UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
ESCUELA DE POSTGRADO Y EDUCACIÓN CONTINUA
DEPARTAMENTO DE INGENIERÍA MATEMÁTICA

SELECCIÓN DE CARACTERÍSTICAS INTEGRADA EN UN MODELO ADITIVO DE REGRESIÓN

TESIS PARA OPTAR AL GRADO DE MAGÍSTER EN CIENCIAS DE DATOS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MATEMÁTICO

JOHNNY GODOY SÁNCHEZ

PROFESOR GUÍA:
JOAQUÍN FONTBONA TORRES

MIEMBROS DE LA COMISIÓN:
JORGE SILVA SÁNCHEZ
FELIPE TOBAR HENRÍQUEZ

SANTIAGO DE CHILE
2024

SELECCIÓN DE CARACTERÍSTICAS INTEGRADA EN UN MODELO ADITIVO DE REGRESIÓN

En un mundo donde los modelos de aprendizaje automático se aplican en diversas áreas cruciales y de alto riesgo, desde medicina hasta finanzas, la transparencia algorítmica se vuelve esencial para comprender las decisiones automatizadas y asegurar que se puedan tomar con responsabilidad. Sin embargo, los algoritmos caja negra de alto rendimiento que dominan actualmente el panorama carecen de esta transparencia.

En este contexto, los modelos aditivos han resurgido como una alternativa en el campo de la inteligencia artificial interpretable. Son modelos de regresión simples de entender que, cuando se combinan con técnicas modernas como los árboles potenciados por gradiente, ofrecen un rendimiento similar a los mejores modelos caja negra. Sin embargo, surge un desafío cuando los conjuntos de datos tienen un gran número de características o dimensiones, como suele ser el caso en aplicaciones contemporáneas de aprendizaje de máquinas: Los modelos aditivos a menudo proporcionan explicaciones extensas que dificultan la obtención de la transparencia deseada.

Esta tesis afronta este desafío presentando un nuevo algoritmo de entrenamiento para modelos aditivos que integra selección de características, combinando las ideas de árboles potenciados por gradiente con el criterio de mínima redundancia y máxima relevancia. Esto permite obtener explicaciones más concisas sin comprometer el rendimiento ofrecido por los modelos de alta complejidad. Para construir una solución accesible para la comunidad de científicos de datos y expertos en inteligencia artificial, se desarrolló como una librería en Python, siguiendo la API de scikit-learn, una de las librerías más populares en el campo del aprendizaje automático. Se asegura la precisión y eficiencia de esta implementación utilizando una estructura de datos eficiente para la representación de un árbol, y un algoritmo especializado de programación dinámica para su ajuste.

El trabajo también valida la efectividad del modelo propuesto a través de dos experimentos. Primero, se demuestra que el modelo logra un rendimiento de generalización comparable con otros modelos ampliamente utilizados, según su error en la validación cruzada en un repositorio masivo de diversos conjuntos de datos. Segundo, se verifica que el modelo efectivamente selecciona un conjunto de variables relevantes al entrenarlo en un conjunto de datos al cual se le introdujeron variables irrelevantes o redundantes, lo que confirma su capacidad para eliminar información inútil para la resolución del problema.

Además, se crean dos experimentos exploratorios con el objetivo de mejorar el modelo. Primero, se comparan de tiempos de entrenamiento de un algoritmo ejecutado normalmente, contra uno cuyo entrenamiento se forzó a detener antes. Segundo, se crea una propuesta de hiperparámetros por defecto fáciles de recordar, que mantengan un buen desempeño relativo con respecto a los hiperparámetros óptimos.

El enfoque innovador del algoritmo propuesto ofrece soluciones precisas a problemas de regresión críticos, brindando confianza a expertos y usuarios en la toma de decisiones respaldadas por modelos de aprendizaje automático, de una forma fácil de implementar para científicos de datos. Además, sienta las bases para futuras investigaciones en el campo del aprendizaje automático interpretable.

Wir müssen wissen.
Wir werden wissen.
- David Hilbert

Agradecimientos

Quiero agradecer a mi familia, en especial a mi hermano Eduardo, por su valioso apoyo durante esta investigación.

Agradezco también a los amigos que me han acompañado, tanto por toda la vida como Jorge y Juan, como a los nuevos y con los que ya nos hemos distanciado. Aprecio cada sonrisa con ustedes.

Finalmente, agradezco a la comunidad de la Asociación de Ética en Datos e Inteligencia Artificial, en especial a mis compañeros cofundadores Felipe y Camilo, por llevar a cabo esta iniciativa para que todos aprendamos más de un tema fascinante, que pasó de ser un interés, a volverse una real convicción por construir un mundo mejor, inspirando nuestros trabajos de investigación.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Actualmente, los modelos de aprendizaje de máquinas tienen uso ubicuo en la toma automática de decisiones, incluso en áreas de alto riesgo como salud o justicia criminal. Ejemplos incluyen predicción de consecuencias graves en pacientes con neumonía [1] o evaluación de riesgo en reincidencia criminal [2].
Debido al impacto significativo que las decisiones automatizadas pueden tener en la vida humana, influenciando el tratamiento de pacientes o el juicio de criminales, se ha vuelto evidente la necesidad de desarrollar algoritmos transparentes, que sean capaces de generar explicaciones claras y comprensibles de su toma de decisiones. Nuevas regulaciones legales han aparecido, tal como la *General Data Protection Regulation* de la Unión Europea, que explicita el derecho del usuario a tener una explicación sobre decisiones que puedan tener efectos legales sobre él, restringiendo así el uso de algoritmos caja negra. [3].
La necesidad de generar explicaciones presenta un desafío, ya que modelos interpretables como la regresión lineal o los árboles de decisión CART no logran competir en desempeño con modelos caja negra como los ensamblajes de árboles de decisión, que dominan en el área del aprendizaje en datos tabulares [4]. El área de estudio denominada *Explainable Artificial Intelligence* (XAI) se interesa en resolver esta dificultad, estudiando modelos que tengan simultáneamente alto desempeño e interpretabilidad [5].
Los modelos aditivos han captado la atención en XAI como una herramienta precisa con interpretabilidad inherente. Logra incorporar interpretabilidad separando el modelo en componentes aditivas univariadas para cada característica, sin aprender interacciones, y el efecto de cada variable es fácil de describir gráficamente. Además, destacan por la posibilidad de editar errores de modelos en postprocesamiento, lo que significa que es posible mejorar las predicciones manualmente después de que el modelo fue entrenado, mejorando así la precisión y la confiabilidad de las decisiones automatizadas [6]. La implementación moderna conocida como ExplainableBoostingMachine (EBM) [7] demuestra ser una alternativa prometedora a modelos caja negra como XGBoost [8] o CatBoost [9], que resuelven una variedad de problemas de regresión con alta precisión gracias a su uso de árboles potenciados por gradiente (*gradient boosted trees*, o GBT) [10].
Sin embargo, cuando un conjunto de datos presenta alta dimensionalidad, la interpretabilidad global que brinda el modelo aditivo se ve comprometida, pues la forma de interpretabilidad se basa en atribución de contribuciones a cada característica, representada como gráficos. Cuando existe un gran número de gráficos, es infactible analizar todos para obtener interpretaciones completas, por lo que surge la necesidad de entrenar modelos aditivos que utilicen un subconjunto reducido de características, permitiendo así un mejor análisis sin comprometer

significativamente el rendimiento de generalización del modelo.
Para resolver este problema, el trabajo presente propone un nuevo algoritmo para entrenar modelos aditivos, que integra selección de un conjunto reducido de características, buscando maximizar su relevancia de predicción al objetivo y minimizar redundancia mutua. Así, se reduce la cantidad de características utilizadas para tomar el mejor provecho de la interpretabilidad de los modelos aditivos.
La solución propuesta se diseñó e implementó como *software* libre y de código abierto, en la librería `asboostreg` de Python liberada bajo licencia MIT. La librería sigue las especificaciones comunes para modelos de aprendizaje de máquinas, como los de la librería scikit-learn. Se modificó un algoritmo especializado de regresión con árboles en una dimensión, basado en programación dinámica para encontrar el óptimo global de un problema de optimización regularizado que se propone para el ajuste. También se diseñó una estructura de datos especial para representar un árbol de forma que permita eficientemente calcular la suma de distintos árboles y realizar inferencia.
Finalmente, para evaluar la eficacia de la propuesta, se llevaron a cabo experimentos exhaustivos que abordan diversas facetas del rendimiento del nuevo algoritmo. Primero, se midió el error de generalización con respecto a otros modelos competitivos en una variedad de conjuntos de datos. Luego, se exploró la capacidad del modelo para seleccionar características relevantes y su resistencia ante la adición de características redundantes. Además, se examinó el impacto de la reducción del número de iteraciones en el proceso de entrenamiento y se propuso un conjunto de hiperparámetros por defecto que buscan encontrar un equilibrio práctico de desempeño en múltiples conjuntos de datos. Estas evaluaciones proporcionan una visión más completa de la utilidad y aplicabilidad del algoritmo propuesto en una variedad de contextos y conjuntos de datos.

Capítulo 2

Antecedentes

Los antecedentes de la tesis se dividen en dos secciones: antecedentes teóricos y antecedentes del estudio. En la primera sección, se presentan los fundamentos teóricos relacionados con los modelos de regresión, en particular los aditivos aditivos, y la selección de características, mientras que la segunda sección aborda la relevancia y el contexto específico del presente estudio.

2.1. Antecedentes teóricos

2.1.1. Modelos de regresión

Primero, se resumen los modelos de regresión base con las ideas más importantes para el funcionamiento del modelo propuesto. Para todos los modelos en adelante, consideramos una tarea de regresión con un conjunto de datos de entrenamiento $D = \{(x_i, y_i) \in \mathbb{R}^m \times \mathbb{R}\}_{i \leq n}$.

Árboles de decisión

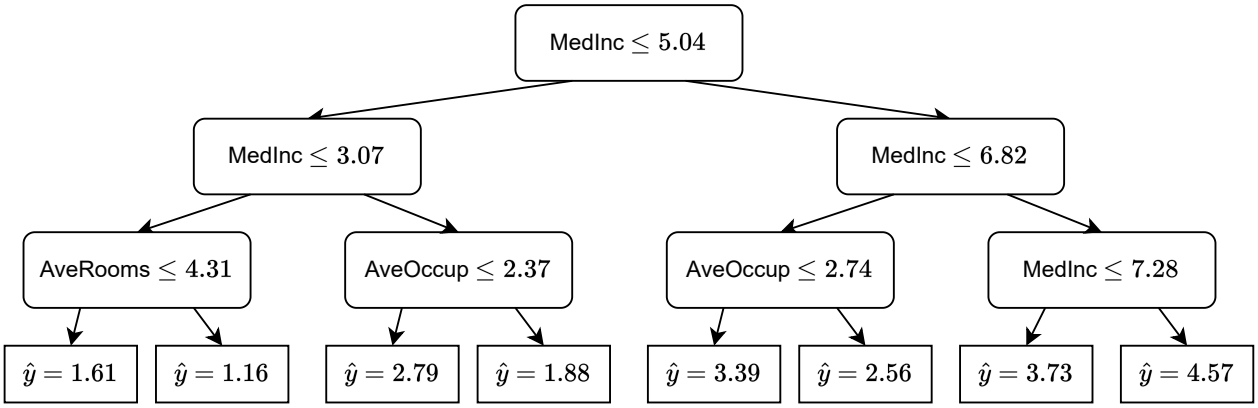
Un árbol binario de decisión $T = (L, R, Q)$ es una tupla donde:

- L y R pueden ser un árbol binario de decisión o una función constante real, cuya evaluación es el valor de una hoja.
- Q es una tupla $(i, s) \in [m] \times \mathbb{R}$, que indica la comparación $x_i \leq s$, induciendo una bipartición de los datos.

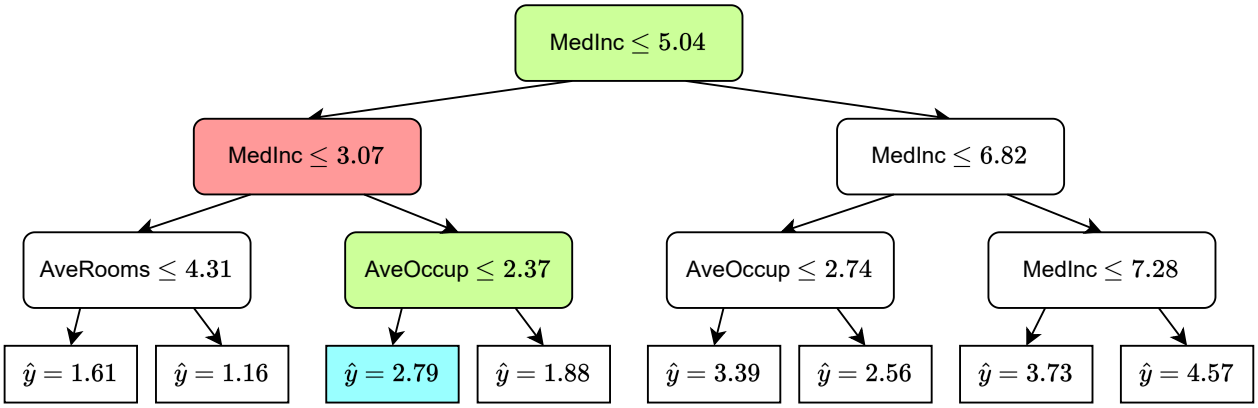
Definimos una función de evaluación del árbol T, abusando notación, también llamada T, de forma recursiva:

$$T(x) = \begin{cases} L(x) \text{ si } x_i \leq s \\ R(x) \text{ si } x_i > s \end{cases}$$

Recordando que en el caso base, se evalúa una función constante. Así, se ve que un árbol biparticiona el espacio de forma recursiva, formando una función constante por partes dividida en los ejes de cada característica. Se llama profundidad del árbol a la profundidad de la recursión, y usualmente es un hiperparámetro que controla el usuario. Los árboles de regresión son árboles binarios de decisión optimizados para minimizar un error de regresión que se haya definido, tal como error cuadrático medio. Un árbol, y la forma de evaluarlo, se muestran en la figura 2.1.



(a) Representación de un árbol de profundidad 3, entrenado para el conjunto de datos de *California Housing* [11].



(b) Evaluación del árbol para $x = (\text{MedInc} = 4, \text{AveOccup} = 1, \text{AveRooms} = 10)$. El diagrama permite evaluar $\hat{y} = T(x) = 2.79$

Figura 2.1: Diagramas de un árbol de decisión, y su evaluación. Poder ser diagramados hace que sean considerados popularmente como inteligencia artificial explicable.

Encontrar un árbol óptimo, es decir, aquel que tenga error mínimo de regresión para un tamaño fijo, es un problema NP-completo [12]. Si bien existen algoritmos que se dedican a resolver el problema de forma exacta [13], no existen garantías de ejecución a tiempo polinomial, las cuales solamente serían posibles si P=NP. Por lo tanto, un algoritmo más común para realizar regresión es CART [14], que selecciona particiones de los datos de forma avara, que se explicará adelante.

Para evaluar una comparación $Q = (i, s)$ en datos D, se definen los lados de la partición $Q_\ell = \{y | x_j \leq s, (x, y) \in D\}, Q_r = \{y | x_j > s, (x, y) \in D\}$, y con ellos la impureza de la comparación:

$$G(Q) = |Q_\ell| \mathbb{V}[Q_\ell] + |Q_r| \mathbb{V}[Q_r]$$

Entonces, primero se busca Q que minimice $G(Q)$, es decir, una partición tal que ambos de sus lados tengan la varianza más pequeña posible, ponderada por la cantidad de datos, de forma que sean mejor aproximados por constantes. El valor óptimo de Q se puede encontrar exhaustivamente en el espacio de las comparaciones, probando cada característica i, y cada valor s que haya tomado esa característica en el conjunto de entrenamiento.

Para encontrar L y R, se aplica recursivamente el algoritmo para encontrar sus propias particiones, es decir, buscando las particiones óptimas para los conjuntos de datos Q_ℓ y Q_r. Se pueden implementar distintas condiciones de quiebre para detener la recursión: $|Q|$ suficientemente pequeño, $G(Q)$ suficientemente pequeño, suficiente profundidad de recursión, etc. Cumpliendo una de estas condiciones para los datos D', la hoja tendrá como constante la media del objetivo en D'.
Según la documentación de scikit-learn [15], traducida al español, la complejidad de su implementación es como sigue:

> En general, el costo de tiempo de ejecución para construir un árbol binario equilibrado es $O(nm\log(n))$ y el tiempo de consulta es $O(\log(n))$. Aunque el algoritmo de construcción del árbol intenta generar árboles balanceados, no siempre lo logrará. Suponiendo que los subárboles permanezcan aproximadamente equilibrados, el costo en cada nodo consiste en buscar a través de O(m) para encontrar la característica que ofrece la mayor reducción en el criterio de impureza, por ejemplo, la pérdida logarítmica (que es equivalente a una ganancia de información). Esto tiene un costo de $O(mn\log(n))$ en cada nodo, lo que lleva a un costo total sobre todo el árbol (sumando el costo en cada nodo) de $O(mn^2\log(n))$.

Debido a su enfoque, un árbol de decisión es un modelo ideal para ajustarse a una función con muchas discontinuidades, algo que no es posible para una variedad de otros algoritmos de aprendizaje. Sin embargo, a pesar de ser considerados como una herramienta clásica entre los modelos interpretables, que se utiliza para explicar a otros modelos complejos [16], la facilidad de interpretar un árbol ha sido puesta en duda en los últimos años. Estudios demuestran que las explicaciones dadas por un árbol de decisión son innecesariamente extensas [17] [18] y difíciles de acortar a un largo razonable [19], por lo que no es siempre tangible. En contraste, el modelo propuesto evita redundancia en características, y por tanto, en largo de explicaciones. Además, es difícil obtener una buena generalización: Árboles muy profundos tienen un grave sobreajuste a los datos, y con muy poca profundidad se subajustan a los datos. Aún con una profundidad óptima, entrenar el árbol de forma avara tiene resultados muy subóptimos, con nodos que buscan deshacer decisiones incorrectas realizadas en iteraciones anteriores [13]. Para mantener la capacidad de aprender discontinuidades y lidiar con el problema de generalización, es común utilizar los árboles de decisión como un modelo base de una técnica de ensamble de modelos, que combina la decisión de varios árboles, sea reduciendo la varianza de los árboles grandes o el sesgo de los árboles pequeños.

Boostrap Aggregating (Bagging)

Boostrap Aggregating es una técnica de ensamble que realiza M muestreos aleatorios con reemplazo (*bootstrap*) de los datos, y entrena un modelo base h en cada muestreo. Esta técnica regulariza a un modelo base h que pueda sobreajustar, para disminuir su varianza. El modelo final retorna un promedio simple de los M modelos base, como muestra la figura 2.2.

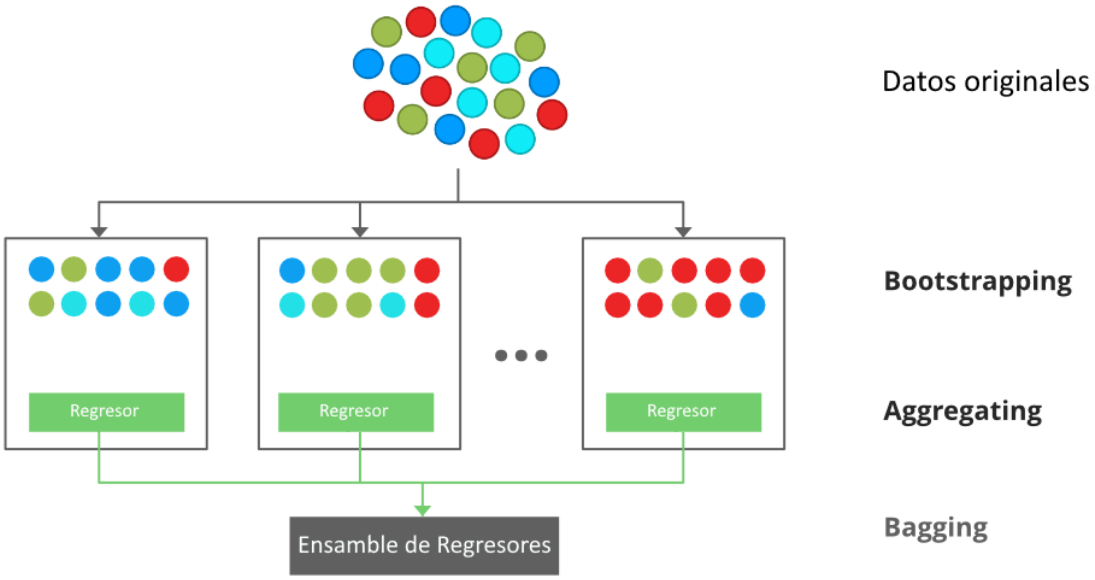


Figura 2.2: Ejemplo de *Bagging*. Adaptado del portal Geeks for Geeks [20].

Si en vez de muestrear los datos se muestrean las características, el método se llama ensamble de subespacio aleatorio. Al combinar muestreo tanto de características como datos, se mejora la diversidad del ensamble al volver a cada modelo base h más distinto, necesario para disminuir la varianza. La analogía es que cada modelo h es un "experto" para un subconjunto de los datos que solamente ve algunas características, y el modelo integra la visión de cada experto como el promedio de todos. El uso de estas dos técnicas con un árbol CART de alta profundidad como modelo base forma al algoritmo *Random Forest* (Bosque aleatorio), un método popular para el aprendizaje tabular debido a su desempeño superior en una variedad de tareas [21].

Potenciamiento de gradiente

El potenciamiento de gradiente [10] es un algoritmo de ensamble que suma modelos de forma iterativa, de manera que el nuevo modelo a sumar pueda predecir el error del ensamble ya formado, como ilustra la figura 2.3. Más formalmente, sea h un modelo, que llamaremos modelo base. El modelo de potenciamiento de gradiente F_M aprende una secuencia de M modelos base y retorna como predicción la suma de ellos:

$$\hat{y} = F_M(x) = \sum_{i \leq M} h_i(x)$$

El modelo F_M se construye de forma recursiva:

$$F_0 = \overline{y}$$
$$F_{i+1}(x) = F_i(x) + h_i(x)$$

Idealmente, en una iteración tendríamos que $F_{i+1}(x) \approx y$, o equivalentemente, $h_i(x) \approx y - F_i(x)$, el residuo del modelo en la i-ésima iteración. Para lograrlo, el modelo h_i se entrena en los datos $D_i = (x, y - F_i(x))$.
Los modelos del estado del arte en regresión para datos tabulares XGBoost [8], LightGBM

[22] y Catboost [9] son implementaciones de árboles de profundidad baja potenciados por gradiente, y son unos de los principales algoritmos usados competencias de la plataforma Kaggle [23], superando consistentemente a las redes neuronales profundas en datos tabulares [24]. El algoritmo de ajuste del árbol de decisión base es distinta para cada uno de los modelos, pero siempre es similar a CART, realizando optimización avara. Por ejemplo, XGBoost penaliza valores grandes en las hojas con regularización L^1 y L^2, cambia la función de impureza y el valor de las hojas, pero el resto del algoritmo se mantiene similar.
Se mencionarán dos de las modificaciones que utilizan las implementaciones estado del arte que mejoran la regularización de potenciamiento de gradiente, debido a que el modelo propuesto también las implementa:

- Potenciamiento estocástico [25]: En cada iteración, en vez de entrenar en los datos D_i, se entrena un subconjunto más pequeño de D_i. La selección del subconjunto puede ser total o parcialmente aleatoria, y con una distribución uniforme o pesada según los gradientes más grandes. CatBoost en particular tiene con 5 distintas opciones para el muestreo (incluyendo determinista, seleccionando todo D_i) [26], 3 de las cuales también implementa `asboostreg`.

- Tasa de Aprendizaje: A modo de regularización, se toma una tasa de aprendizaje $\mu \in (0, 1]$, cambiando a la fórmula $F_{i+1}(x) = F_i(x) + \mu h_i(x)$. Así, todavía se entrena h_i en $(x, y - F_i(x))$, pero la predicción de h reduciendo su magnitud en μ, con un efecto regularizador. Las librerías mencionadas utilizan una tasa de aprendizaje constante en cada iteración, en vez de un esquema adaptativo.

Se presenta el pseudocódigo del algoritmo con ambas modificaciones 1.

Algoritmo 1: Potenciamiento de gradiente

Data: $X \in \mathbb{R}^{n \times m}$ datos, $y \in \mathbb{R}^n$ objetivo, $I \in \mathbb{N}$ cantidad de iteraciones, tasa de aprendizaje μ, proporción de submuestreo s
Result: Modelo final F
$F = \overline{y}$; `/* Inicializa el primer predictor como constante */`
$r \leftarrow y - F_0$; `/* Inicializa el residuo */`
for $i \in [1 \dots I]$ **do**
  Genera X' como un submuestreo aleatorio de filas de X;
  Ajustar un árbol de regresión T en los datos (X', r);
  $r \leftarrow r - \mu T(X)$;
  $F \leftarrow F + \mu T$; `/* Suma de funciones */`
end

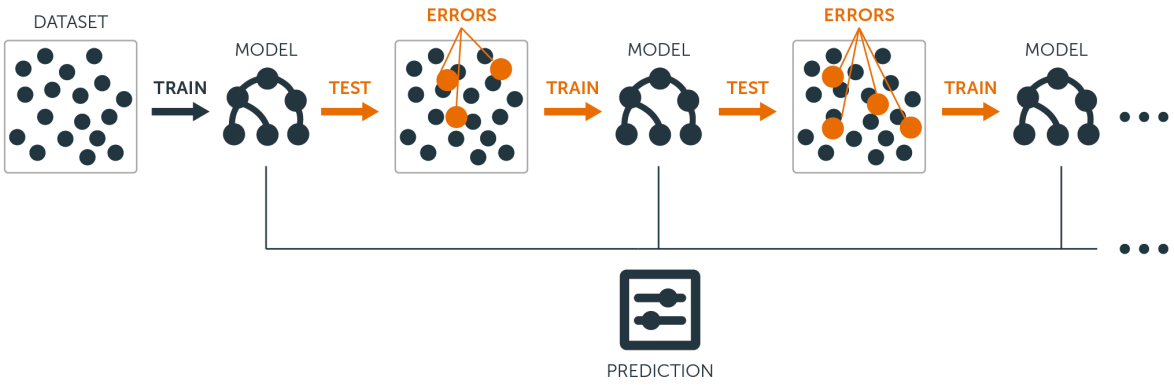


Figura 2.3: Diagrama de potenciamiento de gradiente. Obtenido directamente del blog de BigML [27].

2.1.2. Modelos aditivos

Un modelo aditivo [14] predice un objetivo de regresión univariado Y con variables predictoras $(x_i)_{i\leq m}$ mediante las ecuaciones:

$$\mathbb{E}[Y] = \beta + \sum_{i\leq m} f_i(x_i)$$
$$\mathbb{E}[f_i] = 0 \ : \ \forall i \leq m$$

Donde $\beta \in \mathbb{R}$ y $f_i : \mathbb{R} \to \mathbb{R}$ son los parámetros por estimar. La primera ecuación es la condición de aditividad, que establece que la predicción total es la suma de las contribuciones individuales de cada característica, permitiendo así interpretar al modelo por componentes. La segunda ecuación es la condición de centralización, que asegura que la media de las funciones f_i sea cero, facilitando la identificación del modelo, y permitiendo interpretar los valores de f_i como desviaciones con respecto al valor medio.
Gracias a la condición de aditividad, el modelo tiene 3 mecanismos de interpretación distintos, que tienen análogos a modelos de explicación agnósticos de modelo populares, evitando algunas de sus limitaciones:

- Explicación global: Al graficar la función f_i podemos entender el efecto que tiene la característica x_i sobre el objetivo, no a través de marginalización como los *partial dependence plots* [14], y son válidos para variables correlacionadas. Los m gráficos de cada f_i forman una reconstrucción exacta del modelo, y de ellos podemos visualizar propiedades como monotonía, tasa de crecimiento, convexidad, periodicidad, simetrías y extremos de cada función, como se ejemplifica en la figura 2.4.

- Explicación local: Para explicar localmente la predicción de una instancia x, el vector $(f_i(x_i))_{i\leq m}$ nos da la contribución de cada característica al modelo, de forma que la suma de contribuciones con el valor base β corresponde al valor de la predicción $f(x)$. La contribución es consistente con los valores de SHAP [28] [29], como se demuestra en el anexo A, pero se puede calcular fácilmente sin requerir aproximaciones computacionalmente costosas. Los valores de SHAP, provenientes de la teoría de juegos cooperativos, asignan una contribución equitativa a cada característica en función de su impacto marginal en

la predicción del modelo. Su enfoque proporciona una medida justa y coherente de la importancia de cada característica en la interpretación local del modelo.

- Importancia de características: La importancia global de cada característica es la media de $|f_i|$. Esta medida también se interpreta como la contribución de Shapley media, y no está sesgada por las características granulares, a diferencia de la impureza de árboles, que favorece a variables con muchos valores únicos por permitir más divisiones.

- Transformación a modelo lineal: Para el algoritmo propuesto, una función f_i es constante por partes, de decir, existe S_i una partición en intervalos del dominio de x_i tal que $f_i(x) = \sum_{I \in \mathcal{S}_i} w_I \mathbf{1}_I(x)$. Utilizando la transformación $\phi_i : \mathbb{R} \to \mathbb{R}^{|\mathcal{S}_i|}$ dada por $\phi_i(x) = (\mathbf{1}_I(x))_{I \in \mathcal{S}_i}$, tenemos que f_i es un modelo lineal en $\phi_i(x_i)$, lo que nos permite utilizar técnicas de explicabilidad en modelos lineales, las cuales se traducen en explicaciones del problema original pues la transformación ϕ también es interpretable. Como ejemplo, si queremos entender la importancia de una variable de edad, la función ϕ podrá dividir esa característica en 3: ["es menor de edad", "es adulto", "es adulto mayor"], y le asigna una importancia a cada una. El uso de esta transformación también abre también al uso de técnicas más complejas, como preguntas de explicabilidad formal [30][31].

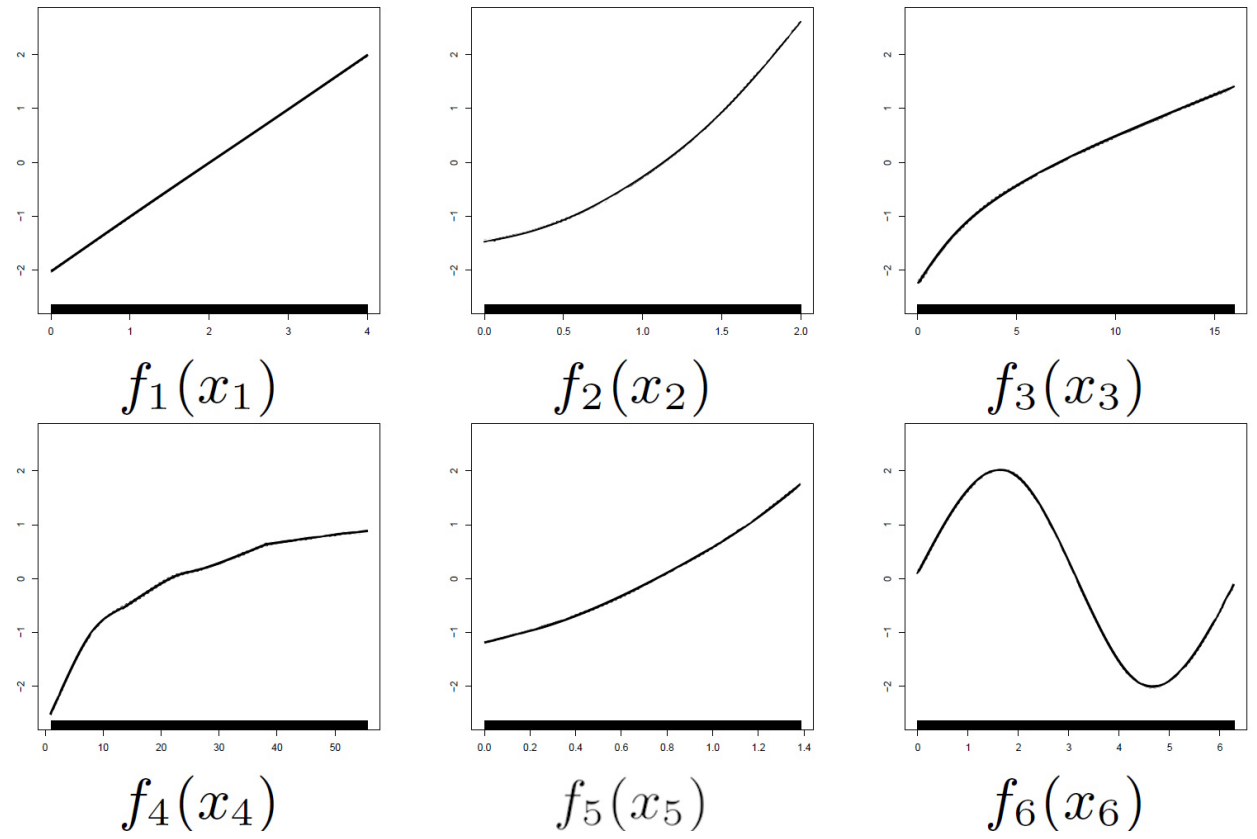


Figura 2.4: Ejemplo de la descomposición de una función aditiva f. De los gráficos, podemos obtener información importante del modelo, tal como monotonía, tasa de crecimiento, convexidad, periodicidad, simetrías y extremos de cada función. Figura adaptada de [32].

Explainable Boosting Machine

La *Explainable Boosting Machine* (EBM) es una implementación de modelo aditivo que logra un desempeño competitivo con modelos caja negra de vanguardia en problemas de regresión y clasificación [32], al combinar la idea de modelos aditivos con árboles potenciados por gradiente.
Un EBM es un modelo más complejo que incorpora interacciones entre dos componentes, llamado GA^2M, y es de la forma:

$$y = f(x) \approx \beta + \sum_{i \leq m} f_i(x_i) + \sum_{ij \in S} f_{ij}(x_i, x_j)$$

Pero es posible tomarlo como un modelo aditivo al considerar $S = \emptyset$. Para entrenar un EBM, se utiliza potenciamiento de gradiente, pero en cada iteración se entrena un bosque aleatorio con 1 sola característica, iterando por las características de forma cíclica, como ejemplifica el pseudocódigo 2.

Algoritmo 2: Entrenamiento del Explainable Boosting Machine

Data: $X \in \mathbb{R}^{n \times m}$ datos, $y \in \mathbb{R}^n$ objetivo, $I \in \mathbb{N}$ cantidad de iteraciones, tasa de aprendizaje $\mu \in (0, 1]$
Result: Lista de funciones $[f_i]_{i \leq m}$ entrenadas
$f = [0]_{i \leq m}$; `/* Todas las funciones se inicializan como la nula */`
$r \leftarrow y - \bar{y}$; `/* Inicializa el residuo */`
for $i \in [1 \dots I]$ **do**
  for $j \in [1 \dots m]$ **do**
    $x \leftarrow X_{\cdot j}$;
    Ajustar un bosque aleatorio T en los datos (x, r);
    $r \leftarrow r - \mu T(x)$;
    $f_i \leftarrow f_i + \mu T$; `/* Suma de funciones */`
  end
end

Cabe destacar que el algoritmo 2 no garantiza que se cumpla la condición de centralización $\mathbb{E}[f_i] = 0$, y no aprende la constante β. Estimando $\mathbb{E}[f_i] = \frac{1}{n} \sum_{j \leq n} f_i(x_{ij})$, se aprovecha la linealidad de la esperanza para hacer la traslación:

$$f_i' = f_i - \mathbb{E}[f_i] \, \forall i \leq m$$
$$\beta = \sum_{i \leq m} \mathbb{E}[f_i]$$

Que garantiza centralización sin sesgar al modelo, pues se trasladó de forma que $\sum f_i = \beta + \sum f_i'$.
En el algoritmo 2, las características se entrenan en un orden específico, lo que introduce un sesgo. Para mitigar su efecto, EBM utiliza una tasa de aprendizaje muy pequeña, lo que también significa que requiere un gran número de iteraciones en comparación a los otros algoritmos de potenciamiento de gradiente, como muestra la tabla 2.1.
Además de requerir tantas iteraciones, EBM no utiliza un árbol de decisión como regresor base, sino que un bosque aleatorio de 25 árboles.

Tabla 2.1: Configuración por defecto de diferentes algoritmos GBT. CatBoost elige la tasa de aprendizaje en función de otros hiperparámetros y del tamaño de los datos.

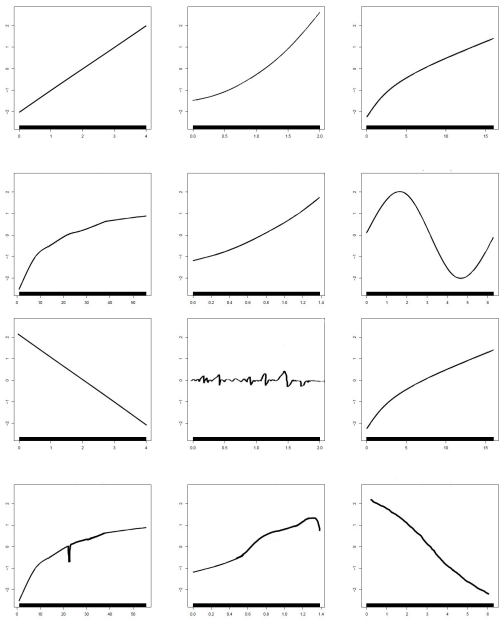
Algoritmo	Versión	Tasa de aprendizaje	Número de iteraciones
XGBoost	2.1.0	0.3	10
LightGBM	4.2.0	0.1	100
CatBoost	1.2.2	Dinámica	1000
EBM	0.5.0	0.01	5000

Por último, para mejorar más los resultados, se utiliza comúnmente *bagging* de 8 modelos entrenados con el algoritmo EBM. Estas decisiones de diseño mejoran el desempeño del modelo a expensas de un aumento significativo en el tiempo de entrenamiento.
El modelo EBM está disponible en la librería de Python llamada `interpret`. Su implementación permite entrenar el modelo fácilmente, usarlo para inferencia, y generar gráficos de manera automática.

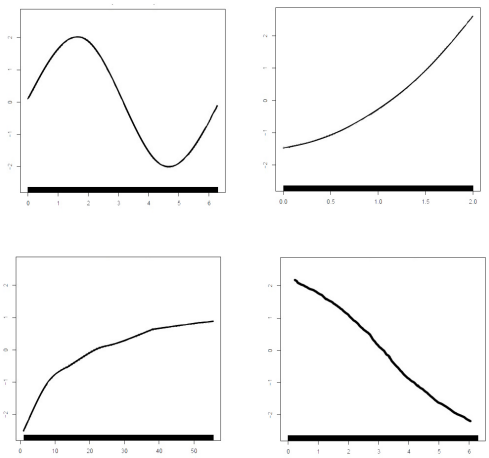
2.1.3. Selección de características

En problemas como procesamiento de texto o expresiones genéticas, los datos presentan un gran número de características, por lo que un modelo aditivo pierde su gran interpretabilidad, pues es infactible para un lector visualizar todos los gráficos generados para obtener una explicación completa, y un número más bajo por el orden de las decenas de características ya puede construir una explicación confusa.
Eliminar variables menos relevantes alivia el problema, permitiendo que el lector se enfoque solo en los efectos importantes para la predicción. De igual forma, variables altamente correlacionadas dificultan la interpretación, pues significa que el modelo no tendrá representación única, y las contribuciones deben dividirse entre las variables.
Por ejemplo, si tenemos $y = 0$ constante con variables de predicción $x_1 = x_2$, las funciones $f_1(x_1) = x_1, f_2(x_2) = -x_2$ forman una solución correcta con $\beta = 0$, pero los gráficos nos hacen pensar que x_1 y x_2 son relevantes y que tienen relaciones monótonas con y, cuando no es así.
Como un ejemplo más concreto, consideremos que estamos construyendo un modelo aditivo para predecir la eficacia de un nuevo medicamento basado en múltiples características. Tenemos datos de miles de pacientes, y cada paciente está descrito por un conjunto extenso de variables, como edad, género, condiciones médicas previas y múltiples biomarcadores.
Entre estas variables, tenemos algunas relacionadas con la música favorita de los pacientes, que no tiene relevancia con la eficacia del medicamento. Sin embargo, de no realizar selección de características, el modelo aditivo intentará asignarles alguna contribución, creando gráficos para variables que no aportan información valiosa al problema médico.
Supongamos que tenemos dos biomarcadores, A y B, altamente positivamente correlacionados. Aunque ambos proporcionan información similar sobre la respuesta al medicamento, la alta correlación podría dificultar que el atribuya contribuciones específicas a cada biomarcador, y sus gráficos generados pueden tener resultados contradictorios (como crecencia en A pero decrecencia en B) que deben ser sumados para tener sentido, interpretando o graficando $f_A + f_B$ en vez de cada función por separado.

Este ejemplo ilustra cómo la presencia de variables irrelevantes o altamente correlacionadas pueden complicar la interpretación del modelo aditivo en conjuntos de datos de alta dimensionalidad. Aquí es donde la selección de características se vuelve crucial: al identificar y utilizar solo las características relevantes, podemos mejorar la interpretabilidad del modelo y reducir la complejidad visual generada por variables no informativas.



(a) Gráficos de modelo aditivo entrenado en 12 características.



(b) Gráficos de modelo aditivo que selecciona las 4 características más importantes

Figura 2.5: Comparación de la explicación entre un modelo entrenado con todas las características y otro que selecciona las más importantes y menos redundantes. El segundo modelo admite una explicación más concisa y útil para el usuario.

Los algoritmos de selección de características permiten entrenar el modelo con solamente un subconjunto de las variables, y para el resto considerar $f_i \equiv 0$, resolviendo el problema anterior. Además, reducen el sobreajuste al eliminar ruido y reducir el número de parámetros.

Se pueden distinguir 3 metodologías de selección de características [33], dependiendo de en qué punto del entrenamiento del modelo se evalúen características:

- *Filter*: Se evalúan las características sin pasarlas por el modelo de predicción, con métodos clásicos como el test χ^2. Si bien es veloz no tener que entrenar un modelo, la selección no es informada por el mismo sesgo inductivo del modelo, y las variables más importantes en selección no coinciden con las más usadas por el modelo, causando una selección subóptima.

- *Wrapper*: Para evaluar un subconjunto de características, se calcula el desempeño de un modelo que fue entrenado usando solamente estas características, y se selecciona el conjunto que muestre el mejor desempeño. Este tipo de algoritmos en principio es capaz de encontrar un conjunto de características minimal optimal, pero cada subconjunto evaluado requiere un reentrenamiento del modelo, y existe una cantidad exponencial de subconjuntos, por lo que es inviable de realizar exhaustivamente. Por esto, se prefiere explorar el espacio de los subconjuntos de características con estrategias heurísticas como eliminación de características recursiva o algoritmos genéticos.

- *Embedded* (Integrada): El modelo incluye regularizaciones o restricciones de tamaño, por lo que éste mismo selecciona las característica a utilizar en tiempo de entrenamiento. Esta técnica busca un balance de velocidad y optimalidad, por requerir un único entrenamiento que cumple optimalidad por definición del modelo, pero no es aplicable universalmente a cualquier modelo. Un caso clásico es usar una penalización de la norma L^1 en regresión lineal, conocida como el modelo LASSO.

Esta tesis propone una modificación del algoritmo EBM que tiene una selección integrada para el caso de regresión, que es beneficioso pues puede utilizar información del modelo sin requerir un parámetro indicando cuántas características utilizar o costosos reentrenamientos, dado que una debilidad importante de EBM es su alto tiempo de entrenamiento. La modificación busca maximizar relevancia y minimizar redundancia de las características seleccionadas sin aumentar significativamente el tiempo de entrenamiento, inspirándose en el algoritmo mRMR [34].
El algoritmo mRMR es de tipo *filtro*, y construye el conjunto S de las k mejores características de forma avara, como tal:

$$S_k = \begin{cases} \{\text{argmax}_{i \leq m} \, \text{rel}(X_i, y)\} & \text{si } k = 0 \\ S_{k-1} \cup \left\{ \text{argmax}_{i \notin S_{k-1}} \dfrac{\text{rel}(X_i, y)}{\text{red}(X_i, S_{k-1})} \right\} & \text{si no} \end{cases}$$

Donde rel es una función que mide que tan relevante es la columna i para predecir a y, y red mide la redundancia que tiene la columna i con respecto a un conjunto de características ya seleccionadas (S_{k-1}).
En la propuesta original del algoritmo [35], se utilizaba como medida de relevancia la información mutua $I(X_i, y)$, y como redundancia el promedio de las informaciones mutuas $\frac{1}{|S|} \sum_{i \in S} I(X_i, X_j)$, pues esta tiene la capacidad de medir relaciones no lineales entre las variables, aunque tiene un alto costo computacional de estimación.
Sin embargo, el estudio [34] propone utilizar medidas más simples de correlación lineal que pueden ser estimadas de manera veloz y robusta.

En particular, se ha determinado en una variedad de conjuntos de datos sintéticos y reales que las medidas simples logra mejores resultados que estimar información mutua [36], tomando las medidas:

$$\text{rel}(X_i, y) = F(X_i, y) := (n-1)\frac{\rho(X_i, y)^2}{1-\rho(X_i, y)^2}$$
$$\text{red}(X_i, S) = \frac{1}{|S|}\sum_{j\in S}|\rho(X_i, X_j)|$$

Donde ρ es la correlación de Pearson, que puede ser estimada eficientemente con complejidad temporal $O(n)$.

2.2. Antecedentes del estudio

Seleccionar un subconjunto óptimo de características es un problema NP-difícil [37], lo que ha impulsado esfuerzos previos para emplear estrategias de aproximación eficientes en la selección de características para algunos modelos aditivos.
Se destacan 4 enfoques, incluyendo el uso de métodos de optimización estocástica en modelos lineales [38], la comparación de estrategias de selección en modelos suaves basados en funciones *spline* [39], algoritmos de optimización para la relajación convexa del problema combinatorial [40], y selección avara de características relevantes en EBM [41].
Los tres primeros enfoques utilizan funciones f_i en una clase más restringida que las empleadas por EBM [32]. El último enfoque es el más cercano al de esta tesis, pues trabaja con un modelo similar, pero ignora el efecto de la redundancia de características, y solamente selecciona por relevancia.
Más recientemente, se propuso otra modificación de EBM, que selecciona características como una etapa de postprocesamiento del modelo con LASSO [42]. El algoritmo que proponen también es capaz de resolver el problema planteado en esta tesis con una metodología distinta, pero debido a la recencia de su publicación, no pudo ser considerado para la comparación de modelos.

Capítulo 3

Métodos matemáticos

3.1. Algoritmo SAB

El nuevo algoritmo denominado SAB (*Sparse Additive Boosting*) entrena modelos aditivos con selección de características, manteniendo un rendimiento competitivo a través de un ensamble por *gradient boosting* [10]. SAB generaliza el concepto de*maximum relevancy minimum redundancy* (mRMR) [36] para seleccionar una característica en cada iteración de potenciamiento. Esquemas del algoritmo se pueden ver en el pseudocódigo 3 y también en la figura 3.1, con un ejemplo de referencia en la figura 3.2.

Algoritmo 3: Potenciamiento de gradiente con selección avara

Data: $X \in \mathbb{R}^{n \times m}$ datos, $y \in \mathbb{R}^n$ objetivo, $I \in \mathbb{N}$ cantidad de iteraciones, función de selección ϕ, tasa de aprendizaje $\mu \in (0, 1]$

Result: Lista de funciones $[f_i]_{i \leq m}$ entrenadas

$f = [0]_{i \leq m}$; /* Todas las funciones se inicializan como la nula */

$h = [0]_{i \leq m}$; /* Comienza un contador del uso de cada variable */

$r \leftarrow y - \bar{y}$;

for $i \in [1 \dots I]$ **do**

  $j \leftarrow \phi(X, r, h)$; /* Encuentra el índice de la mejor característica */

  $h[j] \leftarrow h[j] + 1$; /* Actualiza el contador */

  $x \leftarrow X_{\cdot j}$;

  Ajustar un árbol de regresión T en los datos (x, r);

  $r \leftarrow r - \mu T(x)$;

  $f_i \leftarrow f_i + \mu T$; /* Suma de funciones */

end

Algoritmo 4: Potenciamiento de gradiente con selección avara

Data: $X \in \mathbb{R}^{n \times m}$ datos, $y \in \mathbb{R}^n$ objetivo, $I \in \mathbb{N}$ cantidad de iteraciones, función de selección ϕ, tasa de aprendizaje $\mu \in (0, 1]$

Result: Lista de funciones $[f_i]_{i \leq m}$ entrenadas

$f = [0]_{i \leq m}$; /* Todas las funciones se inicializan como la nula */

$h = [0]_{i \leq m}$; /* Comienza un contador del uso de cada variable */

$r \leftarrow y - \bar{y}$;

for $i \in [1 \dots I]$ **do**

  $j \leftarrow \phi(X, r, h)$; /* Encuentra el índice de la mejor característica */

  $h[j] \leftarrow h[j] + 1$; /* Actualiza el contador */

  $x \leftarrow X_{\cdot j}$;

  Ajustar un árbol de regresión T en los datos (x, r);

  $r \leftarrow r - \mu T(x)$;

  $f_i \leftarrow f_i + \mu T$; /* Suma de funciones */

end

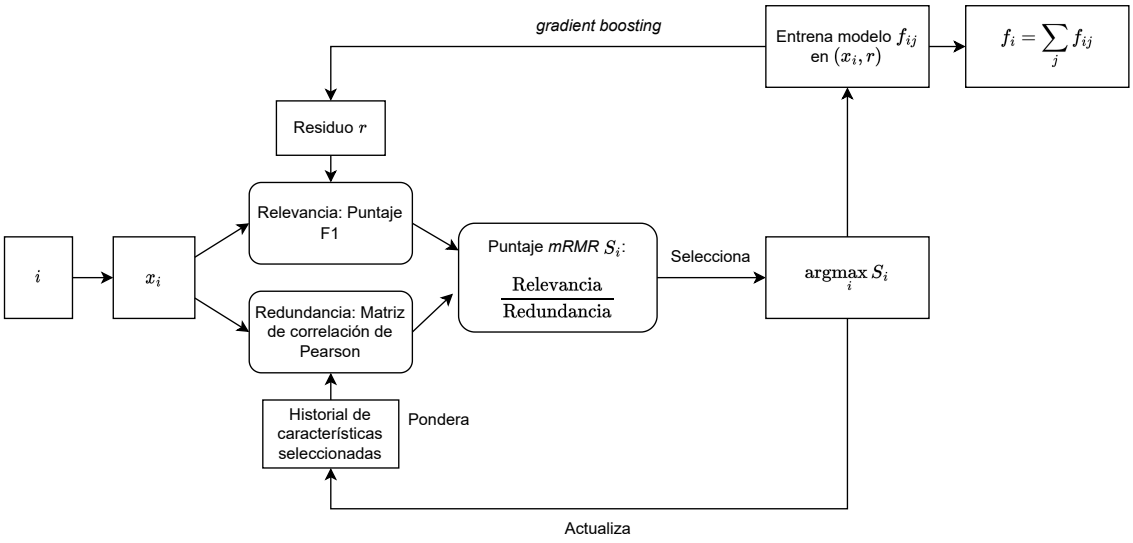


Figura 3.1: Esquema del algoritmo SAB.

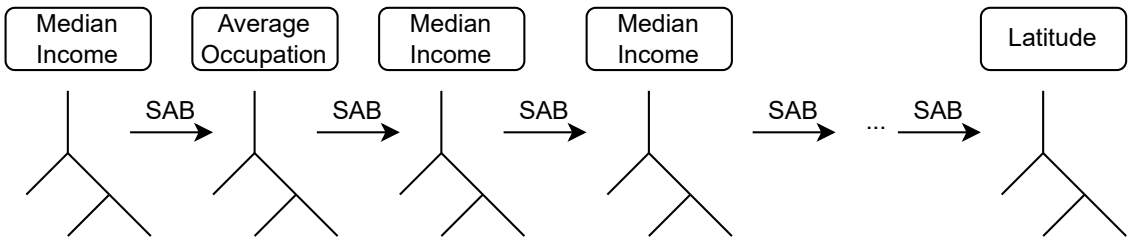


Figura 3.2: Ejemplo del funcionamiento del algoritmo para características del *California Housing Dataset* [11]. Utiliza SAB para determinar cual característica utilizar en el árbol de regresión de cada iteración del potenciamiento.

Sea h el vector donde h_j indica la cantidad de iteraciones en las que se seleccionó la variable x_j. Específicamente, la característica j es seleccionada solo si $h_j \neq 0$. Dado que la cantidad de iteraciones suele ser considerablemente mayor que el número de características, algunas características se seleccionan múltiples veces, por principio del palomar. La selección se realiza por una función ϕ, que toma los datos X, el residuo r y h en la iteración actual, y retorna el índice de la característica seleccionada. ϕ función balancea relevancia y redundancia de manera simple que modifica el algoritmo mRMR, y se define como tal:

$$\phi(X, r, h) = \begin{cases} \text{argmax} \left\{ i \leq m \middle| F(X_i, r) \right\} & \text{si } h = 0 \\ \text{argmax} \left\{ i \leq m \middle| \frac{F(X_i, r)}{(C[i]h)^\alpha} \right\} & \text{si no} \end{cases}$$

Donde $\alpha \geq 0$ es un hiperparámetro que controla el balance entre relevancia y redundancia, ρ es la correlación de Pearson, y C es la matriz definida como tal:

$$C_{i,j} = \begin{cases} |\rho(X_i, X_j)| & \text{si } i \neq j \\ 0 & \text{si } i = j \end{cases}$$

Que tiene el valor absoluto de la matriz de correlación en cada entrada salvo en la diagonal, donde solo tiene ceros.

ϕ mide relevancia de la misma manera que se mencionó antes, pero generaliza al caso de seleccionar una variable múltiples veces, al ponderar cada correlación por la cantidad de veces que se selecciona una característica (que se almacena en el vector h). C tiene 0 en la diagonal para incentivar que el modelo vuelva a seleccionar múltiples veces características que ya seleccionó antes y evite seleccionar nuevas, salvo que aporten suficiente valor en la predicción. El caso $h = 0$ se trata por separado, pues al no tener historial, no se define la redundancia, y solamente se elige la característica de mayor relevancia. Finalmente, para características que nunca se seleccionaron, f_i no se actualiza y queda en su valor inicial como la función nula.

El hiperparámetro α controla que tanto se valora que las soluciones sean poco redundantes. El caso $\alpha = 0$ la elección es equivalente a maximización de relevancia, y aumentar el valor otorga más importancia a que las características no sean redundantes, pues al crecer h el término bajo la potencia superará a 1. En otras palabras, a valor de α más grande, se eligen menos características.

Generalizando, también se puede definir de distintas formas la redundancia F, la matriz C, y otras funciones ϕ que las combinen.

Por ejemplo, F puede utilizar información mutua con respecto al objetivo o relevancia basada en otro modelo de aprendizaje, mientras que C puede incluir información mutua entre las características o coeficientes de correlación aleatorizados. Estos ajustes pueden ser realizados fácilmente por un usuario de la librería implementada que busque más robustez, velocidad, o un modelamiento más apropiado para su uso. En el caso de información mutua, experimentos preliminares mostraron un costo computacional demasiado elevado como para ser práctico, por lo que se mantuvieron como funciones por defecto las antes consideradas.

Cabe destacar que el algoritmo SAB es una base simplificada, y se puede implementar con varias modificaciones al potenciamiento de gradiente para mejorar el rendimiento y desempeño del modelo, como *stochastic gradient boosting* [43] y el algoritmo DART [44], y esto lo permite en la implementación propuesta.

3.2. Inferencia

Como se mencionó, las funciones f_i son constantes por partes, comúnmente representadas como un árbol de decisión T con una estructura de datos recursiva. Sin embargo, cuando la función es unidimensional, un árbol T puede representarse en una estructura de datos más eficiente. En particular, se considera $T = (D, H)$, donde D es un arreglo ordenado con las d divisiones del eje x, y H es un arreglo con $d + 1$ hojas correspondientes a cada división. Denominamos a esta representación como la representación de listas de un árbol, y es la que naturalmente se encontrará en el algoritmo de entrenamiento detallado en la siguiente sección. Un ejemplo de un árbol en representación de listas se encuentra en la figura 3.3.

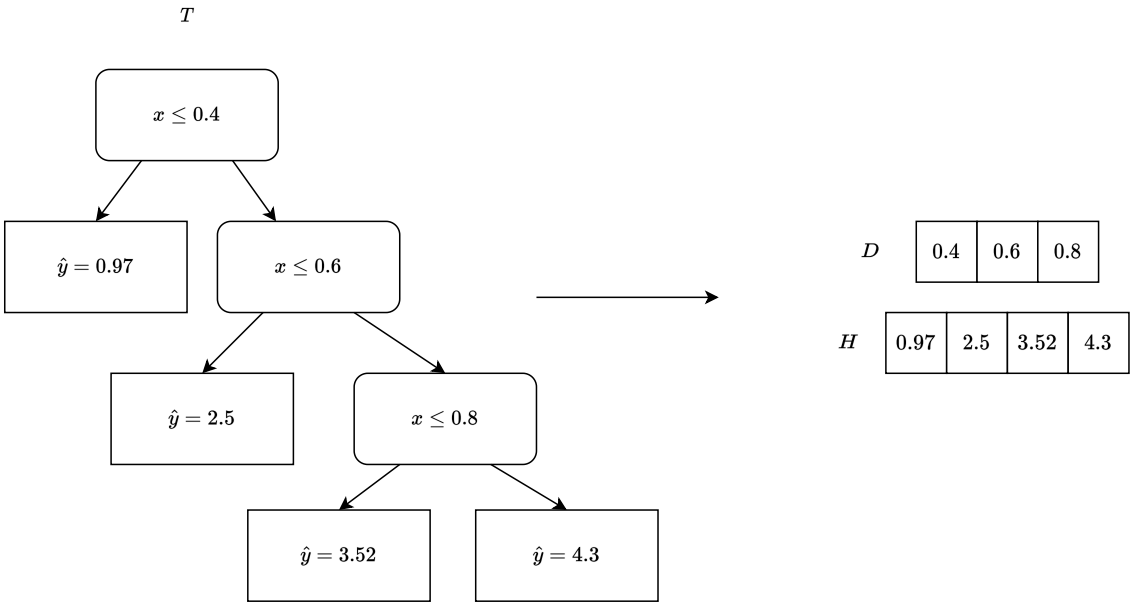


Figura 3.3: Representación de una función constante por partes en 1 dimensión.

La representación de listas permite evaluar $T(x)$ de forma simple y eficiente, basta encontrar el índice i de D que corresponda con x mediante búsqueda binaria, y luego retornar $H[i]$. Este algoritmo se ejecuta a complejidad $O(\log D)$, con una cantidad óptima de iteraciones para algoritmos de búsqueda basados en comparación [45], y se implementa de forma vectorizada para evaluar eficientemente varios valores de x en paralelo. Esta evaluación es más eficiente que la representación de árbol de búsqueda, ya que esta última puede formar un árbol desbalanceado y, por lo tanto, en el peor de los casos se evalúa con complejidad $O(D)$. Otro beneficio de la representación de listas es que si tenemos dos árboles T_1, T_2 que están entrenados en la misma característica, podemos calcular la representación del árbol $T = T_1 + T_2$ de manera sencilla y eficiente. El arreglo D es la concatenación ordenada de D_1 y D_2 (concatenar y ordenar se puede en orden $O(|D| \log |D|)$). El arreglo H tiene como primer elemento $H[0] = H_1[0] + H_2[0]$, y luego se agregan los elementos $T_1(x) + T_2(x)$ para cada $x \in D$, cálculo que también se ejecuta con complejidad $O(|D| \log |D|)$. Este proceso se puede generalizar a una colección de árboles $(T_1, T_2, \ldots, T_r)$ de forma vectorizada sin cambiar la complejidad.

Durante tiempo de entrenamiento, se aprende una colección de árboles para cada característica. Si una característica no es seleccionada, se representa por la función nula, por lo que se

considera que tiene un único árbol ($[], [0]$). En cualquier caso, la colección de árboles de cada característica i se puede comprimir a un solo árbol T_i con el algoritmo anterior.
Al igual que en EBM, para estimar β y cumplir con la condición de centralización, solo es necesario realizar una traslación de los datos, aprovechando la linealidad de la esperanza. En este caso, además, se debe considerar que los árboles cumplen la propiedad $\alpha T + c = (D, \alpha H + c)$, donde la ponderación y la suma de constantes se realizan por componentes sobre el arreglo, y el árbol que representan es el que tiene una evaluación modificada de la misma forma. Así, el algoritmo finaliza definiendo:

$$f_i = T_i - \mathbb{E}[T_i] = (D_i, H - \mathbb{E}[T_i])$$
$$\beta = \sum_{i \leq m} \mathbb{E}[T_i]$$

Como cada característica tiene a lo más n hojas, el costo de inferencia queda acotado por $O(|S| \log n)$, donde S es el conjunto de las variables seleccionadas. En la práctica, el número de hojas está acotado, resultando en un costo $O(|S|)$.

3.3. Aprendizaje de árboles

Un paso que no se detalló en el algoritmo 3 es como entrenar un árbol de regresión T en datos unidimensionales (x, y).
Anteriormente se discutió el algoritmo CART, que encuentra árboles de regresión subóptimos con un costo de entrenamiento $O(n^2 \log(n))$, pues encontrar un árbol óptimo en general es un problema NP-completo. Sin embargo, es posible aprovechar que estamos en un contexto unidimensional para mejorar significativamente la complejidad temporal.
ExplainableBoostingMachine opta por discretizar la variable x en d intervalos y calcular los estadísticos necesarios (como la suma de la variable de respuesta) por cada intervalo, evitando recalcularlos ociosamente, y así encuentra el árbol equivalente a CART, pero con complejidad lineal $O(dn)$. Si no se discretiza, o se hace una discretización trivial donde cada dato está en su propio intervalo, entonces el costo es cuadrático.
Por otro lado, en el modelo propuesto se opta por encontrar una solución óptima global al problema, lo que es posible a tiempo polinomial en el caso unidimensional utilizando programación dinámica para resolver el problema de optimización de Potts:

$$\min_{\mathcal{P}, (\mu_I)_{I \in \mathcal{P}}} \gamma(|\mathcal{P}| - 1) + \sum_{I \in \mathcal{P}} \sum_{i \in I} d(y_i, \mu_I)$$

Donde $\mathcal{P}$ es una partición de x en intervalos, μ_I es el valor de la hoja correspondiente al intervalo I, γ es un hiperparámetro que penaliza la cantidad de elementos que tenga $\mathcal{P}$ para promover el uso de árboles más pequeños, y d es una medida de fidelidad a los datos.
En la literatura ya está propuesto un algoritmo que resuelve el problema de Potts en $O(n^2)$ iteraciones [46], detallado en el anexo B, en adelante denominado algoritmo de Potts. En cada una de las iteraciones, se evalúa un intervalo $I = [a, b]$, sobre el cual se resuelve el problema de optimización:

$$\min_{\mu_I} \sum_{i \in I} d(y_i, \mu_I)$$

Por ejemplo, para la distancia cuadrática $d(y_i, \mu_I) = (y_i - \mu_I)^2$, donde se tiene que μ_I es la media muestral de $(y_i)_{i \in I}$. Idealmente, este problema debe resolverse con complejidad

constante si se permite precalcular valores con complejidad $O(n^2)$, para poder resolver el problema de Potts sin agregar a la complejidad $O(n^2)$ de sus iteraciones. En el modelo se utiliza otra función d que se propone más adelante, debido a que proporciona más robustez que la distancia cuadrática, y admite una optimización igual de eficiente.
En el algoritmo de Potts, se asume que todos los valores de x son distintos, y se ordena y de forma que $x_1 < x_2 < \dots x_n$, pero en problemas de regresión, puede ocurrir que $x_i = x_{i+1}$, con dos valores de x iguales correspondiendo a distintos valores de y. ExplainableBoostingMachine resuelve este problema promediando todos los valores de y que corresponden a un valor de x. En el algoritmo propuesto se sigue una estrategia similar, ponderando la distancia a optimizar por la cantidad de datos promediados, para que las regiones con más datos no se vean subrepresentadas en la función de pérdida. Esta idea nos puede sugerir usar la función $d(\bar{y_i}, \mu_I) = n_i(\bar{y_i} - \mu_I)^2$, donde n_i es la cantidad de puntos correspondiente al mismo valor de x_i, e $\bar{y_i}$ está dado por el promedio:

$$\frac{1}{n_i} \sum_{k \leq n_i} y[x = x_i]$$

Es el promedio de puntos y correspondientes al mismo valor de x_i. Sin embargo, el estimador de media muestral no es robusto, siendo muy susceptible a valores atípicos de y. Por lo tanto, se propone utilizar *stochastic gradient boosting* [25], que mejora el desempeño de modelos potenciados por gradientes simplemente entrenando los árboles con una fracción aleatoria s de los datos muestreada sin reposición en cada iteración. En esta propuesta, la función estará dada por $d(\bar{y_i}, \mu_I) = W_i(\bar{y_i} - \mu_I)^2$, donde:

$$\begin{aligned} \bar{y_i} &= \frac{1}{W_i} \sum_{k \leq n_i} w_i y[x = x_i] \\ W_i &= \sum_{k \leq n_i} w_i \\ w_i &\sim \text{Bernouilli}(s) \end{aligned}$$

Es decir, la media solamente se calcula en los elementos muestreados aleatoriamente. Cabe destacar que también es posible utilizar otras distribuciones para el muestreo, tales como $\text{Binomial}\left(n, \frac{s}{n}\right)$, que es equivalente a muestrear con reposición, y $\text{Poisson}(s)$, que aproxima al caso anterior por ley de los eventos infrecuentes con potencial mejor rendimiento computacional [47]. CatBoost y el modelo propuesto implementan las tres estrategias mencionadas. Por simplicidad, no se incluyeron estrategias adaptativas de muestreo como *gradient one sided sampling* o *minimum variance sampling* [43], que muestrean con mayor probabilidad los elementos que hayan causado más error en las iteraciones anteriores.
Por último, se utiliza también regularización L^2 para reducir los pesos de las hojas, tal como lo hace XGBoost, proporcionando mayor robustez al ruido y valores atípicos. Con todo lo anterior en cuenta, el problema de optimización a resolver es:

$$\min_{\mathcal{P}, (\mu_I)_{I \in \mathcal{P}}} \gamma(|\mathcal{P}| - 1) + \lambda \|(\mu_I)_{I \in \mathcal{P}}\|_2^2 + \sum_{I \in \mathcal{P}} \sum_{i \in I} W_i(\bar{y_i} - \mu_I)^2$$

Donde $\lambda > 0$ es un hiperparámetro que penaliza la magnitud del valor de las hojas de los árboles. Implementar la solución al problema solamente requiere modificaciones pequeñas al algoritmo original, detalladas en el anexo B.

Para garantizar resolver el problema en tiempo $O(n^2)$, se debe resolver el siguiente problema de forma eficiente:

$$\min_{\mu_I} \lambda|I|\mu_I^2 + \sum_{i\in I} W_i(\bar{y}_i - \mu_I)^2$$

Cuya solución es el promedio ponderado y regularizado:

$$\mu_I = \frac{\sum_{i\in I} W_i \bar{y}_i}{\lambda|I| + \sum_{i\in I} W_i}$$

Esta expresión está bien definida cuando $\lambda > 0$ o $W_i \not\equiv 0$, y ambas de estas condiciones están garantizadas en la implementación del algoritmo. Para calcularlo de forma eficiente, primero se precalculan las sumas acumuladas en los arreglos S y P:

$$S[k] = \sum_{i=0}^{k} W_i y_i$$

$$P[k] = \sum_{i=0}^{k} W_i$$

Que se puede calcular a complejidad $O(n)$ antes de iniciar las iteraciones. Luego, simplemente evaluando para $I = [a, b]$

$$\mu_{[a,b]} = \frac{S[b] - S[a-1]}{\lambda(b-a) + P[b] - P[a-1]}$$

Donde $S[-1]$ y $P[-1]$ se interpretan como 0, dado que corresponden a una suma vacía. La fórmula permite calcular $\mu_{[a,b]}$ a complejidad constante en cada iteración, resolviendo el problema a complejidad $O(n^2)$.

Cabe destacar que, aunque la complejidad temporal cuadrática puede parecer demasiado alta en comparación a la lineal lograda por EBM, en la práctica la solución propuesta logró ajustes mejores y es de 5 a 10 veces más rápida que la implementación de *scikit-learn* de árboles de decisión, usando el mismo número de hojas [48]. La figura 3.4 muestra el ajuste a datos generados por una función lineal y por una constante por tramos con ruido agregado. Los errores de ajuste y tiempos de entrenamiento se presentan en la tabla 3.1.

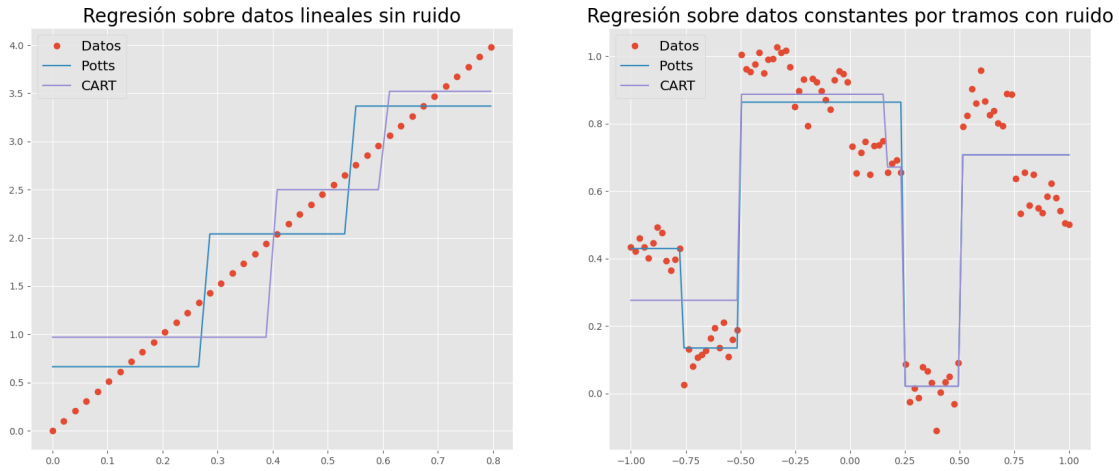


Figura 3.4: Ajuste dado por el algoritmo CART en comparación por resolución exacta del problema de Potts. En ambos casos, Potts encuentra mejores divisiones y así un mejor ajuste.

Tabla 3.1: Resultados de ambos algoritmos de entrenamiento de árboles. Potts es un modelo más veloz y tiene menor error de ajuste con la misma cantidad de hojas.

Experimento	Algoritmo	SSE	Tiempo de entrenamiento (ms)
Lineal	CART	8.64	1.37
	Potts	6.16	0.14
Constante por tramos	CART	1.59	1.34
	Potts	1.21	0.22

Una dificultad es que la relación entre el hiperparámetro de regularización γ y la profundidad del árbol no es clara, dificultando la elección de γ en la práctica. Para resolver este problema, [46] propone una manera de encontrar todos los árboles óptimos para todos los valores $\gamma \in [L, R]$ con cotas $0 \leq L < R$, a complejidad temporal $O(n^3)$, pero que en la práctica se comportaba similar a $O(n^2)$. El algoritmo primero busca un valor válido de regularización $q \in [L, R]$, y luego recursivamente en $[L, q]$ y $[q, R]$. Para evitar sobreajuste, queremos acotar el número de hojas de un árbol por un hiperparámetro p. Así, si q corresponde a una regularización con más de p hojas, no se explora $[L, q]$, ya que estas regularizaciones más pequeñas corresponden a árboles con aún más hojas. Además, esta estrategia logra reducir el espacio de exploración, mejorando así el tiempo de ejecución.
Este procedimiento se ejecuta en cada iteración del potenciamiento de gradiente, para que cada árbol pueda tener un distinto valor de γ, y al usuario solo se le piden los valores de cotas L, R. Dado que en una iteración se hayan calculado los valores de γ, se elige el árbol que minimice el error de generalización en los datos no seleccionados por el submuestreo del gradiente estocástico.

Capítulo 4

Implementación computacional

El algoritmo se desarrolló como una librería Python [49] llamada `asboostreg`. Se eligió Python por ser uno de los principales lenguajes en el desarrollo de algoritmos de aprendizaje de máquinas. Esto se debe en parte gracias a la librería *scikit-learn*, que implementa una variedad de algoritmos y especifica un diseño de API [50] que permite una fácil extensión con nuevos algoritmos.
Se adoptó esta API para simplificar el uso del algoritmo, del ambiente de pruebas y la reproducción de los experimentos. Programar siguiendo esta API es tan sencillo como se mostrará en el ejemplo de uso, y se puede combinar con otros componentes de scikit-learn para validación y búsqueda de parámetros.

4.1. Instalación y uso inicial

Para instalar la librería `asboostreg`, se puede usar el comando `pip` de la siguiente manera:

Código 4.1: Instalación de la librería

```
pip install git+https://github.com/thesis-jdgs/additive-sparse-boost-regression.git
```

Alternativamente, o en el caso de que ocurra algún error en un sistema distinto al probado, es posible clonar el repositorio `git`:

```
git clone git+https://github.com/thesis-jdgs/additive-sparse-boost-regression.git
```

En cuyo caso, debe compilar el código C en su propio sistema, ejecutando los siguientes comandos de `gcc` desde el directorio de su clon local:

Código 4.2: Ejemplo de compilación con gcc

```
cd potts
gcc -c -o l2_potts.o -fPIC -Ofast -Wall -Wextra l2_potts.c
gcc -shared -o l2_potts.dll l2_potts.o -Wall -Wextra
```

Una vez la librería esté instalada correctamente, puede utilizar el modelo tal como cualquier otro estimador de scikit-learn, instanciando la clase SparseAdditiveBoostingRegressor:

Código 4.3: Ejemplo de uso

```
from asboostreg import SparseAdditiveBoostingRegressor

reg = SparseAdditiveBoostingRegressor()
reg.fit(X_train, y_train)  # Ajusta el regresor en datos de entrenamiento, con sus etiquetas
y_pred = reg.predict(X_test)  # Predice el resultado en nuevos datos
```

Los métodos y parámetros más complejos se detallarán en la siguiente sección.

4.2. Diseño de Software

Como inspiración para el diseño, se toman 3 algoritmos de árboles potenciados por gradiente que son fuente abierta, populares y muestran gran éxito: XGBoost, CatBoost y LightGBM. Se implementan sus hiperparámetros más importantes para el despeño del algoritmo según las guías de usuario de estas librerías, en particular: cantidad de estimadores, tasa de aprendizaje, razón de submuestreo, tamaño de la discretización, penalización L^2, etc. El algoritmo se implementa como una clase de Python llamada SparseAdditiveBoostingRegressor (acortado a SABRegressor para este documento), cuyo diagrama UML se presenta en la figura 4.1.

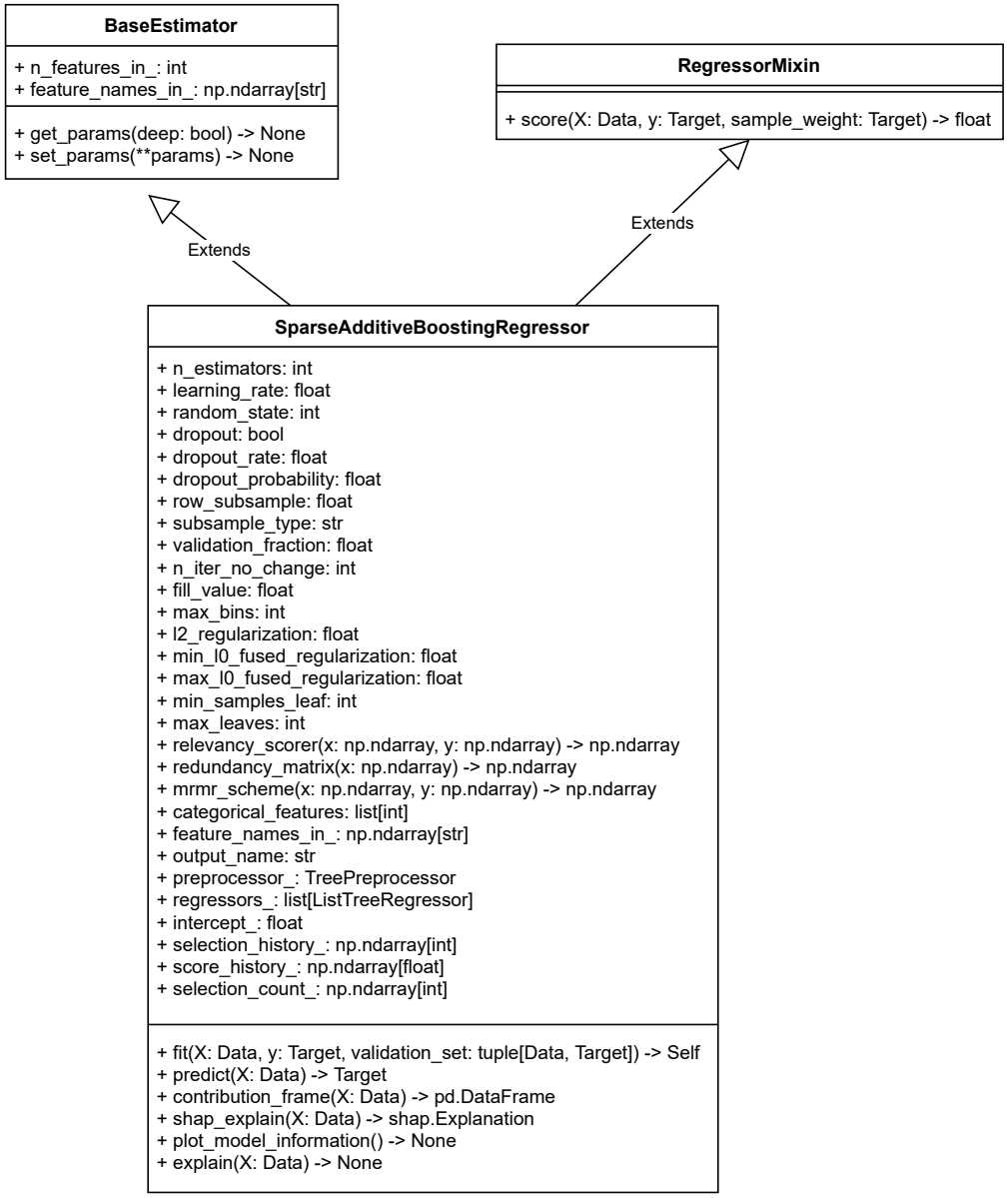


Figura 4.1: Diagrama UML del modelo SparseAdditiveBoostingRegressor.

Para preprocesar los datos, se utiliza por defecto una clase preexistente denominada TreeRegressor, que se puede modificar de ser necesario. Además, el estimador base del regresor univariado es una clase ListTreeRegressor, cuya responsabilidad es contener la información de una única variable. Sus diagramas se presentan en las figuras 4.2 y 4.3 respectivamente.

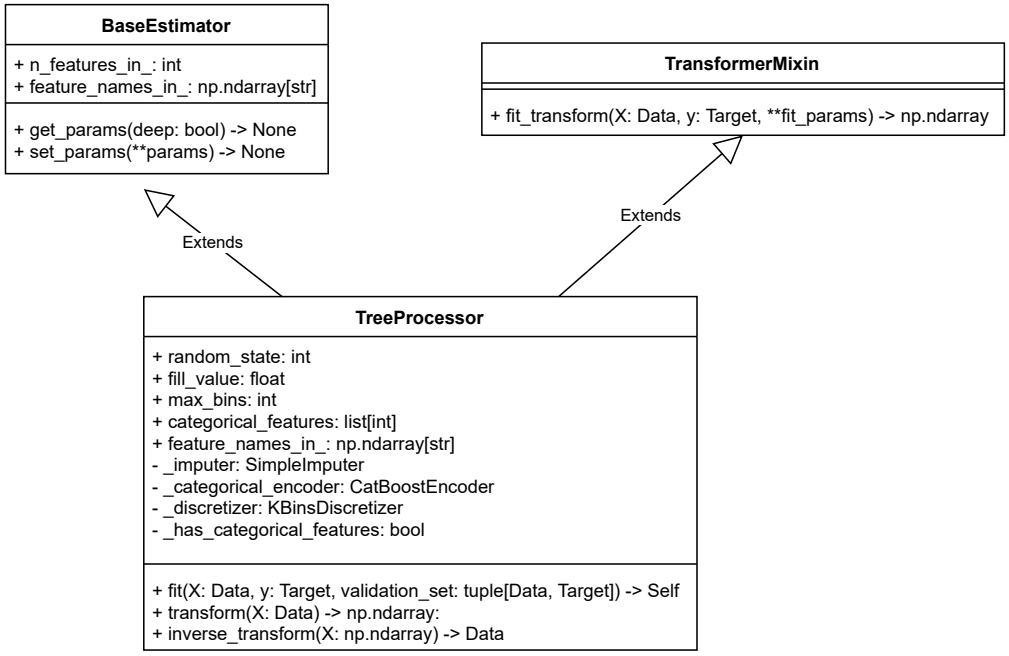


Figura 4.2: Diagrama UML del procesador TreePreprocessor

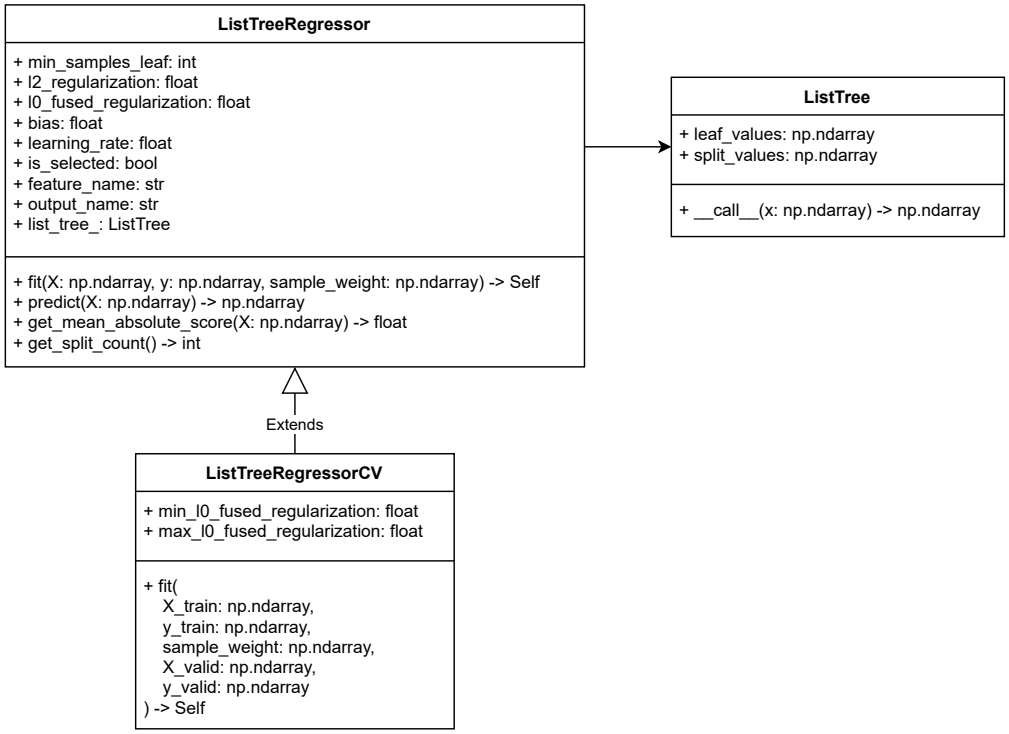


Figura 4.3: Diagrama UML del regresor univariado ListTreeRegressor

Para asegurarse que la implementación del entrenamiento e inferencia sean eficientes se utiliza la librería de cálculo numérico NumPy [51], que optimiza las operaciones de arreglos. Además, el entrenamiento de un árbol óptimo es el paso de mayor costo computacional, por lo que se programó en el lenguaje C con un envoltorio en Python para llamar a la función que resuelve el problema [52].

4.3. Documentación

El código se documentó en inglés siguiendo la guía de estilo de NumPy. A continuación, se presenta la traducción al español de la documentación simplificada de SABRegressor. No se incluyen las otras clases, pues no son relevantes para un usuario del modelo, sino para un desarrollador.
La tabla 4.1 presenta los hiperparámetros básicos utilizados para la inicialización de SABRegressor. Además, existen otros hiperparámetros avanzados que no se modificaron durante los experimentos de esta tesis, pero se incluyeron en el código y se reportan en la tabla 4.2.

Tabla 4.1: Guía de hiperparámetros básicos del modelo.

Hiperparámetro	Rango	Explicación
`n_estimators`	$\mathbb{N}_0$	Cantidad de árboles aprendidos, o iteraciones de potenciamiento.
`learning_rate`	$[0, 1)$	Tasa de aprendizaje, multiplica a cada función aprendida.
`row_subsample`	$(0, 1)$	Cantidad de datos muestreados en cada iteración para entrenar a un árbol.
`max_bins`	$\mathbb{N}_0$	El tamaño de la discretización de las características que se realiza en preprocesamiento, para acelerar el entrenamiento de los árboles.
`l2_regularization`	$\mathbb{R}_+$	Regularización L^2 sobre las hojas en el ajuste de árboles. Aplica un efecto de *shrinkage* sobre el valor de las hojas.
`min_samples_leaf`	$\mathbb{N}_+$	El mínimo número de valores que puede tomar una hoja de un árbol.
`min_leaves`	$\mathbb{N}$	El máximo número de hojas de un árbol.
`redundancy_exponent`	$\mathbb{R}_+$	Penalización sobre la redundancia.
`dropout`	$\{0, 1\}$	Si se utiliza el mecanismo de *dropout* o no. No funciona con *early stopping*. Si es 0, desactiva los otros parámetros de *dropout*.
`dropout_probability`	$(0, 1)$	Probabilidad de que se utilice el mecanismo de *dropout* en una iteración, es decir que se remuevan estimadores pasados del ensamble.
`dropout_rate`	$(0, 1)$	Proporción de estimadores removidos en una iteración de *dropout*.

Tabla 4.2: Guía de hiperparámetros avanzados del modelo.

Hiperparámetro	Rango	Explicación
`random_state`	$\mathbb{N}$	Semilla para el generador de números aleatorios usado para muestrear datos en cada iteración.
`sampling_type`		La distribución de muestreo: `bernouilli`, `binomial` o `poisson`
`validation_fraction`	$(0, 1)$	En caso de no proveer un conjunto de validación al método `fit`, es la proporción de datos que se usan para validación interna del método, usando `1 - validation_fraction` de la proporción de datos para el entrenamiento.
`n_iter_no_change`	$\mathbb{N}_0$	La cantidad máxima de iteraciones que el modelo puede continuar entrenando sin mejorar su desempeño en el conjunto de validación.
`l0_fused_regularizations`	$\mathbb{R}_+^2$	Valores mínimos y máximos de regularización L^0 sobre valores consecutivos de las hojas en el ajuste de árboles. Controla la cantidad de hojas del árbol.
`fill_value`	$\mathbb{R}$	Valores para imputar a los valores faltantes.
`relevancy_scorer`	$\mathbb{R}^{n\times m} \mapsto \mathbb{R}^m$	Función para evaluar la relevancia de las características.
`redundancy_matrix`	$\mathbb{R}^{n\times m} \mapsto \mathbb{R}^{m\times m}$	Función para calcular la matriz de redundancia.
`mrmr_scheme`	$\mathbb{R}^m \times \mathbb{R}^m \mapsto \mathbb{R}^m$	Esquema de mínima redundancia y máxima relevancia a utilizar, es decir, que combine ambas funciones anteriores.
`categorical_features`	$\mathcal{P}(\mathbb{N})$	Índices de las características categóricas.
`feature_names_in`		Nombres de las características, para usar en gráficos.
`output_name`		Nombre del objetivo, para usar en gráficos.

Después del ajuste, la clase guarda la información aprendida en sus atributos. Los atributos públicos (que el usuario puede acceder) que son modificados en tiempo de entrenamiento se denotan con un guion bajo al final, que por simplicidad se ignora en este reporte. Consisten en:

- `preprocessor`: Un estimador de scikit-learn que preprocesa los datos. Se configura según la forma de los datos y los otros hiperparámetros.
- `regressors`: Una lista con las funciones f_i.
- `intercept`: El valor de β.
- `selection_history`: Un arreglo con el índice de la característica seleccionada en cada iteración del algoritmo.
- `score_history`: Un arreglo con el puntaje de validación en cada iteración del algoritmo.
- `selection_count_`: Un arreglo con la cantidad de veces que es seleccionada cada característica.

Por último, se detallan los métodos públicos de la clase. En general, si un método pide una colección de datos, no necesariamente debe ser un arreglo de NumPy, pero debe poder convertirse en uno con el método `np.asarray`.

- `fit`: Entrena el modelo. Retorna una instancia de la clase ya entrenada. Parámetros:
  - X datos de entrenamiento.
  - y objetivo de entrenamiento, debe poder convertirse a un arreglo de NumPy.
  - `validation_set` es un conjunto (X', y') con el cual validar. Si no se provee, entonces se dividen los datos (X, y) en conjuntos de entrenamiento y validación.
- `predict`: Predice el valor de un conjunto de instancias. Retorna las predicciones de cada instancia como un arreglo de NumPy. Falla si no se ha llamado a `fit` antes.:
  - X conjunto de instancias a predecir.
- `contribution_frame`: Entrega un DataFrame de Pandas con los valores de f_i para una colección de instancias. Falla si no se ha llamado a `fit` antes.
  - X conjunto de instancias a evaluar.
  - `only_selected`: Valor de verdad indicando si se consideran características no seleccionadas. Por defecto, falso.
- `shap_explain`: Entrega un objeto de Explicación de la librería SHAP, para graficar los valores de SHAP, con distintos resúmenes. Falla si no se ha llamado a `fit` antes.
  - X conjunto de instancias a explicar.
  - `only_selected`: Valor de verdad indicando si se consideran características no seleccionadas. Por defecto, falso.
- `explain`: Grafica todas las funciones f_i seleccionadas (es decir, no nulas) en los datos entregados.

- X conjunto de instancias a graficar. Pueden ser los datos de entrenamiento para inspeccionar el modelo exacto.

- `plot_model_information`: Grafica información almacenada del modelo que no requiere datos de entrada, como las características seleccionadas, la curva de validación, la complejidad de los estimadores y la importancia promedio de cada característica.

4.4. Mecanismos de interpretabilidad

Por simplicidad de uso, e inspirado por EBM, la clase SABRegressor implementa métodos que generan explicaciones automáticas del modelo, como gráficos interactivos de *plotly* [53], que permiten al usuario modificarlas fácilmente. El texto de los gráficos se generó automáticamente por el código, y por lo tanto están en inglés al igual que la documentación. Cada forma de interpretabilidad se detallará en las siguientes secciones.

4.4.1. Explicación por valores de SHAP

Para un dato $x \in \mathbb{R}^m$, tenemos que los valores de Shapley están dados por el vector $(f_i(x_i))_{i \leq m}$. Para calcuarlo, se implementó el método `contributions_matrix(X)`, que recibe una matriz de datos $X \in \mathbb{R}^{n \times n}$ y retorna una matriz de contribuciones $F \in \mathbb{R}^{n \times n}$ tal que $F_{ij} = f_j(x_{ij})$ para $i \leq n, j \leq m$, de la siguiente forma:

$$X \mapsto F$$

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix} \mapsto \begin{bmatrix} f_1(x_{11}) & f_2(x_{12}) & \cdots & f_m(x_{1m}) \\ f_1(x_{21}) & f_2(x_{22}) & \cdots & f_m(x_{2m}) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_{n1}) & f_2(x_{n2}) & \cdots & f_m(x_{nm}) \end{bmatrix}$$

Cabe recordar que la selección de características significa que varias de las funciones son idénticamente nulas, por lo que varias columnas de F son nulas.
La matriz F se transforma a un `DataFrame` de Pandas [54][55], debido a su facilidad de lectura, manejo en ciencias de datos, y uso para gráficos. Además, se implementa el método `shap_explain`, que transforma la matriz a un objeto `Explanation` de la librería SHAP [29] [56], permitiendo utilizar cualquiera de sus funcionalidades de visualización ya existentes, como muestran la figuras 4.4 y 4.5, entre otras posibilidades que se muestran en la siguiente sección.

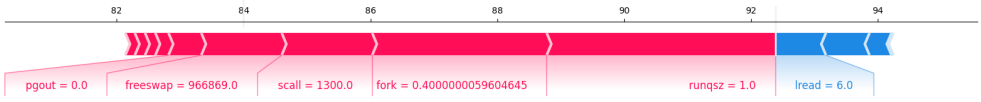


Figura 4.4: Explicación local de SABRegressor para una instancia, generada gracias a la librería SHAP. Acá se visualiza la contribución de cada característica como una "fuerza" con una dirección, que se suman para llegar al valor de la predicción.

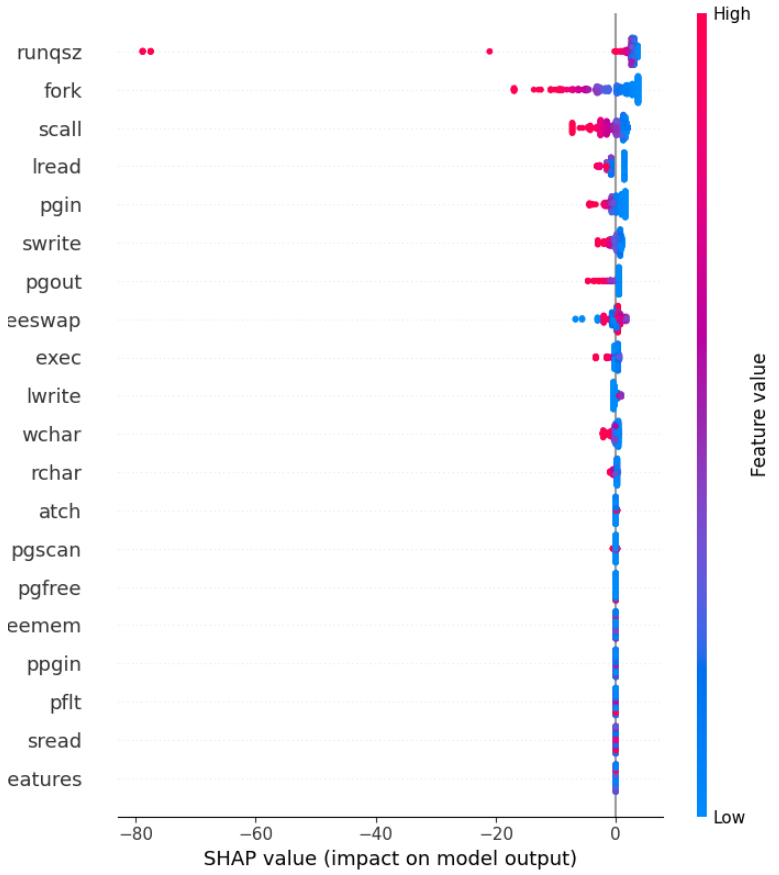
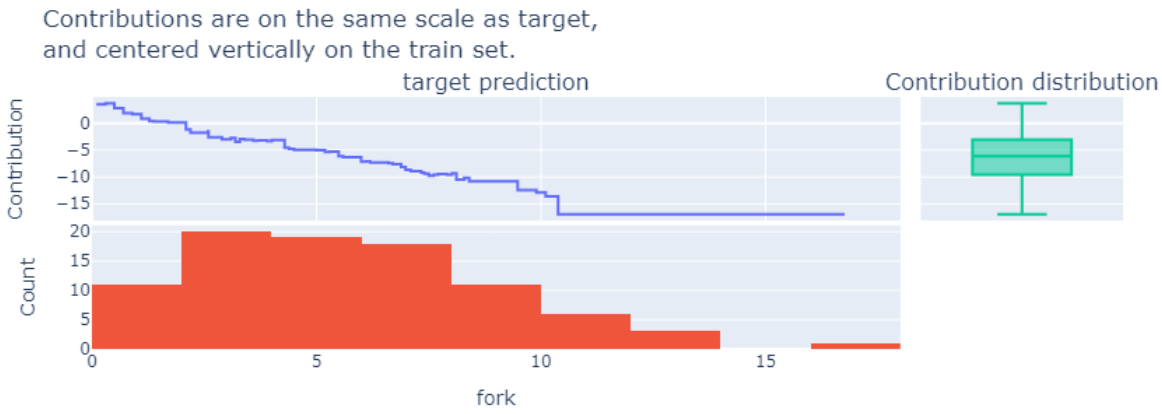


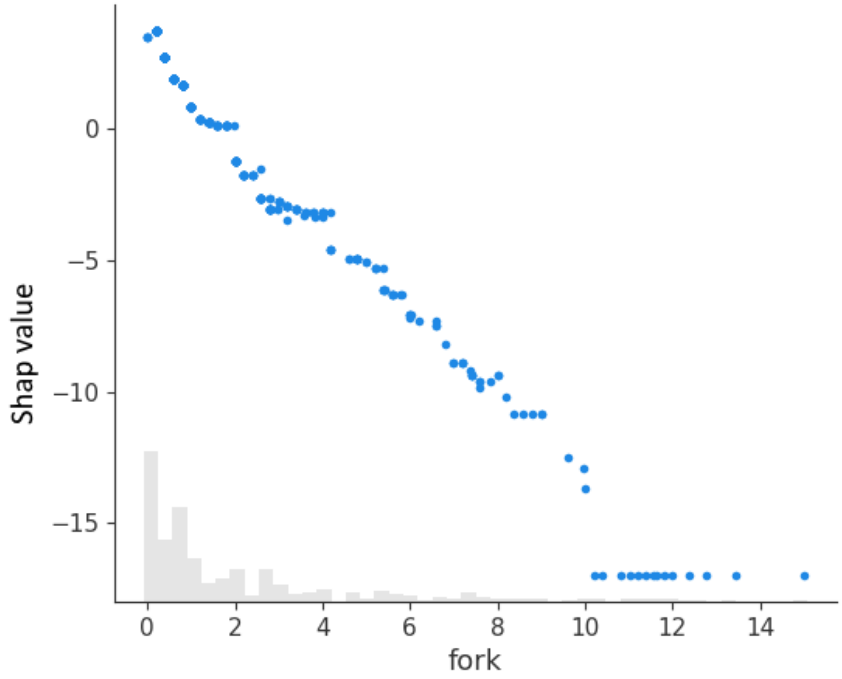
Figura 4.5: Explicación global de SABRegressor, mostrando la distribución generada de los valores de Shapley, generada gracias a la librería SHAP. Es similar a una representación de las funciones f_i, donde la entrada es el color y la salida es la contribución.

4.4.2. Explicación gráfica de los modelos aditivos

Para un conjunto de datos $X \in \mathbb{R}^{n \times n}$, se puede resumir la explicación global de la característica i-ésima como el gráfico $f_i(X_{\cdot,i})$. El método `explain(X)` genera los gráficos f_i de cada una de las características seleccionadas (ignorando las nulas).
Es importante entender la distribución de la entrada y salida de la función, por lo que el gráfico de $f_i(X_{\cdot,i})$ lo resume en sus secciones marginales. En el inferior, tiene un histograma de $X_{\cdot,i}$, que permite ver si una sección fue entrenada con muchos o muy pocos datos, indicando un nivel de confianza a la predicción. Al lado derecho, se tiene un gráfico de caja de la distribución de $f_i(X_{\cdot,i})$, que muestra el rango intercuartil como medida robusta de variabilidad y cantidad de valores atípicos.
Cabe destacar que como los valores de predicción son precisamente los valores de Shapley, un gráfico similar también se puede generar con la librería SHAP. Se pueden ver ejemplos de una función de característica graficada por SHAP y por `asboostreg` en las figuras 4.6 y 4.7.

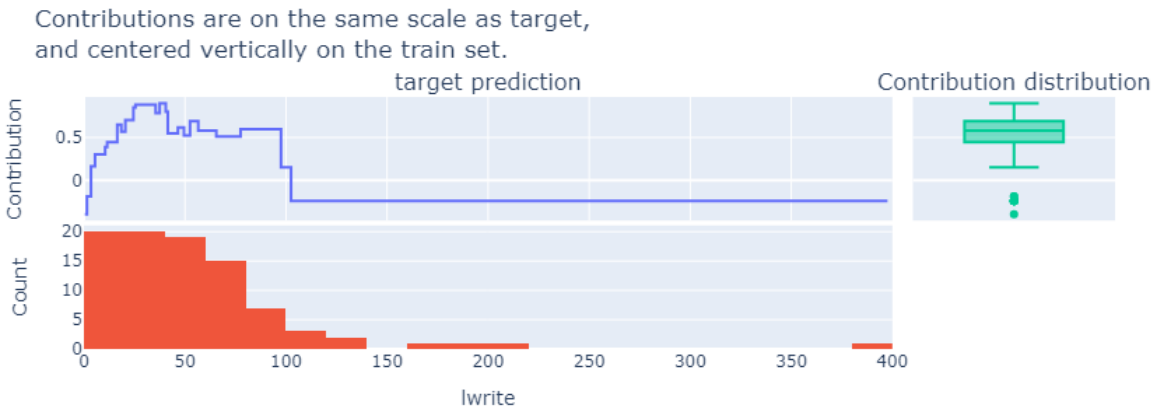


(a) Gráfico generado por `asboostreg`.

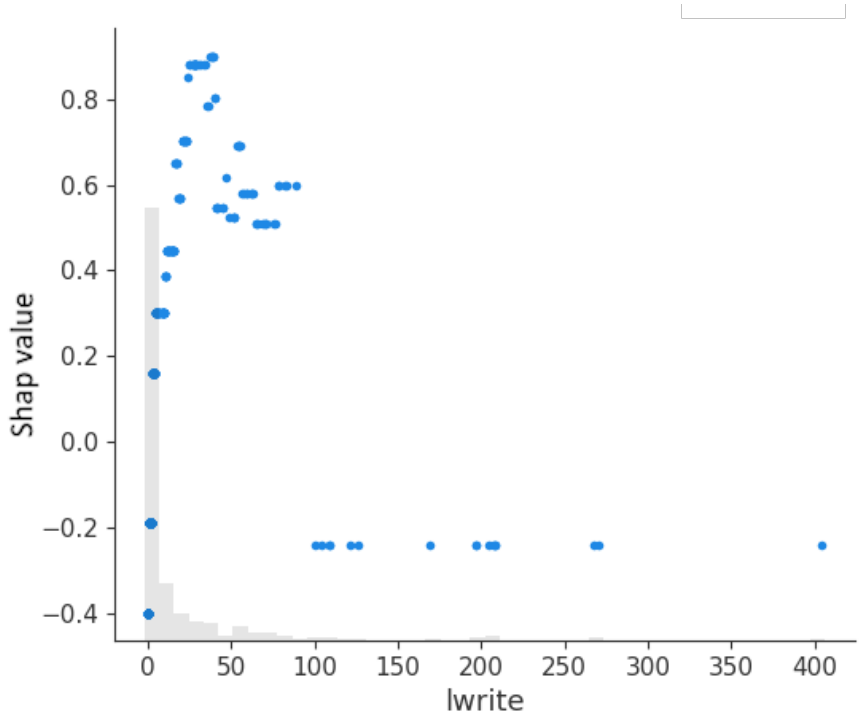


(b) Gráfico generado por la librería SHAP.

Figura 4.6: Función f_i para un problema, graficada tanto por `asboostreg`, como por la librería SHAP, utilizando su equivalencia



(a) Gráfico generado por `asboostreg`

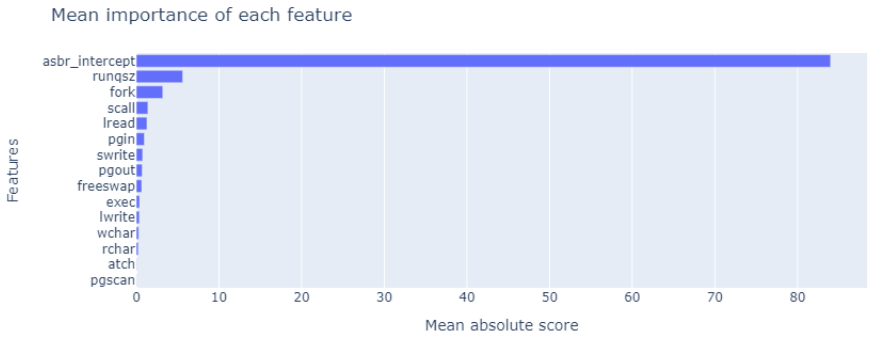


(b) Gráfico generado por la librería SHAP.

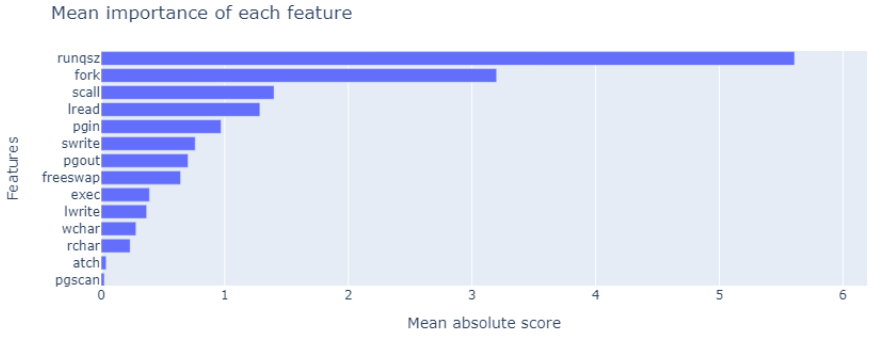
Figura 4.7: Función f_i para un problema, graficada tanto por `asboostreg`, como por la librería SHAP, utilizando su equivalencia

Además, como muestra la figura 4.8, se genera un gráfico de barras que resume la importancia de características, medida como el promedio del valor absoluto de sus contribuciones, es decir, de los valores:

$$I(i) = \frac{1}{n} \sum_{i \leq n} |f(x_i)|$$



(a) Importancia de características, considerando la constante $|\beta|$, para comparar con el valor base.



(b) Importancia de características, sin considerar la constante $|\beta|$, para comparar las características entre ellas.

Figura 4.8: Promedio de las contribuciones del valor de SHAP de cada característica utilizada por el modelo.

La asignación de importancia coincide con el valor que considera la función `summary` de la librería SHAP, que permite generar el mismo gráfico, como se ve en la figura 4.9. El gráfico generado por SHAP, sin embargo, no considera que el modelo selecciona características y las muestra teniendo una importancia 0, en vez de removerlas del gráfico por completo.

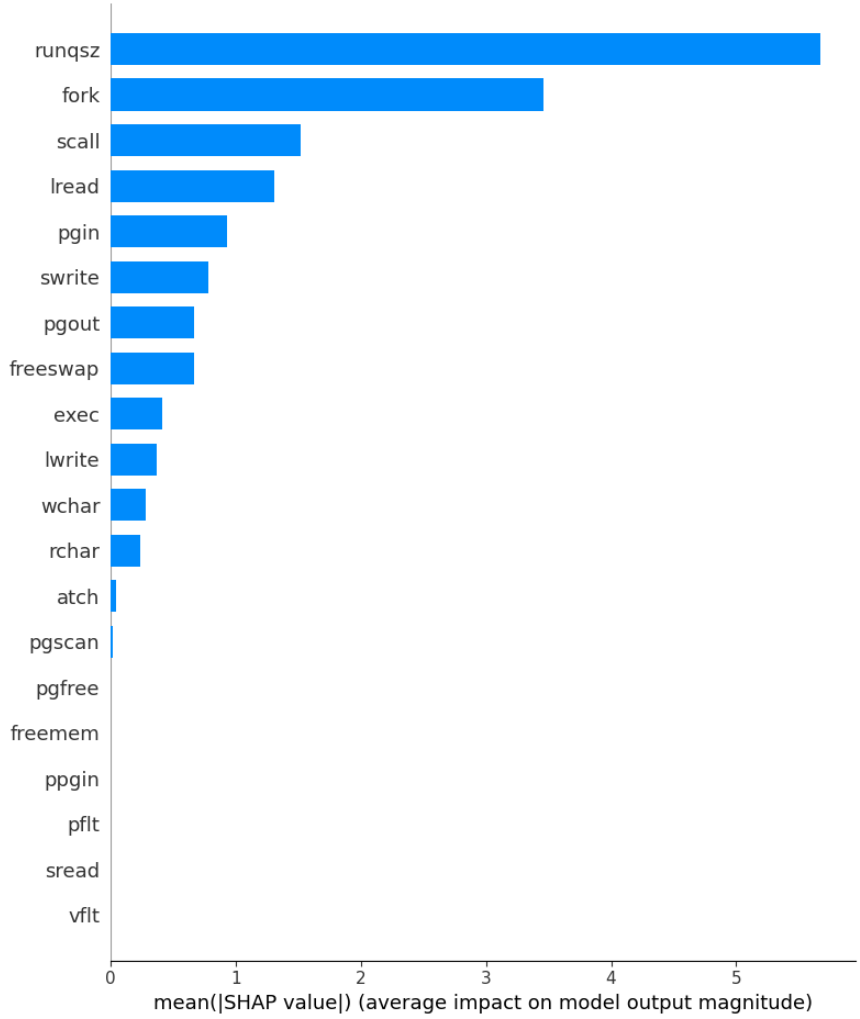


Figura 4.9: Promedio de los valores absolutos del valor de SHAP de cada característica, como su importancia global en SABRegressor, generada gracias a la librería SHAP. Vemos que algunos son nulos, por la selección de características, y la propiedad de "jugador vacío" de los valores de Shapley, que asignan importancia nula a características no utilizadas. Es el equivalente de la librería SHAP a la figura 4.8.b.

Si se llama a `explain(X)` en los mismos datos de entrenamiento, los gráficos construidos son exactamente las funciones f, mientras que explicar otros datos solamente evalúa las funciones en ellos.

4.4.3. Resumen del modelo

El método `plot_model_information()` permite obtener otras explicaciones globales del modelo f, generando 2 gráficos que explican información del modelo en general, y no de una colección de instancias.
El primer gráfico muestra las curvas de entrenamiento y validación, que miden la raíz del error cuadrático medio (RECM) cada iteración en cada uno de los conjuntos de datos, mostrando el error en el eje y y la cantidad de iteraciones en el eje x. Comparando ambas curvas, se pueden encontrar patrones de subajuste y sobreajuste, que pueden ser mejor entendidos al

además graficar cual es la característica elegida en cada iteración, pues permite visualizar si aprender una característica en particular causó un aumento o disminución en una de las curvas de error. Ejemplos están en las figuras 4.10 y 4.11.

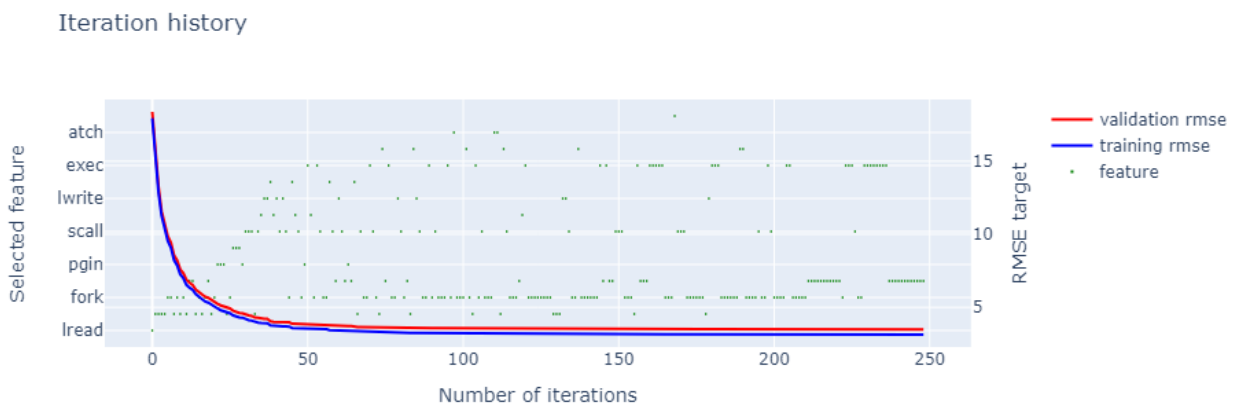


Figura 4.10: Curva de aprendizaje del modelo de predicción de actividad de CPU. Vemos mejoras en el error, tanto en entrenamiento como en validación, además de la introducción de nuevas características.

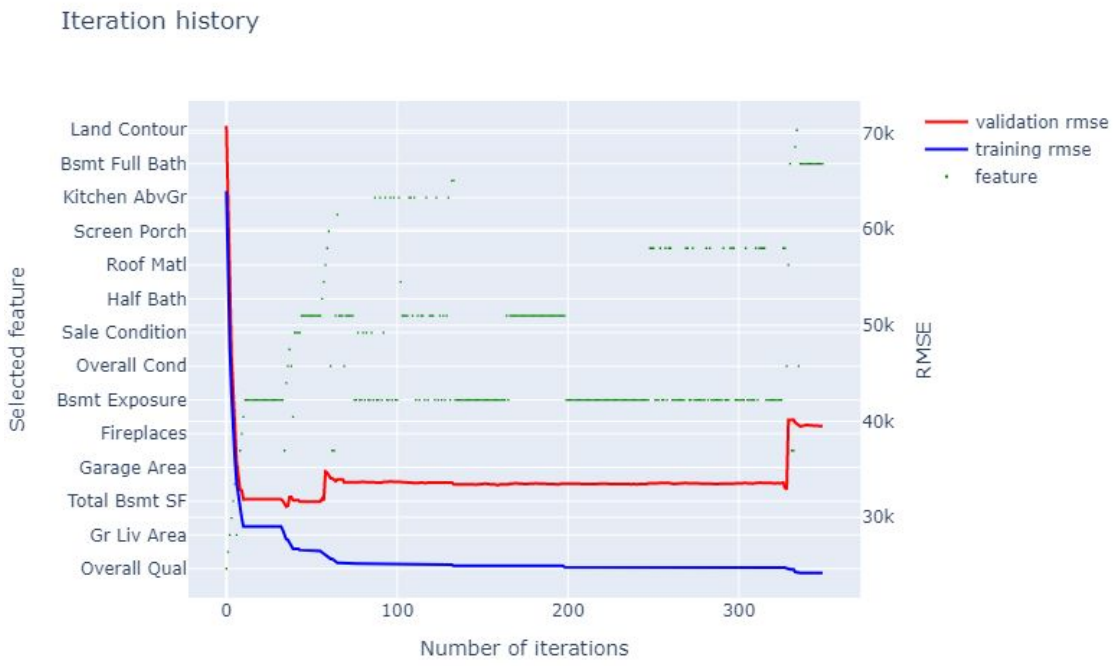


Figura 4.11: Curva de aprendizaje del modelo de predicción de valores de casas de Ames [57]. Acá vemos que ciertas iteraciones causan un grave sobreajuste, donde el error de entrenamiento disminuye, y el de validación aumenta.

En el primer ejemplo, vemos una curva de aprendizaje ideal, donde el error de entrenamiento y de validación decrece de forma similar al aumentar las iteraciones, es decir, no hay subajuste ni sobreajuste. En el segundo ejemplo, vemos un caso con un error de grave sobreajuste. En ciertas iteraciones, el error de entrenamiento disminuye, y el de validación aumenta. Al inspeccionar el gráfico, se notó que el modelo utiliza casi exclusivamente las características "Bsmt exposure" y "Pool QC", pero en las dos iteraciones en las cuales se seleccionó "Roof Matl", el error de validación disparó. En general, inspeccionar la curva de aprendizaje permite a un desarrollador o científico de datos que utilice SABRegressor realizar "debugging", pues identifica una variable que potencialmente cause contaminación de datos, de una forma que modelos multivariados no serían capaces de identificar.
Por último, el método `explain` genera un gráfico de barras que mide la "complejidad" de una función f_i, medida como el número de nodos de la función. Funciones con más complejidad pueden haber sido más utilizadas, mostrando relevancia, pero también pueden ser excesivamente ruidosas, por lo que es útil investigarlas. Funciones de baja complejidad tienen pocos nodos, pueden haber sido subajustadas o capturan un comportamiento simple. Un ejemplo se visualiza en la figura 4.12.

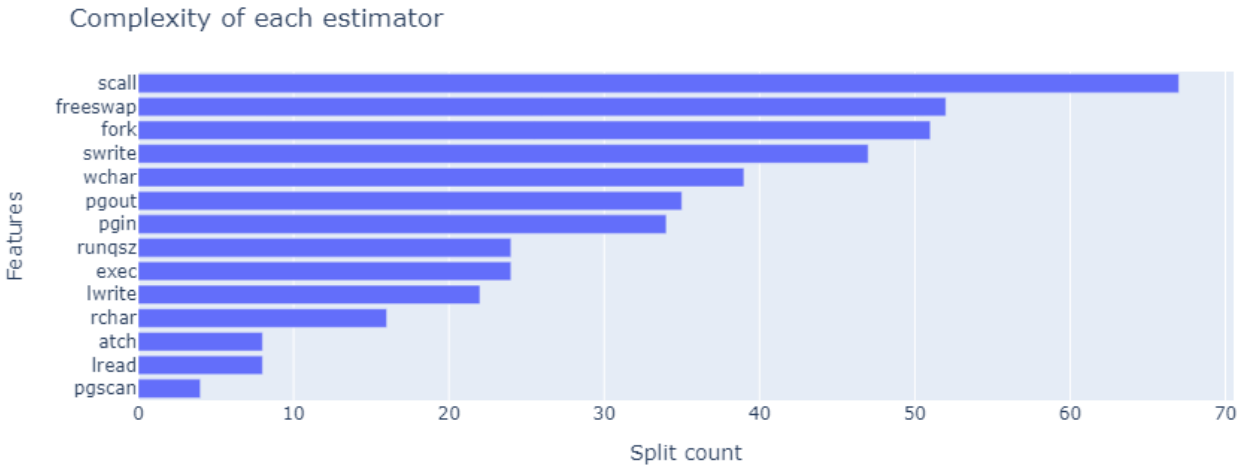


Figura 4.12: Ejemplo de cantidad de nodos utilizados por cada característica de un modelo. Vemos que *scall* es un árbol con más de 60 hojas, mientras que *pgscan* necesita menos de 5.

4.5. Preprocesamiento y detalles de implementación

Por simplicidad de uso, el modelo primero preprocesa los datos de una manera compatible con los modelos basados en árboles. En particular:

- Discretización: Existe una enorme ventaja al discretizar las variables a d valores únicos antes de entrenar los árboles, pues reduce la cantidad de posibles divisiones a d (y así, los algoritmos de la sección anterior cambian su complejidad de función a d en vez de n), además de tener un efecto regularizador. Por eso mismo, es implementado como un paso necesario por las librerías de árboles potenciados por gradiente CatBoost, LightGBM y EBM, y con el lanzamiento de la versión 2.0.0 de XGBoost, se volvió parte de su algoritmo por defecto. La discretización agrega un costo $O(d \log d)$ al entrenamiento y $O(\log d)$ a la predicción, que usualmente se ignora pues en general $d \ll n$, con un valor

por defecto de $d = 256$ en las otras librerías, dándole al usuario la opción de modificarlo.

- Imputación de datos: Los datos inválidos se rellenan con valores extremos como 10.000, pues los árboles pueden separarlos, pero sí puede sesgar el valor de los datos en los extremos.

- Variables constantes: Todas las variables con 1 valor único se eliminan, pues no aportan información a los modelos, y tienen correlación indefinida, haciendo al algoritmo fallar.

- Variables categóricas: Se usa el codificador propuesto por CatBoost [9], que transforma las categorías a valores numéricos de manera que mantenga correlaciones con el objetivo de entrenamiento de forma robusta y sin causar contaminación de datos.

Como otros detalles de implementación, se regulariza multiplicando cada árbol por una tasa de aprendizaje μ, que se puede calcular de forma simple pues $\mu T = (D, \mu H)$, es decir, la estructura de árbol no cambia. Esta identidad también permite recalcular los pesos de cada árbol en el caso de usar *dropout*.
Además, se implementa *early stopping*, es decir, se separa un conjunto de validación, y se evalúa en cada iteración el desempeño del modelo hasta el momento, deteniendo el entrenamiento cuando el desempeño en validación deja de mejorar por una cantidad de iteraciones. No se recomienda combinar *early stopping* con *dropout*, pues el peso de árboles de iteraciones anteriores no es constante.

Capítulo 5

Estudio experimental: Comparación de desempeño

En el primer estudio, se quiere evaluar el desempeño del modelo en términos de error de generalización de forma robusta, comparando un puntaje de generalización con el de otros modelos competitivos en una colección de conjuntos de datos.

5.1. Metodología

Para evaluar el desempeño, es necesario primero proponer y definir los siguientes elementos:

- Regresores base de comparación
- Una metodología de validación cruzada
- Una colección de problemas de regresión, con sus conjuntos de datos para entrenamiento
- Una metodología para normalizar el número de características utilizado

Cada parte se detallará en una de las secciones siguientes. Todos los experimentos se realizan en un computador personal, con especificaciones detalladas en el anexo D.

5.1.1. Regresores a comparar

Los modelos por comparar se eligieron por que se espera que puedan competir con SABRegressor por su interpretabilidad, integración de selección de características, o por su alto desempeño. En base a estos parámetros, se compara el desempeño de SABRegressor con los modelos propuestos en la tabla 5.1.

Tabla 5.1: Modelos a comparar

	Fuerte	Selección de características	Interpretable
Árboles de decisión (CART)		✓	✓
Modelo lineal (Ridge)			✓
Random Forest	✓		
XGBoost	✓	✓	
Explainable Boosting Machine	✓		✓
Sparse Additive Regressor	?	?	✓

Algunos algoritmos requieren distintos pasos de preprocesamiento. En la tabla 5.2 se incluye el código Python creado para inicializar cada uno de los modelos, incluyendo la semilla aleatoria por reproducibilidad.
Dado que el modelo propuesto todavía no tiene un conjunto de hiperparámetros por defecto definidos que funcionen en amplios conjuntos de datos, se ejecuta búsqueda de hiperparámetros con el algoritmo *Tree-Structured Parzen Estimator* [58], implementado en la librería Optuna [59]. El rango de los valores aceptados en la búsqueda se encuentran en la tabla 5.3.

Tabla 5.3: Grilla de búsqueda de hiperparámetros.

Hiperparámetro	Rango
`n_estimators` (I)	$[500, 10000]$
`learning_rate` (μ)	$[0.001, 0.5]$
`random_state`	0
`row_subsample` (s)	$[0.15, 0.9]$
`sampling_type`	{`bernouilli`}
`validation_fraction`	$\{0.15\}$
`n_iter_no_change`	15
`max_bins` (d)	[64, 4096]
`l2_regularization` (λ)	[0.01, 10.0]
`l0_fused_regularizations`	$\{(0, 100)\}$
`min_samples_leaf` (ℓ)	[1, 15]
`max_leaves` (p)	[3, 64]
`redundancy_exponent` (α)	[0.0, 3.0]
`dropout_probability` (p_{drop})	[0.0, 0.5]
`dropout_rate` (r_{drop})	[0.0, 0.1]

5.1.2. Validación

Para medir el error en cada conjunto de datos, proponemos un puntaje de regresión S, que mide robustamente que tan buen predictor es $\hat{y}$ del objetivo real y, al evaluarlo con validación cruzada 5-*fold*. El puntaje S se define como tal:

$$S(\hat{y}, y) = 1 - \frac{\text{MAE}(\hat{y}, y)}{\text{MAE}(\hat{y}, \text{median}(y))}$$

El puntaje S es similar a R^2, está en el rango $(-\inf, 1]$, donde un puntaje de 1 significa predicción perfecta, 0 es el puntaje del predictor constante de la mediana (que minimiza el MAE), y valores negativos son aún peores que un modelo constante. La diferencia es que el puntaje R^2 se define con error cuadrático medio (y su minimizador, la media), que es una medida de error menos robusta a valores atípicos. Para cada problema y modelo, se calcula la media y el error estándar de los 5 puntajes S. Se reportan estas cantidades a dos cifras significativas, salvo que el redondeo haga que el error sea nulo, en cuyo caso se omite por simplicidad.
Para manejar los expeirmentos de forma organizada y trazable, se diseñan experimentos con la plataforma MLflow [60].
Además, se valida la correctitud de cada uno de los *folds* con validación cruzada adversarial [61], donde se utiliza un modelo de clasificación (en este caso, XGBoost) para buscar discriminar si un dato está en el conjunto de entrenamiento o en el de validación. La clasificación debería ser similar a una aleatoria (con AUC cercano a 0.5), pues de otra forma, las distribuciones de entrenamiento y validación son distinguibles.

5.1.3. Datos

Se comparara el error de generalización de los modelos a comparar en varios conjuntos de datos de *Penn Machine Learning Benchmark* [62], una colección de 128 conjuntos de datos tabulares diverso en problemas de regresión. De estos conjuntos, se eligen 7 para realizar la evaluación.
Por simplicidad, solamente se consideran conjuntos de datos en los que se pueda alcanzar un buen desempeño sin requerir ingeniería de características. Para encontrarlos, primero de ejecutaron los otros modelos clásicos $\mathcal{M}$, y se filtraron los conjuntos de datos donde el mejor puntaje $\max_{m \in \mathcal{M}} S_m$ sea menor que 0.7, quedando con 37 conjuntos de datos en total. Para que los problemas a evaluar tengan una escala realista y relevante para un modelo que selecciona características, se filtran también conjuntos de datos con menos de 1000 registros o menos de 10 características, por lo que se acaban con 7 conjuntos de datos, que se describen en el anexo C.

5.1.4. Normalización

Se quiere verificar si los modelos son comparables al igualar el número de características, así que una vez que se haya completado el experimento anterior, se evalúa el número de características k que utiliza SABRegressor, y se restringe al resto de modelos a solamente utilizar a lo más k características, a través del algoritmo 5.

Algoritmo 5: Normalización del número de características

Data: $X \in \mathbb{R}^{n \times m}$ datos, $y \in \mathbb{R}^n$ objetivo, modelo M, SABRegressor S entrenado en los datos

Result: Modelo M' entrenado en tantas características como S

$k \leftarrow S$.seleccionadas;

Entrena M en (X, y);

$X' \leftarrow k$ características con importancia más grande según M;

Entrena M' en (X', y);

Retorna M'

Tabla 5.2: Código Python para generar cada modelo, utilizando clases y funciones de utilidad de scikit-learn.

Modelo	**Código Python**

Ridge	`make_pipeline(`
	`  StandardScaler(),`
	`  RidgeCV(),`
	`)`

DT	`DecisionTreeRegressor(`
	`  random_state=0,`
	`  max_depth=3,`
	`)`

RF	`make_pipeline(`
	`  KBinsDiscretizer(`
	`  random_state=0,`
	`  n_bins=255,`
	`  encode="ordinal",`
	`  ),`
	`  RandomForestRegressor(`
	`  random_state=0,`
	`  n_estimators=50,`
	`  ),`
	`)`

XGB	`XGBRegressor(`
	`  n_estimators=50,`
	`  random_state=0,`
	`)`

EBM	`ExplainableBoostingRegressor(`
	`  random_state=0,`
	`  max_rounds=1_000,`
	`  interactions=0,`
	`)`

5.2. Resultados

Tabla 5.4: Puntajes promedio (y error estándar, cuando es significativo) de regresión en calculado en 5-folds para cada conjunto de datos y modelo. Mayor significa un mejor puntaje, 1 representa un puntaje perfecto.

Datos	SAB	Ridge	EBM	DT	RF	XGB
215_2dplanes	0.00	0.47	0.47	0.56	0.08	0.78
344_mv	0.61	0.60	0.61	0.74	0.99	0.99
562_cpu_small	0.75	0.36	0.77 (0.01)	0.59 (0.01)	0.79	0.80
197_cpu_act	0.78	0.37 (0.01)	0.80 (0.01)	0.62 (0.03)	0.82	0.83
227_cpu_small	0.75	0.36 (0.02)	0.77 (0.01)	0.59 (0.03)	0.79	0.80
564_fried	0.66	0.50	0.67	0.36	0.75	0.77
201_pol	0.40	0.08	0.38 (0.01)	0.55 (0.01)	0.94	0.92

Tabla 5.5: Puntajes promedio (y error estándar, cuando es significativo) de regresión en calculado en 5-folds para cada conjunto de datos y modelo. Mayor significa un mejor puntaje, 1 representa un puntaje perfecto. Todos los modelos seleccionan un subconjunto de características más relevantes.

Datos	Seleccionadas	Totales	Ridge	EBM	DT	RF	XGB
215_2dplanes	1	10	0.26	0.26	0.26	0.03	0.26
344_mv	3	10	0.59	0.61	0.73	0.99	0.98
562_cpu_small	11	12	0.36 (0.03)	0.76	0.59 (0.01)	0.79	0.78
197_cpu_act	15	21	0.09	0.80	0.62	0.82	0.80
227_cpu_small	11	12	0.36 (0.03)	0.76	0.59 (0.01)	0.79	0.78
564_fried	6	10	0.50	0.67	0.36	0.75	0.77
201_pol	25	26	0.07 (0.02)	0.38	0.55	0.94	0.91

Tabla 5.6: Mejores hiperparámetros encontrados para cada conjunto de datos. Solamente se reportan los parámetros de *dropout* si el modelo los utilizó.

Datos	I	μ	s	d	λ	ℓ	p	α	p_{drop}	r_{drop}
215_2dplanes	1	0.34	0.72	70	2.05	7	30	2.00		
344_mv	19	0.58	0.8	394	0.08	14	19	0.0		
562_cpu_small	4549	0.18	0.62	750	0.89	15	60	0.72		
197_cpu_act	962	0.16	0.8	541	0.61	18	19	1.00	0.18	0.03
227_cpu_small	1622	0.26	0.8	417	0.08	9	34	0.16	0.03	0.002
564_fried	510	0.50	0.30	57	0.39	1	14	1.08		
201_pol	734	0.27	0.88	786	0.9	1	39	1.00		

La validación adversarial fue efectiva en cada caso, pues en cada *fold* de cada conjunto de datos, un modelo XGBoost fue incapaz de discriminar si un dato era parte del entrenamiento o de la validación mejor que un clasificador aleatorio (con un AUC de 0.5 exactamente), por lo que las distribuciones son similares y no se detectan problemas de contaminación de datos.

5.3. Discusiones

Se analizan los 7 conjuntos de datos por separado según sus resultados, como se profundizará en las secciones siguientes. La tabla 5.4 contiene los resultados de los experimentos, y la tabla 5.5 replica los experimentos utilizando la normalización de modelos, donde todos tienen el mismo número de características que SABRegressor.

Como una primera observación general, notamos que, en 6 de los 7 conjuntos, SABRegressor está a solamente 0.02 de diferencia en puntaje S, considerando que además el error de medición es de 0.01 en varios casos. Efectivamente se logró un puntaje similar al esperado, utilizando menos características. Si bien EBM mejoró su desempeño por sobre SABRegressor al eliminar las características menos relevantes, la metodología dependía de la cantidad de características usadas por SABRegressor, y no es claro **cuantas** características considerar a priori para seleccionar sin el algoritmo propuesto.

En 4 de los 7 conjuntos de datos (227_cpu_small, 562_cpu_small, 564_fried y 197_cpu_act), SABRegressor supera a los modelos simples, como regresión lineal Ridge y árboles de decisión simple, aunque su desempeño está por debajo de los modelos de complejidad completa por su forma aditiva restringida, y está un poco debajo del desempeño de EBM. Sin embargo, EBM está forzado a utilizar todas las características, mientras que SAB utiliza respectivamente 91.6 %, 91.6 %, 60 %, 71.4 % y 69.4 % de las características totales para obtener ese puntaje. Los mejores casos para SABRegressor con los del conjunto de datos de actividad de CPU, detallados en el anexo C.3, donde obtiene una pérdida de puntaje pequeña (de 0.05) con respecto a los modelos al modelo de mejor desempeño, XGBoost, con el beneficio de ser totalmente transparente.

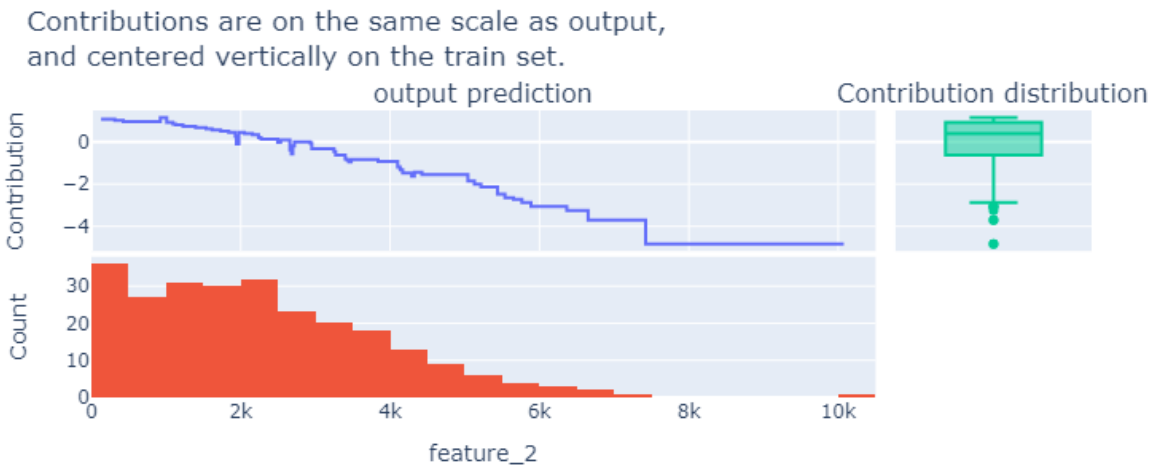
En el conjunto 564_fried, la pérdida de desempeño es mayor (de 0.11), por lo que el modelo puede ser insuficiente para una predicción precisa, tal como lo sería EBM en esta situación, pues ningún modelo aditivo se acercó al puntaje deseado. Sin embargo, todavía se puede utilizar SABRegressor para explorar los datos de forma más fiel y completa que una sobresimplificación de los datos ofrecida por modelos como regresión lineal o de árboles simples.

En el resto de los conjuntos de datos (344_mv, 201_pol, y 215_2dplanes), los modelos aditivos también tuvieron gran dificultad ajustándose a los datos, obteniendo un desempeño considerablemente peor al de un simple árbol de decisión, que es capaz de encontrar interacciones entre 2 o más variables. En ninguno de estos casos es recomendable utilizar un modelo aditivo para las predicciones si es que la precisión se valora por sobre la interpretabilidad, aunque sí puede ser una fuerte herramienta para tener la exploración de datos y *debugging*.

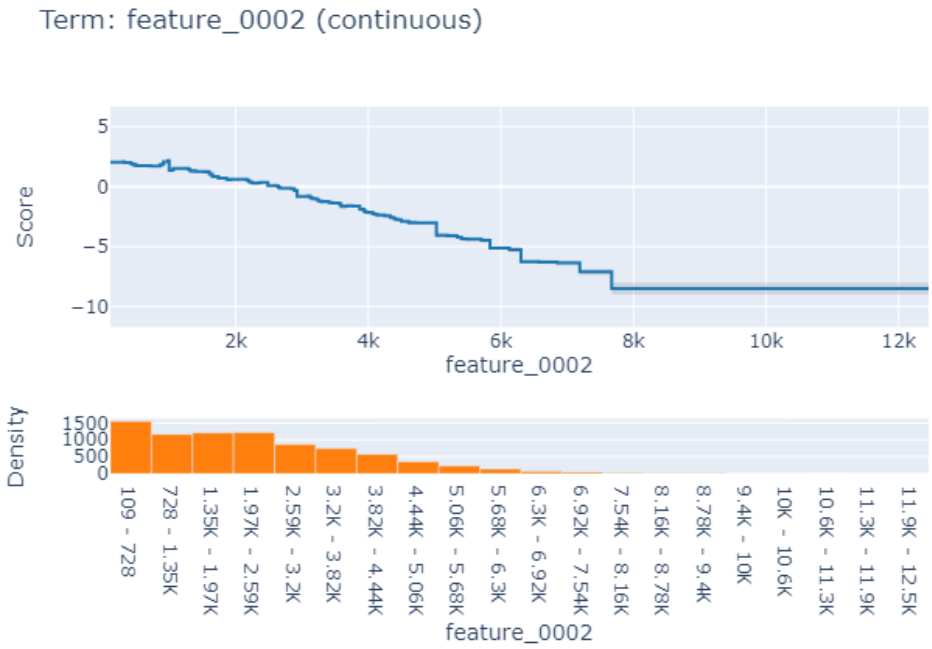
Originalmente, estos 3 conjuntos de datos tuvieron un desempeño mucho peor que el modelo lineal, llegando a ser negativo o muy pequeño, porque el modelo final se degeneraba a uno univariado, reduciendo enormemente su desempeño.

El caso de 344_mv tenía un puntaje de 0.11, siempre degenerándose a un modelo univariado (es decir, aprendiendo una única función) salvo que se desactivara primero el efecto de la redundancia en el algoritmo (es decir, fijando $\alpha = 0$), que permitía tener un puntaje similar al de los otros modelos aditivos. Es esperable que el modelo final no sea aditivo, pues es un conjunto de datos sintéticos con una regla de generación con interacciones complejas C.2.

En el conjunto 201_pol existían variables constantes, lo que indefinía la matriz de correlación, y así la selección de variables, por lo que se tenía un puntaje peor que el de un regresor constante, de -0.24. Eliminar las variables constantes hizo que SABRegressor logre superar a EBM en este conjunto.
Por último, para 215_2dplanes no se logró ningún tipo de arreglo, donde el puntaje fue el de un predictor constante, y la función aprendida fue esencialmente la nula, y este fue el único caso donde el modelo no se acercó a EBM. Inspeccionando el modelo, se nota que en cada iteración de potenciamiento, el desempeño empeoraba en vez de mejorar, por lo que al truncarse al modelo con mejor desempeño en validación, quedó uno constante. Sin embargo, notamos los datos son sintéticos, dados por una función lineal por partes especificada en el anexo C.1, con 3 variables irrelevantes y ninguna redundante. Sin embargo, el modelo no es capaz de aproximar a la función que genera los datos, debido a su estructura "if-else" con interacciones, que no se puede reproducir de forma aditiva. Si bien EBM logra una mejor solución, no se acerca a los modelos de (ensamble) de árboles, que son capaces de aprender la interacción necesaria.
Además, todas las variables son categóricas binarias o ternarias, por lo que un modelo aditivo de 3 hojas es equivalente a un modelo lineal utilizando codificación *one-hot* de las variables, solamente que asume que existe una estructura ordinal entre las variables cuando no la hay. Efectivamente, eliminar la estructura ordinal inexistente mejora mucho el desempeño del modelo, pues un modelo lineal con penalización *elastic net* [63] y codificación *one-hot* logra competir con los modelos de árboles, obteniendo un puntaje de 0.7 y un error estándar despreciable, superando por creces a todos los modelos excepto XGBoost - que dominó en este conjunto de datos -, pero con total interpretabilidad.
Es interesante ver que el poderoso modelo Random Forest también tuvo un pésimo desempeño en este conjunto de datos, peor que los otros modelos aditivos, a pesar de ser un modelo fuerte en el resto de los conjuntos de datos. Cuando el resto de los modelos se restringe a utilizar la característica que consideran más relevante, su puntaje es de 0.26, considerablemente peor, pero curiosamente, igual para un modelo Ridge que tiene un único parámetro.
Aprovechando la interpretabilidad intrínseca, se analiza si SABRegressor aprende un modelo con la misma interpretación que EBM, visualizando las funciones f_i en el caso de 197_cpu_act para ambos regresores (figuras 5.1, 5.3, 5.2). Las figuras se generan automáticamente con los métodos de gráficos de cada librería, por lo que su texto explicativo está en inglés.
Se verifica que las funciones tienen formas similares (en tipo de monotonía y discontinuidades mayores), pero que EBM tiene valores de contribuciones más bajos porque toma por defecto valores parámetros de regularización muy altos, y porque utiliza todas las características, así que debe dividir las contribuciones entre todas ellas, en especial si dos características son redundantes, mientras que SABRegressor debe contener las contribuciones correctas en menos características. Esta intuición no solamente se verifica a nivel de funciones, sino que al promediar las contribuciones de forma global, como muestra la figura 5.4.

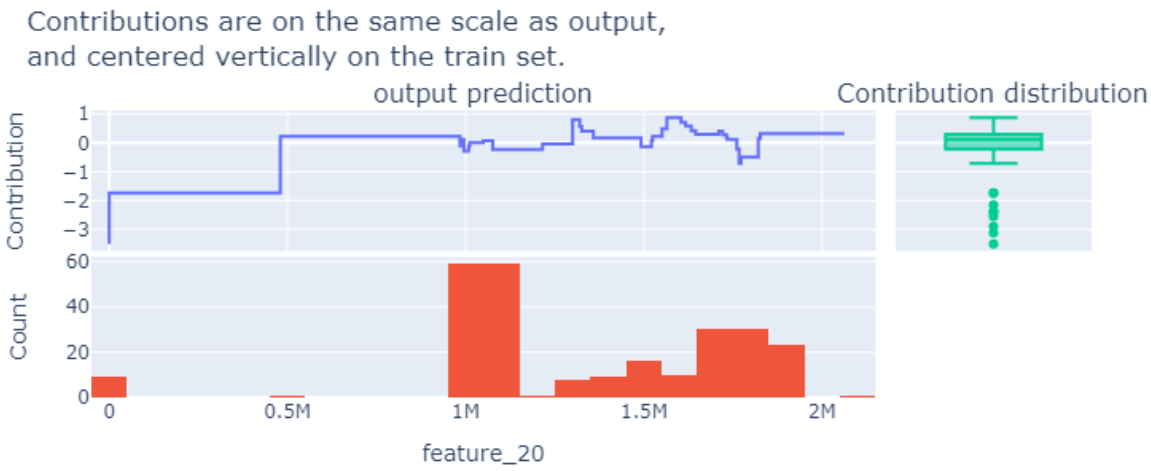


(a) SparseAdditiveBoostingRegressor

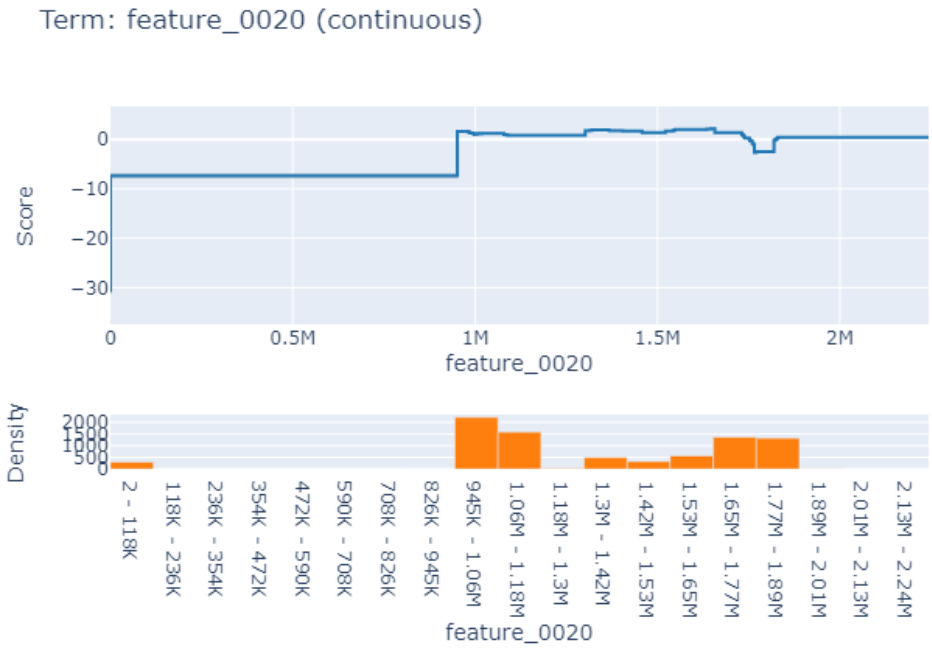


(b) ExplainableBoostingRegressor

Figura 5.1: Gráfico de la segunda característica aprendida por los modelos aditivos. En ambos casos, se ve que la contribución es decreciente en función de la característica, volviéndose constante al llegar a un valor de $x = 8000$.

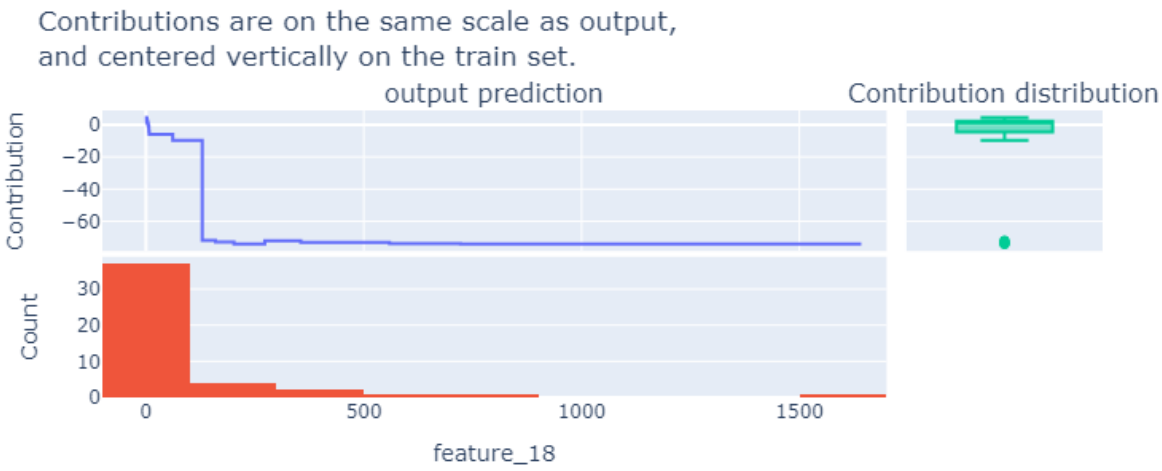


(a) SparseAdditiveBoostingRegressor

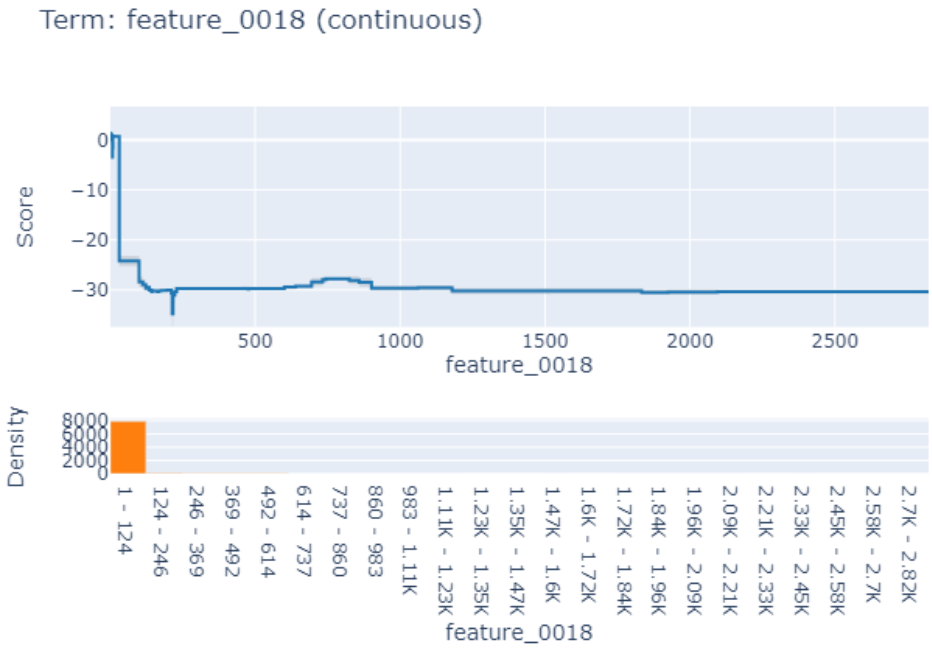


(b) ExplainableBoostingRegressor

Figura 5.2: Gráfico de la característica número 20 aprendida por los modelos aditivos. Es la única característica que no es estrictamente decreciente, y eso se respeta en ambos casos.

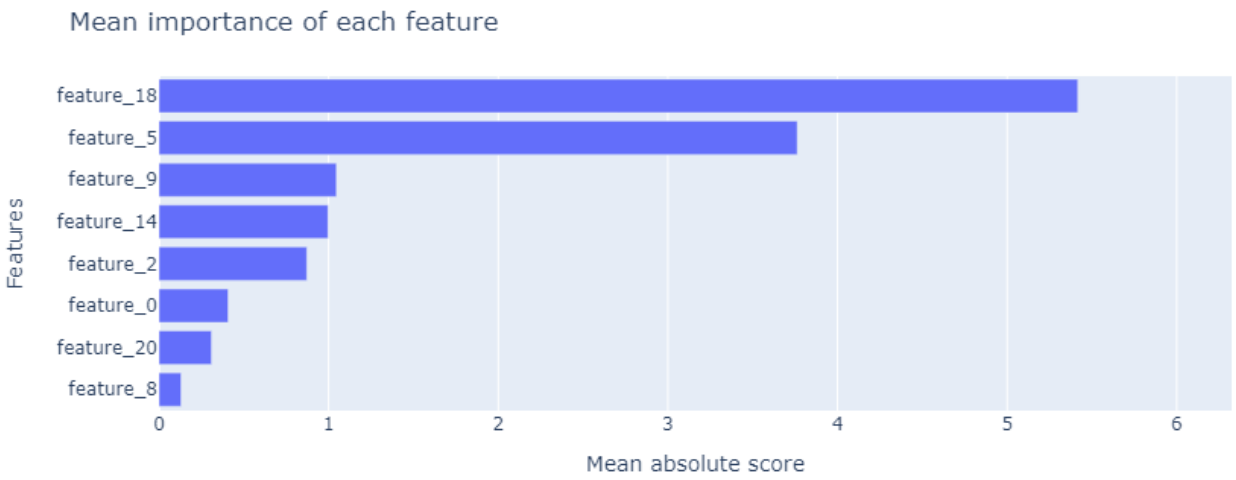


(a) SparseAdditiveBoostingRegressor

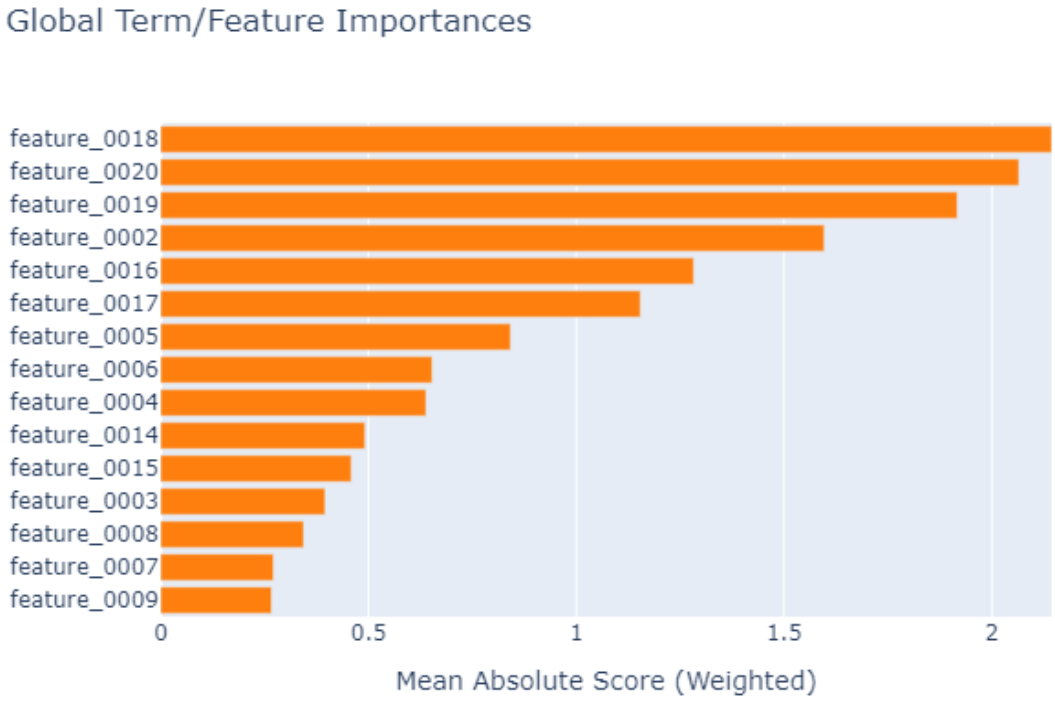


(b) ExplainableBoostingRegressor

Figura 5.3: Gráfico de la característica número 18 aprendida por los modelos aditivos. En ambos casos, hay una caída importante entre los primeros valores de la característica, y luego la contribución es casi constante.



(a) SparseAdditiveBoostingRegressor



(b) ExplainableBoostingRegressor

Figura 5.4: Importancia de características de cada modelo. El modelo Sparse solamente considera 8 de las 20 características, y divide las importancias entre ellas de manera menos uniforme, poniendo la mayoría de la responsabilidad en 2 de ellas.

5.4. Conclusión

En total, vemos los resultados esperados, donde SAB logra un puntaje similar a EBM en casi todos los conjuntos de datos - salvo en uno patológico - utilizando una fracción de las características. Truncar los otros modelos para que utilicen la misma cantidad de características que SAB logró una mejora en desempeño, pues les permitía reducir el ruido del conjunto de datos.

El desempeño se midió utilizando validación cruzada 5-*fold* en cada conjunto de datos, asegurando primero que calidad de los *fold* verificando que las distribuciones de entrenamiento y validación sean indistinguibles por un clasificador XGBoost.

Al analizar los resultados de los 7 conjuntos de datos, se observa que en datos como 227_cpu_small, 562_cpu_small, 564_fried, 197_cpu_act se obtienen los resultados esperados, donde SABRegressor supera el desempeño de modelos simples, aunque queda por debajo de modelos de complejidad completa y de EBM.

En particular, en el conjunto de datos de actividad de CPU, SABRegressor muestra una pérdida mínima de puntaje en comparación con el modelo de mejor rendimiento, XGBoost, lo que resalta su utilidad en situaciones donde la transparencia es crucial. Contrastando, en 564_fried, la pérdida de rendimiento es más significativa, se sugiere que los modelos aditivos podrían no ser suficiente para lograr predicciones precisas. Aun así, se destaca su capacidad para facilitar la exploración de datos de manera más completa que modelos más simples.

Sin embargo, en otros 3 casos, el modelo propuesto no alcanza un desempeño deseable, tarea que tampoco se logró por EBM, indicando que los modelos aditivos no son apropiados para modelar estos conjuntos de datos. Se señala que aún si se elige utilizar un modelo no interpretable, los modelos aditivos pueden ser valiosos para la exploración de datos y el *debugging*.

Otro problema identificado fue la selección demasiado conservadora, donde el modelo elige solo una variable, lo cual requiere desactivar la penalización por redundancia, que se puede con SABRegressor tomando el parámetro $\alpha = 0$.

Entre los casos de bajo desempeño, se destaca 201_pol, donde todos los modelos aditivos tienen un muy bajo puntaje en comparación a otros modelos de árboles, posiblemente debido a su incapacidad de capturar interacciones. Además, se señala la importancia de la limpieza de datos en este conjunto, donde la presencia de columnas constantes afecta la selección de características muy negativamente.

Además, se nota similitud en la interpretación entre las formas funciones aprendidas por EBM y SABRegressor, a pesar de que los valores específicos que evalúa en cada función son distintos, debido a que EBM debe dividir cada contribución en un mayor número de características.

La validación ayudó a entender el desempeño del SABRegressor en diferentes conjuntos de datos, destacando sus fortalezas y limitaciones, y señalando áreas de mejora en la propuesta del modelo.

Capítulo 6

Estudio experimental: Selección de características

El segundo experimento consiste en evaluar la capacidad del modelo de eliminar características innecesarias, sea por su irrelevancia o por su redundancia, comparando la cantidad de características seleccionadas por el modelo al ir sucesivamente agregando más información inútil a los datos de entrenamiento en forma de características sintéticamente creadas.

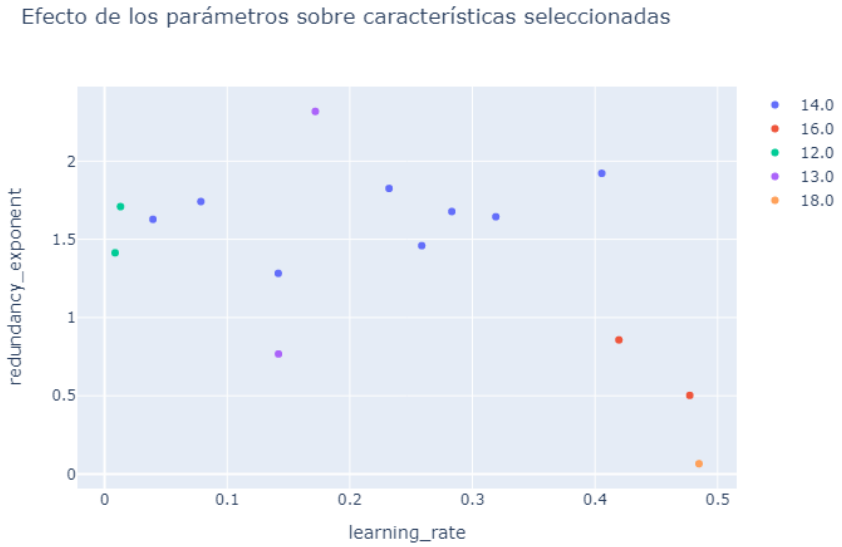
6.1. Metodología

Para este experimento, se decide utilizar el conjunto de datos 197_cpu_act, pues es el modelo logró un buen desempeño con respecto al mejor de los modelos encontrados, utilizando una fracción de las características. Se usan los mismos hiperparámetros encontrados en la sección anterior, salvo aquellos que modifican la cantidad de características seleccionadas.
Tal como en los experimentos de mRMR [36], se agregaron características redundantes o irrelevantes al conjunto de datos:

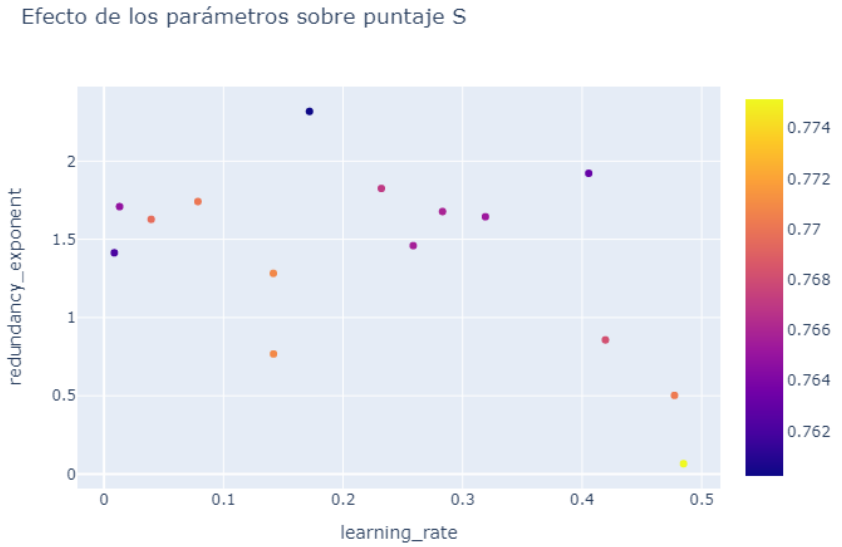
- Redundantes: i repeticiones de otras columnas con ruido agregado generado según la distribución $\mathcal{N}(0, \varepsilon)$, para $i \in [1, 20]$
- Irrelevantes: i columnas generadas aleatoriamente según la distribución $\mathcal{N}(0, \varepsilon)$, para $i \in [1, 20]$.
- Ambas: i características redundantes e i características irrelevantes generadas como se describió en los puntos anteriores, para $i \in [1, 10]$

Eligiendo ε como un décimo de la mínima varianza en todas las columnas.
Después de entrenar SABRegressor en los datos modificados, se calculó el porcentaje de características utilizadas, y cuantas de ellas son las generadas.
Es necesario tener en cuenta que dos hiperparámetros controlan implícitamente la cantidad de características seleccionadas: tasas de aprendizaje más bajas hacen que una característica se elija múltiples veces seguidas, y el exponente de la redundancia más alto hace que se penalice más la repetición de características. Por lo tanto, también se explora la relación que tienen los hiperparámetros con el puntaje y la cantidad de características elegidas, realizando una búsqueda de hiperparámetros que maximice el puntaje, y guardando todos los modelos que fueron evaluados por el algoritmo de búsqueda.

6.2. Resultados



(a) Características seleccionadas según los parámetros.



(b) Puntajes según los parámetros.

Figura 6.1: Efecto de distintos hiperparámetros para SABRegressor en 197_cpu_act. Vemos que se seleccionan menos características con tasas de aprendizaje y exponente de redundancia más pequeños. Verificamos que existen puntos que logran un buen puntaje con 13 características, y que hay una mayor pérdida para 12 características, aunque no de más de 0.1 de puntaje S.

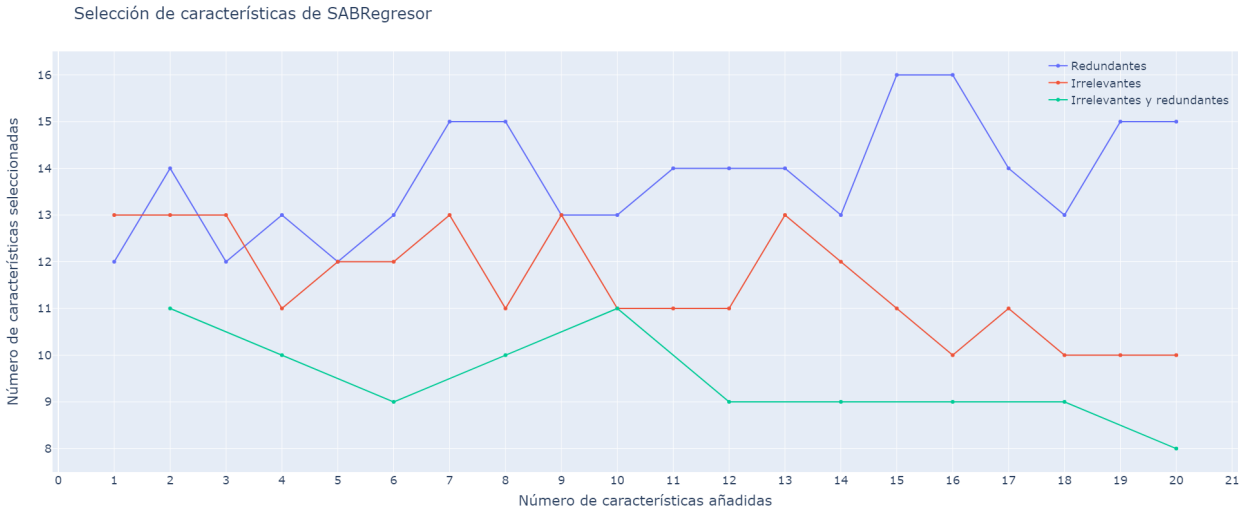


Figura 6.2: Cantidad de características seleccionadas para distintas perturbaciones de los datos.

6.3. Discusiones y conclusión

En el segundo experimento, se evaluó la capacidad de SABRegressor para eliminar características innecesarias, ya sea por su irrelevancia o redundancia. Se utilizó el conjunto de datos 197_cpu_act, seleccionado por su buen desempeño en el experimento anterior.
Para entender el efecto de los hiperparámetros μ y α, se grafica en el plano (μ, α) la cantidad de características seleccionadas y el puntaje obtenido por modelos optimizados con Optuna. Los resultados están en la figura 6.1.
Primero, se observa que existen agrupaciones para distintos números de características. Al aumentar μ y reducir ϕ, se utilizan más características. Al comparar, vemos que los puntajes más altos se obtuvieron por modelos con 13, 14 o 18 características. No hubo modelos con 15, 17, o más de 19 características, y el resto de los valores tuvieron un menor puntaje.
Luego, se repitió el experimento para los mejores hiperparámetros (con 14 características seleccionadas). Para los tres tipos de perturbaciones de datos, se observa su efecto en la figura 6.2, donde se nota lo siguiente:

- Al agregar características redundantes, en ciertas ocasiones se seleccionan más características, aunque este comportamiento no es estable. Si características que son más relevantes son elegidas para ser repetidas, entonces tienen mejor chance a ser seleccionadas que características que son irrelevantes y redundantes.

- Sorprendentemente, al agregar características irrelevantes, la cantidad de características seleccionadas parece disminuir. Cada característica se crea por ruido aleatorio, y en principio, ninguna debería ser elegida, y el algoritmo funcionaría igual. En el peor de los casos, la característica será espúreamente relevante, lo cual aumentaría las características usadas.

- Al agregar ambos tipos de características, se nota más aún la tendencia a disminuir la cantidad de selecciones. Efectivamente, los únicos modelos que usaron menos de 10 características fueron aquellos que tuvieron más de 12 características inyectadas.

Los resultados sugieren que SABRegressor tiende a ser más conservador al seleccionar características en presencia de características irrelevantes y redundantes, lo cual puede ser beneficioso para evitar la inclusión de información no relevante en el modelo final.
En conclusión, este experimento proporciona información valiosa sobre la capacidad de SABRegressor para manejar características innecesarias, destacando su sensibilidad a los hiperparámetros y su comportamiento ante perturbaciones en los datos. Estos hallazgos pueden ser fundamentales para afinar la configuración del modelo en situaciones donde la selección de características es crucial.

Capítulo 7

Estudio experimental: Detención más temprana

7.1. Introducción

Para evitar sobreajuste, SABRegressor utiliza *early stopping* basado en validación, es decir, que el entrenamiento se detiene si es que no presenta mejoras en un conjunto de validación por `n_iter_no_change` iteraciones (por defecto, 15).
Al examinar las curvas de validación en el experimento previo (Sección 5), se observa un punto de inflexión o "codo", indicando que después de un cierto número de iteraciones, los beneficios en la capacidad de generalización del modelo disminuyen. Este fenómeno sugiere que el modelo podría simplificarse mediante la reducción del número de iteraciones, lo que no solo aceleraría el proceso de entrenamiento, sino que también podría reducir la cantidad de características seleccionadas.
Este experimento se centra en evaluar la calidad de los modelos generados por SABRegressor al reducir la cantidad de iteraciones. El objetivo es mejorar la eficiencia del entrenamiento y determinar si hay un valor óptimo para el hiperparámetro `n_iter_no_change` que podría recomendarse a los usuarios de SABRegressor, especialmente en comparación con EBM, que por defecto utiliza un número considerablemente mayor de iteraciones por cada característica, y en la literatura se sugiere que requiere hasta millones de iteraciones [64].
A través de este experimento, se busca entender cómo la modificación del número de iteraciones afecta el rendimiento de SABRegressor en diversos conjuntos de datos, lo que proporciona información valiosa para la elección del hiperparámetro, que se pudo utilizar para el siguiente experimento (Sección 8), que aborda los hiperparámetros por defecto.

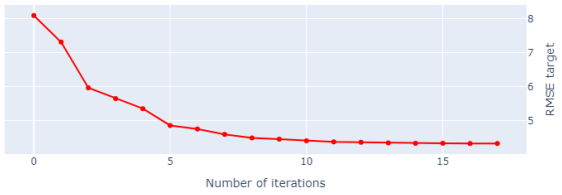
7.2. Metodología

Para los conjuntos de datos del experimento 5, se entrenó un modelo SABRegressor con 80 % de los datos, utilizando el resto como validación. Se elimina el conjunto 215_2*dplanes* del experimento, ya que su curva de validación es plana. Luego del entrenamiento, se utilizó el método `plot_model_information` para observar la curva de validación.
Visualmente, se identifica un "codo" de la figura, donde el puntaje sea suficientemente cercano al puntaje final utilizando la menor cantidad de iteraciones, bajo la consideración subjetiva del autor, similar al método del codo para K-medias.

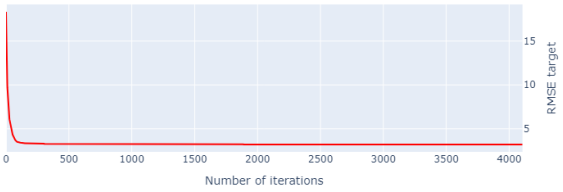
Una vez se haya identificado visualmente un número de iteraciones $\hat{I}$ menor al total I, se entrena el modelo con ambos números de iteraciones 5 veces. El mínimo de los tiempos medidos es una estimación más robusta del tiempo de ejecución sin factores externos como tiempo de carga del sistema, y así definimos los tiempos de entrenamiento t_I, $t_{\hat{I}}$. Además, se mide el puntaje de generalización de ambos, con la misma metodología del experimento 5, usando validación cruzada 5-*fold* para medir el puntaje S_I y $S_{\hat{I}}$.
Finalmente, se mide la mejora del tiempo de entrenamiento y decaimiento del puntaje S al reducir el número de iteraciones, en términos porcentuales, de la siguiente forma:

$$s(I, \hat{I}) = 100\frac{S_I - S_{\hat{I}}}{S_I}$$
$$t(I, \hat{I}) = 100\frac{t_I - t_{\hat{I}}}{t_I}$$

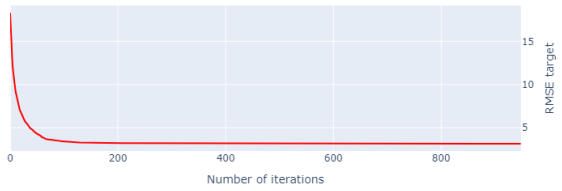
7.3. Resultados



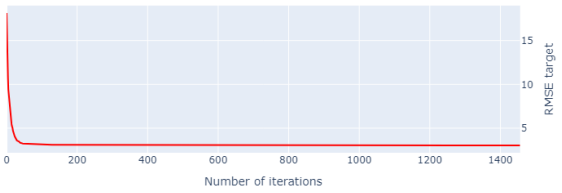
(a) Datos: 344_mv. Codo identificado: 12



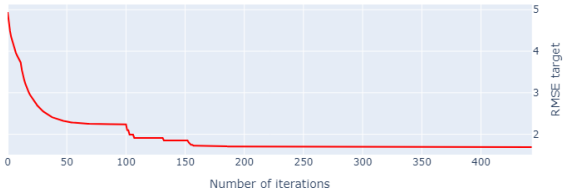
(b) Datos: 562_cpu_small. Codo identificado: 320



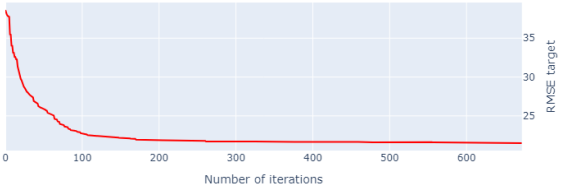
(c) Datos: 197_cpu_act. Codo identificado: 200



(d) Datos: 227_cpu_small. Codo identificado: 100



(e) Datos: 564_fried. Codo identificado: 190



(f) Datos: 201_pol. Codo identificado: 180

Figura 7.1: Curvas de puntaje de validación de cada iteración. Para cada conjunto de datos, se identifica un "codo" distinto, donde la raíz del error cuadrático medio ya no mejora substancialmente.

Tabla 7.1: Comparación de tiempo de entrenamiento y Raíz del Error Cuadrático Medio (RECM) en validación del modelo original (sección izquierda), en comparación a un modelo cortado para utilizar menos variables (sección derecha).

Datos	Recorte	Tiempo (s)	RECM	Tiempo (s)	RECM
344_mv	12	0.23	4.33	0.24	4.36
562_cpu_small	320	25.13	3.17	10.48	3.22
197_cpu_act	200	4.12	3.17	3.66	3.23
227_cpu_small	100	6.42	3.09	3.36	3.14
564_fried	190	2.62	1.69	2.62	1.7
201_pol	180	2.6	21.58	2.00	21.89

Tabla 7.2: Cambio porcentual del tiempo y el puntaje RECM al recortar un modelo utilizando un porcentaje de los estimadores totales. Se ve que utilizando una fracción menor de los estimadores que se aprendieron en el experimento anterior, no se pierde más de 2 % del puntaje.

Datos	Estimadores usados	Mejora tiempo	Decremento RECM
344_mv	63 %	0 %	0.5 %
562_cpu_small	7 %	58 %	1.4 %
197_cpu_act	21 %	11 %	1.9 %
227_cpu_small	6 %	48 %	1.6 %
564_fried	37 %	0 %	0.6 %
201_pol	25 %	23 %	1.4 %

7.4. Discusión y conclusiones

Como se identifica en la figura 7.1, efectivamente existen "codos" en la curvas de aprendizaje para cada uno de los conjuntos de datos. En los datos de CPU, es particularmente extremo, donde el algoritmo utilizó miles de iteraciones por defecto, cuando se pueden lograr resultados casi idénticos con 200-300 iteraciones.

Además de la identificación visual, nos interesa saber si existe un beneficio práctico de usar menos iteraciones en términos de tiempo de ejecución, y si lo vale un decaimiento de puntaje, lo cual se verifica en la tabla 7.2. Se observan mejoras de tiempo entre 0 y 58 %, que es sustancial, por pérdidas de puntaje que están entre 0.5 y 1.9 %, que, dependiendo del contexto de aplicación, puede ser una diferencia no significativa.

El conjunto de datos 344_mv es atípico, pues la cantidad de estimadores ya era muy pequeña (17), por lo que la mayor parte del tiempo de ejecución no es en el potenciamiento de gradiente, sino que, en otros pasos de preprocesamiento, como la discretización y calcular la matriz de correlación, por lo que no hubo mejoras del tiempo de entrenamiento. Lo mismo ocurre con 564_fried, a pesar de pasar de 510 a 190 iteraciones, el tiempo de las iteraciones todavía no dominaba el tiempo total del algoritmo.

Una vez que se superan las 800 iteraciones, se ven mejoras de tiempo mayores, pues el codo solamente requiere entre un octavo a un tercio de la cantidad. Lo ligero de la reducción deja claro que 320 iteraciones es un valor por defecto suficiente para considerar en todos los casos, pues logra un balance entre precisión y velocidad que uno desea para exploraciones, permitiendo al usuario aumentar la cantidad de iteraciones si es que necesita un modelo con mejor capacidad de generalización.

Capítulo 8

Estudio Experimental: Hiperparámetros por defecto

8.1. Introducción

Uno de los desafíos en el aprendizaje de máquinas es la selección de hiperparámetros. La API de scikit-learn pide que todos los hiperparámetros del modelo tengan un valor por defecto, es decir, que sea posible para el usuario utilizar el modelo sin tener que antes definirlos.
Para poder cumplir con este requisito, idealmente se deberían tener hiperparámetros que logren un desempeño "aceptable" para una variedad de conjuntos de datos, y en lo posible, que sean números fáciles de recordar para el usuario, y no valores flotantes con muchos dígitos.
En este experimento, se busca proponer un conjunto de valores por defecto y evaluar su desempeño con respecto al experimento 5, para determinar si existe una pérdida mayor por no haber ajustado los hiperparámetros a cada problema específico. El desempeño mostrado en esta sección es una representación más realista de las expectativas que debe tener un usuario previo a ajustar hiperparámetros.

8.2. Metodología

Para encontrar los valores por defecto a utilizar, se considera la tabla 5.3, que nos da los parámetros óptimos encontrados para cada conjunto de datos. Para cada hiperparámetro, se calcula la media como una estimación inicial. Luego, la estimación se aproxima por un número más fácil de recordar, sea a través de una aproximación decimal, o si no, aproximando a la potencia de 2 más cercana, dado lo comunes que son como opciones por su potencial ventaja computacional de operar.
Se excluye el conjunto 344_mv, debido a que este conjunto solamente funciona desactivando la redundancia al tomar $\alpha = 0$, sesgando los resultados. Se excluye también 215_2dplanes por haber sido un conjunto donde el método falló.
También hay dos hiperparámetros que no se estimaron con la metodología mencionada antes. Por un lado, ya se determinó en el experimento 7 que se puede detener el entrenamiento antes, usando un máximo de 320 estimadores para todos los conjuntos de datos, por lo que se toma este como el valor por defecto. Además, se considera que la elección $\alpha = 1$ es la más natural, que sigue la definición del criterio mRMR de la literatura [34].
Luego, una vez se tenga una propuesta de hiperparámetros, se valida con el mismo esquema del 5 para medir el desempeño del modelo con esos parámetros en los 5 conjuntos de datos

restantes, es decir, se mide el puntaje S con validación cruzada 5-*fold*. Para comparar a los resultados que se lograron anteriormente, también se mide en cuánto difiere el puntaje S en este experimento con respecto al experimento 5 con parámetros óptimos.

8.3. Resultados

Tabla 8.1: Parámetros por defecto.

Hiperparámetro	Estimación	Valor final
`n_estimators` (I)		320
`learning_rate` (μ)	0.274	0.3
`row_subsample` (s)	0.68	0.7
`max_bins` (d)	510.2	512
`l2_regularization` (λ)	0.574	0.6
`min_samples_leaf` (ℓ)	8.8	8
`max_leaves` (p)	33.2	32
`redundancy_exponent` (α)		1.0

Tabla 8.2: Modelo entrenado con nuevos parámetros "por defecto" uniformes para cada conjunto de datos.

Datos	Puntaje	Decremento puntaje S
562_cpu_small	0.74	2 %
197_cpu_act	0.76	2 %
227_cpu_small	0.74	2 %
564_fried	0.61	8 %
201_pol	0.38	5 %

8.4. Discusión y conclusiones

En la sección de resultados, se presenta la estimación y aproximación de los hiperparámetros por defecto obtenidos a partir de la metodología descrita. Los valores se detallan en la tabla 8.1, donde se muestra la estimación inicial y la aproximación final para cada hiperparámetro. Posteriormente, se evaluó el desempeño del modelo utilizando los nuevos hiperparámetros por defecto en los conjuntos de datos restantes. Los resultados se presentan en la Tabla 8.2, donde se muestra el puntaje obtenido en cada conjunto de datos, así como el decremento porcentual en comparación con el experimento anterior (5). Se evidencia que, si bien los hiperparámetros por defecto proporcionan un desempeño aceptable en términos generales, hay una ligera disminución en el puntaje en comparación con los parámetros óptimos ajustados para cada

conjunto de datos específico, que es comúnmente de 2 %, y en el peor de los casos sube hasta 8 %.
En conclusión, los hiperparámetros por defecto propuestos ofrecen una opción inicial razonable para los usuarios que deseen utilizar el modelo sin preocuparse por realizar ajustes personalizados. Sin embargo, se destaca la importancia de ajustar los hiperparámetros según las características específicas de cada conjunto de datos para lograr un rendimiento óptimo. Este experimento proporciona una visión más realista de las expectativas de desempeño antes de cualquier ajuste, brindando una referencia valiosa para los usuarios que buscan un punto de partida conveniente en sus aplicaciones de aprendizaje de máquinas.

Capítulo 9

Conclusiones

En el transcurso de la investigación, se abordó el desafío fundamental de desarrollar un algoritmo de aprendizaje de máquinas que logre un equilibrio entre alto rendimiento y transparencia, con enfoque en un nuevo tipo de modelos aditivos y su potencial en Explainable Artificial Intelligence (XAI), un campo en constante crecimiento que busca conciliar la complejidad de los modelos con la necesidad apremiante de comprender las decisiones automatizadas en entornos críticos.

El algoritmo propuesto en el presente trabajo logró exitosamente combinar la selección de características de máxima relevancia y mínima redundancia (mRMR) con potenciamiento de gradiente de árboles univariados, que se ajustan encontrando el óptimo global de problema de optimización regularizado, utilizando una estructura de datos especializada para inferencia veloz. El algoritmo se implementó como una librería en Python, y representa un paso hacia la mejora de la interpretabilidad de los modelos aditivos en entornos de alta dimensionalidad, contribuyendo directamente a la investigación de XAI.

Los experimentos realizados proporcionan evidencia de la eficacia de este enfoque. Se demostró que el nuevo algoritmo no solo compite en términos de error de generalización en comparación con modelos existentes, sino que también es capaz de mejorar su interpretabilidad seleccionando características relevantes, y mostrando resistencia ante la presencia de características redundantes.

Además, se exploró la optimización de hiperparámetros, identificando configuraciones predeterminadas que ofrecen un compromiso equilibrado entre desempeño y eficiencia, lo que facilita a los usuarios una implementación inicial sin necesidad de ajustes personalizados.

Esta tesis no solo presenta un algoritmo innovador para entrenar modelos aditivos con enfoque en la interpretabilidad y eficiencia, sino que también aporta conocimientos valiosos sobre la influencia de la reducción del número de iteraciones y la elección de hiperparámetros en el desempeño general del modelo.

Al terminar el proyecto, además de las contribuciones teóricas dentro de XAI, se ofrece una herramienta práctica y versátil para investigadores y científicos de datos que buscan resolver problemas de regresión donde la interpretabilidad es un elemento crítico. Confiamos en que este trabajo inspirará investigaciones adicionales y promoverá un mayor desarrollo en la búsqueda de soluciones que equilibren la utilidad de los modelos con la necesidad imperante de comprender las decisiones automatizadas en entornos de importancia y riesgo crítico.

Bibliografía

[1] Cooper, G. F., Abraham, V., Aliferis, C. F., Aronis, J. M., Buchanan, B. G., Caruana, R., Fine, M. J., Janosky, J. E., Livingston, G., Mitchell, T., y et al., "Predicting dire outcomes of patients with community acquired pneumonia," Journal of Biomedical Informatics, vol. 38, no. 5, p. 347–366, 2005, doi:10.1016/j.jbi.2005.02.005.

[2] Brennan, T. y Dieterich, W., "Correctional offender management profiles for alternative sanctions (compas)," Handbook of Recidivism Risk/Needs Assessment Tools, p. 49–75, 2017, doi:10.1002/9781119184256.ch3.

[3] Goodman, B. y Flaxman, S., "European union regulations on algorithmic decision-making and a "right to explanation"," AI Magazine, vol. 38, no. 3, p. 50–57, 2017, doi:10.1609/aimag.v38i3.2741.

[4] Shwartz-Ziv, R. y Armon, A., "Tabular data: Deep learning is not all you need," Information Fusion, vol. 81, pp. 84–90, 2022, doi:https://doi.org/10.1016/j.inffus.2021.11.011.

[5] Islam, S. R., Eberle, W., Ghafoor, S. K., y Ahmed, M., "Explainable artificial intelligence approaches: A survey," CoRR, vol. abs/2101.09429, 2021, https://arxiv.org/abs/2101.09429.

[6] Wang, Z. J., Kale, A., Nori, H., Stella, P., Nunnally, M. E., Chau, D. H., Vorvoreanu, M., Vaughan, J. W., y Caruana, R., "Interpretability, then what? editing machine learning models to reflect human knowledge and values," en Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, ACM, 2022, doi:10.1145/3534678.3539074.

[7] Nori, H., Jenkins, S., Koch, P., y Caruana, R., "Interpretml: A unified framework for machine learning interpretability." ArXiv, 2019, http://arxiv.org/abs/1909.09223.

[8] Chen, T. y Guestrin, C., "Xgboost: A scalable tree boosting system," en Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '16, (New York, NY, USA), p. 785–794, Association for Computing Machinery, 2016, doi:10.1145/2939672.2939785.

[9] Dorogush, A. V., Ershov, V., y Gulin, A., "Catboost: gradient boosting with categorical features support," 2018.

[10] Friedman, J. H., "Greedy function approximation: A gradient boosting machine.," The Annals of Statistics, vol. 29, no. 5, pp. 1189 – 1232, 2001, doi:10.1214/aos/1013203451.

[11] Kelley Pace, R. y Barry, R., "Sparse spatial autoregressions," Statistics & Probability Letters, vol. 33, no. 3, pp. 291–297, 1997, doi:https://doi.org/10.1016/S0167-7152(96)00140-X.

[12] Hyafil, L. y Rivest, R. L., "Constructing optimal binary decision trees is np-complete,"

Information Processing Letters, vol. 5, no. 1, pp. 15–17, 1976, doi:https://doi.org/10.1016/0020-0190(76)90095-8.

[13] Hu, X., Rudin, C., y Seltzer, M. I., "Optimal sparse decision trees," en Neural Information Processing Systems, 2019, https://api.semanticscholar.org/CorpusID:139104649.

[14] Hastie, T., Friedman, J., y Tibshirani, R., Additive Models, Trees, and Related Methods, pp. 257–298. New York, NY: Springer New York, 2001, doi:10.1007/978-0-387-21606-5_9.

[15] "User guide: Decision trees.", https://scikit-learn.org/stable/modules/tree.html\#complexity.

[16] Ribeiro, M. T., Singh, S., y Guestrin, C., ""why should I trust you?": Explaining the predictions of any classifier," en Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016, pp. 1135–1144, 2016.

[17] Izza, Y., Ignatiev, A., y Marques-Silva, J., "On explaining decision trees," 2020.

[18] Izza, Y., Ignatiev, A., y Marques-Silva, J., "On tackling explanation redundancy in decision trees," Journal of Artificial Intelligence Research, vol. 75, p. 261–321, 2022, doi:10.1613/jair.1.13575.

[19] Arenas, M., Barceló, P., Romero, M., y Subercaseaux, B., "On computing probabilistic explanations for decision trees," 2022.

[20] "Bagging vs boosting in machine learning.", https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/.

[21] Breiman, L., "Random forests," Machine Learning, vol. 45, pp. 5–32, 2001, https://api.semanticscholar.org/CorpusID:89141.

[22] Ke, G., Meng, Q., Finely, T., Wang, T., Chen, W., Ma, W., Ye, Q., y Liu, T.-Y., "Lightgbm: A highly efficient gradient boosting decision tree," en Advances in Neural Information Processing Systems 30 (NIP 2017), 2017, https://www.microsoft.com/en-us/research/publication/lightgbm-a-highly-efficient-gradient-boosting-decision-tree/.

[23] "State of data science and machine learning 2021," 2021, https://www.kaggle.com/kaggle-survey-2021.

[24] Grinsztajn, L., Oyallon, E., y Varoquaux, G., "Why do tree-based models still outperform deep learning on tabular data?," 2022.

[25] Friedman, J. H., "Stochastic gradient boosting," Computational Statistics & Data Analysis, vol. 38, no. 4, pp. 367–378, 2002, doi:https://doi.org/10.1016/S0167-9473(01)00065-2. Nonlinear Methods and Data Mining.

[26] "How training is performed: Bootstrap options.", https://catboost.ai/en/docs/concepts/algorithm-main-stages_bootstrap-options.

[27] "Introduction to boosted trees.", https://blog.bigml.com/2017/03/14/introduction-to-boosted-trees/.

[28] Shapley, L., 7. A Value for n-Person Games. Contributions to the Theory of Games II (1953) 307-317., pp. 69–79. Princeton: Princeton University Press, 1997, doi:doi:10.1515/9781400829156-012.

[29] Lundberg, S. M. y Lee, S.-I., "A unified approach to interpreting model predictions," en Advances in Neural Information Processing Systems 30 (Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., y Garnett, R., eds.), pp. 4765–4774, Curran Associates, Inc., 2017, http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[30] Barceló, P., Monet, M., Pérez, J., y Subercaseaux, B., "Model interpretability through the lens of computational complexity," 2020.

[31] Marques-Silva, J. y Ignatiev, A., "Delivering trustworthy ai through formal xai," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, pp. 12342–12350, 2022, doi:10.1609/aaai.v36i11.21499.

[32] Lou, Y., Caruana, R., y Gehrke, J., "Intelligible models for classification and regression," en Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, (New York, NY, USA), p. 150–158, Association for Computing Machinery, 2012, doi:10.1145/2339530.2339556.

[33] Guyon, I. y Elisseeff, A., "An introduction to variable and feature selection," Journal of Machine Learning Research, vol. 3, p. 1157–1182, 2003.

[34] Ding, C. y Peng, H., "Minimum redundancy feature selection from microarray gene expression data," en Computational Systems Bioinformatics. CSB2003. Proceedings of the 2003 IEEE Bioinformatics Conference. CSB2003, pp. 523–528, 2003, doi:10.1109/CSB.2003.1227396.

[35] Peng, H., Long, F., y Ding, C., "Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 27, no. 8, pp. 1226–1238, 2005, doi:10.1109/TPAMI.2005.159.

[36] Zhao, Z., Anand, R., y Wang, M., "Maximum relevance and minimum redundancy feature selection methods for a marketing machine learning platform," en 2019 IEEE International Conference on Data Science and Advanced Analytics (DSAA), pp. 442–452, 2019, doi:10.1109/DSAA.2019.00059.

[37] Nguyen, T. T., Soussen, C., Idier, J., y Djermoune, E.-H., "Np-hardness of $\ell 0$ minimization problems: revision and extension to the non-negative setting," 2019 13th International conference on Sampling Theory and Applications (SampTA), pp. 1–4, 2019, https://api.semanticscholar.org/CorpusID:197862528.

[38] Liu, J., Zhong, C., Seltzer, M., y Rudin, C., "Fast sparse classification for generalized linear and additive models," en Proceedings of The 25th International Conference on Artificial Intelligence and Statistics (Camps-Valls, G., Ruiz, F. J. R., y Valera, I., eds.), vol. 151 de Proceedings of Machine Learning Research, pp. 9304–9333, PMLR, 2022, https://proceedings.mlr.press/v151/liu22f.html.

[39] Kovács, L., "Feature selection algorithms in generalized additive models under concurvity," Computational Statistics, 2022, doi:10.1007/s00180-022-01292-7.

[40] Petersen, A., Witten, D., y Simon, N., "Fused lasso additive model," Journal of Computational and Graphical Statistics, vol. 25, no. 4, pp. 1005–1025, 2016, http://www.jstor.org/stable/44861906 (visitado el 2023-08-02).

[41] Chang, C.-H., Tan, S., Lengerich, B., Goldenberg, A., y Caruana, R., "How interpretable

and trustworthy are gams?," en Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, KDD '21, (New York, NY, USA), p. 95–105, Association for Computing Machinery, 2021, doi:10.1145/3447548.3467453.

[42] Greenwell, B. M., Dahlmann, A., y Dhoble, S., "Explainable boosting machines with sparsity – maintaining explainability in high-dimensional settings," 2023.

[43] Ibragimov, B. y Gusev, G., "Minimal variance sampling in stochastic gradient boosting," en Proceedings of the 33rd International Conference on Neural Information Processing Systems, (Red Hook, NY, USA), Curran Associates Inc., 2019.

[44] Rashmi, K. V. y Gilad-Bachrach, R., "DART: dropouts meet multiple additive regression trees," CoRR, vol. abs/1505.01866, 2015, http://arxiv.org/abs/1505.01866.

[45] Knuth, D. E., Searching an ordered table, vol. 3, p. 409–414. Addison-Wesley, 2 ed., 2021.

[46] Friedrich, F., Kempe, A., Liebscher, V., y Winkler, G., "Complexity penalised m-estimation: Fast computation," Journal of Computational and Graphical Statistics, 2005, https://api.semanticscholar.org/CorpusID:225060441.

[47] Chamandy, N., Muralidharan, O., Najmi, A., y Naidu, S., "Estimating uncertainty for massive data streams," rep. tec., Google, 2012.

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

[49] Van Rossum, G. y Drake Jr, F. L., Python reference manual. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[50] Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., y Varoquaux, G., "API design for machine learning software: experiences from the scikit-learn project," en ECML PKDD Workshop: Languages for Data Mining and Machine Learning, pp. 108–122, 2013.

[51] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., y Oliphant, T. E., "Array programming with NumPy," Nature, vol. 585, pp. 357–362, 2020, doi:10.1038/s41586-020-2649-2.

[52] Kernighan, B. W. y Ritchie, D. M., The C programming language. Prentice Hall, 2016.

[53] Inc., P. T., "Collaborative data science," 2015, https://plot.ly.

[54] pandas development team, T., "pandas-dev/pandas: Pandas," 2020, doi:10.5281/zenodo.3509134.

[55] Wes McKinney, "Data Structures for Statistical Computing in Python," en Proceedings of the 9th Python in Science Conference (Stéfan van der Walt y Jarrod Millman, eds.), pp. 56 – 61, 2010, doi:10.25080/Majora-92bf1922-00a.

[56] Molnar, C. A., "Shap (shapley additive explanations)," 2022, https://christophm.githu

b.io/interpretable-ml-book/shap.html.

[57] Cock, D. D., "Ames, iowa: Alternative to the boston housing data as an end of semester regression project," Journal of Statistics Education, vol. 19, no. 3, 2011, doi:10.1080/10691898.2011.11889627.

[58] Bergstra, J., Bardenet, R., Bengio, Y., y Kégl, B., "Algorithms for hyper-parameter optimization," en Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11, (Red Hook, NY, USA), p. 2546–2554, Curran Associates Inc., 2011.

[59] Akiba, T., Sano, S., Yanase, T., Ohta, T., y Koyama, M., "Optuna: A next-generation hyperparameter optimization framework," en Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019.

[60] Zaharia, M. A., Chen, A., Davidson, A., Ghodsi, A., Hong, S. A., Konwinski, A., Murching, S., Nykodym, T., Ogilvie, P., Parkhe, M., Xie, F., y Zumar, C., "Accelerating the machine learning lifecycle with mlflow," IEEE Data Eng. Bull., vol. 41, pp. 39–45, 2018, https://api.semanticscholar.org/CorpusID:83459546.

[61] Qian, H., Wang, B., Ma, P., Peng, L., Gao, S., y Song, Y., "Managing dataset shift by adversarial validation for credit scoring," CoRR, vol. abs/2112.10078, 2021, https://arxiv.org/abs/2112.10078.

[62] Romano, J. D., Le, T. T., La Cava, W., Gregg, J. T., Goldberg, D. J., Chakraborty, P., Ray, N. L., Himmelstein, D., Fu, W., y Moore, J. H., "PMLB v1.0: an open-source dataset collection for benchmarking machine learning methods," Bioinformatics, vol. 38, pp. 878–880, 2021, doi:10.1093/bioinformatics/btab727.

[63] Zou, H. y Hastie, T., "Regularization and Variable Selection Via the Elastic Net," Journal of the Royal Statistical Society Series B: Statistical Methodology, vol. 67, pp. 301–320, 2005, doi:10.1111/j.1467-9868.2005.00503.x.

[64] Agarwal, R., Melnick, L., Frosst, N., Zhang, X., Lengerich, B., Caruana, R., y Hinton, G., "Neural additive models: Interpretable machine learning with neural nets," 2021.

[65] Molnar, C. A., "Shapley values," 2022, https://christophm.github.io/interpretable-ml-book/shapley.html.

[66] Storath, M. y Weinmann, A., "Pottslab," 2019, https://github.com/mstorath/Pottslab.

[67] Breiman, L., Friedman, J. H., Olshen, R. A., y Stone, C. J., "Classification and regression trees," Biometrics, vol. 40, p. 874, 1984, https://api.semanticscholar.org/CorpusID:29458883.

[68] Torgo, L., "Regression data sets.", https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html.

[69] Torgo, L. y Revow, M., "Comp-activ database," 1996, https://www.cs.toronto.edu/~delve/data/comp-activ/compActivDetail.html.

[70] Friedman, J. H., "Multivariate Adaptive Regression Splines," The Annals of Statistics, vol. 19, no. 1, pp. 1 – 67, 1991, doi:10.1214/aos/1176347963.

[71] Weiss, S. M. y Indurkhya, N., "Rule-based machine learning methods for functional prediction," Journal of Artificial Intelligence Research, vol. 3, pp. 383–403, 1995.

Anexos

Anexo A. Valores de Shapley

Introducción

Un juego cooperativo (v, N) es un conjunto finito de jugadores N y una función de valoración $v : \mathcal{P}(N) \to \mathbb{R}$ tal que $v(\emptyset) = 0$.
En particular, $S \subseteq N$ representa a una coalición de jugadores, que, al cooperar, ganan una recompensa colectiva de $v(S)$. El valor de Shapley $\phi_i(v) \in \mathbb{R}$ de un juego coalicional mide la recompensa justa que merece un jugador $i \in N$ si es que todos los jugadores cooperan y deben dividir la recompensa $v(N)$ entre ellos. En particular, es la única función que cumple [65]:

1. Eficiencia: $\sum_{i \in N} \phi_i(v) = v(N)$
2. Simetría: $\forall S \subseteq N \setminus \{i, j\}, v(S \cup \{i\}) = v(S \cup \{j\}) \implies \phi_i(v) = \phi_j(v)$
3. Linealidad: $\phi_i(\alpha v - w) = \alpha \phi_i(v) - \phi_i(w)$
4. Jugador nulo: $\forall S \subseteq N \setminus \{i\}, v(S \cup \{i\}) = v(S) \implies \phi_i(v) = 0$

Y está dada por:

$$\phi_i(v) = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} (v(S \cup \{i\}) - v(S))$$

Uso en aprendizaje de máquinas

Para un modelo f y una instancia $x \in \mathbb{R}^N$, definimos un juego cooperativo con cada característica (coordenada) de x como jugador, con función de valoración $v(S) = \mathbb{E}[f(x)|x_S]$, es decir, el valor del modelo si solamente se ven o entrena con las características S.
Asumiendo independencia de las características, como lo hace la librería SHAP, se tiene que en un modelo aditivo $v(S) = \sum_{i \in S} f_i(x_i)$, que verifica $v(\varnothing) = 0$.

Por lo tanto, el valor de SHAP de la característica i es:

$$\begin{aligned}
\phi_i(v) &= \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!}(v(S \cup \{i\}) - v(S)) \\
&= \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!}(\sum_{j \in S \cup \{i\}} f_j(x_j) - \sum_{j \in S} f_j(x_j)) \\
&= \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} f_i(x_i) \\
&= f_i(x_i) \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!}
\end{aligned}$$

Donde además se tiene que:

$$\begin{aligned}
\sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} &= \frac{1}{|N|!} \sum_{S \subseteq N \setminus \{i\}} |S|!(|N| - |S| - 1)! \\
&= \frac{1}{|N|!} \sum_{j \leq |N|} \sum_{\substack{S \subseteq N \setminus \{i\} \\ |S| = j}} |S|!(|N| - |S| - 1)! \\
&= \frac{1}{|N|!} \sum_{j \leq |N|} \sum_{\substack{S \subseteq N \setminus \{i\} \\ |S| = j}} j!(|N| - j - 1)! \\
&= \frac{1}{|N|!} \sum_{j \leq |N|} \binom{|N| - 1}{j} j!(|N| - j - 1)! \\
&= \frac{1}{|N|!} \sum_{j \leq |N|} (|N| - 1)! \\
&= \frac{|N|(|N| - 1)!}{|N|!} \\
&= 1
\end{aligned}$$

De forma que $\phi_i(v) = f_i(x_i)$, y las explicaciones del modelo aditivo coinciden con las de SHAP.

Anexo B. Soluciones al problema de Potts

Una simplificación de la implementación en C del algoritmo propuesto en el estudio [46] está en el pseudocódigo 6, adaptada de Pottslab, una implementación en Java [66]. En contraste, al agregar pesos aleatorios y regularización L^2, se tienen que agregar las modificaciones del algoritmo que usa `asboostreg`, con el pseudocódigo 7, mostrando los cambios en azul.

Algoritmo 6: Algoritmo original para solucionar el problema de Potts

Data: $x \in \mathbb{R}^n$ característica, $y \in \mathbb{R}^n$ objetivo, $\gamma > 0$ regularización de saltos, $p \in \mathbb{N}$ máximo de valores en cada hoja

Result: Árbol (D, H)

Sean S, Q arreglos de $n+1$ componentes, inicializados con ceros; /* Precálculos */

for $i \in [1 \dots n]$ **do**

  $S[i+1] \leftarrow S[i] + y[i]$; /* Suma acumulada de y */

  $Q[i+1] \leftarrow Q[i] + y[i]^2$; /* Suma acumulada de y^2 */

end

/* Programación dinámica */

Sea J arreglo de n componentes; /* Discontinuidades de la función */

Sea B arreglo de n componentes; /* Valor de la función de Bellman del P.D. */

for $r \in [1 \dots n]$ **do**

  $B[r-1] \leftarrow Q[r] - \dfrac{S[r]^2}{r}$;

  $J[r-1] \leftarrow 0$;

  for $l \in [r-p-1 \dots 1]$ **do**

    $d \leftarrow \gamma + Q[r] - Q[l] - \frac{(S[r]-S[l])^2}{r-l}$;

    if $d > B[r-1]$ **then**

      Romper el ciclo;

    end

    $c \leftarrow B[l-1] + d$; /* Pérdida de un candidato a discontinuidad */

    if $c > B[r-1]$ **then**

      /* Mejora encontrada */

      $B[r-1] = c$;

      $J[r-1] = l$;

    end

  end

end

Sean D, H listas vacías; /* Construir el árbol */

$r \leftarrow n, l \leftarrow J[n-1]$;

while $r > 0$ **do**

  $\mu \leftarrow \frac{S[r]-S[l]}{r-l}$; /* Valor medio de y en $[l, r]$ */

  Agregar μ al principio de H;

  Agregar $x[l]$ al principio de D, salvo en la primera iteración del ciclo;

  $r \leftarrow l$;

  if $r < 1$ **then**

    Romper el ciclo;

  end

  $l \leftarrow J[r-1]$;

end

Algoritmo 7: Algoritmo para solucionar el problema de Potts regularizado y con pesos

Data: $x \in \mathbb{R}^n$ característica, $y \in \mathbb{R}^n$ objetivo, $w \in \mathbb{R}^n$ pesos aleatorios, $\gamma > 0$ regularización de saltos, $\lambda > 0$ regularización de hojas, $p \in \mathbb{N}$ máximo de valores en cada hoja

Result: Árbol (D, H)

Sean S, Q, W arreglos de $n+1$ componentes, inicializados con ceros; /* Precálculos */

for $i \in [1 \dots n]$ **do**

  $S[i+1] \leftarrow S[i] + w[i]y[i]$; /* Primer momento acumulado de y */

  $Q[i+1] \leftarrow Q[i] + w[i]y[i]^2$; /* Segundo momento acumulado de y */

  $W[i+1] \leftarrow W[i] + w[i]$; /* Suma acumulada de los pesos w */

end

/* Programación dinámica */

Sea J arreglo de n componentes; /* Discontinuidades de la función */

Sea B arreglo de n componentes; /* Valor de la función de Bellman del P.D. */

for $r \in [1 \dots n]$ **do**

  $B[r-1] \leftarrow Q[r] - \dfrac{S[r]^2}{W[r] + \lambda r}$;

  $J[r-1] \leftarrow 0$;

  for $l \in [r-p-1 \dots 1]$ **do**

    $d \leftarrow \gamma + Q[r] - Q[l] - \frac{(S[r]-S[l])^2}{W[r]-W[l]+\lambda(r-l)}$;

    if $d > B[r-1]$ **then**

      Romper el ciclo;

    end

    $c \leftarrow B[l-1] + d$; /* Pérdida de un candidato a discontinuidad */

    if $c > B[r-1]$ **then**

      /* Mejora encontrada */

      $B[r-1] \leftarrow c$;

      $J[r-1] \leftarrow l$;

    end

  end

end

Sean D, H listas vacías; /* Construir el árbol */

$r \leftarrow n, l \leftarrow J[n-1]$;

while $r > 0$ **do**

  $\mu \leftarrow \frac{S[r]-S[l]}{W[r]-W[l]+\lambda(r-l)}$; /* Valor medio regularizado y ponderado de y en $[l, r]$ */

  Agregar μ al principio de H;

  Agregar $x[l]$ al principio de D, salvo en la primera iteración del ciclo;

  $r \leftarrow l$;

  if $r < 1$ **then**

    Romper el ciclo;

  end

  $l \leftarrow J[r-1]$;

end

Anexo C. Descripción de los datos

C.1. 2D Planes

40768 datos, 10 características, generado sintéticamente como tal [67]:

$$
\begin{aligned}
x_1 &\sim U\{-1,1\} \\
x_i &\sim U\{-1,0,1\} \ : \ i \in [2,10] \\
y &= \begin{cases} 3+3x_2+2x_3+x_4+\sigma(0,2) \text{ si } x_1 = 1 \\ -3+3x_5+2x_6+x_7+\sigma(0,2) \text{ si } x_1 = -1 \end{cases}
\end{aligned}
$$

Es decir, dos funciones lineales, dependiendo del valor de x_1. Las características x_8, x_9, x_{10} son irrelevantes. Cabe destacar que no es aditivo, y requiere modelar la interacción fuerte de x_1 con el resto de las variables.

C.2. MV

40768 datos, 10 características, generado sintéticamente como tal [68]:

$$
\begin{aligned}
x_1 &\sim U[-5,5] \\
x_2 &\sim U[-15,-10] \\
x_3 &= \begin{cases} \text{“green”} & \text{si } x_1 > 0 \\ \text{“red”} & \text{con probabilidad 0.4, “brown”con probabilidad 0.6} \end{cases} \\
x_4 &= \begin{cases} x_1 + 2x_2 & \text{si } x_3 = \text{“green”} \\ \dfrac{x_1}{2} & \text{con probabilidad 0.3, } \dfrac{x_2}{2} \text{con probabilidad 0.7} \end{cases} \\
x_5 &\sim U[-1,1] \\
x_6 &= x_4\epsilon, \text{ con } \epsilon \sim U[0,5] \\
x_7 &= \begin{cases} \text{“yes”} & \text{con probabilidad 0.3, “no” con probabilidad 0.7} \end{cases} \\
x_8 &= \begin{cases} \text{“normal”} & \text{si } x_5 < 0.5 \\ \text{“large”} & \text{en otro caso} \end{cases} \\
x_9 &\sim U[100,500] \\
x_{10} &\sim U[[1000,1200] \cap \mathbb{N}] \\
y &= \begin{cases} 35-0.5x_4 & \text{si } x_2 > 2 \\ 10-2x_1 & \text{si } -2 \leq x_4 \leq 2 \\ 3-\dfrac{x_1}{x_4} & \text{si } x_7 = \text{“yes”} \\ x_6+x_1 & \text{si } x_8 = \text{“normal”} \\ \dfrac{x_1}{2} & \text{en otro caso} \end{cases}
\end{aligned}
$$

Este modelo contiene interacciones complejas entre variables, que no se modelan aditivamente.

C.3. CPU

Cada conjunto tiene 8192 datos. 21 características para 197_cpu_act y 12 para los otros. Son conjuntos reales, que se tratan de la misma tarea: Predecir la porción de uso de un CPU en modo usuario, en base a distintas variables, como el número de llamadas a distintas funciones, cantidad de lecturas y escrituras, uso de memoria y paginación de memoria, etc. Las características específicas con las siguientes [69]:

1. **lread:** Lecturas (transferencias por segundo) entre la memoria del sistema y la memoria del usuario.
2. **lwrite:** Escrituras (transferencias por segundo) entre la memoria del sistema y la memoria del usuario.
3. **scall:** Número de llamadas del sistema de todos los tipos por segundo.
4. **sread:** Número de llamadas de lectura del sistema por segundo.
5. **swrite:** Número de llamadas de escritura del sistema por segundo.
6. **fork:** Número de llamadas de bifurcación del sistema por segundo.
7. **exec:** Número de llamadas de ejecución del sistema por segundo.
8. **rchar:** Número de caracteres transferidos por segundo por llamadas de lectura del sistema.
9. **wchar:** Número de caracteres transferidos por segundo por llamadas de escritura del sistema.
10. **pgout:** Número de solicitudes de expulsión de página por segundo.
11. **ppgout:** Número de páginas expulsadas por segundo.
12. **pgfree:** Número de páginas por segundo colocadas en la lista de libres.
13. **pgscan:** Número de páginas verificadas si pueden liberarse por segundo.
14. **atch:** Número de adhesiones de página (satisfaciendo una falta de página reclamando una página en memoria) por segundo.
15. **pgin:** Número de solicitudes de entrada de página por segundo.
16. **ppgin:** Número de páginas introducidas por segundo.
17. **pflt:** Número de fallas de página causadas por errores de protección (copias al escribir).
18. **vflt:** Número de fallas de página causadas por la traducción de direcciones.
19. **runqsz:** Tamaño de la cola de ejecución de procesos.
20. **freemem:** Número de páginas de memoria disponibles para los procesos de usuario.
21. **freeswap:** Número de bloques de disco disponibles para el intercambio de páginas.

22. **usr (objetivo):** Porcentaje de tiempo (%) que las CPU ejecutan en modo usuario.

Donde **usr** es la variable por predecir. Los conjuntos cpu_small son similares, pero sin la información de paginación, utilizando solamente las primeras 12 características. Cabe destacar determinó que los conjuntos cpu_small no son duplicados, aunque PMLB sí incluye algunos duplicados que fueron eliminados de este estudio.

C.4. Fried

40768 datos, 10 características, generado sintéticamente como tal [70]:

$$y = 10\sin(\pi x_1 x_2) + 20(x_3 - 0.5)^2 + 10x_4 + 5x_5 + \sigma(0, 1)$$

Con $x_i \sim U[0, 1]$ i.i.d. La única interacción que no es aditiva es entre x_1 y x_2.

C.5. Pol

15000 datos, 48 atributos continuos. Esto es una aplicación comercial descrita en [71]. Los datos describen un problema de telecomunicaciones. No hay información adicional disponible en los sitios de referencia [68].

Anexo D. Información de reproducibilidad

Todos los experimentos se realizaron en un computador personal, con las siguientes especificaciones:

Tabla D.1: Especificaciones

Componente	**Detalle**
Procesador (CPU)	AMD Ryzen 5 2600X
Memory (RAM)	16.0 GB
Sistema Operativo	Windows 10 64 bits
Versión de Python	3.11.3
Compilador de C	gcc 13.1.0

Utilizando las siguientes librerías de Python:

Código D.1: Listado de dependencias

```
# Librería
attrs>=23.1.0  # P.O.O. con menos boilerplate
category_encoders>=2.6.3  # CatBoostEncoder
numpy>=1.25.2  # Operaciones eficientes en arreglos
pandas>=2.1.3  # Manipulación de datos
plotly>=5.18.0  # Gráficos interactivos automáticos
scikit-learn>=1.3.2  # API de modelos
shap>=0.43.0  # Explicaciones

# Experimentos
jupyter>=1.0.0  # Notebooks de computación interactiva
```

```
interpret>=0.4.4  # ExplainableBoostingMachine
mlflow>=2.8.1  # Manejo y organización de experimentos
optuna>=3.4.0  # Búsqueda de hiperparámetros
pmlb>=1.0.1post3  # Conjuntos de datos de Benchmark
xgboost>=2.0.2  # XGBRegressor
```