Guía de estudio - Modelos de ensamble (parte II)

 ¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

En esta oportunidad, presentaremos un nuevo modelo de ensamble que opera en forma secuencial llamado **Boosting**, un algoritmo popular ya que presenta mejor desempeño que Random Forest. Este modelo cambia la idea de trabajar con estimadores fuertes -como se hace en Bagging y Random Forest- y en lugar de ello utiliza "aprendices" débiles que, al ser dirigidos en forma correcta y luego combinados se logra potenciar los "aprendices" débiles en uno fuerte.

Se dará a conocer en detalle dos implementaciones de Boosting: refuerzo adaptativo (adaBoost) y refuerzo del gradiente (gradientBoosting). Veremos en forma gráfica el aporte que van realizando los aprendices débiles en cada iteración y los pasos a seguir para cada algoritmo.

Luego de mostrar los fundamentos teóricos de los algoritmos, usaremos implementaciones de ellos en *sklearn* y los aplicaremos a un conjunto de datos, presentando sus hiper parámetros. Además, realizaremos una comparación de los resultados de ambos algoritmos.

Poco a poco iremos explicando los algoritmos con los cálculos necesarios en cada etapa y aplicando estos en Python con la librería sklearn. Al concluir esta guía podrás entender el funcionamiento de los ensambles boosting, dominando su aplicación para ser usado en problemáticas de diversa índole mediante la librería *sklearn*.

¡Iniciemos esta zambullida en modelos de ensambles secuenciales, que serán una gran herramienta para mejorar el desempeño de nuestros modelos!

¡Vamos con todo!

Tabla de contenidos

¡Comencemos!

Boosting

Boosting es un enfoque de ensamble secuencial, en el que se usa un conjunto de "aprendices" débiles, entendiendo por estos a estimadores cuyo desempeño es apenas superior a un estimador aleatorio. Comúnmente se emplean árboles de decisión con profundidad uno (*Decision stumps),* los cuales son potenciados dando origen a un estimador fuerte.

Se pueden distinguen dos diferencias importantes entre ensamble paralelo y secuencial:

1. Los ensambles paralelos entrenan estimadores en forma independiente, lo que significa que pueden aprovechar arquitecturas con múltiples procesadores para el entrenamiento y lograr mejorar el tiempo empleado. En cambio, en los ensambles secuenciales cada estimador depende de la salida del estimador anterior, lo que se traduce en que los ensambles secuenciales requieren mayor tiempo de cómputo que los paralelos.
2. Mientras que en los ensambles paralelos se usan "aprendices" fuertes, es decir, aquellos que logran niveles altos de desempeño (por ejemplo, Random Forest utiliza árboles de decisión que crecen sin restricción de profundidad), los ensambles paralelos emplean "aprendices" débiles, *Decision stumps,* que al pasar por un proceso secuencial con una estrategia de ir enfocándose en los errores en cada iteración y finalmente combinando estos estimadores forman un estimador fuerte.

Refuerzo Adaptativo (Adaptive Boosting)

En 1990, Robert E. Sapphire formuló la idea de AdaBoost, y en el año 2003 Freund y Schapire lograron el premio Gödel, que es otorgado a aquellas publicaciones más importantes en el campo de la informática.

AdaBoost se compone de aprendices débiles, los cuales se entrenan en forma secuencial. Así, una vez entrenado el primer aprendiz se identifican aquellas observaciones mal clasificadas y se realiza una ponderación a estas con el objetivo que el siguiente aprendiz se enfoque en los errores detectados. De esta forma estaremos dirigiendo a las aprendices débiles para que aprendan de los ejemplos mal clasificados ponderados por el aprendiz débil anterior, y con ello se mejora el rendimiento conjunto.

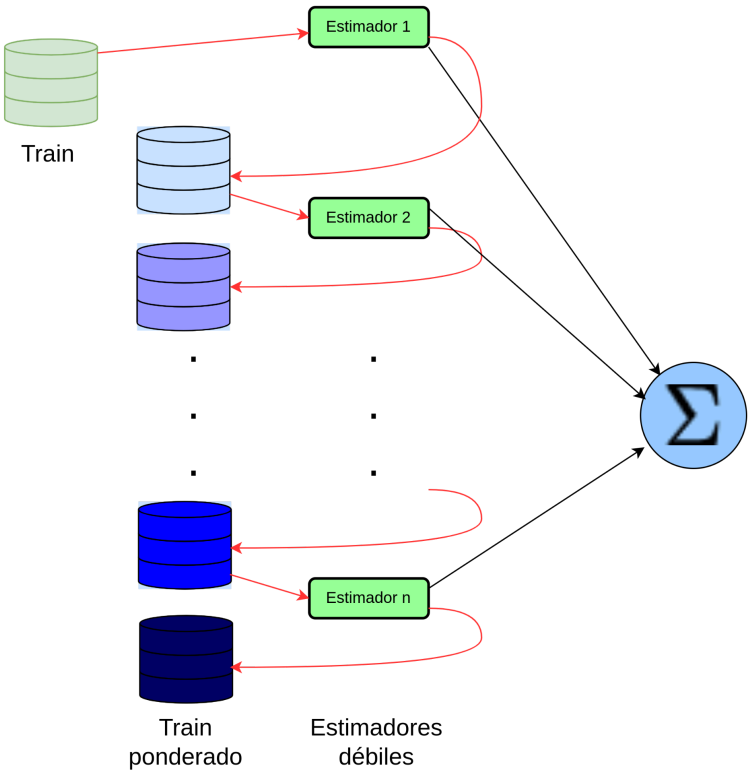

Figura 1. Ensamble secuencial con weak learners
Fuente: Fuente propia.

Este algoritmo no solamente pondera a las observaciones mal clasificadas, sino que disminuye la ponderación de aquellas bien clasificadas con el objetivo que el siguiente aprendiz se concentre en los sujetos que no han sido bien clasificados.

Proceso paso a paso en AdaBoost

1. **Inicialización de pesos:** cada valor en el conjunto de datos de entrenamiento se asocia con un peso inicial. Al principio, todos los pesos son iguales, y la suma de todos ellos es igual a 1.
2. **Entrenamiento de un modelo débil:** se entrena un clasificador débil (por ejemplo, un árbol de decisión) en el conjunto de datos de entrenamiento. El modelo se evalúa y se calcula la tasa de error ponderada, teniendo en cuenta los pesos de las instancias (considerar nuevamente que, en principio, todas las ponderaciones son iguales)
3. **Cálculo de la importancia del modelo:** cuanto menor sea el error ponderado, mayor será la importancia del modelo. Considerando esto se asigna un peso al modelo en función de su importancia. Modelos con menor error tendrán un peso mayor.
4. **Actualización de los pesos de las instancias:** se aumentan los pesos de las instancias mal clasificadas por el modelo débil.
5. **Iteración**: se repiten los pasos 2-4 varias veces (según el número especificado de iteraciones o hasta que se alcance cierto criterio de parada).

6. **Construcción del modelo final:** se combinan los modelos débiles ponderados para formar un modelo fuerte. La combinación se realiza asignando más peso a los modelos con menor error.
7. **Predicción**: el modelo final se utiliza para hacer predicciones en nuevas instancias.

Evolución en la disminución del error

En la Figura 2, se muestra un conjunto de datos sintéticos en que se aprecian dos clases (verdes y rojos) y que la separación es no lineal. Por medio de AdaBoost podemos ver cómo en cada iteración el error disminuye, como se ve en la Figura 3.

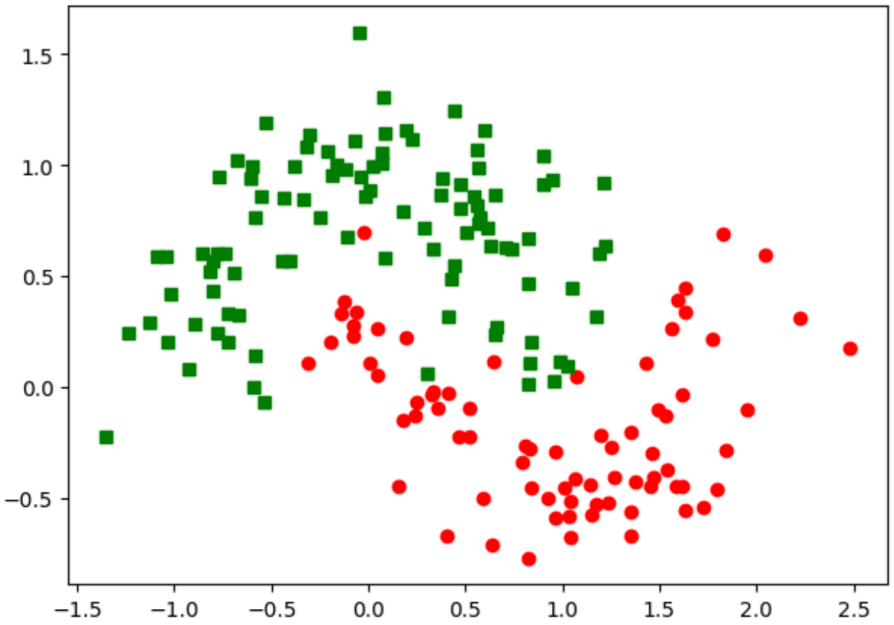

Figura 2. Datos sintéticos de separación no lineal
Fuente: Fuente propia.

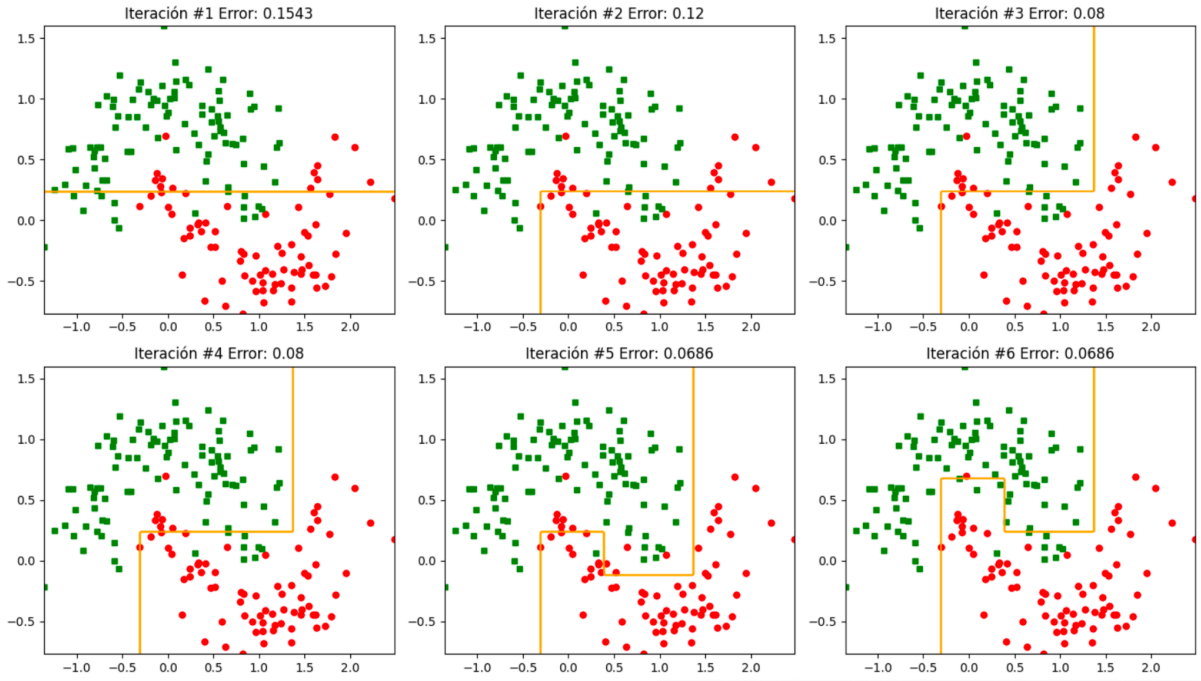

Figura 3. Resultados de AdaBoost para seis aprendices débiles
Fuente: Fuente propia.

Actividad guiada: Aplicando AdaBoost con sklearn

El algoritmo de refuerzo adaptativo en sklearn se encuentra en el paquete ***AdaBoostClassifier,*** cuyos parámetros más importantes son:

- ***base_estimator:*** corresponde al estimador base que usará en el ensamble.
- ***n_estimators:*** cantidad de aprendices débiles que se emplearán para el entrenamiento en el ensamble.
- ***learning_rate:*** con esto podemos controlar el aporte de los aprendices débiles, en que valores pequeños en *learning_rate* se traduce que los α se hagan más pequeños. En consecuencia, las variaciones en las ponderaciones asociadas a los sujetos disminuyen induciendo a menor cantidad de diversidad en los aprendices débiles. Por otro lado, un *learning_rate* alto genera mayor diversidad en los aprendices débiles.

Trabajaremos con datos asociados a cáncer de mama, donde tenemos dos clases: *'malignant'* y *'benign'*. Para esto, analiza el archivo **02 - Guía**, sección **AdaBoost**

Refuerzo del Gradiente (Gradient Boosting)

Gradient Boosting, al igual que Adaptive Boosting, trabaja con aprendices secuenciales. Pero podemos señalar algunas diferencias:

AdaBoost	Gradient Boosting
Ponderación individual de los estimadores	Tasa de aprendizaje
Árboles con profundidad 1	Árboles con profundidad entre 3 y 6
Uso de errores previos para ajustar los pesos de los ejemplos	Uso de los errores residuales para ir entrenando los estimadores.

Gradient Boosting en cada iteración va ajustando su aprendiz de acuerdo a los residuos del estimador anterior, cada nuevo aprendiz es ajustado de acuerdo al gradiente negativo de la función de pérdida.

Procedimiento para Gradient Boosting

Para revisar en detalle este algoritmo es necesario conocer el algoritmo de optimización de gradiente descendente, se explicará a continuación.

Tenemos una función cualquiera derivable, y queremos encontrar en qué coordenadas se encuentra su valor mínimo comenzando en algún punto inicial dado. Para ello, debemos calcular el gradiente negativo de la función para conocer el vector que apuntará en la dirección negativa del máximo crecimiento, es decir, apuntará hacia el máximo decrecimiento. Entonces, normalizamos el vector de la dirección y nos movemos desde el punto inicial a un nuevo punto, que será el punto inicial más una cantidad dada por "el paso" multiplicado por el vector de dirección. Esto lo repetimos una cantidad fija de veces o hasta que la magnitud de cambio sea cercana a cierto umbral. De esta forma encontraremos un mínimo local o si tenemos suerte un mínimo global.

Un valor de "paso" pequeño hará que encontrar un mínimo sea más lento que si usamos un valor más grande; pero un valor muy grande puede ser un problema ya que podríamos alejarnos de un mínimo que esté próximo al punto en el que estamos.

El descenso del gradiente es la forma en que Gradient Boosting consigue aproximar un modelo fuerte usando aprendices débiles.

Algoritmo descenso del gradiente

1. Inicializamos los parámetros a buscar
2. Se fija un valor para el "paso"
3. Repetimos los siguientes pasos mientras no haya convergencia:
    a. Calcular la dirección del gradiente negativo para la función de pérdida, con tamaño 1, evaluado en el punto actual.
    b. Actualizar los parámetros = parámetro anterior + paso * dirección
    c. Calcular la variación entre el nuevo parámetro y el anterior. Si esta diferencia es menor a un umbral se detiene el ciclo, de lo contrario almacenamos en el parámetro anterior el nuevo valor. También podemos fijar la detención estableciendo con una cantidad de ciclos.

Algoritmo refuerzo del gradiente (Gradient Boosting)

1. Se inicializa con un F que representa nuestra estimación inicial con un valor constante.

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, \gamma)$$

2. Para cada estimador que compone el ensamble:
    2.1. Calculamos el *pseudo residuo* para cada ejemplo de entrenamiento. Esto equivale a la diferencia entre el verdadero valor menos el F estimado anterior.

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}, \; i = 1, \cdots N$$

    2.2. Entrenamos un aprendiz débil con los datos de entrenamiento asociados al residuo calculado anteriormente.

$$\{(x_i, r_{im})\}_{i=1}^{N}$$

    Este aprendiz debe aproximar un valor real (residuos) que representa al gradiente de la función de pérdida, por lo cual, estará resolviendo un problema de regresión.

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

2.3. Actualizamos el nuevo valor para nuestro F, que será:

$$F_m(x) = F_{m-1}(x) + \eta h(x, \gamma_m)$$

El proceso de aprendizaje de Gradient Boosting se muestra en la Figura 4.

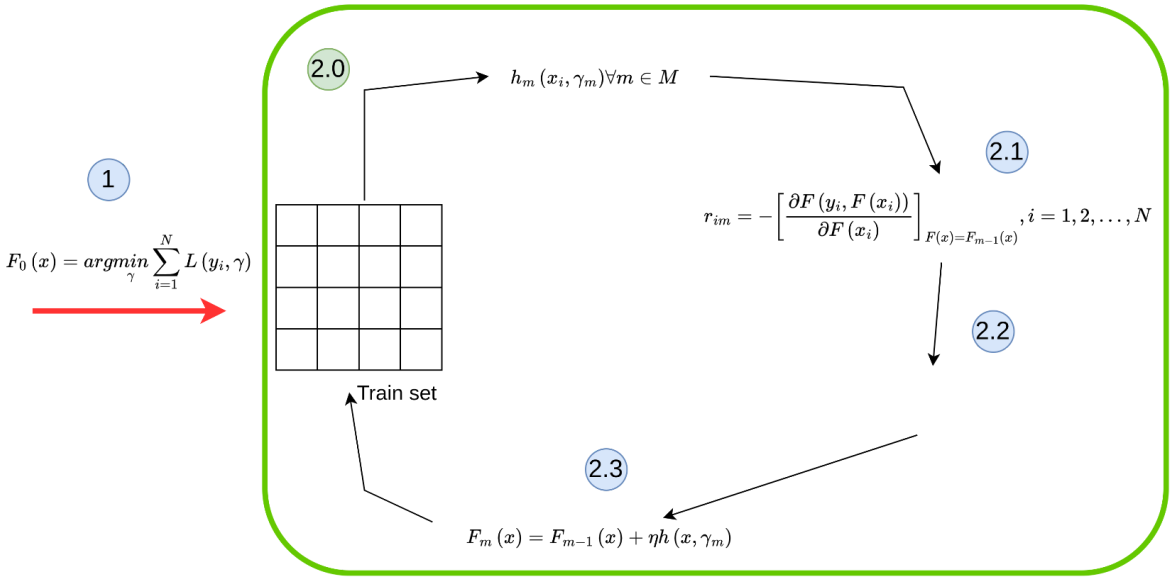

Figura 4. Proceso de aprendizaje para Gradient Boosting
Fuente: Fuente propia.

La diferencia entre el proceso de aprendizaje usando el algoritmo del descenso del gradiente y el que se obtiene usando el refuerzo del gradiente (Gradient Boosting) corresponde a que en el descenso del gradiente se actualiza la nueva posición en la función de pérdida usando directamente el gradiente de la función, en cambio en Gradient Boosting el gradiente de la función se aproxima usando aprendices débiles. En la Figura 5 vemos como cada vez que concluye una iteración nos encontramos con un modelo aditivo que va acercándose a un mínimo en la función de pérdida.

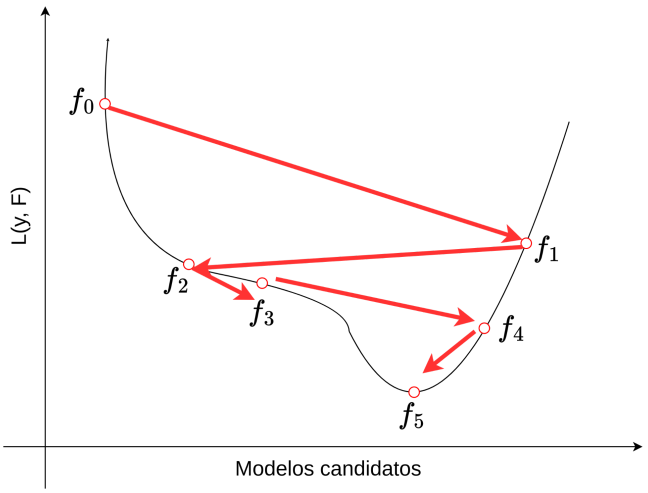

Figura 5. Gradient Boosting aproximando el gradiente y moviéndose en la función de pérdida
Fuente: Fuente propia.

Actividad guiada: Construcción de un modelo de clasificación usando Gradient Boosting en sklearn

Para esta actividad se utilizará el mismo conjunto de datos que en Adaptive Boosting, es decir, la base de cáncer de mamas. Se debe considerar que comenzaremos ya con los conjuntos separados de entrenamiento y test que se realizó en la actividad anterior.

Se crea y se entrena un modelo de Gradient Boosting usando la librería ***GradientBoostingClassifier*** usando como hiper parámetro de la tasa aprendizaje ***0.01*** y para una cantidad de estimadores de ***40***. Analiza lo planteado en el archivo Jupyter anterior utilizado para AdaBoost, ahora en la sección **Gradient Boosting**

Refuerzo del Gradiente Extremo (XGBoost)

Como ya se ha descrito anteriormente, los modelos de ensamble secuenciales pueden ser lentos en su entrenamiento. Considerando esto, Extreme Gradient Boosting es una implementación de Gradient Boosting que acelera sustancialmente el entrenamiento, además de presentar buenas predicciones. Sabemos que XGBoost goza de popularidad ya que ha ganado varios concursos de Kaggle. Para su uso, en todo caso, es preciso considerar algunos aspectos:

Es recomendable usar XGBoost cuando estamos en presencia de:

- Datos complejos no lineales.
- Grandes conjuntos de datos.

- Alta dimensión
- Requerimientos de alta precisión

Por el contrario, no se recomienda usar XGBoost si:

- Basta con utilizar modelos simples
- Es necesario interpretar el modelo
- Se cuenta con recursos computacionales limitados.

Actividad guiada: Construcción de un modelo de clasificación usando XGBoost

Vamos a replicar el mismo caso anterior, ahora con XGBoost. Para esto, analiza lo planteado en el documento de Jupyter, sección XGBoost

Reflexiona:

- ¿Qué tipos de problemas que ya hemos visto en la carrera podrían resolverse de mejor forma aplicando los algoritmos vistos en esta sesión?
- ¿Qué complicaciones puede tener la aplicación de estos algoritmos? ¿Qué aspectos se deben observar para evitarlas?

Referencias bibliográficas

- Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, *55*(1), 119-139.
- Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: data mining, inference, and prediction* (Vol. 2, pp. 1-758). New York: springer.