Algoritmo de Gradient Boosting con modelos predictivos

Iván Miranda Balastegui
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
Correo UVUS: ivamirbal@alum.us.es

Daniel López Ramos
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
Correo UVUS: danlopram@alum.us.es

El objetivo principal de este trabajo es desarrollar un algoritmo de ensamblado secuencial de modelos predictivos de inteligencia artificial, con el fin de evaluar su rendimiento y contrastarlo con el desempeño obtenido al utilizar dichos modelos de manera individual mediante el uso de problemas de regresión.

A partir del desarrollo de esta investigación, se concluye que el uso del algoritmo de ensamblado secuencial aporta mejoras en comparación con los modelos individuales. No obstante, dichas mejoras no resultan especialmente significativas, lo que podría deberse a un número insuficiente de iteraciones empleadas durante la ejecución del algoritmo.

I. Introducción

La inteligencia artificial (IA) constituye una de las áreas de investigación más activas y con mayor impacto en la actualidad, abarcando desde el procesamiento del lenguaje natural, el diagnóstico médico o la predicción financiera. La inteligencia artificial puede ser definida como "La automatización de actividades que vinculamos con procesos de pensamiento humano, actividades como la toma de decisiones, resolución de problemas, aprendizaje…" [1].

Dentro de este amplio campo, el aprendizaje automático es una de las ramas más fundamentales, la cual estudia como dotar a las máquinas de capacidad de aprendizaje, basándose en algoritmos capaces de identificar patrones en grandes bases de datos y aprender de ellos [2]. Sin embargo, en contextos reales, un solo modelo raramente ofrece un rendimiento óptimo de forma consistente. Por esta razón, se han desarrollado técnicas de combinación de modelos que intentan ayudar a mejorar el rendimiento de los modelos de Machine Learning al mejorar su precisión. Este es un proceso mediante el cual se construyen estratégicamente varios modelos de Machine Learning para resolver un problema particular [3].

Entre los diferentes ensamblados de modelos encontramos el "**Boosting"** o ensamble secuencial.

En este contexto, el trabajo que se presenta se enmarca en el estudio y desarrollo de un algoritmo de ensamblado secuencial de modelos de regresión.

Se pretende evaluar en qué medida la combinación iterativa de estimadores simples permite mejorar el rendimiento predictivo, empleando métricas estándar como el coeficiente de determinación. Para ello, se elaborará un algoritmo que siga la dinámica del ensamble secuencial y se realizará una búsqueda de parámetros óptimos de los modelos usados. Por último, se experimentará con el algoritmo y los modelos individuales y se compararán resultados.

Este documento seguirá una estructura basada en secciones, dentro de las cuales distinguiremos como las principales:

- Preliminares: información necesaria para el entendimiento del ensamble secuencial.
- Metodología: desarrollo de la investigación y el trabajo.
- Resultados: datos obtenidos y su interpretación.
- Conclusiones
- Referencias

II. Preliminares

En esta sección trataremos las ideas clave y aspectos generales a tener en cuenta para el entendimiento del funcionamiento del ensamblado secuencial, así como trabajos relacionados que pueden llegar a ser de utilidad.

A. Métodos empleados

La idea principal del ensamble de modelos predictivos consiste en entrenar varios modelos sobre un mismo problema y luego combinarlos para obtener un modelo final denominado **meta-modelo**, con mayor capacidad predictiva que los modelos que lo componen. Cada uno de los modelos que componen el meta-modelo se les denomina **modelos débiles**, ya que individualmente no ofrecen en general una buena predicción al problema, pero cada modelo se especializa en una parte del problema dando en conjunto una solución más robusta. Existen diferentes técnicas para construir ensambles de modelos predictivos, pero este proyecto se centra en el ensamblado secuencial para problemas de regresión.

En la estrategia de ensamblado secuencial, cada modelo débil centra su aprendizaje en los errores cometidos por el modelo anterior. El algoritmo de **potenciación del gradiente (Gradient Boosting)** sigue esta metodología donde cada modelo corrige los errores residuales de los predecesores. El objetivo principal del método es el de minimizar una función de perdida agregando modelos débiles utilizando el algoritmo

de optimización de descenso de gradiente como podemos observar en la Fig. 1.

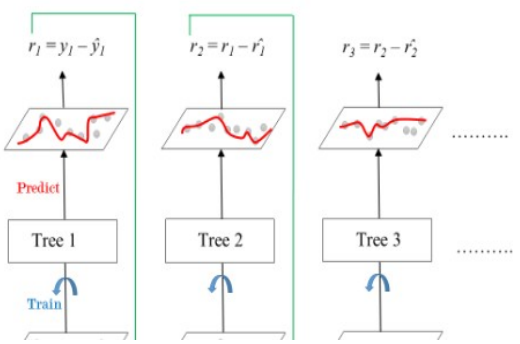

Fig. 1. Potenciación del gradiente mediante el residuo

La función de pérdida a minimizar es el error cuadrático calculado sobre cada ejemplo (1).

$$L\left(y_i, F(x_i)\right) = \frac{1}{2}\left(y_i - F(x_i)\right)^2 \qquad (1)$$

Donde F es la predicción del ensamble para un ejemplo x_i e y_i el valor de la variable respuesta para ese ejemplo.

Teniendo en cuenta que queremos reducir el error, cambiamos el signo. De esta forma, la nueva variable respuesta a predecir en cada iteración será el gradiente del error cometido sobre cada ejemplo en la iteración anterior denominado **residuo** (2).

$$r_{im} = -\left[\frac{\partial L\left(y_i, F_{m-1}(x_i)\right)}{\partial F_{m-1}(x_i)}\right] = y_i - F_{m-1}(x_i) \qquad (2)$$

Donde $F_{m-1}(x_i)$ es la predicción del ensamble en la iteración m-1 sobre el ejemplo i y r_{im} es el residuo de la predicción sobre dicho ejemplo.

Para suavizar el proceso de minimización del error, es común modular la aportación de cada modelo a la predicción final mediante un factor de aprendizaje. Teniendo esto en cuenta, la predicción que realizará el meta-modelo vendrá dada según la expresión (3).

$$\hat{y}(x) = F_m(x) = \sum_{m=1}^{M} \propto h_m(x) \qquad (3)$$

Para la evaluación de los modelos entrenados en los diferentes experimentos que han sido realizados y que se especificarán más adelante, utilizaremos **validación por retención** y **validación cruzada por k-pliegues**. La validación por retención consiste en dividir el conjunto de ejemplos conocidos en dos subconjuntos: un subconjunto de entrenamiento a partir del cual se construirá el modelo y un subconjunto de prueba del que se podrá calcular el rendimiento del modelo. Como los conjuntos de entrenamientos utilizados para la experimentación no son considerablemente grandes, se utilizará de forma adicional la validación cruzada por k-pliegues, que consiste en subdividir el conjunto de ejemplos en k subconjuntos y por cada uno de estos subconjuntos se entrena el modelo con el resto de los subconjuntos y se calcula el rendimiento del modelo con el subconjunto seleccionado. De esta forma el rendimiento del modelo se calcula como la media de las estimaciones sobre cada subconjunto.

Finalmente, en la experimentación se seleccionarán los valores más adecuados de los hiperparámetros del meta-modelo mediante una **búsqueda en rejilla**. Esta búsqueda consiste en fijar unos valores a considerar para cada uno de estos hiperparámetros y para cada combinación de esos valores se entrenará y evaluará un modelo que los implemente.

B. Trabajo Relacionado

En esta sección se comentan algunos trabajos relacionados, de los que se puede extraer información relevante.

- **Métodos de ensamblado en Machine Learning** [5]**:** este trabajo de fin de Máster llevado a cabo por Ricardo Recarey en el curso 2020/2021 realiza un estudio sobre las diferentes técnicas de ensamblado, así como la realización de ejercicios usando múltiples métodos (CART, KNN, Bagging, etc.) sacando conclusiones interesantes como la disminución de la varianza usando métodos de ensamblado en comparación con los métodos individuales.
- **Modelo híbrido basado en inteligencia artificial** [7]**:** este trabajo de grado realizado en la Universidad Francisco de Paula en Santander usa un algoritmo de ensamblado combinando métodos como arboles de decisión o regresión lineal múltiple para controlar de manera automática el despacho de energía dentro de una microrred de energías renovables. El uso de dicho algoritmo permitió una predicción de las energías renovables considerablemente precisa mostrando su efectividad en casos prácticos reales.

III. METODOLOGÍA

Para facilitar el uso del algoritmo de ensamble secuencial se ha decidido implementar la clase **SequentialRegressor**, que encapsulará toda esta lógica. Dicha clase utilizará gran parte de la funcionalidad de la librería Scikit-learn [6]. Para simplificar la implementación y mejorar la usabilidad, la clase implementada **hereda de BaseEstimator** lo que permitirá el uso de funciones de validación cruzada que se expondrán más adelante.

Para el funcionamiento de la clase SequentialRegressor se han especificado diferentes parámetros de entradas que se especificarán a continuación:

- **Estimator:** modelo predictivo que se usará para la construcción en secuencia que constituirá el meta-modelo. Dicho modelo debe ser de la librería Scikit-learn o en su defecto un modelo que debe heredar de la clase BaseEstimator.
- **N_estimators:** número de modelos predictivos ensamblados secuencialmente. Toma cualquier valor a partir de uno.

- **Sample_size:** proporción de ejemplos del conjunto de entrenamiento que se utilizará para entrenar cada modelo. Es un número en el rango (0,1].
- **Lr:** factor de aprendizaje del meta-modelo.
- **Hyperparameters:** hiperparámetros de los modelos predictivos utilizados.

A continuación, se exponen las diferentes funciones implementadas por la clase SequentialRegressor para el entrenamiento, predicción y evaluación de modelos en secuencia, así como su funcionamiento:

- **Fit:** construye y entrena el meta-modelo**.** Recibe los atributos y variable objetivo que constituyen el conjunto de entrenamiento. En la figura Pseudocódigo 1, se explica el procedimiento de entrenamiento del meta-modelo.

Procedimiento Fit
Entrada:
- Un Array o DataFrame con los atributos de entrenamiento
- Un Array o DataFrame con la variable objetivo

Salidas:
- Conjunto de modelos entrenados

Algoritmo:
1. Inicializar la primera predicción $pred_0$ todo a zeros
2. $pred_{actual} = pred_0$
3. Por cada i en n_estimators:
   a. $residuo_i = y - pred_{actual}$
   b. Seleccionar las sample_size*N filas del conjunto de entrenamiento a considerar, siendo N el número de ejemplos del conjunto de entrenamiento, para entrenar el $modelo_i$
   c. Entrenar el $modelo_i$ usando $residuo_i$ como variable objetivo
   d. Obtener las predicciones, $pred_i$, del $modelo_i$
   e. $pred_{actual} = pred_{actual} + lr*pred_i$

Pseudocódigo 1. Algoritmo de entrenamiento

- **Predict:** recibe un conjunto de datos de prueba y proporciona una predicción final para cada ejemplo de este. En la figura Pseudocódigo 2, se explica el procedimiento de predicción del meta-modelo.

Procedimiento Predict
Entrada:
- Un Array o DataFrame conjunto de datos de prueba

Salidas:
- Predicciones sobre el conjunto de prueba

Algoritmo:
1. Inicializar la predicción $pred_{final}$ todo a zeros
2. Por cada $modelo_i$ que constituyen el meta-modelo:
   a. Realizar la predicción, $pred_i$, del $modelo_i$ sobre el conjunto de prueba
   b. $pred_{final} = pred_{final} + lr*pred_i$

Pseudocódigo 2. Algoritmo de predicción

- **Evaluate:** evalúa el meta-modelo mediante validación cruzada por k-pliegues. En la figura Pseudocódigo 3, se explica el procedimiento de evaluación del meta-modelo para unos hiperparámetros dados. La validación cruzada no ha sido implementada, sino que se ha utilizado la función cross_validate de la librería Scikit-learn [8]. Esto es posible gracias a heredar a la clase BaseEstimator y haber declarado las funciones fit y predict. Además, es necesario declarar como atributos de la clase SequentialRegressor, todos los valores de entrada del constructor.

Procedimiento Evaluate
Entrada:
- Un Array o DataFrame con los atributos de entrenamiento
- Un Array o DataFrame con la variable objetivo
- Un entero k que indica el número de pliegues

Salidas:
- Coeficiente de determinación medio conseguido
- Error absoluto medio negativo conseguido

Algoritmo:
1. Dividir el conjunto de datos en k particiones
2. Por cada partición i realizar:
   a. Entrenar un ensamble de modelos sobre las particiones restantes
   b. Usar el meta-modelo obtenido para generar prediciones sobre la partición i
   c. Medir el rendimiento sobre la partición de prueba i usando como métricas el coeficiente de determinación y el error absoluto medio
3. Devolver el rendimiento medio para cada una de las métricas

Pseudocódigo 3. Algoritmo de evaluación

A continuación, se describirá la metodología seguida para la fase de experimentación, que se dividirá en tres partes:

- **Experimentación sobre el meta-modelo:** en esta fase se entrena y se evalúa el rendimiento de un meta-modelo, mediante validación por retención y por validación cruzada por k-pliegues. Estos experimentos se realizan con dos modelos predictivos de la librería Scikit-learn, el modelo **DecisionTreeRegressor** [9] que implementa el algoritmo de predicción CART y el modelo **KNeighborsRegressor** [10] que implementa el algoritmo de KNN.
- **Experimentación sobre modelos predictivos individuales:** para la comparación del rendimiento de los diferentes meta-modelos entrenados en la

sección anterior, se dispone a realizar los mismos experimentos, con las mismas configuraciones de hiperparámetros, con los modelos DecisionTreeRegressor y KNeighborsRegressor de manera individual sin el ensamble secuencial.

- **Experimentación con hiperparámetros:** una vez analizado el impacto del uso del ensamblado secuencial para unos hiperparámetros cualesquiera, en esta fase de la experimentación se hace una exploración en profundidad de la capacidad de mejora del meta-modelo con la variación de los diferentes hiperparámetros tanto del meta-modelo como de los modelos predictivos utilizados en la construcción de este. Para esta exploración se utilizará una **búsqueda en rejilla.**

Para cada una de las partes de esta experimentación, se han utilizado dos conjuntos de entrenamiento diferentes. El primer archivo "house_prices.csv" contiene información que describe las características, el estado y el precio de viviendas que han sido vendidas a través de una agencia inmobiliaria. El segundo archivo "parkinsons.csv" contiene información de pacientes en fase inicial de la enfermedad de Parkinson.

Antes de empezar las diferentes fases de experimentación se han tratado mínimamente los diferentes archivos. Para el archivo de las viviendas se codifican las columnas con valores de texto a valores numéricos con la clase **OrdinalEncoder** [12] y a continuación, se escalan todas las columnas para que se encuentren en el rango [0,1] con la clase **MinMaxScaler** [13]. Para el archivo de la enfermedad de Parkinson no es necesario realizar la codificación, ya que todas las columnas tienen valores numéricos y se realiza el escalado de la misma forma que con el archivo anteriormente comentado. Para efectuar la validación por retención se dividen los conjuntos de entrenamiento con la función **train_test_split** [14] en dos subconjuntos uno que se utilizará para el entrenamiento de los modelos y otro para su evaluación.

A. Experimentación sobre el meta-modelo.

En primer lugar, se establecieron los parámetros para usar con el algoritmo tanto para **DecisionTreeRegressor** como **KNeighborsRegressor**:

- **Lr**: 0,5
- **N_estimators:** 4
- **Sample_size:** 0,75

Posteriormente se llevan a cabo las pruebas sobre los dos conjuntos de datos mencionados.

DecisionTreeRegressor: en el caso del uso del árbol de decisión, se escogió sin estudio previo una profundidad máxima del árbol de 10 ('max_depth') y un mínimo de dos muestras para realizar la división ('min_samples_split') con razón de observar los resultados del modelo en primera instancia y comprobar su funcionamiento. Para la comprobación de los resultados se calculó el coeficiente de determinación y el error absoluto medio usando validación por retención y posteriormente validación cruzada. En el caso de la validación cruzada se usó la función **evaluate** de la clase implementada, para obtener los resultados de la evaluación mientras que para la validación por retención se usaron funciones de la biblioteca de Sickit-Learn.

KNeighborsRegressor: el proceso para la obtención de resultados con el algoritmo KNN es muy similar al anterior con la única diferencia de los hiperparámetros usados, se eligió la distancia euclídea ('euclidean') y el total de cinco vecinos dentro del algoritmo ('n_neighbors') para la predicción.

B. Experimentación sobre modelos predictivos individuales.

Una vez obtenido los primeros resultados del modelo de ensamblado, es necesario analizar el comportamiento de los mismos modelos, pero aislados, con los mismos hiperparámetros y las mismas métricas para evaluar su rendimiento. De esta forma se llevará a cabo una comparación entre los modelos individuales y los modelos usados dentro del ensamblado.

Se volverán a calcular predicciones sobre los dos conjuntos de datos, a través de los algoritmos de árbol de decisión y KNN. Tras realizar predicciones volviendo a usar validación cruzada y validación por retención, se utilizan de nuevo el error absoluto medio y el coeficiente de determinación.

C. Experimentación con hiperparámetros.

A continuación, se detalla el proceso seguido para mejorar el meta-modelo, centrado en la búsqueda de hiperparámetros que ofrezcan el mejor rendimiento posible, considerando las limitaciones computacionales de los dispositivos utilizados, así como el tiempo de ejecución y el coste computacional.

El procedimiento se divide en cuatro fases:

- **Búsqueda de hiperparámetros para el ensamblado:** Con el objetivo de encontrar la mejor configuración de hiperparámetros y optimizar el meta-modelo, se empleó la función GridSearchCV [11], la cual implementa una búsqueda en rejilla usando en el proyecto un número de 10 pliegues y utilizando como estimador la clase del modelo de ensamblado con el método de predicción correspondiente. Los hiperparámetros a evaluar se definieron mediante un diccionario de pares clave-valor, donde cada clave representa un hiperparámetro y cada valor el conjunto de posibles valores a explorar.

  En el proceso de búsqueda se consideraron tanto los parámetros del modelo de ensamblado como los hiperparámetros de los algoritmos base utilizados en el mismo. Estos conformaron la rejilla sobre la que GridSearchCV realizó la exploración exhaustiva. Como métrica de evaluación para seleccionar la mejor combinación de hiperparámetros, se utilizó nuevamente el coeficiente de determinación (R^2), por su capacidad para reflejar el grado de ajuste del modelo.

Los rangos seleccionados para los parámetros del ensamblador fueron:

- **Lr:** valores entre 0,5 y 0,9 con saltos de dos décimas.
- **N_estimators:** valores entre 4 y 10.
- **Sample_size:** valores entre 0,5 y 0,9 con saltos de dos décimas.

Se llevaron a cabo un total de cuatro búsquedas en rejilla: dos utilizando el algoritmo DecisionTreeRegressor (una por cada conjunto de datos) con las que se comprobaron profundidades máximas ('max_depth') de [3,5,7,10] y mínimo número de muestras para división de [2,5,10]. Paralelamente otras dos empleando el algoritmo KNeighborsRegressor, usando como número de vecinos('n_neighbors') los valores [3,5,7], como pesos de los vecinos('weights') se utilizaron pesos uniformes y pesos en función de la distancia, como métrica de distancia se probó con la distancia euclídea y la de manhattan. Finalmente, los hiperparámetros óptimos identificados por cada búsqueda, junto con sus respectivos valores de R^2, fueron almacenados para su uso en la siguiente fase del experimento.

- **Cálculo de eficiencia del ensamblado con los mejores hiperparámetros:** En esta etapa, se evalúa el rendimiento del modelo de ensamblado utilizando los hiperparámetros previamente seleccionados como óptimos. Para ello, se calculan dos métricas fundamentales: el coeficiente de determinación (R^2) y el error absoluto medio (MAE). Estos cálculos se realizan aplicando validación cruzada con 10 pliegues, ya que se ha considerado el enfoque más representativo para reflejar el comportamiento general del modelo. El análisis se lleva a cabo empleando los dos algoritmos base en el ensamblado ya utilizados en la experimentación sobre el meta-modelo: el árbol de decisión (**DecisionTreeRegressor)** y el método K-Nearest Neighbors(**KNeighborsRegressor**). Para obtener los resultados, se ejecutan dos predicciones independientes con el modelo ensamblado, una para cada algoritmo mencionado. A la clase correspondiente al ensamblado se le proporcionan los hiperparámetros óptimos, y los valores de las métricas se obtienen mediante la función evaluate definida en dicha clase.

  Los hiperparámetros óptimos a usar fueron:

  - **DecisionTreeRegressor:** se utilizó para el conjunto de datos de viviendas una profundidad máxima de 3, un número mínimo de dos muestras por división, un factor de aprendizaje de 0,5, 9 iteraciones para el ensamblado y un tamaño de datos de prueba de 0,5. En el caso del conjunto de la enfermedad del Parkinson se mantuvieron los datos de factor de aprendizaje e iteraciones del ensamblador, mientras que la profundidad máxima pasó a ser de 10, el conjunto de datos de prueba a 0,9 y el número mínimo de muestras por división a 5.
  - **KNeighborsRegressor:** En el caso del conjunto de datos de viviendas el mejor rendimiento del modelo KNN se obtuvo utilizando 7 vecinos con una métrica de distancia de Manhattan es decir con p igual a 1 y una ponderación de tipo distancia lo que significa que los vecinos más cercanos tienen mayor peso, además se emplearon 7 iteraciones para el ensamblado, una tasa de aprendizaje de 0,9 y un tamaño de muestra del 0,5 mientras que para el conjunto de datos de Parkinson el mejor resultado se consiguió con 5 vecinos también con distancia de Manhattan y ponderación por distancia junto con 5 estimadores, una tasa de aprendizaje de 0,7 y un tamaño de muestra del 0,9.

- **Búsqueda de hiperparámetros para los modelos individuales:** Para determinar los mejores valores de los hiperparámetros en los modelos individuales, se empleó nuevamente la función GridSearchCV, utilizada previamente en la primera fase. En este caso, se proporcionó un diccionario con los hiperparámetros a evaluar, limitado exclusivamente a los correspondientes a cada modelo individual (excluyendo los parámetros del modelo ensamblador).

  La métrica seleccionada para evaluar el rendimiento fue, una vez más, el coeficiente de determinación (R^2), con el objetivo de mantener la coherencia y facilitar la comparación de resultados entre ambas fases del experimento. Se utilizaron los mismos rangos de valores que en la primera fase para los algoritmos analizados.

  Se realizaron un total de cuatro búsquedas en rejilla: dos utilizando el algoritmo DecisionTreeRegressor y dos con KNeighborsRegressor, cada una correspondiente a los distintos conjuntos de datos disponibles. Los hiperparámetros óptimos obtenidos, junto con sus respectivos valores de R^2, fueron almacenados para su posterior análisis.

- **Cálculo de eficiencia de métodos individuales con mejores hiperparámetros:** en esta última fase se usaron los hiperparámetros óptimos obtenidos en la experimentación anterior para evaluar el rendimiento de los modelos individuales óptimos y así compararlos con los resultados obtenidos con el método de ensamblado.

Se realizó el análisis de las mismas métricas usadas en el cálculo de eficiencia del meta-modelo mejorado por hiperparámetros(R^2 y MAE). Una vez más se utilizaron los algoritmos de árbol de decisión y KNN, en este caso con las siguientes entradas óptimas:

- **DecisionTreeRegressor:** el conjunto de datos de las viviendas se le aplicó una máxima profundidad de 7 y mínimo número de muestras para la división de 10, mientras que para el conjunto de la enfermedad de Parkinson se aplicó una máxima profundidad de 10 y un mínimo de 2 muestras.
- **KNeighborsRegressor:** al conjunto de datos de viviendas se le aplicó un total de 7 vecinos de proximidad, la métrica de distancia de manhattan y pesos ponderados por distancia, mientras que para el conjunto de la enfermedad de Parkinson se aplicaron 5 vecinos, métrica manhattan y pesos ponderados por distancia.

IV. Resultados

En esta sección se exponen los resultados obtenidos en los diferentes experimentos anteriormente descritos.

En la Tabla 1 se exponen los resultados obtenidos para la experimentación sobre los diferentes meta-modelos.

Modelos predictivos	*Precio de viviendas*			
	Validación por retención		*Validación cruzada*	
	R^2	*MAE*	R^2	*MAE*
CART	0,65777584	29325,109	0,64141334	29888,189
KNN	0,67180996	28768,631	0,64701402	29389,375
Modelos predictivos	*Enfermedad de Parkinson*			
	Validación por retención		*Validación cruzada*	
	R^2	*MAE*	R^2	*MAE*
CART	0,84198872	3,0497759	0,83715977	3,0107721
KNN	0,53580558	5,0899081	0,45677398	5,3439219

Tabla 1. Resultados experimentación sobre el meta-modelo

En la Tabla 2 se exponen los resultados obtenidos para la experimentación sobre los modelos predictivos individuales.

Modelos predictivos	*Precio de viviendas*			
	Validación por retención		*Validación cruzada*	
	R^2	*MAE*	R^2	*MAE*
CART	0,61116899	34126,178	0,47064096	33203,516
KNN	0,69360834	29221,305	0,67369376	28452,665
Modelos predictivos	*Enfermedad de Parkinson*			
	Validación por retención		*Validación cruzada*	
	R^2	*MAE*	R^2	*MAE*
CART	0,88166319	1,4747014	0,85156093	1,8189509
KNN	0,57410969	4,6179443	0,53269261	4,7187009

Tabla 2. Resultados experimentación sobre modelos individuales

En la Tabla 3 se exponen los resultados obtenidos para la mejor configuración de hiperparámetros encontrados para los diferentes meta-modelos y modelos individuales.

Modelos predictivos	*Precio de viviendas*			
	Meta-modelos		*Modelos individuales*	
	R^2	*MAE*	R^2	*MAE*
CART	0,64950076	27386,114	0,49273382	32148,577
KNN	0,72095424	25581,735	0,71317533	25591,467
Modelos predictivos	*Enfermedad de Parkinson*			
	Meta-modelos		*Modelos individuales*	
	R^2	*MAE*	R^2	*MAE*
CART	0,91141112	1,7023483	0,84269208	1,8755817
KNN	0,60210957	4,3092090	0,60473113	4,2671918

Tabla 3. Resultados experimentación con mejores hiperparámetros encontrados

Si se observan los resultados obtenidos para los meta-modelos y los modelos individuales para unos hiperparámetros elegidos al azar, los resultados de los meta-modelos son en general muy parecidos a los resultados obtenidos para los modelos individuales, llegando a mejorar en algunas ocasiones a estos últimos, pero sin llegar a ser cambios significativos como para teorizar todavía. Para el conjunto de entrenamiento de las viviendas, los meta-modelos que usan el algoritmo de CART dan una mejoría considerable frente a los modelos individuales, tanto en el coeficiente de determinación como para el error absoluto medio, sin embargo, para los que usan el algoritmo de KNN los resultados son algo más ajustados. Para el conjunto de entrenamiento de la enfermedad de Parkinson, los meta-modelos en general dan resultados muy parecidos a los modelos individuales, tanto para el coeficiente de determinación como para el error absoluto medio.

Si se observan los resultados obtenidos para la exploración de hiperparámetros, es posible sacar mejores conclusiones que con la información anterior. Para los dos conjuntos de entrenamiento el uso del algoritmo de árboles CART para la construcción de los meta-modelos, ha significado una mejora considerable en la eficiencia de estos. Por otro lado, los meta-modelos construidos a partir de KNN no han supuestos cambios significativos en la eficiencia de estos. En la Tabla 4

se puede apreciar la diferencia entre los resultados de los meta-modelos y los modelos individuales y como se puede observar la mejora para CART es bastante considerable.

Modelos predictivos	***Precio de viviendas***	
	Meta-modelos – Modelo individual	
	R^2	***MAE***
CART	0,15676694	-4762,463
KNN	0,00777891	-9,732
Modelos predictivos	***Enfermedad de Parkinson***	
	Meta-modelos – Modelo individual	
	R^2	***MAE***
CART	0,06871904	-0,1732334
KNN	-0,00262156	0,0420172

Tabla 4. Comparación entre meta-modelos y modelos individuales

Los resultados para los otros experimentos y para este último con KNN han sido más ajustados. Debido a estos resultados podemos teorizar dos hipótesis:

1) El modelo predictivo de árboles CART, por lo general, es el modelo que da mejores resultados en el ensamble secuencial: esto se puede apreciar en una mejora general en los experimentos que usan los modelos CART y en la falta de una considerable mejora con el uso de modelos KNN.

2) La capacidad predictiva del ensamble secuencial es mayor a la de los modelos predictivos de forma individual: esto se puede deducir debido a la notable mejora de los meta-modelos con los hiperparámetros optimizados.

V. Conclusiones

Tras el desarrollo del presente proyecto, se han realizado diversas pruebas orientadas a comparar el rendimiento de modelos individuales frente a un modelo de ensamblado secuencial construido a partir de ellos. El estudio se ha estructurado en tres fases principales: evaluación del ensamblado, evaluación de los modelos individuales y análisis comparativo.

Los resultados obtenidos muestran que el modelo de ensamblado presenta un rendimiento superior, especialmente cuando se emplea el algoritmo CART como base. Esta mejora ha sido posible gracias a una adecuada selección de hiperparámetros, que ha permitido optimizar el comportamiento del meta-modelo frente a sus componentes individuales.

No obstante, se identificó una limitación importante en los recursos computacionales disponibles, lo que restringió la posibilidad de realizar experimentos con configuraciones más exigentes. Es razonable suponer que, con mayor capacidad computacional, los resultados podrían mejorar aún más, especialmente en lo que respecta al número de iteraciones y al tamaño del espacio de búsqueda de hiperparámetros.

Como líneas futuras de trabajo, se propone ampliar el número de algoritmos base utilizados en el ensamblado, incrementar el tamaño y variedad de los conjuntos de datos, y evaluar el rendimiento del modelo en contextos aplicados reales. Estas mejoras permitirían validar la robustez del enfoque propuesto y profundizar en el análisis de técnicas de ensamblado secuencial.

VI. Uso Autodeclarativo de IA

En el desarrollo de este estudio se ha llevado a cabo el uso de la IA con el fin de ofrecer apoyos conceptuales en el desarrollo del trabajo, así como ayudas funcionales para la mejora de la expresión escrita con el uso de un lenguaje más técnico.

Los prompts proporcionados han sido:

- Estoy usando la libreria de sickit learn para hacer un algoritmo de ensamble secuencial de modelos. Necesito que me ayudes a utilizar la función GridSearchCV ya que necesito pasarle hiperparametros relacionados con el estimator DecisionTreeRegressor, es decir max_depth.
- Escríbeme este párrafo haciéndolo más formal para su uso en un proyecto científico: El objetivo principal de este trabajo es elaborar un algoritmo de ensamblaje secuencial de modelos predictivos de inteligencia artificial para comprobar la mejora de estos en comparación con el uso de dichos modelos, pero de forma individual.
- ¿Qué debo modificar en la siguiente clase, para poder realizar una búsqueda en rejilla con GridSearchCV?:

  class SequentialRegressor(BaseEstimator): ….

Referencias

[1] S. Russell, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed, Pearson, 2010.

[2] Artículo de la universidad complutense de madrid https://www.masterdatascienceucm.com/que-es-machine-learning/. Consultado el 2/06/2025

[3] Página web sobre ensamble de modelos.https://aprendeia.com/2019/03/29/metodos-de-ensamble-de-modelos-machine-learning-ensemble-methods-en-espanol/ Consultada el 2/06/2025

[4] Página web sobre los métodos de ensamble. https://medium.com/@oscars.cortezmo/introducci%C3%B3n-a-los-m%C3%A9todos-de-ensamble-y-al-algoritmo-de-xgboost-caso-pr%C3%A1ctico-e8cb0d58394b#84e6

[5] Trabajo de fin de grado sobre métodos de ensamblado. http://eio.usc.es/pub/mte/descargas/ProyectosFinMaster/Proyecto_1686.pdf

[6] Documentación de la librería Scikit-learn. https://scikit-learn.org/stable/api/index.html

[7] Trabajo de fin de grado sobre modelo híbrido de inteligencia artificial: https://repositorio.ufps.edu.co/handle/ufps/6779

[8] Documentación de cross_validate. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html

[9] Documentación de DecisionTreeRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

[10] Documentación de KNeighborsRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

[11] Documentación de GridSearchCV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[12] Documentación de OrdinalEncoder. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OrdinalEncoder.html

[13] Documentación de MinMaxScaler. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[14] Documentación de train_test_split. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html