Ensamble secuencial de modelos predictivos

Inteligencia Artificial (IS) 2024/25 – Propuesta de trabajo

Juan Galán Páez

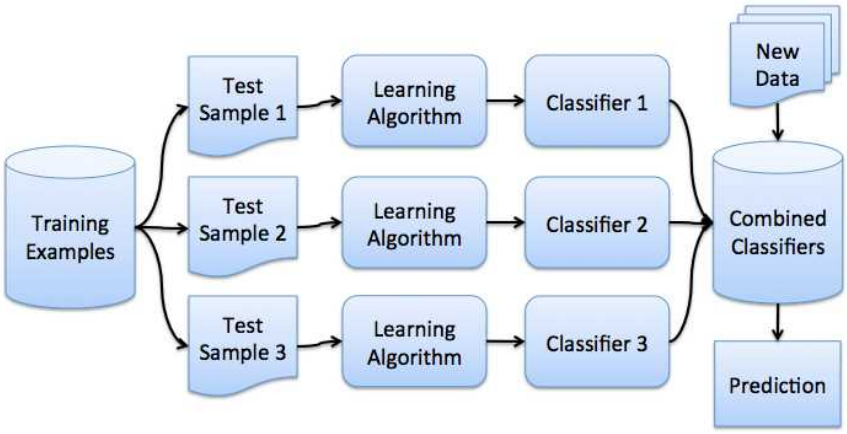
1. Introducción y objetivo

El presente trabajo se centra en el desarrollo de un algoritmo para construir modelos de clasificación y regresión basados en ensambles (combinación o composición) de otros modelos predictivos. En este trabajo, para construir el ensamble, los modelos se combinan secuencialmente, de forma que cada nuevo modelo que añadimos se centre en aprender los ejemplos en los que más error ha cometido el modelo anterior. Nuestro algoritmo de construcción del ensamble minimiza el error en cada iteración (añadiendo un nuevo modelo) mediante descenso por el gradiente. Una vez construido el algoritmo de ensamble de modelos, se aplicará sobre varios conjuntos de datos y se explorarán los principales hiperparámetros del algoritmo y cómo afectan sus valores al comportamiento del mismo.

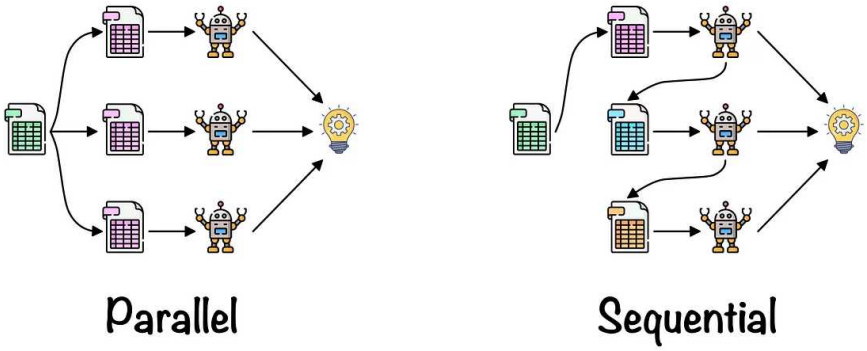
A continuación se introducen una serie de conceptos, que serán de utilidad para contextualizar el trabajo:

1.1. Ensamble de modelos predictivos

La idea del ensamble de modelos consiste en entrenar varios modelos sobre un mismo problema y luego combinarlos para obtener un modelo final (al que llamaremos meta-modelo) con mayor capacidad predictiva que los modelos que lo componen. Existen numerosas formas de agregar estos modelos (técnicas de ensamble). Las más sencillas consisten en calcular la predicción media o la más frecuente entre las predicciones de cada uno de los modelos.



Existen diferentes técnicas para construir ensambles de modelos. Podemos agruparlas en dos grupos, según la estrategia que siguen a la hora de combinar los diferentes modelos: en **paralelo** y de forma **secuencial**. Como ya se ha comentado, en este trabajo implementaremos un algoritmo de ensamble de modelos **secuencial**.



Fuente: Medium

Uno de los requisitos necesarios para obtener un buen ensamble de modelos es que estos sean diferentes entre si. Esto se puede conseguir de dos formas: 1) combinando modelos obtenidos a partir de algoritmos diferentes (redes neuronales, regresión lineal, Knn, árboles de decisión, etc.) sobre el mismo conjunto de entrenamiento. 2) Combinando modelos obtenidos mediante el mismo algoritmo pero sobre variaciones del conjunto de entrenamiento. En este trabajo nos centraremos en la segunda opción.

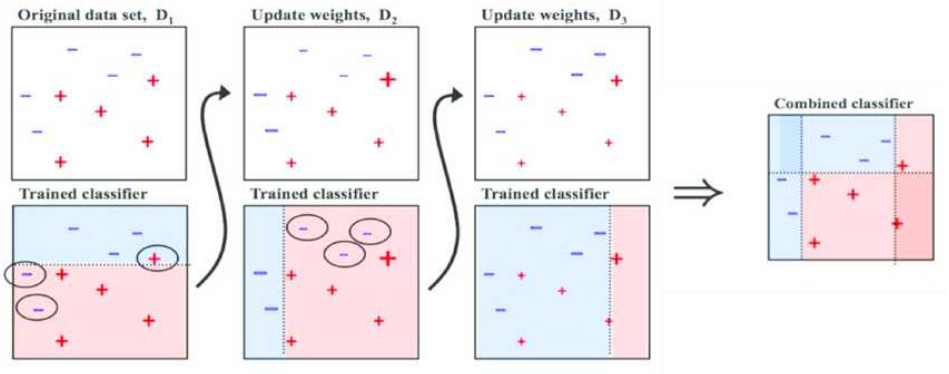
La idea detrás de esto es que, en vez de obtener un modelo que intente capturar toda la casuística del problema, obtenemos varios modelos, cada uno especializado en una parte del problema, que al ponerlos en común proporcionan una solución más robusta.

Para entrenar cada modelo con un conjunto de entrenamiento con variaciones, consideraremos dos técnicas:

- Muestreo de filas: Cada modelo de nuestro ensamble será entrenado considerando un subconjunto de filas extraídas aleatoriamente (e.g. el 75% de las filas). La muestra aleatoria será diferente para cada uno de los modelos. De esta forma, cada modelo podrá especializarse en instancias de datos diferentes.
- Gradiente del error en lugar de variable respuesta: Se ha comentado que en nuestro ensamble secuencial, cada modelo tiene como objetivo corregir los errores cometidos por los anteriores. Para esto, reemplazamos la variable respuesta original, por el gradiente del error que los modelos anteriores cometen sobre cada ejemplo. A continuación veremos esta parte en detalle.

1.2. **Ensamble secuencial de modelos**

En la estrategia de *ensamblado secuencial*, hacemos que cada modelo débil entrenado, centre su aprendizaje en los errores cometidos (su *gradiente*) por el modelo anterior. Es decir, entrenamos un modelo, calculamos el gradiente del error cometido sobre cada ejemplo del conjunto de entrenamiento y dirigimos el aprendizaje del siguiente modelo para corregir dicho error.



La figura anterior, explica de forma intuitiva, el funcionamiento de un ensamble de modelos secuencial en una tarea de clasificación:

1. Entrenamos un modelo que comete errores en tres instancias (dos pertenecientes a la clase negativa y una a la positiva).
2. Entrenamos otro modelo que se especializa en aprender las dos instancias de la clase negativa que falló el primer modelo.
3. Entrenamos un tercer modelo que aprende la instancia positiva que también falló el primer modelo. Es decir, el tercer modelo se centra en los fallos cometidos por el conjunto de los modelos anteriores.

Como podemos ver, el segundo y tercer modelo son malos clasificadores, incluso peores que el primero, sin embargo, si combinamos la información aprendida por cada modelo, obtenemos un clasificador perfecto. Nótese que la imagen anterior no representa exactamente la solución a implementar en este trabajo, sino que nos da una idea general del funcionamiento del ensamble secuencial de modelos débiles.

1.3. **Conceptos**

En esta sección se presentan algunos conceptos y se detallan las partes más importante del meta-algoritmo.

Alcance del meta-algoritmo: Vamos a trabajar tanto con problemas de **clasificación binaria** como de **regresión**, es decir, quedan excluidos los problemas de clasificación multiclase. Cada ensamble se construye usando **un único tipo de algoritmo**. Por ejemplo, podemos construir un meta-modelo donde todos los modelos que lo componen sean *DecisionTreeRegressor*, o bien, uno donde todos los modelos sean de tipo *KneighborsRegressor.*

Clasificación binaria como problema de regresión: Es posible entrenar un modelo de regresión sobre una variable objetivo binaria (valores 1 o 0). Sin embargo, las predicciones en vez de ser 1 o 0, serán valores continuos entre 0 y 1, que luego debemos traducir al valor de clasificación correspondiente (normalmente fijando 0,5 como umbral de decisión). Por ejemplo, predicciones con valor 0,07, 0,25 o 0,4 serán interpretadas como la clase 0, mientras que predicciones con valor 0,6, 0,8 o 0,95, serán interpretadas como la clase 1.

Error de clasificación continuo: Si miramos el error cometido al predecir un objetivo binario desde la óptica de la clasificación, no hay predicciones mejores o peores, solo correctas o incorrectas (el valor de predicción de un ejemplo coincide o no con el valor real). Sin embargo, desde la óptica de la regresión, diremos que las predicciones (valores continuos) son mejores, cuanto más cercanas sean al valor real. Por ejemplo, si para un registro concreto, la clase o valor real es 0, será preferible una predicción de 0,1 que una de 0,3, ya que la primera denota mayor seguridad (confianza).

Función de pérdida: Durante el proceso de construcción del meta-modelo, daremos el mismo tratamiento a los problemas de clasificación y de regresión. Se ha comentado que necesitamos predicciones continuas para poder medir un error continuo. Por esto, nuestro meta-modelo, independientemente del tipo de tarea, **siempre estará compuesto por algoritmos de regresión** (*LinearRegression*, *DecisionTreeRegressor*, etc.). Y usaremos como función de pérdida a minimizar el **error cuadrático** calculado sobre cada ejemplo:

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

Donde F es la predicción del ensamble (no de un solo modelo) para un ejemplo x_i e y_i es el valor de la variable respuesta para ese ejemplo. Se ha comentado que en cada iteración vamos a calcular el gradiente del error de cada ejemplo. El gradiente respecto de F, sería:

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

Teniendo en cuenta que queremos reducir el error, cambiamos el signo, al igual que hicimos en la actualización de pesos de las redes neuronales:

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

Minimización del error: El objetivo al construir el ensamble es añadir secuencialmente nuevos modelos que reduzcan el error cometido sobre el conjunto de entrenamiento. La nueva variable respuesta a predecir en cada iteración será el gradiente del error cometido sobre cada ejemplo en la iteración anterior, a la que llamaremos, a partir de ahora, **residuo**.

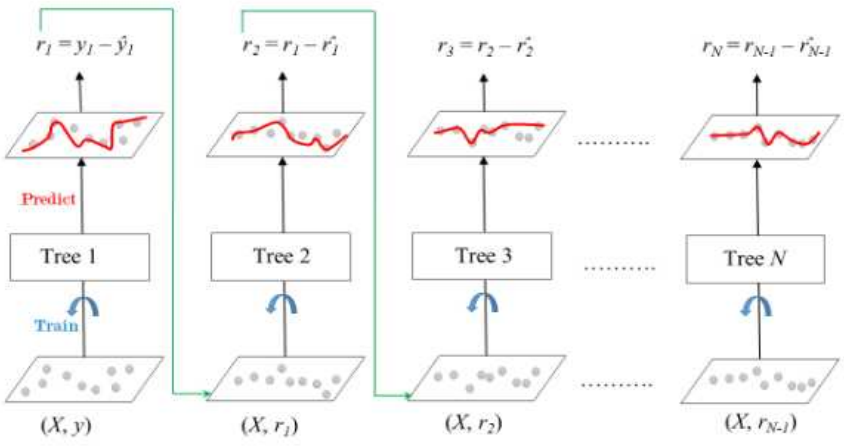
$$\mathrm{r}_{im} = -\left[\frac{\partial L(y_i,F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}\right] = y_i - F_{m-1}(x_i)$$

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 (error) de la predicción sobre dicho ejemplo, que será la variable a predecir cuando entrenemos y añadamos un nuevo modelo en la iteración *m*.

El objetivo es que el residuo de cada uno de los ejemplos del conjunto de entrenamiento sea lo más cercano posible a cero, a medida que vamos añadiendo modelos (iteraciones). Veamos un ejemplo para clasificación binaria:

y	$F_{m-1}(x)$	$r_m = (y - F_{m-1}(x))$
1	0,8	**0,2**
1	0,95	**0,05**
0	0,1	**-0,1**
1	0,6	**0,4**
0	0,4	**-0,4**
0	0,05	**-0,05**

La imagen a continuación ilustra el proceso en el que vamos añadiendo modelos (árboles de decisión en el ejemplo) secuencialmente para ir minimizando el residuo en cada iteración:



Predicción y factor de aprendizaje: Para suavizar el proceso de minimización del error, modularemos la aportación de cada modelo a la predicción final mediante un factor de aprendizaje. Teniendo en cuenta la tasa de aprendizaje, podríamos definir, la predicción del meta-modelo en cada iteración según la siguiente expresión aditiva:

$$F_m(x) = F_{m-1}(x) + \alpha \cdot h_m(x), \quad m = 1, \ldots, M$$

Donde $F_{m-1}(x)$ es la predicción del meta-modelo en la iteración anterior, α es la tasa de aprendizaje y $h_m(x)$ es la predicción del modelo entrenado en la iteración m. Teniendo en cuenta esto, la expresión para calcular una predicción final a partir de la secuencia completa de modelos que componen nuestro meta-modelo, sería:

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

Es decir, la suma de las predicciones de cada uno de los modelos multiplicada por la tasa de aprendizaje.

Predicción en regresión y clasificación: Se ha comentado que el proceso de aprendizaje y la función de pérdida serán los mismos para tareas de regresión y clasificación. En el caso de la predicción final hay una leve diferencia.

Veamos un ejemplo de cálculo de predicción final en un problema de **regresión**, que coincide con lo comentado hasta ahora (para simplificar el ejemplo, se ha usado una tasa de aprendizaje elevada):

α = 0,4										
$h_1(x)$	$h_2(x)$	$h_3(x)$	$h_4(x)$	$h_5(x)$	$h_1(x)*\alpha$	$h_2(x)*\alpha$	$h_3(x)*\alpha$	$h_4(x)*\alpha$	$h_5(x)*\alpha$	$\hat{y}(x) = F_5(x)$
0,7	0,37	0,32	0,28	0,27	0,28	0,148	0,128	0,112	0,108	**0,776**
-0,12	0,05	0,1	0,07	0,02	-0,048	0,02	0,04	0,028	0,008	**0,048**
0,44	-0,3	-0,2	-0,25	-0,2	0,176	-0,12	-0,08	-0,1	-0,08	**-0,204**
0,6	0,5	0,45	0,37	0,33	0,24	0,2	0,18	0,148	0,132	**0,9**
0,15	-0,05	-0,07	-0,06	-0,05	0,06	-0,02	-0,028	-0,024	-0,02	**-0,032**
0,87	0,3	0,27	0,26	0,16	0,348	0,12	0,108	0,104	0,064	**0,744**

Nótese que la predicción final de nuestro meta-modelo es un valor continuo que no tiene por qué estar entre 0 y 1. Por tanto, en el caso de los problemas de **clasificación**, debemos convertir esta predicción en un valor de clasificación (0 o 1):

1. Escalar las predicciones para que estén dentro del rango [0, 1]. Por ejemplo:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

2. Asignar la clase más cercana a cada valor.

Veamos el ejemplo anterior, como si fuese una tarea de clasificación:

α = 0,4												
$h_1(x)$	$h_2(x)$	$h_3(x)$	$h_4(x)$	$h_5(x)$	$h_1(x)*\alpha$	$h_2(x)*\alpha$	$h_3(x)*\alpha$	$h_4(x)*\alpha$	$h_5(x)*\alpha$	$F_5(x)$	Escalado	$\hat{y}(x)$
0,7	0,37	0,32	0,28	0,27	0,28	0,148	0,128	0,112	0,108	**0,776**	0,8877	**1**
-0,12	0,05	0,1	0,07	0,02	-0,048	0,02	0,04	0,028	0,008	**0,048**	0,2283	**0**
0,44	-0,3	-0,2	-0,25	-0,2	0,176	-0,12	-0,08	-0,1	-0,08	**-0,204**	0	**0**
0,6	0,5	0,45	0,37	0,33	0,24	0,2	0,18	0,148	0,132	**0,9**	1	**1**
0,15	-0,05	-0,07	-0,06	-0,05	0,06	-0,02	-0,028	-0,024	-0,02	**-0,032**	0,1558	**0**
0,87	0,3	0,27	0,26	0,16	0,348	0,12	0,108	0,104	0,064	**0,744**	0,8587	**1**

Nota: Se ha elegido esta aproximación por simplicidad. En realidad, este tipo de soluciones tienen una formulación más compleja que convierte las predicciones del meta-modelo en probabilidades.

Evaluación de las predicciones: Tanto para clasificación como para regresión, estamos minimizando el error cuadrático durante el entrenamiento. Sin embargo, las predicciones finales, que normalmente obtendremos sobre un conjunto de prueba, las evaluaremos usando métricas propias de la tarea a resolver. Es decir, en el caso de clasificación, usaríamos métricas como la tasa de aciertos (accuracy) y en el caso de regresión métricas como el error absoluto medio (MAE) o el coeficiente de determinación (r2).

2. Objetivos

El **objetivo principal** de este trabajo es implementar tanto la fase de entrenamiento como de predicción de un ensamble secuencial de modelos según las ideas introducidas anteriormente. Una vez implementado, exploraremos su comportamiento mediante la experimentación sobre diferentes conjuntos de datos. Este objetivo se descompone en los siguientes objetivos específicos (en la sección 3 se proporcionan detalles sobre la implementación y experimentación):

- Realizar una implementación general. El código desarrollado no debe estar vinculado a un conjunto de datos o problema concreto, ni a algoritmos específicos. El código debe poder ejecutarse sobre nuevos conjuntos de datos (que tengan la misma estructura que los proporcionados). El código debe permitir cambiar el algoritmo con el que construir el ensamble de modelos. Puede realizar la implementación usando funciones sueltas o como una clase. También puede heredar de *BaseEstimator* de *Scikit-learn*.
- Implementar la función de entrenamiento del meta-modelo. Esta función entrenará secuencialmente los diferentes modelos que lo componen, devolviendo los elementos necesarios para luego realizar predicciones. No es necesario implementar el algoritmo de entrenamiento de los modelos individuales, para lo que puede usar las diferentes clases de *Scikit-learn* (*DecisionTreeRegressor, LinearRegression*, etc.).
- Implementar una función de predicción para el meta-algoritmo. Esta función, a partir de la salida de la función de entrenamiento, calcula las predicciones del meta-modelo sobre un conjunto de prueba proporcionado.
- Evaluación mediante validación cruzada. Se evaluarán, mediante validación cruzada, las configuraciones de hiperparámetros del meta-modelo para identificar las que mejor funcionan con cada conjunto de datos.
- La segunda parte del trabajo consiste en experimentar con el meta-algoritmo implementado sobre diferentes conjuntos de datos. Se pide que se exploren los valores más adecuados para los diferentes hiperparámetros del meta-algoritmo y que se razonen los resultados obtenidos.
- Documentar el trabajo en un documento con formato de artículo científico, explicando con precisión las decisiones de diseño en la implementación del meta-algoritmo. También se explorarán y documentarán las configuraciones de hiperparámetros con las que nuestro meta-algoritmo tiene un buen desempeño y aquellas en las que el rendimiento sea malo.
- Realizar una presentación de los resultados obtenidos en la defensa del trabajo.

3. Detalles de implementación

A continuación se introduce la metodología a seguir para el correcto desarrollo del trabajo. Algunos detalles, como si se trabajará sobre problemas de regresión o clasificación, dependen de la convocatoria en la que realice el trabajo (ver sección 4).

3.1. Implementación de los algoritmos

El uso de clases para construir el meta-algoritmo facilitará su diseño y usabilidad, sin embargo, no es obligatorio. Se permite (y se recomienda) el uso funciones ya implementadas en *Scikit-learn*, o en otra librería, para el cálculo de métricas de evaluación, la obtención de muestras aleatorias a partir de conjuntos de datos y el entrenamiento de los modelos que componen el meta-modelo. No se permite el uso de funciones de alto nivel, o código de terceros, que implementen total o parcialmente las funciones de entrenamiento y predicción del meta-algoritmo, dado que representan uno de los objetivos principales de este trabajo.

Todo el trabajo se realizará en un cuaderno de Jupyter. En la parte superior del cuaderno se proporcionarán las funciones y clases desarrolladas (también puede importarlas desde un fichero .py), así como todas las importaciones de librerías externas que se hayan utilizado. A continuación se mostrarán los diferentes experimentos realizados. Es importante, documentar tanto el código como los experimentos, así como incluir y comentar las diferentes pruebas realizadas (no solo las que proporcionen mejor resultado). Cualquier experimento relevante que mencione en la memoria, debe poder ser reproducido en el cuaderno.

El **algoritmo de entrenamiento** tendrá como entrada, al menos, los siguientes hiperparámetros:

- Conjunto de entrenamiento.
- Número de estimadores (n_estimators): Tomará cualquier valor a partir de uno. Es el número de iteraciones a realizar, es decir, el número de modelos a entrenar.
- Proporción de ejemplos (*sample_size*): Es un número en el rango [0, 1] que indica la proporción de ejemplos o filas del conjunto de entrenamiento a considerar cuando se entrene cada modelo. Por ejemplo, un valor de 0,75 implica que debemos tomar aleatoriamente una muestra que contenga aproximadamente *0,75*N* ejemplos del conjunto de entrenamiento (donde N es el número de filas del conjunto de datos). Este muestreo aleatorio se repetirá para cada modelo del ensamble. **Importante**: el muestreo debe realizarse sin reemplazamiento.
- Tasa de aprendizaje (*lr*): Es el parámetro que modula el aporte de las predicciones de cada modelo en el proceso de aprendizaje del meta-modelo, es decir, a la hora de minimizar el error cometido.
- Estimador: Clase del algoritmo de Scikit-learn (*DecisionTreeRegressor, LinearRegression*, *KneighborsRegressor,* etc.) a usar para entrenar los diferentes modelos que formarán parte del meta-modelo.
- Hiperparámetros del estimador: Aquellos hiperparámetros del estimador que considere relevante configurar. Por ejemplo, *max_depth* para *DecisionTreeRegressor* o *n_neighbors* para *KneighborsRegressor.* No optimice demasiados hiperparámetros del estimador, solo los que considere más relevantes.

A continuación se resume el procedimiento de **entrenamiento** del meta-algoritmo:

1. *Inicializar la primera predicción con* $pred_0$
2. $pred_{actual} = pred_0$
3. *Por cada i en n_estimators:*
    1. $residuo_i = y - pred_{actual}$
    2. *entrenar* $estimador_i$ *usando* $residuo_i$ *como variable objetivo*
    3. *obtener las predicciones,* $pred_i$*, de* $estimador_i$
    4. $pred_{actual} = pred_{actual} + pred_i * lr$
4. *Devolver el conjunto de modelos entrenados*

Nota: Para $pred_0$ puede usar 0, la media de *y,* u otro valor constante que considere. Además de los modelos puede devolver la información que considere necesario guardar.

La **función de predicción** recibirá un conjunto de datos de prueba (no usado en el entrenamiento) y proporcionará una predicción final para cada ejemplo del mismo. Para eso debemos:

1. Agregar las predicciones de los modelos obtenidos, según se vio en el apartado anterior, es decir, teniendo en cuenta la tasa de aprendizaje.
2. (Solo clasificación) Escalar las predicciones para que estén dentro del rango [0, 1].
3. (Solo clasificación) Asignar la clase más cercana a cada valor.

Una vez que hemos obtenido las predicciones sobre el conjunto de prueba, mediremos la capacidad predictiva del meta-modelo mediante diferentes métricas de evaluación.

Para la **evaluación** de las diferentes configuraciones de hiperparámetros deberá aplicar validación cruzada. Es decir:

1. *Dividir el conjunto de datos en K particiones*
2. *Por cada partición i realizar un experimento en el que:*
    1. *Entrenará un ensamble de modelos sobre las particiones restantes*
    2. *Usará el meta-modelo obtenido para generar predicciones sobre la partición de prueba i*
    3. *Medirá el rendimiento sobre la partición de prueba i usando las métricas indicadas (ver sección 4)*
3. *Devolver el rendimiento medio para cada una de las métricas*

Nota: Para esto puede usar las funciones de validación cruzada que proporciona Scikit-learn (lo que requiere que su meta-algoritmo se implemente como una clase que hereda de BaseEstimator), aunque es posible que le resulte más sencillo implementar su propio proceso de validación cruzada.

3.2. *Experimentación*

El objetivo de esta fase del trabajo es, por un lado, mostrar el correcto funcionamiento del algoritmo desarrollado, y por otro, explorar el espacio de hiperparámetros del mismo (incluyendo los hiperparámetros de los estimadores).

Conjuntos de datos

Para experimentar con el algoritmo desarrollado, se proporcionan dos conjuntos de datos (ver sección 4), aunque los alumnos son libres de incluir conjuntos adicionales si lo consideran oportuno. Los conjuntos de datos han sido tratados y casi no necesitan preprocesado. Solo deben

convertirse las variables categóricas (texto) a numéricas, usando el método que considere más adecuado.
Importante: Debe trabajar con los ficheros de datos proporcionados, y no con otras versiones del mismo conjunto de datos que encuentre por la red.

Exploración de hiperparámetros

Los hiperparámetros de un algoritmo permiten configurar el proceso de entrenamiento, como por ejemplo, la tasa de aprendizaje del meta-algoritmo, la profundidad máxima hasta la que se puede desarrollar un árbol y el número de vecinos a considerar en KNN. Los modelos de aprendizaje automático son muy sensibles a estos parámetros, y de ellos dependerá en gran medida la capacidad predictiva que tenga el modelo entrenado. Es importante tener en cuenta que los mejores hiperparámetros para un conjunto de datos, no lo serán para otros. El objetivo de esta fase es explorar y comprender como varía el comportamiento de nuestro meta-algoritmo al modificar tanto sus hiperparámetros, como los del estimador que lo compone, y obtener aquellas combinaciones de que proporcionan mayor capacidad predictiva.

A la hora de explorar la capacidad predictiva del algoritmo desarrollado, es recomendable tener una **referencia de partida**. Para esto, podemos realizar un experimento de entrenamiento, predicción y evaluación (mediante validación cruzada) de un algoritmo simple (DecisionTreeClassifier/DecisionTreeRegressor, LogisticRegression/LinearRegression, etc.) según el tipo de tarea clasificación/regresión. A partir de ahí, se evaluarán, mediante validación cruzada, diferentes configuraciones de nuestro meta-algoritmo, con el fin de identificar aquellas que mejor rendimiento proporcionan. Un modelo obtenido con nuestro meta-algoritmo, debería ser capaz de mejorar (o igualar en algunos casos) el rendimiento proporcionado por el modelo sencillo. Lo ideal es usar el mismo esquema de particiones de la validación cruzada en todos los experimentos (al trabajar con conjuntos de datos pequeños, la variabilidad de los resultados es alta).

En cada trabajo será necesario experimentar con meta-modelos construidos a partir de dos tipos de estimadores diferentes. Uno fijo y el otro de libre elección. El estimador fijo será DecisionTreeClassifier o DecisionTreeRegressor según el tipo de tarea.

Los hiperparámetros a explorar son:

- Número de estimadores (n_estimators): Tomará cualquier valor a partir de uno. Valores entre 1 y *300* son aceptables. Nótese, que dependiendo de la implementación y del tamaño del conjunto de datos, el entrenamiento del meta-algoritmo sobre *200* o *300* modelos puede requerir un tiempo considerable.
- Proporción de ejemplos (*sample_size*): Se recomienda usar valores superiores a *0.5* y siempre menores o iguales a 1.
- Tasa de aprendizaje (*lr*): Valores entre 0 y 1.
- Hiperparámetros del estimador: Aquelos hiperparámetros del estimador que considere relevante configurar. Por ejemplo, *max_depth* para *DecisionTreeRegressor* o *n_neighbors* para *KneighborsRegressor*. No optimice demasiados hiperparámetros, solo los que considere más relevantes.

Para poder reproducir los resultados, se recomienda fijar la misma **semilla** en todos los procesos aleatorios que utilice en su trabajo. Es importante que todos los experimentos que mencione en la memoria puedan ser reproducidos por el profesor, es decir, no anote resultados y borre el código usado para obtenerlos. Adicionalmente, se recomienda guardar los resultados de las diferentes

combinaciones de hiperparámetros evaluadas en alguna estructura de datos (e.g. DataFrame) que facilite su exploración.

3.3. Mejora opcional

Se propone como mejora opcional la implementación de un mecanismo de parada temprana (*early stopping)* del algoritmo de entrenamiento. Este tipo de mecanismos se usan en algoritmos de minimización iterativa del error como el que nos ocupa. La idea es evaluar la mejora (reducción del error) que se produce en cada iteración y detener el entrenamiento cuando hayan transcurrido una serie de iteraciones consecutivas sin que se produzca mejora alguna. El hiperparámetro que indica dicho número de iteraciones sin mejora se denomina *paciencia* (*early stopping patience*).

Para realizar una evaluación honesta debemos usar datos que no sean parte del conjunto de entrenamiento ni del de prueba. Recordemos que cada estimador del ensamble es entrenado usando una muestra aleatoria del conjunto de entrenamiento, cuyo tamaño viene dado por *el* hiperparámetro *sample_size.* Si, por ejemplo, *sample_size=0,75* hay un 25% del conjunto de entrenamiento que no ha sido usado para entrenar cada estimador (aunque ese 25% sea diferente entre estimadores). Usaremos ese 25% de datos que no se han usado, para evaluar el meta-modelo en cada iteración (por tanto, cuando *sample_size=1,* no podemos aplicar *early stopping*). Vamos anotando los resultados y las iteraciones que transcurren sin mejora (respecto de la anterior), y detenemos el entrenamiento cuando se haya superado el número de iteraciones de *paciencia*. Nótese que en cada iteración evaluamos el meta-modelo completo hasta el momento, no el último estimador.

Esta información también es útil para explorar la convergencia del modelo. Es interesante representar gráficamente el error que se comete en cada iteración. Esta gráfica se puede enriquecer, calculando también el error en cada iteración sobre el conjunto de entrenamiento, lo que permite detectar sobreajuste.

4. Descripción del trabajo y su evaluación

4.1. ***Primera convocatoria – Junio de 2025:***

Tipo de tarea: **Regresión**

Ensamble de modelos usando como estimador:
- **DecisionTreeRegressor**
- Algoritmo de libre elección

Medida de rendimiento principal:
- **R2** - Coeficiente de determinación[1]
- Puede añadir otras métricas que considere interesantes.

Conjuntos de datos:
- Datos sobre precio de viviendas
- Datos sobre la enfermedad de Parkinson

1 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.r2_score.html

4.2. Segunda y tercera convocatoria – Julio y noviembre de 2025:

Tipo de tarea: **Clasificación**

Ensamble de modelos usando como estimador:

- **DecisionTreeClassifier**
- Algoritmo de libre elección

Medida de rendimiento principal:

- **Tasa de aciertos balanceada**[2]
- Puede añadir otras métricas que considere interesantes.

Conjuntos de datos:

- Datos sobre pasajeros del Titanic
- Datos clínicos sobre pacientes con PCOS

4.3. Presentación del código

El trabajo debe presentarse en forma de cuadernos (notebooks) de Jupyter. Se proporcionarán las implementaciones solicitadas así como su aplicación sobre los conjuntos de datos proporcionados. El código y los experimentos realizados deben estar debidamente documentados, las decisiones tomadas debidamente justificadas y los resultados obtenidos deben ser interpretados.

4.4. Documentación y entrega

El trabajo deberá documentarse siguiendo un formato de artículo científico, con una extensión mínima de 6 páginas. En la página web de la asignatura se pueden encontrar plantillas donde se sugiere una estructura general. Estas plantillas siguen el formato de los IEEE conference proceedings, cuyo sitio web guía para autores[3] ofrece información más detallada. El documento entregado deberá estar en formato PDF. Se valorará el uso del sistema LaTeX[4].

La estructura general del documento debe ser como sigue: en primer lugar realizar una **introducción** al trabajo explicando el objetivo fundamental, incluyendo un breve repaso de antecedentes en relación con la temática del trabajo. A continuación, describir la **estructura** del trabajo, las **decisiones de diseño** que se hayan tomado a lo largo de la elaboración del mismo, y la **metodología** seguida al implementarlo (**nunca poner código**, pero sí pseudocódigo), y seguidamente detallar los **experimentos** llevados a cabo, **analizando los resultados obtenidos** en cada uno de ellos. Por último, el documento debe incluir una sección de **conclusiones**, y una **bibliografía** donde aparezcan no solo las referencias citadas en la sección de introducción, sino cualquier documento consultado durante la realización del trabajo (incluidas las referencias web a páginas o repositorios).

La entrega del trabajo consistirá en **un único fichero comprimido zip** conteniendo la memoria, el código implementado (algoritmo y experimentos) y los conjuntos de datos, así como cualquier otro fichero necesario para ejecutar los experimentos. Es importante usar **rutas relativas** para que el

2 https://scikit-learn.org/stable/modules/generated/sklearn.metrics.balanced_accuracy_score.html
3 https://www.ieee.org/conferences/publishing/templates.html
4 https://es.wikibooks.org/wiki/Manual_de_LaTeXa

profesor pueda ejecutar el cuaderno en otro ordenador sin modificar el código. Todos los resultados obtenidos y mencionados en el documento deben ser reproducibles en el cuaderno de Jupyter.

4.5. Documentación y entrega

El día de la defensa se deberá realizar una breve presentación (PDF, PowerPoint o similar) de 10 minutos en la que participarán activamente todos los miembros del grupo que ha desarrollado el trabajo. Esta presentación seguirá, a grandes rasgos, la misma estructura que el documento, pero se deberá hacer especial mención a los resultados obtenidos y al análisis crítico de los mismos. Se podrá usar un portátil (personal del alumno), diapositivas y/o pizarra. En los siguientes 10 minutos de la defensa, el profesor realizará preguntas sobre el trabajo, que podrán ser tanto del documento como del código fuente. Adicionalmente, el profesor podrá proporcionar un nuevo conjunto de datos con el que probar el algoritmo implementado.

5. Criterios de evaluación

Para la evaluación del trabajo se tendrán en cuenta los siguientes criterios, considerando una nota total máxima de 4 puntos:

Implementación y código fuente (1.75 puntos): Se valorará la claridad y buen estilo de programación, corrección, eficiencia y usabilidad de la implementación, y calidad de los comentarios. Se tendrá en cuenta que la implementación de los algoritmos cubra todos los objetivos definidos.

Experimentación y resultados (1.25 puntos): Con respecto a la experimentación, se valorará la calidad y completitud de los experimentos realizados. Además, se tendrá en cuenta el rendimiento del algoritmo y el conjunto de mejores parámetros proporcionado. Es importante que los experimentos realizados y los resultados obtenidos hayan sido documentados e interpretados. Se tendrá en cuenta también la presentación de resultados. Por último, no se tendrán en cuenta aquellos resultados experimentales que no sean reproducibles.

El documento – artículo científico (1 punto): Se valorará el uso adecuado del lenguaje y el estilo general del documento (por ejemplo, el uso de la plantilla sugerida). Se valorará en general la claridad de las explicaciones, el razonamiento de las decisiones, y especialmente el análisis y presentación de resultados en las secciones de experimentación y conclusiones.

Mejora: Se valorará hasta con 0.75 puntos extra sin superar el máximo de 4 puntos del trabajo.

La presentación y defensa: se valorará la claridad de la presentación y la buena explicación de los contenidos del trabajo así como, especialmente, las respuestas a las preguntas realizadas por el profesor, lo que dará lugar para cada alumno por separado a un factor multiplicativo en el intervalo [0, 1] de la nota total obtenida a partir de los apartados anteriores.

Uso de inteligencia artificial generativa: El uso de sistemas de inteligencia artificial generativa está permitido con las siguientes condiciones:

- Debe explicarse para qué se han utilizado esos sistemas, así como describir las entradas (prompts) proporcionadas a los mismos.
- En la defensa del trabajo ambos alumnos deben demostrar conocimiento y entendimiento de la totalidad del trabajo, lo que incluye las respuestas obtenidas de esos sistemas.
- Es muy importante que los alumnos muestren que han entendido el código desarrollado.

IMPORTANTE: Cualquier **plagio**, **compartición de código** o uso de material que no sea original y del que no se cite convenientemente la fuente, significará automáticamente la **calificación de cero** en la asignatura para **todos los alumnos involucrados**. Por tanto, a estos alumnos **no se les conserva**, ni para la actual ni para futuras convocatorias, **ninguna nota** que hubiesen obtenido hasta el momento. Todo ello sin perjuicio de las correspondientes **medidas disciplinarias** que se pudieran tomar.