Ensamble Secuencial de Modelos para predicción de precios de vivienda y enfermedad de Parkinson

Joaquín Borja León
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
joaborleo@alum.us.es

Héctor Guerra Prada
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
hecguepra@alum.us.es

***Resumen*—Este trabajo presenta la implementación desde cero de un meta-algoritmo de ensamble secuencial para tareas de regresión. Se utilizan dos estimadores débiles: `DecisionTreeRegressor` y `LinearRegression`. Se aplican ambos enfoques a dos conjuntos de datos: precios de viviendas (`house_prices.csv`) y datos clínicos de la enfermedad de Parkinson (`parkinsons.csv`). La búsqueda manual de hiperparámetros se realiza mediante validación cruzada, optimizando `n_estimators`, `sample_size`, `lr`, `max_depth` para árboles y `fit_intercept` para regresión lineal. También se incluye un mecanismo opcional de parada temprana (*early stopping*). Los resultados demuestran que `DecisionTreeRegressor` capta relaciones no lineales en Parkinson (R^2=0.9323) y `LinearRegression` funciona bien en precios de vivienda (R^2=0.7875).**

***Palabras clave*—Inteligencia Artificial, Ensamble Secuencial, Regresión, Boosting, Precios de Vivienda, Enfermedad de Parkinson.**

I. Introducción

Los métodos basados en ensambles han demostrado una capacidad predictiva superior a la de modelos individuales en múltiples dominios. En particular, los ensambles secuenciales construyen modelos débiles de forma iterativa, de modo que cada nuevo estimador corrige los errores residuales del conjunto previo. Este esquema es la base de algoritmos como AdaBoost y Gradient Boosting [1], [2].

El objetivo de este trabajo es implementar y estudiar un meta-algoritmo de ensamble secuencial para tareas de regresión, aplicándolo a dos conjuntos de datos con características muy distintas:

- Conjunto de datos de precios de viviendas (`house_prices.csv`): 560 registros con variables físicas y categóricas, prediciendo el precio de venta.
- Conjunto de datos de la enfermedad de Parkinson (`parkinsons.csv`): 2000 registros con datos clínicos y de señal de voz, prediciendo la puntuación total del UPDRS.

La métrica principal de evaluación es el coeficiente de determinación (R^2), complementado con el error absoluto medio (MAE). Los objetivos específicos son:

1) Implementar las fases de entrenamiento y predicción del ensamble secuencial sin usar librerías de boosting de alto nivel.
2) Comparar dos estimadores débiles: `DecisionTreeRegressor` y `LinearRegression`.
3) Realizar búsqueda manual de hiperparámetros (`n_estimators`, `sample_size`, `lr`, `max_depth` para árboles y `fit_intercept` para lineal) mediante validación cruzada.
4) Implementar un mecanismo opcional de *early stopping* basado en un subconjunto de validación interno.
5) Documentar todas las decisiones de diseño, exponer resultados reproducibles y extraer conclusiones.

La estructura del documento es la siguiente:

- Sección II: Preliminares.
- Sección III: Metodología.
- Sección IV: Resultados.
- Sección V: Conclusiones.
- Referencias bibliográficas.

II. Preliminares

A. Métodos empleados

En este trabajo se utiliza el enfoque de ensamble secuencial (boosting). Cada estimador débil se entrena sobre los residuos del conjunto acumulado anterior. La función de pérdida es el error cuadrático medio, de modo que el residuo en la iteración m es:

$$r_{im} = y_i - F_{m-1}(x_i). \tag{1}$$

Se consideran dos familias de estimadores débiles:

1) `DecisionTreeRegressor` (árbol de decisión) con profundidad máxima (`max_depth`) variable.
2) `LinearRegression` (regresión lineal).

B. Trabajo relacionado

Los ensambles de boosting (AdaBoost, Gradient Boosting, XGBoost) han sido ampliamente estudiados en literatura reciente [1]–[3]. A diferencia de nuestro enfoque, se suele utilizar implementaciones optimizadas de alto nivel. En este trabajo implementamos desde cero la lógica de boosting.

III. Metodología

En esta sección se describe el algoritmo de ensamble secuencial implementado. La implementación sigue el siguiente esquema:

Procedimiento EntrenarEnsambleConParadaTemprana
Entrada:
- X: matriz de entrenamiento (n_muestras × n_atributos)
- y: vector objetivo de entrenamiento (longitud n_muestras)
- n_estimators: número máximo de iteraciones (modelos débiles)
- sample_size: proporción de muestras a usar en cada iteración (0 < sample_size ⩽ 1)
- lr: tasa de aprendizaje (learning rate)
- seed: semilla para la generación aleatoria
- patience: número máximo de iteraciones sin mejora antes de detener
- max_depth (opcional): profundidad máxima para cada DecisionTree (si aplica)

Salidas:
- Modelos: lista de submodelos entrenados (hasta la última iteración antes de detener)

Algoritmo:
1. Inicializar Modelos ← ∅
2. init_pred ← media(y)
3. pred_actual ← vector de longitud n_muestras, cada posición = init_pred
4. rng ← InicializarGenerador(seed)
5. N ← número de filas en X
6. best_score ← −∞
7. best_iter ← 0
8. patience_counter ← 0
9. Para m = 1 hasta n_estimators:
   a. residuo ← y − pred_actual
   b. k ← ⌊sample_size × N⌋
   c. idx_train ← rng.choice(N, k, replace = False)
   d. X_sub ← X[idx_train]
   e. y_sub ← residuo[idx_train]
   f. modelo_m ← clonar(estimator_base)
      i. Si max_depth ≠ None y si el modelo_m admite parámetro max_depth: modelo_m.set_max_depth(max_depth = max_depth)
      ii. Si modelo_m admite parámetro random_state: modelo_m.set_params(random_state = seed)
   g. modelo_m.fit(X_sub, y_sub)
   h. Añadir modelo_m a Modelos
   i. pred_actual ← pred_actual + lr × modelo_m.predict(X)
   j. Si X_val y y_val están definidos entonces:
      i. pred_val ← vector de longitud filas(X_val), cada posición = init_pred
      ii. Para cada m_i en Modelos: pred_val ← pred_val + lr × m_i.predict(X_val)
      iii. score ← R^2 (y_val, pred_val)
      iv. Si score > best_score entonces: best_score ← score; best_iter ← m; patience_counter ← 0
   k. Sino:
      i. patience_counter ← patience_counter + 1
      ii. Si patience_counter ⩾ patience entonces: Modelos ← primeros (best_iter) elementos de Modelos; Romper el bucle
10. Devolver Modelos

Pseudocódigo 1. Procedimiento EntrenarEnsambleConParadaTemprana

Fig. 1.

Procedimiento EntrenarEnsambleSecuencial
Entrada:
- X: matriz de entrenamiento (n_muestras × n_atributos)
- y: vector objetivo de entrenamiento (longitud n_muestras)
- n_estimators: número de iteraciones (modelos débiles)
- sample_size: proporción de muestras para cada submodelo (0 < sample_size ⩽ 1)
- lr: tasa de aprendizaje (learning rate)
- seed: semilla para la generación aleatoria
- max_depth (opcional): profundidad máxima para cada DecisionTree (si aplica)

Salidas:
- Modelos: lista de submodelos entrenados (hasta la última iteración antes de detener)

Algoritmo:
1. Inicializar Modelos ← ∅
2. init_pred ← media(y)
3. pred_actual ← vector de longitud n_muestras, cada posición = init_pred
4. rng ← InicializarGenerador(seed)
5. N ← número de filas en X
6. Para m = 1 hasta n_estimators:
   a. residuo ← y − pred_actual
   b. k ← ⌊sample_size × N⌋
   c. idx_train ← rng.choice(N, k, replace = False)
   d. X_sub ← X[idx_train]
   e. y_sub ← residuo[idx_train]
   f. modelo_m ← clonar(estimator_base)
      i. Si max_depth ≠ None y si el modelo_m admite parámetro max_depth: modelo_m.set_max_depth(max_depth = max_depth)
      ii. Si modelo_m admite parámetro random_state: modelo_m.set_params(random_state = seed)
   g. modelo_m.fit(X_sub, y_sub)
   h. Añadir modelo_m a Modelos
   i. pred_actual ← pred_actual + lr × modelo_m.predict(X)
7. Devolver Modelos

Pseudocódigo 2. Procedimiento EntrenarEnsambleSecuencial

Fig. 2.

Modelos base: DecisionTreeRegressor, LinearRegression y Early Stopping:

a) DecisionTreeRegressor: El `DecisionTreeRegressor` es un modelo de aprendizaje supervisado que predice valores numéricos dividiendo el espacio de características en regiones, siguiendo una estructura jerárquica en forma de árbol. En cada nodo, se

Procedimiento PredecirEnsamble
Entrada:
- Modelos: lista de submodelos entrenados (finales)
- X: matriz de datos nuevos a predecir (n_muestras_test × n_atributos)
- lr: tasa de aprendizaje (misma que usada en fit)
- init_pred: valor inicial (media calculada fit)

Salidas:
- pred_final: vector de predicciones (longitud n_muestras_test)

Algoritmo:
1. pred_final ← vector de longitud filas(X), cada posición = init_pred
2. Para cada modelo_i en Modelos:
   a. pred_final ← pred_final + lr × modelo_i.predict(X)
3. Devolver pred_final

Pseudocódigo 3. Procedimiento PredecirEnsamble

Fig. 3.

elige la variable y el umbral que mejor separen los datos para minimizar el error de predicción. Es un modelo *no lineal*, capaz de capturar relaciones complejas entre variables, aunque puede sobreajustar si no se controla su profundidad.

b) LinearRegression: El modelo `LinearRegression` ajusta una función lineal a los datos, minimizando el error cuadrático medio entre las predicciones y los valores reales. Es un modelo *lineal y simple*, adecuado cuando hay una relación aproximadamente lineal entre las variables predictoras y la variable objetivo. No capta relaciones no lineales ni interacciones, pero tiene bajo riesgo de sobreajuste y es computacionalmente eficiente.

c) Early Stopping: Para implementar el mecanismo de *early stopping*, en cada iteración se reserva un 20% del conjunto de entrenamiento (subconjunto de validación). Se calcula el error en validación después de cada nuevo modelo y se detiene el entrenamiento si no hay mejora tras `patience` iteraciones consecutivas.

IV. Resultados

A. Ensamble con DecisionTreeRegressor

Configuración de partida: `DecisionTreeRegressor(max_depth=3)`, `n_estimators=50`, `sample_size=0.8`, `lr=0.1`, `random_state=42`. Validación cruzada 5-fold:

- **House Prices**: R^2 por fold = [0.72356832 0.75864956 0.76608286 0.78143207 0.77203406], media = 0.7603533739391801.
- **Parkinsons**: R^2 por fold = [0.63276246 0.66040138 0.63794864 0.65623765 0.6812172], media = 0.6537134652991631.

Primera búsqueda manual de hiperparámetros (Grid Search simple):

$$n\_estimators \in \{10, 50, 100\},$$
$$lr \in \{0.01, 0.1, 0.2\},$$
$$sample\_size \in \{0.6, 0.8, 1.0\},$$
$$max\_depth \in \{3, 5, 7\}.$$

Evaluación final en test:

- **House Prices (óptimo):** $n\_estimators = 150$, $lr = 0.3$, $sample\_size = 1.0$, $max\_depth = 3$.
  Test R^2 = 0.7191232968866706, MAE = 29074.149712968017.

TABLA I
TOP-10 HOUSE PRICES – PRIMERA BÚSQUEDA

$n\_estimators$	lr	$sample\_size$	max_depth	r^2_mean
100	0.2	0.8	3	0.790717
100	0.1	0.6	3	0.790011
100	0.2	1.0	3	0.788360

TABLA II
TOP-10 PARKINSONS – PRIMERA BÚSQUEDA

$n\_estimators$	lr	$sample\_size$	max_depth	r^2_mean
100	0.1	1.0	7	0.919854
100	0.2	1.0	7	0.916761
50	0.2	1.0	7	0.915149

- **Parkinsons (óptimo):** $n\_estimators = 100$, $lr = 0.1$, $sample\_size = 1.0$, $max\_depth = 7$.
  Test R² = 0.936545781009529, MAE = 1.615012580924182.

B. Ensamble con LinearRegression

Configuración inicial: `LinearRegression(fit_intercept=True)`, `n_estimators=50`, `sample_size=0.8`, `lr=0.1`, `random_state=42`. Validación cruzada 5-fold:

- **House Prices**: R² por fold = [0.60978187 0.77420332 0.65521267 0.82114321 0.44958941], media = 0.6619860969721221.
- **Parkinsons**: R² por fold = [0.20806173 0.2282059 0.12500756 0.12577325 0.1387287], media = 0.16515542749688544.

Primera búsqueda manual de hiperparámetros:

$$n\_estimators \in \{10, 50, 100\},$$
$$lr \in \{0.01, 0.1, 0.2\},$$
$$sample\_size \in \{0.6, 0.8, 1.0\},$$
$$fit\_intercept \in \{\text{True}, \text{False}\}.$$

TABLA III
TOP-5 HOUSE PRICES – PRIMERA BÚSQUEDA

$n\_estimators$	lr	$sample\_size$	$fit\_intercept$	r^2_mean
10	0.2	1.0	True	0.720497
10	0.2	0.8	True	0.715898
50	0.1	1.0	True	0.703330

TABLA IV
TOP-5 PARKINSONS – PRIMERA BÚSQUEDA

$n\_estimators$	lr	$sample\_size$	$fit\_intercept$	r^2_mean
10	0.2	1.0	True	0.166249
10	0.2	0.8	True	0.166206
50	0.2	0.8	True	0.166206

Evaluación final en test:

- **House Prices (lineal óptimo):** $n\_estimators = 10$, $lr = 0.2$, $sample\_size = 1.0$, $fit\_intercept = True$.
  Test R² = 0.766222639788844, MAE = 25257.28627191641.
- **Parkinsons (lineal óptimo):** $n\_estimators = 10$, $lr = 0.2$, $sample\_size = 1.0$, $fit\_intercept = True$.
  Test R² = 0.14895427840774145, MAE = 8.073492674590106.

C. Early Stopping (Parada Temprana)

- **House (LR + early stopping):** Test R² = 0.766222639788844, MAE = 25257.28627191643.
- **Parkinsons (LR + early stopping):** Test R² = 0.14887395829827899, MAE = 8.077265793598144.

V. CONCLUSIONES

- Este trabajo ha abordado la implementación desde cero de un meta-algoritmo de ensamble secuencial para tareas de regresión, aplicándolo a dos conjuntos de datos de naturaleza diferente: precios de viviendas y progresión de la enfermedad de Parkinson. Se han comparado dos tipos de modelos base (`DecisionTreeRegressor` y `LinearRegression`) y se ha evaluado su rendimiento mediante validación cruzada y métricas como el coeficiente de determinación (R^2) y el error absoluto medio (MAE). Asimismo, se ha incorporado un mecanismo de parada temprana para optimizar el proceso de entrenamiento.
- Los resultados muestran que el desempeño del meta-modelo depende en gran medida del tipo de estimador base utilizado. El modelo basado en árboles obtuvo un rendimiento excelente en el conjunto de Parkinson ($R^2 = 0.936545781009529$), gracias a su capacidad para capturar relaciones no lineales complejas. En cambio, su rendimiento fue inferior en el conjunto de precios de vivienda, donde alcanzó $R^2 = 0.678402654711951$, posiblemente debido al tamaño limitado del dataset, que restringe la capacidad de generalización del ensamble. Por otro lado, la regresión lineal se comportó mejor en este conjunto ($R^2 = 0.766222639788844$), mostrando mayor estabilidad con pocos datos y relaciones más simples. Sin embargo, en el conjunto de Parkinson, su desempeño fue muy bajo ($R^2 = 0.14895427840774145$), debido a que los valores no siguen una relación lineal clara con las variables predictoras. La búsqueda manual de hiperparámetros permitió identificar configuraciones óptimas en ambos casos.

REFERENCIAS

[1] Y. Freund and R. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an application to boosting," *Journal of Computer and System Sciences*, 1997.

[2] J. Friedman, T. Hastie, and R. Tibshirani, "The Elements of Statistical Learning," Springer, 2001.

[3] T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[4] Scikit-learn Developers, "LinearRegression," *https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html*.

[5] Scikit-learn Developers, "DecisionTreeRegressor," *https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html*.