gradiente,

$$\rho_b = \arg\min_{\rho} L(\boldsymbol{f}_{b-1} - \rho \boldsymbol{g}_b). \tag{2-41}$$

Entonces la siguiente $\boldsymbol{f}_m$ está en el paso que se dio, donde la función de pérdida decrecía más rápidamente con la información (las observaciones x_i) a la mano, hacia el mínimo local. Es conocida como **búsqueda en línea** o *line search*.

$$\boldsymbol{f}_b = \boldsymbol{f}_{b-1} - \rho_b \boldsymbol{g}_b. \tag{2-42}$$

Estas últimas expresiones fueron tomadas de la subsección 10.10.1 de Hastie *et al.* (2009).

Boosting por gradiente

Las predicciones obtenidas por MAHAE son análogas a *steepest descent*, pues en cada paso se encuentra la mejor solución f_b con base en el modelo actual f_{b-1} (expresión 2-34). Sin

embargo, la dirección del paso tiene la restricción de ser en forma de árboles de decisión, todos del mismo tamaño, a diferencia de la dirección del paso en *steepest descent* (expresión 2-40) donde no se tienen restricciones de ese tipo. La búsqueda en línea en MAHAE (expresión 2-35) es análoga a aquella de *steepest descent*, con la particularidad de que se hace una búsqueda en línea para cada nodo final en los árboles de decisión, llegando a $\hat{\gamma}_{mb}$.

Por otra parte, el gradiente en *steepest descent* (expresión 2-40) solo está definido en las N observaciones de $\boldsymbol{d}$, por lo que no puede ser generalizado a datos desconocidos. La solución de Friedman (2001) es hacer que los árboles en MAHAE generen predicciones $\boldsymbol{t}_b = (T(x_1;\Theta_b),\ldots,T(x_N;\Theta_b))$ paralelas al gradiente negativo. De esta manera, el gradiente $\boldsymbol{g}_b$ se correlaciona con $\boldsymbol{t}_b$ bajo los datos observados. Entonces, se usa el error cuadrático para alcanzar esta aproximación:

$$\tilde{\Theta}_b = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{ib} - T(x_i,\Theta))^2. \tag{2-43}$$

Esto significa que se ajusta un árbol T al gradiente negativo (2-40). Al construir el árbol T con (2-43), las constantes de predicción se consiguen con (2-35). Este procedimiento queda definido en el siguiente algoritmo, obtenido de Hastie *et al.* (2009), subsección 10.10.3, para el caso de regresión:

Algoritmo 2.3.2 *Boosting por gradiente (caso de regresión)*

1. *Inicializar con* $f_0(X) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i,\gamma)$.

2. *Para* $b = 1$ *y hasta* B*:*

   a) *Para* $i = 1,2,\ldots,N$ *calcular*

   $$r_{ib} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{b-1}}.$$

   b) *Ajustar un árbol de regresión hacia* r_{ib} *con regiones terminales* R_{mb}, $m = 1,2,\ldots,M_b$.

c) Para $m = 1, 2, \ldots, M_b$ *calcular*

$$\hat{\gamma}_{mb} = \arg\min_{\gamma} \sum_{x_i \in R_{mb}} L(y_i, f_{b-1}(x_i) + \gamma).$$

d) Actualizar $f_b(X) = f_{b-1}(X) + \sum_{m=1}^{M_b} \hat{\gamma}_{mb} \mathbb{1}_{\{X \in R_{mb}\}}$.

3. El resultado es $\hat{f}(X) = f_B(X)$.

El algoritmo considera al gradiente negativo r_{ib} como un *pseudorresidual*.

La mayoría de las veces se hace uso de la **regularización** para evitar el sobreajuste, pues la suma sucesiva de los pseudorresiduales puede ocasionar una predicción demasiado fiel al conjunto de prueba, llevando al sobreajuste. Por ello, se introduce una constante ν de *encogimiento*, aquí referida como *learning rate* ("ritmo de aprendizaje"). Su propósito es escalar la predicción de cada árbol antes de ser sumado a la predicción, por lo que $0 < \nu < 1$. De este modo, el paso 2 d) del algoritmo se regulariza como:

$$f_b(x) = f_{b-1}(x) + \nu \sum_{m=1}^{M_b} \gamma_{mb} \mathbb{1}_{\{X \in R_{mb}\}}, \tag{2-44}$$

de acuerdo con la subsección 10.12.1 de Hastie *et al.* (2009).

Los valores pequeños de ν implican un crecimiento lento hacia la predicción final, por lo que se requerirán más árboles y, por ende, mayor tiempo de cómputo. Valores de ν cercanos a 1 aceleran la construcción, necesitando menos árboles, a costa de un riesgo incrementado de sobreajuste. La recomendación en Hastie *et al.* (2009), subsección 10.12, es elegir uno muy pequeño ($\nu < 0.1$). La evidencia en Friedman (2001) indica que es preferible elegir un número de iteraciones B tan alto como sea posible, y ajustar ν hasta que se alcance un ajuste óptimo. Si al alcanzar las B iteraciones el error de prueba no indicaba que había alcanza su punto mínimo, se puede aumentar el valor de ν o B, preferiblemente de B, pues su experiancia empírica muestra que un ν muy pequeño mejora drásticamente los resultados. Lo encontrado en el último capítulo de este trabajo es que las aseveraciones anteriores son ciertas, y se muestran resultados gráficos.

Cuadrado de los residuos como L

En este trabajo se usarán dos funciones de pérdida en el algoritmo anterior. La primera de ellas es el cuadrado de los residuos, con una leve modificación por conveniencia numérica: $\frac{1}{2}[y_i - f(x_i)]^2$.

El paso 1 del algoritmo se vuelve un problema de minimización por medio de la derivada de primer orden:

$$\frac{\partial}{\partial\gamma}\sum_{i=1}^{N}\frac{1}{2}(y_i-\gamma)^2 = \frac{1}{2}\sum_{i=1}^{N}\frac{\partial}{\partial\gamma}(y_i-\gamma)^2 = \frac{1}{2}\sum_{i=1}^{N}(-2)(y_i-\gamma) = \sum_{i=1}^{N}(\gamma-y_i) = N\gamma - \sum_{i=1}^{N}y_i. \quad (2\text{-}45)$$

Igualando a cero, se concluye que el primer árbol debe construirse para llegar al promedio de la respuesta:

$$\hat{\gamma} = \frac{\sum_{i=1}^{N} y_i}{N}.$$

El paso 2a) se trata de calcular la diferencia entre la respuesta observada y la respuesta predicha en la iteración anterior.

$$r_{ib} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{b-1}} = -\left[\frac{\partial}{\partial f(x_i)}\frac{1}{2}(y_i - f(x_i))^2\right]_{f=f_{b-1}} = y_i - f_{b-1}(x_i). \quad (2\text{-}46)$$

Los árboles del paso 2b) entonces intentarán predecir los residuales del paso anterior r_{ib}, en lugar de y_i. En c), la tarea es asignarle un valor constante a las M_b regiones formadas por las r_{ib}. De manera similar al paso 1, se minimiza con derivación, llegando a que la media de los residuos optimiza la función de pérdida:

$$\gamma_{mb} = \frac{\sum_{x_i \in R_{mb}}(y_i - f(x_i))}{|x_i \in R_{mb}|}. \quad (2\text{-}47)$$

En cada iteración $f_b(x)$ se suman los residuales a la estimación inicial, llegando al resultado final $\hat{f}(x) = f_B(x)$.

Función de Huber como L

La segunda función de pérdida a utilizar es aquella de Huber (ver subsección 1.2.1), donde para la iteración b se tiene:

$$L(y_i, f(x_i)) = \begin{cases} \frac{1}{2}(y_i - f(x_i))^2 & \text{si } |y_i - f(x_i)| \leq \delta \\ \delta(|y_i - f(x_i)| - \frac{\delta}{2}) & \text{si } |y_i - f(x_i)| > \delta, \end{cases} \tag{2-48}$$

donde $\delta = \text{cuantil}_\alpha\{|y_i - f(x_i)|\}$, $\alpha \in (0, 1)$. Las siguientes afirmaciones se basan Friedman (2001), que a su vez cita al trabajo de Peter J. Huber.

El algoritmo inicia con la mediana de las respuestas: $f_0(X) = mediana\{y_i\}_{i=1}^N$. El pseudorresidual de la observación i en la iteración m es:

$$r_{ib} = \begin{cases} y_i - f_{b-1}(x_i) & \text{si } |y_i - f_{b-1}(x_i)| \leq \delta_b \\ \delta_b[\text{signo}(y_i - f_{b-1}(x_i))] & \text{si } |y - f_{b-1}(x_i)| > \delta_b, \end{cases} \tag{2-49}$$

donde $\delta_b = cuantil_\alpha\{|y - f_{b-1}(x_i)|\}$ y la función signo devuelve -1 si el argumento es negativo, 1 si es positivo o 0 en otro caso.

Aquí se usará med_{mb} para referirse a la mediana de aquellos residuales resultantes de la iteración $b-1$, correspondientes a las región m de la iteración actual b, i.e. $d_{mb} = mediana_{x_i \in R_{mb}}\{y_i - f_{b-1}(x_i)\}$.

Ya que se construye al árbol que predice los pseudorresiduales, la constante γ de cada región m es:

$$\gamma_{mb} = med_{mb} + \frac{1}{|x_i \in R_{mb}|} \sum_{x_i \in R_{mb}} \text{signo}((y_i - f_{b-1}(x_i)) - d_{mb}) * min\{\delta_b, |(y_i - f_{b-1}(x_i)) - d_{mb}|\}. \tag{2-50}$$

Caso de clasificación

Si la tarea es una de clasificación, entonces se producen K modelos, uno para cada clase k:

$$f_k(X) = \sum_{b=1}^{B} T_{kb}(X), \quad k = 1, 2, \ldots, K, \tag{2-51}$$

cada uno relacionado con

$$p_k = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}, \; k = 1, 2, \ldots, K, \tag{2-52}$$

cumpliéndose que $0 \leq p_k \leq 1$ y $\sum_{k=1}^{K} p_k = 1$. La relación sucede a través de la siguiente transformación, definida por Friedman (2001):

$$f_k(x_i) = \log p_k(x_i) - \frac{1}{K} \sum_{l=1}^{K} \log p_l(x_i). \tag{2-53}$$

En clasificación binaria ($K = 2$), solo se necesita un modelo (2-51).

La función de pérdida propuesta en aquel trabajo es:

$$L(\{y_k, f_k(x_i)\}_{k=1}^{K}) = -\sum_{k=1}^{K} y_k \, \log p_k(x_i), \tag{2-54}$$

donde $y_k = \mathbb{1}_{\{clase=k\}} \in \{0, 1\}$ y $p_k(x_i) = P(y_k = 1 \mid x_i)$.

Cuyo pseudorresidual se da por medio del gradiente:

$$r_{ik} = -\left[\frac{\partial L(\{y_{il}, f_l(x_i)\}_{l=1}^{K})}{\partial f_k(x_i)}\right]_{\{f_l(x) = f_{l,m-1}(x)\}_{l=1}^{K}} = y_{ik} - p_{k,\,m-1}(x_i). \tag{2-55}$$

Siguiendo de lo anterior, en las tareas de clasificación se construyen K árboles en cada iteración, correspondientes a los residuos en unas escala de probabilidad de cada una de las K clases. Por ello, para la clase k en la iteración b, los nodos terminales de los árboles son $\{R_{jmb}\}_{j=1}^{J}$, con sus constantes:

$$\{\gamma_{jmb}\} = \underset{\{\gamma_{jk}\}}{\arg\min} \sum_{i=1}^{N} \sum_{k=1}^{K} \phi \left(y_{ik}, f_{k,\,b-1}(x_i) + \sum_{j=1}^{J} \gamma_{jk} \mathbb{1}_{\{x_i \in R_{mb}\}} \right), \tag{2-56}$$

donde $\phi(y_k, f_k(x)) = -y_k \log p_k$. El trabajo de Friedman aproxima la solución mediante Newton-Raphson, resultando en las regiones disjuntas:

$$\gamma_{jmb} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jmb}} r_{ikb}}{\sum_{x_i \in R_{jmb}} |r_{ikb}|(1-|r_{ikb}|)}. \tag{2-57}$$

Lo anterior se ve reflejado en el algoritmo para clasificación, como se presenta al final del capítulo 10 de Hastie *et al.* (2009):

Algoritmo 2.3.3 *Boosting por gradiente (caso de clasificación)*

1. *Inicializar con* $f_{k0}(X) = 0, k = 1, 2, \ldots, K$.

2. *Para* $b = 1$ *hasta* B*:*

   a) *Calcular*

   $$p_k = \frac{e^{f_k(x)}}{\sum_{l=1}^{K} e^{f_l(x)}}, \ k = 1, 2, \ldots, K.$$

   b) *Para* $k = 1$ *hasta* K*:*

      i. *Calcular* $r_{ikb} = y_{ik} - p_k(x_i), \ i = 1, 2, \ldots N$.

      ii. *Ajustar un árbol de regresión a los objetivos* $r_{ikb}, \ i = 1, 2, \ldots N$*, con regiones terminales* $R_{jmb}, j = 1, 2, \ldots, J_m$.

      iii. *Calcular*

      $$\gamma_{jmb} = \frac{K-1}{K} \frac{\sum_{x_i \in R_{jmb}} r_{ikb}}{\sum_{x_i \in R_{jmb}} |r_{ikb}|(1-|r_{ikb}|)}, \ j = 1, 2, \ldots J_b.$$

      iv. *Actualizar* $f_{kb}(x) = f_{k,b-1}(x) + \sum_{j=1}^{J_b} \gamma_{jmb} \mathbb{1}(x \in R_{jmb})$.

3. *El resultado es* $\hat{f}_k(x) = f_{kB}(x), \ k = 1, 2, \ldots K$.

En *boosting*, así como en *Random Forest*, la importancia de las variables y la dependencia parcial puede ser calculada. Esto se logra con lo presentado en la subsección 2.2.2, con los métodos que no involucran la muestra *OOB*.

2.3.3. Aspectos adicionales de *boosting*

La profundidad de los árboles

Una práctica estándar es que todos los árboles en la expansión $f_B(x) = \sum_{b=1}^{B} T(x; \Theta_b)$ tengan el mismo tamaño, es decir, $M_b = M \; \forall b$, definida por el número de nodos finales. Por ello, la elección de M se vuelve una estimación de parámetro fundamental.

Con este propósito, vale la pena pensar en las propiedades de

$$\eta(X) = \arg\min_{f} E_{XY} L(Y, \hat{f}(X)), \tag{2-58}$$

donde $\eta(x)$ es la función con el mínimo riesgo de predicción en datos futuros de la distribución conjunta (X, Y), encontrada en Hastie *et al.* (2009). Para aproximarla, es útil una propiedad acerca del grado de interacción que las variables $X = (X_1, X_2, \ldots, X_p)$ tienen entre sí. La propuesta explicada en Friedman *et al.* (2000) es la expansión ANOVA (análisis de varianza):

$$\eta(X) = \sum_{j} \eta_j(X_j) + \sum_{jk} \eta_{jk}(X_j, X_k) + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l) + \ldots. \tag{2-59}$$

La primera suma $\sum_j \eta_j(X_j)$ solo considera variables individuales; las funciones $\eta_j(X_j)$ son conjuntamente la mejor aproximación a $\eta(X)$, y cada una de ellas es llamada el "efecto principal" de X_j. Consecuentemente, $\sum_{jk} \eta_{jk}(X_j, X_k)$ es sobre las funciones de dos variables que mejor se ajustan a $\eta(X)$, ya considerando los efectos principales. Estas funciones son las interacciones de segundo orden de cada par (X_j, X_k). Lo mismo aplica para las interacciones de tercer orden para (X_j, X_k, X_l), y así sucesivamente.

Estas interacciones están reflejadas en la profundidad M de los árboles de decisión. En este método, el mínimo de interacciones es $M = 2$, cuando el árbol solo tiene una partición y solo usa los efectos principales. $J = 3$ incluye las interacciones entre dos variables. Por lo tanto, el máximo de interacciones posibles es de $M - 1$. Para muchos problemas encontrados en la práctica, los efectos de interacción de pocas variables son suficientes para explicar un fenómeno, por lo que árboles muy profundos tienen menor precisión. El valor de M, por ende,

debería reflejar el nivel de interacciones en $\eta(x)$.

La experiencia de Hastie *et al.* (2009) indica que interacciones derivadas de $4 \leq M \leq 8$ tienen buenos resultados en la mayoría de los casos. Usualmente $M = 2$ será insuficiente, y $M > 10$ no mejora las predicciones sustancialmente.

Submuestreo

Así como en *bagging* y *Random Forest* se usa el remuestreo *bootstrap* para construir árboles (ver sección 2.2), Friedman (1999) trabajó en *stochastic gradient boosting*, en el que en cada iteración del algoritmo de *boosting* por gradiente se elige una fracción η de los datos de entrenamiento, sin reemplazo.

Las ventajas encontradas en el artículo mencionado son la reducción en los tiempos de cómputo, así como mayor precisión en la predicción. Esto deja como los parámetros a elegir en *boosting* a J (profundidad), B (árboles), ν (*learning rate*, encogimiento) y η, el submuestreo.

Capítulo 3

Aplicaciones

If a machine is expected to be infallible, it cannot also be intelligent.

— Alan Turing

En este capítulo se pondrán en práctica los conceptos vistos a lo largo de este trabajo, desde los métodos para evaluar a los ajustes, hasta las diferentes herramientas de interpretación de los algoritmos de *Random Forest*, incluyendo *bagging*, y *boosting*. En cada uno de los tres ejemplos se mostrarán las capacidad de predicción de estos dos algoritmos, así como comparaciones con modelos de regresión, ya sea lineal o logística. Esto se logrará a lo largo de la evaluación y selección de sus ajustes, y se complementarán los análisis con gráficas. Estas gráficas son de creación propia, tomando como inspiración los trabajos de Efron y Hastie (2016), Hastie *et al.* y James *et al.* (2017) principalmente. Al empezar, se verán brevemente los paquetes en `R` que se usaron en los ejemplos.

En primer lugar, se tendrá una tarea de regresión con la base de datos `California Housing`, donde se agrupan datos de viviendas en conjuntos y el objetivo es predecir la mediana del valor de los hogares en esos grupos. Lo que se explorará con la base son la importancia de variables, la dependencia parcial, y los resultados de las muestras *Out-of-bag*. Se revisarán los resultados de cuatro algoritmos: *bagging*, *Random Forest* y *boosting* con dos funciones de pérdida diferentes.

Posteriormente, se ilustrará una tarea de clasificación de gran escala, tanto en número

de observaciones como número de variables explicativas. La base de datos `MNIST`, donde se tienen números manuscritos y se intentará predecir qué dígito se escribió, servirá para probar el submuestreo en *boosting* y se verá el primer ejemplo de una matriz de confusión.

Finalmente, la base de datos `Ancestry`, donde se muestra información genética para predecir la ascendencia de una persona de entre cuatro clases distintas, proveerá otra tarea de clasificación, con un número bajo de observaciones. Como se verá, esto traerá comparaciones interesantes entre *Random Forest* y *boosting*, pues tendrán rendimientos distintos a los de las bases de datos anteriores. Se discutirá también de las diferencias entre la importancia de variables e índices como la V de Cramér.

Paquetes en `R`

Los algoritmos vistos de *bagging* y *Random Forest* pueden ser implementados en `R` con el paquete `randomForest`, basado en el código elaborado por Leo Breiman y Adele Cutler, los creadores del método. Con él pueden realizarse ajustes modificando parámetros como profundidad y variables permitidas por nodo, además de obtener información como la importancia de variables.

El paquete `gbm` implementa el algoritmo de boosting por gradiente con una variedad de funciones de pérdida, aunque no incluye la de Huber en el caso de regresión. Además, se encuentra "retirada", por lo que se le da el mantenimiento necesario para seguir siendo instalada y usada, pero ya no es actualizada ni activamente desarrollada.

`H2O.ai` es una plataforma dedicada al *machine learning* y puede ser utlizada en `R` a través del paquete `h2o`. Mediante sus funciones `h2o.randomForest` y `h2o.gbm` pueden aplicarse los algoritmos que se han revisado, con una amplia variedad de parámetros como los de los paquetes anteriormente mencionadas, además de ofrecer las métricas de rendimiento de los conjuntos de entrenamiento, prueba y validación cruzada. Otra de las opciones es *early stopping*, que detiene el entrenamiento del modelo cuando el promedio móvil de una medida, como la media del cuadrado de los residuos o la devianza, no mejora por una cantidad y un número de iteraciones predeterminados.

3.1. Viviendas de California

La base de datos de las viviendas de California, Estados Unidos, recopila información agrupada por conjuntos de casas, reuniendo 20,460 de ellos en el censo de aquel estado norteamericano en 1990. Esta base de datos fue trabajada en Hastie *et al.* (2009), subsección 10.14.1. La base de datos original contenía algunas variables diferentes a las que se usaron en los siguientes ajustes; estas son `Longitude`, `Latitude`, `HouseAge`, `TotalRooms`, `TotalBedrooms`, `Population`, `Households`, `MedInc` y `MedianHouseValue`:

Longitude	Latitude	HouseAge	TotalRooms	TotalBedrooms	Population	Households	MedInc	MedianHouseValue
-122.23	37.88	41	880	129	322	126	8.3252	452600
-122.22	37.86	21	7099	1106	2401	1138	8.3014	358500
-122.24	37.85	52	1467	190	496	177	7.2574	352100
-122.25	37.85	52	1274	235	558	219	5.6431	341300
-122.25	37.85	52	1627	280	565	259	3.8462	342200

Tabla 3-1: Primeras cinco observaciones de la base de datos `California Housing` original ($n =$ 20,460).

La variable respuesta, `MedianHouseValue`, fue transformada para estar expresada en unidades de 10,000. Asimismo, se trabajaron con los promedios de habitaciones, dormitorios y personas en las viviendas del conjunto. Por ende, no se trabajó con `TotalBedrooms`, `TotalRooms` y `Households`, para evitar trabajar con cantidades numéricas grandes. Estas también fueron las variables usadas en Hastie *et al.* (2009).

De esta manera, los datos a trabajar lucen como:

Longitude	Latitude	HouseAge	Population	MedInc	AveBedrms	AveRooms	AveOccup	MedianHouseValue
-122.23	37.88	41	322	8.33	1.02	6.98	2.56	4.52
-122.22	37.86	21	2401	8.30	0.97	6.24	2.11	3.59
-122.24	37.85	52	496	7.26	1.07	8.29	2.80	3.52
-122.25	37.85	52	558	5.64	1.07	5.81	2.55	3.41
-122.25	37.85	52	565	3.85	1.08	6.28	2.18	3.42

Tabla 3-2: Observaciones de la base de datos `California Housing` modificada ($n =$ 20,460). Los datos aquí se presentan redondeados a dos dígitos.

Se realizaron diversos ajustes de *bagging*, *Random Forest*, *boosting* con dos funciones de pérdida distintas y regresión lineal. Fueron evaluados con validación cruzada de cinco partes (ver subsección 1.3.2), y con base en sus resultados se eligieron tres de ellos, en los que se estimó el error de prueba con *repeated training-test* (ver subsección 1.3.1), diviendo al conjunto

de datos en $\frac{2}{3}$ para entrenar y $\frac{1}{3}$ para evaluar.

Ajuste	ECM	
	`Train`	`Test`
Random Forest	.034	.235
Boosting (cuadrado de residuos)	.045	.199
Boosting (Huber)	.074	.196
Regresión lineal	.526	.530

Tabla 3-3: Resultados de los ajustes seleccionados de cada algoritmo para `California Housing`. El ECM se calculó en `train` ajustando con todas las observaciones y evaluando con los mismos (n = 20,460; error aparente), en `test` usando *repeated training-test* con una división de $\frac{2}{3}/\frac{1}{3}$, promediando 100 veces para los algoritmos basados en árboles y 1000 veces para la regresión lineal.
La R^2 de la regresión lineal es 0.606. Las tablas 3-4, 3-5 y 3-6 muestran los parámetros elegidos para *Random Forest* y *boosting*.

Los algoritmos dedicados a predicción tienen un rendimiento claramente más eficiente que la regresión lineal, especialmente aquellos de *boosting*, el error de prueba alcanzando 0.196, tres veces menor que aquél de regresión lineal. A propósito del modelo de regresión lineal: fueron omitidas la variables explicativas `AveOccup` y `Population`, pues se descubrió que incluirlas sobreajustaba en varias de las repeticiones de *repeated training-test* (el ECM alcanzaba valores muy altos, algunas veces mayores que cinco). Las cuestiones de estas dos variables solo fueron notadas después de realizar la regresión lineal; los algoritmos basados en árboles, realizados en primer lugar, demostraron no tener esa clase de problemas. Por otro lado, la regresión lineal, a pesar de tener el error `test` más grande, es el método que tiene la menor diferencia entre los resultados `train` y `test`. Además, un ajuste de regresión lineal es casi instantáneo, por lo que se aprovechó para hacer 1,000 repeticiones, a comparación de las 100 de los otros algoritmos, en las que cada repetición tomó entre 1 y 2 minutos en computarse.

Las repeticiones de *repeated training-test* de la regresión lineal sirvieron para estabilizar su promedio, ya que las estimaciones de sus errores de prueba fueron más variables. Esto se puede afirmar ya que la desviación estándar de los errores de regresión lineal fue de 0.013, mientras que las de los otros algoritmos fueron menores que 0.008, como se visualiza en la figura 3-1.

Adicionalmente, la figura mencionada permite observar que ambas formas de *boosting* son

esencialmente iguales, aunque se usará la versión de la función de pérdida de Huber para un análisis más adelante, por tener el menor error basado en la base de datos disponible. De cualquier manera, ambos son consistentemente más precisos que *Random Forest*.

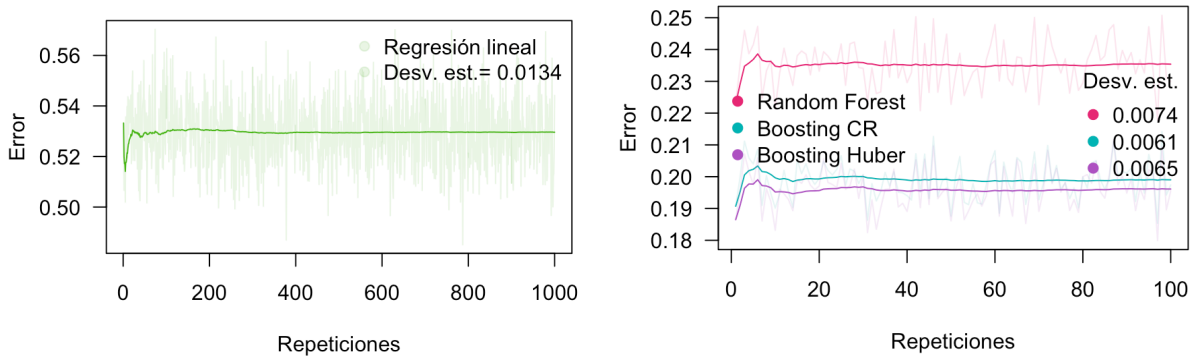


Figura 3-1: Errores (media del cuadrado de los residuos) en cada repetición de *repeated training-test* (líneas tenues), donde se dividió a la base de datos `California Housing` ($n = 20{,}460$) en $\frac{2}{3}$ para entrenar y $\frac{1}{3}$ para evaluar. Las curvas sólidas son los promedios de los errores a lo largo del proceso. CR: pérdida de cuadrado de los residuos.

Hastie *et al.* (2009), subsección 10.14.1, reportaron un error de prueba de 0.31 en esta base de datos, calculado como la media del valor absoluto de los residuales $|Y - \hat{f}(X)|$. Su ajuste fue de $B = 800$ iteraciones, realizado con la función de pérdida de Huber (α no especificada), una profundidad de $J = 6$ y un *learning rate* de $\nu = 0.1$. Comparado con el ajuste aquí realizado con la función de Huber, $\alpha = 0.9$, $B = 1750$, $J = 6$ y $\nu = 0.099$ (ver tabla 3-6), el error como la media de los residuos en valor absoluto fue de 0.278, logrando un error similar.

Una última comparación relevante es con el mejor error de prueba obtenido con un solo árbol de decisión, como se vio en la figura 2-6 de la sección 2.1. Su error, como la media de los residuos al cuadrado, fue de 0.372. Es interesante observar cómo se logró disminuir esa cantidad: con *Random Forest* se llegó a 0.235; con *boosting* con el cuadrado de los residuos como función de pérdida, a 0.199 y con la función de Huber como pérdida, a 0.196 (tabla 3-3).

La importancia de las variables según el ajuste de *boosting* con pérdida de Huber (figura 3-2) muestra que la mediana del ingreso es, por mucho, la variable más importante en la determinación del precio de las viviendas (naturalmente, se espera que casas de mayor valor pertenezcan a personas con un ingreso más elevado). La ubicación usualmente es un factor importante, y

se confirma en este ajuste, ya que la longitud y latitud son la segunda y tercera variable más importante, respectivamente. El número de ocupantes del hogar tiene una relevancia similar a estas últimas. Contraintuitivamente, la edad de la vivienda y el promedio de cuartos y habitaciones no fueron factores relevantes. Además, la población fue la de menor importancia, y como se reveló más adelante en el proceso de experimentación, esta era la variable con problemas de multicolinealidad en el modelo de regresión.

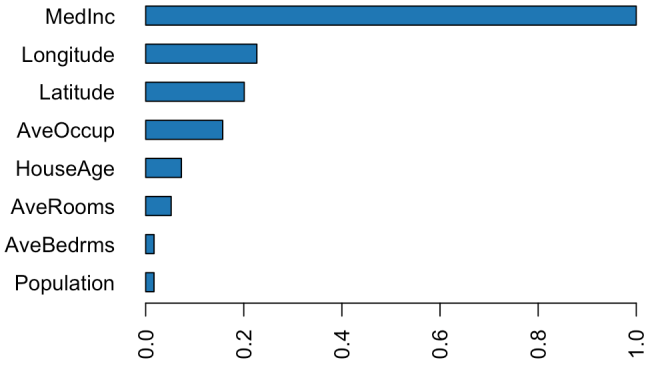


Figura 3-2: Importancia de variables de la base de datos `California Housing` escalada acorde al mayor valor (`MedInc`, mediana del ingreso). Correspondiente al ajuste de *boosting* con función de pérdida de Huber. Importancia calculada como la mejora relativa en las particiones de los nodos (ver subsección 2.2.2).

Es posible conocer más de las variables más influyentes (ver subsección 2.2.2) gracias a las gráficas de dependencia parcial en la figura 3-3, que en este caso mostrarán el comportamiento entre la distribución marginal de las variables explicativas y la media de la variable respuesta. Con estas visualizaciones se puede saber que en promedio, el precio de las viviendas crece de 20,000 a 35,000 dólares aproximadamente mientras el ingreso aumenta, pero cuando éste llega a 10 mil dólares, el precio se mantiene en 35,000. Por otra parte, el precio de las viviendas incrementa ligeramente su varianza en función de la edad, pero la media permanece estable. Estas aseveraciones están en línea con lo encontrado en la importancia de las variables, que mostraban el ingreso como lo más influyente, mientras que la edad de las casas exhibió poca relevancia.

Se puede conocer la dependencia parcial respecto a dos variables simultáneamente. Sabiendo que la ubicación está dada por longitud y latitud (es decir, las dos variables más importantes

después del ingreso), resulta de interés observar su comportamiento conjunto. Esto se obtiene en la figura 3-4, donde se observa que los conjuntos de casas más al sureste, en la costa, tienen las viviendas más costosas, y los precios disminuyen progresivamente mientras se alejan de ese punto.

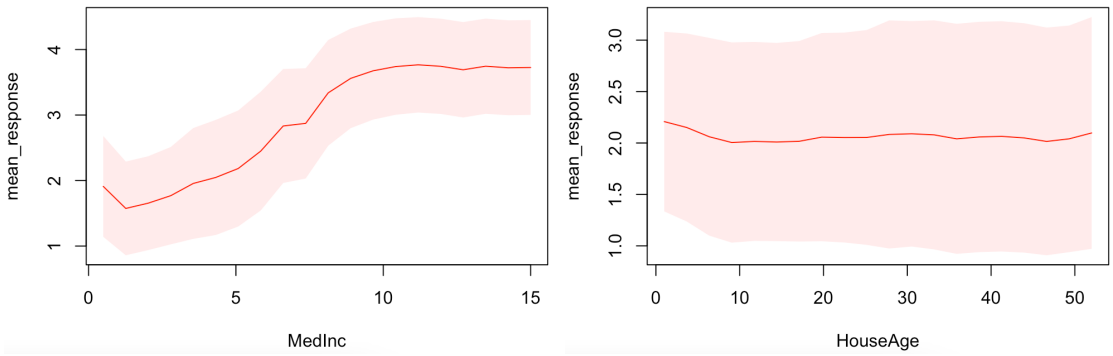


Figura 3-3: Gráficas de dependencia parcial de la media de la respuesta (mediana del valor de los hogares en unidades de 10,000 dólares) con las variables `MedInc` (mediana del ingreso en miles de dólares) y `HouseAge` (mediana de la edad en años de las viviendas) de la base de datos `California Housing`. La desviación estándar de la respuesta está sombreada.

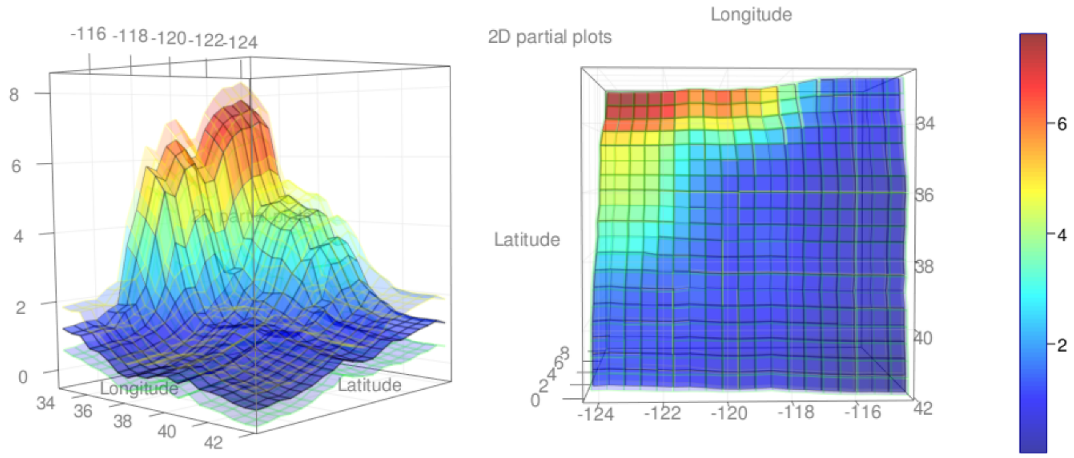


Figura 3-4: Gráfica de dependencia parcial de la media de la respuesta (mediana del valor de los hogares en unidades de 10,000 dólares) con longitud y latitud en conjunto de la base de datos `California Housing`. Se muestran dos perspectivas distintas del mismo objeto.

3.1.1. Selección del ajuste

Para seleccionar los parámetros de los ajustes que se presentaron anteriormente, se calculó el error como la media de los residuos al cuadrado por medio de validación cruzada, repitiéndolo cinco veces.

Random Forest

La implementación de *Random Forest* de `h2o` tiene una diferencia con el algoritmo de Adele Cutler y Leo Breiman: no realiza muestreo con reemplazo en cada árbol, sino que selecciona sin reemplazo 63.2 % de los datos (ver subsección 2.2.2), simulando el número de observaciones que se obtendría con *bootstrap*, mas no el número de veces que tales observaciones aparecerían. Además, el tamaño de los árboles no está controlada por el número mínimo de observaciones que deben tener las regiones finales, sino por la profundidad, que por *default* es de 20. En el apéndice correspondiente a `California Housing` se hace una exploración similar a la siguiente con el algoritmo original, haciendo uso del paquete `randomForest`. Se obtuvieron resultados equivalentes y adicionalmente, se comprueba que con el muestreo con *bootstrap* se obtiene, en promedio, 63.2 % de los datos.

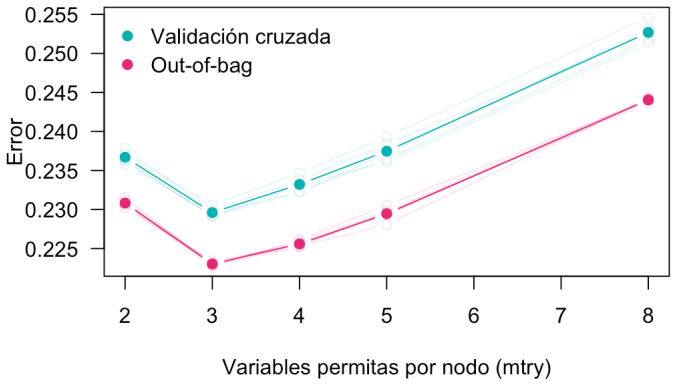


Figura 3-5: Resultados de `California Housing` ($n = 20{,}460$). Comparación del error de prueba entre estimaciones *out-of-bag* y validación cruzada en cinco partes, con el algoritmo de *Random Forest* construidos con 1000 árboles. El error es la media de los residuos al cuadrado y se presenta el promedio de cinco repeticiones. Cada repetición tomó alrededor de una hora en completarse.

Se calculó el error por medio de validación cruzada teniendo en mente que se busca hacer

una comparación con *boosting*, cuyo error de prueba será calculado de esa manera, al no tener observaciones *out-of-bag* en aquel algoritmo. Se encontró que el error por validación cruzada en cinco partes estima un error de prueba levemente mayor al de *OOB*. La diferencia entre las dos estimaciones es de solo 0.007 en promedio, pero lo más importante es que siguen la misma tendencia: el error más pequeño se consigue con la recomendación de `mtry`= $\sqrt{8} \approx 3$. El error más grande se consigue con *bagging* (es decir, `mtry`= 8, cuando todas las variables posibles son probadas para hacer las particiones en cada nodo de los árboles).

`mtry`	ECM		
	`Train`	OOB	5-VC
2	.040	.231	.237
3	.034	.223	.230
4	.034	.226	.233
5	.034	.229	.237
8	.035	.244	.253

Tabla 3-4: Resultados de `California Housing` (n =20,460) entre el ECM de entrenamiento y estimaciones *out-of-bag* y validación cruzada en cinco partes.

Boosting con el cuadrado de los errores como función de pérdida

La primera función de pérdida con la que se hizo la búsqueda del mejor ajuste en *boosting* por gradiente es el cuadrado de los residuos. Los parámetros a regular son la profundidad J, definido como el número de particiones en los árboles, ubicado entre dos y ocho, y el *learning rate* ν, que se encuentra entre 0.001 y 0.1. El número de árboles quedó definido por la convergencia de la medio de los residuos al cuadrado: se detuvo el ajuste cuando el promedio móvil de aquella función de pérdida no disminuyó por 0.0001 durante cinco iteraciones. Si esto no sucedió, 10,000 fue el máximo de árboles permitidos. La decisión de este espacio de parámetros se basó en lo presentado en Hastie *et al.* (2009) en la subsección 10.12.1: "la mejor estrategia parece ser tener un ν muy pequeño ($\nu < 0.1$) y elegir el número de árboles con *early stopping*". Se hicieron cinco repeticiones de este diseño, con las que se promedió la media de los residuos al cuadrado como un estimado del error de prueba.

Profundidad	*Learning rate*	Árboles	ECM	
			`Train`	`Test`
8	.051	1979	.036	.195
8	.032	2447	.043	.194
7	.096	1749	.041	.195
7	.059	2219	.048	.193
7	.029	2774	.056	.193
6	.084	2179	.059	.194
6	.081	2066	.061	.194
6	.032	3531	.068	.194
6	.027	3963	.064	.193
6	.020	4260	.075	.194
5	.078	2821	.076	.194
5	.065	2956	.079	.193
5	.014	5876	.096	.196
4	.092	2815	.094	.197
4	.090	3062	.097	.196
3	.067	4831	.128	.203
3	.036	6211	.137	.206
3	.019	7712	.148	.210
3	.014	9496	.151	.211
2	.069	6558	.174	.220

Tabla 3-5: 20 ajustes generados de *boosting* con cuadrado de los residuos como función de pérdida para `California Housing` (n = 20,460). Se presenta el ECM promediado de cinco repeticiones de validación cruzada en cinco partes. La desviación estándar del error de prueba es de, a lo más, 0.0035. Cada repetición de los 20 ajustes tomó una hora con 40 minutos.

Para facilitar el entendimiento de los resultados presentados en la tabla 3-5, se incluye la gráfica 3-6, donde se muestran los errores de entrenamiento y prueba en el mismo orden. La primera impresión que pueden dar estos valores es que, mientras más profundos son los árboles, más grande es la brecha entre los resultados `train` y `test`. Es decir, entre más información capturan los ajustes, más aprenderán de las estructuras particulares del conjunto en el que son entrenados, disminuyendo el error `train`. Aunque ante esta perspectiva es importante vigilar que no se produzcan sobreajustes, lo que se nota en los errores `test` (más visible en la figura 3-7) es que los mejores ajustes tienen mayor profundidad, por lo que la elección conjunta del número de árboles y el *learning rate* fue esencial para no sobreajustar.

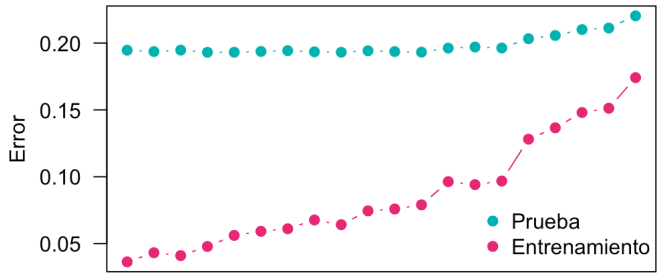


Figura 3-6: Resultados promedio de cinco repeticiones para los 20 ajustes generados para `California Housing`, con parámetros de profundidad J entre dos y ocho y aprendizaje ν entre 0.001 y 0.1. Se presenta el error como la media de los residuos al cuadrado, promediado de cinco repeticiones de validación cruzada en cinco partes. Están ordenados descendentemente, primero por J y después por ν. Más detalles se encuentran en la tabla 3-5.

Dado que tres ajustes obtuvieron un error de prueba menor que 0.193 (ver tabla 3-5), y que es muy probable que no haya diferencias reales importantes en su rendimiento, se elegirá al que tuvo menor desviación estándar (0.0005) en su error de prueba, es decir, el de profundidad 7 y aprendizaje 0.029.

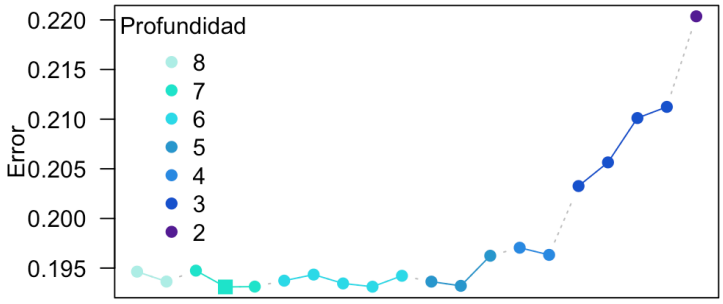


Figura 3-7: Error de prueba en los ajustes de *boosting* con el cuadrado de los residuos como función de pérdida para `California Housing`. Se presenta el error como la media de los residuos al cuadrado, promediado de cinco repeticiones de validación cruzada en cinco partes. De izquierda a derecha, primero están ordenados por profundidad y después por *learning rate*, de forma descendente como en la tabla 3-6. La figura cuadrada (en el cuarto lugar) representa al ajuste elegido.

Boosting con Huber como función de pérdida

Para la segunda función de pérdida, la función de Huber, se empezó probando cinco ajustes con parámetros de profundidad J y α de Huber aleatorios (este último parámetro es el umbral

por el que los residuos serán tratados con la función de pérdida de residuos al cuadrado o con el error absoluto, ver subsección 1.2.1), eligiendo 2250 árboles para los ajustes con árboles menos profundos, y 500 menos si eran de mayor profundidad por razones de tiempo de cómputo. Además, se fijó el *learning rate* ν en 0.099. Con base en los primeros ajustes, se probaron tres más, con la intención de encontrar estimaciones más bajas del error.

Árboles	Profundidad	α de Huber	ECM		Minutos
			`Train`	`Test`	
1750	8	0.8	0.040	0.195	80
	7	0.8	0.074	0.193	35
	6	0.9	0.074	60.191	25
	6	0.2	0.118	0.202	37
2250	4	0.9	0.108	0.193	12
	4	0.1	0.160	0.213	16
	3	0.5	0.171	0.213	15
	3	0.3	0.177	0.217	9

Tabla 3-6: Ajustes para `California Housing` ($n = 20{,}460$) de *boosting* con Huber como función de pérdida y $\nu = 0.099$. Se presenta el error de prueba promediado de cinco repeticiones de validación cruzada en cinco partes.

Es claro que el mejor ajuste se encuentra con una combinación óptima de profundidad y α de Huber: es favorable una profundidad moderada con un valor de α alto ($J = 6$, $\alpha = 0.9$: error de prueba de 0.1913). Un α elevado, sin importar si la profundidad es baja ($J = 4$) o relativamente alta ($J = 7$, $J = 8$) termina en buenos resultados, aunque no tan convincentes. En cambio, si tanto α como J son bajos, los resultados son, en comparación, desfavorables. Esto implica que tratar solo a los residuos más altos con la función absoluta como pérdida es ideal para esta base de datos.

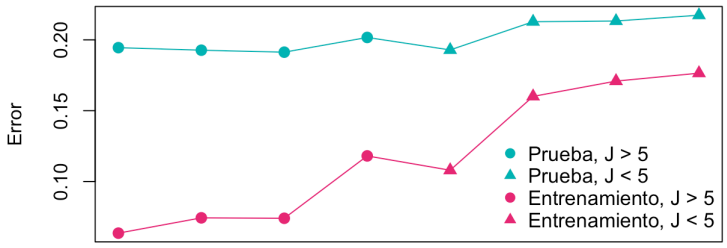


Figura 3-8: Resultados promedio de cinco repeticiones para los ocho ajustes de *boosting* con pérdida de Huber para `California Housing`. Se presenta el error como la media de los residuos al cuadrado, promediado de cinco repeticiones de validación cruzada en cinco partes. $J \in [3, 8]$, $\alpha \in [0.3, 0.9]$ y $\nu = 0.099$. Ordenados como en la tabla 3-6.

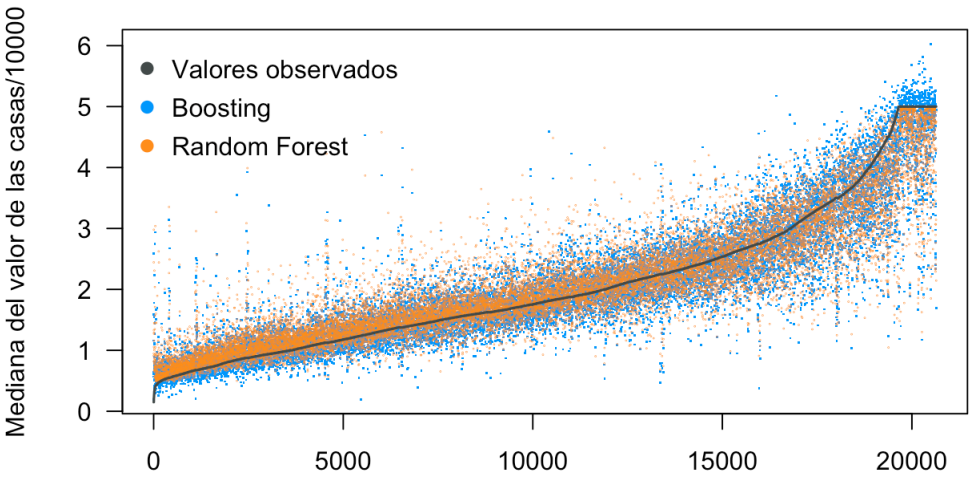


Figura 3-9: Comparación de las 20,640 estimaciones de validación cruzada con el ajuste de *Random Forest* y el de *boosting* con función de Huber para `California Housing`. La varianza de las estimaciones crece conforme el valor de las casas aumenta en ambos métodos, aunque es más pronunciado en *Random Forest*. Por construcción, esas predicciones están acotadas por los valores extremos de los datos reales; en *boosting* no aplica esta restricción.

3.2. MNIST

La base de datos `MNIST`[1] consiste en dígitos manuscritos, procesados para facilitar su manejo. Las 70,000 observaciones fueron tomadas de empleados de la Oficina del Censo de Estados Unidos, así como de estudiantes de bachillerato, y se dividió por los creadores en un conjunto de entrenamiento de 60,000 y un conjunto de prueba de 10,000. La tarea es clasificar el número manuscrito en uno de los 10 dígitos, es decir, la variable respuesta es categórica con 10 niveles: $0, 1, \ldots, 9$. Se tienen 784 variables predictoras de tipo continuo, sin embargo, algunas de ellas son constantes (todas las observaciones tenían el mismo valor), por lo se trabajó con 707. Esta base de datos es trabajada en Efron y Hastie (2016), subsección 18.2.

Figura 3-10: Los dígitos fueron transformados en imágenes de 28 pixeles de largo y 28 de ancho, cada pixel es representado en la base como un número que indica su cantidad de gris. Esto resulta en 784 pixeles que son usados como variables predictoras. Fuente: Efron y Hastie (2016).

Debido al gran tamaño de la base de datos y las limitaciones de cómputo particulares a este trabajo, se usó la metodología *train, validation, test* (sección 1.3, figura 1-5) para explorar los parámetros que mejor funcionan con esta base: se entrenaron ajustes de *Random Forest* y *boosting* con aproximadamente 48,000 observaciones, es decir, el 80 % del conjunto de entrenamiento original, y se estimó el error de prueba de cada una con el 20 % restante, funcionando como conjunto de validación. Se eligió a un grupo selecto de ellos, así como a un modelo de regresión logística con fines comparativos (ver tabla 3-8), cuyo error de validación fue estimado con la técnica *repeated training-test* (subsección 1.3.1). Finalmente, se seleccionó al mejor de ellos, para que su estimación del error fuera evaluada con el conjunto `test` original.

[1] por sus siglas en inglés, la base de datos modificada del Instituto Nacional de Estándares y Tecnología.

Conjunto (Total)	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
`Train` ($n = 48,005$)	9.88	11.26	9.97	10.22	9.68	9.04	9.88	10.38	9.76	9.92
`Valid` ($n = 11,995$)	9.84	11.14	9.75	10.20	9.96	9.03	9.78	10.70	9.70	9.90
`Test` ($n = 10,000$)	9.80	11.35	10.32	10.10	9.82	8.92	9.58	10.28	9.74	10.09

Tabla 3-7: Representación porcentual de cada dígito en los subconjuntos de datos. Los conjuntos `Train` ($n = 48,005$) y `Valid` ($n = 11,995$) representan 80 % y 20 % del conjunto de entrenamiento original ($n = 60,000$), respectivamente, y fueron usados para los ajustes iniciales, donde se exploraron los parámetros de los algoritmos. El conjunto `Test` ($n = 10,000$) fue determinado por los proveedores de la base. Cada subconjunto tiene proporciones similares de cada dígito.

Se probaron ajustes de *Random Forest* y seis de *boosting*, estos últimos con diferentes profundidades J, pero fijando el *learning rate*, $\nu = 0.1$. Posteriormente, se repitieron los ajustes de *boosting* pero con un submuestreo de $\eta = \frac{1}{4}$ (ver subsección 2.3.3), siguiendo la sugerencia de Hastie *et al.* (2009): "para conjuntos de datos grandes, η puede ser sustancialmente menor [que un medio]". Todos los ajustes fueron realizados con 1000 árboles, aunque se revisó la evolución del error de prueba a través de las iteraciones para encontrar su punto mínimo, así como evitar el sobreajuste.

Ajuste	Error de predicción (%)			Devianza		
	`Train`	`Valid`	`Test`	`Train`	`Valid`	`Test`
Random Forest	0.02	3.38	-	0.07	0.25	-
Boosting	0.00	2.26	2.09	0.00	0.13	0.12
Reg. logística	5.52	9.56	-	0.20	0.47	-

Tabla 3-8: Resultados para `MNIST` del ajuste elegido de *boosting*, comparado con el ajuste generado de *Random Forest* y el modelo de regresión logística. *Random Forest* se hizo con 1000 árboles, mientras que *boosting* se hizo con 700 y una profundidad $J = 6$.
Los resultados `train` se midieron sobre el conjunto de entrenamiento completo original ($n =$ 48,005). El error `valid` se calculó con *repeated training-test* con división de $\frac{2}{3}$ y $\frac{1}{3}$ del conjunto de entrenamiento, promediando el resultado múltiples veces (ver tabla 3-9). El conjunto `test` ($n =$ 10,000) se guardó hasta el final, para probarlo con uno solo de los ajustes selectos.

Los algoritmos enfocados en predicción tuvieron un rendimiento considerablemente mejor que el modelo de regresión logística. La estimación del error de prueba, promediado múltiples veces con un conjunto de validación `valid`, fue de 2.26 % en un ajuste de *boosting* de $B = 700$ árboles, con una profundidad $J = 6$ y un *learning rate* de $\nu = 0.1$. Se verá más adelante que con un número mayor de iteraciones, el algoritmo comienza a sobreajustar con estas especificaciones.

Random Forest, por otro lado, fue resistente al sobreajuste (como se previo en la subsección 2.2.1), aunque tuvo un error de 3.38 %, que no es competitivo. La tasa de error de la regresión logística es más de tres veces mayor al de *boosting*, en 9.56 %. Los tres métodos lograron tener devianzas bajas (calculada como se ve en la subsección 1.2.1), aunque de nuevo, *boosting* fue notablemente menor, de solo 0.134. El resultado en el conjunto `test` mostró que dicho ajuste fue la elección correcta, pues da otra estimación del error de prueba, una donde ninguna observación había sido usada anteriormente, del 2.09 %.

Ajuste	Repeticiones	Desviación estándar	
		Error de pred.	Devianza
Random Forest	30	.0023	.0013
Boosting	30	.0080	.0009
Reg. logística	30	.0189	.0022
	100	.0187	.0021

Tabla 3-9: Desviaciones estándar de las estimaciones del error de prueba en conjuntos `valid`, correspondientes los ajustes descritos en la tabla 3-8 para `MNIST`. Se incluyen los resultados de un subconjunto de 30 repeticiones de regresión logística, del total de 100.

En la tabla 3-9 se encuentran los detalles de las repeticiones para conseguir el error de prueba a través de repetidas particiones del conjunto de entrenamiento original en `train` y `valid` en $\frac{2}{3}$ y $\frac{1}{3}$ respectivamente. Dado que cada ajuste de *Random Forest* tomó cerca de una hora con 20 minutos, y los de *boosting* dos horas con 10 minutos, el número de repeticiones fue solo de 30 cada uno. Incluso con esa limitante, la desviación estándar de aquellos ajustes fue baja para ambas funciones de pérdida, probando que son robustos en cuanto a cambios en los conjuntos de datos. Por otro lado, la regresión logística, con 100 repeticiones, tuvo mayor varianza en su estimación. Se agregó el cálculo con un subconjunto de las primeras 30 iteraciones, para tener una comparación más justa con los otros dos ajustes.

Los resultados de la subsección 18.2 de Efron y Hastie (2016), figura 18.4, muestran que con *Random Forest* se logró un error de 2.8 % aproximadamente, menor que la aquí conseguida de 3.38 %. Asimismo, la regresión logística que ellos reportan es de 7.2 %.

La tabla 3-10 es la matriz de confusión obtenida del conjunto `test`, según el ajuste de

boosting con profundidad $J = 6$. El dígito más difícil de clasificar fue el `5`, con un error de 2.80 %. Fue más erróneamente clasificado como `3`, seguido del `6` y el `8`. Por otro lado, el dígito con menor error de clasificación (1.15 %) fue el `1`. En términos absolutos, los pares de dígitos que más frecuentemente fueron confundidos fueron el `7` con el `2` y el `4` con el `9`, con 13 instancias cada uno.

Reales	**Predicciones**										**Error (%)**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	
0	967	0	1	0	0	2	4	1	4	1	1.33
1	0	1122	2	3	1	1	3	1	2	0	1.15
2	3	0	1006	6	0	1	1	5	10	0	2.52
3	0	0	3	989	0	5	1	7	3	2	2.08
4	1	0	5	0	959	0	2	0	2	13	2.34
5	2	0	1	8	1	867	4	2	4	3	2.80
6	4	2	1	0	2	9	936	0	4	0	2.30
7	1	1	13	1	1	0	0	1002	2	7	2.53
8	2	0	2	1	3	0	0	1	960	5	1.44
9	1	4	2	7	5	0	0	5	2	983	2.58

Tabla 3-10: Matriz de confusión para las 10,000 observaciones del conjunto *test* con el ajuste de *boosting* con profundidad $J = 6$, *learning rate* $\nu = 0.1$ y 700 árboles para la base de datos `MNIST`. El error general fue de 2.09 %: 209 errores.

3.2.1. Selección de los ajustes

A continuación se muestran los resultados de los ajustes probados, donde inicialmente se estimó el error de prueba con una división aleatoria del conjunto de entrenamiento en 80 %-20 %. El proceso se hizo una sola vez, pues fue de manera exploratoria para descubrir los parámetros que mejor funcionan con la base de datos.

Random Forest fue mejor que los ajustes poco profundos ($J \leq 3$) de *boosting*, sin embargo, a partir de la profundidad $J = 4$, este último logró resultados más precisos. Esto es cierto tanto para el error de predicción, como en la devianza, y se cumple, sea *boosting* con submuestreo o sin él. Sobre este punto, los ajustes que se hicieron con el conjunto de entrenamiento completo ($\eta = 1$) fueron más eficientes que aquellos en los que no ($\eta = \frac{1}{4}$) únicamente en profundidades mayores ($J \geq 4$). En este ejemplo particular, por lo tanto, el submuestreo benefició a los ajustes

poco profundos.

Ajuste	η	J	Error (%)		Devianza	
			`Train`	`Valid`	`Train`	`Valid`
Random Forest	-	20	0.02	3.10	0.074	0.240
Boosting	1	2	1.84	5.03	0.164	0.301
		3	0.35	3.73	0.064	0.226
		4	0.00	2.26	0.00	0.107
		5	0.00	2.12	0.00	0.128
		6	0.00	2.21	0.00	0.143
		7	0.00	2.18	0.00	0.138
	$\frac{1}{4}$	2	0.76	4.40	0.051	0.153
		3	0.01	3.53	0.011	0.122
		4	0.00	2.97	0.002	0.113
		5	0.00	2.83	0.001	0.118
		6	0.00	2.63	0.00	0.120
		7	0.00	2.51	0.00	0.131

Tabla 3-11: Características y resultados de todos los ajustes realizados en la base de datos `MNSIT`, cada uno de 1,000 árboles. Se probó el error una sola vez en los conjuntos `train` ($n =$ 48,005) y `valid` ($n =$ 11,995). El error `test` fue calculado sobre todos los datos del conjunto de entrenamiento original. η representa la proporción de submuestreo y J la profundidad de los árboles.
En *Random Forest*, el valor de `mtry` fue el predeterminado de $\sqrt{p} = 28$, mientras que el *learning rate* en *boosting* fue de $\nu = 0.1$.

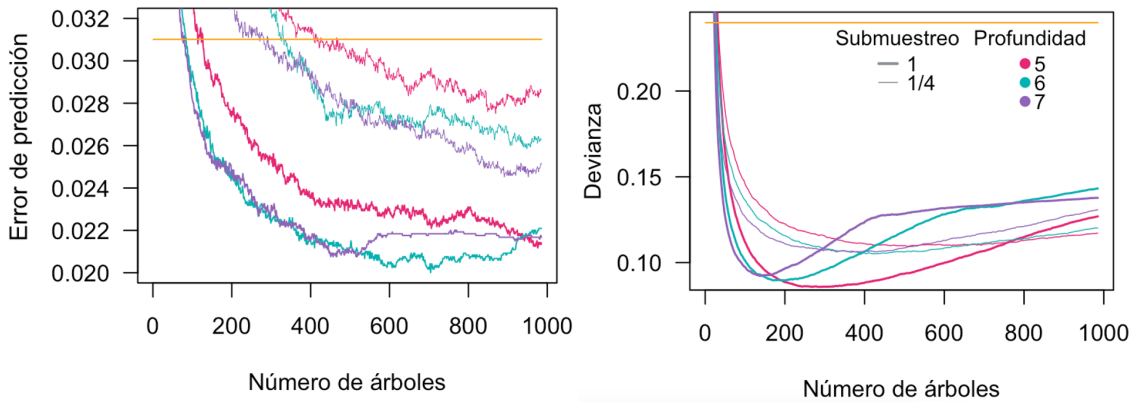


Figura 3-11: Evolución del error de predicción y la devianza para `MNIST` en los ajustes más profundos de *boosting* ($J = \{5, 6, 7\}$), tanto con submuestreo ($\eta = \frac{1}{4}$) como sin él. El *learning rate* es de 0.01. La línea amarilla representa el resultado de *Random Forest* después de 1000 árboles. Las estimaciones de las funciones de pérdida son con la sección `valid` de los datos (20 % del conjunto de entrenamiento original).

Un aspecto importante de estos ajustes es que no se aplicaron los parámetros de paro: los algoritmos recorrieron las 1,000 iteraciones solicitadas, sin importar si el error en el conjunto `valid` se elevaba. Esto trae una oportunidad de analizar qué parámetros son propensos a sobreajustar, además de interpretar los resultados con base en ambas funciones de pérdida utilizadas. El error de predicción disminuyó consistentemente en todos los ajustes hasta las 1000 iteraciones, con excepción de los más profundos ($J = \{6, 7\}$), sin submuestreo ($\eta = 1$), que alcanzaron los menores de este ejemplo. No obstante, el error comenzó a incrementar en algún punto del entrenamiento, un signo de sobreajuste. Por esta razón, se decidió usar el ajuste de profundidad $J = 6$, pero solo con $B = 700$ árboles.

La devianza sugiere aspectos interesantes: en los ajustes sin submuestreo, alcanza sus mínimos rápidamente, alrededor de los 200 árboles. Esto no se ve reflejado en un mínimo de los errores de predicción, por lo que una menor devianza no implica necesariamente un menor error de clasificación. Esta medida puede crecer, mientras se mantenga en niveles aceptables (permaneció por debajo de 0.15, aproximadamente) y el error de predicción siga disminuyendo. De hecho, Friedman (2001) habló de ello: "degradar la devianza con el sobreajuste en realidad mejora el error de clasificación. Aunque sea contraintuitivo, no es una contradicción; la devianza y el error miden diferentes aspectos de la calidad del ajuste. Por ende, el error de clasificación es menos sensible al sobreajuste".

Por último, se presenta el comportamiento del error de entrenamiento (conjunto `train`) y de prueba (conjunto `valid`). Aunque el error de entrenamiento llega rápidamente a cero, la evolución del error de prueba es evidencia de que no sobreajusta sino hasta después de los 700 árboles.

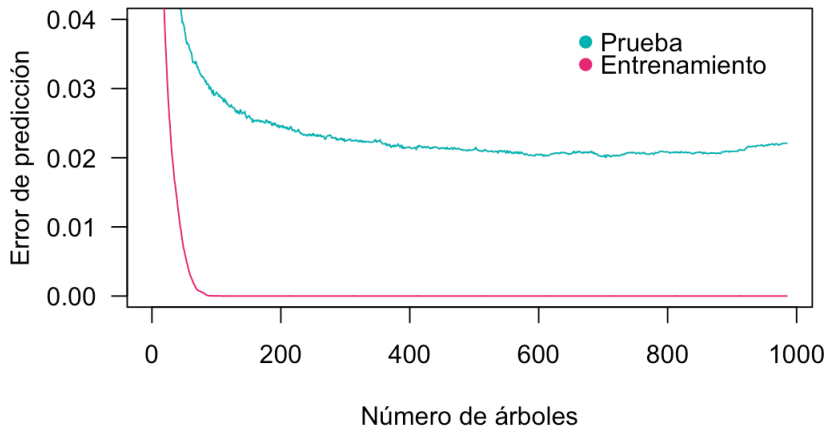


Figura 3-12: Error del ajuste de *boosting* con profundidad $J = 6$ y *learning rate* $\nu = 0.1$ para la base de datos `MNIST`. Se dividió al conjunto de entrenamiento original en subconjuntos `train` (80 %) y `valid` (20 %).

3.3. Ascendencia

Para la última ilustración se usará una base de datos dedicada a la composición étnica humana. Con base en información genómica, se intentará clasificar la ascendencia de personas estadounidenses en cuatro clases posibles: africana, afroamericana, europea y japonesa.

race	Snp1	Snp2	Snp3	Snp4	Snp5	...
AfricanAmerican	0	0	0	0	0	
AfricanAmerican	1	0	0	1	0	
European	1	0	0	0	0	
European	0	0	0	0	0	...
Japanese	0	0	0	0	0	
Japanese	0	0	0	0	0	
African	0	0	0	2	0	
African	0	0	0	1	0	

Tabla 3-12: Base de datos `Ancestry`, mostrando cinco de las 100 variables explicativas y dos observaciones de cada una de las clases existentes. En total, hay 47 observaciones de la clase `AfricanAmerican` y 50 de cada una de las demás. Una columna identificadora de individuos fue removida.

El número de observaciones es $n = 197$, considerablemente menor al de los ejemplos anteriores, mientras que el número de variables explicativas es 100: cada una es un polimorfismo de nucleótido único (SNP, por sus siglas en inglés). Dos de ellas son constantes, por lo que se

trabajará con $p = 98$ variables predictoras. Un SNP presenta medidas de dos alelos, donde cada uno puede ser A (tipo natural) o a (la mutación), resultando en tres niveles: $\{AA, Aa, aa\}$, que respectivamente se codifican como 0, 1 y 2. La base de datos proviene de Efron y Hastie (2016), presentada en su sección 13.5. A continuación se muestra un subconjunto de la base de datos.

Preámbulo

Esta base presenta un problema común en la estadística: datos faltantes. Existen diferentes maneras de lidiar con ellos, como eliminando las observaciones o imputándolas. En esta ocasión se optó por asignar la moda del SNP correspondiente de acuerdo a la clase. Por ejemplo, el dato faltante de la décima observación en `Snp13` fue imputada con la moda de su clase, `AfricanAmerican`, en esa variable.

La distribución de datos faltantes fue casi uniforme entre las cuatro clases:

`AfricanAmerican`	`European`	`Japanese`	`African`
9	11	12	10

Tabla 3-13: Datos faltantes de `Ancestry` ($n = 197$) según la clase. Esas 42 medidas fueron imputadas con la moda de las respectivas observaciones de SNP por clase.

La búsqueda de parámetros se basó en realizar validación cruzada en cinco partes (ver subsección 1.3.2) para obtener el error de prueba, además de repetir el proceso 10 veces. Como se verá más adelante, el pequeño tamaño de la base de datos implica una varianza significativa en las diferentes estimaciones por medio de validación cruzada. 10 repeticiones no son suficientes para estabilizar el error de prueba estimado, pero pueden ser útiles para identificar qué parámetros dan mejores resultados. Para que esta primera impresión sea más precisa, se puede intentar reducir la varianza evitando que cada división de la base de datos tenga alguna clase sobrerrepresentada en validación cruzada. Como hay ciertas clases más difíciles de clasificar, una mayor presencia de estas en una división (que implicaría una aparición menor en el resto) puede elevar el valor de la función de pérdida, a pesar de que la elección de parámetros sea razonablemente buena.

Para evitar los posibles desbalances entre clases que podrían resultar de particiones comple-

tamente aleatorias, se crearon 10 variables indicadoras, una para cada repetición, para asignar equitativamente las observaciones en las cinco divisiones. Como hay 50 observaciones por clase (excepto en una de ellas, que tiene 47), en una partición de validación cruzada hay 10 observaciones de cada una en una división dada (en dos o tres divisiones, habrá ocho o nueve de la clase de menor tamaño). Para ilustrar este concepto, se exhibe una tabla correspondiente a la distribución por clase de una de las particiones creadas.

`race`	División 1	División 2	División 3	División 4	División 5	Totales
`African`	10	10	10	10	10	50
`AfricanAmerican`	10	9	9	10	9	47
`European`	10	10	10	10	10	50
`Japanese`	10	10	10	10	10	50

Tabla 3-14: Distribución por clase de la base de datos `Ancestry` ($n = 197$) de una de las 10 particiones de datos que se usarán para validación cruzada. Todas las clases, salvo `AfricanAmerican`, tendrán 10 observaciones en cada división. En algunas particiones, en lugar de tener tres divisiones con nueve datos de aquella clase, habrá una con nueve y otra con ocho.

Evaluación de ajustes selectos

La evaluación de los ajustes se basó en dos métricas: la devianza y el error de predicción, tanto de manera global como por clase. En la tabla 3-15 se comparan ajustes elegidos de *Random Forest* y *boosting*, además de un modelo de regresión logística, en los cuales se estimaron las funciones de pérdida con 1000 repeticiones de validación cruzada de cinco partes.

Las menores devianzas (calculada como se ve en la subsección 1.2.1) se lograron con *boosting*, teniendo un valor de alrededor de 0.46, una décima menor que *Random Forest*. No obstante, en esta ocasión el ajuste de 1000 árboles de *Random Forest* fue el que obtuvo mejores resultados, con un error general de 12.79 %, impulsado principalmente por menores errores en las clases `African`, `European` y `Japanese`. La devianza de regresión logística fue de 1.93, muy alta a comparación de los algoritmos de predicción, aunque la diferencia en los errores de clasificación no fue tan remarcada, al ser de 26.64 %. La única clase donde tuvo un rendimiento comparable al de los demás fue `AfricanAmerican`.

De forma general, esa última clase fue la más complicada de clasificar correctamente, ya

que en todos los ajustes tuvo errores por arriba del 40 %. La clase `African`, que se espera que sea similar genéticamente a ella, fue la segunda con mayores error de clasificación, de al menos 22 %, con excepción del ajuste de *RandomForest*, donde se obtuvo 12.5 %.

Ajuste	**Conjunto**	**Devianza**	**Error de predicción (%)**				
			Global	**Af**	**AA**	**Eu**	**Jp**
Random Forest	`Train`	0.151	0.00	0.00	0.00	0.00	0.00
	`Test`	0.569	14.39	12.47	46.89	0.13	0.00
Boosting	`Train`	0.218	3.05	2.00	10.64	0.00	0.00
	`Test`	0.457	18.11	26.07	42.26	4.41	1.16
Regresión logística	`Train`	0.00	0.00	0.00	0.00	0.00	0.00
	`Test`	1.933	26.64	34.50	47.52	12.26	13.56

Tabla 3-15: Tres de los ajustes entrenados en la base de datos `Ancestry` ($n = 197$). De izquierda a derecha, las abreviaciones del error de clasificación son de `African`, `AfricanAmerican`, `European` y `Japanese`.
El error `train` se calculó con la totalidad de la base de datos (error aparente). El error `test` se obtuvo al promediar 1000 repeticiones de validación cruzada de cinco partes. Las características de los ajustes selectos pueden ser encontrados en la tabla 3-18.

Random Forest logró una clasificación casi perfecta para las clases `European` y `Japanese`. Esto explica su menor error de predicción global, a pesar de haber tenido un error en `AfricanAmerican` más alto que *boosting*.

El proceso de repetir validación cruzada 1000 veces (figura 3-13) reveló que la devianza es considerablemente más variable en *boosting*, al tener una desviación estándar de 0.023, contra aquella de *Random Forest* de 0.008. Aun así, la primera de ellas se mantiene notablemente más baja que la segunda. Las desviaciones estándar del error de clasificación fueron más similares entre ellas, aunque es posible decir los resultados de *Random Forest* en esta medida son más favorables.

Es importante mencionar que, al seleccionar un ajuste, se deben analizar sus errores por clase, como se ha hecho en los párrafos anteriores. Dependiendo del objetivo de la investigación, puede ser que haya interés en separar correctamente a los individuos pertenecientes a la clase `African` de aquellos de `AfricanAmerican`, por lo que se aceptaría un error más elevado en las otras clases, si fuese necesario.

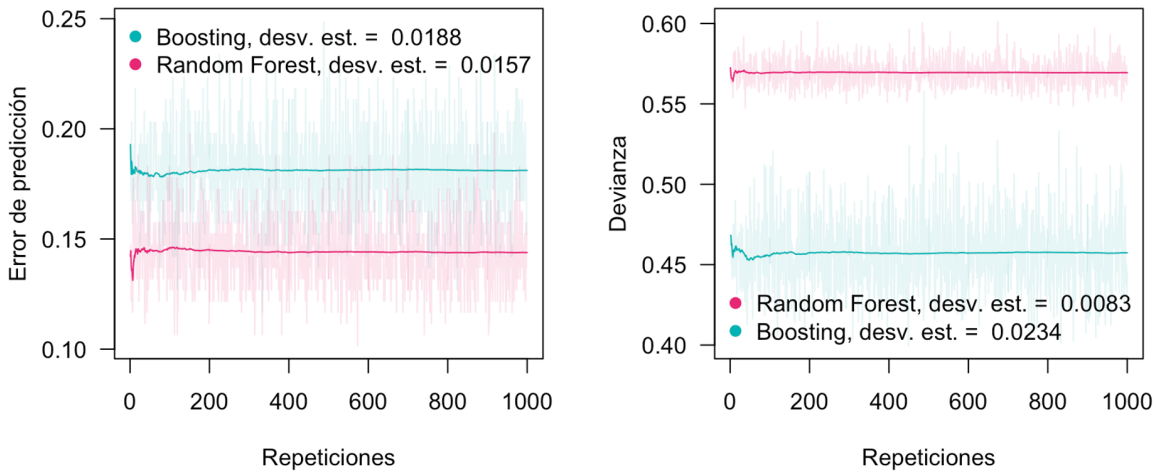


Figura 3-13: Estimaciones de las funciones de pérdida de prueba por medio de 1000 repeticiones de validación cruzada de cinco partes, donde se pretende clasificar `Race` de la base `Ancestry`. Las líneas tenues representan la estimación de cada repetición, y la sólida es el promedio. *Random Forest* es un ajuste de 1000 árboles, mientras que *boosting* tiene profundidad $J = 2$, $B = 1000$ árboles y *learning rate* $\nu = 0.01$. La desviación estándar de la regresión logística fue de 0.029 para el error y 0.353 para la devianza, ambas medidas fuera de la escala de estas gráficas.

En este trabajo, al no existir una preferencia por disminuir el error en una clase en particular, se selecciona como mejor ajuste al de *Random Forest*, considerando que tiene una devianza baja y estable por los diferentes errores de clasificación, que exhiben un clasificador relativamente eficiente. Una de sus matrices de confusión (producto de una de las 1000 repeticiones) se presenta en la tabla 3-16:

Reales	**Predicciones**				**Error (%)**
	`African`	`AfricanAmerican`	`European`	`Japanese`	
`African`	45	5	0	0	5/50 (10.00)
`AfricanAmerican`	18	26	3	0	21/47 (44.68)
`European`	0	0	50	0	0/50 (0.00)
`Japanese`	0	0	0	50	0/50 (0.00)
Totales	63	31	53	50	26/197 (13.20)

Tabla 3-16: Matriz de confusión para `Ancestry` ($n = 197$) con *Random Forest*, usando 1000 árboles. Resultados de una de las repeticiones de validación cruzada de cinco partes.

En contraste, la matriz de confusión para un ajuste de *boosting* se presenta en la tabla 3-17.

Una vez que se sabe la tasa de error en una clase, una matriz de confusión ayuda a entender cómo se distribuyeron las clasificaciones erróneas hacia las demás clases. En el caso de *Random*

Forest, se observa que la clase `AfricanAmerican` sí es principalmente calificada incorrectamente como `African`, pero también algunas de esas observaciones terminaron como `European`, pudiendo deberse a las mezclas étnicas presentes especialmente en Estados Unidos. En cambio, las observaciones de `African` fueron clasificadas como tal o como `AfricanAmerican`, no como `European`, por lo que la hipótesis de la ambigüedad étnica se mantiene.

Reales	**Predicciones**				**Error (%)**
	`African`	`AfricanAmerican`	`European`	`Japanese`	
`African`	40	10	0	0	10/50 (20.00)
`AfricanAmerican`	13	31	1	2	16/47 (34.04)
`European`	0	3	46	1	4/50 (8.00)
`Japanese`	0	0	1	49	1/50 (2.00)
Totales	53	44	48	52	31/197 (15.74)

Tabla 3-17: Matriz de confusión para `Ancestry` ($n = 197$) con *boosting*, con 100 árboles, *learning rate* de 0.1 y profundidad de 2. Resultados de una de las repeticiones de validación cruzada de cinco partes.

Por otra parte, el ajuste de *boosting* obtiene un menor error en `AfricanAmerican` al clasificar a más observaciones como tal (44, en comparación con 31 en *Random Forest*). Por ende, los errores en las demás clases aumentan, duplicándose en `African` y creciendo a 8 % en `European`.

Selección del ajuste

Se aplicaron *Random Forest* (incluyendo *bagging*) y *boosting* a través de la plataforma `h2o`. Mientras se realizaban los primeros ajustes, se observó que, con la opción `early_stopping`, el entrenamiento se detenía antes de los primeros 25 árboles. Además, como con validación cruzada no es posible monitorear el aprendizaje del algoritmo a través de las iteraciones (como en la figura 3-11 del ejemplo `MNIST`, ya que se tienen diferentes conjunto de prueba, en lugar de solo uno como en dicha gráfica), se probaron ajustes de diferentes números de árboles, con énfasis en menores o iguales a 100 ($B = \{25, 50, 75, 100, 500, 1000\}$). De esta manera, se puede estudiar el comportamiento con pocas iteraciones, así como con los números mayores que han brindado resultados satisfactorios en otros ejemplos.

Ajuste	B	J	ν	Devianza		Error de predicción (%)	
				`Train`	`Test`	`Train`	`Test`
Bagging	1000	20	-	.108	.646	0.00	18.88
Random Forest	1000	20	-	.151	.554	0.00	12.79
Boosting	1000	1	.01	.218	.441	3.05	17.16
Boosting	100	1	.10	.213	.439	3.55	17.16
Boosting	1000	2	.01	.043	.453	0.00	18.63

Tabla 3-18: Cinco de los ajustes entrenados con la base de datos `Ancestry`. B es el número de árboles, J la profundidad y ν el *learning rate*.
Los resultados `train` se calcularon con toda la base de datos ($n = 197$). Las estimaciones `test` se obtuvieron al promediar 10 repeticiones de validación cruzada de cinco partes. Sus desviaciones estándar son iguales o menores que 0.06. El valor de `mtry` en los ajustes de *Random Forest* son los predeterminados: $p = 98$ para *bagging* y $\sqrt{p} \approx 10$ para *Random Forest*.

Con esos números de árboles, se probó un ajuste de *bagging* y otro de *Random Forest* con la profundidad predeterminada de 20. En los ajustes de *boosting* se probaron diferentes profundidades ($J = \{1, 2, 4, 6, 8\}$), así como diferentes *learning rates* ($\nu = \{0.01, 0.1, 0.5, 1\}$). En esta instancia, es posible apreciar el diferente uso de los árboles que tienen estos dos algoritmos: por un lado, *Random Forest* tiene árboles profundos con el fin de reducir el sesgo, mientras que *boosting* se beneficia de árboles con pocas interacciones (producto de baja profundidad). En este ejemplo, los árboles menos profundos obtienen los mejores resultados en *boosting*.

Al tener 122 ajustes diferentes, la mejor manera de presentar sus resultados es gráficamente. En este ejercicio se repitió validación cruzada 10 veces, y aunque su promedio no es la mejor estimación del error de prueba, al tener pocas repeticiones, fue útil para seleccionar los ajustes para los que se repetiría el proceso 1000 veces.

Random Forest mantuvo una devianza baja (relativa al resto de los ajustes) a partir de las primeras iteraciones, estando alrededor de 0.55. Su error de clasificación tuvo una tendencia descendiente, particularmente dentro de las primeras 100 iteraciones, alcanzando a tener la más baja para esta base de datos, de alrededor de 13 %, en el árbol 1000. La devianza de *bagging*, aunque alta en un inicio, disminuyó hasta 0.65, mientras que el error de clasificación estuvo rondado el 19 % a través de los diferentes ajustes. Estas observaciones pueden visualizarse en la figura 3-14.

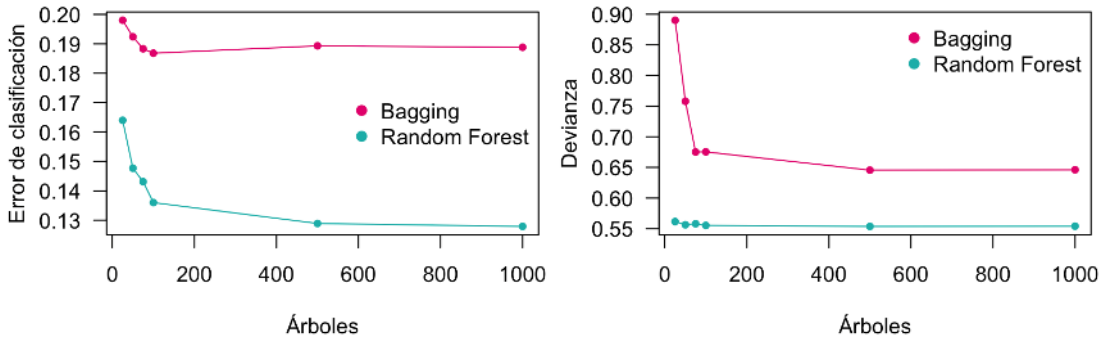


Figura 3-14: Error de clasificación (izquierda) y la devianza para los ajustes de *bagging* y *Random Forest* para `Ancestry`. Se muestra el promedio de 10 repeticiones de validación cruzada de cinco partes.

En *boosting*, la menor profundidad (i.e. $J = 1$) logró los mejores resultados. Los ajustes con *learning rate* $\nu = 0.1$ (curva roja) y número de árboles $B = 100$, y con $\nu = 0.01$ (curva azul) y $B = 1000$, fueron seleccionados para la tabla 3-18 por tener la combinación de error de clasificación bajo (en comparación con otros ajustes del mismo algoritmo) y los resultados más bajos en devianza de esta base de datos, estando alrededor de 0.4. Ambas métricas en aquella profundidad mostraron signos de sobreajuste (es decir, el error de clasificación estimado con validación cruzada comenzó a aumentar en iteraciones avanzadas) en todos los *learning rates*, excepto para el de 0.01.

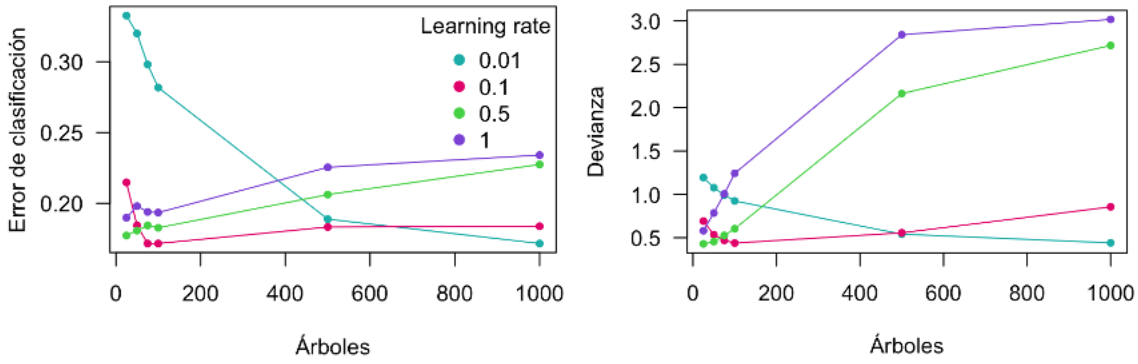


Figura 3-15: Ajustes de *boosting* con profundidad $J = 1$ para `Ancestry`. El error de clasificación y la devianza fueron calculados como promedio de 10 repeticiones de validación cruzada de cinco partes.

Es posible observar la relación inversamente proporcional que hay entre el *learning rate* ν y el número de árboles B: los ajustes mencionados en el párrafo anterior tienen resultados

muy similares. Lo mismo sucede si se comparan aquellos correspondiente a 50 y 500 árboles. Sin embargo, esto no se cumple una vez que el algoritmo tiene signos de sobreajuste, ya que tanto la devianza, como el error de clasificación, son notablemente mayores en el árbol 1000 con *learning rate* de 1, que en el árbol 100 con *learning rate* de 0.1.

En la figura 3-16 se muestra el resto de los ajustes de *boosting* entrenados. Aunque la diferencia entre los ajustes de $B = 100$ y $B = 500$ con $\nu = 0.1$ deja claro que con 500 árboles las funciones de pérdida son mayores, es razonable preguntarse si poco después de los 100 árboles se encontrarían resultados de funciones de pérdida aún menores. Similarmente, los descensos observados entre $B = 500$ y $B = 1000$, cuando $\nu = 0.01$, pueden llevar a cuestionar si posiblemente más árboles reducirían tanto el error como la devianza. La respuesta a los diferentes puntos expresados es no. Se hicieron los respectivos ajustes con 10 repeticiones para poner a prueba esas hipótesis, y no se encontró una mejora: con $\nu = 0.1$ el error se eleva después de los 100 árboles, y con $\nu = 0.01$ lo mismo sucede después de 1000.

Los ajustes de profundidad 2, 4, 6 y 8 tuvieron comportamientos similares entre sí: los errores de clasificación se mantuvieron, desde $B = 25$ hasta $B = 1000$, alrededor de 19 % para todos los *learning rates* salvo el de 0.01. Este último comienza teniendo un error más elevado, sobre 23 %, pero disminuye al nivel de los demás en $B = 500$. Finalmente, un *learning rate* de 0.5 termina con un error levemente más alto que el resto con estas tres profundidades.

Aunque los resultados pueden parecer alentadores, incluso los de aquellos sin regularización ($\nu = 1$), la historia es distinta cuando se analiza la devianza. Todos los *learning rates* mayores o iguales que 0.1 muestran signos de sobreajuste a partir de las primeras iteraciones; con $\nu = 0.01$ sucede lo mismo más tarde, al aumentar del árbol 500 al 1000.

Se ha mostrado que, para una base de datos pequeña, el riesgo de sobreajuste aparece rápidamente, incluso con un *learning rate* pequeño como $\nu = 0.1$. Al probar diferentes parámetros, se expuso por qué es importante conjugar el número de árboles y la profundidad con un *learning rate* apropiado. En este caso, $\nu = 0.01$ con 1000 árboles y profundidad $J = 2$ obtuvo resultados satisfactorios, obteniendo una menor devianza que *Random Forest*, que tuvo la tasa de error de clasificación más baja.

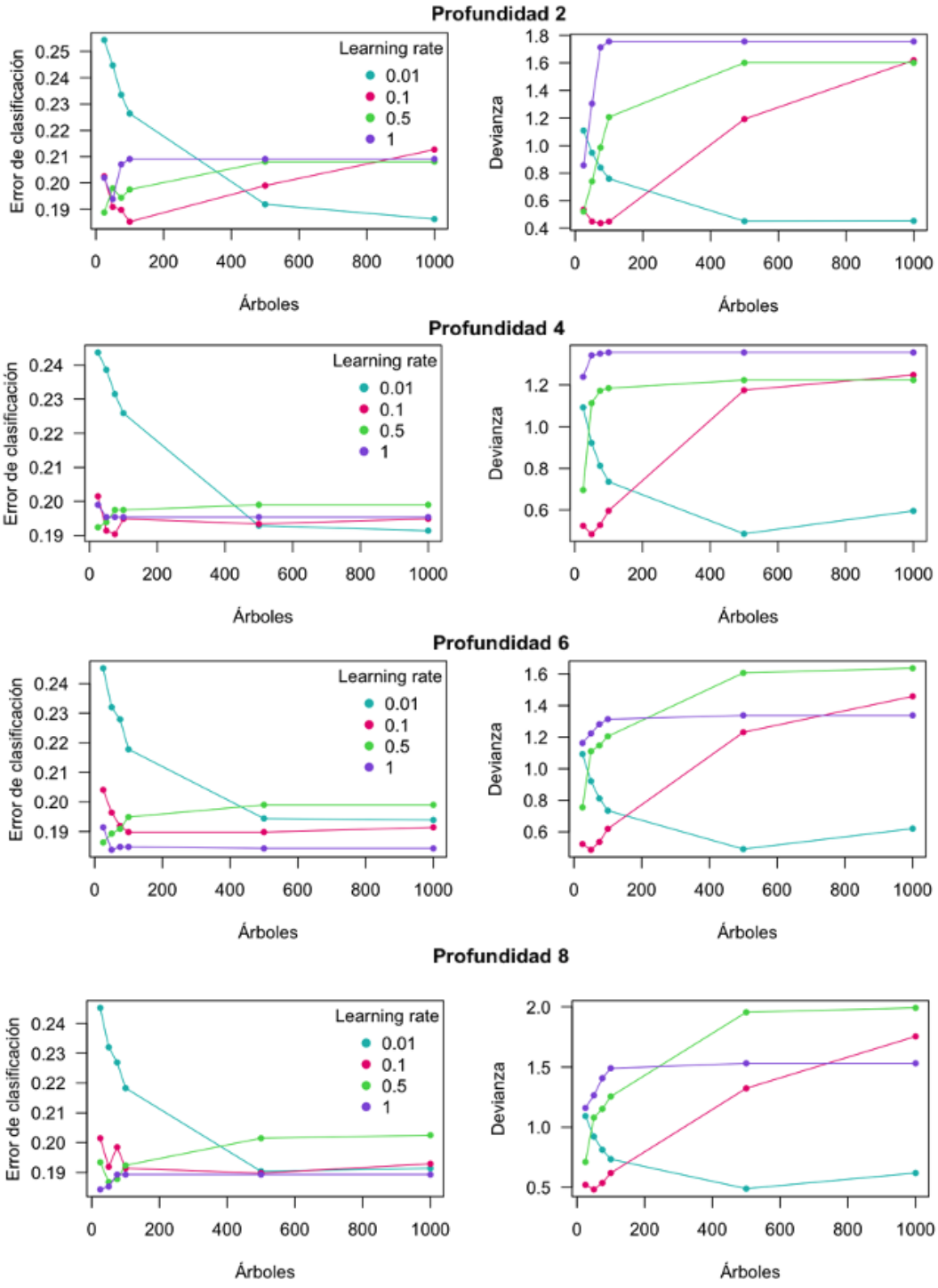


Figura 3-16: Resumen gráfico para `Ancestry` de los 96 ajustes de *boosting*, combinando seis posibilidades de árboles, cuatro de *learning rates* y cuatro de profundidad. Se muestra el promedio de 10 repeticiones de validación cruzada de cinco partes. Se observan los errores de clasificación (izquierda) y la devianza.

Importancia de variables

Adicional a la tarea de predicción que realizan *Random Forest* y *Boosting*, estos algoritmos son capaces de determinar la importancia de variables en función de la mejora en la precisión después de particionar nodos en los árboles del ajuste (sección 2.2.2). De ahora en adelante, se hará referencia al ajuste de *Random Forest* (ver tabla 3-18), ya que obtuvo el menor error de clasificación.

Una opción alternativa para conocer variables explicativas relevantes, que no necesariamente implica interés en predicción, es la V de Cramér. Es una medida de asociación entre dos variables categóricas, ubicada en $[0, 1]$, donde 1 representa una asociación fuerte; se calculará la V de Crámer de cada variable predictora con la variable respuesta `Race`. En fuentes como Sheskin (2000), sección IV del capítulo 16, apartado 12, esta medida de asociación tiene otro nombre: coeficiente de Phi de Crámer. Si se piensa en una tabla de contingencia entre las dos variables de interés, la V de Crámer se calcula usando la χ^2 de la tabla, con n como el número de observaciones en ella, y k como el valor más pequeño entre el número de columnas y el número de renglones:

$$\phi = \sqrt{\frac{\chi^2}{n(k-1)}}. \tag{3-1}$$

A continuación, se muestra una comparación de ambas medidas y posteriormente se discutirán sus diferencias.

De manera general, la V de Cramér ofrece una medida fácilmente interpretable. La V más alta de las variables predictivas de `Ancestry` con la variable respuesta es de 0.62, reflejando que, en el mejor de los casos, se alcanza una relación moderada. Es razonable suponer que estas variables no aportarán un valor predictivo muy importante, especialmente porque solo tres de ellas tienen un valor mayor que 0.5. Por otra parte, la importancia de variables de *Random Forest* es *relativa*, es decir, no ofrece una escala fija. Se puede saber cuál es la más relevante desde un punto de vista predictivo, mas se desconoce qué tan fuerte es la relación con la variable respuesta.

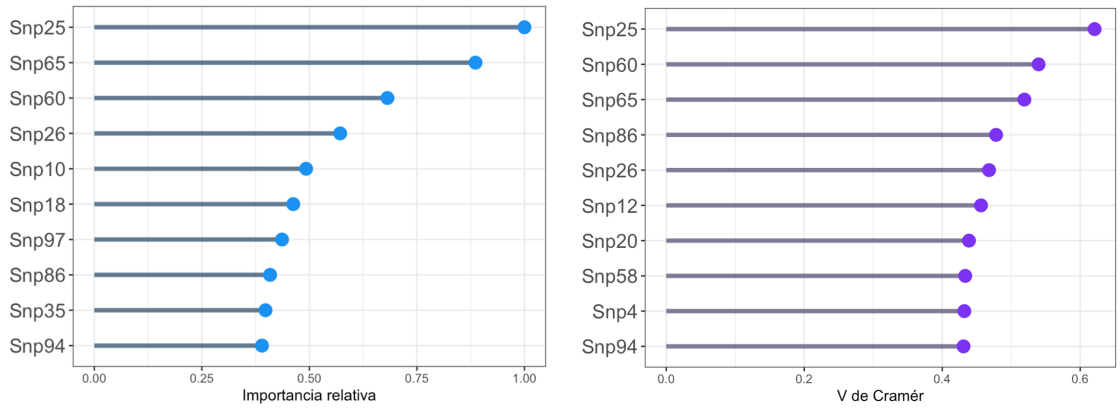


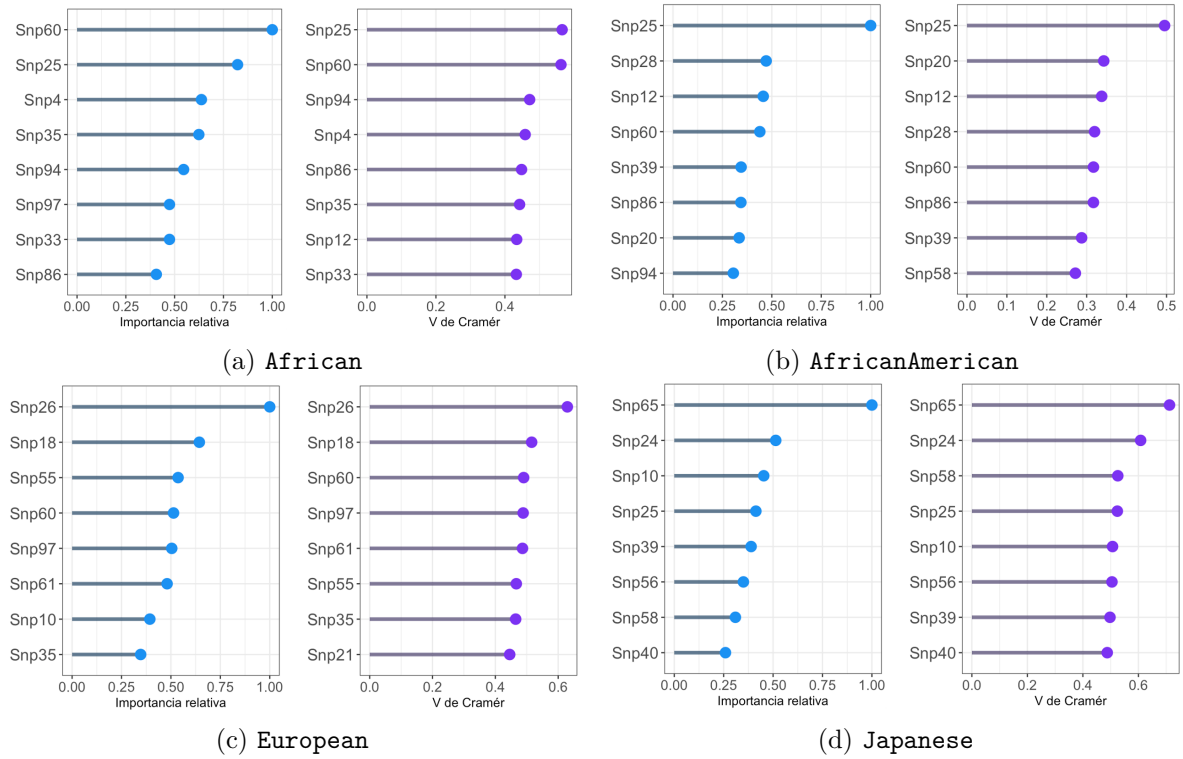
Figura 3-17: 10 valores más altos de V de Cramér (una medida de asociación entre dos variables categóricas con valores entre 0 y 1, donde valores altos representa más fuerza en la asociación), y de la importancia de variables relativa según el ajuste de *Random Forest* con 1000 árboles para la base de datos `Ancestry`. Se muestra la asociación de las variables explicativas, de dos o tres niveles cada una, con la variable `Race`, de cuatro niveles.

`Snp25` es la variable con más relevancia en ambas medidas. `Snp65` y `Snp60` y `Snp26` ocupan del segundo al cuarto lugar en importancia, respetivamente, en *Random Forest*. Aquellas variables ocupan lugares altos según la V de Cramér, aunque en un orden levemente distinto. `Snp86` y `Snp94`, entre los 10 más importantes por la primer medida, también están entre las 10 variables más fuertemente asociadas.

`Snp10` tiene una V de 0.4097 con `Race`; `Snp18` de 0.4081, `Snp97` de 0.3932 y `Snp35` de 0.4077. Estas valores son moderados, y aunque no son parte de los 10 más altos en la V de Crámer con `Race`, sí son parte de las variables más importantes en *Random Forest*. Por tanto, el cálculo de la importancia relativa en el algoritmo de predicción fue capaz de determinar variables relevantes, si bien no está totalmente en línea con lo hecho por la V de Crámer.

Teóricamente, es posible conocer la importancia de variables *por clase* (expresión 2-27 en la subsección 2.2.2), no solo de forma general. Sin embargo, ni la librería `h2o` ni `randomForest` ofrecen esta opción, por lo que se recurrió a crear variables *dummy* para cada clase de la variable `Race` (se codifica un 1 si una persona es de la clase en cuestión y 0 si no lo es; una persona tiene un 1 en solo una de las cuatro variables *dummy*). Posteriormente, se realizaron ajustes de *Random Forest* de 1000 árboles, uno para cada variable *dummy*, para extraer sus importancias de variables. De esta manera, se pueden conocer los SNPs que son más relevantes

para identificar la ascendencia de una persona. Finalmente, se hará una comparación con la V de Cramér, que es calculada con la misma variable *dummy*.



(a) `African` (b) `AfricanAmerican`

(c) `European` (d) `Japanese`

Figura 3-18: Comparación de la importancia de variables (azul) resultantes de ajustes de *Random Forest* de 1000 árboles y de V de Cramér (morado), una medida de asociación entre variables categóricas. Se comparan las SNPs con cada una de las cuatro clases de `Race`, provenientes de la base `Ancestry` que fueron codificadas como variables *dummy* para este propósito. Se muestran las ocho variables con valores más altos.

En la figura 3-18 se hacen las comparaciones entre estas dos medidas. Para `African`, las ocho variables más importantes que determinan si una persona es o no de ascendencia africana, también fueron las más fuertemente asociadas según la V de Cramér, aunque en un orden distinto.

En el caso de `AfricanAmerican`, la única variable de las ocho más importantes según *Random Forest* que no está en las ocho más fuertemente asociadas según la V de Cramér es `Snp94`.

En `European`, `Snp10` es la que aparece en las variables más importantes, pero no en las más fuertemente asociadas. Finalmente, en `Japanese` coinciden las ocho más relevantes en ambas medidas.

La fuerza de asociación de las ocho variables mostradas en cada clase es usualmente moderada, al encontrarse la mayoría entre 0.4 y 0.6. No obstante, las personas de ascendencia europea y japonesa tienen al menos una variable con una V mayor que 0.6, donde la asociación puede empezar a ser considerada fuerte. Entonces, no es sorpresa que el ajuste de *Random Forest* haya logrado clasificaciones perfectas para aquellas clases. En contraste, una proporción mayor de las variables más importantes asociadas con `African` y `AfricanAmerican` tienen una asociación muy cercana a 0.4, o incluso menores. Esto se mantiene en línea con los resultados obtenidos en los algoritmos de predicción, donde los problemas para clasificar a aquellas observaciones se hicieron presentes.

El propósito de las comparaciones anteriores es para mostrar el poder que puede tener un algoritmo como *Random Forest*. Además de conocer variables explicativas relevantes clase por clase, como similarmente lo logra la V de Cramér, también puede generar predicciones eficaces. Por ende, un investigador que desee indagar en la ascendencia de una persona a través de esta información genética, podría basarse en *Random Forest* y su importancia de variables, a pesar de no ser una herramienta estadística tradicional. La regresión logística puede hacer predicciones, pero no logró un nivel de precisión competitivo, al tener altos errores en las clases europea y japonesa (ver tabla 3-15). Sin embargo, vale la pena notar que se presentó la regresión logística como un modelo aditivo, sin interacciones o transformaciones de variables, por lo que la implementación de estas últimas o el uso de modelos regularizados pudo haber resultado en mejores predicciones.

Conclusiones

Los algoritmos dedicados a la predicción son herramientas efectivas y poderosas, que siguen la larga tradición de la disciplina estadística de contestar preguntas basándose en datos. En este sentido, deben ser aplicados siempre considerando el contexto del problema y la temporalidad de la información para obtener resultados útiles. Su gran capacidad los vuelve adiciones innegables a los modelos de regresión lineal, logística y otras técnicas formales, aunque no son un reemplazo de ellos. Es la tarea del actuario, o de quien desee realizar un análisis estadístico, conocer los alcances y limitaciones de estas herramientas con el fin de alcanzar todo su potencial.

De cada uno de los algoritmos presentados se pueden conseguir detalles para interpretar los datos: la arquitectura misma de los árboles de decisión permite entender la estructura de la base de datos, mientras que en *Random Forest* y *boosting* pueden aprovecharse herramientas útiles como la importancia de variables y la dependencia parcial.

La aplicación de estos algoritmos en computadoras personales, sin embargo, trae dificultades que los modelos clásicos de estadística no tienen, como tiempos muy extensos de cómputo, especialmente si se quieren conseguir buenas estimaciones del error de prueba como era el objetivo de este trabajo. No obstante, existen opciones en la nube, es decir, con una conexión a internet es posible trabajar con CPUs externas con un mayor poder de cómputo, como se explica en Monajemi *et al.* (2019). Las opciones más rápidas tienen un precio, pero hay otras oportunidades gratuitas.

Aunque los algoritmos aquí presentados tuvieron su origen alrededor del año 2000, siguen siendo vigentes. Búsquedas en `www.kaggle.com` y otros sitios de ciencia de datos y *machine learning* confirmarán que son de los métodos más utilizados en la predicción. Al tener la opor-

tunidad de dar una plática sobre esta tesis en el Taller Estudiantil de Cómputo, un proyecto interno por alumnos de la Facultad de Ciencias, también fue palpable el interés en este tema: se tuvo una audiencia de alrededor de 60 personas, un récord para nuestro taller, además de despertar el interés de personas externas a la comunidad.

Asimismo, en fuentes más recientes, como Efron y Hastie (2016) y Efron (2020), la experimentación e investigación con estas propuestas continúa. No es absurdo declarar que están superando la prueba del tiempo, y su popularidad se mantendrá, a pesar del crecimiento de otros algoritmos de predicción a veces más poderosos, como las redes neuronales profundas (*deep neural networks*).

Random Forest y *boosting*, además, (ya) no son "cajas negras", como suelen ser llamadas (es decir, que son un misterio el funcionamiento y el porqué de su éxito en la predicción). Cada una tiene fundamentos que explican por qué son tan efectivas: en el caso de *Random Forest*, la ley fuerte de los grandes números, así como el estudio de correlación entre árboles, son algunas de las razones detrás de su precisión. Por otro lado, *boosting* basa su éxito en el cálculo diferencial y en modelos aditivos generalizados[2], que datan de tiempo atrás.

Es posible afirmar que ambos algoritmos tienen una eficacia similar en términos de predicción. Una ventaja de *Random Forest* es que requiere poca exploración de parámetros, los mejores resultados que se pueden obtener son casi automáticos. A diferencia de otros métodos del aprendizaje estadístico, no sobreajusta, lo que le da una ventaja peculiar. *Boosting*, en contraste, requiere de más tiempo y dedicación para encontrar los ajustes con mejor rendimiento, además de que su cuidadosa selección debe evitar el sobreajuste. Al menos en dos de los tres ejemplos aquí trabajados, eso se reflejó en un error de predicción más bajo.

Por último, es interesante pensar en los mecanismos de los algoritmos en términos alegóricos. Cada árbol en *Random Forest* hace una estimación con los datos a los que tiene acceso. Estos datos solo representan una faceta del mundo real, una pequeña sección de la base de datos original, y aun así logran predicciones no despreciables. Sin embargo, si se reúnen suficientes árboles, se obtendrá un panorama más completo del fenómeno de interés, por lo que la "sabiduría

[2] *Generalized Additive Models*, presentados en 1990 por Trevor Hastie y Robert Tibshirani, son un acercamiento más flexible a métodos como regresión lineal múltiple, regresión logística y el modelo de Cox.

de las masas" llegará a mejores predicciones que un árbol individual. De forma similar, cada persona experimenta una muy pequeña parte del mundo, diferente a la de otros aunque con detalles en común. Nosotros hacemos lo mejor con la información que tenemos, y que otros tomen decisiones diferentes se explica por el conjunto de información que ellos mismos tienen; no quiere decir que estén equivocados. Si las personas trabajan y toman decisiones en equipo, usualmente se descubre que el trabajo colectivo es mejor que el individual, como en *Random Forest*.

En *boosting*, cada árbol f_b se construye para enmendar los errores de los anteriores para que, al finalizar, la suma de todos, f_B, sea la mejor predicción posible. Se puede pensar que esta tesis, o cualquier proyecto personal, funciona como un ajuste de *boosting*, puesto que las aportaciones de personas a mi alrededor mejoran mi estimación inicial f_0 (es decir, mis primeros borradores). Observaciones constantes y valiosas de mi directora de tesis, aunado a la bibliografía consultada, el apoyo de mi familia y amigos, así como los comentarios que me harán llegar mis sinodales en su momento, harán, en suma, una mejor tesis que la que yo pude haber hecho solo.

Cada parte de este trabajo fue tan desafiante como fue gratificante. Las lecturas de libros y artículos conllevaron retos distintos que los del código de `R`, a la vez que esto último fue un desafío muy diferente a presentar los algoritmos y reportar los resultados de la experimentación. Espero haber superado cada reto exitosamente.

Apéndice A

Código de R para árboles de decisión

Paquete y datos

Los ejemplos presentados en la sección 2.1 fueron hechos con la bibilioteca `rpart`, que está basada en la metodología `CART`. Asimismo, se usaron las bases de datos de `California Housing` y `Ancestry`, que se trabajan en los apéndices B y D, respectivamente. Dichas bases fueron editadas para su manejo, por lo que se recomienda consultar su procesamiento de datos en los apéndices siguientes si se desea replicar estos ejemplos.

```
library(rpart)
cali <- read.csv("cali.csv")
ances <- read.csv("ancestry.csv")
ances <- ances[,-1] #se quita columna de id
ances[,] <- lapply(ances[,], factor)
```

Árboles y otras figuras

La figura 2-1, donde se muestra un árbol de regresión podado:

```
mytree2 <- rpart(MedianHouseValue ~ ., data=cali, method="anova", cp=0.06)
rpart.plot(mytree2, compress = TRUE, type = 5, fallen.leaves = FALSE,
           box.palette = "Greens")
```

A partir de los puntos de corte de ese árbol, se hicieron las figuras 2-2 y 2-3, es decir, las gráficas donde se explicaba cómo se particiona la base y cómo funciona la minimización de las

sumas de cuadrados para elegir puntos de corte. Basado en el código de Andreas Buja, profesor de la Universidad de Pennsylvania.[1]

```
# Divisiones
spl <- 5.035
spl2 <- 3.074
sel.left  <- cali[,"MedInc"] < spl2
sel.center <- cali[,"MedInc"] >= spl2 & cali[,"MedInc"] < spl
sel.right <- cali[,"MedInc"] >= spl
plot(cali[,"MedInc"], cali[,"MedianHouseValue"],
     xlab="MedInc", ylab="MedianHouseValue", pch=16, cex=.3,
     col = ifelse(sel.left, "#11f4a8",
                  ifelse(sel.center, "#2dce98", '#2e896b')))
# Se divide en regiones:
lines(c(spl,spl), c(0,5))
lines(c(spl2,spl2), c(0,5))
# Se ajusta la media de cada región
lines(c(0,spl2),  rep(mean(cali[sel.left,"MedianHouseValue"]), 2),  lwd=2)
lines(c(spl2,spl),  rep(mean(cali[sel.center,"MedianHouseValue"]), 2),  lwd=2)
lines(c(spl,15), rep(mean(cali[sel.right,"MedianHouseValue"]), 2), lwd=2)

# Visualización de sumas de cuadrados
lstat.vals <- sort(unique(cali[,"MedInc"]))
split.vals <- (lstat.vals[-1] + lstat.vals[-length(lstat.vals)])/2 # midpoints
crit.vals <- rep(NA, length(split.vals))
for (i in 1:(length(split.vals))) {
  spl <- split.vals[i]
  sel.left  <- cali[,"MedInc"] < spl;      sel.right <- !sel.left
  crit.vals[i] <-
    sum((cali[sel.left,"MedianHouseValue"]  - mean(cali[sel.left,"
    MedianHouseValue"]))^2) +
    sum((cali[sel.right,"MedianHouseValue"] - mean(cali[sel.right,"
    MedianHouseValue"]))^2)
}
```

[1] http://www-stat.wharton.upenn.edu/~buja/STAT-961/Chapter11-Trees.R

```
plot(split.vals, crit.vals, pch=16, cex=.2, col='#00b3b3',
     xlab = "Puntos de corte de MedInc", ylab = "Sumas de cuadrados")
# El mínimo es
split.vals[crit.vals==min(crit.vals)]
```

El árbol de regresión sin podar de la figura 2-4:

```
mytree <- rpart(MedianHouseValue ~ ., data = cali, method = "anova", cp = 0,
    minbucket = 800)
rpart.plot(mytree, compress = FALSE, type = 5, fallen.leaves = FALSE,
           box.palette = "Greens")
```

El árbol de clasificación de la figura 2-5:

```
tree_clas1 <- rpart(race ~ ., data = ances, method = "class")
rpart.plot(tree_clas1, type = 5, fallen.leaves = TRUE,
           box.palette = list("#94e88f", "#f7ea85", "#85f7e2", "#f1c4f2"))
```

La gráfica del error en función de la complejidad de los árboles de decisión, enfocada en la descomposición de varianza y sesgo (figura 2-6):

```
# Varianza y sesgo
# Probaremos diferentes profundidades; entre más profundo, más complejo es el á
    rbol
n <- nrow(cali)
(min_sizes <- c(seq(from = 100, to = 10, by = -10), 5, 1))
# Tablitas para registrar los errores
error.train <- matrix(nrow = 100, ncol = length(min_sizes))
error.test <- error.train
# Seeds para reproducir resultados
set.seed(19); seeds <- sample(999999, size = 100, replace = FALSE)
# Se obtendrán 100 estimaciones de errores, los cuales se promediarán
t <- proc.time()
for (j in 1:100) {
  # Se usará repeated training-test,
  # dividiendo en 2/3 para entrenar y 1/3 para evaluar
  set.seed(seeds[j]); test <- sample(n, size = n/3, replace = FALSE)
  for (i in 1:length(min_sizes)) {
```

```
    # Se ajusta el árbol con datos de entrenamiento para predecir
    MedianHouseValue
    arbol <- rpart(MedianHouseValue ~ .,data = cali[-test, ],method ="anova",
                   minbucket = min_sizes[i], cp = 0)
    # Se calculan predicciones para esa variable...
    preds.train <- predict(arbol, newdata = cali[-test, ])
    preds.test <- predict(arbol, newdata = cali[test, ])
    # ...y son comparadas con los valores reales por medio del ECM
    error.train[j, i] <- mean((preds.train - cali[-test, 'MedianHouseValue'])^2
    )
    error.test[j, i] <- mean((preds.test - cali[test, 'MedianHouseValue'])^2)
  }
}
(proc.time() - t) #16 min. con 39 s.
# Calculamos el promedio de ambos tipos de error
media.train <- colMeans(error.train); media.test <- colMeans(error.test)
# Parámetros para graficar:
plot.min <- min(cbind(error.test, error.train)); plot.max <- max(cbind(error.
    test, error.train))
 # Gráfica
plot(media.train, type = "l", col = "#44aef4", lwd = 2.2,
     ylim = c(plot.min, plot.max), xlim = c(1.5,11.5),
     ylab = "Error", xlab = "", xaxt = "n")
lines(media.test, type = "l", col = "#e54485", lwd = 2.2)
# Después, las 100 repeticiones
for (i in 1:100) {
  lines(error.train[i,], col = "#44aef4", lwd = .06)
  lines(error.test[i,], col = "#e54485", lwd = .06)
}
# Leyendas:
text(x = c(3, 10),  par("usr")[3],
     labels = c("Complejidad baja (árboles menos profundos)",
                "Complejidad alta (árboles más profundos)"), pos = 1, xpd =
    TRUE)
legend("bottomleft",legend = c("Test", "Train"),col = c("#e54485", "#44aef4"),
```

```
48        pch=19, cex=1, bty = "n")
49 legend(x = 0.5, y = 0.6, legend = "Sesgo alto, varianza baja",
50        bty = "n")
51 legend(x = 8, y = 0.6, legend = "Sesgo bajo, varianza alta",
52        bty = "n")
```

Apéndice B

Código de `R` para `California Housing`

Los primeros ajustes que se realizaron fueron hechos con la librería `randomForest`, basado en el código original de Adele Cutler y Leo Breiman. Estos se presentan a continuación.

El único parámetro que definir para en *Random Forest* fue `mtry`, es decir, el número de variables a probar permitidas por nodo. Las candidatas a examinar fueron aquellas alrededor del valor recomendado para regresión, i.e. $\frac{p}{3} \approx 2.66$: dos y tres, además de cuatro y cinco para conocer el comportamiento del error de prueba mientras el parámetro aumenta. También se probó el resultado con ocho variables, con el fin de hacer una comparación con *bagging*. Dado que la disminución del sesgo se consigue con bosques profundos, se permitió la máxima profundidad posible, sujeto al parámetro por *default* de `nodesize`, el número mínimo de observaciones que deben tener los nodos finales. Este valor es 5, y en cada árbol construido resultó tener entre 6,750 y 7,000 nodos finales, aproximadamente. Se crearon 1000 árboles para cada opción de `mtry`.

Se realizaron cinco repeticiones con las mismas características. El mejor resultado se obtiene cuando las posibles variables a probar son 3, con un ECM de 0.0435, mientras que el error de prueba en *bagging* (`mtry` $= 8$) es 6.13 % mayor. Como observación adicional, el error de entrenamiento es muy similar con todos los parámetros, minimizándose en realidad con *bagging*.

mtry	ECM	
	Train	Test
2	0.0470	0.2317
3	0.0435	0.2287
4	0.0427	0.2327
5	0.0425	0.2363
8	0.0421	0.2427

Tabla B-1: Promedio de cinco iteraciones para el ECM de entrenamiento y prueba con muestras *OOB* de la base de datos `California Housing`. Cada una de las iteraciones, considerando el tiempo para realizar las predicciones y, con ello, el error, tomó alrededor de 40 minutos. Se usó el paquete `Random Forest`, en lugar de `h2o` como en los otros ejemplos.

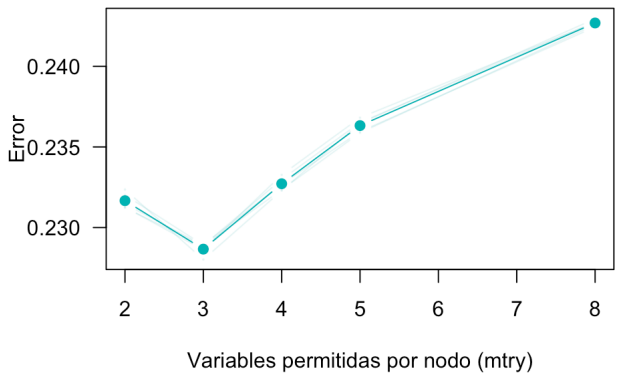


Figura B-1: Error de prueba (media de los residuos al cuadrado), resultado del promedio de cinco estimaciones con muestras OOB en ajustes de *Random Forest* con 1000 árboles cada uno de la base de datos `California Housing`. Se usó el paquete `Random Forest`, en lugar de `h2o` como en los otros ejemplos.

Es conveniente recordar que, al usar muestreo con reemplazo cada que se construyen los árboles, una sola observación es usada en aproximadamente 0.632 de ellos, cuando el número de árboles tiende a infinito. Puede conocerse el número de veces que una observación no fue usada con la función `oob.times`. A lo largo de los 25 ajustes realizados, la observación promedio fue seleccionada en el 63.21 % de los árboles, confirmando así la aproximación mencionada. Ello también implica que la estimación del error de prueba está calculada con alrededor de 368 de los 1000 árboles para cada observación.

Procesamiento de los datos

```
    cali <- read.csv("path/cal_housing.csv", sep = ",", header = FALSE)
    names(cali) <- c("Longitude", "Latitude", "HouseAge",
                     "TotalRooms", "TotalBedrooms", "Population",
                     "Households", "MedInc", "MedianHouseValue")
    cali$MedianHouseValue <- cali$MedianHouseValue/10**5
    cali$AveBedrms <- cali$TotalBedrooms/cali$Households
    cali$AveRooms <- cali$TotalRooms/cali$Households
    cali$AveOccup <- cali$Population/cali$Households
    cali <- cali[,-c("TotalBedrooms", "TotalRooms", "Households")]
    y = "MedianHouseValue"; x = setdiff(names(cali), y)
```

Ajustes con el paquete randomForest

```
library(randomForest)
# Matrices para registrar errores test (oob) y train de cinco repeticiones
rf.oob <- matrix(nrow = 5, ncol = 7)
rf.trn <- matrix(nrow = 5, ncol = 7)
colnames(rf.oob) <- c('mtry', "1o", "2o", "3o", "4o", "5o", "Media")
colnames(rf.trn) <- c('mtry', "1o", "2o", "3o", "4o", "5o", "Media")
# Se probará el parámetro mtry con 2, 3, 4, 5 y 8:
rf.oob[,1] <- c(2:5, 8)
rf.trn[,1] <- c(2:5, 8)

# Lista para almacenar los bosques
# rf <- rep(list(rep(list(0), 5)), 5)

seeds <- c(19, 431, 133, 572, 93) # para repeticiones
mtries <- c(2:5, 8) # diferentes parámetros

t1 <- proc.time()
for (i in 1:5) { # seeds: 1,
  for (j in 1:5) { # mtries
    set.seed(seeds[i])
```

```
    rf[[i]][[j]] <- randomForest(MedianHouseValue ~ .,
                                 data = cali,
                                 mtry = mtries[j],
                                 ntree = 1000,
                                 importance = F)
    rf.oob[j, i + 1] <- rf[[i]][[j]]$mse[1000]
    rf.trn[j, i + 1] <- mean((predict(rf[[i]][[j]], newdata = cali) -
    cali$MedianHouseValue)^2)
  }
}
(proc.time() - t1)

# Promedio de las cinco repeticiones
rf.oob[,ncol(rf.oob)] <- rowMeans(rf.oob[,2:6])
rf.trn[,ncol(rf.trn)] <- rowMeans(rf.trn[,2:6])

# Error de prueba a través de 1000 árboles para mtry=3
rf.scoring <- matrix(nrow = 1000, ncol = 6)
for (i in 1:5) rf.scoring[,i] <- rf[[i]][[2]]$mse
rf.scoring[,6] <- rowMeans(rf.scoring[,-6])

# Obtener el número promedio de veces que las obs están oob
oob.times = matrix(nrow = 6, ncol = 6)
for (i in 1:5) {
  for (j in 1:5) {
    oob.times[j,i] <- mean(rf[[i]][[j]]$oob.times)/1000
  }
}
oob.times[,6] = rowMeans(oob.times[,-6])
oob.times[6,] = colMeans(oob.times[-6,])
```

Random Forest (Plataforma h2o)

```
library(h2o)
h2o.init()
```

```
h2o.removeAll()
cali <- as.h2o(cali)
y <- "MedianHouseValue"; x <- setdiff(names(cali), y)

promedio = function(df){
  df[, ncol(df) + 1] = rowMeans(df[,-1])
  colnames(df)[ncol(df)] = 'Mean'
  return(df)
}
rf_train <- data.frame("mtries" = c(2:5, 8))
rf_test <- data.frame("mtries" = c(2:5, 8))
rf_oob <- data.frame("mtries" = c(2:5, 8))
rf <- rep(list(rep(list(0), 5)), 5)
seeds <- c(19, 78, 356, 444, 963) # i
mtry <- c(2:5, 8) # j

t1 = proc.time()
for (i in 0) { # 1:5
  for (j in 0) { # 1:5
    rf[[i]][[j]] <- h2o.randomForest(x = x, y = y,
                                     training_frame = cali,
                                     mtries = mtry[j],
                                     ntrees = 1000,
                                     nfolds = 5,
                                     seed = seeds[i],
                                     model_id = paste("seed", i, "_",
                                                "mtry", mtry[j], sep = ''))

    rf_test[j, i+1] <- rf[[i]][[j]]@model$cross_validation_metrics@metrics$MSE
    rf_oob[j, i+1] <- rf[[i]][[j]]@model$training_metrics@metrics$MSE
    rf_train[j, i+1] <- h2o.performance(rf[[i]][[j]], newdata = cali[-1,])
    @metrics$MSE
  }
}
(proc.time() - t1)
```

```
# tiempo: mtry=2 ~550s, ... ,mtry=8 ~980s
rf_test <- promedio(rf_test)
rf_oob <- promedio(rf_oob)
rf_train <- promedio(rf_train)
rf_final <- data.frame("mtries" = rf_train[,'mtries'], "train" = rf_train[,'
    Mean'],
                       "test" = rf_test[,'Mean'], "oob" = rf_oob[,'Mean'])
round(rf_final, 4)
#Diferencia promedio entre oob y cross-validation:
mean(rf_test[,7] - rf_oob[,7])
```

Boosting. Función de pérdida: cuadrado de los residuos

```
seeds <- c(162, 161, 399, 359, 19)
gbm_grid <- list()
gbm_fits <- list()
gbm_tables <- list()
for (i inn 1:5) gbm_tables[[i]] <- matrix(nrow = 20, ncol = 6)

hyper_params <- list(max_depth = 2:8,
                     learn_rate = seq(0.001, 0.1, 0.001))
search_criteria <- list(strategy = "RandomDiscrete",
                        max_models = 20,
                        seed = 5693,
                        stopping_rounds = 5,
                        stopping_metric = "MSE",
                        stopping_tolerance = 1e-4)
t1 = proc.time()
for (i in 5) { #1 2 3 4
  gbm_grid[[i]] <- h2o.grid("gbm", x = x, y = y,
                            training_frame = cali,
                            hyper_params = hyper_params,
                            search_criteria = search_criteria,
                            ntrees = 10000,
                            distribution = 'gaussian',
```

```
                            nfolds = 5,
                            stopping_rounds = 5,
                            stopping_tolerance = 1e-3,
                            stopping_metric = "MSE",
                            seed = seeds[i],
                            parallelism = 0)
  gbm_fits[[i]] <- lapply(gbm_grid[[i]]@model_ids, h2o.getModel)
}
(proc.time() - t1)
# extraer info: profundidad, aprendizaje, número de árboles, mse train, mse cv,
    cv sd
for (i in 5) { # 1 2 3 4
  for (j in 1:20) {
    gbm_tables[[i]][j, 1] <- gbm_fits[[i]][[j]]@allparameters$max_depth
    gbm_tables[[i]][j, 2] <- gbm_fits[[i]][[j]]@allparameters$learn_rate
    gbm_tables[[i]][j, 3] <- gbm_fits[[i]][[j]]@allparameters$ntrees
    gbm_tables[[i]][j, 4] <- gbm_fits[[i]][[j]]@model$training_metrics@metrics$
    MSE
    gbm_tables[[i]][j, 5] <- gbm_fits[[i]][[j]]@model$cross_validation_metrics_
    summary$mean[3]
    gbm_tables[[i]][j, 6] <- gbm_fits[[i]][[j]]@model$cross_validation_metrics_
    summary$sd[3]
    class(gbm_tables[[i]]) <- "numeric"
  }
}

# Ordenar para preparar la tabla resumen final
for (i in 1:5) {
  gbm_tables[[i]] <- cbind(gbm_tables[[i]][,1] + gbm_tables[[i]][,2],
                           gbm_tables[[i]])
  gbm_tables[[i]] <- gbm_tables[[i]][order(gbm_tables[[i]][,1],decreasing=TRUE)
    ,]
}
#desv est del test error
desv.est <- cbind(gbm_tables[[1]][,6], gbm_tables[[2]][,6],
```

```
                  gbm_tables[[3]][,6], gbm_tables[[4]][,6],
                  gbm_tables[[5]][,6], rep(NA, 20))
for (i in 1:20) {
  desv.est[i, 6] <- sd(desv.est[i,1:5])
}
#tabla final
gbm_final <- matrix(nrow = 20, ncol = 6)
gbm_final[,1] <- gbm_tables[[1]][,2]
gbm_final[,2] <- gbm_tables[[1]][,3]
gbm_final[,6] <- desv.est[,6]
for (i in 4:6) {
  gbm_final[,i-1] <- rowMeans(cbind(gbm_tables[[1]][,i], gbm_tables[[2]][,i],
                                    gbm_tables[[3]][,i], gbm_tables[[4]][,i],
                                    gbm_tables[[5]][,i]))
}
colnames(gbm_final) = c('J', 'v', 'n', 'trn', 'tst', 'sd tst')
```

Boosting. Función de pérdida: Huber

```
seeds <- c(652, 13, 27, 450, 199)
a <- c(0.1, 0.8, 0.3, 0.8, 0.5, 0.9, 0.9, 0.2)
J <- c(4, 8, 3, 7, 3, 6, 4, 6)
t <- c(2250, 1750, 2250, 1750, 2250, 1750, 2250, 1750)
gbm_hub_fits <- rep(list(rep(list(0), 7)), 5)

t1 <- proc.time()
for (i in 5) {
  for (i in 8) {
    gbm_hub_fits[[i]][[j]] <- h2o.gbm(x = x, y = y,
                                      training_frame = cali,
                                      nfolds = 5,
                                      ntrees = t[j],
                                      distribution = 'huber',
                                      huber_alpha = a[j],
                                      max_depth = J[j],
```

```
                                        learn_rate = 0.099,
                                        stopping_rounds = 5,
                                        stopping_tolerance = 1e-3,
                                        stopping_metric = "MSE",
                                        seed = seeds[i])
  }
}
(proc.time() - t1)
#El octavo 38, 37, 35, 37, 36.5
#El séptimo 14.3, 11.63, 10.85
#El sexto 23 min, 23.7, 22.5, 29
#El quinto 9 min, 8 min, 8.45, 9
#El cuarto 33 min, 34, 36, 37
#El tercero 21 min, 9.3, 8, 9
#El segundo 52 min, 1hr 23, 1hr 18, 1hr 19
#El primero 38 min, 13min, 16min

# Tabla resumen
hub_table <- matrix(nrow = 8, ncol = 6)
colnames(hub_table) = c('J', 'alpha', 'Arbol', 'Train', 'Test', 'Test sd')

for (j in 1:8) {
  hub_table[j, 1] <- gbm_hub_fits[[1]][[j]]@allparameters$max_depth
  hub_table[j, 2] <- gbm_hub_fits[[1]][[j]]@allparameters$huber_alpha
  hub_table[j, 3] <- gbm_hub_fits[[1]][[j]]@allparameters$ntrees
  class(hub_table) <- "numeric"
}

# Promedio de las iteraciones
hub_trn <- matrix(nrow = 8, ncol = 5) # error train
hub_tst <- matrix(nrow = 8, ncol = 5) # error test
hub_sd <- matrix(nrow = 8, ncol = 5) # desv. est. error test
for (j in 1:8) {
  for (i in 1:5) {
    hub_trn[i, j] <- gbm_hub_fits[[j]][[i]]@model$training_metrics@metrics$MSE
```

```
    hub_tst[i, j] <- gbm_hub_fits[[j]][[i]]
    @model$cross_validation_metrics_summary$mean[3]
    hub_sd[i, j] <- gbm_hub_fits[[j]][[i]]
    @model$cross_validation_metrics_summary$sd[3]
  }
}
class(hub_trn) <- class(hub_tst) <- class(hub_sd) <- "numeric"
hub_table[, 4] <- round(rowMeans(hub_trn), 4)
hub_table[, 5] <- round(rowMeans(hub_tst), 4)
hub_table[, 6] <- round(rowMeans(hub_sd), 4)
```

Repeticiones de ajustes selectos y regresión lineal

```
#### Regresión lineal ####
library(glmnet)
set.seed(23234); seeds_split1 <- sample(5000000, size = 10000)
set.seed(912); seeds_fit1 <- sample(5000000, size = 1000)
glm_mse <- c()
glm_mse_means  <- c()
n <- nrow(california)

for (i in 1:10000) {
  set.seed(seeds_split1[i])
  train <- sample(n, size = n*2/3, replace = FALSE)
  x_trn <- as.matrix(california[train, -c(6,4)]) # se elimina respuesta y
    variable population
  y_trn <- as.matrix(california[train, 6])
  x_tst <- as.matrix(california[-train, -c(6,4)])
  y_tst <- as.matrix(california[-train, 6])
  fit <- glmnet(x_trn, y_trn, family = "gaussian", lambda = 0)
  preds <- predict(fit, newx = x_tst)
  glm_mse[i] <- mean((preds - y_tst)^2)
  glm_mse_means[i] <- mean(glm_mse[1:i])
}
# error aparente
```

```r
x_mat <- as.matrix(california[, -c(6,4)])
y_mat <- as.matrix(california[, 6])
fit <- glmnet(x_mat, y_mat, family = "gaussian", lambda = 0)
preds <- predict(fit, newx = x_mat)
glm_trn <- mean((preds - y_mat)^2)

#### Random forest ####
rf_mse <- c()
rf_oob <- c()
rf_mse_means <- c()
rf_oob_means <- c()

t <- proc.time()
for (i in 1:100) {
  datos <- h2o.splitFrame(cali, ratios = 2/3, seed = seeds_split1[i])
  rf <- h2o.randomForest(x = x, y = y,
                         training_frame = datos[[1]],
                         validation_frame = datos[[2]],
                         mtries = 3,
                         ntrees = 1000,
                         seed = seeds_fit1[i])
  rf_mse[i] <- h2o.mse(rf, valid = TRUE)
  rf_oob[i] <- h2o.mse(rf, train = TRUE)
  rf_mse_means[i] <- mean(rf_mse[1:i])
}
(proc.time() - t)
# hizo 10 en 33 min; 5 en 7.8
rf_trn  <- h2o.randomForest(x = x, y = y,
                                training_frame = cali,
                                mtries = 3,
                                ntrees = 1000,
                                seed = seeds_fit1[i])
rf_trn <- h2o.performance(rf_trn, cali)

#### Boosting con residuos al cuadrado como pérdida ####
```

```
gbm_mse <- c()
gbm_mse_means <- c()
t <- proc.time()
for (i in 100) {
  datos <- h2o.splitFrame(cali, ratios = 2/3, seed = seeds_split1[i])

  gbm <- h2o.gbm(x = x, y = y,
                 training_frame = datos[[1]],
                 validation_frame = datos[[2]],
                 max_depth = 7,
                 ntrees = 2219,
                 learn_rate = 0.059,
                 seed = seeds_fit1[i])
  gbm_mse[i] <- h2o.mse(gbm, valid = TRUE)
  gbm_mse_means[i] <- mean(gbm_mse[1:i])
}
(proc.time() - t)
gbm_trn <- h2o.gbm(x = x, y = y,
                   training_frame = cali,
                   max_depth = 7,
                   ntrees = 2219,
                   learn_rate = 0.059,
                   seed = seeds_fit1[i])
gbm_trn <- h2o.performance(gbm_trn, cali)

#### Boosting con Huber como función de pérdida ####
hub_mse <- c()
hub_mse_means <- c()

t <- proc.time()
for (i in 1:100) {
  datos <- h2o.splitFrame(cali, ratios = 2/3, seed = seeds_split1[i])
  hub <- h2o.gbm(x = x, y = y,
                 training_frame = datos[[1]],
                 validation_frame = datos[[2]],
```

```
                  distribution = "huber",
                  huber_alpha = 0.9,
                  max_depth = 6,
                  ntrees = 1750,
                  learn_rate = 0.099,
                  seed = seeds_fit1[i])
  hub_mse[i] <- h2o.mse(hub, valid = TRUE)
  hub_mse_means[i] <- mean(hub_mse[1:i])
}
(proc.time() - t)
hub_trn <- h2o.gbm(x = x, y = y,
                   training_frame = cali,
                   distribution = "huber",
                   huber_alpha = 0.9,
                   max_depth = 6,
                   ntrees = 1750,
                   learn_rate = 0.099,
                   seed = seeds_fit1[i])
hub_trn <- h2o.performance(hub_trn, cali)
```

Gráficas

La figura 3-1, donde se muestran las gráficas con los resultados de las repeticiones de la estimación del error de prueba de los ajustes seleccionados, así como su promedio:

```
# Algoritmos de árboles
min_plot = min(rf_mse, gbm_mse, hub_mse)
max_plot = max(rf_mse, gbm_mse, hub_mse)
plot(rf_mse, type = 'l', col = '#e6007322', ylim = c(min_plot, max_plot),
     xlab = "Repeticiones", ylab = "Error")
lines(rf_mse_means, col = "#e60073")
lines(gbm_mse, col = "#00b3b322")
lines(gbm_mse_means, col = "#00b3b3")
lines(hub_mse, col = "#ad51c122")
lines(hub_mse_means, col = "#ad51c1")
```

```
legend("left", legend = c('Random Forest', 'Boosting CR', 'Boosting Huber'),
       pch=19, cex=1, bty = "n", col = c("#e60073", "#00b3b3", "#ad51c1"))
legend("right", title = "Desv. est.",  pch=19, cex=1, bty = "n",
       legend = c(round(sd(rf_mse), 4), round(sd(gbm_mse), 4),
                  round(sd(hub_mse), 4)),
       col = c("#e60073", "#00b3b3", "#ad51c1"))

# Regresión lineal
plot(glm_mse[1:i], type = 'l', col = '#76e04522', ylim = c(0.5,0.7),
     xlab = "Repeticiones", ylab = "Error")
lines(glm_mse_means[1:i], col = "#49b716")
legend("topright", pch=19, cex=1, bty = "n", col = c("#76e04522", "#76e04522"),
       legend = c("Regresión lineal",
                  paste("Desv. est.=", round(sd(glm_mse), 4))))
```

La figura 3-9, donde se exhiben las predicciones de *Random Forest* y *boosting* por medio de validación cruzada, necesita de los ajustes de `h2o` con los parámetros deseados, entrenado con validación cruzada.

```
# Los modelos a usar se llaman gbm_hub_fits_pred y rf_pred
gbm_preds <- h2o.cross_validation_predictions(gbm_hub_fits_pred)
predicciones <- matrix(nrow = dim(cali)[1], ncol = length(gbm_preds) + 1)
for (i in 1:length(gbm_preds)) {
  predicciones[,i] <- as.vector(gbm_preds[[i]])
}
for (i in 1:dim(cali)[1]) {
  predicciones[i,6] <- sum(predicciones[i,1:5])
}

rf_preds <- h2o.cross_validation_predictions(rf_pred)
predicciones_rf <- matrix(nrow = dim(cali)[1], ncol = length(rf_preds) + 1)
for (i in 1:length(rf_preds)) {
  predicciones_rf[,i] <- as.vector(rf_preds[[i]])
}
for (i in 1:dim(cali)[1]) {
  predicciones_rf[i,6] <- sum(predicciones_rf[i,1:5])
```

```
}
plot(predicciones_rf[,6], type = 'l')

comparacion <- cbind('original' = as.vector(cali[y]),
                          'boosting' = predicciones[,6],
                          'forest' = predicciones_rf[,6])

comparacion <- comparacion[order(comparacion[,'original']), ]

plot(comparacion[,2], type = 'p', col = '#0097fc', cex = 0.007, pch = '.',
     xlab = '', ylab = 'Mediana del valor de las casas/10000')
points(comparacion[,3], col = '#ff8d1a', cex = 0.04)
lines(comparacion[,1], col = '#424a4a', lwd = 1.6)
legend("topleft", legend = c('Boosting', 'Random Forest'),
       pch=19, cex=1, bty = "n", col=c('#0097fc', '#ff8d1a'))
```

Las figuras 3-2, 3-3 y 3-4, relativas a la importancia de variables y la dependencia parcial:

```
# Importancia de variables
h2o.varimp_plot(hub_fits1[[1]])
# Dependencia parcial de variable explicativa:
h2o.partialPlot(hub_fits1[[1]], data = cali, cols = "MedInc")
# ...para dos variables explicativas en conjunto:
library(plot3Drgl)
h2o.partialPlot(hub_fits1[[1]], data = cali,
                col_pairs_2dpdp = list(c("Longitude", "Latitude")))
```

La figura 3-5, donde se comparan las estimaciones de error *OOB* y de validación cruzada de *boosting* con cuadrado de los residuos como función de pérdida:

```
matplot(mtry, cbind(rf_test[,2:6], rf_oob[,2:6]),
        col = c(rep("#00b3b322", 5), rep("#e6007322", 5)),
        pch = 1, type="b", ylab = "Error",
        xlab = "Variables permitas por nodo (mtry)", lty = 1, cex = 1)
matlines(mtry, cbind(rf_test[,7], rf_oob[,7]), col = c("#00b3b3", "#e60073"),
         pch=19, type="b", ylab = "ECM", lty = 1)
legend("topleft", legend=c("Validación cruzada", "Out-of-bag"), pch=19,
```

```
       cex=1,col=c("#00b3b3","#e60073"), bty = "n")
```

La figura 3-6, donde se comparan resultados `train` contra `test` de *boosting* con cuadrado de los residuos como función de pérdida:

```
matplot(1:20, gbm_final[,5:4],
        col = c("#00b3b3", "#e60073"),
        pch=19, type="b", ylab = "Error", xlab = "",
        lty = 1, cex = 1, xaxt = "n")
legend("bottomright", legend = c("Prueba", "Entrenamiento"),
       col = c("#00b3b3", "#e60073"), pch=19, cex=1, bty = "n",)
```

La figura 3-7, donde se hace *zoom in* en el error `test` de la gráfica anterior:

```
cols = c("#530994", "#1751cc", "#2989e1", "#2996cc", "#2bd9e7", "#1fe3ca", "#
    adede5")
matplot(1:20, gbm_final[,5],
        col = "grey", pch=1, type="b", lty = 3, cex = .1, xaxt = "n",
        ylab = "Error", xlab = "")
for (i in 8:2) {
  matlines(c(which(gbm_final[,1]==i)),
           gbm_final[which(gbm_final[,1]==i),5], col = cols[i-1])
  matpoints(c(which(gbm_final[,1]==i)),
            gbm_final[which(gbm_final[,1]==i),5], col = cols[i-1], pch = 19)
}
matpoints(4, gbm_final[4,5], col="#1fe3ca", pch = 15, cex = 1.4)
legend("topleft", title = "Profundidad",
       legend=c(8:2), pch=19, cex=1, bty = "n",
       col=rev(cols))
```

La figura B-1, donde se observa el error a través de diferentes parámetros `mtry` con el paquete `randomForest`:

```
matplot(mtries, rf.oob[,7], col = "#00b3b3",
        pch=19, type="b", lty = 1, ylim = c(0.228, 0.243),
        xlab = "Variables permitidas por nodo (mtry)", ylab = "Error")
matlines(mtries, rf.oob[,-7], col = "#00b3b322",
         pch=1, type="b", lty = 1, cex = .1)
```

Apéndice C

Código de R para MNIST

```
library(h2o)
setwd("/Users/luiscarlos/Documents/Tesis/Ejemplos/MNIST")
load("~/Documents/Tesis/Ejemplos/MNIST/mnist.RData")
h2o.init(min_mem_size="7G")
```

Prepración de los datos

```
mnist <- h2o.importFile("path/mnist_train.csv", sep = ",", header = TRUE)
mnist <- h2o.splitFrame(mnist, ratios = 0.80, seed = 19)
train <- mnist[[1]]; valid <- mnist[[2]]
test <- h2o.importFile("path/mnist_test.csv", sep = ",", header = TRUE)

# Tabla de la presencia de los dígitos en los diferentes conjuntos
digitos <- rbind(round(table(as.vector(train["label"]))/dim(train)[1]*100, 2),
                 round(table(as.vector(valid["label"]))/dim(valid)[1]*100, 2),
                 round(table(as.vector(test["label"]))/dim(test)[1]*100, 2))

# La variable respuesta es categórica
train["label"] <- as.factor(train["label"])
valid["label"] <- as.factor(valid["label"])
test["label"] <- as.factor(test["label"])
y <- "label"; x <- setdiff(names(train), y)
```

Ajustes

```
t <- proc.time()
rf1 <- h2o.randomForest(x = x, y = y,
                        training_frame = train,
                        validation_frame = valid,
                        mtries = -1,
                        ntrees = 1000,
                        seed = 36,
                        model_id = "rf1")
(proc.time() - t) # 1 hora con 45 minutos

# por default, learn_rate es 0.1 en h2o.gbm
t <- proc.time()
gbm1 <- h2o.gbm(x = x, y = y,
                training_frame = train,
                validation_frame = valid,
                distribution = "multinomial",
                max_depth = 5,
                ntrees = 1000,
                seed = 63,
                model_id = "gbm1")
(proc.time() - t) #11537.70 3 horas 12 minutos

gbm_depths <- list(max_depth = c(2, 3, 4))

t <- proc.time()
gbm_grid1 <- h2o.grid("gbm", x = x, y = y,
                      training_frame = train,
                      validation_frame = valid,
                      hyper_params = gbm_depths,
                      distribution = "multinomial",
                      ntrees = 1000,
                      score_each_iteration = FALSE,
                      seed = 834,
```

```
                      grid_id = "gbm_grid1",
                      parallelism = 0)
(proc.time() - t) # 5 horas con 37 minutos
gbm_fits1 <- lapply(gbm_grid1@model_ids, h2o.getModel)

gbm_depths <- list(max_depth = c(6, 7))

t <- proc.time()
gbm_grid2 <- h2o.grid("gbm", x = x, y = y,
                      training_frame = train,
                      validation_frame = valid,
                      hyper_params = gbm_depths,
                      distribution = "multinomial",
                      ntrees = 1000,
                      score_each_iteration = FALSE,
                      seed = 84,
                      grid_id = "gbm_grid2",
                      parallelism = 0)
(proc.time() - t) 7 horas 50 minutos
gbm_fits2 <- lapply(gbm_grid2@model_ids, h2o.getModel)

# Lo mismo pero con submuestreo:
gbm_depths <- list(max_depth = c(2, 3, 4))

t <- proc.time() #12:20
gbm_grid1_sub <- h2o.grid("gbm", x = x, y = y,
                          training_frame = train,
                          validation_frame = valid,
                          hyper_params = gbm_depths,
                          distribution = "multinomial",
                          ntrees = 1000,
                          sample_rate = 0.25,
                          score_each_iteration = FALSE,
                          seed = 834,
                          grid_id = "gbm_grid1_sub",
```

```
                          parallelism = 0)
(proc.time() - t) #3 horas con 16 minutos
gbm_fits1_sub <- lapply(gbm_grid1_sub@model_ids, h2o.getModel)

gbm_depths <- list(max_depth = c(6, 7))
t <- proc.time()
gbm_grid2_sub <- h2o.grid("gbm", x = x, y = y,
                      training_frame = train,
                      validation_frame = valid,
                      hyper_params = gbm_depths,
                      distribution = "multinomial",
                      ntrees = 1000,
                      sample_rate = 0.25,
                      score_each_iteration = FALSE,
                      seed = 84,
                      grid_id = "gbm_grid2",
                      parallelism = 0)
(proc.time() - t) # 3 horas con 33 minutos
gbm_fits2_sub <- lapply(gbm_grid2_sub@model_ids, h2o.getModel)

t <- proc.time()
gbm1.1 <- h2o.gbm(x = x, y = y,
                  training_frame = train,
                  validation_frame = valid,
                  distribution = "multinomial",
                  max_depth = 5,
                  ntrees = 1000,
                  sample_rate = 0.25,
                  seed = 63,
                  model_id = "gbm1.1")
(proc.time() - t)

# Regresión logística sin regularizar
t1 <- proc.time()
reg_log3 <- h2o.glm(x = x, y = y,
```

```
                    training_frame = train,
                    validation_frame = valid,
                    family = 'multinomial',
                    lambda = 0,
                    max_iterations = 1000)
(proc.time() - t1) # un minuto

extract_info <- function(fit){
  metrics <- c()
  # Test performance:
  metrics[1] <- fit@model$validation_metrics@metrics$logloss
  metrics[2] <- fit@model$validation_metrics@metrics$cm$table[11,11]
  # Train performance:
  metrics[3] <- fit@model$training_metrics@metrics$logloss
  metrics[4] <- fit@model$training_metrics@metrics$cm$table[11,11]
  return(metrics)
}

# Resumen de ajustes:
all_fits <- list(rf1, gbm_fits1[[3]], gbm_fits1[[2]], gbm_fits1[[1]], gbm1,
                      #J=2            J= 3            J=4             J=5
                 gbm_fits2[[2]], gbm_fits2[[1]], gbm_fits1_sub[[3]],
                 #J=6            J=7             J=2 sub
                 gbm_fits1_sub[[2]], gbm_fits1_sub[[1]],
                 #J=3 sub            J=4 sub
                 gbm1.1, gbm_fits2_sub[[1]], gbm_fits2_sub[[2]])
                 #J=5 sub J=6 sub            J=7 sub

all_fits_summ <- lapply(all_fits, extract_info)
all_fits_summ <- as.data.frame(do.call(rbind, all_fits_summ))
colnames(all_fits_summ) <-c('trn_err', 'tst_err', 'trn_logloss', 'tst_logloss')
```

Estimación de errores *test*

```
# Elección del ajuste con profundidad 6
```

```
t <- proc.time()
gbm3 <- h2o.gbm(x = x, y = y,
                training_frame = train,
                validation_frame = valid,
                distribution = "multinomial",
                ntrees = 700,
                max_depth = 6,
                score_each_iteration = FALSE,
                seed = 84,
                model_id = "gbm3")
(proc.time() - t) # 2 horas con 27 minutos

test_perf <- h2o.performance(gbm3, newdata = test)
test_cm <- test_perf@metrics$cm

# Repetición de ajustes:
#### Boosting ####
gbm_table <- matrix(nrow = 1000, ncol = 4)
colnames(gbm_table) <- c('test_logloss', 'test_error', 'train_logloss', '
    train_error')
gbm_means <- gbm_table

for (k in 1:30) {
  datos <- h2o.splitFrame(mnist, ratios = 2/3, seed = seeds_split[i])
  gbm <- h2o.gbm(x = x, y = y,
                 training_frame = datos[[1]],
                 validation_frame = datos[[2]],
                 distribution = "multinomial",
                 ntrees = 700,
                 max_depth = 6,
                 seed = seeds_fit[i])
  gbm_table[i,] <- extract_info(gbm)
  if (i==1) {
    gbm_means[i,] <- gbm_table[1,]
  }else{
```

```
    gbm_means[i,] <- colMeans(gbm_table[1:i,])
  }
}

#### Random Forest ####
rf_table <- matrix(nrow = 1000, ncol = 4)
colnames(rf_table) <- c('test_logloss', 'test_error', 'train_logloss', '
    train_error')
rf_means <- rf_table

set.seed(1989)
seeds_split2 <- sample(100000, 1000, replace = FALSE)
set.seed(7)
seeds_fit2 <- sample(100000, 1000, replace = FALSE)

t <- proc.time()
for (k in in 30) { #20:00
  datos <- h2o.splitFrame(mnist, ratios = 2/3, seed = seeds_split2[i])
  rf <- h2o.randomForest(x = x, y = y,
                training_frame = datos[[1]],
                validation_frame = datos[[2]],
                mtries = -1,
                ntrees = 1000,
                seed = seeds_fit2[i])

  rf_table[i,] <- extract_info(rf)
  if (i==1) {
    rf_means[i,] <- rf_table[1,]
  }else{
    rf_means[i,] <- colMeans(rf_table[1:i,])
  }
}
(proc.time() - t)

#### Regresión logística ####
```

```
set.seed(83749)
seeds_split3 <- sample(100000, 1000, replace = FALSE)

rl_table <- matrix(nrow = 1000, ncol = 4)
colnames(rl_table) <- c('test_logloss', 'test_error', 'train_logloss', '
    train_error')
rl_means <- rl_table

t <- proc.time()
for (k in 1:100) {
  datos <- h2o.splitFrame(mnist, ratios = 2/3, seed = seeds_split3[i])
  reg_log <- h2o.glm(x = x, y = y,
                     training_frame = datos[[1]],
                     validation_frame = datos[[2]],
                     family = 'multinomial',
                     lambda = 0,
                     lambda_search TRUE,
                     max_iterations = 1000)

  rl_table[i,] <- extract_info(reg_log)
  if (i==1) {
    rl_means[i,] <- rl_table[1,]
  }else{
    rl_means[i,] <- colMeans(rl_table[1:i,])
  }
}#un minuto por fit
(proc.time() - t)

#### Desviaciones estándar de los ajustes elegidos ####
tablas <- list(rf_table, gbm_table, rl_table[1:30,], rl_table[,])
desv_est <- function(list){
  n <- length(list)
  desviaciones <- data.frame('Ajuste' = rep(NA, n),
                             'Error' = rep(NA, n),
                             'Devianza' = rep(NA, n))
```

```
  desviaciones[,1] <- c('Rf', 'Gbm', 'RegLog30', 'RegLog100')
  for (i in 1:n) {
    for (j in 1:2) {
      desv_est <- sd(list[[i]][,j], na.rm = TRUE)
      desviaciones[i, j + 1] <- round(desv_est, 4)
    }
  }
  return(desviaciones)
}
desv_est(tablas)
```

Gráficas

La figura 3-11, donde se comparan el error y la devianza de ajustes de *boosting* con diferentes profundidades y submuestreos. Se muestra el código de devianza, la gráfica del error es análoga, pero con `validation_classificacion_error` en lugar de `validation_logloss`:

```
plot(gbm1@model$scoring_history$validation_logloss[1:985],
     type = "l", col = "#e60073", lwd = 1.5, ylim = c(0.09, 0.24),
     ylab = "Devianza", xlab = "Número de árboles") # depth 5
lines(gbm1.1@model$scoring_history$validation_logloss[1:985],
      col = "#e60073", lwd = 0.7)
lines(gbm_fits2[[2]]@model$scoring_history$validation_logloss[1:985],
      col = "#00b3b3", lwd = 1.5) # 6
# se agregan los ajustes deseados, análogamente, con la función lines()
legend(x = 740, y = 0.238, title = "Profundidad", legend = c("5", "6", "7"),
       pch=19, cex = 0.9, col=c("#e60073", "#00b3b3", "#9063b8"), bty = 'n')
legend(x=400, y=0.238, title="Submuestreo", legend=c("1","1/4"),
       lty=1, lwd=c(2,0.8), cex=0.9, col=c("#8c9299","#8c9299"), bty='n')
```

La figura 3-12, donde se comparan los errores `train` y `test` de un ajuste de *boosting*:

```
plot(gbm_fits2[[2]]@model$scoring_history$validation_classification_error,
     type = "l", ylim = c(0, 0.04), col = "#00b3b3",
     ylab = "Error de predicción", xlab = "Número de árboles")
lines(gbm_fits2[[2]]@model$scoring_history$training_classification_error,
```

```
      col = "#e60073")
legend(x = 650, y = 0.04, legend = c("Prueba", "Entrenamiento"),
       pch=19, cex=1, bty = "n", col=c("#00b3b3", "#e60073"))
```

Apéndice D

Código de R para Ancestry

La base de datos se encuentra en la página oficial de Efron (2016):

https://web.stanford.edu/~hastie/CASI/data.html

```
library(h2o) #ajustes
library(rcompanion) # V de Cramér
library(ggplot2) # gráficas de importancia de variables y V de Cramér
```

Preparación de los datos

```
datos <- read.csv("ancestry.csv", header = TRUE)
datos <- datos[-1] # quitar columna de ID
datos[which(datos$race == "African American"),1] <- "AfricanAmerican"
datos[,] <- lapply(datos[,], factor) # hacer cada variable factor

# Una tabla que indica el número de NA's por clase
faltantes <- matrix(nrow = 4, ncol = 2)
for (i in 1:4) {
  faltantes[i, 1] <- levels(datos$race)[i]
  indices <- which(datos$race == levels(datos$race)[i])
  faltantes[i, 2] <- sum(is.na(datos[indices,]), na.rm = TRUE)
}

# Se imputará con la moda de la columna correspondiente,
```

```r
# solo considerando observaciones de la misma clase de "race".
# Para ello, se crea función moda:
getmode <- function(v) { # imputación
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
} # fuente: https://www.tutorialspoint.com/r/r_mean_median_mode.htm
for (i in 1:nrow(datos_falt)) {
  clase <- datos[datos_falt[i,1],1] # "race" del dato faltante
  filas <- which(datos[,1] == clase) # observaciones de "race" correspondientes
  col <- datos_falt[i,2] # columna donde está el dato faltante
  moda <- getmode(datos[filas, col])
  datos[datos_falt[i,1], col] <- moda # reemplazo
}
table(is.na(datos)) # comprobar que no hay NA's

# Agregar 10 columnas indicadoras de cada fold para cross-validation
for (i in 1:10) datos[paste("Fold", i, sep="")] <- NA
for (i in 1:4) { # para cada clase de "race"...
  filas <- which(datos$race == levels(datos$race)[i]) #...sus respectivas filas
  n_filas <- length(filas)
  set.seed(1998)
  for (j in 102:111) { #ubicar en las columnas fold:
    datos[filas, j] <- sample(rep(1:5, times = 10), n_filas, replace = FALSE)
  }
}
```

Ajustes

```r
setwd("~/Documents/Tesis/Ejemplos/Ancestry")
h2o.init(min_mem_size = "7G")

ancestry <- as.h2o(datos)
y <- "race"
folds <- colnames(ancestry)[102:111] # las columnas indicadoras para cross-
    validation
```

```r
x <- colnames(ancestry)[2:101]
p <- length(x) -2 # 2 columnas son constantes

# Primeros 98 ajustes para probar parámetros
rf_grids <- rf_fits <- list()
seeds <- c(38, 1989, 2021, 19, 123, 5, 22, 4444, 9876, 510)

# Random forest (mtries = -1) y bagging (mtries = p) variando el número de á
    rboles
hyper_params <- list(ntrees = c(25, 50, 75, 100, 500, 1000),
                     mtries = c(-1, p))
t1 <- proc.time()
for (i in in 1:10) {
  rf_grids[[i]] <- h2o.grid("randomForest", x = x, y = y,
                            training_frame = ancestry,
                            hyper_params = hyper_params,
                            fold_column = folds[i],
                            seed = seeds[i],
                            grid_id = paste("rf_grid_", i, sep = ""),
                            parallelism = 0)
  rf_fits[[i]] <- lapply(rf_grids[[i]]@model_ids, h2o.getModel)
}
(proc.time() - t1) # Cada ajuste de grid tarda alrededor de 2 minutos

# En los ajustes de boosting se probará la hipótesis del bajo número de
# árboles, observando el efecto que tiene la profundidad y el learning rate
hyper_params <- list(max_depth = c(1, 2, 4, 6, 8),
                     learn_rate = c(0.01, 0.1, 0.5, 1),
                     ntrees = c(25, 50, 75, 100, 500, 1000))

gbm_grids <- gbm_fits <- list()
t1 <- proc.time()
for (i in in 10) {
  gbm_grids[[i]] <- h2o.grid("gbm", x = x, y = y,
                             training_frame = ancestry,
```

```
                              distribution = "multinomial",
                              hyper_params = hyper_params,
                              fold_column = folds[i],
                              seed = seeds[i],
                              grid_id = paste("gbm_grid_", i, sep = ""),
                              parallelism = 0)
  gbm_fits[[i]] <- lapply(gbm_grids[[i]]@model_ids, h2o.getModel)
}
(proc.time() - t1) # cada iteración tarda alrededor de 24 minutos

# Función para métricas principales
extract_info <- function(list_objects){
  tabla <- matrix(nrow = length(list_objects), ncol = 9)
  for (i in 1:length(list_objects)){
    fit <- list_objects[[i]]
    cm <- fit@model$cross_validation_metrics@metrics$cm$table

    # Paramétros:
    tabla[i, 1] <- fit@allparameters$max_depth
    if(fit@allparameters$max_depth < 20) { #si es boosting
      tabla[i, 2] <- fit@allparameters$learn_rate
    }else{ # si es random forest
      tabla[i, 2] <- fit@allparameters$mtries
    }
    tabla[i, 3] <- fit@allparameters$ntrees

    # General performance:
    tabla[i, 4] <- fit@model$cross_validation_metrics@metrics$logloss
    tabla[i, 5] <- cm$Error[5]

    # Performance by class:
    for (j in in 1:4) {
      tabla[i, j+5] <- cm$Error[j]
    }
  }
```

```
  colnames(tabla) <- c("J", "v/m", "B", "logloss", "error",
                       "error Af", "error AA", "error Eu", "error Jp")
  # Se ordena la tabla para facilitar cálculos posteriores
  tabla <- tabla[order(tabla[, 1], tabla[, 2], tabla[, 3]),]
  return(tabla)
}
# Función para promediar las 10 repeticiones:
avg_table <- function(list_tables){
  tabla_sd <- matrix(nrow = nrow(list_tables[[1]]), ncol = 6)
  colnames(tabla_sd) <- c("sd logloss", "sd error", "sd Af",
                          "sd AA", "sd Eu", "sd Jp")
  tabla_final <- cbind(list_tables[[1]], tabla_sd)
  valores_dummy <- c()

  # Como se recibirá una lista de 10 tablas, cada una con información de varios
     ajustes, se tiene que promediar las entradas que estén en la misma posición
     en cada tabla.
  for (i in in 1:nrow(tabla_final)) { # para cada renglón
    for (j in in 4:ncol(list_tables[[1]])) { # para cad columna
      for (k in in 1:length(list_tables)) { # para cada tabla
        valores_dummy[k] <- list_tables[[k]][i, j] # se obtienen los 10 valores
        tabla_final[i, j] <- round(mean(valores_dummy), 4) # se promedian
        tabla_final[i, j+6] <- round(sd(valores_dummy), 4)
      }
    }
  }
  if (tabla_final[1,1] > 10) { # si es random forest se ignora la columna de
    profundidad
    tabla_final <- tabla_final[, -1]
  }
  return(tabla_final)
}
rf_summaries <- lapply(rf_fits, extract_info)
rf_table <- avg_table(rf_summaries)
gbm_summaries <- lapply(gbm_fits, extract_info)
```

```
gbm_table <- avg_table(gbm_summaries)
```

Los ajustes adicionales y las 1000 repeticiones

```
# El siguiente ajuste fue para verificar que el ajuste elegido de boosting (J=2
    , learn_rate = 0.01) no mejoraba después de los 1000 árboles.
hyper_params <- list(ntrees = c(1000, 1500, 2000))
gbm_grid_J2_v0.01 <- gbm_fits_J2_v0.01 <- list()
t1 <- proc.time()
for (i in 1:10) {
  gbm_grid_J2_v0.01[[i]] <- h2o.grid('gbm', x = x, y = y,
                                     training_frame = ancestry,
                                     distribution = 'multinomial',
                                     hyper_params = hyper_params,
                                     max_depth = 1,
                                     learn_rate = 0.01,
                                     fold_column = folds[i],
                                     seed = seeds[i],
                                     parallelism = 0)
  gbm_fits_J2_v0.01[[i]] <- lapply(gbm_grid_J2_v0.01[[i]]@model_ids,
                                     h2o.getModel)
}
(proc.time() - t1) # 20 minutos en total

gbm_J2_v0.01_summaries <- lapply(gbm_fits_J2_v0.01, extract_info)
gbm_J2_v0.01_table <- avg_table(gbm_J2_v0.01_summaries)

# El siguiente fue para comprobar que el ajuste de J=2, learn_rate=0.01 no
    mejoraba entre los 100 y 500 árboles:
hyper_params <- list(ntrees = c(100, 200, 300, 400, 500))
gbm_grid_J2_v0.1 <- gbm_fits_J2_v0.1 <- list()

t1 <- proc.time()
for (i in 1:10) {
  gbm_grid_J2_v0.1[[i]] <- h2o.grid('gbm', x = x, y = y,
```

```
                                    training_frame = ancestry,
                                    distribution = 'multinomial',
                                    hyper_params = hyper_params,
                                    max_depth = 1,
                                    learn_rate = 0.1,
                                    fold_column = folds[i],
                                    seed = seeds[i],
                                    grid_id = paste('gbm_grid_J2_v0.1_', i, sep
    = ''),
                                    parallelism = 0)
  gbm_fits_J2_v0.1[[i]] <- lapply(gbm_grid_J2_v0.1[[i]]@model_ids, h2o.getModel
    )
}
(proc.time() - t1) # 20 minutos en total

gbm_J2_v0.1_summaries <- lapply(gbm_fits_J2_v0.1, extract_info)
gbm_J2_v0.1_table <- avg_table(gbm_J2_v0.1_summaries)

# Los ajustes que se reptieron 1000 veces
# Se registrará la devianza y el error general y por clase, después se hará el
    promedio.
# Random Forest
rf_tabla <- matrix(ncol = 6, nrow = 1000)
colnames(rf_tabla) <- c('logloss', 'error', 'error_Af',
                        'error_AA', 'error_Eu', 'error_Jp')
set.seed(19)
seeds <- sample(100000, size = 1000, replace = FALSE)

extract_info <- function(fit){
  metrics <- c()
    cm <- fit@model$cross_validation_metrics@metrics$cm$table
    # General performance:
    metrics[1] <- fit@model$cross_validation_metrics@metrics$logloss
    metrics[2] <- cm$Error[5]
    # Performance by class:
```

```
    for (j in 1:4) {
      metrics[j+2] <- cm$Error[j]
    }
  return(metrics)
}

t1 <- proc.time()
for (i in 1:1000) {
  rf <- h2o.randomForest(x = x, y = y,
                         training_frame = ancestry,
                         nfolds = 5,
                         ntrees = 1000,
                         seed = seeds[i])
  rf_tabla[i,] <- extract_info(rf)
  print(i)
}
(proc.time() - t1) # cada grupo 50 ajustes tardó 55 minutos aprox.

rf_means <- rf_tabla
for (i in 2:1000) {
  rf_means[i,] <- colMeans(rf_tabla[1:i,])
}
# Train
rf_train <- h2o.randomForest(x = x, y = y,
                             training_frame = ancestry,
                             ntrees = 1000,
                             seed = seeds[1])
rf_train <- h2o.performance(rf_train, ancestry)
# Boosting
gbm_tabla <- matrix(ncol = 6, nrow = 1000)
colnames(gbm_tabla) <- c('logloss', 'error', 'error_Af',
                         'error_AA', 'error_Eu', 'error_Jp')

t1 <- proc.time()
for (i in 1:1000) {
```

```
  gbm <- h2o.gbm(x = x, y = y,
                 training_frame = ancestry,
                 nfolds = 5,
                 ntrees = 1000,
                 learn_rate = 0.01,
                 max_depth = 1,
                 seed = seeds[i])
  gbm_tabla[i,] <- extract_info(gbm)
  print(i)
}
(proc.time() - t1) # cada grupo 50 ajustes tardó 45 minutos aprox.

gbm_means <- gbm_tabla
for (i in 2:1000) {
  gbm_means[i,] <- colMeans(gbm_tabla[1:i,])
}
# Regresión logística
rl_tabla <- matrix(ncol = 6, nrow = 1000)
colnames(rl_tabla) <- c('logloss', 'error', 'error_Af',
                        'error_AA', 'error_Eu', 'error_Jp')
rl_means <- rl_tabla
t1 <- proc.time()
for (i in in 1:1000) {
  reg_log <- h2o.glm(x = x, y = y,
                     training_frame = ancestry,
                     family = 'multinomial',
                     nfolds = 5,
                     lambda = 0,
                     max_iterations = 1000,
                     seed = seeds[i])
  rl_tabla[i,] <- extract_info(reg_log)
  if (i==1) {
    rl_means[i,] <- rl_tabla[1,]
  }else{
    rl_means[i,] <- colMeans(rl_tabla[1:i,])
```

```
  }
}
(proc.time() - t1)
# Desviaciones estándar
# devianza
sd(gbm_tabla[,1]); sd(rf_tabla[,1])
# error
sd(gbm_tabla[,2]); sd(rf_tabla[,2])
```

Comparación con V de Crámer

```
# Función que extrae, promedia y grafica la importancia de variables de los
    grids y sus repeticiones:
mean_varimp <- function(fit = 'boosting', ntrees = 1000, mtries = -1,
                        learn_rate = 0.1, max_depth = 2, n_var = 1:10,
                        objeto = 'grafica'){
  # en una tabla se guardará la importancia de cada variable en cada repetición
  tabla_imp_resumen <- as.data.frame(matrix(nrow = 98, ncol = 12))
  for (i in 1:10) { # para cada repetición
    if (fit == 'rf') {
      indice_m <- which(rf_grids[[i]]@summary_table[,'mtries'] == mtries)
      indice_B <- which(rf_grids[[i]]@summary_table[,'ntrees'] == ntrees)
      indice <- intersect(indice_m, indice_B)
      tabla_imp <- as.data.frame(h2o.varimp(rf_fits[[i]][[indice]])) #se extrae
    la importancia de variables. el indice indica en qué lugar del grid está el
    ajuste con los parámetros seleccionados
    }else{
      indice_v <- which(gbm_grids[[i]]@summary_table[,'learn_rate'] ==
    learn_rate)
      indice_J <- which(gbm_grids[[i]]@summary_table[,'max_depth'] == max_depth
    )
      indice_B <- which(gbm_grids[[i]]@summary_table[,'ntrees'] == ntrees)
      indice <- intersect(intersect(indice_v, indice_J), indice_B)
      tabla_imp <- as.data.frame(h2o.varimp(gbm_fits[[i]][[indice]]))
    }
```

```
20
21    tabla_imp <- tabla_imp[order(tabla_imp[,'variable']),] # se ordenan las
      variables por orden alafabético
22
23    if (is.na(tabla_imp_resumen[1,1]) == TRUE) {
24      tabla_imp_resumen[,1] <- tabla_imp[,'variable'] # si no se ha hecho, se
      pone el nombre de la variable en la primera columna
25    } # cada columna tendrá la importancia de la repetición respectiva
26    tabla_imp_resumen[,i+2] <- tabla_imp[,'scaled_importance']
27  }
28  # Se promedian las 10 repeticiones y se deja una tabla solo con el nombre de
      la variable y la importancia relativa promedio
29  tabla_imp_resumen[,2] <- rowMeans(tabla_imp_resumen[,3:12])
30  colnames(tabla_imp_resumen) <- c('variable', 'imp_promedio')
31  tabla_imp_resumen <- tabla_imp_resumen[,1:2]
32  tabla_imp_resumen <- tabla_imp_resumen[order(tabla_imp_resumen['imp_promedio'
      ],
33                                               decreasing = TRUE),]
34  tabla_imp_resumen[,2] <- round(tabla_imp_resumen[,2], 4)
35  orden <- tabla_imp_resumen[rev(n_var), 1]
36
37  # Si se desea graficar:
38  # Gráficas inspiradas en https://stackoverflow.com/questions/56304698/how-do-
      i-plot-the-variable-importance-of-my-trained-rpart-decision-tree-model
39  if (objeto == 'grafica') {
40    ggplot(tabla_imp_resumen[n_var,]) +
41      geom_segment(aes(x = variable, y = 0,
42                       xend = variable, yend = imp_promedio),
43                   size = 1.5, alpha = 0.7, col = '#0d3b60') +
44      geom_point(aes(x = variable, y = imp_promedio),
45                 size = 4, show.legend = F, col = '#1a91f2') +
46      ggtitle(ifelse(fit == 'rf', 'Importancia de variables para Random Forest'
      ,
47                     'Importancia de variables para boosting')) +
48      xlab('') + ylab('Importancia relativa') +
```

```
49        coord_flip() +
50        scale_x_discrete(limits = orden) +
51        theme_bw() +
52        theme(plot.title = element_text(hjust = 0.5),
53              axis.text.y = element_text(size = 14))
54    }else{ # si se desea ver la tabla:
55      return(tabla_imp_resumen)
56    }
57  }
58
59  # V de Cramer
60  # En una tabla se almacenarán las V general y de cada clase, una por columna.
      La primera columna es el nombre de la variable
61  tabla_v <- as.data.frame(matrix(nrow = 100, ncol = 2 + length(levels(datos$race
      ))))
62  tabla_v[, 1] <- colnames(datos)[2:101]
63  colnames(tabla_v) = c('index', levels(datos$race), "General")
64
65  # Se hará el cálculo de la V de Cramer una variable a la vez
66  for (j in 1:100) {
67    for (i in 1:(length(levels(datos$race)) + 1)) { # y un nivel (general y por
      clase de raza) a la vez
68
69      # se hacen tablas de contingencia entre raza (y sus niveles) y la variable
      respuesta
70      if (i == 5) { # tabla general
71        t <- table(datos[,1], datos[,j+1])
72      }else{ # tablas por clase
73        t <- table(datos[,111 + i], datos[,j+1])
74      }
75
76      if (ncol(t) == 1) { # si es una variable constante, la V se hace 0 automá
      ticamente
77        tabla_v[j, i + 1] <- 0
78      }else{ #se hacce el cálculo
```

```
      tabla_v[j, i + 1] <- round(unname(cramerV(t)), 4)
    }
  }
}

# Función para la gráfica
grafica_Vcramer <- function(nivel = 'General', n_vars = 1:10){ #n_vars son las
    variables a graficar

  n_col <- which(colnames(tabla_v) == nivel) #columna de la clase de raza
    deseado
  tabla <- tabla_v[order(tabla_v[, n_col], decreasing = TRUE), c(1, n_col)] #
    ordena las V's de mayor a menor
  orden <- tabla[rev(n_vars), 1] # orden para graficar

  ggplot(tabla[n_vars,]) +
    geom_segment(aes(x = index, y = 0,
                     xend = index, yend = eval(parse(text=nivel))),
                 size = 1.5, alpha = 0.7, col = '#46396b') +
    geom_point(aes(x = index, y = eval(parse(text=nivel))),
               size = 4, show.legend = F, col = '#7b1af2') +
    # ggtitle(nivel) +
    xlab('') + ylab('V de Cramér') +
    coord_flip() +
    scale_x_discrete(limits = orden) +
    theme_bw() +
    theme(plot.title = element_text(hjust = 0.5),
          axis.text = element_text(size=12),
          axis.text.y = element_text(size = 14),
          axis.text.x = element_text(size = 12))
}

grafica_Vcramer(nivel = 'General', n_vars = 1:10)
grafica_Vcramer(nivel = 'African', n_vars = 1:8)
grafica_Vcramer(nivel = 'AfricanAmerican', n_vars = 1:8)
```

```
grafica_Vcramer(nivel = 'European', n_vars = 1:8)
grafica_Vcramer(nivel = 'Japanese', n_vars = 1:8)
```

Gráficas

La figura 3-13, donde se visualizan el avance de las 1000 repeticiones:

```
# Devianza
plot(gbm_tabla[,1], type = "l", col = "#00b3b322",
     ylim = c(min(gbm_tabla[,1]), max(rf_tabla[,1])),
     xlab = "Repeticiones", ylab = "Devianza")
lines(gbm_means[,1], type = "l", col = "#00b3b3")
lines(rf_tabla[,1], type = "l", col = "#e6007322")
lines(rf_means[,1], type = "l", col = "#e60073")
legend("bottomleft", pch=19, cex=1, bty = "n", col = c("#e60073", "#00b3b3"),
       legend = c(paste("Random Forest, desv. est. = ", round(sd(rf_tabla[,1]),
    4)),
                  paste("Boosting, desv. est. = ", round(sd(gbm_tabla[,1]),4)))
    )
# de forma análoga para el error, pero con la columna 2 de las tablas, e. g.
    rf_means[,2]
```

Las figuras 3-14, 3-15 y 3-16, donde se visualizan las funciones de pérdida para los diferentes ajustes, se hicieron con una función:

```
# Función para crear gráficas resumen de los 98 ajustes:
graficas <- function(J = 2, metric = "error", fit = "Boosting",
                     leg_pos = "topright"){
  # metric es "error" o "logloss"
  colores <- c("#00b3b3", "#e60073", "#47d147", "#8654dd")

  if (fit == "Boosting") {
    indices_J <- which(gbm_table[,"J"] == J)
    indices_v <- indices_Jv <- list()
    rates <- c(0.01, 0.1, 0.5, 1)
    # para ajustar limites del eje y:
```

```
    minimo <- min(gbm_table[indices_J, metric])
    maximo <- max(gbm_table[indices_J, metric])

    for (i in 1:4) {
      indices_v[[i]] <- which(gbm_table[,"v/m"] == rates[i])
      indices_Jv[[i]] <- intersect(indices_J, indices_v[[i]])
    }

    plot(x = gbm_table[indices_Jv[[1]], "B"], y = gbm_table[indices_Jv[[1]],
   metric], col = colores[1], pch = 20, type = "o", xlab = "  rboles  ", ylab =
   metric, main = paste(fit, "con profundidad", J, sep = " "), ylim = c(minimo,
    maximo))

    for (i in 2:4) {
      lines(x = gbm_table[indices_Jv[[i]], "B"], y = gbm_table[indices_Jv[[i]],
    metric], col = colores[i], pch = 20, type = "o")
    }

    legend(leg_pos, legend = c("v=0.01", "v=0.1", "v=0.5", "v=1"), pch=19, cex=
   1, bty = "n", col = colores)

  }else{ # random forest
    indices_m <- which(rf_table[,"v/m"] == -1)
    minimo <- min(rf_table[, metric])
    maximo <- max(rf_table[, metric])

    plot(x = rf_table[indices_m, "B"], y = rf_table[indices_m, metric],
         col = colores[1], pch = 20, type = "o", xlab = "  rboles  ", ylim = c(
        minimo, maximo), ylab = metric)
    lines(x = rf_table[-indices_m, "B"], y = rf_table[-indices_m, metric], col
   = colores[2], pch = 20, type = "o")

    legend(leg_pos, legend = c("Bagging", "Random Forest"), pch = 19, cex = 1,
   bty = "n", col = colores[c(2,1)])
  }
```

```
}
graficas(J = 2, metric = "logloss", leg_pos = "right")
graficas(J = 2, metric = "error") # J = 1, 2, 4, 6, 8
```

Las figuras 3-17 y 3-18 se hicieron como se indica en la subsección de la comparación con V de Crámer del presente apéndice.

Bibliografía

[1] Bates, S., Hastie, T. y Tibshirani, R. (2021). *Cross-validation: what does it estimate and how well does it do it?* Por ser publicado próximamente.

[2] Breiman, L. (1996). *Bagging predictors*, Machine Learning 26: 123–140.

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

[4] Breiman, L. (2001a). *Random forests*, Machine Learning 45: 5–32.

[5] Breiman, L. (2001b). *Statistical Modeling: The Two Cultures*, Statistical Science 16: 199–231.

[6] Breiman, L., Friedman, J., Olshen, R. y Stone, C. (1984). *Classification and Regression Trees*. Estados Unidos: Wadsworth.

[7] Buja, Andreas. (2019). *STATISTICS 961, Fall Semester 2019*. `http://www-stat.wharton.upenn.edu/~buja/STAT-961/`.

[8] Click, C., Malohlava, M., Parmar, V. y Candel, A. (2021). *Gradient Boosting Machine with H20*. `http://h2o.ai/resources/`.

[9] Efron, B. (2010). *Large-scale inference*. Estaods Unidos: Cambridge University Press.

[10] Efron, B. (2020). *Prediction, Estimation, and Attribution*, Journal of the American Statistical Association, 115:530, 636-655.

[11] Efron, B. y Hastie, T. (2016). *Computer Age Statiscal Inference*. 1ra edición. Estados Unidos: Institute of Mathematical Statistics Monographs.

[12] Friedman, J. (1999). *Stochastic gradient boosting*, Technical report, Stanford University.

[13] Friedman, J. (2001). *Greedy function approximation: A gradient boosting machine*, Annals of Statistics 29(5): 1189–1232.

[14] Friedman, J., Hastie, T. y Tibshirani, R. (2000). *Additive Logistic Regression: A statistical view of boosting*, Annals of Statistics 28: 337-407.

[15] Hastie, T., Tibshirani, R. y Friedman, J. (2009). *The Elements of Statistical Learning*. 2da edición. Nueva York: Springer.

[16] James, G., Witten, D., Hastie, T. y Tibshirani, R. (2017). *An Introduction To Statistical Learning With Applications In R*. 1ra edición. Nueva York: Springer.

[17] Monajemi, H., Murri, R., Jonas, E., Liang, P., Stodden, V. y Donoho, D. (2019). Ambitious Data Science Can Be Painless. Harvard Data Science Review, 1(1).

[18] Nykodym, T., Kraljevic T., Hussami, N., Rao, A. y Wang, A. (2021). *Generalized Linear Modeling with H2O*. `http://h2o.ai/resources/`.

[19] Sheskin, D. (2000). *Handbook of Parametric and Nonparametric Statistical Procedures*. 2da edición. Estados Unidos: Chapman & Hall/CRC.

[20] Vázquez Alamilla, J., Naranjo Albarrán, L., Fuentes García, R. y Chávez Cano, M. (2019). *Inferencia estadística para estudiantes de ciencias*. 1era edición. Ciudad de México: Las Prensas de Ciencias.

Software

[21] Friedman, J., Hastie, T. y Tibshirani, R. (2010). *Regularization Paths for Generalized Linear Models via Coordinate Descent*. Journal of Statistical Software, 33(1), 1-22. `https://www.jstatsoft.org/v33/i01/`.

[22] LeDell, E., Gill, N., Aiello S., Fu, A., Candel A., Cliff Click C., Kraljevic T., Nykodym T., Aboyoun P., Kurka M. y Malohlava M. (2021). *h2o: R Interface for the 'H2O' Scalable Machine Learning Platform*. R package version 3.32.0.5. `https://github.com/h2oai/h2o-3`

[23] Liaw, A. y Wiener, M. (2002). *Classification and Regression by randomForest*. R News 2(3), 18–22.

[24] R Core Team (2021). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing, Vienna, Austria. `https://www.R-project.org/`

[25] RStudio Team (2020). *RStudio: Integrated Development Environment for R*. RStudio, PBC, Boston, MA. `http://www.rstudio.com/`

[26] Soetaert, K. (2016). *plot3Drgl: Plotting Multi-Dimensional Data - Using 'rgl'*. R package version 1.0.1. `https://CRAN.R-project.org/package=plot3Drgl`

[27] Mangiafico, S. (2021). *rcompanion: Functions to Support Extension Education Program Evaluation*. R package version 2.4.0. `https://CRAN.R-project.org/package=rcompanion`

[28] Therneau, T. y Atkinson, B. (2019). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-15

[29] Wickham, H. (2016). *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York.