UNIVERSIDAD DE SEVILLA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Inteligencia Artificial – Curso 2024/2025

Ensamble Secuencial de Modelos

Trabajo práctico

Autores

Javier Clavijo Martínez
Manuel Roberto López Pavía

Sevilla, Junio de 2025

Ensamble Secuencial de Modelos

Javier Clavijo Martínez
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
javclamar@alum.us.es

Manuel Roberto López Pavía
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
manloppav@alum.us.es

***Resumen*—En este proyecto se crea un algoritmo para construir modelos de regresión basados en ensambles de otros modelos base (Árbol de decisión, KNN, Regresión Lineal). Dentro del ensamble, cada modelo se especializa en aprender y enmendar los errores que ha cometido el modelo anterior, minimizando así el error en cada iteración mediante el descenso por el gradiente.**

Aparte de la implementación del meta-modelo, se incluyen varios cuadernillos con la experimentación de varios modelos base sobre dos CSV diferentes. En cada uno de los cuadernillos se prueba el meta-modelo con diferentes hiperparámetros, viendo cuáles son más eficientes y en qué dataset lo son.

***Palabras clave*—Meta-Modelo, ensamble de modelos, hiperparámetro, gradiente, regresión, CSV, estimador,**

I. Introducción

En la Inteligencia Artificial, el ensamble de modelos predictivos ha demostrado ser una estrategia eficaz para mejorar el rendimiento y la tolerancia de los sistemas de Aprendizaje Automático.

El ensamble de diferentes modelos permite aprovechar las fortalezas de modelos más simples, referidos como modelos base, para construir un meta-modelo que tenga mucha mayor capacidad predictiva. Existen dos tipos de estrategias a seguir a la hora de implementar un ensamble, en paralelo o en secuencial, siendo esta última en la que nos centramos en este trabajo.

Las ventajas de los ensambles son varias, desde una mejor predicción, mayor robustez ante fallos que dan los modelos individuales, o una gran flexibilidad en las tareas que puede llevar a cabo, todo esto dependiendo del modelo base (o conjunto de modelos base) que se escoja[2]

Como ya hemos dicho, en este proyecto aplicamos la técnica de ensamble secuencial para la creación de un meta-modelo, que en este caso va a estar orientado a tareas de regresión, aunque también se podría implementar para tareas de clasificación.

El ensamble de modelos secuencial se puede implementar de dos formas, usando diferentes algoritmos dentro de los modelos-base, o utilizando el mismo algoritmo para todos los modelos base pero sobre variaciones del conjunto de entrenamiento. Nosotros nos centramos en la segunda opción, en la que cada modelo se centra en predecir los errores cometidos por el ensamble hasta el momento. Para eso se utiliza el gradiente del error como métrica objetivo en cada iteración, lo cual convierte esta estrategia en un descenso por el gradiente.

Viendo lo anterior, nuestro ensamble cae dentro del método de Boosting, más concretamente Gradient Boosting. Aparte de este método, en los ensambles se pueden aplicar también Bagging (el ensamble toma la mejor predicción mediante Bootstrapping y Aggregation) y Stacking (los modelos base son diferentes unos de otros) [1]

Habiendo explicado los conceptos teóricos, la implementación se ha realizado en una clase general de Python compatible con la interfaz de Scikit-Learn, que permite construir ensambles secuenciales a partir de cualquier modelo regresor dentro de la biblioteca Scikit-Learn y otros hiperparámetros que se explicarán más adelante.

Además de la implementación del meta-modelo, se han desarrollado varios cuadernillos de experimentación, teniendo como objetivo analizar cómo afectan diferentes valores de los hiperparámetros a las predicciones finales, utilizando como métrica final de evaluación R^2.

Por tanto, este trabajo no aborda solo el diseño e implementación de un ensamble, sino que también, y con bastante extensión, se analiza empíricamente su comportamiento sobre los distintos conjuntos de datos e hiperparámetros.

Por último, esta memoria cuenta con otros 4 apartados, siendo estos:

- Preliminares
- Metodología
- Resultados
- Conclusiones

II. Preliminares

En esta sección se hace una breve introducción de las técnicas empleadas.

A. Métodos empleados

En esta sección se recorren todos los métodos empleados dentro de la implementación y experimentación del meta-modelo.

1) Métodos de implementación:

- Submuestreo sin reemplazo: En cada iteración se entrena al modelo con un subconjunto aleatorio, favoreciendo así la diversidad de los modelos dentro del ensamble.
- Minimazción del error: La función de perdida que se usa en el entrenamiento del modelo será el error cuadrático[1] calculado en cada iteración, por tanto se quiere reducir el gradiente de los ejemplos respecto a sus predicciones[2].

A partir de esto, conseguimos la variable objetivo a predecir en cada iteración, o sea, en la adición de un nuevo modelo base, que será el gradiente cometido sobre cada ejemplo en la iteración anterior, llamado residuo[3].

$$\mathrm{L}(\mathrm{y}_i, F(x_i)) = \tfrac{1}{2}(y_i - F(x_i))^2$$

Fig. 1. Error cuadrático

$$-\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right] = y_i - F(x_i)$$

Fig. 2. Gradiente del error

$$\mathrm{r}_{im} = -\left[\frac{\partial L(y_i, F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}\right] = y_i - F_{m-1}(x_i)$$

Fig. 3. Residuo

1) $F(x_i)$: Predicción del ensamble sobre el ejemplo x_i
2) y_i: Valor objetivo real del ejemplo x_i
3) F_{m-1}: Predicción del modelo base F_{m-1} para el ejemplo x_i

- Clase Genérica: El meta-modelo se ha implementado en una clase genérica usando BaseEstimator y RegressorMixin, lo que permite usar el modelo con métodos como GridSearch o cross-val-score, dando una gran modularidad.
- Predicción: La predicción final del meta-modelo se definiría como la suma de las predicciones de todos los modelos base dentro del ensamble, multiplicadas por la tasa de aprendizaje

$$\hat{y}(x) = F_m(x) = \textstyle\sum_{m=1}^{M} \alpha \cdot h_m(x)$$

Fig. 4. Predicción del meta-modelo

1) $\hat{y}(x)$: Predicción del ensamble sobre sobre el ejemplo x
2) $h_m(x)$:Predicción del modelo entrenado en la iteración m sobre el ejemplo x
3) α: Tasa de aprendizaje
4) M: Número total de modelos dentro del ensamble

- Early-Stopping: Se ha añadido el mecanismo de early-stopping al algoritmo de entrenamiento. Esto permite evaluar la mejora que se produce en cada iteración y detener el entrenaiemitno cuando hayan pasado unas iteraciones sin una mejora sustancial. Este mecanismo traea varias mejora, evitando overfitting en el algoritmo, evitando una perdidad de rendimiento computacional, y haciendo que el algoritmo sea más robusto frente a cambios en el dataset[3]

2) Métodos de experimentación:

- Validación Cruzada: Para la evaluación se ha usado el método cross-val-score de Scikit-Learn con la métrica principal R^2 para medir el rendimiento del meta-modelo con diferentes hiperparámetros
- Exploración de hiperparámetros: Se han añadido funciones explorar hiper-parámetros para cada tipo de modelo base y así sistematizar la experimentación

III. Metodología

Esta sección se dedica a la descripción del método implementado en el trabajo, entrando en profundidad en los algoritmos de entrenamiento, predicción y evaluación, así como los modelos base que se han utilizado.

El meta-modelo esta implementado mediante una clase de Python que extiende de BaseRegressor y de RegressorMixin, lo que permite utilizarlo en métodos de validación de la librería Scikit-Learn agilizando mucho el proceso.

Dentro de esta clase se implementan los algoritmos de entrenamiento y predicción.

A. Algoritmo de Entrenamiento

El algoritmo de entrenamiento está implementado dentro del método fit(), método que hay que implementar ya que nuestra clase extiende de BaseRegressor. Este algoritmo aplica el método del descenso por el gradiente, utilizando el residuo obtenido anteriormente como variable objetivo para cada iteración, haciendo que en cada iteración el modelo corrija los errores de la anterior. El algoritmo tiene como entrada un conjunto de entrenamiento, un número de estimadores(modelos

base), la proporción de ejemplos que se usará del conjunto del entrenamiento en cada iteración, la tasa de aprendizaje a utilizar, el estimador base(el modelo base a utilizar en el ensamble), y por último los hiperparámetros que se pueden modificar del modelo base.

En la siguiente figura se añade el pseudocódigo que define el comportamiento al completo del algoritmo de entrenamiento:

Algorithm 1 Entrenamiento del Ensamble Secuencial

pred_actual $\leftarrow$ media(y)
residual $\leftarrow y -$ pred_actual
models $\leftarrow [\,]$
if early_stopping activado **then**
    best_val_mse $\leftarrow \infty$
    no_improve_count $\leftarrow 0$
end if
for $i = 1$ **to** $n\_estimators$ **do**
    Seleccionar aleatoriamente sin reemplazo un subconjunto de índices: idx
    $X_{train} \leftarrow X[idx]$, $y_{train} \leftarrow$ residual$[idx]$
    $model \leftarrow$ clonar($base\_estimator$)
    Entrenar $model$ con (X_{train}, y_{train})
    $pred_i \leftarrow model.predict(X)$
    pred_actual $\leftarrow$ pred_actual $+ lr \cdot pred_i$
    residual $\leftarrow y -$ pred_actual
    Agregar $model$ a $models$
    if early_stopping **y** sample_size < 1.0 **then**
        $mask \leftarrow$ vector booleano de verdad excepto en idx
        if hay datos fuera de muestra **then**
            $X_{val} \leftarrow X[mask]$, $y_{val} \leftarrow y[mask]$
            $pred_{val} \leftarrow$ predict(X_{val})
            $val\_mse \leftarrow$ MSE($y_{val}, pred_{val}$)
            if $val\_mse < best\_val\_mse - \varepsilon$ **then**
                $best\_val\_mse \leftarrow val\_mse$
                $no\_improve\_count \leftarrow 0$
            else
                $no\_improve\_count \leftarrow no\_improve\_count + 1$
            end if
            if $no\_improve\_count \geq patience$ **then**
                break
            end if
        end if
    end if
end for
return modelo entrenado con todos los submodelos en $models$

1) Inicialización:

- Inicializar el conjunto de modelos que contiene el ensamble a $\emptyset$.
- Inicializar la predicción inicial como la media de la variable objetivos en los ejemplos
- Inicializar la lista que guarda las predicciones del meta-modelo
- Inicializar el residuo, como la resta de la predicción inicial(media) y la realidad

2) Iteración:

- Bucle que itera hasta llegar a n (número de estimadores)
- Selecionar un conjunto aleatorio de los datos, dependiendo de la proporción s e inicializar X_i e r_i
- Se clona el estimador base
- Se entrena el estimador clonado sobre X_i e r_i (para predecir los siguientes r_i)
- Actualizamos las predicciones con las del modelo que se acaba de entrenar y se le suma a las predicciones acumuladas
- Se actualiza el residuo
- Se añade el modelo entrenado al conjunto de modelos del ensamble

B. Algoritmo de Predicción

El algoritmo de predicción[2] es bastante simple, consistiendo en la suma de todas las predicciones de los modelos que se encuentran en el ensamble, multiplicadas por la tasa de aprendizaje.

Algorithm 2 Predicción con el Ensamble Secuencial

Require: Conjunto de datos X, lista de modelos $\mathcal{F}$, tasa de aprendizaje α, predicción inicial $\hat{y}^{(0)}$
Ensure: Predicciones acumuladas del meta-modelo
Inicializar vector de predicciones: $pred \leftarrow [\hat{y}^{(0)}, \ldots, \hat{y}^{(0)}]$
for all $modelo \in \mathcal{F}$ **do**
    $pred \leftarrow pred + \alpha \cdot modelo$.predecir($X$)
end for
return $pred$

1) Inicialización:

- Inicializar el vector de predicciones de los modelos

2) Iteración:

- Para cada modelo base del meta-modelo
- Añadir la predicción del modelo para el ejemplo multiplicada por el factor de aprendizaje

C. Algoritmo de Exploración de Hiperparámetros

El algoritmo de Exploración de Hiperparámetros[3] no estaba requerido en la propuesta del proyecto, pero nos ayuda a sistematizar combinaciones de estos sin que sea muy tedioso y lento. Este algoritmo tiene como entrada un estimador base, el que se utiliza para el meta-modelo, un atributo paramgrid, que es un diccionario que contiene los hiperparámetros a explorar del meta-modelo, dos atributos X/Y que son los datos

de entrenamiento, y por último el atributo cv, que debe ser el generador KFold para la validación cruzada.

Además el algoritmo no devuelve los resultados de cada meta-modelo que se prueba, sino que los presenta en un dataset , donde se ven los hiperparámetros de cada meta-modelo y su media de R^2, que es la métrica a utilizar en la evaluación. Se puede observar un ejemplo en la Tabla 1[I]

Algorithm 3 Exploración de hiperparámetros del meta-modelo

Require: Clase del estimador base C, diccionario de hiperparámetros P, datos X, Y, y por último un generador de validación cruzada cv

Ensure: Tabla ordenada con combinaciones de hiperparámetros y su rendimiento medio R^2

Inicializar lista de resultados: $resultados \leftarrow [\,]$
for all $n \in P[n\_estimators]$ **do**
  for all $\alpha \in P[lr]$ **do**
    for all $s \in P[sample\_size]$ **do**
      for all $d \in P[max\_depth]$ **do**
        $base \leftarrow$ nueva instancia de C
        if $d \neq$ None **then**
          Establecer $base.max\_depth \leftarrow d$
        end if
        Crear modelo: $M \leftarrow$ `SequentialEnsembleRegressor` con $base$, n, α, s
        Calcular puntuaciones: $scores \leftarrow cross\_val\_score(M, X, y, cv, scoring =' r2')$
        Agregar resultado a $resultados$:
        guardar $\{n, \alpha, s, d, \text{media}(scores)\}$
      end for
    end for
  end for
end for
Crear y ordenar tabla T con $resultados$ por $r2\_mean$ descendente
return T

TABLA I
EJEMPLO DE SALIDA DEL ALGORITMO DE EXPLORACIÓN DE HIPERPARÁMETROS

n_estimators	lr	sample_size	max_depth	R^2 medio
50	0.1	0.75	3	0.8421
100	0.1	0.75	3	0.8573
100	0.05	0.80	4	0.8610
150	0.05	0.80	5	0.8594

IV. RESULTADOS

En esta sección se detallarán tanto los experimentos realizados como los resultados conseguidos.

Los experimentos se han dividido en dos notebooks, uno de ellos para un ensamble que utiliza árboles de decisión como modelos base, y el último que utiliza kNN. En cada uno de estos notebooks se ha intentado conseguir los mejores hiperparámetros del meta-modelo y de los modelos base que contiene. Estos experimentos se han llevado a cabo con los dos datasets que se proporcionan en la propuesta de proyecto, asegurando unos resultados que no estan ligados a un conjunto de datos en concreto.

Al principio de cada notebook se hace un procesado de los dos CSV, tanto como de "house-prices" como de "parkinsons", eligiendo los atributos "SalePrice" y "total-UDPRS" como atributos objetivo, respectivamente. Aparte de esto, se define el KFold que va a usar la validación cruzada, con una misma semilla de reproducibilidad para que los resultados sean objetivos.

A. Ensamble de Árboles de Decisión

1) Experimentos llevados a cabo: Se ha realizado una busqueda exhaustiva de los hiperparámetros que más afectan al rendimiento, así como sus valores más óptimos. Esto se ha realizado mediante el Algoritmo de Exploración de Hiperparámetros[3] que hemos mencionado anteriormente, sistematizando los experimentos.

Como media de evauación utilizamos la media del coeficiente de determinación obtenido mediante validación cruzada.

a) House Prices:

Como valor base para la evaluación, se utiliza la media del coeficiente de determinación (R^2) obtenida mediante validación cruzada para el modelo base, en este caso un único Árbol de Decisión, y conseguimos un valor de 0.5256 (redondeado a 4 decimales).

- R^2 medio del modelo base $\leftarrow$ 0.5256

Un valor bastante pobre, indicando que un único Árbol de Decisión mejora un 50 por ciento la predicción con respecto a la media. A partir de este valor, veremos con que hiperparámetros se mejora la predicción al utilizar el ensamble, y si es eficiente o no.

Como hiperparámetros iniciales del meta-modelo se han elegido:

- base estimator: DecisionTreeRegresor(max-depth=5)
- n estimators: 100
- sample size: 0.8
- lr: 0.1
- random state: 42

Con estos valores obtenemos un R^2 medio de 0.7403, indicando ya una mejoría solo por utilizar el ensamble, sin optimizar los hiperparámetros.

Después de esto, se han realizado varias iteraciones de exploración de hiperparámetros, en los que se han elegido los valores que parecen más óptimos a priori (valores que se repiten en la parte alta de la tabla), y se ha realizado una última iteración con parámetros:

- n estimators: [50,100]
- lr: [0.05, 0.1]
- sample size: [0.6, 0.8]
- max depth: [3, 5, 7]

Y da resultados:

TABLA II
RESULTADOSDEL ALGORITMO CON DISTINTAS CONFIGURACIONES

n_estimators	lr	sample_size	max_depth	R^2 medio
100	0.05	0.80	5	0.7606
50	0.10	0.60	5	0.7595
50	0.05	0.60	7	0.7522
100	0.05	0.60	5	0.7510
50	0.05	0.60	5	0.7503
100	0.10	0.80	3	0.7488
50	0.05	0.80	5	0.7482
50	0.10	0.80	5	0.7452
100	0.05	0.60	7	0.7436
100	0.10	0.60	7	0.7422
100	0.05	0.80	3	0.7400

b) Parkinson:

Para este conjunto de datos tenemos:

- R^2 medio del modelo base $\leftarrow$ 0.4939

Como hiperparámetros iniciales del meta-modelo se han elegido:

- base estimator: DecisionTreeRegresor(max-depth=5)
- n estimators: 100
- sample size: 0.8
- lr: 0.1
- random state: 42

Con estos valores obtenemos un R^2 medio de 0.8862, mejorando el doble el poder de predicción comparando con un solo Árbol de Decisión.

Después de esto, se han realizado varias iteraciones de exploración de hiperparámetros, en los que se han elegido los valores que parecen más óptimos a priori (valores que se repiten en la parte alta de la tabla), y se ha realizado una última iteración con parámetros:

- n estimators: [50,100]
- lr: [0.05, 0., 0.21]
- sample size: [0.8]
- max depth: [20, 13]

Y da resultados:

TABLA III
RESULTADOS DEL ALGORITMO CON DIFERENTES CONFIGURACIONES

n_estimators	lr	sample_size	max_depth	R^2 medio
100	0.10	0.8	10	0.9388
100	0.05	0.8	10	0.9382
100	0.05	0.8	13	0.9371
50	0.10	0.8	10	0.9369
100	0.10	0.8	13	0.9345
50	0.10	0.8	13	0.9343
100	0.20	0.8	10	0.9275
50	0.20	0.8	10	0.9272
100	0.20	0.8	13	0.9261
50	0.20	0.8	13	0.9252
50	0.05	0.8	13	0.9241

Aparte de esto, se ha realizado un experimento para ver cómo interactúa el ensamble con early-stopping, viendo cuántas iteraciones se realizan por cada fold(5 folds del dataset Home Prices) y si en algunos de estos se aplica. Para esto se utiliza un ensamble con hiperparámetros:

- base estimator: DecisionTreeRegresor(max-depth=5)
- n estimators: 100
- sample size: 0.8
- lr: 0.1
- random state: 42
- early stopping: True
- patience: 10

- R^2 medio House Prices: 0.7396
- Iteraciones por fold: [62, 100, 68, 85, 100]

2) Resultados: Primero, analizaremos los datasets por separado:

a) House Prices:

- El valor más óptimo del parámetro max-depth(correspondiente al modelo base, no al ensamble) es 5, ya que es el que más se repite en las 11 mejores configuraciones
- El learning rate más óptimo es 0.05, un valor bajo, pero que asegura un buen aprendizaje cuando hay un gran número de estimadores(n estimators). Sin embargo cuando los n estimators son bajos, el learning rate 0.10 es más eficiente

- Los demás hiperparámetros no generan un gran cambio en los resultados, siempre y cuando se mantegan en los valores que aparecen en la tabla, que dan un rango de valores óptimos.
- El early stopping aplica en 3 de lo 5 folds, dando una mejora de rendimiento, pero una pequeña bajada en la prediccción

b) Parkinson:

- La predicción para este dataset es es mejor ya que el dataset contiene variables que tienen umbrales muy definidos, cosa que mejora mucho la eficiencia del algoritmo
- Con respecto a los hiperparámetros, no se puede sacar conclusiones muy concretas, excepto en el parámetro max-depth, donde se ve que tiene que tener un valor bastante mayor comparado con el dataset anterior

B. Ensamble de kNN

1) Experimentos llevados a cabo: El método de experimentación es el mismo que para el Ensamble de Árboles de Decisión, se ha dividido por datasets.

a) House Prices:

- R^2 medio del modelo base $\leftarrow$ 0.6707

Como hiperparámetros iniciales del meta-modelo se han elegido:

- base estimator: KNeighboursRegressor(n-neighbours=5)
- n estimators: 500
- sample size: 0.8
- lr: 0.005
- random state: 42

Con estos valores obtenemos un R^2 medio de 0.6836, por lo que al aplicar ensambles no se mejora casi nada, ahora veremos si optimizando los parámetros se consigue una mejora suatancial.

Igual que en el ensamble anterior, se han realizado varias iteraciones de exploración de hiperparámetros, en los que se han elegido los valores que parecen más óptimos a priori (valores que se repiten en la parte alta de la tabla), y se ha realizado una última iteración con parámetros:

- n estimators: [300]
- lr: [0.01, 0.015]
- sample size: [0.7, 0.8]
- n neighbours: [7, 9]
- metric: manhattan

Y da resultados:

TABLA IV
RESULTADOSDEL ALGORITMO CON DISTINTAS CONFIGURACIONES

n_estimators	lr	sample_size	n	metric	R^2 medio
300	0.010	0.70	9	manhattan	0.7052
300	0.010	0.80	9	manhattan	0.7025
300	0.010	0.70	7	manhattan	0.7017
300	0.010	0.80	7	manhattan	0.6920
300	0.015	0.70	9	manhattan	0.6726
300	0.015	0.80	9	manhattan	0.6655
300	0.015	0.70	7	manhattan	0.6625
300	0.015	0.80	7	manhattan	0.6397

b) Parkinson:

Para este conjunto de datos tenemos:

- R^2 medio del modelo base $\leftarrow$ 0.5155

Como hiperparámetros iniciales del meta-modelo se han elegido:

- base estimator: KNeigghboursRegressor(n-neighbours=5)
- n estimators: 200
- sample size: 0.8
- lr: 0.01
- random state: 42

Con estos valores obtenemos un R^2 medio de 0.5042, por lo que hay incluso una disminución en el poder predictivo del modelo

Después de esto, se han realizado varias iteraciones de exploración de hiperparámetros, en los que se han elegido los valores que parecen más óptimos a priori (valores que se repiten en la parte alta de la tabla), y se ha realizado una última iteración con parámetros:

- n estimators: [200]
- lr: [0.005, 0.01]
- sample size: [0.7, 0.8, 0.9]
- n neighbours: [3, 5]
- metric: manhattan

Y da resultados:

TABLA V
RESULTADOSDEL ALGORITMO CON DISTINTAS CONFIGURACIONES

n_estimators	lr	sample_size	n	metric	R^2 medio
200	0.010	0.70	5	manhattan	0.5763
200	0.010	0.70	3	manhattan	0.5755
200	0.010	0.80	5	manhattan	0.5714
200	0.010	0.90	5	manhattan	0.5646
200	0.010	0.80	3	manhattan	0.5612
200	0.005	0.90	3	manhattan	0.5369
200	0.005	0.70	3	manhattan	0.5237
200	0.005	0.80	3	manhattan	0.5237

Aparte de esto, se ha realizado un experimento para ver cómo interactúa el ensamble con early-stopping, viendo cuántas iteraciones se realizan por cada fold(5 folds del dataset Home Prices) y si en algunos de estos se aplica. Para esto se utiliza un ensamble con hiperparámetros:

- base estimator: KNeighborsRegressor(n-neighbors=7)
- n estimators: 500
- sample size: 0.8
- lr: 0.005
- random state: 42
- early stopping: True
- patience: 10

- R^2 medio House Prices: 0.117
- Iteraciones por fold: [33, 23, 24, 19, 19]

2) Resultados: Primero, analizaremos los datasets por separado:

a) House Prices:

- Los resultados son casi iguales a los del KNN único, por lo que un ensamble para este dataset no merecería la pena
- El learning rate más óptimo es 0.010, un valor bajo, pero que asegura un buen aprendizaje cuando hay un gran número de estimadores(n estimators).
- La distancia manhattan es la más óptima en todos los casos
- El early stopping actua muy rapidamente, ya que al tener un learning rate muy pequeño, los cambios son mínimos, pero un learning rate más grande afecta negativamente al ensamble.

b) Parkinson:

- La mejora de la predicción es mejor que en el deataset anterior comaprada con el modelo base, pero sigue sin ser suficiente como para respaldar el uso del ensamble
- Con respecto a los hiperparámetros, claramente se ve una diferencia entre los n-neighbours, siendo 5 el más óptimo, y en el learning rate, aunque esto puede ser porque los estimadores son bajos

V. Conclusiones

Este trabajo ha permitido implementar desde cero un algoritmo de ensamble secuencial de modelos para tareas de regresión, explorando su funcionamiento interno, sus ventajas, limitaciones y el impacto de sus hiperparámetros clave. El meta-modelo desarrollado ha sido evaluado con éxito sobre dos conjuntos de datos reales, precios de viviendas y enfermedad de Parkinson, utilizando como modelos base tanto ‘DecisionTreeRegressor‘ como ‘KNeighborsRegressor‘.

Los resultados confirman que los ensambles secuenciales pueden mejorar significativamente el rendimiento predictivo frente a modelos individuales, siempre que se escojan adecuadamente tanto el tipo de estimador como sus parámetros. En este sentido, se observó que el algoritmo basado en árboles de decisión (DecisionTreeRegressor) ofreció un rendimiento más robusto, generalizable y menos sensible a los ajustes, especialmente sobre el dataset de precios de Parkinson. Por el contrario, el ensamble construido con KNeighborsRegressor mostró limitaciones más evidentes, requiriendo normalización previa, parámetros muy específicos y una alta cantidad de iteraciones con tasas de aprendizaje muy bajas para obtener mejoras marginales. En particular, los resultados sugieren que kNN no se adapta bien a la lógica de los ensambles secuenciales, ya que su naturaleza no paramétrica dificulta la corrección progresiva del error residual.

Además, el comportamiento del meta-modelo resultó claramente dependiente del conjunto de datos utilizado: mientras el modelo basado en árboles obtuvo mejores resultados en el dataset de Parkinson, el kNN logró un rendimiento más competitivo, aunque aún por debajo del árbol, en el dataset de House Prices, posiblemente por las características del espacio vectorial y la distribución de las instancias.

Finalmente, el uso de técnicas como la validación cruzada y la parada temprana (early stopping) han servido para asegurar tanto la validez de los resultados como la eficiencia del entrenamiento, ayudando a evitar el sobreajuste. En conjunto, este trabajo ha permitido consolidar el conocimiento sobre algoritmos de boosting aplicados a regresión y ofrece una base sólida para extender este enfoque hacia problemas más complejos.

Como posibles líneas de mejora, una primera opción interesante sería introducir un ajuste automático de los parámetros del modelo, para evitar probar combinaciones manualmente y así ahorrar tiempo. También se podría experimentar con modelos base distintos dentro del mismo ensamble, combinando por ejemplo árboles y regresores lineales para aprovechar distintas formas de aprender del conjunto de datos. Otra mejora valiosa sería incluir más visualizaciones del proceso de aprendizaje, como cómo evoluciona el error a lo largo de las iteraciones, ya que esto facilitaría detectar cuándo el modelo empieza a sobreajustarse. Por último, se podría estudiar cuáles son las variables más importantes en las predicciones del modelo, lo que no solo ayudaría a interpretarlo mejor, sino también a simplificarlo eliminando atributos poco útiles.
trabajo futuro.

References

[1] CFI Team. *Ensemble Methods*. Disponible vía web https://corporatefinanceinstitute.com/resources/data-science/ensemble-methods/. 2024.

[2] Gurpreet Singh. *A complete Review On Ensemble Modeling*. Disponible vía web https://www.debutinfotech.com/blog/ensemble-modeling. 2024.

[3] Juan C Olamendi. *Early Stopping In Deep Learning*. Disponible vía web en https://medium.com/@juanc.olamendy/real-world-ml-early-stopping-in-deep-learning-a-comprehensive-guide-fabb1e69f8cc. 2024.

Nota de autoría asistida

Parte del contenido de este documento ha sido generado o asistido mediante el uso de la herramienta ChatGPT, desarrollada por OpenAI, con fines de redacción, corrección y apoyo técnico.