Boosting

Matías Carrasco

23 de octubre de 2023

Índice

1. Introducción

Bagging y Boosting son métodos de ensemble diseñados para mejorar modelos base. Bagging se centra en reducir la varianza de modelos con alta varianza, mientras que Boosting apunta a reducir el sesgo en modelos con alto sesgo, como los árboles de clasificación de profundidad uno (stumps). Boosting entrena múltiples modelos débiles para combinar sus predicciones y formar un modelo más fuerte.

Aunque ambos son métodos de ensemble y pueden considerarse meta-algoritmos que se construyen sobre otros algoritmos, difieren en cómo se entrenan los modelos base. Bagging entrena modelos en paralelo, mientras que Boosting lo hace secuencialmente, corrigiendo errores del modelo anterior al ajustar el conjunto de datos en cada iteración. La predicción final en Boosting proviene de un promedio ponderado o voto ponderado entre los modelos.

2. AdaBoost

AdaBoost fue la primera implementación práctica exitosa de la idea de boosting. Intenta construir una secuencia de K clasificadores binarios (débiles) $A_1(\boldsymbol{x})$, $A_2(\boldsymbol{x})$, ..., $A_K(\boldsymbol{x})$. Solo consideraremos la predicción final de los modelos base, y no sus probabilidades de clase predichas.

Cualquier modelo de clasificación puede, en principio, ser utilizado como clasificador base. En la práctica, es común usar árboles de clasificación poco profundos. Las predicciones individuales de los miembros del ensemble se combinan en una predicción final, en donde no todas son tratadas por igual. Es decir, asignamos coeficientes $\{\alpha_k\}_{k=1}^{K}$ y construimos el clasificador usando una votación mayoritaria ponderada:

$$A_{\text{boost}}(\boldsymbol{x}) = \text{Signo}\left\{\sum_{k=1}^{K} \alpha_k A_k(\boldsymbol{x})\right\}.$$

Cada miembro del conjunto vota ya sea -1 o $+1$, y la salida del clasificador ensemble es $+1$ si la suma ponderada de los votos individuales es positiva y -1 si es negativa. El coeficiente α_k puede considerarse como un grado de confianza en las predicciones realizadas por el miembro k del ensemble.

En AdaBoost se entrena de manera greedy minimizando una **pérdida exponencial** en cada iteración. La pérdida exponencial está dada por

$$L(m) = \exp(-m),$$

donde $m = yf(\boldsymbol{x})$ es el **margen** de un clasificador. Aquí la función $f(\boldsymbol{x})$ devuelve un valor real el cual es comparado con un umbral para definir la clase a predecir.

Los miembros del ensemble se agregan uno a la vez y, cuando se agrega el miembro k, esto se hace para minimizar la pérdida exponencial de todo el ensemble construido hasta el momento. La principal razón para elegir la pérdida exponencial es que resulta en expresiones explícitas convenientes.

Escribamos el clasificador ensemble después de k iteraciones como

$$A_{\text{boost}}^{k}(\boldsymbol{x}) = \text{Signo}\left\{f^{(k)}(\boldsymbol{x})\right\}$$

donde $f^{(k)}(\boldsymbol{x}) = \sum_{j=1}^{k} \alpha_j A_j(\boldsymbol{x})$. Podemos expresar $f^{(k)}(\boldsymbol{x})$ iterativamente como:

$$f^{(k)}(\boldsymbol{x}) = f^{(k-1)}(\boldsymbol{x}) + \alpha_k A_k(\boldsymbol{x}),$$

inicializado con $A_{\text{boost}}^{(0)}(\boldsymbol{x}) = 0$.

Los miembros del ensemble (así como los coeficientes) se construyen secuencialmente, lo que significa que en la iteración k del procedimiento, la función $f^{(k-1)}(\boldsymbol{x})$ es conocida y fija. Esto es lo que hace que esta construcción sea greedy.

En consecuencia, lo que queda por aprender en la iteración k es el miembro del ensemble $A_k(\boldsymbol{x})$ y su coeficiente α_k. Hacemos esto minimizando la pérdida expo-

nencial en los datos de entrenamiento:

$$\begin{aligned}(\alpha_k, A_k) &= \arg\min_{(\alpha,A)} \sum_{i=1}^{N} L\left(y_i f^{(k)}\left(\boldsymbol{x}_i\right)\right) \\ &= \arg\min_{(\alpha,A)} \sum_{i=1}^{N} \exp\left(-y_i\left(f^{(k-1)}\left(\boldsymbol{x}_i\right) + \alpha A\left(\boldsymbol{x}_i\right)\right)\right) \\ &= \arg\min_{(\alpha,A)} \sum_{i=1}^{N} \underbrace{\exp\left(-y_i f^{(k-1)}\left(\boldsymbol{x}_i\right)\right)}_{=w_i^{(k)}} \exp\left(-y_i \alpha A\left(\boldsymbol{x}_i\right)\right),\end{aligned}$$

donde

- para la primera igualdad hemos utilizado la definición de la función de pérdida exponencial y la estructura secuencial del clasificador potenciado;
- y en la última igualdad es donde aparece la conveniencia de la pérdida exponencial, a saber, el hecho de que $\exp(a+b) = \exp(a)\exp(b)$.

Esto nos permite definir las cantidades:

$$w_i^{(k)} \stackrel{\text{def}}{=} \exp\left(-y_i f^{(k-1)}\left(\boldsymbol{x}_i\right)\right),$$

que pueden interpretarse como pesos para los datos individuales en los datos de entrenamiento. Obsérvese que los pesos $w_i^{(k)}$ son independientes de α y A. Es decir, al aprender $A_k(\boldsymbol{x})$ y su coeficiente α_k podemos considerar $\left\{w_i^{(k)}\right\}_{i=1}^{N}$ como constantes.

Comenzamos reescribiendo la pérdida como:

$$\sum_{i=1}^{N} w_i^{(k)} \exp\left(-y_i \alpha A\left(\boldsymbol{x}_i\right)\right) = e^{-\alpha} \underbrace{\sum_{i=1}^{N} w_i^{(k)} \mathbb{1}_{\{y_i = A(\boldsymbol{x}_i)\}}}_{=W_c} + e^{\alpha} \underbrace{\sum_{i=1}^{N} w_i^{(k)} \mathbb{1}_{\{y_i \neq A(\boldsymbol{x}_i)\}}}_{=W_e},$$

donde hemos utilizado la función indicatriz para dividir la suma en dos partes: la primera abarca todos los datos de entrenamiento correctamente clasificados por A y la segunda abarca todos los datos clasificados erróneamente por A. Además, para simplificar la notación, definimos W_c y W_e para la suma de pesos de los datos clasificados correctamente y erróneamente, respectivamente. Además, sea $W = W_c + W_e$ la suma total de pesos, $W = \sum_{i=1}^{N} w_i^{(k)}$.

La minimización se realiza en dos etapas, primero con respecto a A y luego con respecto a α. Esto es posible ya que el argumento minimizador en A resulta ser

independiente del valor actual de $\alpha > 0$, otro efecto conveniente de usar la función de pérdida exponencial.

Observar que podemos escribir la pérdida como:

$$e^{-\alpha}W + \left(e^{\alpha} - e^{-\alpha}\right) W_e$$

Dado que la suma total de pesos W es independiente de A y dado que $e^{\alpha} - e^{-\alpha} > 0$ para cualquier $\alpha > 0$, minimizar esta expresión con respecto a A es equivalente a minimizar W_e con respecto a A. Es decir,

$$A^{(k)} = \arg\min_{A} \sum_{i=1}^{N} w_i^{(k)} \mathbb{1}_{\{y_i \neq A(\boldsymbol{x}_i)\}}.$$

Es decir, el miembro k-ésimo del ensemble debe ser entrenado minimizando la pérdida de clasificación ponderada, donde a cada dato $(\boldsymbol{x}_i, y_i)$ se le asigna un peso $w_i^{(k)}$. La intuición detrás de estos pesos es que, en la iteración k, deberíamos centrar nuestra atención en los datos previamente mal clasificados para corregir los errores cometidos por el conjunto de los primeros $k-1$ clasificadores.

Cómo se resuelve el problema en la práctica depende de la elección del clasificador base que utilicemos, es decir, de las restricciones específicas que pongamos en la función A. Excepto por los pesos $w_i^{(k)}$, este es nuestro problema de clasificación estándar.

Una vez entrenado el miembro k-ésimo del ensemble, $A_k(\boldsymbol{x})$, queda por aprender su coeficiente α_k. Al diferenciar con respecto a α e igualar la derivada a cero, obtenemos la ecuación

$$-\alpha e^{-\alpha} W + \alpha \left(e^{\alpha} + e^{-\alpha}\right) W_e = 0 \Leftrightarrow W = \left(e^{2\alpha} + 1\right) W_e \Leftrightarrow \alpha = \frac{1}{2} \ln\left(\frac{W}{W_e} - 1\right).$$

Por lo tanto, al definir

$$E_{\text{train}}^{(k)} \stackrel{\text{def}}{=} \frac{W_e}{W} = \sum_{i=1}^{N} \frac{w_i^{(k)}}{\sum_{j=1}^{N} w_j^{(k)}} \mathbb{1}_{\{y_i \neq A_k(\boldsymbol{x}_i)\}}$$

como el error de clasificación ponderado para el clasificador k-ésimo, podemos expresar el valor óptimo de su coeficiente como

$$\alpha_k = \frac{1}{2} \ln\left(\frac{1 - E_{\text{train}}^{(k)}}{E_{\text{train}}^{(k)}}\right)$$

El hecho de que α_k dependa del error de entrenamiento del miembro k-ésimo del ensemble es natural, ya que, como mencionamos anteriormente, podemos interpretar α_k como la confianza en las predicciones de este miembro.

La derivación de AdaBoost supone que todos los coeficientes $\{\alpha_k\}_{k=1}^K$ son positivos. Para ver que esto es efectivamente el caso, observar que la función $\ln((1-x)/x)$ es positiva para cualquier $0 < x < 1/2$. Por lo tanto, α_k será positivo siempre que el error de entrenamiento ponderado para el clasificador k-ésimo, $E_{\text{train}}^{(k)}$, sea menor que $1/2$. Es decir, el clasificador solo tiene que ser ligeramente mejor que lanzar una moneda al aire, lo cual siempre es el caso en la práctica. De hecho, si $E_{\text{train}}^{(k)} > 1/2$, entonces simplemente podríamos invertir el signo de todas las predicciones hechas por A_k.

Algorithm 1 Entrenamiento del clasificador AdaBoost

Require: Datos de entrenamiento $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$
Ensure: K clasificadores base

1: Asignar pesos $w_i^{(1)} = 1/N$ a todas las observaciones.
2: **for** $k = 1$ to K **do**
3: Entrenar clasificador base $A_k(\boldsymbol{x})$ con $\left\{\left(\boldsymbol{x}_i, y_i, w_i^{(k)}\right)\right\}_{i=1}^N$.
4: Calcular $E_{\text{train}}^{(k)} = \sum_{i=1}^N w_i \mathbb{1}_{\{y_i \neq A_k(\boldsymbol{x}_i)\}}$.
5: Calcular $\alpha_k = \frac{1}{2} \ln\left(\frac{1-E_{\text{train}}^{(k)}}{E_{\text{train}}^{(k)}}\right)$.
6: Calcular $w_i^{(k+1)} = w_i^{(k)} \exp(-\alpha_k y_i A_k(\boldsymbol{x}_i)), \forall i = 1, \ldots, N$.
7: Sea $w_i^{(k+1)} \leftarrow \frac{w_i^{(k+1)}}{\sum_{j=1}^N w_j^{(k+1)}}, \forall i = 1, \ldots, N$.
8: **end for**

AdaBoost, y de hecho cualquier algoritmo de boosting, requiere dos decisiones de diseño importantes:

(i) qué clasificador base usar, y

(ii) cuántas iteraciones K ejecutar en el algoritmo.

Como se señaló anteriormente, podemos usar esencialmente cualquier método de clasificación como clasificador base. Sin embargo, la elección más común en la práctica es usar un árbol de clasificación de poca profundidad. Esta elección está guiada por el hecho de que boosting reduce eficientemente el sesgo y, por lo tanto, puede aprender buenos modelos a pesar de usar un modelo base muy débil (de alto sesgo). Dado que los árboles de poca profundidad pueden ser entrenados rápidamente, son una buena elección.

Los modelos base se aprenden secuencialmente en boosting, y cada iteración introduce un nuevo modelo base con el objetivo de reducir los errores cometidos por el modelo actual. Como efecto, el modelo ensemble se vuelve más y más flexible a medida que el número de iteraciones K aumenta y usar demasiados modelos base puede resultar en sobreajuste (a diferencia de bagging, donde un aumento de K no puede llevar a sobreajuste). Sin embargo, se ha observado en la práctica que este sobreajuste a menudo ocurre lentamente y el rendimiento tiende a ser bastante insensible a la elección de K. No obstante, es una buena práctica seleccionar K de alguna manera sistemática, por ejemplo, usando **early stopping** durante el entrenamiento. Otro aspecto negativo de la naturaleza secuencial de boosting es que no es posible paralelizar el aprendizaje.

En el método discutido anteriormente, hemos supuesto que cada clasificador base produce una predicción de clase, $A_k(\boldsymbol{x}) \in \{-1, 1\}$. Sin embargo, muchos modelos de clasificación generan, una estimación de la probabilidad de clase $p(y = 1 \mid \boldsymbol{x})$. En AdaBoost, es posible usar las probabilidades predichas en lugar de la predicción binaria al construir la predicción, aunque a costa de una expresión más complicada. Esta extensión se conoce como **Real AdaBoost**.

3. Gradient boosting

La pérdida exponencial penalizará fuertemente los márgenes negativos grandes, haciéndola sensible al ruido. Para mitigar este problema y construir algoritmos de boosting más robustos, podemos considerar elegir alguna otra función de pérdida (más robusta). Sin embargo, esto será a expensas de un procedimiento de entrenamiento computacionalmente más engorroso.

En la discusión anterior, hemos descrito boosting como el aprendizaje de una secuencia de clasificadores débiles, donde cada clasificador intenta corregir los errores cometidos por los anteriores. Desde una perspectiva matemática, quizás una interpretación más útil es que boosting es una forma de entrenar un **modelo aditivo**:

$$f_{\text{boost}}(\boldsymbol{x}) = \sum_{k=1}^{K} \alpha_k f_k(\boldsymbol{x})$$

donde α_k son coeficientes con valores reales y $f_k(\boldsymbol{x})$ son **funciones base**. Para un problema de regresión, la función $f_{\text{boost}}(\boldsymbol{x})$ puede usarse directamente como la predicción del modelo. Para un problema de clasificación, esta puede ser umbralizada por una función de signo para obtener una predicción de clase definida, o transformadola en una probabilidad de clase pasándola a través de una función adecuada.

AdaBoost sigue esta forma aditiva, donde los clasificadores débiles (miembros del ensemble) son las funciones base y sus puntuaciones de confianza son los coeficientes. Pero por ejemplo, los árboles de regresión pueden ser utilizados para resolver problemas de clasificación en el contexto de gradient boosting. Esta es también la razón por la que usamos f en lugar de A en la notación anterior. Incluso para un problema de clasificación, las salidas de los miembros del ensemble no tienen que corresponder a predicciones de clase en general.

Hay dos propiedades que distinguen boosting de otros modelos aditivos:

1. las funciones base se aprenden de los datos y, específicamente, cada función (es decir, miembro del ensemble) corresponde a un modelo en sí mismo: el modelo base del procedimiento de boosting;
2. las funciones base y sus coeficientes se aprenden secuencialmente. Es decir, añadimos un componente a la suma en cada iteración y después de K iteraciones, el algoritmo de aprendizaje termina.

El objetivo al entrenar un modelo aditivo es seleccionar $\{\alpha_k, f_k(\boldsymbol{x})\}_{k=1}^{K}$ de modo que el $f_{\text{boost}}(\boldsymbol{x})$ final minimice

$$J(f) = \frac{1}{N}\sum_{i=1}^{N} L\left(y_i, f\left(\boldsymbol{x}_i\right)\right)$$

para alguna función de pérdida arbitraria L. Por ejemplo, en un contexto de clasificación binaria, elegir la pérdida logística (u otra función de pérdida robusta) en lugar de la pérdida exponencial resultará en un modelo menos sensible a valores atípicos que AdaBoost.

Al igual que en AdaBoost, en la iteración k introducimos un nuevo miembro del ensemble con el objetivo de reducir el valor de la función de pérdida, pero ahora sin requerir que sea minimizado. Esto nos lleva a la idea de gradient boosting.

Considera la k-ésima iteración del procedimiento de entrenamiento. Como antes, podemos usar la naturaleza secuencial del modelo para escribir

$$f_{\text{boost}}^{(k)}(\boldsymbol{x}) = f_{\text{boost}}^{(k-1)}(\boldsymbol{x}) + \alpha_k f_k(\boldsymbol{x}),$$

y el objetivo es seleccionar $\{\alpha_k, f_k(\boldsymbol{x})\}$ para reducir el valor de la función de pérdida. Es decir, queremos elegir el k-ésimo miembro del ensemble de modo que

$$J\left(f_{\text{boost}}^{(k-1)} + \alpha_k f_k\right) < J\left(f_{\text{boost}}^{(k-1)}\right).$$

Hacemos esto tomando un paso en la dirección negativa del gradiente de la función de pérdida.

¿Con respecto a qué deberíamos calcular el gradiente de la función de pérdida? La idea detrás de gradient boosting, es tomar un enfoque **no paramétrico** y representar el modelo $f(\boldsymbol{x})$ por los valores que asigna a los N puntos de datos de entrenamiento. Es decir, calculamos el gradiente de la función de pérdida directamente con respecto al (vector de) valores $f(\boldsymbol{X}) = [f(\boldsymbol{x}_1) \cdots f(\boldsymbol{x}_N)]^\top$. Esto nos da un vector de gradiente de N-dimensiones

$$\nabla J\left(f_{\text{boost}}^{(k-1)}\right) \overset{\text{def}}{=} \left[\begin{array}{c} \frac{\partial J}{\partial f(\boldsymbol{x}_1)} \\ \vdots \\ \frac{\partial J}{\partial f(\boldsymbol{x}_N)} \end{array}\right]_{f=f_{\text{boost}}^{(k-1)}} = \frac{1}{N}\left[\begin{array}{c} \left.\frac{\partial L(y_1,f)}{\partial f}\right|_{f=f^{(k-1)}(\boldsymbol{x}_1)} \\ \vdots \\ \left.\frac{\partial L(y_n,f)}{\partial f}\right|_{f=f^{(b-1)}(\boldsymbol{x}_n)} \end{array}\right],$$

donde hemos asumido que la función de pérdida L es diferenciable. Por lo tanto, deberíamos seleccionar $f_k = -\nabla J\left(f_{\text{boost}}^{(k-1)}\right)$ y luego elegir el coeficiente α_k - que toma el papel de tasa de aprendizaje.

Sin embargo, seleccionar el k-ésimo miembro del ensemble f_k de manera que coincida exactamente con el negativo del gradiente generalmente no es posible. La razón es que los miembros del ensemble están restringidos a alguna forma funcional específica, por ejemplo, el conjunto de funciones que pueden ser representadas por un modelo basado en árboles de cierta profundidad.

La solución a este inconveniente es entrenar el k-ésimo miembro del ensemble f_k como un modelo de machine learning, con el objetivo de entrenamiento de que sus predicciones en los datos (es decir, el vector $f_k(\boldsymbol{X})$) estén cerca del negativo del gradiente. La cercanía puede ser evaluada por cualquier función de distancia adecuada, como la distancia al cuadrado. Esto corresponde a resolver un problema de regresión donde los valores objetivos son los elementos del gradiente, y la función de pérdida (por ejemplo MSE) determina cómo medimos la cercanía. Incluso cuando el problema en estudio sea de clasificación, los valores del gradiente serán generalmente valores reales.

Una vez encontrado el k-ésimo miembro del ensemble, queda por calcular el coeficiente α_k, que corresponde a la tasa de aprendizaje en el descenso por gradiente. En la versión más simple del descenso por gradiente, se considera un parámetro de ajuste. Sin embargo, también puede encontrarse resolviendo un problema de optimización unidimensional en cada iteración. En gradient boosting, se hace con mayor frecuencia de esta última manera. Si multiplicamos el α_k óptimo por una constante < 1, se obtiene un efecto regularizador que es útil en la práctica.

Algorithm 2 Entrenamiento del clasificador Gradient Boosting

Input: Datos de entrenamiento $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{N}$, multiplicador $\gamma < 1$.
Output: Clasificador $f_{\text{boost}}(\boldsymbol{x})$
Inicializar (constante), $f_{\text{boost}}^{(0)}(\boldsymbol{x}) = \arg\min_c \sum_{i=1}^{N} L(y_i, c)$.
for $k = 1, \ldots, K$ **do**
    Calcular el negativo del gradiente $d_i^{(k)} = -\frac{1}{N} \frac{\partial L(y_i, c)}{\partial c}\Big|_{c=f_{\text{boost}}^{(k-1)}(\boldsymbol{x}_i)}$.
    Entrenar un modelo de *regresión* $f_k(\boldsymbol{x})$ con $\left\{\left(\boldsymbol{x}_i, d_i^{(k)}\right)\right\}_{i=1}^{N}$.
    Hallar $\alpha_k = \arg\min_\alpha \sum_{i=1}^{N} L\left(y_i, f_{\text{boost}}^{(k-1)}(\boldsymbol{x}_i) + \alpha f_k(\boldsymbol{x}_i)\right)$.
    Actualizar el modelo $f_{\text{boost}}^{(k)}(\boldsymbol{x}) = f_{\text{boost}}^{(k-1)}(\boldsymbol{x}) + \gamma \alpha_k f_k(\boldsymbol{x})$.
end for