Aprendizaje Acumulativo: Un Enfoque Secuencial para Modelos de Regresión

Emilio Manuel Vázquez Cruz
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
correo: emivazcru@alum.us.es, UVUS: HRY9017

Ignacio Gutiérrez Serrera
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
correo: igngutser@alum.us.es, UVUS: NHD9927

Este trabajo tiene como objetivo realizar una implementación en python al problema de crear un ensamble de modelos de aprendizaje automático supervisado para la resolución de tareas de regresión, esto incluye el entrenamiento de dichos modelos y la evaluación de sus predicciones.

Para ello, se ha utilizado una técnica similar a GradientBoosting (potenciación del gradiente) descrita en el pseudocódigo del documento introductorio[1], evaluando los modelos mediante validación cruzada con k pliegues, como se exigía en la propuesta. Además, se realiza una experimentación para, a partir de los conjuntos de datos dados, obtener el mejor valor de la métrica R^2 en base al valor de los hiperparámetros, obtenido mediante una búsqueda en rejilla.

A partir del análisis realizado, se ha determinado que los árboles de decisión CART muestran un mejor desempeño en ensambles secuenciales, ya que su precisión aumenta con más datos y mayor cantidad de iteraciones. En contraste, kNN no logra mejoras significativas en cada iteración, lo que indica que su potencial para ensamblados es más limitado. También se ha identificado la necesidad de considerar funciones de error más flexibles, explorar ensambles exclusivos para redes neuronales con retropropagación y adaptar estos enfoques para algoritmos de aprendizaje por refuerzo. Este estudio abre la puerta a futuras investigaciones en optimización de ensambles, adaptaciones a conjuntos de datos más complejos y mejoras en la eficiencia computacional.

Palabras Clave—Inteligencia Artificial (Artificial Intelligence), Potenciación del Gradiente (Gradient Boosting), Ensamble secuencial (Sequential Ensemble), regresión (regression), Árboles de decisión (Decission Trees), K vecinos más cercanos (k nearest neighbors, kNN), Parada temprana (early stopping), scikit-learn, numpy, pandas.

I. INTRODUCCIÓN

Este trabajo se realiza como proyecto práctico de los alumnos matriculados en la asignatura obligatoria de 6 créditos ETC Inteligencia Artificial del tercer curso del grado universitario de la Universidad de Sevilla Ingeniería informática – Ingeniería del software que han seleccionado la propuesta de ensamble secuencial de modelos. En la asignatura se ha proporcionado un conocimiento introductorio sobre diversos temas, siendo, en orden, aprendizaje automático supervisado, redes neuronales, procesamiento del lenguaje natural, planificación automática y aprendizaje por refuerzo, enmarcándose este trabajo en el primero de ellos, aunque con posibilidad de tener aplicación para el segundo y tercer tema.

El tema que sirve principalmente de base para este trabajo, aprendizaje automático supervisado[5], consiste en obtener predicciones sobre datos específicos a partir de un modelo resultante de entrenar un algoritmo de aprendizaje automático sobre un conjunto de ese tipo de datos, es decir, de su dominio, que puede entenderse como un conjunto de variables de entrada (que serían los atributos de la entidad con la que se trabaja) que dan resultado a una variable objetivo. Dichas predicciones pueden ser de valores continuos, conllevando que a que sean tareas de regresión o valores discretos, siendo clasificación multiclase si hay más de dos clases a predecir o clasificación binaria si la decisión es booleana.

En concreto, en clase se han visto tres algoritmos de aprendizaje automático supervisado, siendo estos:

- Naïve Bayes, que se sirve de cálculos probabilísticos para su funcionamiento, obviando las interconexiones entre las variables de entrada del problema.
- K nearest neighbors (kNN), que predice los resultados en función de la distancia a los k ejemplos más cercano contenidos en el fichero de datos al ejemplo a predecir.
- Árboles de decisión, objeto principal del estudio de este trabajo. El algoritmo tiene una complejidad teórica logarítmica, al seguir la estrategia divide y vencerás (Divide Et Impera) para construir un árbol, ya que calcula valores umbrales para cada atributo o variable de entrada, con lo que divide el conjunto de datos inicial, lo que usa para calcular purezas para cada valor umbral y, selecciona el umbral con mayor pureza para que sea el padre de nuevas hojas obtenidas de igual manera.

Por último, en la asignatura hemos estudiado métodos de evaluación de la precisión de las predicciones de un modelo para tareas tanto de clasificación y de regresión. Para tareas de clasificación, hemos estudiado la matriz de confusión, de la que derivan varias métricas. Para tareas de regresión como la que nos ocupa hemos estudiado varias métricas estadísticas, como

el error cuadrático medio (MSE), el error medio absoluto (MAE), la raíz cuadrada del error cuadrático medio (RMSE), que traduce la magnitud para que sea interpretable. Por último, estudiamos el coeficiente de determinación (R^2), métrica en la que centraremos nuestros análisis del ensamble secuencial de modelos.

En cuanto al ensamble de modelos, este puede hacerse secuencial o paralelamente, siendo el primero el objetivo a implementar en este trabajo. Antes de explicar ambos ensambles brevemente, cabe destacar que se puede obtener un buen ensamble a partir de modelos entrenados con diferentes algoritmos sobre el mismo conjunto de datos o a partir de modelos entrenados con un mismo algoritmo sobre variaciones (aleatorias) del conjunto de datos inicial. El segundo es el procedimiento seguido en este trabajo. Esto se realiza debido a que algunos algoritmos de aprendizaje automático supervisado no obtienen buenos resultados por sí solos y este procedimiento permite combinar la predicción de muchos modelos entrenados con ese algoritmo, es decir, el ensamble de varios modelos débiles da lugar a un meta-modelo más preciso.

Procediendo a la explicación del ensamble, cabe explicar brevemente el ensamble paralelo, que consiste en entrenar múltiples instancias del mismo modelo con diferentes subconjuntos de datos y/o distintos hiperparámetros. Luego, los resultados se combinan, generalmente mediante una media de las predicciones de cada uno.

En el caso del ensamble secuencial, el propuesto, el resultado se obtiene haciendo que cada modelo entrene con el gradiente del error obtenido por el anterior, corrigiendo así algunos de sus errores, y así sucesivamente hasta llegar a la última iteración, tratándose por tanto de un proceso de minimización del gradiente del error en cada paso. En este trabajo se ha implementado la mejora opcional de la parada temprana, consistente en evaluar el meta-modelo parcial en la iteración actual con los datos no utilizados para entrenamiento en esa iteración para considerar si esa iteración ha aportado una mejora significativa. De no ser así un número determinado de veces el algoritmo parará antes de realizar todas las iteraciones especificadas.

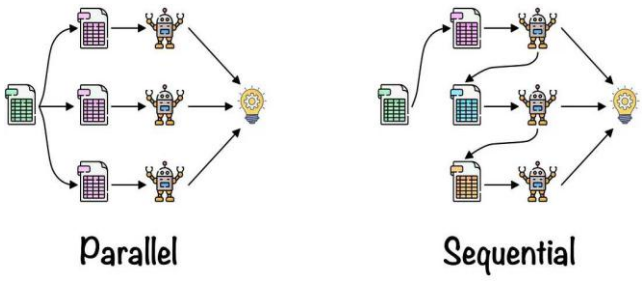

Ilustración 1 – Ejemplo gráfico de ensamble secuencial y paralelo

En lo que a la estructura del documento concierne, está dividido en:

- Preliminares: se introducen brevemente las técnicas utilizadas y los trabajos relacionados.
- Metodología: se describe el enfoque seguido y su implementación, es decir, lo que realmente se ha hecho.
- Resultados: se detallan tanto los experimentos realizados como los resultados conseguidos.
- Conclusiones: donde se interpretan los resultados y se proponen líneas futuras de trabajo, así como un resumen global del mismo.

II. PRELIMINARES

A. *Métodos empleados*

En este trabajo se ha empleado un conjunto de técnicas correspondientes al ámbito del aprendizaje automático supervisado, concretamente centrado en problemas de regresión. Las principales técnicas y métodos utilizados han sido los siguientes:

- Árboles de decisión (Decission Tree): técnica de clasificación y regresión basada en la división del espacio de entrada mediante condiciones sobre los atributos. Su funcionamiento sigue una estrategia recursiva de divide y vencerás, lo que permite modelar relaciones no lineales entre variables.
- Potenciación del gradiente (Gradient Boosting): técnica de ensamble secuencial que permite combinar múltiples modelos débiles (en este caso, árboles de decisión y kNN) entrenados secuencialmente. Cada modelo nuevo se entrena con el error residual del modelo anterior, minimizando así el error general paso a paso.
- Parada temprana (early stopping)**:** técnica que permite detener el proceso de entrenamiento del ensamble secuencial cuando se detecta que no se está produciendo una mejora significativa en el conjunto de validación, lo cual ayuda a prevenir el sobreajuste.
- Validación cruzada (cross validation) mediante el coeficiente de determinación (R^2): métrica que cuantifica la proporción de la varianza explicada por el modelo respecto a la varianza total, facilitando la comparación objetiva entre diferentes enfoques y configuraciones

B. *Trabajo Relacionado*

Se puede realizar un recorrido en la literatura sobre trabajos anteriores que estén relacionados y que sea por tanto interesante comentar aquí. Por supuesto, añadir las referencias bibliográficas correspondientes.

El algoritmo de ensamble secuencial desarrollado en este trabajo puede considerarse una variante simplificada del Gradient Boosting, que ha sido ampliamente estudiado. Existen implementaciones bastante avanzadas de este proceso, como el GradientBoostingRegressor[6] y el HistGradientBoostingRegressor[7], ambos disponibles en la biblioteca de Scikit-learn, que emplean versiones optimizadas del algoritmo de potenciación del gradiente.

Otros trabajos relevantes incluyen técnicas de ensamble con modelos de aprendizaje por refuerzo, como Q-learning, y enfoques de ensamble paralelo como el bagging (ej. Random Forest), que también combinan múltiples modelos para mejorar la precisión y robustez de las predicciones.

Así, el trabajo aquí presentado se sitúa dentro de un marco teórico consolidado, adoptando y adaptando ideas de la literatura existente para ofrecer una implementación accesible y didáctica de un ensamble secuencial con árboles de decisión, con fines experimentales y educativos.

III. METODOLOGÍA

El trabajo se empezó a partir del pseudocódigo presente en la propuesta[1]. Sin embargo, antes de realizar la propuesta, se ha tratado de asegurar la reproducibilidad de algunos de los ejemplos en el notebook de jupyter mediante el establecimiento de semillas para los métodos aleatorios de las librerías. Posteriormente, se obtuvo una primera versión funcional del ensamble, definido como una clase de Python con tres funciones, init, fit y predict, aunque no compatible con la validación cruzada implementada en scikit learn, cross_val_score[4], ya que el fit y el predict no tenían el formato que esperaba dicho método y la clase no heredaba ni de BaseEstimator[15] ni de RegressorMixin[3].

Sin embargo, se hicieron algunas pruebas midiendo el coeficiente de determinación sin utilizar validación cruzada, simplemente llamando al ensamble con diferentes valores de los hiperparámetros, para observar que tenía un correcto funcionamiento y se probaron, además DecisionTreeRegressor[14], varios regresores de scikit learn, específicamente los siguientes: GradientBoostingRegressor[6], HistGradientBoostingRegressor[7], kNeighborsRegressor[8], LinearRegression[9], BayesianRidge[10], GaussianProcessRegressor [11] y KerasRegressor [12].

Luego, se corrigieron estos defectos para que el ensamble fuese totalmente compatible con scikit learn, es decir, la clase se inicializaba con la variable objetivo, el fit recibía el conjunto de entrenamiento completo y el predict solo el conjunto de entrenamiento, sin la variable objetivo y se empezó a realizar pruebas con validación cruzada utilizando cross_val_score. Primero se hicieron todas las pruebas con el csv house_prices, debido a su brevedad, lo que conllevaba una ejecución más rápida de la evaluación cruzada del meta-modelo.

Todo esto sirvió para clarificar la parte programática del trabajo, es decir, los parámetros que debía recibir la clase, refactorizaciones (funcionalidad extraíble para un código más claro), mejoras de eficiencia…

Finalmente, se decidió que ejecutar cada celda muchas veces cambiando los valores a mano era poco eficiente y en contra de un principio básico de la informática, ahorrar tiempo, así que se optó por utilizar el GridSearchCV[13] para, a partir de una rejilla de hiperparámetros, entre los que se encuentran n_estimators, lr (learning rate), sample size e hiperparámetros de los algoritmos a ensamblar, en este caso han sido max_depth para los árboles de decisión y para kNN n_neighbors, metric (forma de calcular la distancia, se ha probado manhattan, euclidean y minkowski), tras largas ejecuciones con ambos conjuntos de prueba (house_prices y parkinsons) encontrar la combinación más óptima de estos hiperparámetros, siempre con cinco (cuatro para kNN) divisiones, o iteraciones de la validación cruzada.

Durante todo el proceso se fue perfeccionando la documentación a nivel de código de los métodos desarrollados, la más importante, la del ensamble secuencial, acorde con su desarrollo.

En cuanto a la metodología de trabajo, se ha usado git como sistema de control de versiones distribuido y se ha trabajado por una rama por aspecto del trabajo. El trabajo ha sido desarrollado mayoritariamente por ambos miembros conjuntamente, para poder así tener el mismo conocimiento de la funcionalidad desarrollada, las pruebas realizadas y agilizar así tanto la implementación, como la detección y corrección de errores que iban apareciendo.

A continuación, antes de observar el pseudocódigo de la funcionalidad implementada se listan una serie de decisiones de diseño tomadas que facilitarán la comprensión del mismo. Algunas de ellas son provenientes del documento de la propuesta de este trabajo[1].

DECISIONES DE DISEÑO:

1. Tipo de Datos y Almacenamiento:

- La lectura de los datos ha sido implementada como un método auxiliar válido para cualquier csv con variable objetivo en la última columna, el cual se encarga de codificar los tributos discretos nominales. Además, se encarga de la normalización en caso de que sea necesaria.
- El formato de los datos es csv y se almacena (en memoria) en un objeto de tipo DataFrame de Pandas.

2. Capacidades del Meta-Modelo:

- El meta-modelo puede aceptar tanto tareas de regresión como de clasificación.

- 3. Inicialización y Entrenamiento:
- Se inicializa la predicción inicial como la media de la variable objetivo, ya que, en base a los experimentos realizados, se ha notado que eso garantiza un mejor resultado en menos iteraciones que inicializarla a 0 en el mayor de los casos.
- Cada modelo entrena con un subconjunto de datos aleatorio.

4. Evaluación y Validación:

- Se usa validación cruzada para la evaluación del ensamble.

- La métrica usada tanto para la parada temprana como para la validación cruzada es el coeficiente de determinación, R^2.
- Parada temprana (early stopping): Tras varias pruebas con ambos conjuntos de datos se ha definido un valor umbral epsilon para considerar si la mejora de la predicción del modelo actual respecto del anterior es útil. Esta comparación entre iteraciones consecutivas se realiza con los datos de prueba no utilizados en el entrenamiento de cada modelo y los coeficientes de determinación obtenidos. Se tiene que trabajar con valores bastante elevados del umbral y con una paciencia baja, es decir, son necesarias pocas ocasiones en las que se supere al umbral para que la paciencia se agote y el entrenamiento finalice para que llegue a notarse el efecto de la parada.

5. Compatibilidad con Modelos:

- El ensamble soporta regresores de redes neuronales.

6. Integración con Scikit-Learn:

- El ensamble hereda de BaseEstimator y RegressorMixin para poder utilizar diversos métodos de scikit-learn que se salen del alcance de este trabajo.

Procedimiento Ensamble secuencial (GradientBoosting)
Entrada:
- Array numpy de datos
- Array numpy de variable objetivo
- Algoritmo base de entrenamiento
- Número de estimadores
- Tasa de aprendizaje (lr)
- Tamaño de la muestra (sample_size)
- Paciencia (para la parada temprana)
- Épsilon (para la parada temprana)
- Tipo de la tarea (regresión o clasificación)

Salidas:
- Predicción sobre los datos de prueba del ensamble de modelos entrenado con los datos de entrenamiento.

Algoritmo:
Entrenamiento:
1. Inicializar la primera predicción con pred0
2. pred_actual = pred0
3. Por cada i en n_estimators:
    1. $residuo_i$ = y – pred_actual
    2. entrenar $estimador_i$ con un subconjunto aleatorio de los datos marcado por sample_size y usando $residuo_i$ como variable objetivo.
    3. obtener las predicciones, $pred_i$, de $estimador_i$
    4. pred_actual = pred_actual + $pred_i$*lr
    5. Calcular el R^2 de la predicción del meta-modelo hasta la iteración actual y comprobar si la diferencia con el R^2 de la predicción anterior es menor que épsilon:
        1. Si no: no hacer nada
        2. Si: decrementar en 1 la paciencia
    6. si paciencia = 0 entonces finalizar el bucle de cálculo de estimadores.
4. Devolver el conjunto de modelos entrenados

Predicción:
Inicializar la primera predicción como la media de la variable objetivo.
6. pred_actual = media(array variable objetivo)
7. Por cada $modelo_j$ en modelos entrenados:
    1. pred_actual += lr*modelo.predict(datos)
8. Devolver el conjunto con la predicción para cada elemento.

Pseudocódigo 1. Algoritmo de GradientBoosting modificado para la asignatura

IV. RESULTADOS

Para la experimentación y análisis de la posible combinatoria resultante de los hiperparámetros definidos, se han probado los csv house_prices y parkinsons tal y como estaban colgados en la página de la asignatura sin realizar ninguna modificación sobre ellos, tal como se pedía. Tras aplicar, para cada uno de ellos, un grid search con el ensamble para los modelos base árboles de decisión CART y los k vecinos más cercanos y la rejilla de hiperparámetros definidos se han obtenido los resultados para la métrica R^2 de aplicar validación cruzada para cada combinación de hiperparamétros. Para árboles de decisión se han realizado cinco iteraciones o divisiones en la validación cruzada y en kNN cuatro, debido a su mayor tiempo de ejecución.

Para parkinsons, se ha utilizado la siguiente configuración para cada algoritmo:

- Árboles de decisión:
  Tasa de aprendizaje (learning rate) en el rango (0.01, 0.21), con un paso de 0.02.
  Número de estimadores o iteraciones (n_estimators) comprendido en el rango (50, 301), con un paso definido de 25 unidades.
  Tamaño de la muestra para el entrenamiento (sample_size) se encuentra en el rango (0.75, 0.91), con un paso de 0.075).
  La máxima profundidad del árbol (max_depth) está contenida en el rango (3,7), avanzando de unidad en unidad.
- K vecinos más cercanos:
    - Tasa de aprendizaje (learning rate) en el rango (0.01, 0.21), con un paso de 0.04.
    - Número de estimadores o iteraciones (n_estimators) comprendido en el rango (100, 301), con un paso definido de 50 unidades.
    - Tamaño de la muestra para el entrenamiento (sample_size) se encuentra en el rango (0.75, 0.92), con un paso de 0.04).
    - El número de vecinos a tomar en cuenta en la comparación (n_neighbors) está contenida en el rango (1,5), avanzando de unidad en unidad.

Cabe destacar que no se han probado tantos valores como habría sido deseable debido al elevadísimo tiempo de ejecución para los portátiles poseídos, llegando a superar las 10 horas en una ocasión. De haber tenido menores tiempos de ejecución, se habría disminuido los pasos para los rangos definidos de cada hiperparámetro, habiendo realizado así una mayor exploración, conllevando un análisis más completo.

A pesar de tener muchos más valores de prueba, en estas tablas se han puesto los diez mejores resultados para cada uno.

En esta sección se detallará tanto los experimentos realizados como los resultados conseguidos:

- Los experimentos realizados, indicando razonadamente la configuración empleada, qué se quiere determinar, y como se ha medido.
- Los resultados obtenidos en cada experimento, explicando en cada caso lo que se ha conseguido.
- Análisis de los resultados, haciendo comparativas y obteniendo conclusiones.

Tabla 1. Resultados con árbol de decisión para house_prices

lr	*n_estimators*	*sample_size*	*max_depth*	R^2
0.15	100	0.9	5	0.76589
0.13	250	0.9	5	0.76016
0.13	275	0.9	4	0.75598
0.15	175	0.9	4	0.75577
0.09	175	0.9	4	0.75441
0.15	150	0.9	4	0.75259
0.19	50	0.9	4	0.75149
0.15	175	0.9	3	0.75053
0.07	300	0.9	4	0.74997
0.07	200	0.9	4	0.74953

Tabla 2. Resultados con árbol de decisión para parkinsons.csv

lr	*n_estimators*	*sample_size*	*max_depth*	R^2
0.09	250	0.9	6	0.92504
0.09	200	0.9	6	0.92319
0.09	300	0.9	6	0.92316
0.05	225	0.9	6	0.92294
0.11	175	0.9	6	0.92292
0.07	200	0.9	6	0.922752
0.09	175	0.9	6	0.92233
0.05	250	0.9	6	0.922112
0.07	225	0.9	6	0.92193
0.13	250	0.9	6	0.92187

Tabla 3. Resultados con kNN para house_prices.csv

lr	*estimators*	*sample_size*	*metric*	*n_neighbors*	R^2
0,01	200	0,75	mh	2	0,72721
0,01	200	0,83	mh	3	0,72296
0,01	200	0,75	mh	3	0,72223
0,01	150	0,75	mh	2	0,72204
0,01	200	0,75	mh	5	0,71920
0,01	200	0,83	mh	2	0,71823
0,01	150	0,75	mh	3	0,71346
0,01	200	0,91	mh	3	0,71289
0,01	200	0,75	mh	4	0,71188
0,01	150	0,83	mh	2	0,71021

mh representa la métrica distancia manhattan

Tabla 4. Resultados con kNN para parkinsons.csv

lr	*n_estimators*	*sample_size*	*metric*	*neighbors*	R^2
0.01	150	0.75	mh	3	0.55590
0.01	150	0.79	mh	3	0.55009
0.01	150	0.83	mh	3	0.54087
0.01	200	0.75	mh	3	0.53639
0.01	150	0.75	mh	2	0.53363
0.01	150	0.87	mh	3	0.53360
0.01	200	0.79	mh	3	0.52941
0.01	150	0.79	mh	2	0.52466
0.01	150	0.91	mh	3	0.52307
0.01	100	0.75	mh	3	0.52139

Procederemos a la comparación de los dos modelos para los dos csv (es decir, comparar, para el mismo algoritmo base su rendimiento y R^2 entre house_prices y parkinsons) y entre ambos algoritmos para cada csv por separado. Antes de comenzar, cabe mencionar que los resultados de R^2 han sido truncados a cinco decimales para un correcto encaje en el formato dado para el documento.

Comparación de árboles de decisión para ambos conjuntos de datos:

- Parkinsons: se puede observar que los mejores coeficientes de determinación están en torno a 0.92, todos ellos sirviéndose del mayor sample size que podían coger y de la mayor profundidad, debido a que los árboles tienen una relación entre el número de datos con el que entrenan (y los atributos de esto) y la profundidad. Por tanto, podemos observar que no se ha producido memorización u overfitting en los datos (ya que es el objetivo de la validación cruzada). Por este motivo (y por tiempo de ejecución) se ha limitado la profundidad de los árboles, para que no memorizase los datos. Sin embargo, sorprendentemente, se puede constatar que no necesita el máximo número de estimadores para un rendimiento óptimo, ya que el máximo solo aparece una vez, llegando a aparecer un valor (175) que es casi la mitad de este. Es cierto que esto puede haber sido debido a que en las 300 iteraciones se haya aplicado la parada temprana. Para la tasa de aprendizaje puede observarse que requiere de valores nada bajos, llegando a tener uno de la tabla 0.13, que para lo visto en las prácticas de la asignatura es un valor relativamente alto.

- House_prices: se puede observar que con menos datos de prueba (unas 3.6 veces inferior a parkinsons) los resultados quedan bastante heterogéneos, oscilando los mejores valores de R^2 entre 0.76589 y 0.74953, aunque con más variabilidad que el anterior en sus parámetros. Se observa que entre en los diez mejores hay uno que tiene 50 estimadores y otro 300, oscilando el resto entre 100 y 200, por lo que no son tan determinantes a la hora de la precisión. Además, la tasa de aprendizaje es relativamente alta, ya que está entre 0.07 y 0.19. También se constata que los árboles CART se nutren de unos amplios datos, ya que todas las columnas de los diez mejores tienen sample_size de 0.9, que era el máximo valor que se le proporcionó en la exploración. Puede observarse que la profundidad máxima obtiene valores 4, 5 y un 3, lo que indica que generalmente tiende a mejores resultados con una mayor profundidad, aunque tenía un valor máximo de 6 y no lo ha seleccionado.

Antes de empezar la comparación de k vecinos más cercanos para ambos conjuntos de datos cabe destacar que se beneficia clarísimamente de una tasa de aprendizaje muy baja (los veinte valores mostrados utilizan la menor posible), seguramente debido a que el primer estimador ya es bastante bueno y no se nutre tanto de la aportación de los siguiente, lo que concuerda con el valor máximo de los estimadores, ya que en ningún caso sobrepasa los 200, siendo el máximo 300. También puede observarse que la métrica predominante es manhattan y que el sample size no es determinista, ya que hay valores en todo el rango que se definió en la búsqueda en rejilla:

- Parkinsons: puede observarse que el máximo coeficiente de determinación es de 0.55590 y que se obtiene para el menor sample size posible. También es destacable que se obtiene para 150 estimadores, la mitad del valor máximo para este hiperparámetro, y para tres vecinos, siendo el máximo 4. Cabe destacar que en este conjunto de datos no se usa el máximo valor del número de vecinos, siendo este cuatro.

- House_prices: puede observarse que para este conjunto de datos el máximo coeficiente de determinación es de 0,72721 y que el número de vecinos no es importante, ya que tiene una variabilidad completa dentro de su rango.

Esta diferencia tan notoria entre ambos conjuntos de datos puede deberse al hecho de que, para casas con atributos similares, sus precios son parecidos, pero para personas con parkinson estos atributos de la voz no tienen por qué ser tan parecidos.

Entrando en la comparativa entre ambos algoritmos base para este ensamble secuencial de modelos, se puede comprobar que los árboles de decisión CART tienen un claro dominio, con un poco más de exactitud en el conjunto de datos house_prices y una mucho mejor precisión en el conjunto de datos parkinsons, lo que indica que es mucho más susceptible a beneficiarse de ser ensamblado que el algoritmo de los k vecinos más cercanos al ser este un mayor dataset. De hecho, puede deducirse que a mayor tamaño tenga el conjunto de datos de entrenamiento mayor será la calificación de los árboles CART respecto a kNN.

V. Conclusiones

Finalmente, se dedica la última sección para indicar las conclusiones obtenidas del trabajo. Se puede dedicar un párrafo para realizar un resumen sucinto del trabajo, con los experimentos y resultados. Seguidamente, uno o dos párrafos con conclusiones. Se suele dedicar un párrafo final con ideas de mejora y trabajo futuro.

En este documento se ha atestiguado un profundo análisis del ensamble secuencial de modelos de aprendizaje automático supervisado, técnica mediante la cual se entrenan modelos con un algoritmo a elegir y van a corrigiendo los errores de sus predecesores sirviéndose de la minimización del gradiente (derivada) del error cometido por estos. En adición, el algoritmo deja de ejecutarse si la mejora de una cantidad definida de modelos es considerada infructuosa.

Recapitulando el análisis de varios algoritmos, puede verse que los árboles CART predicen mejor si se nutren de más datos y se le permite realizar más iteraciones del ensamble, además de tasas de aprendizaje altas, lo que no se cumple para kNN, que no usa todas las iteraciones, lo que puede deberse a la parada temprana, aunque esto no hace más que reforzar la idea de que no es tan adecuado para ensambles, al no mejorar en cada iteración lo suficiente como para ser considerada una mejor útil. Además, utiliza la tasa de aprendizaje más baja siempre y predice mejor para el menor conjunto de datos, lo que lleva a pensar que estos dos algoritmos se comportan de forma opuesta al ser ensamblados.

Posteriormente al detallado análisis del problema puede determinarse que los árboles de decisión CART responden mejor a ser ensamblados que kNN al tener mayor potencial, sobre todo con mayores conjuntos de datos, que son los utilizados en el mundo real. Cierto es que requieren de más iteraciones que kNN para obtener sus mejores resultados, al contrario que kNN, pero precisamente por eso su potencial (al menos en los conjuntos de datos dados) es menor que el de los árboles CART, tarda más en entrenarse y da peores resultados, además de requerir más preprocesamiento (al necesitar datos normalizados para evitar degeneraciones en sus cálculos).

Ideas de mejora y trabajo futuro:

- Permitir que el ensamble acepte cualquier función de error de las que hay de estándar en el campo de la inteligencia artificial.

- Realizar un ensamble secuencial exclusivo para redes neuronales que soporte retropropagación para corregir el error de las redes neuronales completas anteriores en vez del de las capas anteriores en cada paso.
- Realizar un ensamble secuencial para algoritmos de aprendizaje por refuerzo, como q-learning.

VI. Autodelcaración del uso de IA generativa

En este proyecto la IA, en concreto ChatGPT de OpenAI[2], se ha usado exclusivamente para corrección de errores en tiempo de ejecución y para la ayuda puntual de alguna interpretación o la mejora del código existente (más legible y/u optimizado).

Referencias

[1] Documento PDF de la propuesta del trabajo, Ensamblaje secuencial de modelos predictivos, Juán Galan Paez.

[2] Versión web de ChatGPT. https://chatgpt.com/.

[3] Página web de scikit learn, documentación de la API, sección de RegressorMixin. https://scikit-learn.org/stable/modules/generated/sklearn.base.RegressorMixin.html .

[4] Página web de scikit learn, documentación de la API, sección de cross_val_score. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html.

[5] Página web de la asignatura de Inteligencia Artifical, US curso 2024-25, temario de Aprendizaje automático y su respectiva práctica. https://www.cs.us.es.

[6] Página web de scikit learn, documentación de la API, sección de GradientBoostingRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html

[7] Página web de scikit learn, documentación de la API, sección de HistGradientBoostingRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingRegressor.html

[8] Página web de scikit learn, documentación de la API, sección de kNeighborsRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

[9] Página web de scikit learn, documetación de la API, sección de LinearRegression. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

[10] Página web de scikit learn, documentación de la API, sección de BayesianRidge. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.BayesianRidge.html

[11] Página web de scikit learn, documentación de la API, sección de GaussianProcessRegressor https://scikit-learn.org/stable/modules/generated/sklearn.gaussian_process.GaussianProcessRegressor.html

[12] Página web de scikit learn, documentación de la API, sección de KerasRegressor https://keras.io/api/utils/sklearn_wrappers/

[13] Página web de scikit learn, documentación de la API, sección de GridSearchCV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[14] Página web de scikit learn, documentación de la API, sección de DecisionTreeRegressor. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html

[15] Página web de scikit learn, documentación de la API, sección de BaseEstimator. https://scikit-learn.org/stable/modules/generated/sklearn.base.BaseEstimator.html