UNIVERSITAT D'ALACANT
Facultat de Ciències
Facultad de Ciencias

FACULTAD DE CIENCIAS

GRADO EN MATEMÁTICAS

TRABAJO FIN DE GRADO

CURSO ACADÉMICO 2023-2024

TÍTULO:

MACHINE LEARNING AVANZADO CON ALGORITMOS HÍBRIDOS

AUTOR:

MANUEL GARCÍA PLAZA

Resumen

Este trabajo realiza un recorrido teórico por diversos algoritmos de Machine Learning supervisado abarcando métodos paramétricos, como la Regresión, y no paramétricos, entre los que se encuentran los métodos basados en árboles de decisión, que engloban a los propios CART y las familias de bagging y boosting, con el objetivo de presentar los modernos algoritmos que combinan ambos mundos.

En la primera parte se hace énfasis en el aspecto matemático de los algoritmos. Para empezar, se explica cómo se hallan los coeficientes óptimos en las técnicas de Regresión Lineal y Logística. Seguidamente, se analiza el proceso de entrenamiento de un árbol de decisión CART en profundidad, sentando las bases para los modelos sucesivos. A continuación se esbozan las ideas detrás de Random Forest y se detallan los procesos de optimización que rigen el funcionamiento de Gradient Boosting y XGBoost.

La segunda parte del trabajo es en la que se introducen los algoritmos híbridos. Primeramente: Linear Tree, que es un CART que ajusta modelos de Regresión en sus nodos. En segundo lugar, se propone Linear Random Forest: un Random Forest compuesto de Linear Trees. Le sigue Regression-Enhanced Random Forest, que es un Random Forest entrenado sobre los errores que comete una Regresión Lineal o Logística enriqueciendo las predicciones de esta. Después aparece Explainable Boosted Regression, que implementa en los modelos de Regresión relaciones no lineales basadas en errores de estimación. Por último, se explica Piecewise Linear Gradient Boosting, que aporta al poderoso paradigma del boosting la capacidad de extrapolación además de optimizaciones de software.

La sección final se dedica a realizar una comparativa entre todos los algoritmos tratados teóricamente durante el trabajo para comprobar si estos novedosos métodos que unen árboles de decisión y sus derivados con Regresiones son realmente eficaces. Los modelos se evalúan en términos de precisión y de tiempo de ejecución a lo largo de seis casos de uso ejemplificantes. Además, se sugieren posibles trabajos futuros que investigar.

Palabras clave: Machine Learning, Linear Tree, Regression-Enhanced Random Forest, Explainable Boosted Regression, Piecewise Linear Gradient Boosting.

Abstract

This research carries out a theoretical overview of different supervised Machine Learning algorithms including parametric methods, such as Regression, and non-parametrics ones, among which are decision trees-based methods, which involve CARTs themselves and the bagging and boosting families, with the purpose of presenting the state-of-the-art algorithms that combine both worlds.

The first part focuses on the mathematical aspect of the algorithms. To begin with, it is explained how the optimal coefficients are calculated in Linear and Logistic Regression techniques. Then, the CART's training process is analysed in depth, laying the foundations for the successive models. Afterwards, the ideas behind Random Forest are sketched and the optimization processes that drive the functioning of Gradient Boosting and XGBoost are detailed.

The second part of the work introduces hybrid algorithms. First of all: Linear Tree, that is a CART fitting Regression models at its nodes. In second place, it is proposed Linear Random Forest: a Random Forest made up of Linear Trees. It is followed by Regression-Enhanced Random Forest, that is a Random Forest trained over the errors made by a Linear or Logistic Regression, enriching its predictions. Then it comes Explainable Boosted Regression, which implements non-linear relationships based on estimation errors in Regression models. Lastly, Piecewise Linear Gradient Boosting is explained, providing extrapolation capability as well as software optimizations to the powerful boosting approach.

The final section is intended to make a comparison between all of the algorithms discussed theoretically during the work to prove whether these new methods that join decision trees and their derived methods with Regressions are actually effective. Models are evaluated in terms of accuracy and running time along six exemplary use cases. In addition, possible future work to be investigated is suggested.

Key words: Machine Learning, Linear Tree, Regression-Enhanced Random Forest, Explainable Boosted Regression, Piecewise Linear Gradient Boosting.

Índice

1. Introducción

La forma de aprendizaje más primitiva que el ser humano aplica en su vida es el método de prueba y error. Esta estrategia se basa en identificar situaciones y patrones observados en el pasado y actuar en consecuencia de la manera más beneficiosa según el conocimiento previo. Hoy en día, vivimos plenamente digitalizados y cualquier sitio que visitemos, cualquier cosa que digamos, cada clic en nuestro ordenador o móvil queda registrado, obteniéndose así una cantidad ingente de valiosa información que utilizar ya sea en favor propio o de otras personas.

Durante el siglo XIX, en plena Revolución Industrial, se desarrollan los primeros estudios descriptivos para encontrar relaciones entre variables y los métodos estadísticos donde nace el concepto de recta de regresión, entre otros muchos hallazgos. No obstante, no es hasta el siglo siguiente que aparecen las primeras soluciones a problemas de clasificación, incluida la Regresión Logística [11]. A finales del siglo XX e inicio del XXI nacen los métodos basados en árboles de decisión, desde los CART originales hasta los más vanguardistas. En el transcurso de estos centenarios, los humanos han ido refinando y evolucionando la metodología de la prueba y error ayudándose de la estadística hasta llegar al punto de desarrollar el *Machine Learning* explotando la capacidad de cálculo de los ordenadores para aprender de datos pasados y realizar predicciones de manera mucho más eficiente que las personas.

Este trabajo se estructura en dos secciones principales. A lo largo de la sección 2, se estudian teóricamente los algoritmos fundamentales: la Regresión Lineal y Logística y los árboles de decisión junto a algunos de sus derivados. Posteriormente, en la sección 3, se exploran los enfoques más recientes. Estos relacionan los árboles de decisión con la Regresión y son: *Linear Tree*, *Linear Random Forest*, *Regression-Enhanced Random Forest*, *Explainable Boosted Regression* y *Piecewise Linear Gradient Boosting*.

Paralelamente a esta memoria se ha desarrollado un proyecto disponible en GitHub (`https://github.com/mgp165/TFG`) donde se ponen a prueba todos estos algoritmos en varios casos de uso con el fin comprobar si existe una diferencia consistente entre los algoritmos iniciales y los híbridos en la práctica.

1.1. Evaluación y selección de modelos

Durante todo el trabajo consideraremos un conjunto de datos conformado por n individuos (filas) y $p + 1$ variables (columnas). Llamamos $X_1, X_2, \ldots, X_p$ a las variables

independientes o explicativas; y a la variable dependiente u objetivo; x_i al conjunto de valores de las variables independientes para el individuo i; y_i al valor de la variable dependiente del individuo i y x_{ij} al valor de la variable $j \in \{1, \ldots, p\}$ sobre el individuo $i \in \{1, \ldots, n\}$.

Las métricas elegidas para evaluar el rendimiento de los algoritmos que se ajustan en los datos de los casos de uso son, según el tipo de problema que se aborde:

- **Problemas de regresión:** la medida escogida en este caso es la raíz cuadrada del error cuadrático medio (RMSE, por sus siglas en inglés). Se calcula así:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \ ,$$

  donde $\hat{y}_i$ es el valor predicho por el modelo para la observación i.

  La expresión de esta métrica es esencialmente la misma que la de la distancia euclídea, tan solo difieren en un factor positivo, lo cual quiere decir que cuanto menor sea el RMSE, menor será la distancia entre los valores reales y los predichos, esto es, mejor será el ajuste realizado por el modelo. Por lo tanto, un modelo con un RMSE elevado es menos preciso que un modelo con RMSE cercano a cero.

- **Problemas de clasificación:** el criterio elegido para este caso es el AUC (*Area Under the Curve*), es decir, el área bajo la curva ROC, la cual representa la sensibilidad o tasa de verdaderos positivos (TPR):

$$\text{TPR} = \frac{\text{Verdaderos Positivos}}{\text{Verdaderos Positivos} + \text{Falsos Negativos}} \ ,$$

  frente al complementario de la especificidad o tasa de falsos positivos (FPR):

$$\text{FPR} = \frac{\text{Falsos Positivos}}{\text{Falsos Positivos} + \text{Verdaderos Negativos}} \ ,$$

  a lo largo de diferentes umbrales de clasificación.

  El AUC se puede interpretar como la probabilidad de que el modelo ordene un individuo positivo aleatorio más alto que un individuo negativo aleatorio a través del valor de la predicción. De alguna forma, nos da información sobre la capacidad que tiene el modelo de distinguir entre clases. Esta medida toma valores en el intervalo $[0, 1]$, donde un AUC de 0.5 indica el rendimiento esperado de un modelo que decide

azarosamente, como lanzar una moneda al aire. Con un modelo que clasifique perfectamente se obtendría un AUC de 1, mientras que un modelo con un AUC inferior a 0.5 es menos fiable que el azar, siendo un AUC de 0 el obtenido por un modelo que clasificase todo al revés.

La metodología empleada para examinar los algoritmos se basa en realizar una partición de cada conjunto de datos en tres segmentos:

1. **Datos de entrenamiento:** esta fracción es la encargada de ajustar los modelos de *Machine Learning*. Es importante que los modelos sean capaces de captar la mayor cantidad de patrones y relaciones entre variables posible, por ello se le asigna un 70 % de la cantidad total de los datos.

2. **Datos de validación:** esta parte se emplea para decidir qué hiperparámetros proporcionan un modelo final mejor tanto en precisión como en sobreajuste. Esta parte consta de un 15 % del total de datos iniciales.

3. **Datos de testeo:** este es el 15 % restante del conjunto inicial de datos. Se utiliza para evaluar el rendimiento real del modelo seleccionado previamente porque son datos independientes y el resultado obtenido en cuanto a precisión está insesgado.

Todas estas divisiones se realizan de manera aleatoria excepto en caso de haber datos temporales, los cuales se separarían cronológicamente mediante la llamada validación *Out Of Time*, ya que las predicciones lógicas son a futuro. Si se distribuyesen de forma aleatoria, estaríamos entrenando con datos que queremos predecir, lo cual no tiene sentido.

Por otro lado, se propone una serie de posibles hiperparámetros para cada modelo y se ajusta un modelo con cada posible selección. Después de esto, se ponen a prueba con el conjunto de validación. Una vez obtenidos el AUC o el RMSE de cada subconjunto de datos, calculamos la diferencia relativa de rendimiento, a lo que llamamos delta, lo cual nos permite decidir si nuestro modelo peca de *overfitting*, es decir, si el modelo ha aprendido demasiado de los datos de entrenamiento y al enfrentarse a datos nuevos proporciona predicciones deficientes. En este momento se eligen los hiperparámetros que mejor resultado proporcionan y posteriormente, los modelos se reentrenan con los datos de entrenamiento y de validación juntos para enriquecerlos. Finalmente, los sometemos al conjunto de datos de testeo, de donde se obtiene el valor que indica cómo de bien funcionan en realidad y que se va a usar para comparar entre los diferentes tipos de algoritmos.

1.2. Datos empleados

Los seis conjuntos de datos que van a ser utilizados como casos de uso son:

Datos para clasificación:

- **Cáncer de mama:** aquí hay recogidas 569 muestras de células cancerígenas en las que se han medido 30 atributos como lo son, por ejemplo, el radio, la simetría o la cantidad de concavidades en el contorno de la célula. El objetivo es predecir según todas estas mediciones si las células son benignas o malignas.

- **Defectos de software:** estos datos proceden de programas que extraen características McCabe y Halstead de códigos fuente. Estos parámetros fueron definidos en los años 70 cuando se intentaron caracterizar objetivamente los atributos del código asociados con la calidad del software. Disponemos de 101763 muestras con 22 variables más la variable objetivo, es decir, la que queremos predecir, que indica si hay algún defecto o no en el código.

- **Seguros de coche:** la meta en este caso es captar qué perfil de cliente es más propenso a hacer un reclamo de su seguro. Tenemos 10000 clientes con 17 características adicionales como la franja de edad, tipo de vehículo o el número de accidentes anteriores entre otras.

Datos para regresión:

- **Transformador de electricidad:** tenemos registro de 2 años enteros discretizados cada 15 minutos de 8 valores de un transformador de electricidad en China, lo que nos da un total de 69680 observaciones. El objetivo es predecir la temperatura del combustible, a través de la cual se refleja en qué condiciones está el transformador. Este *dataset* contiene datos temporales, por lo cual se le añaden también variables autorregresivas para enriquecerlo.

- **Superconductores:** aquí disponemos de 21263 muestras de superconductores con 81 características fisicoquímicas asociadas. Además se le han añadido 86 columnas indicando el número de átomos de cada elemento químico con número atómico del 1 al 86. Por tanto tenemos 168 columnas contando con la variable objetivo que es la temperatura crítica de cada material, esto es, la temperatura por encima de la cual pierde sus propiedades superconductoras.

- **Precios de casas:** en este conjunto de datos tenemos 1460 muestras de casas con 79 variables explicativas como por ejemplo número de habitaciones, baños, si hay piscina o garaje, la cercanía a una carretera principal o el tipo de calefacción. La finalidad es poder estimar el precio de una vivienda en función de sus características.

2. Algoritmos iniciales

2.1. Regresión Lineal

La Regresión Lineal es una técnica estadística cuyo objetivo es investigar y modelar la relación de dependencia entre y cuando esta es continua y las p variables independientes mediante una expresión lineal. El modelo presupone la siguiente ecuación:

$$y_i = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i \ , \ i = 1, \ldots, n \ . \tag{1}$$

Aquí se hace referencia al valor de y para la observación i en particular. Difícilmente todos los valores observados van a pertenecer exactamente a un hiperplano, de ahí que se añada el término de error ε_i. La interpretación de los parámetros es la siguiente:

- β_0 es el *intercept*, se corresponde con la media de y cuando todas las variables explicativas son cero.

- β_j es el coeficiente de regresión asociado a X_j, representa el cambio que sufre y cuando x_j aumenta una unidad, manteniendo todas las demás variables predictoras constantes. Cabe destacar que la magnitud de cada coeficiente depende de la escala de la variable predictora correspondiente, es decir, si $\beta_r \gg \beta_s$ no significa necesariamente que X_r sea más importante que X_s.

- ε_i es el residuo o error, la diferencia entre el valor real y el dado por el hiperplano (el estimado). Se trata de un error aleatorio que existe debido a posibles errores de medición o a la suma de efectos producidos por otras variables explicativas no consideradas en el problema y se presupone que se distribuye con valor medio 0.

Por lo general, los coeficientes de Regresión no son conocidos, por lo que hay que estimarlos; este proceso es lo que comunmente se conoce como ajustar el modelo a los datos. Lo más frecuente es hacer este cálculo mediante Mínimos Cuadrados, método que determina como mejor modelo el hiperplano que logra minimizar la suma de los residuos al cuadrado. Escribimos matricialmente el modelo para facilitar el proceso:

$$y = X\beta + \varepsilon \ , \tag{2}$$

o bien,

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

La función a minimizar es la suma de los residuos al cuadrado respecto de β:

$$SSE = \sum_{i=1}^{n} \varepsilon_i^2 = \varepsilon^T \varepsilon = (y - X\beta)^T (y - X\beta), \tag{3}$$

buscamos la solución analíticamente:

$$\frac{\partial SSE}{\partial \beta} = -2X^T(y - X\beta) = 0 \Longleftrightarrow X^T X\beta = X^T y \Longrightarrow \hat{\beta} = (X^T X)^{-1} X^T y.$$

Una vez hecho esto, se pueden obtener las estimaciones del conjunto de datos, según (2): $\hat{y} = X\hat{\beta}$, o para un individuo en concreto, atendiendo a (1): $\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}$.

Por último, hay que tener en cuenta las siguientes hipótesis:

1. La matriz $X^T X$ no es singular, o equivalentemente, X es rango columna completo. Esta condición permite el cálculo de $\hat{\beta}$. Esto se conoce como ausencia de multicolinealidad.

2. $\varepsilon | X \sim N(0, \sigma^2 I_n)$. Esto implica dos cosas:

   a) $E(\varepsilon|X) = 0$, lo cual a su vez implica que $E(y|X) = X\beta$, es decir, que exista relación lineal entre la variable objetivo y las explicativas.

   b) $Var(\varepsilon|X) = Var(y|X) = \sigma^2 I_n$, lo que conlleva $Cov(\varepsilon_i, \varepsilon_j) = 0 \ \forall i \neq j$ y que $Var(\varepsilon_i) = \sigma^2 \ \forall i$, esto es, homocedasticidad e independencia en los residuos; implícitamente también estamos diciendo que $y \sim N(X\beta, \sigma^2 I_n)$.

Es raro que se cumplan todas ellas en la práctica, sin embargo, esto no quita que el modelo sea útil. Estas asunciones se encuentran subyacentes en el desarrollo matemático

y hay que tenerlas presentes y ser conscientes de las consecuencias que pueden tener en los resultados finales.

Una de las limitaciones de la Regresión Lineal es que todas las variables explicativas se incluyen en el modelo aunque no aporten información relevante, lo cual incrementa el tiempo de computación y puede traer problemas de interpretación y de correlaciones. Para mitigar este problema existe la regularización, método que consiste en agregar una función de penalización en el ajuste por Mínimos Cuadrados con el objetivo de reducir la varianza de las predicciones, es decir, evitar *overfitting*, y conseguir así que los modelos generalicen mejor. Entonces, el problema de optimización pasa de minimizar SSE, a ser el siguiente:

$$\begin{aligned} &\text{Min} \qquad (y - X\beta)^T(y - X\beta) + \lambda p(\beta) \\ &\beta \in \mathbb{R}^p \end{aligned}$$

donde $\lambda > 0$ es el parámetro de regularización y p es la función de penalización. Son tres las principales estrategias de regularización:

- **Ridge:** cuando $p(\beta) = ||\beta||_2^2 = \sum_{j=1}^{p} {\beta_j}^2$. Esta tiene el efecto de reducir de forma proporcional el valor de todos los coeficientes del modelo pero sin que estos lleguen a ser nulos. La principal ventaja de aplicar es la reducción de varianza en las predicciones. La desventaja es que el modelo final continúa incluyendo todos los predictores, lo cual supone un problema para la interpretación del modelo.

- **Lasso:** si $p(\beta) = ||\beta||_1 = \sum_{j=1}^{p} |\beta_j|$. En este caso los coeficientes de β son forzados a tender a cero. El principal beneficio es que el modelo excluye variables poco relevantes, lo cual puede dar pie a modelos inestables cuando existen correlaciones entre variables explicativas ya que se vuelca todo el peso las variables que sobreviven.

- **Elastic Net:** esta combina las dos anteriores siendo $p(\beta) = \alpha||\beta||_1 + \frac{1-\alpha}{2}||\beta||_2^2$, donde α controla el peso que tiene cada una de las penalizaciones. Lo que se pretende conseguir con esta penalización es poder eliminar variables poco relevantes pero dando cierta estabilidad por si existiesen altas correlaciones.

Cabe destacar que como los métodos de regularización influyen en la magnitud de los coeficientes de Regresión, es conveniente que todos estén en la misma escala, por ello es recomendable escalar los datos de entrenamiento.

2.2. Regresión Logística

Cuando la variable objetivo y en vez de ser continua es binaria, si se ajusta un modelo de Regresión Lineal, las predicciones dadas por el hiperplano de regresión pueden ser distintas de 0 y 1, lo cual es una incongruencia. Como los valores predichos no tienen por qué caer dentro del intervalo [0, 1], no es viable asociar las predicciones dadas con probabilidad de pertenecer a cada clase. Para solventar estos problemas se realiza una transformación de las predicciones a través de una función que solamente devuelva valores comprendidos entre 0 y 1. Para ello se hace uso de la función sigmoide:

$$\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \ ,$$

dando lugar a la Regresión Logística.

Tal y como decíamos en la Regresión Lineal: $E(y|X) = X\beta$, y en este caso,

$$E(y|X) = 1 \cdot P(y=1|X) + 0 \cdot P(y=0|X) = P(y=1|X) =: p \ ,$$

por tanto, el modelo logístico que se obtiene para una observación i concreta es:

$$p_i = \frac{e^{\beta_0+\beta_1 x_{i1}+\cdots+\beta_p x_{ip}}}{1+e^{\beta_0+\beta_1 x_{i1}+\cdots+\beta_p x_{ip}}} \Longrightarrow \ln\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip} \ . \tag{4}$$

De esta forma, el modelo lineal se aplica al logaritmo del ratio de ambas clases. Es decir, el modelo indica cuán más probable es que un individuo sea clase 1 que clase 0. Interpretar el modelo lineal es confuso, por eso se suele estudiar el *odds ratio*:

$$\frac{p_i}{1-p_i} = e^{\beta_0} \cdot e^{\beta_1 x_{i1}} \cdot \cdots \cdot e^{\beta_p x_{ip}}.$$

Una variación en esta proporción representa el cambio multiplicativo en la propensión de ser clase 1 para un aumento de una unidad en cierta variable predictora mientras se mantienen constantes todas las demás. Es decir, si $e^{\beta_j x_{ij}} > 1$, el incremento en una unidad de x_{ij} conlleva un aumento proporcional de la propensión a pertencer a la clase 1. En cambio, si $e^{\beta_j x_{ij}} < 1$, aumentar una unidad x_{ij} hará que esta misma propensión disminuya.

Para familiarizarse con este valor, supongamos que la probabilidad de éxito de un suceso (la proporción de 1 en y en nuestro *dataset*) es 0.8, luego el *odds ratio* correspondiente (el que devolvería un modelo perfecto) es igual a 4 obedeciendo a la propia expresión, lo cual es equivalente a decir que una observación aleatoria de dicho suceso es

cuatro veces más propensa a ser exitosa (clase 1) que ser un fracaso, o también, que se esperan 4 eventos existosos por cada fracaso.

En este caso, para hallar β se emplea el método de Máxima Verosimilitud. Al tratarse de un problema de clasificación, consideramos que cada variable aleatoria y_i sigue una distribución Bernoulli con probabilidad de éxito p. Por lo tanto, la función de verosimilitud que hay que maximizar respecto de β es:

$$\mathcal{L}(p|y_1, \ldots, y_n) = \prod_{i=1}^{n} f_p(y_i) = \prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i} \ . \tag{5}$$

El vector de coeficientes estimados $\hat{\beta}$ se obtiene aplicando métodos numéricos iterativos.

Este tipo de Regresión devuelve valores reales entre 0 y 1 pero para ser coherentes hay que clasificar las predicciones. Para ello se establece un umbral, en nuestro caso es 0.5, y todo individuo que tenga una estimación superior a este umbral se asigna al grupo 1, y al grupo 0 en caso contrario.

2.3. Árboles de decisión

2.3.1. CART

Los árboles de decisión CART (***C**lassification **A**nd **R**egression **T**rees*) [4], a diferencia de los métodos previos, no desarrollan una ecuación de predicción que se aplica a todo el espacio muestral. En su lugar, hacen particiones en los datos de forma recursiva en regiones simples hasta llegar al punto de tener el espacio suficientemente dividido para poder predecir de manera precisa. Veamos un ejemplo gráfico de un árbol de clasificación:

Supongamos que queremos detectar si los pacientes que entran en urgencias en un hospital son diabéticos lo más apresuradamente posible. Para ello, se toman medidas de 27 pacientes aleatorios de su nivel de glucosa en sangre y su presión arterial, y se clasifican como clase 0 si no son diabéticos y como clase 1 si lo son. La representación del árbol de decisión ajustado con este *dataset* ficticio se encuentra en la Figura 1.

El algoritmo funciona intuitivamente, a partir de los valores de las pruebas realizadas a un paciente, se comprueba si la primera condición, que en este caso es si el nivel de glucosa es menor o igual que 152.5 mg/dL, es verdadera o falsa. Si se cumple dicha condición, se sigue la flecha superior, de lo contrario, la inferior, llegando al siguiente nodo en el que se hará su comprobación respectiva. Este proceso se repite hasta alcanzar el nodo

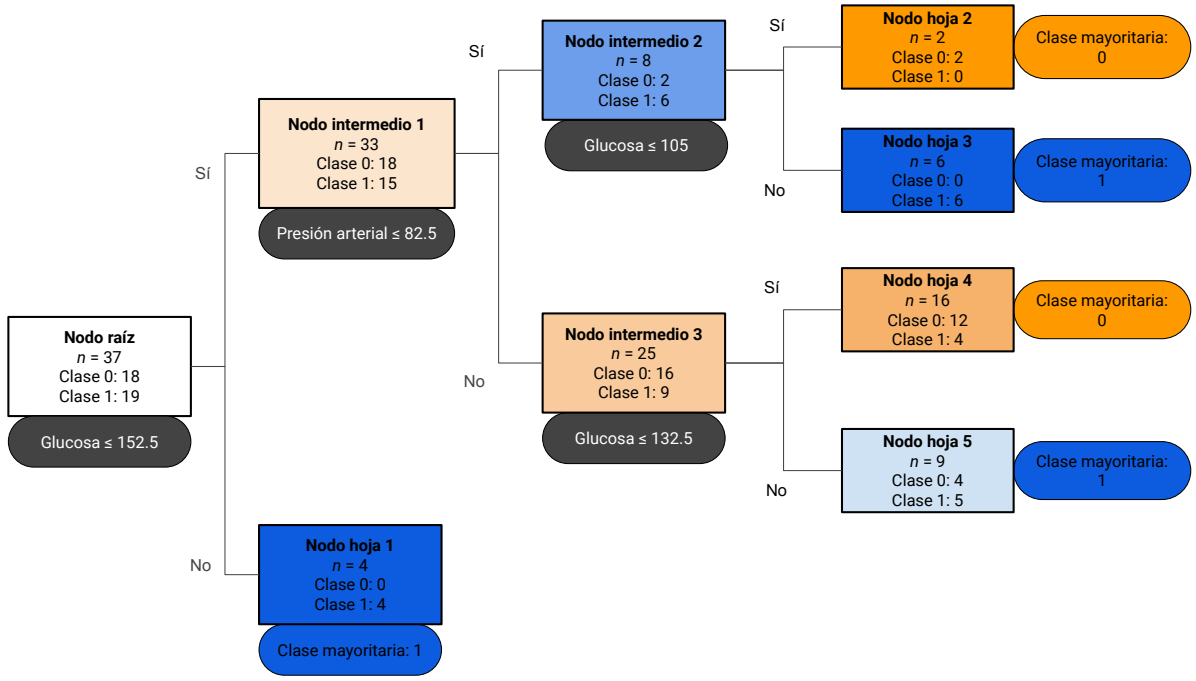


Figura 1: Árbol de decisión de clasificación del ejemplo.

donde no hay condiciones que comprobar. En este paso es donde la predicción se lleva a cabo; este árbol en concreto clasifica como diabéticos los nuevos pacientes que acaben en los nodos finales azules, mientras que los nodos terminales naranjas no diagnostican diabetes. En cada nodo de la Figura 1 se encuentra la información relativa al propio nodo: la condición que se comprueba (si la hay), el número de observaciones que contiene y cuántas observaciones hay de cada clase, además de tener un color en función de la clase mayoritaria contenida cuya tonalidad también varía según el porcentaje relativo de esta clase.

Dado que solo hay dos variables predictoras en nuestro ejemplo, podemos hacer una representación sobre el plano de los individuos. Es fácil ver que las condiciones que impone el árbol de decisión se corresponden con particiones del plano paralelas a los ejes. La Figura 2 muestra la superficie de decisión dada por el modelo ilustrado en la Figura 1. A todos los pacientes que entren en urgencias se les tomarán estas dos medidas inmediatas y si sus resultados se sitúan sobre el área azul, se considerarán diabéticos, mientras que si caen sobre el área naranja según sus atributos, no se catalogarán como tal.

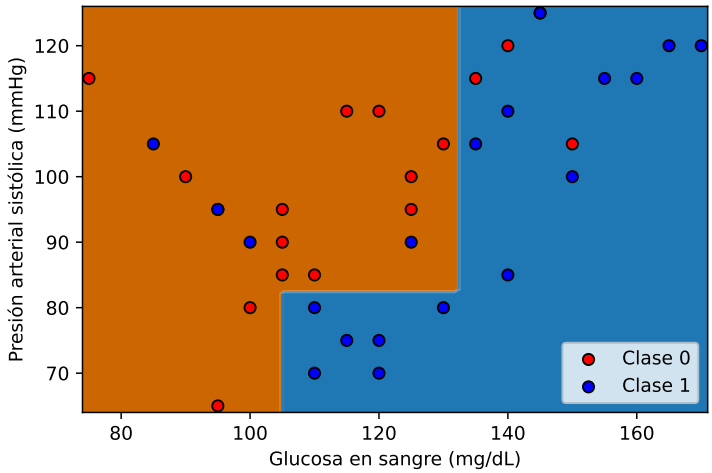


Figura 2: Partición del espacio muestral dado por el árbol de la Figura 1.

Vamos a visualizar también un árbol de regresión con otro ejemplo sencillo:

Supongamos que disponemos de una serie temporal que recoge la temperatura de cierta ciudad medida cada hora durante un día entero. Ajustamos un árbol que solo realice tres fases de división y obtenemos el modelo de la Figura 3.

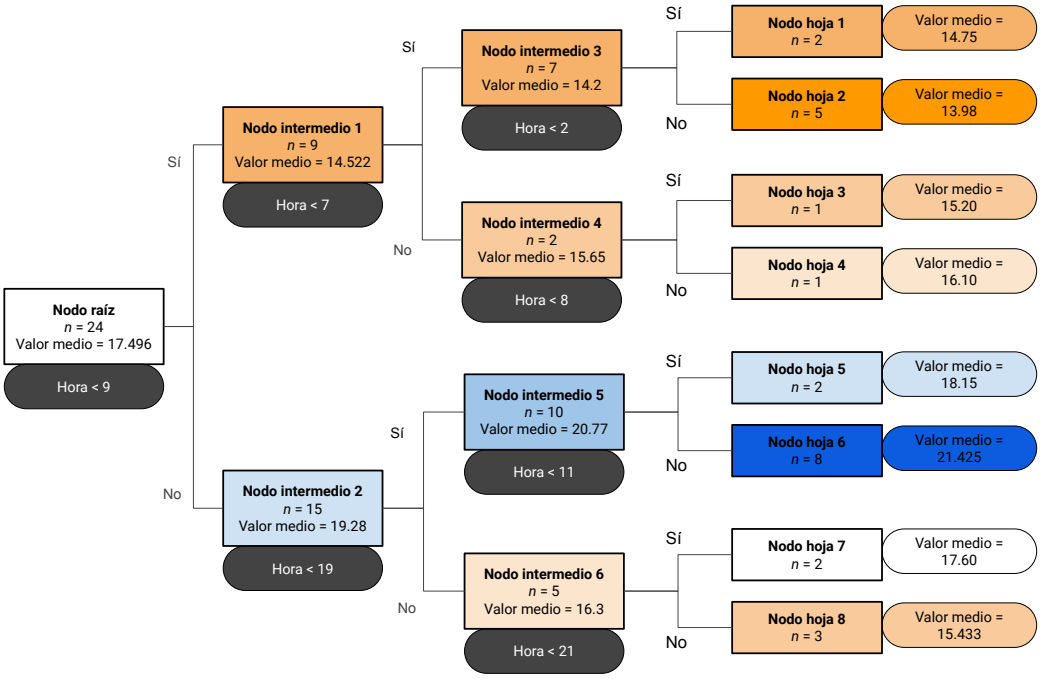


Figura 3: Árbol de decisión de regresión del ejemplo.

Al igual que en los árboles de clasificación, se enseña la condición y el número de individuos de cada nodo pero a diferencia de estos otros, lo que aquí también se muestra es el valor medio de la variable objetivo calculado sobre los individuos que se encuentran en dicho nodo. Este árbol da las predicciones que se ven en la Figura 4:

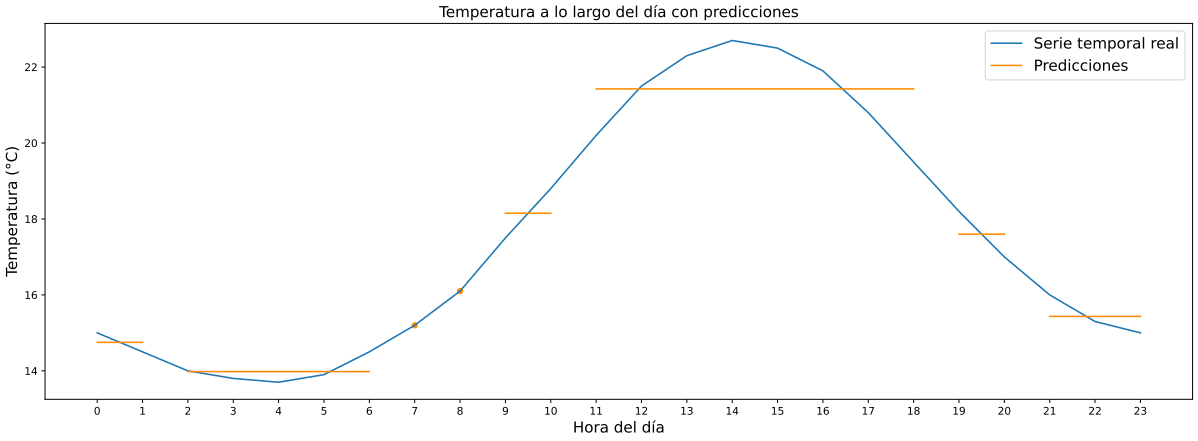


Figura 4: Serie temporal y predicciones dadas por el árbol de la Figura 3.

Cabe observar que las predicciones dadas son efectivamente con el valor medio de cada

nodo hoja.

Ahora que hemos visto intuitivamente cómo funcionan los CART, toca explicar el procedimiento que realmente siguen, pero primero vamos a dar algo de notación. En un árbol de decisión se distinguen tres tipos de nodos, como ya adelantaban las imágenes:

- **Nodo raíz:** es el nodo superior, el que contiene toda la muestra y donde se lleva a cabo la primera partición.
- **Nodo interno:** todo aquel nodo que se encuentra en un punto intermedio del árbol, es decir, que proviene de una división y que realiza otra. Dentro de los nodos internos se distinguen dos tipos: el nodo padre, que es aquel anterior a otro nodo en particular; y el nodo hijo, el cual es el que se crea tras la división de otro nodo concreto.
- **Nodo hoja:** es un nodo terminal, el cual no da lugar a más divisiones y donde se efectúan las predicciones.

Notar que tanto en la Figura 1 como en la Figura 3, hemos llamado intermedios a los nodos internos.

Todo CART deja el espacio muestral separado en M regiones disjuntas que se corresponden con las hojas del propio árbol. A cada región la llamaremos R_m, con $m \in \{1, 2, \ldots, M\}$ y c_m al valor asociado a todo individuo perteneciente a R_m. Llamaremos T al árbol cuando nos refiramos a él como función que toma valores del espacio de variables independientes D y devuelve valores en E, espacio donde esté definida y, ($\mathbb{R}$ si es continua o bien $\{0, 1\}$ si es binaria). Por tanto, se tiene

$$\begin{aligned} &T : D \longrightarrow E \\ &T(x) = \sum_{m=1}^{M} c_m I_{R_m} \ , \end{aligned} \tag{6}$$

donde I_{R_m} es la función indicadora:

$$I_{R_m}(x) = \begin{cases} 1 & \text{si} \quad x \in R_m \\ 0 & \text{si} \quad x \notin R_m \end{cases} . \tag{7}$$

Ahora vamos con los criterios que sigue el algoritmo a la hora de realizar las divisiones en cada nodo, seguidamente de qué reglas obedece para saber cuándo parar de dividirse y cómo decide qué valores devolver en cada hoja. En cada punto se va a analizar

específicamente cada tipo de árbol en función del problema que se aborde (regresión o clasificación).

Criterios de escisión

En la raíz y en cada nodo interno del árbol se lleva a cabo una división binaria la cual se ejecuta en base a cierto valor s de cierta variable predictora X_j. Consideremos la primera escisión del algoritmo, que tiene lugar en el nodo raíz. Esta división da lugar a dos nuevas regiones:

$$R_1 = \{x_i \mid x_{ij} \leq s\} \ \wedge \ R_2 = \{x_i \mid x_{ij} > s\} \ ,$$

las cuales generan dos nodos a los cuales se les asigna como valor la media de y de su propia muestra. Dado que este proceso es recursivo, cada uno de estos dos nuevos nodos se divide a su vez en otras dos regiones determinadas por ciertos valores r y t de ciertas variables explicativas X_k y X_l, respectivamente. Es importante recalcar que estas nuevas divisiones también atienden a las condiciones de las regiones de sus nodos padres. Este proceso se repite sucesivamente hasta que se alcanza alguno de los criterios de detención.

En este punto, el algoritmo ordena ascendentemente los valores que toma cada variable X_j, con $j \in \{1, \ldots, p\}$, e identifica el punto intermedio entre cada par de valores de esa lista como posible punto de corte s.

En los árboles de regresión, se calcula la suma de errores al cuadrado (varianzas) que se consigue con cada posible división registrada:

$$SSE = \sum_{x_i \in R_1} (y_i - \bar{y}_{R_1})^2 + \sum_{x_i \in R_2} (y_i - \bar{y}_{R_2})^2 \ ,$$

y se seleccionan los j y s que determinen las regiones R_1 y R_2 tales que logren el menor resultado posible, lo cual significa que la varianza de las predicciones es mínima (si existe más de un par (j, s) que alcance el mínimo SSE posible, se elige uno de ellos al azar).

Por otro lado, la regla empleada más frecuentemente en los árboles de clasificación para realizar las divisiones de nodos es el Índice Gini, que cuantifica la varianza total de las clases en un nodo. Para cada posible división, se calcula el Índice Gini de los dos nodos resultantes como sigue:

$$G_k = p_{k0}(1 - p_{k0}) + p_{k1}(1 - p_{k1}) \ ,$$

donde p_{ki} es la proporción de individuos de clase $i \in \{0, 1\}$ en el nodo $k \in \{1, 2\}$. Cuando p_{ki} es muy próximo tanto a 0 como a 1, provoca que el producto $p_{ki}(1 - p_{ki})$ sea muy

cercano a 0, por lo tanto, esta métrica es indicadora de la pureza de un nodo; un Índice Gini pequeño implica que p_{k0} y p_{k1} son valores cercanos a 0 o a 1 y entonces ese nodo se dice que es puro.

Posteriormente, se calcula la media ponderada de esas dos medidas. El Índice Gini que se le asocia a la división es:

$$G = \frac{n_1}{n_1 + n_2} G_1 + \frac{n_2}{n_1 + n_2} G_2 \ ,$$

donde n_i es el número de individuos del nodo hijo i dado por R_i, con $i \in \{1, 2\}$ y G_1 y G_2 son los Índices Gini de los nodos 1 y 2, respectivamente.

Finalmente, se escogen los valores j y s que definen las regiones R_1 y R_2 de manera que minimicen G.

Criterios de parada

El proceso de construcción del árbol continúa dividiendo nodos reiteradamente hasta que se cumple alguno de los siguientes criterios, momento en el cual el nodo en cuestión se etiqueta como nodo hoja:

1. Cuando se alcanza la profundidad máxima del árbol, es decir, si se realiza el número máximo de escisiones consecutivas permitidas.

2. Cuando la cantidad de individuos de un nodo es inferior a una cota establecida.

3. Si, al realizar la división, el número de individuos de cada nodo hijo es inferior a una cota dada.

4. Si la división del nodo no induce una disminución de la impureza Δ_k mayor que cierto valor límite. Su expresión es:

   $$\Delta_k := \frac{n_k}{n} \left(H_k - \frac{n_{k1}}{n_k} H_{k1} - \frac{n_{k2}}{n_k} H_{k2} \right) \ ,$$

   donde n es el número total de individuos, n_k es el número de muestras en el nodo k en cuestión, n_{k1} y n_{k2} son los números de individuos en el hijo 1 y en el hijo 2 respectivamente, H_k es la impureza del nodo k e H_{k1}, H_{k2} son las correspondientes impurezas de los nodos hijos. La impureza en los árboles de regresión es la varianza y en el caso de clasificación, el Índice Gini.

5. En los de clasificación, cuando un nodo contiene todos sus individuos de la misma clase.

Un escenario hipotético en el cual se asignen cotas relajadas en exceso, podría resultar en árboles sobreajustados. En contraste, si estas cotas son muy estrictas, los árboles podrían verse forzados a estar *underfitted*, esto es, ni siquiera llegarían a ajustar bien los datos de entrenamiento.

Para conseguir un modelo compensado surge la idea de podar el árbol. Los CART de la librería *scikit-learn* de *Python* basan su poda en el método *Minimal Cost-Complexity Pruning*. Este algoritmo requiere fijar un parámetro $\alpha \geq 0$ para definir el coste de complejidad:

$$C_\alpha(T) := C(T) + \alpha M \ ,$$

donde M es el número total de hojas del árbol y $C(T)$ es la suma de las impurezas según el tipo de árbol de todos los nodos terminales de T.

Primero se crea un árbol T_0 sin condiciones de parada más allá del número mínimo de observaciones por nodo terminal. A continuación, se busca el sub-árbol T de T_0 que minimiza $C_\alpha(T)$, encontrando así un modelo que prediga suficientemente bien pero sin llegar a extenderse excesivamente.

Criterios de predicción

En el nodo hoja $m \in \{1, \ldots, M\}$ de un árbol de regresión se da como predicción el mismo valor constante c_m a los n_m individuos contenidos en la región R_m. Al igual que el la escisión de nodos, aquí también se pretende minimizar el error de predicción, luego

$$c_m = \underset{c}{\operatorname{argmin}} \sum_{i:x_i \in R_m} (y_i - c)^2 \ . \tag{8}$$

Si llamamos $f(c)$ a la función objetivo anterior e igualamos a cero su derivada, obtenemos:

$$f'(c) = -2 \sum_{i:x_i \in R_m} (y_i - c) = 2(n_m c - \sum_{i:x_i \in R_m} y_i) = 0 \iff$$
$$\iff c = \frac{1}{n_m} \sum_{i:x_i \in R_m} y_i \ .$$

Como f es una parábola convexa, su punto crítico $\hat{c} = \bar{y}_{R_m}$ es mínimo. En definitiva, cada nodo hoja da como predicción final c_m la media de su muestra $\bar{y}_{R_m}$ lo cual explica por qué a cada nodo interno se le asocia también su media.

En las hojas de los árboles de clasificación, generalmente se emplea la moda de y como c_m, es decir, la clase que se repite más veces es la asignada. Además, resulta útil conocer el porcentaje de cada clase en los nodos terminales, lo cual aporta información sobre la confianza de las predicciones.

Entre los beneficios de los CART se incluyen su bajo coste computacional, ser fácilmente comprensibles gracias a que se pueden graficar, su utilidad para hacer un análisis exploratorio inicial de los datos o que no necesitan de preprocesamiento de los datos para funcionar. Por otra parte, no son los mejores modelos cuando existe relación lineal entre y y las X_j ni cuando las predicciones pueden estar fuera del rango de valores de entrenamiento; además no es fácil encontrar el equilibrio entre una gran precisión predictiva y no sobreajustarse.

2.3.2. Algoritmos de bagging

Esta familia de algoritmos nace con la finalidad de crear modelos más estables, más precisos con datos ruidosos y que pequen menos de *overfitting* que los CART. Su nombre proviene de ***b****ootstrap* ***agg****regat****ing***. Se caracterizan por ser algoritmos de ensamblaje, es decir, generan diversos árboles de tipo CART independientes que contribuyen a una sola predicción a parte de hacer uso del *bootstrap*, que es una técnica consistente en tomar diversas muestras con repetición aleatorias del conjunto de datos inicial y cada una de ellas es la que sirve para entrenar cada árbol del "bosque" [2].

El algoritmo de *bagging* más empleado es ***Random Forest*** (RF) [3]. Fue publicado por Leo Breiman en el año 2001. El procedimiento de este algoritmo requiere en primer lugar que se especifique el número de árboles o estimadores N del que va a constar el modelo. En segundo lugar, cada estimador T_j es entrenado con una muestra diferente producida por el *bootstrap*, además de estar restringido en cada nodo a m, usualmente $m = \lfloor \sqrt{p} \rfloor$, variables explicativas aleatorias sobre las que poder buscar el mejor umbral de escisión. Finalmente, se determina una predicción democráticamente; en los *Random Forest* de regresión, la respuesta final es la media de las predicciones de todos los árboles:

$$RF(x) = \frac{1}{N} \sum_{j=1}^{N} T_j(x) \ ,$$

y en los de clasificación, se promedian las predicciones porcentuales de los estimadores, y se adjudica la clase con mayor porcentaje, logrando así los dos tipos de predicción, como en los árboles de decisión.

Gracias a estas adaptaciones, los modelos son conscientes de todas las variables y hacen partícipes de las predicciones finales a gran parte de ellas a diferencia de los CART, que en caso de existir alguna variable altamente relacionada con la variable objetivo, esta absorbe todo el protagonismo y no permite encontrar posibles patrones secundarios. De esta forma, Breiman logró indagar más en las variables explicativas, a parte de conseguir modelos cuyas predicciones en datos de testeo sean tan fiables como en datos de entrenamiento. La parte negativa de estos algoritmos es que pierden la faceta de ser fácilmente interpretables.

2.3.3. Algoritmos de boosting

Este conjunto de algoritmos hace uso de varios modelos sencillos predictivos trabajando conjuntamente para dar lugar a un modelo fuerte que sea capaz de lograr predicciones muy precisas. A finales de los años 90 se publican los primeros artículos sobre este tipo de algoritmos pero no es hasta el año 2001 que sale a la luz el trabajo de Jerome Friedman en el que presenta el ***Gradient Boosting*** [8], el primero de los dos algoritmos que veremos en esta sección.

El proceso iterativo llevado a cabo en *Gradient Boosting* es una adaptación del famoso método de optimización del Descenso del Gradiente y consiste en los pasos siguientes:

1. Definir una *Loss Function* o función de pérdida $L(y_i, F(x))$ diferenciable que cuantifique los errores cometidos en las aproximaciones la cual ha de ser minimizada.

2. Inicializar el modelo con una estimación constante $F_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, \gamma)$ para todas las observaciones x_i del *dataset*.

3. Ejecutar un bucle desde $j = 1$ hasta N donde:

   a) Se halla, para cada observación de los datos de entrenamiento, la derivada de $L(y_i, F(x))$ respecto del valor predicho $F(x)$, se evalúa en $F_{j-1}(x)$ y se cambia de signo. Este número se denota r_{ij} y es conocido como pseudorresiduo.

   b) Se entrena un árbol de regresión T_j siendo los r_{ij} los valores de la variable objetivo que genere las regiones R_{mj}, para $m = 1, \ldots, M_j$ dadas por los nodos hoja del propio T_j. Normalmente, estos árboles tienen una profundidad máxima entre 2 y 4.

   c) Para cada nodo hoja m de este árbol T_j, se calcula un nuevo valor de salida $\gamma_{mj} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{mj}} L(y_i, F_{j-1}(x_i) + \gamma)$.

d) Se actualiza la estimación j-ésima sumando el resultado devuelto por el árbol escalado por un tamaño de paso $\nu \in [0, 1]$ llamado tasa de aprendizaje o *learning rate* que sirve para paliar el *overfitting*: $F_j(x) = F_{j-1}(x) + \nu T_j(x)$.

4. Devolver como respuesta final $F_N(x)$.

En el caso de regresión, lo más usual es tomar

$$L(y_i, F(x)) = \frac{1}{2}(y_i - F(x))^2 \ ,$$

ya que su derivada coincide con el error de predicción:

$$r_{ij} = -\left[\frac{\partial L(y_i, F(x))}{\partial F(x)}\right]_{F(x)=F_{j-1}(x_i)} = -\frac{1}{2} \cdot 2(y_i - F_{j-1}(x_i)) \cdot (-1) = y_i - F_{j-1}(x_i) \ .$$

En cuanto al valor inicial,

$$F_0(x) = \underset{\gamma}{\operatorname{argmin}} \ \frac{1}{2} \sum_{i=1}^{n} (y_i - \gamma)^2 \ ,$$

es una situación análoga a la de la expresión (8) en la sección 2.3.1, luego:

$$F_0(x) = \frac{1}{n} \sum_{i=1}^{n} y_i = \bar{y} \ .$$

Una vez construido T_j, el problema de optimización que hay que abordar para hallar γ_{mj} tiene como función objetivo:

$$\frac{1}{2} \sum_{x_i \in R_{mj}} (y_i - F_{j-1}(x_i) - \gamma)^2 = \frac{1}{2} \sum_{x_i \in R_{mj}} (r_{ij} - \gamma)^2 \ ,$$

por lo tanto también es análogo a (8) y, consecuentemente, $\gamma_{mj} = \bar{y}_{R_{mj}}$, es decir, T_j funciona exactamente igual que cualquier árbol regresor.

Cuando la variable respuesta y es dicotómica, *Gradient Boosting* emplea el logaritmo del *odds ratio* como predicción ($F(x)$), que se puede interpretar como el equivalente en Regresión Logística de la media. El papel de función de pérdida lo hace el logaritmo de la función de verosimilitud (5) cambiado de signo, ya que la meta es minimizar L y lo que se buscaba con la de verosimilitud era maximizarla. Se tiene pues:

$$\ln(\mathcal{L}(p)) = \ln\left(\prod_{i=1}^{n} p^{y_i}(1-p)^{1-y_i}\right) = \sum_{i=1}^{n}(y_i \ln(p) + (1-y_i)\ln(1-p)) \ .$$

Como la *Loss Function* se aplica a cada observación por separado, podemos deshacernos del sumatorio:

$$L(y_i, p) = -(y_i \ln(p) + (1-y_i)\ln(1-p)) = y_i \ln(1-p) - y_i \ln(p) - \ln(1-p) =$$
$$= -y_i(\ln(p) - \ln(1-p)) - \ln(1-p) = -y_i \ln\left(\frac{p}{1-p}\right) - \ln(1-p).$$

De las expresiones (4) sustituimos p en el segundo sumando y queda:

$$\ln(1-p) = \ln\left(1 - \frac{e^{\ln\left(\frac{p}{1-p}\right)}}{1+e^{\ln\left(\frac{p}{1-p}\right)}}\right) = \ln\left(\frac{1}{1+e^{\ln\left(\frac{p}{1-p}\right)}}\right) = -\ln\left(1+e^{\ln\left(\frac{p}{1-p}\right)}\right) \ ,$$

con lo que

$$L(y_i, F(x)) = \ln\left(1+e^{F(x)}\right) - y_i F(x) \ .$$

Ahora toca encontrar $F_0(x)$. Sea $f(\gamma) = \sum_{i=1}^{n}(\ln(1+e^{\gamma}) - y_i\gamma)$. Minimizamos esta función derivando e igualando a 0:

$$f'(\gamma) = \sum_{i=1}^{n}\left(\frac{e^{\gamma}}{1+e^{\gamma}} - y_i\right) = n\frac{e^{\gamma}}{1+e^{\gamma}} - \sum_{i=1}^{n} y_i = 0 \iff$$
$$\iff \frac{e^{\gamma}}{1+e^{\gamma}} = \bar{y} \iff e^{\gamma} = \bar{y}(1+e^{\gamma}) = \bar{y} + \bar{y}e^{\gamma} \iff$$
$$\iff e^{\gamma} - \bar{y}e^{\gamma} = e^{\gamma}(1-\bar{y}) = \bar{y} \iff$$
$$\iff e^{\gamma} = \frac{\bar{y}}{1-\bar{y}} \iff \gamma = \ln\left(\frac{\bar{y}}{1-\bar{y}}\right) \ .$$

Dado que la variable y es o bien 0 o bien 1, su media es sencillamente p. A continuación evaluamos la segunda derivada de f en el punto crítico $\hat{\gamma} = \ln\left(\frac{p}{1-p}\right)$ y comprobamos su signo:

$$f''(\gamma) = n\frac{e^{\gamma}(1+e^{\gamma}) - e^{\gamma}e^{\gamma}}{(1+e^{\gamma})^2} = n\frac{e^{\gamma}}{(1+e^{\gamma})^2} > 0 \quad \forall\gamma \in \mathbb{R},$$

por tanto, es positiva en $\hat{\gamma}$ y con ello concluimos que es mínimo. Así, $F_0(x)$ es el logaritmo del *odds ratio*.

Del cálculo de f' obtenemos los pseudorresiduos de este caso:

$$r_{ij} = -\left[\frac{\partial L(y_i, F(x))}{\partial F(x)}\right]_{F(x)=F_{j-1}(x_i)} = y_i - \frac{e^{F_{j-1}(x_i)}}{1+e^{F_{j-1}(x_i)}} = y_i - p_{i(j-1)} \ ,$$

donde hemos reutilizado las expresiones (4) para simplificar. La tarea siguiente es ajustar el árbol regresor T_j con variable objetivo r_j. Ahora el enfoque de calcular γ_{mj} mediante la derivada de L se complica, de modo que utilizamos la aproximación de segundo orden del polinomio de Taylor. Lo hacemos para un i cualquiera para evitar sumatorios:

$$\begin{aligned} &L(y_i, F_{j-1}(x_i) + \gamma) \approx \\ &\approx L(y_i, F_{j-1}(x_i)) + \frac{\partial}{\partial F_{j-1}(x_i)} L(y_i, F_{j-1}(x_i))\gamma + \frac{1}{2}\frac{\partial^2}{\partial F_{j-1}(x_i)^2} L(y_i, F_{j-1}(x_i))\gamma^2. \end{aligned}$$

Tomando pues la derivada respecto de γ:

$$\frac{\partial}{\partial \gamma} L(y_i, F_{j-1}(x_i) + \gamma) \approx \frac{\partial}{\partial F_{j-1}(x_i)} L(y_i, F_{j-1}(x_i)) + \frac{\partial^2}{\partial F_{j-1}(x_i)^2} L(y_i, F_{j-1}(x_i))\gamma \ ,$$

igualamos a cero y resolvemos para γ:

$$\begin{aligned} \frac{\partial}{\partial F_{j-1}(x_i)} L(y_i, F_{j-1}(x_i)) + \frac{\partial^2}{\partial F_{j-1}(x_i)^2} L(y_i, F_{j-1}(x_i))\gamma = 0 \iff \\ \iff \gamma = -\frac{\frac{\partial}{\partial F_{j-1}(x_i)} L(y_i, F_{j-1}(x_i))}{\frac{\partial^2}{\partial F_{j-1}(x_i)^2} L(y_i, F_{j-1}(x_i))} \ . \end{aligned}$$

El numerador coincide con los pseudorresiduos multiplicados por -1 y el denominador es:

$$\begin{aligned} \frac{\partial^2}{\partial F_{j-1}(x_i)^2} L(y_i, F_{j-1}(x_i)) = \frac{\partial}{\partial F_{j-1}(x_i)} \left(\frac{e^{F_{j-1}(x_i)}}{1+e^{F_{j-1}(x_i)}} - y_i\right) = \\ = \frac{e^{F_{j-1}(x_i)}}{\left(1+e^{F_{j-1}(x_i)}\right)^2} = p_{i(j-1)}(1 - p_{i(j-1)}) \ . \end{aligned}$$

Así:

$$\gamma = -\frac{-r_{ij}}{p_{i(j-1)}(1-p_{i(j-1)})} = \frac{y_i - p_{i(j-1)}}{p_{i(j-1)}(1-p_{i(j-1)})} \ .$$

Retomando los sumatorios que se han omitido, gracias a la linealidad de la derivada:

$$\gamma_{mj} = \frac{\sum\limits_{x_i \in R_{mj}} (y_i - p_{i(j-1)})}{\sum\limits_{x_i \in R_{mj}} p_{i(j-1)}(1-p_{i(j-1)})} \ .$$

Por último, en este caso se ha de transformar la estimación ya que no interesa saber el logaritmo del *odds ratio*, si no la propensión de que un individuo sea clase 0 o clase 1. Por ello se vuelve a utilizar la expresión (4) para conseguir esta cantidad y con ella poder etiquetar dependiendo si la probabilidad predicha es menor que 0.5 o no.

Cabe destacar que la elección de la *Loss Function* en estos modelos influye significativamente, ya que determina la forma del valor inicial y de las estimaciones de cada paso; que los T_j devuelvan los valores que cabe esperar y que los pseudorresiduos r_{ij} sean precisamente residuos de estimación.

Más de una década después, fue Tianqi Chen quien desarrolló ***XGBoost*** [6], cuyo nombre proviene de *e**X**treme **G**radient **Boost**ing*. Como su nombre indica, es una modificación del anterior algoritmo la cual resulta en una herramienta sobresaliente en la actualidad, que demuestra su dominio apareciendo en los primeros puestos de las competiciones de *Machine Learning* con datos tabulares.

Esencialmente, *XGBoost* funciona igual que su antecesor; se genera una predicción inicial la cual se va corrigiendo repetidamente con árboles de decisión entrenados con residuos y escalados por la tasa de aprendizaje ν. Aquí se pide una *Loss Function* que a parte de ser diferenciable, sea convexa. También aparecen dos nuevos parámetros de regularización positivos denotados por γ y λ. El cálculo de $F_0(x)$ es igual que en *Gradient Boosting*. En cambio, ahora T_j devuelve en su nodo hoja m el valor:

$$\omega_{mj} = \underset{\omega}{\operatorname{argmin}} \sum_{x_i \in R_{mj}} L(y_i, F_{j-1}(x_i) + \omega) + \gamma M_j + \frac{1}{2}\lambda\omega^2 \ .$$

donde M_j es el número de hojas de T_j. En esta situación, tanto para regresión como para clasificación, se recurre al desarrollo de Taylor para encontrar el ω_{mj} óptimo. Es decir,

$$\omega_{mj} = \underset{\omega}{\operatorname{argmin}} \sum_{x_i \in R_{mj}} (L(y_i, F_{j-1}(x_i)) + g_j(x_i)\omega + \frac{1}{2} h_j(x_i)\omega^2) + \gamma M_j + \frac{1}{2}\lambda\omega^2 \ , \qquad (9)$$

siendo $g_j(x_i) = -r_{ij} = \left[\frac{\partial L(y_i, F(x))}{\partial F(x)}\right]_{F(x)=F_{j-1}(x_i)}$ y $h_j(x_i) = \left[\frac{\partial^2 L(y_i, F(x))}{\partial F(x)^2}\right]_{F(x)=F_{j-1}(x_i)}$.

Sea $f(\omega)$ la función a minimizar en (9). Se tiene entonces:

$$f'(\omega) = \sum_{x_i \in R_{mj}} (g_j(x_i) + h_j(x_i)\omega) + \lambda\omega \ .$$

$$f'(\omega) = 0 \iff \lambda\omega + \sum_{x_i \in R_{mj}} g_j(x_i) + \sum_{x_i \in R_{mj}} h_j(x_i)\omega = 0 \iff$$
$$\iff \omega(\lambda + \sum_{x_i \in R_{mj}} h_j(x_i)) = -\sum_{x_i \in R_{mj}} g_j(x_i) \iff$$
$$\iff \omega = \frac{\sum\limits_{x_i \in R_{mj}} r_{ij}}{\lambda + \sum\limits_{x_i \in R_{mj}} h_j(x_i)} \ .$$

Como L es convexa, h_j es positiva, con lo cual f es una parábola con coeficiente cuadrático positivo y, consecuentemente, $\hat{\omega} = \frac{\sum_{x_i \in R_{mj}} r_{ij}}{\lambda + \sum_{x_i \in R_{mj}} h_j(x_i)}$ es mínimo de f.

Denotemos por $\mathcal{L}_j$ la *Loss Function* de T_j que sea la suma de las funciones objetivo previas para cada hoja omitiendo el término que no depende de ω. Si llamamos $g_{ij} = g_j(x_i),\ h_{ij} = h_j(x_i)$, queda:

$$\mathcal{L}_j = \sum_{m=1}^{M_j} \sum_{x_i \in R_{mj}} (g_{ij}\omega_{mj} + \frac{1}{2} h_{ij}\omega_{mj}^2) + \gamma M_j + \frac{1}{2}\lambda \sum_{m=1}^{M_j} \omega_{mj}^2 \ , \qquad (10)$$

y sustituyendo $\hat{\omega}$:

$$\hat{\mathcal{L}}_j = \gamma M_j - \frac{1}{2} \sum_{m=1}^{M_j} \frac{\left(\sum\limits_{x_i \in R_{mj}} g_{ij}\right)^2}{\lambda + \sum\limits_{x_i \in R_{mj}} h_{ij}} \ .$$

Esta función se puede utilizar análogamente a las impurezas vistas en los criterios de escisión de los CART. Si nos situamos en un nodo k, el algoritmo busca el valor s de la variable independiente X_t que maximice la siguiente función de ganancia:

$$\mathcal{G}_k = -\frac{1}{2}\left(\frac{\left(\sum_{x_i\in R_k} g_{ij}\right)^2}{\lambda + \sum_{x_i\in R_k} h_{ij}} - \frac{\left(\sum_{x_i\in R_{k1}} g_{ij}\right)^2}{\lambda + \sum_{x_i\in R_{k1}} h_{ij}} - \frac{\left(\sum_{x_i\in R_{k2}} g_{ij}\right)^2}{\lambda + \sum_{x_i\in R_{k2}} h_{ij}}\right) - \gamma \ ,$$

donde los subíndices 1 y 2 representan a los nodos hijos del nodo k. Se producirá una escisión en el nodo interno k solamente si $\mathcal{G}_k > 0$.

Cuando se afronta un problema de regresión, se escoge la misma *Loss Function* que en *Gradient Boosting*: $L(y_i, F(x)) = \frac{1}{2}(y_i - F(x))^2$. De las operaciones hechas anteriormente: $g_{ij} = F_{j-1}(x_i) - y_i$, $h_{ij} = 1$. Así, los T_j devuelven:

$$\omega_{mj} = \frac{\sum_{x_i\in R_{mj}} (y_i - F_{j-1}(x_i))}{\lambda + n_{mj}} \ .$$

Y la función de ganancia de una división queda:

$$\mathcal{G}_k = -\frac{1}{2}\left(\frac{\left(\sum_{x_i\in R_k} F_{j-1}(x_i) - y_i\right)^2}{\lambda + n_k} - \sum_{l=1}^{2} \frac{\left(\sum_{x_i\in R_{kl}} F_{j-1}(x_i) - y_i\right)^2}{\lambda + n_{kl}}\right) - \gamma \ .$$

Si se está abordando un problema de clasificación, se toma también la función de pérdida anterior: $L(y_i, F(x)) = \ln\left(1 + e^{F(x)}\right) - y_i F(x)$. Reciclando los cálculos previos: $g_{ij} = p_{i(j-1)} - y_i$, $h_{ij} = p_{i(j-1)}(1 - p_{i(j-1)})$. Entonces:

$$\omega_{mj} = \frac{\sum_{x_i\in R_{mj}} y_i - p_{i(j-1)}}{\lambda + \sum_{x_i\in R_{mj}} p_{i(j-1)}(1 - p_{i(j-1)})} \ ,$$

$$\mathcal{G}_k = -\frac{1}{2}\left(\frac{\left(\sum_{x_i \in R_k} p_{ij} - y_i\right)^2}{\lambda + \sum_{x_i \in R_k} p_{i(j-1)}(1 - p_{i(j-1)})} - \sum_{l=1}^{2} \frac{\left(\sum_{x_i \in R_{kl}} p_{ij} - y_i\right)^2}{\lambda + \sum_{x_i \in R_{kl}} p_{i(j-1)}(1 - p_{i(j-1)})}\right) - \gamma \ .$$

La función que cumple λ es disminuir la relevancia de las hojas con pocos individuos, evitando que los valores atípicos influyan, y γ sirve para controlar la profundidad de los árboles correctores logrando que con pocas iteraciones no se logren mejoras considerables.

Además de estas diferencias, *XGBoost* posee más mecanismos cuando el *dataset* al que se enfrenta es gigante con el fin de reducir el tiempo de computación lo máximo posible:

1. Emplea la estrategia de *binning* en la búsqueda de umbrales de escisión en los T_j. Si tuviese que comprobar todos los posibles s para todas las variables, la búsqueda del óptimo sería demasiado lenta. Esta estrategia consiste en considerar solamente algunos cuantiles (normalmente unos 33) como posibles s. Estos percentiles son ponderados: a cada x_i se le asocia un peso, que es h_{ij}, y cada percentil contiene un número de individuos que hace que la suma de pesos sea igual en cada uno. En regresión son percentiles corrientes, pero en clasificación los individuos peor clasificados tienen más importancia que los que tienen una probabilidad estimada cercana a la verdadera, lo cual implica que los cuantiles no estén distribuidos homogéneamente. A parte de esto, también es una opción no considerar todas las variables explicativas, sino solo un subconjunto aleatorio en cada árbol y entrenarlo con una submuestra aleatoria para rebajar el tiempo de computación.

2. Es capaz de trabajar con datos nulos. En el proceso de escisión de nodos de los T_j, cuando se comprueban todos los posibles umbrales de escisión s de X_t, se separan los datos en dos tablas: una con los individuos que tengan valores observados y otra en la que estén las observaciones con celdas vacías de X_t. Se realizan las comprobaciones de s atendiendo a la primera tabla y se calcula $\mathcal{G}_k$ añadiendo los valores de la segunda tabla a cada nodo hijo estableciendo así una ruta para futuros individuos que haya que predecir con valores nulos.

3. Tiene en cuenta el *hardware* del ordenador aprovechando la memoria caché para almacenar g_{ij} y h_{ij} y lograr hacer todos los cálculos pertinentes lo más rápido posible. Si el conjunto de datos excede la capacidad de la caché y la memoria RAM, se comprime y se descomprime en vez de ser leído directamente del disco duro; en caso

de disponer de más de un disco duro, se fragmenta y cada unidad de almacenamiento se ocupa de una parte de los datos.

Lo malo de estos modelos es que su coste computacional es muy elevado, que son complejos, se pierde toda la interpretabilidad y son propensos a sobreajustarse aunque esto se puede compensar gracias a los parámetros de regularización; en compensación, la precisión que alcanzan es muy elevada.

3. Algoritmos híbridos

En este apartado se van a exponer varios algoritmos novedosos que combinan modelos basados en árboles de decisión y Regresión Lineal o Logística buscando aprovechar las fortalezas de cada técnica, por un lado, el poder predictivo de los métodos paramétricos y su capacidad de extrapolación, y por otro, la aptitud de identificar patrones subyacentes no lineales en diferentes subgrupos de una población.

3.1. Linear Trees

Los *Linear Trees* [12], estructuralmente, son iguales que los CART. La principal diferencia es la forma de dar las predicciones en los nodos hoja. En el caso de regresión, estos nuevos modelos ajustan una Regresión Lineal a la submuestra contenida en la región correspondiente a cada hoja en lugar de devolver directamente la media de tal submuestra; en el caso de clasificación, en vez de predecir con la moda del subconjunto de datos, se ajusta una Regresión Logística.

Visto como función, se define como sigue, de manera análoga a (6):

$$\begin{aligned} LT &: D \longrightarrow E \\ LT(x) &= \sum_{m=1}^{M} f_m(x) I_{R_m} \ , \end{aligned}$$

donde I_{R_m} es la función indicadora definida en (7) y f_m es el modelo de Regresión Lineal/Logística ajustado en la hoja m.

Esta manera de predecir conlleva que a la hora de escindir los nodos también existan diferencias. Ahora cuando se calculan los SSE o los Ínidices Gini de los hijos, en lugar de considerar las medias o las proporciones, respectivamente, hay que ajustar para cada posible umbral de escisión de cada variable una Regresión Lineal o Logística y tomar el

valor de salida de este modelo. Esto significaría que para dividir un nodo k habría que entrenar $2 \cdot p \cdot n_k$ Regresiones, lo cual sería computacionalmente muy ineficiente. Para solucionar esto, se emplea la estrategia de *binning*, reduciendo de esta manera el coste computacional considerablemente.

Visualicemos un ejemplo gráfico con una serie temporal ficticia para comparar el desempeño entre estos árboles y los CART:

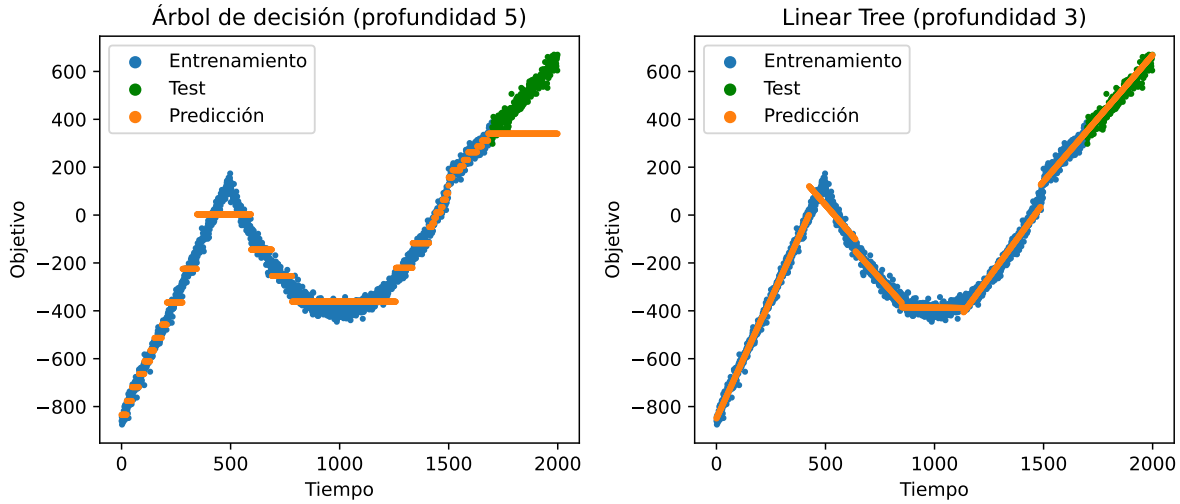


Figura 5: Comparación de predicciones entre CART y *Linear Tree* de regresión sobre la serie temporal ficticia.

Mientras que el árbol de decisión no llega a ajustarse muy bien a los datos de entrenamiento con profundidad 5, el árbol lineal se ajusta prácticamente a la perfección con 3 niveles de profundidad. Por otro lado, por más profundidad que se le permita al CART, nunca va a ser capaz de extrapolar la tendencia ascendente del entrenamiento al test; en cambio, el *Linear Tree* sí que logra esto. Hay que tener en cuenta que es gracias a que la parte final de los datos de entrenamiento cae en una misma hoja y la variable objetivo en el test sigue aumentando. Si el árbol no fuese suficientemente profundo, la regresión lineal de esta hoja no sería tan acertada y las predicciones podrían ser menos precisas que las del CART; o también si la tendencia en el test pasase a ser descendente, las predicciones del árbol lineal serían mucho peores que las del CART (siendo estas también bastante inexactas). Veamos con otro *dataset* de prueba cómo actúan cuando y es binaria:

En este ejemplo, el *Linear Tree* con tan solo una división capta bien los patrones que sigue la muestra; en cambio, el CART necesita una profundidad de 8 niveles para lograr ajustarse a los datos de entrenamiento, y, como veíamos en la Figura 5, tampoco aprende

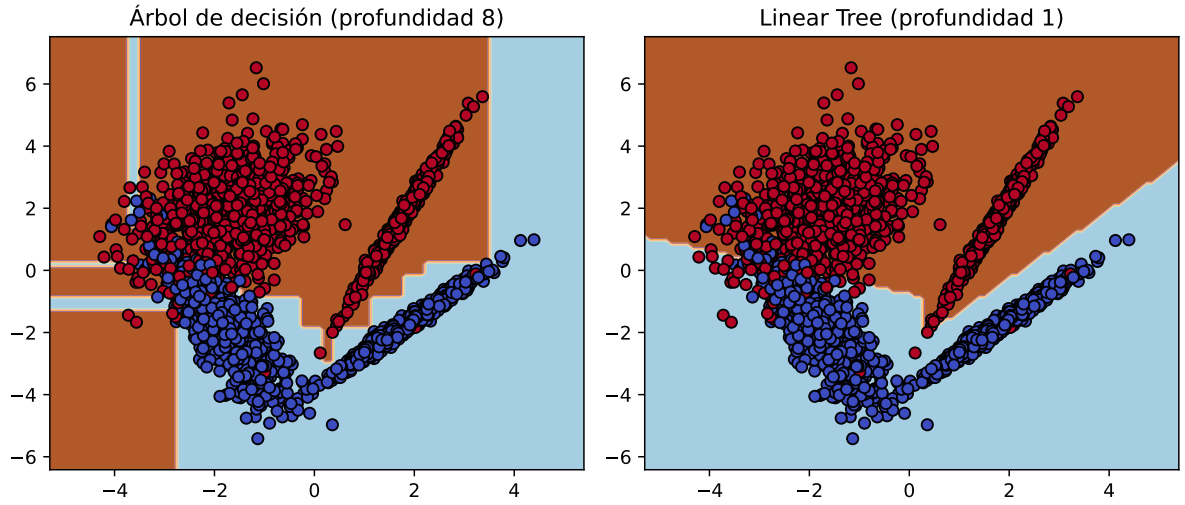


Figura 6: Comparación de predicciones entre CART y *Linear Tree* de clasificación sobre el *dataset* ficticio.

a detectar tendencias que no sean paralelas a los ejes. Además, tanta profundidad conlleva hacer algunas divisiones que sobreajustan el modelo.

Generalmente, los *Linear Trees* consiguen ser bastante precisos con menos profundidad que los CART sin perder la faceta de la fácil interpretabilidad. La mejora más sustancial es la capacidad de extrapolar las predicciones. En contrapartida, el entrenamiento conlleva una cantidad de tiempo mucho mayor y los datos requieren ser tratados antes de ser utilizados.

3.2. Linear Random Forest

Con la misma inquietud que surge *Random Forest* en relación a los CART, proponemos una versión propia donde los estimadores base son *Linear Trees*. Estos son ensamblados a través de la herramienta de *bagging* [1] de *scikit-learn*. Como función, tienen la siguiente expresión:

$$LRF(x) = \frac{1}{N}\sum_{j=1}^{N} LT_j(x) \ ,$$

y, al igual que en RF, cada árbol busca sus divisiones entre una porción aleatoria de variables sobre la submuestra dada por el *bootstrap*. También las Regresiones de las hojas se ajustan con esas mismas variables.

3.3. Regression-Enhanced Random Forest

El algoritmo ***R**egression-**E**nhanced **R**andom **F**orest* (RERF) [14] se puede considerar una evolución del *Random Forest* tradicional buscando mejorar la precisión de las predicciones valiéndose de la fortaleza predictiva de los métodos de Regresión y consiguiendo poder extrapolar las predicciones o se puede ver como una mejora de la Regresión Lineal pretendiendo modelar la nube ruidosa de errores con un *Random Forest*.

Los pasos de los que consta RERF son los siguientes:

1. Ajustar un modelo g de Regresión con y como objetivo y calcular el vector de residuos $r = (r_1, \ldots, r_n)$.
2. Entrenar un modelo de *Random Forest* RF con r como variable objetivo.
3. Repetir los pasos 1 y 2 variando los hiperparámetros de RF (número de árboles, profundidad máxima, etc.) y de la Regresión (parámetro λ y función p de regularización) y seleccionar los óptimos.
4. Devolver $\hat{y}_i = g(x_i) + RF(x_i)$.

En caso de que y sea continua:

$$r_i = y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}) \quad \text{y} \quad \hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip} + RF(x_i) \ ,$$

y si es binaria:

$$r_i = y_i - \sigma(\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}) = \quad \text{y} \quad \hat{y}_i = \sigma(\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}) + RF(x_i) =$$
$$= y_i - \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}}} \qquad = \frac{e^{\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}}}{1 + e^{\hat{\beta}_0 + \hat{\beta}_1 x_{i1} + \cdots + \hat{\beta}_p x_{ip}}} + RF(x_i) \ .$$

La principal mejora respecto a un *Random Forest* es poder predecir fuera del rango de valores de entrenamiento; respecto a la Regresión, la capacidad de capturar relaciones complejas entre variables. El tiempo de computación, aunque obviamente es superior a cada uno de los algoritmos por separado, no es un gran problema ya que tan solo se ajusta un modelo de regresión y un *Random Forest*. La peor parte es que se pierde la interpretabilidad de la Regresión y que los datos requieren ser preprocesados antes de ser utilizados.

3.4. Explainable Boosted Regression

***E**xplainable **B**oosted **L**inear/**L**ogistic **R**egression* (EBLR) [9] es un instrumento que actúa sobre el conjunto de datos enriqueciéndolo con nuevas variables independientes. Este algoritmo está diseñado para capturar relaciones no lineales dentro de la rigidez de los modelos de Regresión sin perder nada de interpretabilidad.

Las etapas a seguir en este algoritmo son:

1. Seleccionar el modelo base g de Regresión, decidir la forma de calcular los errores, crear una copia $X^{(0)}$ del *dataset* de partida y elegir el número de nuevas variables N que se van a crear.

2. Hacer desde $j = 1$ hasta N:

   a) Ajustar g con $X^{(j-1)}$ e y.

   b) Hallar el vector de residuos de predicción $r = (r_1, \ldots, r_n)$.

   c) Entrenar un árbol de regresión T_j con r como objetivo.

   d) Detectar la hoja con el mayor valor absoluto y codificar la ruta hasta esa hoja para generar una nueva variable binaria X_{p+j}.

   e) Agregar la nueva variable al conjunto de datos generando $X^{(j)} = X^{(j-1)} \cup X_{p+j}$.

3. Ajustar g con $X^{(N)}$ e y y devolver sus estimaciones.

De esta forma, la nueva variable creada en cada paso identifica el grupo de observaciones con características similares que mayores errores sufre en las predicciones dadas por la Regresión. Así, el modelo de Regresión del siguiente paso recibe más información y puede corregir sus propias estimaciones previas.

Los modelos base g son, dependiendo el problema, modelos de Regresión Lineal o Logística y pueden incluir términos de regularización. Los errores r_i en el caso de regresión se suelen calcular de manera lineal: $y_i - g(y_i)$, aunque se pueden calcular también los errores al cuadrado, o según criterios exponenciales o de Poisson; en el caso de clasificación, lo más usual es asignar un error de 1 si la clase predicha no es la real y un 0 si la predicción de clase es correcta.

La manera de crear las variables nuevas es señalar las observaciones con predicciones menos precisas asignando 1 a la submuestra de la hoja con mayor valor absoluto y 0 al resto de individuos.

Veamos un breve ejemplo con $N = 2$. Supongamos que disponemos del siguiente *dataset* $(X^{(0)}, y)$:

Fecha	Día de la semana	Promoción	Facturación
736330	Domingo	No	8165.73
736331	Lunes	Sí	6230.86
736332	Martes	No	5226.25
⋮	⋮	⋮	⋮

Tabla 1: Encabezado del conjunto de datos de ejemplo.

En la parte de entrenamiento tenemos registros diarios de 5 años y medio, y en el test, de dos meses. En primer lugar, ajustamos un modelo de Regresión Lineal g sobre $(X^{(0)}, y)$ el cual da las predicciones para el test de la Figura 7.

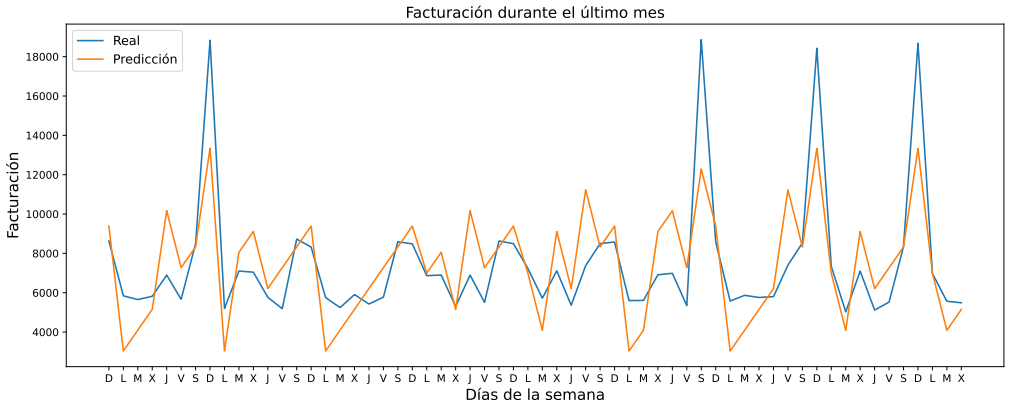
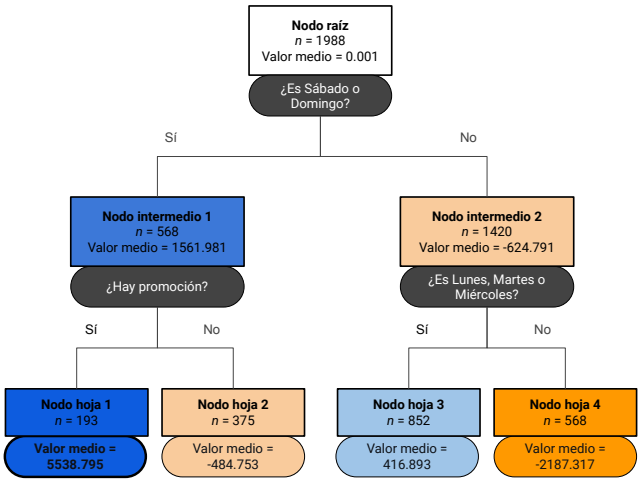


Figura 7: Serie temporal de test y predicción inicial.

Ahora se entrena T_1 con los residuos de las predicciones sobre los datos de entrenamiento como objetivo y se genera la nueva variable X_{3+1} marcando con un 1 a los registros que acaben en la hoja 1, que es la que mayor valor medio absoluto tiene, como se ve en la Figura 8.

En este punto se vuelve a entrenar g pero con $X^{(1)}$ en lugar de $X^{(0)}$. Este nuevo modelo proporciona las predicciones en el test que se muestran en la Figura 9.

De este renovado modelo g se recalcula el vector de residuos sobre el conjunto de entrenamiento y se entrena el árbol de regresión T_2 dando lugar a otra nueva variable



Fecha	Día de la semana	Promoción	Nueva variable 1
736330	Domingo	No	0
736331	Lunes	Sí	0
736332	Martes	No	0
⋮	⋮	⋮	⋮
738268	Sábado	Sí	1
738269	Domingo	Sí	1
⋮	⋮	⋮	⋮
738316	Viernes	No	0
738317	Sábado	No	0

Figura 8: Árbol de decisión T_1 y conjunto de datos actualizado $X^{(1)}$.

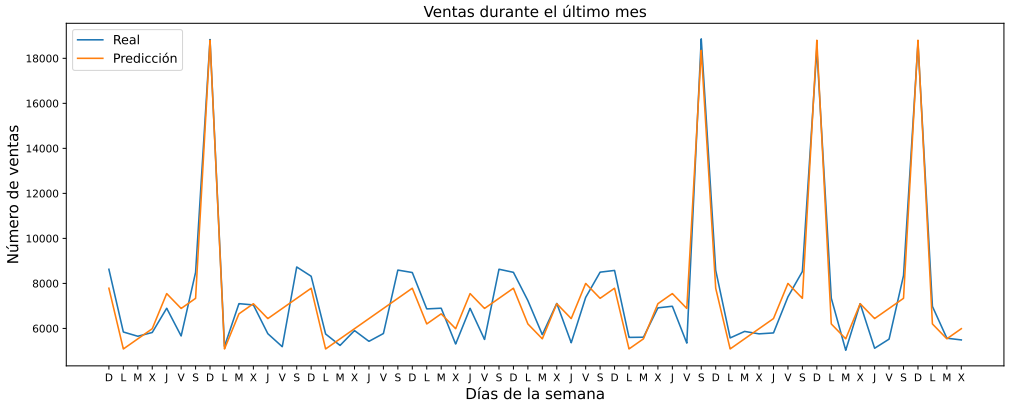
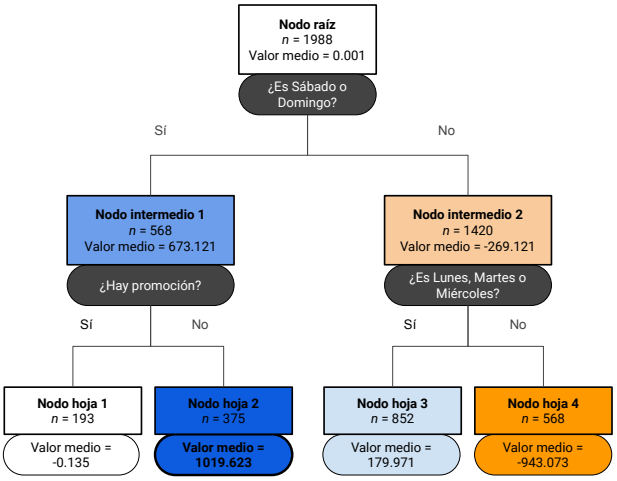


Figura 9: Serie temporal de test y segunda predicción.

explicativa y al nuevo conjunto de datos $X^{(2)}$ que se pueden ver en la Figura 10.

Finalmente, se reajusta g sobre $(X^{(2)}, y)$ y este modelo final predice en el test como se puede observar en la Figura 11.

Notar que T_1 y T_2 se rigen por las mismas restricciones, pero en general no tiene por qué suceder esto. Por otro lado, cabe destacar que las dos nuevas variables son relevantes. Esto lo podemos comprobar a simple vista, o también nos podemos fijar en que los valores medios de los nodos de T_2 son menores en valor absoluto que los valores medios de los nodos de T_1, aprovechando que las reglas de escisión son las mismas en sendos árboles y por tanto las submuestras en cada nodo también lo son.



Fecha	Día de la semana	Promoción	Nueva variable 1	Nueva variable 2
736330	Domingo	No	0	1
736331	Lunes	Sí	0	0
736332	Martes	No	0	0
⋮	⋮	⋮	⋮	⋮
738268	Sábado	Sí	1	0
738269	Domingo	Sí	1	0
⋮	⋮	⋮	⋮	⋮
738316	Viernes	No	0	0
738317	Sábado	No	0	1

Figura 10: Árbol de decisión T_2 y conjunto de datos actualizado $X^{(2)}$.

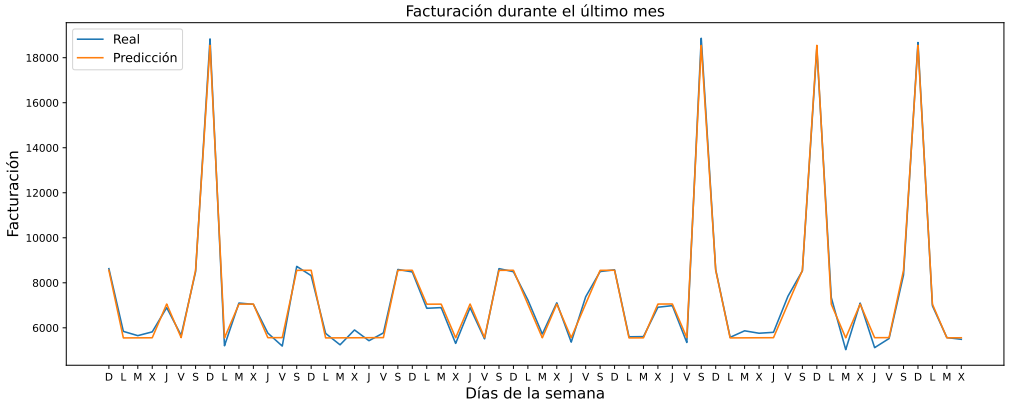


Figura 11: Serie temporal de test y predicción final.

3.5. Piecewise Linear Gradient Boosting

En el año 2019, tres investigadores chinos en ciencias de la computación desarrollaron ***P**iecewise **L**inear **G**radient **B**oosting* (PLGB) [13], dando otro paso en la rama del *boosting*. Este algoritmo continúa con el planteamiento inicial de sus predecesores de ayudarse de las derivadas reiteradamente para encontrar el óptimo de una función que cuantifica los errores de estimación cometidos, pero la diferencia principal es que los árboles T_j pasan de ser CART a ser *Linear Trees*. Al igual que *XGBoost*, aproxima la función de pérdida con el desarrollo de Taylor de segundo orden, por ello, cada nuevo árbol devuelve un valor como en (9), pero en lugar de ser constante en cada hoja m, viene dado por un modelo

lineal f_{mj}. Veamos en primer lugar qué valores devuelven los T_j.

Se sobreentiende que estamos todo el rato trabajando en el árbol de la iteración j-ésima, por ello se omite j de todos los subíndices para no sobrecargarlos. Sea f_m el modelo lineal que se ajusta en la hoja m cuyo vector de coeficientes es $\beta_m = (\beta_{m0}, \beta_{m1}, \ldots, \beta_{mp_m})^T$. Lo que en XGB era ω_{mj} ahora pasa a ser $f_m(x_i)$, y la función objetivo (10) en términos de β_m, restringiéndonos al nodo hoja m, queda:

$$\mathcal{L}_m = \sum_{x_i \in R_m} (g_i f_m(x_i) + \frac{1}{2} h_i f_m(x_i)^2) + \frac{\lambda}{2} ||\beta_m||_2^2 =$$

$$= \sum_{x_i \in R_m} \left(g_i(\beta_{m0} + \sum_{l=1}^{p_m} \beta_{ml} x_{ik_{ml}}) + \frac{1}{2} h_i(\beta_{m0} + \sum_{l=1}^{p_m} \beta_{ml} x_{ik_{ml}})^2 \right) + \frac{\lambda}{2} ||\beta_m||_2^2 \ ,$$

donde $\{k_{ml}\}_{l=1}^{p_m}$ es el conjunto de índices de variables regresoras elegidas para el modelo f_m. Más adelante veremos cómo se eligen estos índices.

Reescribamos $\mathcal{L}_m$ por comodidad y hallemos $\hat{\beta}_m$. Sean $H = \begin{pmatrix} h_1 & 0 & \ldots & 0 \\ 0 & h_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & h_n \end{pmatrix}$ y $g = (g_1, \ldots, g_n)^T$. Denotamos por I_{p_m} la matriz identidad de orden p_m y llamamos respectivamente H_s y g_s a la submatriz y al subvector de H y de g que contienen los h_i y los g_i con i tal que $x_i \in R_m$. Sea también X_s la matriz de datos con la primera columna de unos y las demás con las variables $\{k_{ml}\}_{l=1}^{p_m}$. Con esto:

$$\mathcal{L}_m = g_m^T X_m \beta_m + \frac{1}{2} \beta_m^T (X_m^T H_m X_m + \lambda I_{p_m}) \beta_m \ . \tag{11}$$

Y podemos calcular $\hat{\beta}_m$ analíticamente, de manera similar a cómo se obtiene en la Regresión Lineal:

$$\frac{\partial \mathcal{L}_m}{\partial \beta_m} = (g_m^T X_m)^T + (X_m^T H_m X_m + \lambda I_{p_m}) \beta_m = 0 \implies$$
$$\implies \hat{\beta}_m = -(X_m^T H_m X_m + \lambda I_{p_m})^{-1} X_m^T g_m \ .$$

Con lo cual, sustituyendo $\hat{\beta}_m$ en (11), se consigue el valor mínimo para la *Loss Function* en la hoja m:

$$\hat{\mathcal{L}}_m = -\frac{1}{2} g_m^T X_m (X_m^T H_m X_m + \lambda I_{p_m})^{-1} X_m^T g_m \ .$$

Por lo tanto, cada T_j aporta a la predicción final de un individuo x_i el siguiente valor:

$$f_{m'}(x_i) = \hat{\beta}_{m'0} + \hat{\beta}_{m'1} x_{ik_{m'1}} + \cdots + \hat{\beta}_{m'p_{m'}} x_{ik_{m'p_{m'}}} \ ,$$

siendo m' el nodo terminal donde cae x_i.

Al igual que en *XGBoost*, el criterio de escisión de un nodo k se basa en $\hat{\mathcal{L}}_k$. Para sus hipotéticos nodos hijo k_1 y k_2 se definen análogamente sus respectivas H, g y X. Se han de calcular los $\hat{\beta}$ y $\hat{\mathcal{L}}$ y se escindirá el nodo con la pretensión de maximizar $\mathcal{G}_k = \hat{\mathcal{L}}_{k_1} + \hat{\mathcal{L}}_{k_2} - \hat{\mathcal{L}}_k$ siempre y cuando este resultado sea positivo.

Al igual que los *Linear Trees*, simplemente escindir un nodo conlleva ajustar demasiadas Regresiones, *i.e.* calcular $\hat{\beta}$, por ello se emplea la estrategia de *binning* también. Además de aliviar la cantidad de trabajo disminuyendo el número de umbrales de escisión candidatos, se prescinde de algunas variables para acelerar los cálculos matriciales. Para esto se hace uso de la selección incremental de variables, que consiste en considerar en cada nodo únicamente como variables regresoras las variables de escisión de sus nodos antecesores, es decir, si el modelo lineal del nodo s (con el cual se ha logrado la división de su padre) es $f_s(x_i) = \beta_{s0} + \sum_{l=1}^{p_s} \beta_{sl} x_{ik_{sl}}$, entonces el modelo lineal de s_1 es $f_{s_1}(x_i) = \beta_{s_10} + \sum_{l=1}^{p_s} \beta_{s_1 l} x_{ik_{sl}} + \beta_{s_1(p_s+1)} x_{it}$, siendo t la variable de escisión del nodo s. Esta agregación de variables se hace hasta alcanzar un límite preestablecido.

La tercera propuesta para simplificar los cálculos atiende a los parámetros de f_{s_1}. Hay tres enfoques posibles:

- **Ajuste aditivo:** tan solo se calculan β_{s_10} y $\beta_{s_1(p_s+1)}$. Los $\beta_{s_1 l}$ con $l \in \{1, \ldots, p_s\}$, se corresponden con los del nodo s. Así se ahorra la mayor dosis de cálculos.

- **Ajuste totalmente correctivo:** el vector de coeficientes de f_{s_1} se calcula desde cero. Esta forma es la menos eficiente, pero a cambio se obtiene el mínimo valor de la función de pérdida.

- **Ajuste semiaditivo:** el nodo hijo toma $\sum_{l=1}^{p_s} \beta_{s_1 l} x_{ik_{sl}}$ como variable y aprende tres parámetros: β_{s_10}, $\beta_{s_1(p_s+1)}$ y un parámetro de escala μ, resultando en la expresión: $f_{s_1}(x_i) = \beta_{s_10} + \mu \left(\sum_{l=1}^{p_s} \beta_{s_1 l} x_{ik_{sl}} \right) + \beta_{s_1(p_s+1)} x_{it}$. Este planteamiento combina los otros dos y logra un buen equilibrio entre precisión y eficiencia.

De manera equivalente se razona para el nodo hijo 2.

Si la variable y es continua, la *Loss Function* que se elige normalmente es la misma que en los algoritmos precursores: $L(y_i, F(x)) = \frac{1}{2}(y_i - F(x))^2$. Recopilando los cálculos realizados en la sección 2.3.3, el vector de coeficientes del modelo de Regresión Lineal que da las predicciones en la hoja m de T_j es:

$$\hat{\beta}_{mj} = (X_m^T X_m)^{-1} X_m^T \begin{pmatrix} y_{m_1} - F_{j-1}(x_{m_1}) \\ y_{m_2} - F_{j-1}(x_{m_2}) \\ \vdots \\ y_{m_{n_m}} - F_{j-1}(x_{m_{n_m}}) \end{pmatrix},$$

suponiendo que $\lambda = 0$. Es la misma expresión que la obtenida en la sección 2.1, pero con la variable objetivo siendo los residuos r_{ij} tal y como cabía esperar tras ver el funcionamiento de GB y de XGB.

Cuando nos encontramos en el caso de que y es binaria, la función de pérdida vuelve a ser $L(y_i, F(x)) = \ln\left(1 + e^{F(x)}\right) - y_i F(x)$, y todo el razonamiento previo es válido porque también se trabaja con el logaritmo del *odds ratio*, lo cual es funcional con modelos lineales. Los coeficientes del modelo asociado a $\ln\left(\frac{p}{1-p}\right)$ con $\lambda = 0$ vienen dados por:

$$\hat{\beta}_{mj} = (X_m^T H_{mj} X_m)^{-1} X_m^T \begin{pmatrix} y_{m_1} - p_{m_1(j-1)} \\ y_{m_2} - p_{m_2(j-1)} \\ \vdots \\ y_{m_{n_m}} - p_{m_{n_m}(j-1)} \end{pmatrix}, \quad \text{siendo}$$

$$H_{mj} = \begin{pmatrix} p_{m_1(j-1)}(1 - p_{m_1(j-1)}) & 0 & \dots & 0 \\ 0 & p_{m_2(j-1)}(1 - p_{m_2(j-1)}) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & p_{m_{n_m}(j-1)}(1 - p_{m_{n_m}(j-1)}) \end{pmatrix}.$$

Esta matriz actúa ponderando las columnas de $X_m^T X_m$, lo cual puede traer problemas de mal condicionamiento de esta matriz cuando las probabilidades estimadas son cercanas a las clases reales, por ello es conveniente elegir $\lambda > 0$.

Cabe recordar que para obtener los resultados finales como números comprendidos entre 0 y 1 para interpretarlos como probabilidades hay que transformar el resultado con la función sigmoide.

El mayor logro de este algoritmo es la gran velocidad de computación conseguida con las optimizaciones a nivel informático que implementaron los autores a la hora de construir los histogramas que se usan para el *binning* en CPUs modernas.

Cada individuo del *dataset* tiene asignado un ID único entre 1 y n. Para cada hoja m, se crea una lista llamada $index_m$ de longitud n_m, que registra el ID, denotado id, de cada individuo en la hoja. Para cada variable X_j, se crea otra lista llamada bin_j de longitud n, cuyo elemento id-ésimo contiene en qué *bin* del histograma de X_j cae el *datapoint* con ID id. Con esto se construye otra lista $hist_{j,m}$ que representa un histograma de la variable X_j en la hoja m, y cada uno de sus elementos, que representa un *bin* del histograma, guarda los valores $h_i x_i x_i^T$ y $g_i x_i$ de los x_i que caigan en cada *bin*, que son los cálculos necesarios para el ajuste de las Regresiones.

Para acelerar la creación de esta última lista, se introduce la paralelización SIMD (*Single Instruction Multiple Data*), que es una forma de computación en la cual varios cálculos pueden realizarse simultáneamente. Cuando se está generando $hist_{j,m}$, al acceder a bin_j hay que omitir los individuos que no están en la hoja m lo que causa fallos de caché frecuentes debido a un acceso discontinuo a esta lista. Para solventar este problema, se reorganizan las estructuras de los datos. Se propone la creación de $leafBin_{m,j}$, otro nuevo *array* que almacene los índices de los *bins* de $hist_{j,m}$ solamente para las observaciones del nodo hoja m. Este cambio provoca que cada índice de *bin* se encuentre en un *byte* y que el acceso a $leafBin_{m,j}$ sea continuo, consiguiendo así reducir los fallos de caché y el uso de memoria RAM. Entonces ahora $hist_{j,m}$ se construye atendiendo tan solo a la lista $leafBin_{m,j}$, agregando en cada paso varios elementos gracias al paralelismo a nivel de datos.

Esta estrategia se usa en todos los nodos en la tarea de escindirse, no solo en las hojas. Para el nodo raíz k_0, $leafBin_{k_0,j}$ y bin_j coinciden. Cuando un nodo k se separa en k_1 y k_2, $leafBin_{k,j}$ también se divide en $leafBin_{k_1,j}$ y $leafBin_{k_2,j}$, respectivamente. Esta operación se ha de realizar para todas las variables X_j, y la manera de optimizarla es manipulando los bits.

Dividir $leafBin_{k,j}$ significa obtener los índices de los *bins* de la propia lista y almacenarlos correspondientemente en $leafBin_{k_1,j}$ o $leafBin_{k_2,j}$. Para ello, se necesita previamente conocer en qué nodo hijo acaba cayendo cada individuo de k. Esta información se almacena en la lista $bitVector$ en forma de valores booleanos. Haciendo uso de herramientas de extracción de bits paralelizada, con solamente dos instrucciones se construyen las nuevas listas.

4. Comparación entre modelos

Con la finalidad de mostrar la eficacia de estos nuevos modelos híbridos que han sido explicados, vamos a tratar de sintetizar la mayor cantidad de información posible de todo el trabajo práctico realizado paralelamente a esta memoria en esta sección. La Figura 12 muestra una comparativa de todos los algoritmos tratados en los seis conjuntos de datos de ejemplo. El top 1 del ranking representa el modelo que mejor rendimiento ha mostrado, es decir, el que mayor AUC o menor RMSE ha obtenido en el conjunto de datos de test y el top 9, lo contrario.

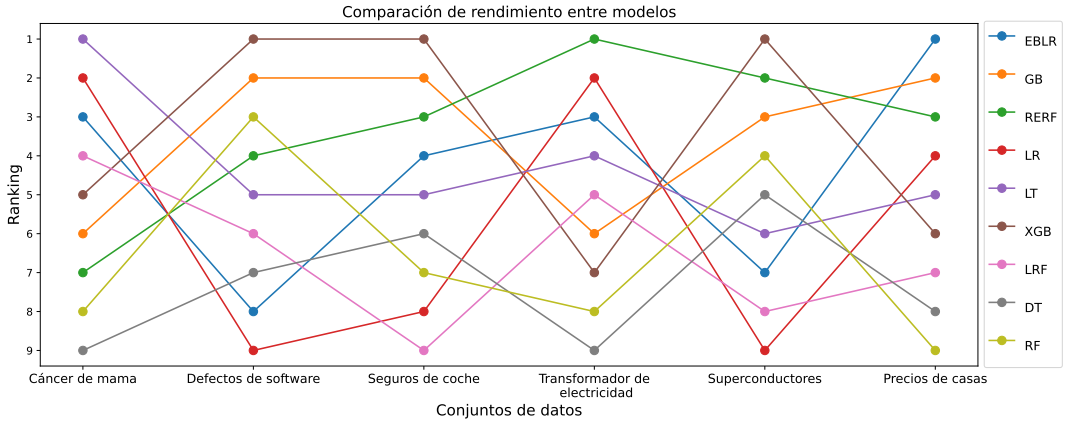


Figura 12: Comparación de rendimiento en el conjunto de testeo.

Cada modelo de Regresión Lineal/Logística (LR), cada árbol de decisión CART (DT), etc. tiene diferentes hiperparámetros para cada *dataset* que se han elegido como se explica en la sección 1.1. Podemos observar que los modelos de Regresión paramétrica no son siempre la mejor opción, ya que necesitan de relaciones lineales para ser realmente efectivos. Como era de esperar, los CART, que son los modelos más simples, generalmente son superados por el resto de modelos junto a los *Random Forest* (RF) y a los *Linear Random Forest* (LRF), que por construcción tampoco pueden llegar a ser los más precisos, a parte de no estar optimizados. Los modelos de *boosting* (GB, XGB) cumplen consistentemente con su cometido de lograr altas precisiones gracias a las estrategias de optimización que usan. En cuanto a los híbridos, podemos ver que en los casos que la Regresión hace un buen papel, los árboles lineales (LT) y la Regresión impulsada (EBLR) también rinden bien, pero cuando la Regresión no es del todo efectiva, estos tampoco son la mejor opción; sin embargo, los RERF tienen un desempeño muy bueno consistentemente.

En cuanto a la diferencia de precisión lograda entre el conjunto de entrenamiento y el de

test, estos nuevos algoritmos híbridos son tan buenos, si no mejores, que los predecesores en la tarea de generalizar sus predicciones, especialmente gracias su capacidad de extrapolar. La información detallada de las métricas obtenidas y de los hiperparámetros de los modelos se encuentra en el Anexo (B).

Hay que tener en cuenta que los resultados que aquí se muestran son obtenidos a partir de modelos que buscan el máximo AUC o el mínimo RMSE en el test dentro de unos límites de *overfitting* pero sin tener en cuenta el coste de computación. Visualicemos por tanto la otra cara de estos modelos, que es el tiempo que han requerido para lograr estos resultados, en la Figura 13.

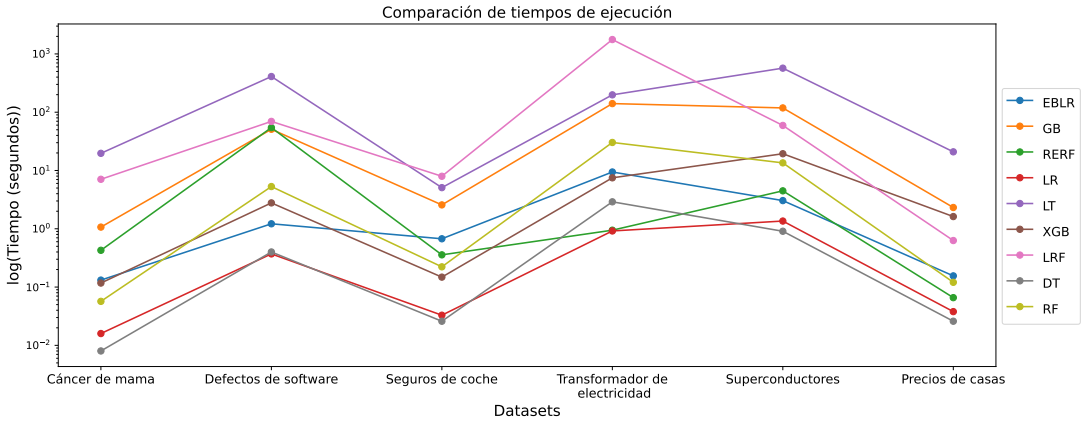


Figura 13: Comparación de tiempos de entrenamiento.

Claramente *Linear Tree* es el algoritmo más costoso computacionalmente seguido de *Gradient Boosting* y de *Linear Random Forest*. Los más rápidos son CART, como era de esperar, y las Regresiones Lineales y Logísticas. Los otros 4 algoritmos tienen altibajos en este aspecto pero se encuentran siempre entre en la media de los tiempos de ejecución.

Notar que los modelos de PLGB no aparecen en estos gráficos, esto es debido a la incompatibilidad con el sistema operativo; en el Anexo (C) se da fe de su valía.

Con esto, podemos decir que LT y LRF no están preparados para ser realmente competentes, pero, por otro lado, RERF y EBLR se codean con *XGBoost* tanto en precisión como en tiempo de computación, lo cual los convierte en unas más que plausibles alternativas en el mundo del *Machine Learning* y los datos tabulares.

5. Conclusión

A lo largo del trabajo se han tratado diversos algoritmos de *Machine Learning* supervisado desde una perspectiva teórica, comenzando por los conocidos métodos de Regresión, pasando por los árboles de decisión y sus evoluciones pertenecientes a las familias de *bagging* y *boosting* hasta llegar a los emergentes modelos híbridos que combinan enfoques paramétricos y no paramétricos, dando pie a nuevos horizontes de exploración en este campo.

El punto de vista teórico ha sido transportado al mundo real realizando simulaciones de seis casos de uso a modo de respaldo. Durante el proyecto práctico se han desempeñado diferentes roles del marco de la ciencia de datos, como la ingeniería de datos, realizando las tareas de preprocesamiento de los *datasets*, el análisis de datos, visualizando e interpretando los datos para extraer información útil o la especialidad en aprendizaje automático, haciendo uso de todos los algoritmos, entrenándolos y retocando sus parámetros.

La idea de hacer crecer la cantidad de algoritmos de *Machine Learning* por medio de la composición de algoritmos basados en árboles de decisión y de métodos de Regresión es, sin lugar a dudas, acertada. Tal y como se ha visto, se pueden lograr tanto modelos que alcanzan grados de exactitud elevadísimos como modelos sencillamente interpretables cuya finalidad sea resultar útil a la hora de la toma de decisiones.

Algoritmo	Implementación Python	Parámetros principales	Control *Overfitting*	Coste Computacional	Consideraciones
Linear Tree	lineartree, scikit-learn: - LinearTreeRegressor - LinearTreeClasifier - LinearRegression, Ridge, Lasso, ElasticNet - LogisticRegression	- *base_ estimator*: modelo de Regresión que se ajusta en las hojas. - *max_ depth*: profundidad máxima del árbol. - *min_ samples_ leaf*: número mínimo de individuos en un nodo hoja. - *min_ samples_ split*: número mínimo de individuos para dividir un nodo. - *min_ impurity_ decrease*: umbral de descenso de impureza para dividir un nodo. - *max_ bins*: número máximo de percentiles para el *binning*.	- Usar como *base_ estimator* un modelo con penalización. - Disminuir *max_ depth*, *max_ bins*. - Aumentar *min_ samples_ leaf*, *min_ impurity_ decrease*, *min_ samples_ split*.	Alto	- Útil para hacer una exploración inicial de los datos. - Se puede graficar el árbol y los coeficientes de los modelos de las hojas. - Fácilmente interpretable. - Puede extrapolar predicciones.
Linear Random Forest	lineartree, scikit-learn: - LinearTreeRegressor - LinearTreeClasifier - BaggingRegressor - BaggingClassifier	- *estimator*: el *Linear Tree* con todos los parámetros correspondientes. - *n_ estimators*: número de árboles. - *max_features*: número de variables con las que entrenar cada árbol (poner $\lfloor\sqrt{p}\rfloor$ para que sea como por defecto en un *Random Forest*).	- Controlar el *overfitting* de los árboles generados mediante las técnicas de *Linear Tree*. - Disminuir *max_features*.	Alto	- Cada árbol se entrena con distintas variables aleatorias. - Se disminuye la varianza de los errores entre entrenamiento y test. - Puede extrapolar predicciones.
Regression-Enhanced Random Forest	lineartree, scikit-learn: - LinearForestRegressor - LinearForestClassifier - LinearRegression, Ridge, Lasso, ElasticNet - LogisticRegression	- *base_ estimator*: modelo de Regresión que se ajusta sobre la variable objetivo. - *n_ estimators*: número de árboles. - *max_ depth*: profundidad máxima de los árboles. - *bootstrap*: si se utilizan submuestras dadas por *bootstrap* (por defecto sí).	- Disminuir la profundidad de los árboles generados. - Elegir un modelo de Regresión con penalización.	Medio	- Mejora la precisión de la Regresión ante datos ruidosos.
Explainable Boosted Regression	lineartree, scikit-learn: - LinearBoostRegressor - LinearBoostClassifier - LinearRegression, Ridge, Lasso, ElasticNet - LogisticRegression	- *base_ estimator*: modelo de Regresión base. - *n_ estimators*: número de etapas de *boosting* a realizar. Corresponde al número de nuevas variables generadas. - *max_ depth*: profundidad máxima de los árboles. - *loss*: función usada para calcular los residuos.	- Disminuir la profundidad de los árboles generados. - Disminuir *n_ estimators*.	Medio-Bajo	- Igual de interpretable que la Regresión Lineal/Logística. - Se captan patrones no lineales.
Piecewise Linear Gradient Boosting	gbdtpl	- *objective*: el tipo de problema. - *num_ trees*: número de árboles. - *max_ depth*: profundidad máxima de los árboles. - *learning_ rate*: tasa de aprendizaje. - *num_ bins*: número de percentiles para el *binning*. - *max_ var*: umbral de variables para la selección incremental de variables. - *leaf_ type*: el tipo de ajuste del modelo en las hojas. - *l1_ reg*, *l2_ reg*: coeficientes de regularización para los modelos de Regresión.	- Disminuir *max_ var*, *max_ depth*, *num_ bins*, *learning_ rate*. - Aumentar *l1_ reg*, *l2_ reg*. - Cambiar *leaf_ type* a un tipo de ajuste subóptimo.	Alto	- Se construye una secuencia de árboles que corrigen los errores cometidos en las predicciones. - Muchos parámetros que optimizar, lo cual lo convierte en un modelo muy flexible. - Puede extrapolar predicciones.

Tabla 2: Resumen de los algoritmos: Linear Tree, Linear Random Forest, Regression-Enhanced Random Forest, Explainable Boosted Regression y Piecewise Linear Gradient Boosting.

6. Trabajos futuros

Según los resultados finales vistos en la sección 4, los algoritmos híbridos en cuanto a precisión son totalmente aceptables, el mayor problema que presentan viene del lado de la eficiencia computacional, especialmente los *Linear Trees* y los *Linear Random Forest*.

Estos dos algoritmos necesitan mejoras de optimización para convertirse en opciones viables, y sería interesante implementar las ideas que se plantean en PLGB de selección incremental de variables y de los ajustes subóptimos de las Regresiones en los nodos, además de las mejoras a nivel de *software* para lograr ese propósito.

Una idea interesante que explorar sin salirse de los modelos basados en árboles de decisión es tomar como esqueleto el algoritmo de EBLR, pero intercambiando el modelo de Regresión base por un CART y que los modelos correctores sean modelos de Regresión en lugar de árboles.

Otra idea que probar sobre la práctica, siguiendo la misma línea de LRF, es desarrollar un *Random Forest* utilizando modelos de Regresión como estimadores base, y cerciorarse de conseguir resultados adecuados.

A. Detalles del desarrollo del trabajo

Todo el código elaborado para realizar la simulación de escenarios reales en los que se usan todos estos algoritmos se puede encontrar en el repositorio de GitHub (`https://github.com/mgp165/TFG`). El proyecto ha sido desarrollado en su plenitud con el lenguaje de programación *Python* usando *notebooks* de *Jupyter*. El ordenador utilizado para llevar a cabo todos los cálculos posee un procesador Intel Core i5-6500 con una velocidad de 3.20 GHz, 4 núcleos físicos y 4 hilos de ejecución. También dispone de 8 GB de memoria RAM y un HDD de 1 TB, todo ello equipado con el sistema operativo Windows 10.

La estructura del repositorio se precisa a continuación:

1. Data:
    a) Raw: aquí están todos los *datasets* sin tratar además de las fuentes de donde se han descargado.
    b) Intermediate: aquí están los *notebooks* donde se limpian los datos, se crean nuevas variables y se hacen las separaciones de entrenamiento, validación y test.
    c) Model input: hay 3 subcarpetas donde se guardan los conjuntos de entrenamiento, validación y test.
    d) Model output: aquí se guardan las métricas proporcionadas por los modelos y los modelos finales entrenados.
    e) Reporting: donde se hace el informe final, graficando todas las métricas y comparando los modelos.
2. Notebooks:
    a) EDA: aquí están todos los análisis de datos exploratorios de cada conjunto de datos.
    b) Models: aquí están los principales *notebooks* del proyecto, en los cuales se ejecutan los algoritmos y los modelos se analizan y se modifican sus hiperparámetros para lograr su mejor versión.
    c) Examples: en esta carpeta está el código de los ejemplos prácticos usados en la memoria.
    d) Utils: esta carpeta contiene funciones que se usan a lo largo de todo el proyecto para un código más limpio.

Dentro de cada uno de estos directorios, hay 6 subcarpetas para cada uno de los *datasets* de ejemplo.

La Tabla 3 recoge la información respectiva al tiempo destinado a la elaboración del trabajo. La Tabla 4, las asignaturas cursadas durante el Grado necesarias para el correcto desarrollo del trabajo.

Tarea	Tiempo (horas)
Recopilación de materiales	20
Estudio de bibliografía	15
Elaboración de resultados gráficos/numéricos	70
Redacción de la memoria	45
Total	150

Tabla 3: Tiempo aproximado de dedicación al trabajo.

Asignatura	Páginas	Descripción
Series Temporales	$6-7$	La separación de datos y la posterior forma de evaluar y seleccionar los modelos se basa en la diapositiva 22 de la presentación *Model Assesment and Selection*.
Análisis de Datos I	$9-11$	La sección de Regresión Lineal está basada en el tema 6 de la asignatura.
Análisis de Datos II	$12-13$	La sección de Regresión Logística se basa en el tema 3 y el *notebook Teoría T3* de la asignatura.
Análisis de Datos II	General	Relación general con el tema 5 de la asignatura.

Tabla 4: Asignaturas relacionadas con el trabajo.

B. Métricas

Este apartado sirve de *backup*. En las tablas 5 y 6 se pueden ver las métricas obtenidas con los mejores modelos para cada *dataset* ordenadas igual que en la Figura 12. Los hiperparámetros que no se han dejado por defecto con los que se han obtenido estos mejores modelos, y por ende, estas métricas, se presentan a continuación:

- **Cáncer de mama:**

1. **LT:** profundidad máxima: 3, penalización del modelo en las hojas: Ridge.
2. **LR:** penalización del modelo: Ridge.
3. **EBLR:** penalización del modelo base: Ridge, número de nuevas variables: 5, profundidad máxima de los árboles: 3.
4. **LRF:** número de árboles: 5, profundidad máxima de los árboles: 5, penalización del modelo en las hojas: Ridge.
5. **XGB:** número de árboles: 50, profundidad máxima de los árboles: 2, tasa de aprendizaje: 0.2.
6. **GB:** número de árboles: 150, profundidad máxima de los árboles: 3, tasa de aprendizaje: 0.2.
7. **RERF:** penalización del modelo base: Elastic Net ($\alpha = 0.75$), número de árboles: 30, profundidad máxima de los árboles: 2.
8. **RF:** número de árboles: 15, profundidad máxima de los árboles: 7.
9. **DT:** profundidad máxima: 2.

- **Defectos de software:**

1. **XGB:** número de árboles: 150, profundidad máxima de los árboles: 3, tasa de aprendizaje: 0.1.
2. **GB:** número de árboles: 200, profundidad máxima de los árboles: 3, tasa de aprendizaje: 0.1.
3. **RF:** número de árboles: 40, profundidad máxima de los árboles: 10.
4. **RERF:** penalización del modelo base: ninguna, número de árboles: 100, profundidad máxima de los árboles: 10.
5. **LT:** profundidad máxima: 7, penalización del modelo en las hojas: ninguna.
6. **LRF:** número de árboles: 5, profundidad máxima de los árboles: 7, penalización del modelo en las hojas: Ridge.
7. **DT:** profundidad máxima: 5.
8. **EBLR:** penalización del modelo base: ninguna, número de nuevas variables: 2, profundidad máxima de los árboles: 1.
9. **LR:** penalización del modelo: ninguna.

- **Seguros de coche:**

1. **XGB:** número de árboles: 50, profundidad máxima de los árboles: 3, tasa de aprendizaje: 0.2.
2. **GB:** número de árboles: 200, profundidad máxima de los árboles: 3, tasa de aprendizaje: 0.1.
3. **RERF:** penalización del modelo base: Ridge, número de árboles: 20, profundidad máxima de los árboles: 7.
4. **EBLR:** penalización del modelo base: Lasso, número de nuevas variables: 2, profundidad máxima de los árboles: 3.
5. **LT:** profundidad máxima: 1, penalización del modelo en las hojas: Ridge.
6. **DT:** profundidad máxima: 5.
7. **RF:** número de árboles: 35, profundidad máxima de los árboles: 10.
8. **LR:** penalización del modelo: Ridge.
9. **LRF:** número de árboles: 20, profundidad máxima de los árboles: 10, penalización del modelo en las hojas: Ridge.

- **Transformador de electricidad:**

1. **RERF:** penalización del modelo base: ninguna, número de árboles: 10, profundidad máxima de los árboles: 2.
2. **LR:** penalización del modelo: ninguna.
3. **EBLR:** penalización del modelo base: ninguna, cálculo de los errores: lineal, número de nuevas variables: 5, profundidad máxima de los árboles: 3.
4. **LT:** profundidad máxima: 1, penalización del modelo en las hojas: Ridge.
5. **LRF:** número de árboles: 5, profundidad máxima de los árboles: 1, penalización del modelo en las hojas: Ridge, número de variables aleatorias para cada árbol: la mitad del total.
6. **GB:** número de árboles: 125, profundidad máxima de los árboles: 3, tasa de aprendizaje: 0.05.
7. **XGB:** número de árboles: 100, profundidad máxima de los árboles: 5, tasa de aprendizaje: 0.05.
8. **RF:** número de árboles: 15, profundidad máxima de los árboles: 7.
9. **DT:** profundidad máxima: 7.

- **Superconductores:**

  1. **XGB:** número de árboles: 400, profundidad máxima de los árboles: 4, tasa de aprendizaje: 0.2.
  2. **RERF:** penalización del modelo base: Elastic Net ($\alpha = 0.25$), número de árboles: 20, profundidad máxima de los árboles: 25.
  3. **GB:** número de árboles: 200, profundidad máxima de los árboles: 4, tasa de aprendizaje: 0.1.
  4. **RF:** número de árboles: 20, profundidad máxima de los árboles: 7.
  5. **DT:** profundidad máxima: 7.
  6. **LT:** profundidad máxima: 2, penalización del modelo en las hojas: Lasso.
  7. **EBLR:** penalización del modelo base: Lasso, cálculo de los errores: lineal, número de nuevas variables: 7, profundidad máxima de los árboles: 1.
  8. **LRF:** número de árboles: 30, profundidad máxima de los árboles: 2, penalización del modelo en las hojas: Ridge.
  9. **LR:** penalización del modelo: Elastic Net ($\alpha = 0.5$).

- **Precios de casas:**

  1. **EBLR:** penalización del modelo base: Ridge, cálculo de los errores: lineal, número de nuevas variables: 7, profundidad máxima de los árboles: 1.
  2. **GB:** número de árboles: 250, profundidad máxima de los árboles: 2, tasa de aprendizaje: 0.1.
  3. **RERF:** penalización del modelo base: Elastic Net ($\alpha = 0.25$), número de árboles: 20, profundidad máxima de los árboles: 1.
  4. **LR:** penalización del modelo: Elastic Net ($\alpha = 0.25$).
  5. **LT:** profundidad máxima: 2, penalización del modelo en las hojas: Lasso.
  6. **XGB:** número de árboles: 400, profundidad máxima de los árboles: 2, tasa de aprendizaje: 0.01.
  7. **LRF:** número de árboles: 2, profundidad máxima de los árboles: 4, penalización del modelo en las hojas: Ridge.
  8. **DT:** profundidad máxima: 5.
  9. **RF:** número de árboles: 10, profundidad máxima de los árboles: 3.

Cáncer de mama			Defectos de software			Seguros de coche		
Modelo	AUC		Modelo	AUC		Modelo	AUC	
	Train	Test		Train	Test		Train	Test
LT	0.9991	0.9925	XGB	0.7976	0.7938	XGB	0.9299	0.9288
LR	0.9988	0.9896	GB	0.7989	0.7938	GB	0.9399	0.9287
EBLR	0.9992	0.9896	RF	0.8177	0.7923	RERF	0.9347	0.9269
LRF	0.9963	0.9896	RERF	0.8201	0.7913	EBLR	0.9125	0.9225
XGB	1.0	0.9861	LT	0.7911	0.7904	LT	0.9161	0.9199
GB	1.0	0.9850	LRF	0.7821	0.7860	DT	0.9112	0.9184
RERF	0.9993	0.9832	DT	0.7871	0.7856	RF	0.9551	0.9174
RF	0.9999	0.9742	EBLR	0.7804	0.7849	LR	0.9048	0.9090
DT	0.9760	0.9366	LR	0.7797	0.7843	LRF	0.8813	0.8807

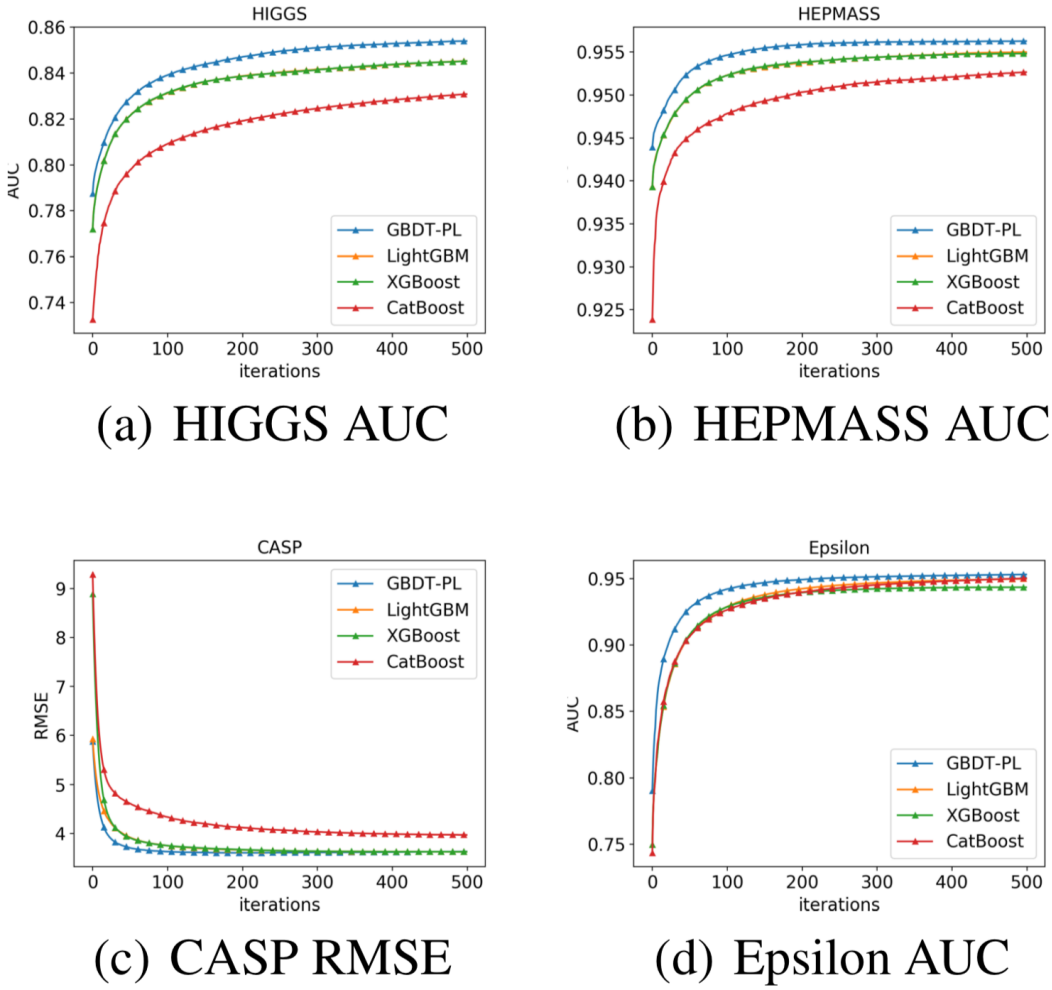
Tabla 5: Métricas obtenidas con los modelos para los conjuntos de datos de clasificación.

Transformador de electricidad			Superconductores			Precios de casas		
Modelo	RMSE		Modelo	RMSE		Modelo	RMSE	
	Train	Test		Train	Test		Train	Test
RERF	0.4527	0.3261	XGB	6.5646	9.7350	EBLR	17.7666	21.2598
LR	0.4537	0.3264	RERF	5.2294	9.8433	GB	14.6796	21.9205
EBLR	0.4515	0.3265	GB	9.1499	10.7280	RERF	25.0021	22.9766
LT	0.4515	0.3266	RF	11.6791	12.8171	LR	26.1887	23.6124
LRF	0.4550	0.3273	DT	12.8773	14.3902	LT	16.1508	25.8673
GB	0.4505	0.3290	LT	15.8250	16.6735	XGB	27.0270	28.4455
XGB	0.4453	0.3317	EBLR	17.1454	17.5645	LRF	36.5368	28.8036
RF	0.4489	0.3357	LRF	17.9055	18.3630	DT	28.3771	36.7113
DT	0.4618	0.3415	LR	19.3188	20.0581	RF	35.4483	37.9263

Tabla 6: Métricas obtenidas con los modelos para los conjuntos de datos de regresión.

C. Resultados PLGB

Este algoritmo se ha implementado para el sistema operativo Linux (`https://github.com/GBDT-PL/GBDT-PL/tree/master`) y por ello no se ha podido aplicarlo en los datos elegidos como ejemplo para este trabajo. Sin embargo, aquí se exponen algunos de los gráficos disponibles en [13] que muestran el poder predictivo de estos modelos. En la Figura 14, se ve que este algoritmo obtiene mejores métricas en 4 *datasets* de ejemplo, 1 de regresión y 3 de clasificación, tanto con un número N de árboles T_j muy pequeño como con un N considerablemente grande (500) que otros tres algoritmos de *boosting*, como son XGB o *LightGBM* y *CatBoost* que no han sido explicados en esta memoria.



(a) HIGGS AUC (b) HEPMASS AUC

(c) CASP RMSE (d) Epsilon AUC

Figura 14: Comparativa de precisión de PLGB.

Referencias

[1] Breiman, L. (1996). Bagging predictors. *Machine Learning*, **24**(2), 123–140. https://doi.org/10.1007/BF00058655

[2] Breiman, L. Pasting Small Votes for Classification in Large Databases and On-Line. *Machine Learning* **36**, 85–103 (1999). https://doi.org/10.1023/A:1007563306331

[3] Breiman, L. (2001). Random Forests. *Machine Learning*, **45**(1), 5–32. https://doi.org/10.1023/A:1010933404324

[4] Breiman, L., Friedman, J., Olshen, R.A. and Stone, C.J. (1984). *Classification and Regression Trees* (1st ed.). Chapman and Hall/CRC. https://doi.org/10.1201/9781315139470

[5] Cerliani, M., Linear Tree: the perfect mix of Linear Model and Decision Tree. https://towardsdatascience.com/linear-tree-the-perfect-mix-of-linear-model-and-decision-tree-2eaed21936b7 (Consultado el 11 de Abril de 2024).

[6] Chen, T. and Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,* 785–794. https://doi.org/10.1145/2939672.2939785

[7] Dillard, L., The Best of Both Worlds: Linear Model Trees. https://medium.com/convoy-tech/the-best-of-both-worlds-linear-model-trees-7c9ce139767d (Consultado el 11 de Abril de 2024).

[8] Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, **29**(5), 1189–1232. https://doi.org/10.1214/aos/1013203451

[9] Ilic, I., Görgülü, B., Cevik, M. and Baydoğan, M. G. (2021). Explainable boosted linear regression for time series forecasting. *Pattern Recognition, 120*, 108144-. https://doi.org/10.1016/j.patcog.2021.108144

[10] Mueller, S., Overcoming the Limitations of Tree-based Models in Time Series Forecasting. https://medium.com/@simon.peter.mueller/overcoming-the-limitations-of-tree-based-models-in-time-series-forecasting-c2c5b (Consultado el 4 de Marzo de 2024).

[11] Peña Sánchez de Rivera, D. (2002). Análisis de datos multivariantes [electronic resource]. McGraw-Hill España.

[12] Quinlan, J.R. (1992). Learning with Continuous Classes. *Proceedings of Australian Joint Conference on Artificial Intelligence, Hobart 16-18 November 1992*, 343-348.

[13] Shi, Y., Li, J., and Li, Z. (2019). Gradient Boosting With Piece-Wise Linear Regression Trees. *arXiv.Org.* `https://doi.org/10.48550/arxiv.1802.05640`

[14] Zhang, H., Nettleton, D. and Zhu, Z. (2019). Regression-Enhanced Random Forests. *arXiv.Org.* `https://doi.org/10.48550/arxiv.1904.10416`