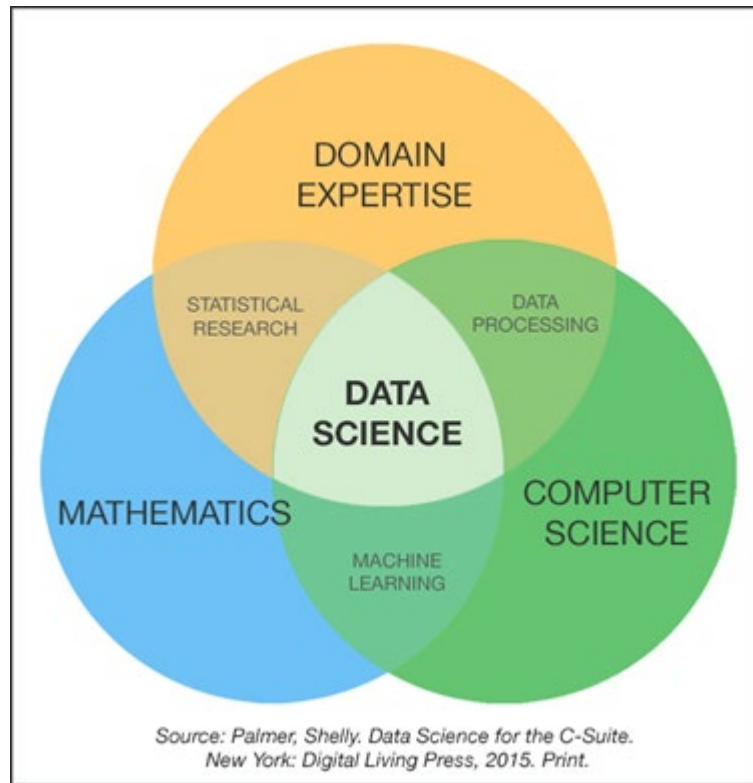




CIENCIA DE DATOS I



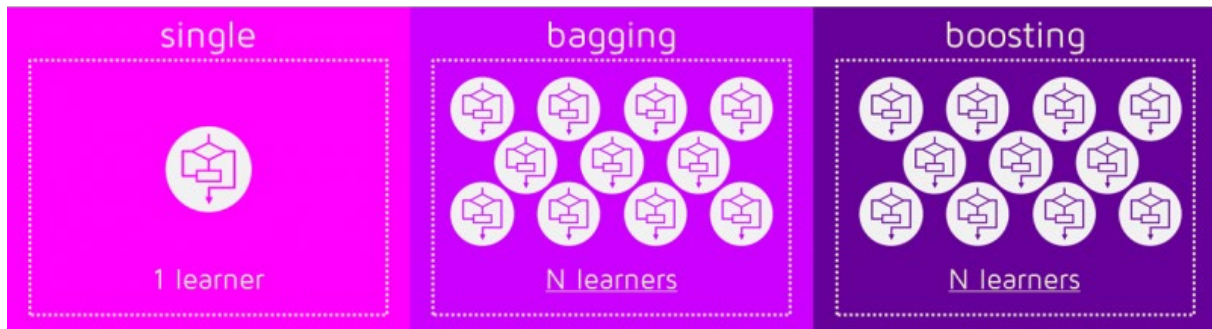
SEPARATA 05

METODOS ENSEMBLE

ÍNDICE

OBJETIVO	4
ENSEMBLE METHODS.....	5
BIAS VARIANZE TRADE-OFF	6
WEAK LEARNERS	7
WEAK LEARNERS	8
BOOTSTRAPING.....	9
BAGGING	10
BAGGING	11
RANDOM FOREST.....	12
BOOSTING	13
BOOSTING ADAPTATIVO	14
BOOSTING ADAPTATIVO	15
BOOSTING ADAPTATIVO	16
GRADIENT BOOSTING.....	17
GRADIENT BOOSTING.....	18
GRADIENT BOOSTING.....	19
STACKING.....	20
STACKING.....	21
STACKING.....	22
CONCLUSIONES	23

OBJETIVO



"La Unión hace la fuerza". Este viejo dicho expresa bastante bien la idea subyacente que rige los muy poderosos *"métodos ensemble"* en el aprendizaje automático. En términos generales, los métodos de aprendizaje por conjuntos, que a menudo llegan a las clasificaciones más altas de muchas competencias de aprendizaje automático (incluidas las competencias de Kaggle), se basan en la hipótesis de que la combinación de varios modelos a menudo puede producir un modelo mucho más poderoso.

El propósito de esta sesión es presentar varias nociones de aprendizaje ensemble. Daremos al estudiante algunas claves necesarias para comprender y utilizar bien los métodos relacionados y poder diseñar soluciones adaptadas cuando sea necesario. Discutiremos algunas nociones bien conocidas como bootstrapping, bagging, random forest, boosting, stacking y muchas otras que son la base del aprendizaje ensemble. Para que el vínculo entre todos estos métodos sea lo más claro posible, intentaremos presentarlos en un marco mucho más amplio y lógico que, esperamos, sea más fácil de entender y recordar.

En la primera parte de esta sesión, presentaremos las nociones de aprendices débiles y fuertes y presentaremos tres métodos principales de aprendizaje en conjunto: bagging, boosting y stacking.

Luego, en la segunda parte nos centraremos en el bagging y discutiremos nociones como bootstrap, agregating y bosques aleatorios. En la tercera parte, presentaremos el boosting y, en particular, sus dos variantes más populares: el boosting adaptativo (adaboost) y el boosting de gradiente. Finalmente, en la cuarta parte, daremos una descripción general del stacking.

ENSEMBLE METHODS

¿Qué son los métodos ensemble?

El aprendizaje ensemble es un paradigma de aprendizaje automático en el que se entrenan varios modelos (a menudo llamados "aprendices débiles") para resolver el mismo problema y se combinan para obtener mejores resultados. La hipótesis principal es que cuando los modelos débiles se combinan correctamente podemos obtener modelos más precisos y/o robustos.

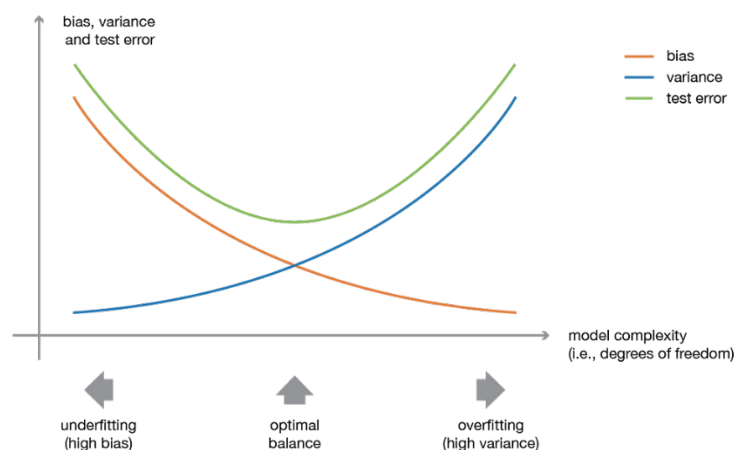
Aprendiz débil único:

En el aprendizaje automático, no importa si nos enfrentamos a un problema de clasificación o de regresión, la elección del modelo es extremadamente importante para tener alguna posibilidad de obtener buenos resultados. Esta elección puede depender de muchas variables del problema: cantidad de datos, dimensionalidad del espacio, hipótesis de distribución...

Un sesgo bajo y una varianza baja, aunque la mayoría de las veces varían en direcciones opuestas, son las dos características más fundamentales que se esperan de un modelo. De hecho, para poder "resolver" un problema, queremos que nuestro modelo tenga suficientes grados de libertad para resolver la complejidad subyacente de los datos con los que estamos trabajando, pero también queremos que no tenga demasiados grados de libertad para evitar alta varianza y ser más robusto. Este es el bien conocido tradeoff sesgo-varianza.

En el contexto de modelos estadísticos y de aprendizaje automático, el término "alto grado de libertad" se refiere a la capacidad de un modelo para ajustarse a los datos con flexibilidad y complejidad. Un modelo con alto grado de libertad tiene la capacidad de adaptarse y capturar patrones complejos en los datos de entrenamiento.

Cuando un modelo tiene alto grado de libertad, generalmente significa que tiene muchos parámetros o características que se pueden ajustar durante el proceso de entrenamiento. Esto permite que el modelo se ajuste de manera más ajustada y detallada a los datos de entrenamiento.



Recordemos el bias-variance trade-off, imaginemos que queremos lanzar una pelota a un cesto de basketball. El sesgo representa qué tan cerca estamos del objetivo promedio en nuestros lanzamientos. Si siempre lanzamos la pelota cerca del centro del cesto, tenemos un bajo sesgo. Pero si nuestros lanzamientos están lejos del cesto, tenemos un alto sesgo.

Por otro lado, la varianza representa qué tan dispersos están nuestros lanzamientos entre sí. Si lanzamos la pelota muy cerca del mismo punto en cada lanzamiento, tenemos una baja varianza. Pero si nuestros lanzamientos están muy dispersos y caen en diferentes lugares, tenemos una alta varianza.

BIAS VARIANCE TRADE-OFF

Ahora, aquí está el trade-off: cuando intentamos mejorar nuestra precisión, debemos considerar tanto el sesgo como la varianza. Si ajustamos nuestra técnica para lanzar la pelota más cerca del objetivo promedio (bajo sesgo), es posible que nuestros lanzamientos sean más dispersos (alta varianza). Del mismo modo, si intentamos reducir la dispersión de nuestros lanzamientos (baja varianza), es posible que nos alejemos del objetivo promedio (alto sesgo).

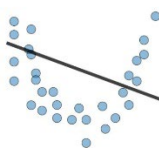


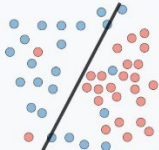
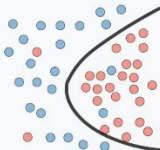
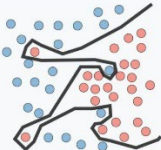



El objetivo es encontrar un equilibrio entre el sesgo y la varianza para obtener resultados óptimos. Queremos minimizar ambos, pero teniendo en cuenta que existe un balance entre ellos. Esto se conoce como trade-off entre sesgo y varianza. El trade-off entre el sesgo y la varianza está estrechamente relacionado con los conceptos de underfitting (subajuste) y overfitting (sobreajuste) en el aprendizaje automático.

El underfitting ocurre cuando un modelo es demasiado simple o no tiene suficiente capacidad para capturar los patrones subyacentes en los datos. En este caso, el modelo tiene un alto sesgo y una baja varianza. Básicamente, el modelo no puede ajustarse correctamente a los datos de entrenamiento ni capturar su complejidad. Como resultado, el modelo subajustado tendrá un rendimiento deficiente tanto en los datos de entrenamiento como en los nuevos datos.

Por otro lado, el overfitting ocurre cuando un modelo es demasiado complejo y se ajusta demasiado a los datos de entrenamiento, capturando incluso el ruido o la variabilidad aleatoria. En este caso, el modelo tiene un bajo sesgo, pero una alta varianza. Aunque el modelo puede ajustarse bien a los datos de entrenamiento, no generaliza bien a nuevos datos debido a su alta sensibilidad a las pequeñas variaciones. Esto puede resultar en un rendimiento deficiente en la predicción de nuevos datos.

El trade-off entre sesgo y varianza implica encontrar el punto óptimo entre underfitting y overfitting. Buscamos un modelo que sea lo suficientemente complejo como para capturar los patrones importantes en los datos (bajo sesgo), pero no tan complejo como para sobreajustarse y perder la capacidad de generalización (alta varianza).

En la práctica, se busca encontrar un equilibrio mediante técnicas como la validación cruzada, la selección del modelo adecuado o el ajuste de los hiperparámetros del modelo. El objetivo es minimizar tanto el sesgo como la varianza para obtener un modelo que se ajuste bien a los datos de entrenamiento y tenga una buena capacidad de generalización a nuevos datos.

	Underfitting	Just right	Overfitting
Symptoms	<ul style="list-style-type: none"> • High training error • Training error close to test error • High bias 	<ul style="list-style-type: none"> • Training error slightly lower than test error 	<ul style="list-style-type: none"> • Very low training error • Training error much lower than test error • High variance
Regression illustration			
Classification illustration			
Deep learning illustration			
Possible remedies	<ul style="list-style-type: none"> • Complexify model • Add more features • Train longer 		<ul style="list-style-type: none"> • Perform regularization • Get more data

WEAK LEARNERS

En la teoría del aprendizaje ensemble, llamamos modelos de aprendices débiles (o modelos base) a los modelos que se pueden usar como bloques de construcción para diseñar modelos más complejos combinando varios de ellos. La mayoría de las veces, estos modelos básicos no funcionan tan bien por sí mismos porque tienen un alto sesgo (modelos de bajo grado de libertad, por ejemplo) o porque tienen demasiada varianza para ser robustos (modelos de alto grado de libertad, por ejemplo). Entonces, la idea de los métodos de conjunto es tratar de reducir el sesgo y/o la varianza de estos aprendices débiles combinando varios de ellos para crear un aprendiz fuerte (o modelo de conjunto) que logre mejores actuaciones.

Combinar weak learners:

Para configurar un método de aprendizaje conjunto, primero debemos seleccionar nuestros modelos base para agregarlos. La mayoría de las veces (incluso en los conocidos métodos de bagging y boosting) se usa un algoritmo de aprendizaje de base único para que tengamos aprendices débiles homogéneos que se entrenan de diferentes maneras. El modelo de conjunto que obtenemos se dice entonces que es "homogéneo". Sin embargo, también existen algunos métodos que utilizan diferentes tipos de algoritmos básicos de aprendizaje: algunos aprendices débiles heterogéneos se combinan luego en un "modelo de conjuntos heterogéneos".

Un punto importante es que nuestra elección de aprendices débiles debe ser coherente con la forma en que agregamos estos modelos. Si elegimos modelos base con bajo sesgo, pero alta varianza, debe ser con un método de agregación que tienda a reducir la varianza, mientras que, si elegimos modelos base con baja varianza, pero alto sesgo, debe ser con un método de agregación que tienda a reducir el sesgo.

Esto nos lleva a la cuestión de cómo combinar estos modelos. Podemos mencionar tres tipos principales de meta-algoritmos que apuntan a combinar aprendices débiles:

bagging, que a menudo considera aprendices débiles homogéneos, aprenden independientemente unos de otros en paralelo y se combinan siguiendo algún tipo de proceso de promedio determinista

boosting, que a menudo considera aprendices débiles homogéneos, aprenden secuencialmente de forma muy adaptativa (un modelo base depende de los anteriores) y los combina siguiendo una estrategia determinista

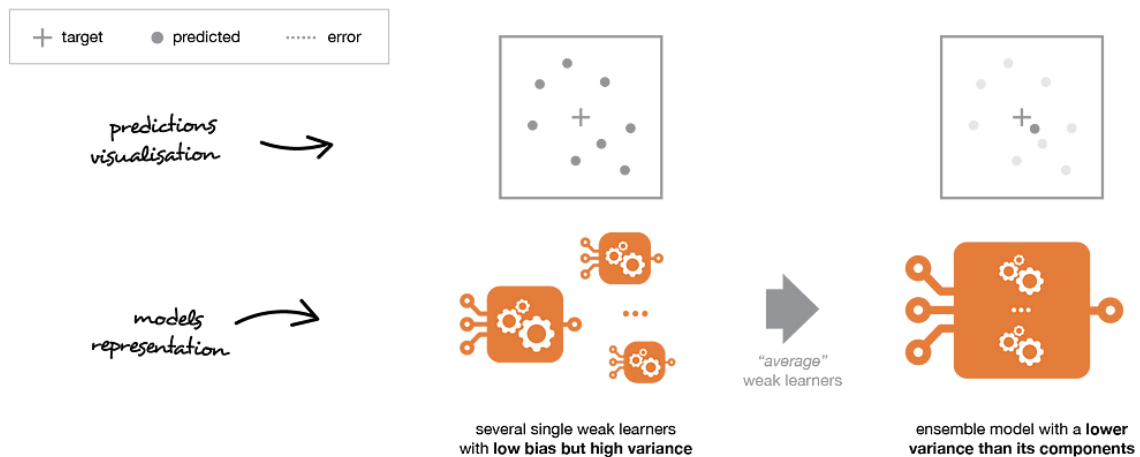
stacking, que a menudo considera aprendices débiles heterogéneos, aprenden en paralelo y se combinan entrenando un metamodelo para generar una predicción basada en las diferentes predicciones de modelos débiles

De manera muy aproximada, podemos decir que el bagging se centrará principalmente en obtener un modelo de conjunto con menos varianza que sus componentes, mientras que el boosting y el stacking intentarán principalmente producir modelos sólidos menos sesgados que sus componentes (incluso si la varianza también se puede reducir).

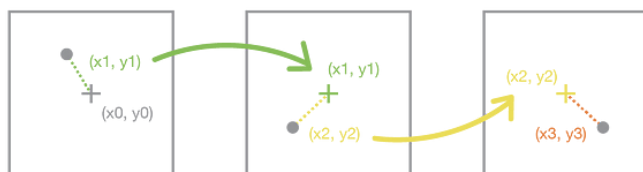
En las siguientes secciones, presentaremos en detalle el bagging y el boosting (que son un poco más utilizados que el stacking y nos permitirán analizar algunas nociones clave del aprendizaje ensemble) antes de dar una breve descripción general del stacking.

WEAK LEARNERS

Weak learners pueden combinarse para obtener un modelo con mejores performances. La forma de combinar los aprendices base debe adaptarse a sus tipos. Los **aprendices débiles de bajo sesgo y alta varianza** deben combinarse de manera que el **modelo ensemble** tenga menos varianza, mientras que los **aprendices débiles de baja varianza y alto sesgo** deben combinarse mejor de manera que el **modelo ensemble** sea menos sesgado.

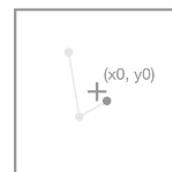


LOW BIAS HIGH VARIANCE WEAK LEARNERS



several single weak learners with **high bias but low variance**:
each model target the error of the previous one

LOW VARIANCE HIGH BIAS WEAK LEARNERS



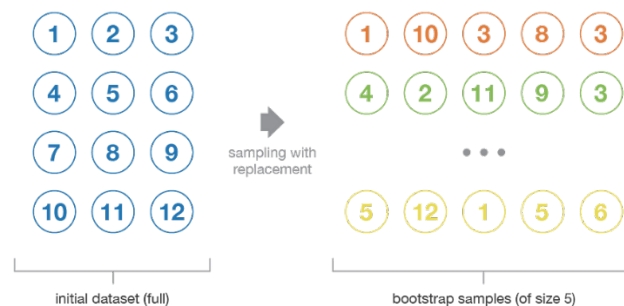
"compose"
weak learners

ensemble model with a **lower bias than its components**

BOOTSTRAPING

En los métodos paralelos, ajustamos a los diferentes aprendices considerados de forma independiente entre sí y, por lo tanto, es posible entrenarlos simultáneamente. El enfoque de este tipo más famoso es el "bagging" (que significa "Bootstrap aggregating") que tiene como objetivo producir un modelo de conjunto que sea más sólido que los modelos individuales que lo componen.

Empecemos definiendo bootstrapping. Esta técnica estadística consiste en generar muestras de tamaño B (llamadas muestras bootstrap) a partir de un conjunto de datos inicial de tamaño N mediante el sorteo aleatorio con observaciones B de reemplazo.



Bajo algunos supuestos, estas muestras tienen propiedades estadísticas bastante buenas: en una primera aproximación, se puede considerar que se extraen directamente de la verdadera distribución de datos subyacente (y, a menudo, desconocida) e independientemente unas de otras. Por lo tanto, pueden considerarse muestras representativas e independientes de la verdadera distribución de datos (casi muestras i.i.d. independiente e idénticamente distribuida). Las hipótesis que hay que verificar para que esta aproximación sea válida son dos. Primero, el tamaño N del conjunto de datos inicial debe ser lo suficientemente grande como para capturar la mayor parte de la complejidad de la distribución subyacente, de modo que el muestreo del conjunto de datos sea una buena aproximación del muestreo de la distribución real (**representatividad**). En segundo lugar, el tamaño N del conjunto de datos debe ser lo suficientemente grande en comparación con el tamaño B de las muestras de bootstrap para que las muestras no estén demasiado correlacionadas (**independencia**). Nótese que a continuación, algunas veces haremos referencia a estas propiedades (**representatividad e independencia**) de las muestras bootstrap: debemos siempre tener en cuenta que esto es solo una aproximación.

Las muestras bootstrap se utilizan a menudo, por ejemplo, para evaluar la varianza o los intervalos de confianza de los estimadores estadísticos. Por definición, un estimador estadístico es una función de algunas observaciones y, por lo tanto, una variable aleatoria con varianza proveniente de estas observaciones. Para estimar la varianza de dicho estimador, necesitamos evaluarlo en varias muestras independientes extraídas de la distribución de interés.

En la mayoría de los casos, considerar muestras verdaderamente independientes requeriría demasiados datos en comparación con la cantidad realmente disponible. Luego podemos usar el bootstrapping para generar varias muestras bootstrap que pueden considerarse "casi representativas" y "casi independientes" (muestras casi i.i.d.). Estas muestras bootstrap nos permitirán aproximar la varianza del estimador, evaluando su valor para cada una de ellas.

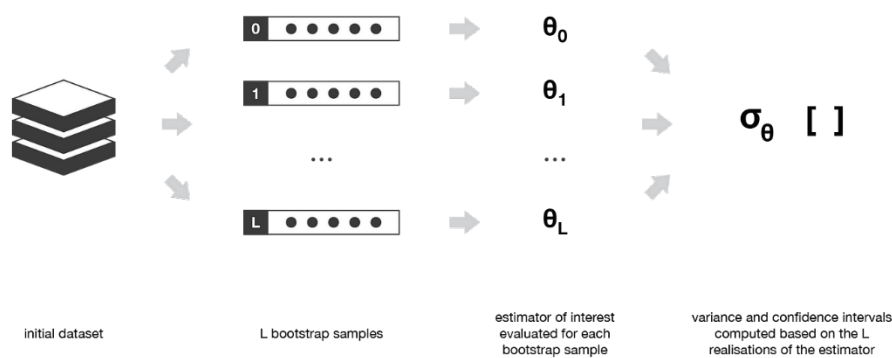
Una muestra i.i.d (independiente e idénticamente distribuida) se refiere a un conjunto de datos u observaciones que se asume que son independientes entre sí y se distribuyen de la misma manera.

Independiente significa que la ocurrencia o el valor de una observación no afecta la ocurrencia o el valor de otra observación. Cada muestra se considera no relacionada con las demás y no hay una relación inherente o correlación entre ellas.

BAGGING

Idénticamente distribuida significa que cada muestra se extrae de la misma distribución de probabilidad. Esta suposición implica que todas las muestras provienen de la misma población y comparten las mismas propiedades estadísticas. En otras palabras, la distribución de los datos es consistente en todas las muestras.

La suposición de muestras i.i.d se hace a menudo en diversas técnicas estadísticas y de aprendizaje automático. Simplifica el análisis y permite la aplicación de ciertos modelos matemáticos y pruebas estadísticas. Sin embargo, es importante tener en cuenta que, en muchos escenarios del mundo real, la suposición de independencia y distribución idéntica puede no cumplirse, y pueden ser necesarias técnicas más complejas para manejar datos correlacionados o no idénticos.



Bootstrapping se utiliza a menudo para evaluar la varianza o el intervalo de confianza de algunos estimadores estadísticos.

Al entrenar un modelo, no importa si se trata de un problema de clasificación o de regresión, modelamos una función que toma una entrada, devuelve una salida y que se define con respecto al conjunto de datos de entrenamiento. Debido a la varianza teórica del conjunto de datos de entrenamiento (recordamos que un conjunto de datos es una muestra observada procedente de una distribución subyacente verdadera desconocida), el modelo ajustado también está sujeto a variabilidad: ***si se hubiera observado otro conjunto de datos, habríamos obtenido un modelo diferente.***

La idea de bagging es entonces simple: queremos ajustar varios modelos independientes y "promediar" sus predicciones para obtener un modelo con una varianza más baja. Sin embargo, en la práctica, no podemos ajustar modelos completamente independientes porque requeriría demasiados datos. Por lo tanto, confiamos en las buenas "propiedades aproximadas" de las muestras Bootstrap (representatividad e independencia) para ajustar modelos que son casi independientes.

Primero, creamos múltiples muestras Bootstrap para que cada nueva muestra Bootstrap actúe como otro conjunto de datos (casi) independiente extraído de la distribución real. Luego, podemos ajustar un aprendiz débil para cada una de estas muestras y finalmente agregarlos de manera que podamos "promediar" sus resultados y, por lo tanto, obtener un modelo de conjunto con menos variación que sus componentes. En términos generales, como las muestras bootstrap son aproximadamente independientes e idénticamente distribuidas (i.i.d.), también lo son los modelos base aprendidos. Entonces, "promediar" los resultados de los estudiantes débiles no cambia la respuesta esperada, pero reduce su varianza (al igual que promediar las variables aleatorias i.i.d. conservan el valor esperado, pero reducen la varianza).

Entonces, suponiendo que tenemos L muestras Bootstrap (aproximaciones de L conjuntos de datos independientes) de tamaño B denotado:

$$\{z_1^1, z_2^1, \dots, z_B^1\}, \{z_1^2, z_2^2, \dots, z_B^2\}, \dots, \{z_1^L, z_2^L, \dots, z_B^L\} \quad z_b^l \equiv b\text{-th observation of the } l\text{-th bootstrap sample}$$

BAGGING

Podemos adaptar L aprendices débiles casi independientes (uno en cada conjunto de datos)

$$w_1(\cdot), w_2(\cdot), \dots, w_L(\cdot)$$

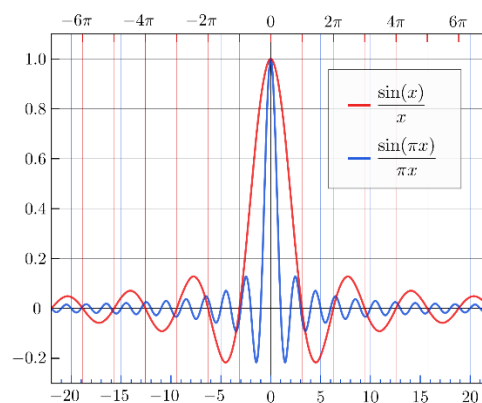
y luego agregarlos en algún tipo de proceso de promedio para obtener un modelo de conjunto con una varianza más baja. Por ejemplo, podemos definir nuestro modelo fuerte tal que:

$$s_L(\cdot) = \frac{1}{L} \sum_{l=1}^L w_l(\cdot) \quad (\text{simple average, for regression problem})$$

$$s_L(\cdot) = \arg \max_k [\text{card}(\{w_l(\cdot) = k\})] \quad (\text{simple majority vote, for classification problem})$$

En matemáticas, los argumentos del máximo y el mínimo (abreviados como $\arg \max/\arg \max$; $\arg \min/\arg \min$) son los puntos del dominio de una función en la cual los valores de la función son maximizados o minimizados.

En contraste al máximo global, refiriéndose a las salidas más grandes, el $\arg \max$ se refiere a las entradas o argumentos en que las salidas de la función son lo más grandes posible.

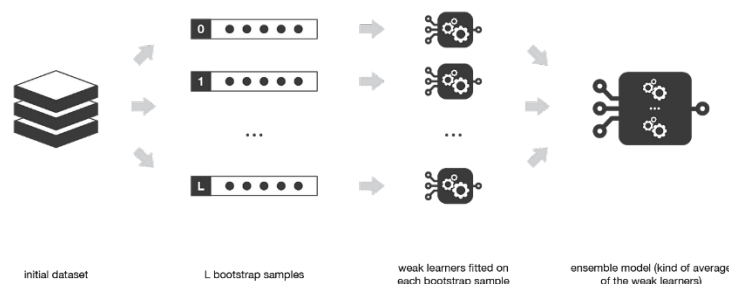


Como ejemplo, las funciones sinc normalizadas y sin normalizar aquí arriba tienen un $\arg \max$ de $\{0\}$, porque ambas alcanzan su valor global máximo de 1 en $x=0$.

La función sinc sin normalizar (roja) tiene un $\arg \min$ de aproximadamente $\{-4.49; 4.49\}$, porque tiene dos valores globales mínimos de aproximadamente -0.217 en $x = \pm 4.49$

Sin embargo, la función sinc normalizada (azul) tiene un $\arg \min$ de $\{-1.43; 1.43\}$ aproximadamente, ya que su mínimo global ocurre en $x = \pm 1.43$, aunque el valor mínimo sea el mismo.

El bagging consiste en ajustar varios modelos base en diferentes muestras bootstrap y construir un modelo de conjunto que "promedie" los resultados de estos aprendices débiles.

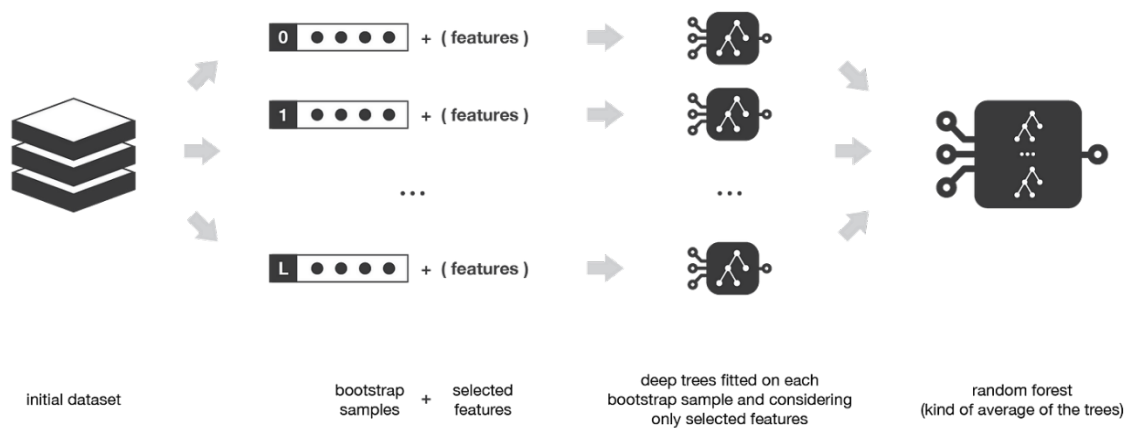


RANDOM FOREST

Los árboles de aprendizaje son modelos base muy populares para métodos de conjuntos. Los aprendices fuertes compuestos de múltiples árboles pueden llamarse "bosques". Los árboles que componen un bosque se pueden elegir para que sean poco profundos (pocas profundidades) o profundos (muchas profundidades, si no están completamente desarrollados). Los árboles *poco profundos tienen menos varianza, pero un mayor sesgo* y, por lo tanto, serán una mejor opción para los **métodos secuenciales** que describiremos a continuación. Los árboles profundos, por otro lado, *tienen un sesgo bajo, pero una varianza alta* y, por lo tanto, son opciones relevantes para el método de **bagging** que se enfoca principalmente en reducir la varianza.

El enfoque de bosque aleatorio es un método de bagging donde los árboles profundos, ajustados en muestras bootstrap, se combinan para producir una salida con una varianza más baja. Sin embargo, los bosques aleatorios también usan otro truco para hacer que los árboles ajustados múltiples estén un poco menos correlacionados entre sí: al hacer crecer cada árbol, en lugar de solo muestrear las observaciones en el conjunto de datos para generar una muestra de bootstrap, también muestreamos las features y mantenemos solo un subconjunto aleatorio de ellos para construir el árbol.

De hecho, el muestreo de features tiene el efecto de que todos los árboles no miran exactamente la misma información para tomar sus decisiones y, por lo tanto, reduce la correlación entre los diferentes resultados devueltos. Otra ventaja del muestreo sobre las features es que hace que el proceso de toma de decisiones sea más sólido frente a los datos faltantes: las observaciones (del conjunto de datos de entrenamiento o no) con datos faltantes aún se pueden realizar una regresión o clasificarse en función de los árboles que tienen en cuenta solo las features donde no faltan datos. Por lo tanto, el algoritmo de bosque aleatorio combina los conceptos de bagging y selección de subespacio de features aleatorias para crear modelos más robustos.



El método de bosque aleatorio es un método de bagging con árboles como aprendices débiles. Cada árbol se ajusta a una muestra de bootstrap considerando solo un subconjunto de features elegidas al azar.

BOOSTING

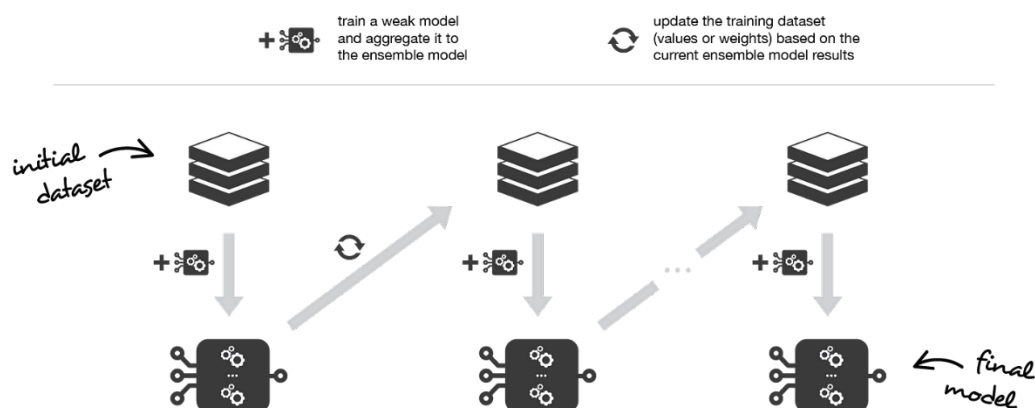
En los métodos secuenciales, los diferentes modelos débiles combinados ya no se ajustan independientemente unos de otros. La idea es ajustar los modelos iterativamente de modo que el entrenamiento del modelo en un paso dado dependa de los modelos ajustados en los pasos anteriores. El "boosting" es el más famoso de estos enfoques y produce un modelo de conjunto que, en general, está menos sesgado que los aprendices débiles que lo componen.

Los métodos de boosting funcionan con el mismo espíritu que los métodos de bagging: construimos una familia de modelos que se agregan para obtener un aprendiz fuerte que se desempeñe mejor. Sin embargo, a diferencia del bagging que apunta principalmente a reducir la varianza, el boosting es una técnica que consiste en ajustar secuencialmente múltiples aprendices débiles de una manera muy adaptativa: *cada modelo en la secuencia se ajusta dando más importancia a las observaciones en el conjunto de datos que fueron mal manejadas por los modelos anteriores en la secuencia*. Intuitivamente, cada nuevo modelo centra sus esfuerzos en las observaciones más difíciles de ajustar hasta el momento, de modo que al final del proceso obtengamos un aprendiz fuerte con un sesgo más bajo (incluso si podemos notar que el boosting también puede tener el efecto de reducir la varianza). El boosting, como el bagging, se puede utilizar para la regresión y también para problemas de clasificación.

Al centrarse principalmente en reducir el sesgo, los modelos base que a menudo se consideran boosting son modelos con baja varianza, pero alto sesgo. Por ejemplo, si queremos usar árboles como nuestros modelos base, elegiremos la mayoría de las veces árboles de decisión poco profundos con solo unas pocas profundidades. Otra razón importante que motiva el uso de modelos de baja varianza, pero alto sesgo como aprendices débiles para boosting es que estos modelos son en general menos costosos computacionalmente para adaptarse (pocos grados de libertad cuando están parametrizados). De hecho, dado que los cálculos para ajustar los diferentes modelos no se pueden realizar en paralelo (a diferencia del bagging), podría resultar demasiado costoso ajustar secuencialmente varios modelos complejos.

Una vez que se han elegido los aprendices débiles, todavía tenemos que definir cómo se ajustarán secuencialmente (¿qué información de los modelos anteriores tenemos en cuenta al ajustar el modelo actual?) y cómo se agregarán (¿cómo agregamos el modelo actual? a los anteriores?). Discutiremos estas preguntas en las dos subsecciones siguientes, describiendo más especialmente dos importantes algoritmos de boosting: adaboost y gradiente de boosting.

En pocas palabras, estos dos meta-algoritmos difieren en cómo crean y agregan a los aprendices débiles durante el proceso secuencial. El boosting adaptativo actualiza los pesos asociados a cada una de las observaciones del conjunto de datos de entrenamiento, mientras que el boosting de gradiente actualiza el valor de estas observaciones. Esta principal diferencia proviene de la forma en que ambos métodos intentan resolver el problema de optimización de encontrar el mejor modelo que se pueda escribir como una suma ponderada de aprendices débiles.



El boosting consiste en, iterativamente, ajustar un aprendiz débil, agregarlo al modelo de conjunto y "actualizar" el conjunto de datos de entrenamiento para tener mejor en cuenta las fortalezas y debilidades del modelo de conjunto actual al ajustar el siguiente modelo base.

BOOSTING ADAPTATIVO

En el boosting adaptativo (a menudo llamado "adaboost"), tratamos de definir nuestro modelo de conjunto como una suma ponderada de L aprendices débiles:

$$s_L(.) = \sum_{l=1}^L c_l \times w_l(.) \quad \text{where } c_l\text{'s are coefficients and } w_l\text{'s are weak learners}$$

Encontrar el mejor modelo de conjunto con esta forma es un problema de optimización difícil. Luego, en lugar de tratar de resolverlo de una sola vez (encontrar todos los coeficientes y los aprendices débiles que dan el mejor modelo aditivo general), hacemos uso de un proceso de optimización iterativo que es mucho más manejable, incluso si puede conducir a una solución subóptima. Más especialmente, agregamos los aprendices débiles uno por uno, buscando en cada iteración el mejor par posible (coeficiente, aprendiz débil) para agregar al modelo de conjunto actual. En otras palabras, definimos recurrentemente los (s_l) de tal manera que:

$$s_l(.) = s_{l-1}(.) + c_l \times w_l(.)$$

Explicación Completa de AdaBoost

AdaBoost (Adaptive Boosting) es un poderoso algoritmo de ensemble que combina múltiples modelos débiles para formar un modelo fuerte y robusto. A continuación, explicaré cómo funciona AdaBoost paso a paso, incorporando todos los conceptos revisados, incluyendo la inicialización de pesos, la iteración de entrenamiento, el cálculo de errores y coeficientes de actualización, y la actualización y normalización de pesos.

1. Inicialización

1. Pesos Iniciales:

- Al comienzo del algoritmo, todas las observaciones en el conjunto de datos tienen pesos iguales.
- Si hay N observaciones, el peso inicial de cada observación es $w_i = \frac{1}{N}$.

2. Iteración del Proceso (Repetir L Veces)

Para cada iteración l de las L iteraciones, se realizan los siguientes pasos:

Paso 1: Entrenar el Modelo Débil

- **Seleccionar y Entrenar:**
 - Entrenamos un modelo débil $h_l(x)$ utilizando los pesos actuales w_i de las observaciones.
 - Un modelo débil puede ser un stump (árbol de decisión simple), una regresión logística, etc.
 - Los pesos influyen en el entrenamiento de manera que las observaciones con mayor peso tienen más influencia en el ajuste del modelo.

Paso 2: Evaluar el Modelo Débil

- **Calcular el Error del Modelo:**
 - Calculamos el error del modelo débil ϵ_l :

$$\epsilon_l = \sum_{i=1}^N w_i \cdot I(y_i \neq h_l(x_i))$$

- Donde I es la función indicadora que es 1 si la predicción es incorrecta y 0 si es correcta.

BOOSTING ADAPTATIVO

Paso 3: Calcular el Coeficiente de Actualización

- Coeficiente de Actualización (α_l):
 - Calculamos el coeficiente de actualización:

$$\alpha_l = \frac{1}{2} \ln \left(\frac{1 - \epsilon_l}{\epsilon_l} \right)$$

- Este coeficiente refleja la confianza en el modelo débil: un menor error ϵ_l conduce a un mayor α_l .

Paso 4: Actualizar el Modelo Fuerte

- Modelo Fuerte:
 - Actualizamos el modelo fuerte $F_l(x)$ combinando el nuevo modelo débil $h_l(x)$ ponderado por su coeficiente:

$$F_l(x) = F_{l-1}(x) + \alpha_l \cdot h_l(x)$$

- Donde $F_{l-1}(x)$ es el modelo fuerte hasta la iteración anterior.

Paso 5: Actualizar Pesos de las Observaciones

- Nuevos Pesos:
 - Actualizamos los pesos de las observaciones para la próxima iteración. Las observaciones que fueron mal clasificadas por el modelo actual obtienen más peso:

$$w_i^{(l+1)} = w_i^{(l)} \cdot \exp(\alpha_l \cdot I(y_i \neq h_l(x_i)))$$

- La función exponencial $\exp(\alpha_l \cdot I(y_i \neq h_l(x_i)))$ aumenta los pesos de las observaciones mal clasificadas de forma exponencial, lo que les da mayor importancia en la siguiente iteración.
- Normalización:
 - Los nuevos pesos se normalizan para asegurar que sumen 1:

$$w_i^{(l+1)} = \frac{w_i^{(l+1)}}{\sum_{j=1}^N w_j^{(l+1)}}$$

- Esto asegura que los pesos se mantengan en una escala consistente y que la influencia de cada observación se distribuya adecuadamente en cada iteración.

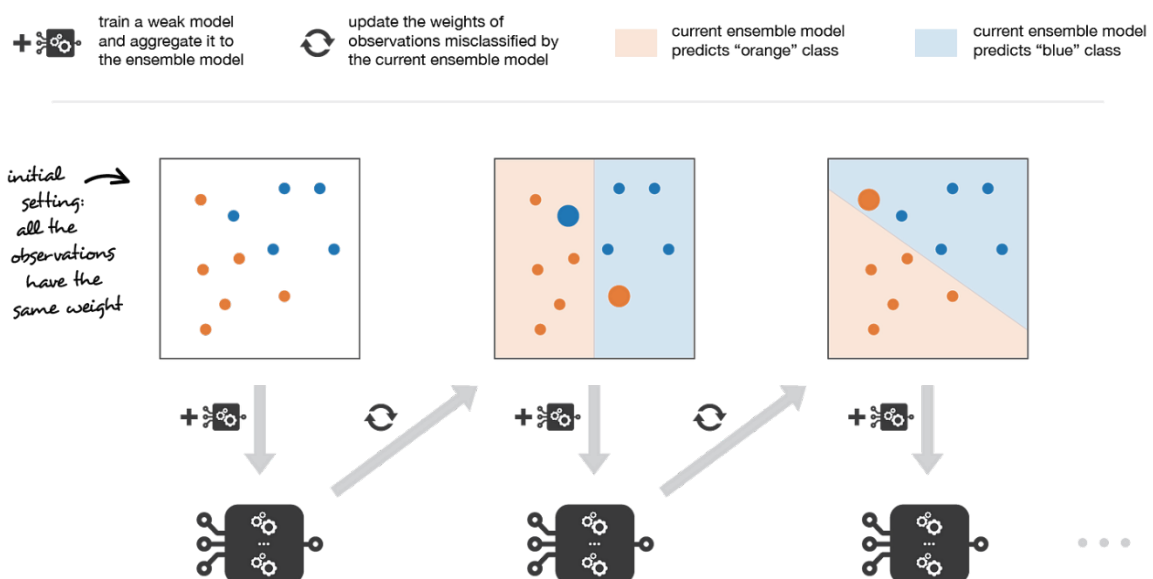
Más especialmente, al considerar una clasificación binaria, podemos mostrar que el algoritmo adaboost se puede reescribir en un proceso que procede de la siguiente manera. Primero, actualiza los pesos de las observaciones en el conjunto de datos y capacita a un nuevo aprendiz débil con un enfoque especial dado a las observaciones mal clasificadas por el modelo de conjunto actual. En segundo lugar, agrega el aprendiz débil a la suma ponderada de acuerdo con un coeficiente de actualización que expresa los desempeños de este modelo débil: cuanto mejor se desempeña un aprendiz débil, más contribuye al aprendiz fuerte.

BOOSTING ADAPTATIVO

Entonces, suponga que enfrentamos un problema de clasificación binaria, con N observaciones en nuestro conjunto de datos y queremos usar el algoritmo adaboost con una familia dada de modelos débiles. Al principio del algoritmo (primer modelo de la secuencia), todas las observaciones tienen los mismos pesos $1/N$. Luego, repetimos L veces (para los L aprendices en la secuencia) los siguientes pasos:

- Entrenar el mejor modelo débil posible con los pesos de las observaciones actuales
- Calcule el error del modelo débil
- Calcule el valor del coeficiente de actualización que es algún tipo de métrica de evaluación escalar del aprendiz débil que indica cuánto se debe tener en cuenta este aprendiz débil en el modelo de ensemble
- Actualice el aprendiz fuerte agregando el nuevo aprendiz débil multiplicado por su coeficiente de actualización
- Actualizar y normalizar los pesos de las observaciones para centrarnos en las que más peso tienen en siguiente iteración (las ponderaciones de las observaciones predichas incorrectamente por el modelo agregado aumentan y las ponderaciones de las observaciones predichas correctamente disminuyen)

Repitiendo estos pasos, construimos secuencialmente nuestros modelos L y los agregamos en una combinación lineal simple ponderada por coeficientes que expresan el desempeño de cada aprendiz. Tenga en cuenta que existen variantes del algoritmo adaboost inicial, como LogitBoost (clasificación) o L2Boost (regresión), que se diferencian principalmente por la elección de la función de pérdida.



GRADIENT BOOSTING

En el boosting de gradiente, el modelo de conjunto que tratamos de construir también es una suma ponderada de aprendices débiles:

$$s_L(\cdot) = \sum_{l=1}^L c_l \times w_l(\cdot) \quad \text{where } c_l\text{'s are coefficients and } w_l\text{'s are weak learners}$$

Tal como mencionamos para adaboost, encontrar el modelo óptimo bajo esta forma es demasiado difícil y se requiere un enfoque iterativo. La principal diferencia con el boosting adaptativo está en la definición del proceso de optimización secuencial. De hecho, el aumento de gradiente convierte el problema en uno de descenso de gradiente: en cada iteración ajustamos un aprendiz débil al opuesto del gradiente del error de ajuste actual con respecto al modelo de conjunto actual. Intentemos aclarar este último punto. Primero, el proceso teórico de descenso del gradiente sobre el modelo de conjunto se puede escribir

$$s_l(\cdot) = s_{l-1}(\cdot) - c_l \times \nabla_{s_{l-1}} E(s_{l-1})(\cdot)$$

donde $E(\cdot)$ es el error de ajuste del modelo dado, c_l es un coeficiente correspondiente al tamaño del paso y

$$-\nabla_{s_{l-1}} E(s_{l-1})(\cdot)$$

es el opuesto del gradiente del error de ajuste con respecto al modelo de conjunto en el paso $l-1$. Este opuesto (bastante abstracto) del gradiente es una función que, en la práctica, solo puede evaluarse para observaciones en el conjunto de datos de entrenamiento (para el cual conocemos entradas y salidas): estas evaluaciones se denominan pseudo-residuales adjuntos a cada observación. Además, incluso si conocemos por las observaciones los valores de estos pseudo-residuales, no queremos agregar a nuestro modelo de conjunto ningún tipo de función: solo queremos agregar una nueva instancia de modelo débil. Por lo tanto, lo natural que se debe hacer es ajustar un aprendiz débil a los pseudo-residuales calculados para cada observación. Finalmente, el coeficiente c_l se calcula siguiendo un proceso de optimización unidimensional (búsqueda de línea para obtener el mejor tamaño de paso c_l).

Entonces, supongamos que queremos usar la técnica de aumento de gradiente con una familia dada de modelos débiles. Al principio del algoritmo (primer modelo de la secuencia), los pseudo-residuales se igualan a los valores de observación. Luego, repetimos L veces (para los L modelos de la secuencia) los siguientes pasos:

- Ajuste el mejor estudiante débil posible a los pseudo-residuales (aproximadamente el opuesto del gradiente con respecto al estudiante fuerte actual)
- Calcular el valor del tamaño de paso óptimo que define cuánto actualizamos el modelo de conjunto en la dirección del nuevo aprendiz débil
- Actualice el modelo de conjunto agregando el nuevo aprendiz débil multiplicado por el tamaño del paso (haga un paso de descenso de gradiente)
- Calcular nuevos pseudo-residuales que indican, para cada observación, en qué dirección nos gustaría actualizar las próximas predicciones del modelo de conjunto

Repetiendo estos pasos, construimos secuencialmente nuestros modelos L y los agregamos siguiendo un enfoque de descenso de gradiente. Tenga en cuenta que, mientras que el impulso adaptativo intenta resolver en cada iteración exactamente el problema de optimización "local" (encontrar el mejor aprendiz débil y su coeficiente para agregar al modelo fuerte), el impulso de gradiente utiliza en su lugar un enfoque de descenso de gradiente y se puede adaptar más fácilmente a un gran número de funciones de pérdida. Por lo tanto, el aumento de gradiente se puede considerar como una generalización de adaboost a funciones de pérdida diferenciables arbitrarias.

GRADIENT BOOSTING

Explicación del Boosting de Gradiente de Manera Sencilla

Gradient Boosting es una técnica de machine learning que combina muchos modelos débiles para formar un modelo fuerte y preciso. Se utiliza para problemas de regresión y clasificación. Vamos a desglosar el proceso paso a paso de una manera sencilla:

Conceptos Básicos

1. **Modelo Débil:** Un modelo simple que, por sí solo, tiene un rendimiento ligeramente mejor que el azar. Un ejemplo común es un árbol de decisión simple.
2. **Residuos:** La diferencia entre los valores observados y las predicciones del modelo. En cada iteración, se ajusta un nuevo modelo débil a estos residuos para mejorar la predicción.
3. **Función de Pérdida:** Mide el error del modelo. Gradient Boosting busca minimizar esta función para mejorar el rendimiento del modelo.
4. **Gradiente:** La dirección en la que se reduce más rápidamente la función de pérdida. Cada iteración ajusta un nuevo modelo débil en la dirección del gradiente de la función de pérdida.

Paso a Paso del Gradient Boosting

Paso 1: Inicialización

- Comienza con un modelo inicial simple $F_0(x)$, que puede ser una constante. Por ejemplo, en un problema de regresión, podría ser la media de los valores objetivo:

$$F_0(x) = \frac{1}{N} \sum_{i=1}^N y_i$$

Paso 2: Iteraciones del Proceso

En cada iteración m de un total de M iteraciones, se realizan los siguientes pasos:

1. Cálculo de los Residuos:

- Los residuos $r_i^{(m)}$ representan los errores de predicción del modelo actual $F_{m-1}(x)$:

$$r_i^{(m)} = y_i - F_{m-1}(x_i)$$

- Estos residuos son las diferencias entre los valores observados y_i y las predicciones actuales $F_{m-1}(x_i)$.

2. Entrenamiento del Modelo Débil:

- Entrena un nuevo modelo débil $h_m(x)$ utilizando los residuos $r_i^{(m)}$ como etiquetas objetivo. Este modelo intenta predecir los residuos.

3. Cálculo del Coeficiente de Actualización:

- Encuentra el coeficiente de actualización γ_m que minimice la función de pérdida. Para problemas de regresión con error cuadrático medio, se calcula así:

$$\gamma_m = \frac{\sum_{i=1}^N r_i^{(m)} h_m(x_i)}{\sum_{i=1}^N h_m(x_i)^2}$$

4. Actualización del Modelo Fuerte:

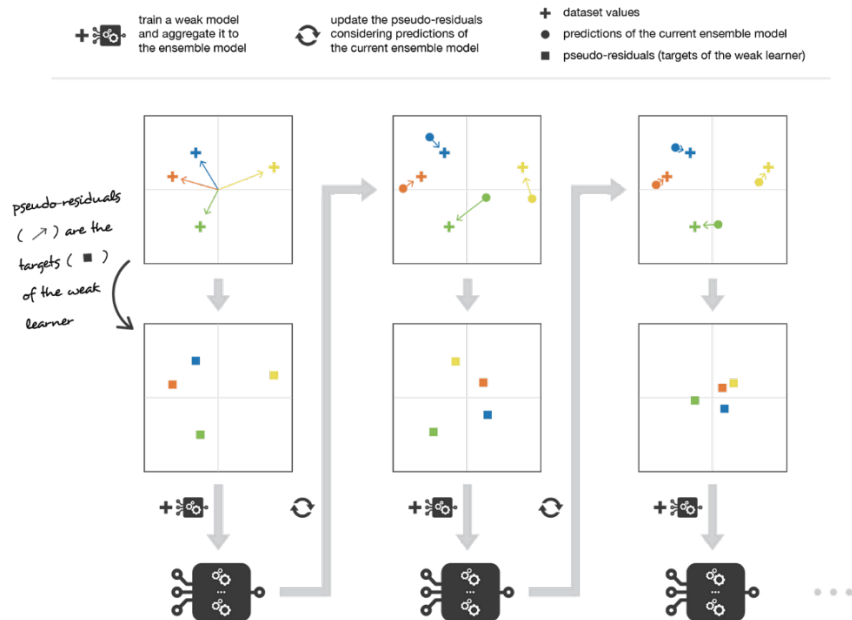
- Actualiza el modelo fuerte combinando el modelo actual y el nuevo modelo débil ponderado por γ_m :

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$$

5. Repetición:

- Repite los pasos 1-4 hasta completar M iteraciones. En cada iteración, se reduce el error residual mejorando la precisión del modelo.

GRADIENT BOOSTING



El gradient boosting actualiza los valores de las observaciones en cada iteración. Los aprendices débiles están capacitados para adaptarse a los pseudo-residuales que indican en qué dirección corregir las predicciones del modelo de conjunto actual para reducir el error.

STACKING

El stacking se diferencia principalmente del bagging y el boosting en dos puntos. En primer lugar, el stacking a menudo considera aprendices débiles heterogéneos (se combinan diferentes algoritmos de aprendizaje), mientras que el bagging y el boosting consideran principalmente estudiantes débiles homogéneos. En segundo lugar, el stacking aprende a combinar los modelos base utilizando un metamodelo, mientras que el bagging y el boosting combinan aprendices débiles siguiendo algoritmos deterministas.

Como ya mencionamos, la idea de apilar es aprender varios aprendices débiles diferentes y combinarlos entrenando un metamodelo para generar predicciones basadas en las múltiples predicciones devueltas por estos modelos débiles. Entonces, necesitamos definir dos cosas para construir nuestro modelo de stacking: los estudiantes L que queremos ajustar y el metamodelo que los combina.

Por ejemplo, para un problema de clasificación, podemos elegir como aprendices débiles un clasificador KNN, una regresión logística y una SVM, y decidir aprender una red neuronal como metamodelo. Luego, la red neuronal tomará como entradas las salidas de nuestros tres aprendices débiles y aprenderá a devolver predicciones finales basadas en ello.

Entonces, supongamos que queremos ajustar un conjunto de stacking compuesto por L aprendices débiles. Entonces tenemos que seguir los pasos a continuación:

Explicación del Stacking (Stalking) de Un Nivel y Varios Niveles

Stacking, también conocido como **Stacked Generalization**, es una técnica de ensemble en machine learning que combina múltiples modelos base mediante un modelo meta que aprende a combinar las predicciones de estos modelos base para mejorar la precisión general.

Stacking de Un Nivel

El stacking de un nivel es el enfoque básico en el que se combinan los modelos base a través de un único modelo meta. Aquí está el proceso detallado:

Paso a Paso del Stacking de Un Nivel

1. Entrenamiento de los Modelos Base

Ecuación de los Modelos Base:

$$h_j(x) = \text{Modelo base } j \text{ entrenado con el conjunto de datos } (X, y)$$

Donde h_j representa el j -ésimo modelo base y (X, y) es el conjunto de datos de entrenamiento.

2. Generación de Predicciones

Predicciones de los Modelos Base:

$$z_j = h_j(X)$$

Estas predicciones se usan como nuevas características para el modelo meta:

$$Z = [z_1, z_2, \dots, z_J]$$

Donde Z es la matriz de predicciones de los J modelos base.

3. Entrenamiento del Modelo Meta

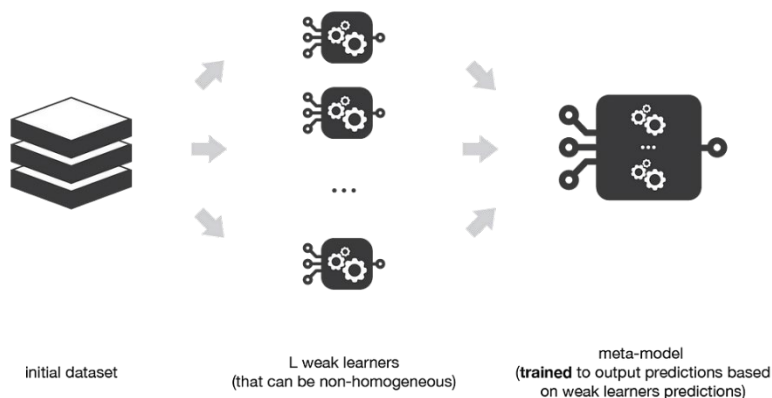
Ecuación del Modelo Meta:

$$g(Z) = \text{Modelo meta entrenado con las predicciones } Z \text{ y las etiquetas originales } y$$

Donde g representa el modelo meta.

STACKING

En los pasos anteriores, dividimos el conjunto de datos en dos porque las predicciones sobre los datos que se han utilizado para el entrenamiento de los aprendices débiles no son relevantes para el entrenamiento del metamodelo. Por lo tanto, un inconveniente obvio de esta división de nuestro conjunto de datos en dos partes es que solo tenemos la mitad de los datos para entrenar los modelos base y la mitad de los datos para entrenar el metamodelo. Sin embargo, para superar esta limitación, podemos seguir algún tipo de enfoque de "entrenamiento cruzado de k-pliegues" (similar a lo que se hace en la validación cruzada de k-pliegues) de modo que todas las observaciones puedan usarse para entrenar el meta- modelo: para cualquier observación, la predicción de los aprendices débiles se realiza con instancias de estos aprendices débiles entrenados en los pliegues k-1 que no contienen la observación considerada. En otras palabras, consiste en entrenar en el pliegue k-1 para hacer predicciones en el pliegue restante y eso iterativamente para obtener predicciones para las observaciones en cualquier pliegue. Al hacerlo, podemos producir predicciones relevantes para cada observación de nuestro conjunto de datos y luego entrenar nuestro metamodelo en todas estas predicciones.



Stacking de varios niveles:

Una posible extensión del stacking es el stacking de varios niveles. Consiste en hacer stacking con múltiples capas. Como ejemplo, consideremos un apilamiento de 3 niveles. En el primer nivel (capa), encajamos los L aprendices débiles que se han elegido. Luego, en el segundo nivel, en lugar de ajustar un único metamodelo a las predicciones de los modelos débiles (como se describió en la subsección anterior), ajustamos M de tales metamodelos. Finalmente, en el tercer nivel ajustamos un último metamodelo que toma como entradas las predicciones devueltas por los M metamodelos del nivel anterior.

Desde un punto de vista práctico, tenga en cuenta que, para cada metamodelo de los diferentes niveles de un modelo de conjunto de stacking de niveles múltiples, tenemos que elegir un algoritmo de aprendizaje que puede ser casi lo que queramos (incluso algoritmos que ya se usan en niveles más bajos). También podemos mencionar que agregar niveles puede ser costoso en datos (si no se usa una técnica similar a k-folds y, luego, se necesitan más datos) o costoso en tiempo (si se usa una técnica similar a k-folds y, luego, muchos modelos necesitan ser ajustado).

STACKING

Paso a Paso del Stacking de Varios Niveles

1. Entrenamiento de los Modelos Base en el Primer Nivel

Ecuación de los Modelos Base del Primer Nivel:

$$h_j^{(1)}(x) = \text{Modelo base } j \text{ en el primer nivel, entrenado con } (X, y)$$

2. Generación de Predicciones del Primer Nivel

Predicciones de los Modelos Base del Primer Nivel:

$$z_j^{(1)} = h_j^{(1)}(X)$$

Estas predicciones forman el conjunto de datos para el segundo nivel:

$$Z^{(1)} = [z_1^{(1)}, z_2^{(1)}, \dots, z_J^{(1)}]$$

3. Entrenamiento de Modelos Base en el Segundo Nivel

Ecuación de los Modelos Base del Segundo Nivel:

$$h_k^{(2)}(Z^{(1)}) = \text{Modelo base } k \text{ en el segundo nivel, entrenado con } (Z^{(1)}, y)$$

4. Generación de Predicciones del Segundo Nivel

Predicciones de los Modelos Base del Segundo Nivel:

$$z_k^{(2)} = h_k^{(2)}(Z^{(1)})$$

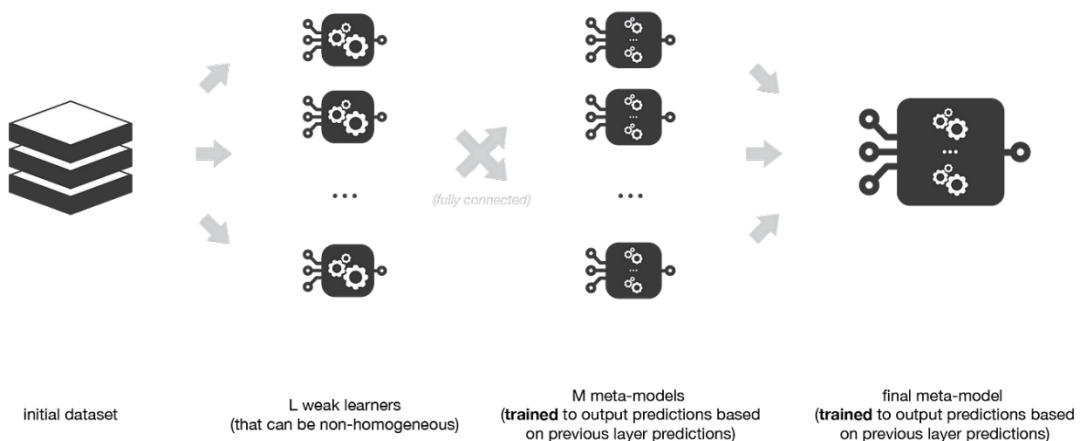
Estas predicciones forman el conjunto de datos para el siguiente nivel:

$$Z^{(2)} = [z_1^{(2)}, z_2^{(2)}, \dots, z_K^{(2)}]$$

5. Entrenamiento del Modelo Meta Final

Ecuación del Modelo Meta Final:

$$g(Z^{(N-1)}) = \text{Modelo meta final entrenado con las predicciones } Z^{(N-1)} \text{ y las etiquetas}$$



El stacking multinivel considera varias capas de stacking: algunos metamodelos se entrenan en los resultados devueltos por metamodelos de capas inferiores, etc. Aquí hemos representado un modelo de apilamiento de 3 capas.

CONCLUSIONES

Las principales conclusiones de esta sesión son las siguientes:

- El aprendizaje conjunto (ensemble) es un paradigma de aprendizaje automático en el que se entrenan varios modelos (a menudo denominados modelos básicos o de aprendizaje débil) para resolver el mismo problema y se combinan para obtener un mejor rendimiento.
- La hipótesis principal es que si combinamos los aprendices débiles de la manera correcta podemos obtener modelos más precisos y/o robustos
- En los métodos de bagging, varias instancias del mismo modelo base se entrenan en paralelo (independientemente entre sí) en diferentes muestras de bootstrap y luego se agregan en algún tipo de proceso de "promedio".
- El tipo de operación de promediación realizada sobre los (casi) modelos i.i.d ajustados en los métodos de bagging nos permite principalmente obtener un modelo de conjunto con una varianza más baja que sus componentes: es por eso que los modelos base con un sesgo bajo, pero una varianza alta se adapta bien para el bagging.
- En los métodos de boosting, varias instancias del mismo modelo base se entrenan secuencialmente de modo que, en cada iteración, la forma de entrenar al aprendiz débil actual depende de los aprendices débiles anteriores y, más especialmente, de cómo se están desempeñando en los datos.
- Esta estrategia iterativa de aprendizaje utilizada en los métodos boosting, que se adapta a las debilidades de los modelos anteriores para entrenar el actual, nos permite principalmente obtener un modelo de conjunto con un sesgo menor que sus componentes: es por eso que los aprendices débiles con baja varianza, pero el sesgo alto está bien adaptado para boosting.
- En los métodos de stacking, diferentes aprendices débiles se ajustan independientemente unos de otros y se entrena un metamodelo además de eso para predecir los resultados en función de los resultados devueltos por los modelos base.

En esta sesión hemos dado una visión básica del aprendizaje de conjuntos y algunas de las principales nociones de este campo: bootstrapping, bagging, random forest, boosting (adaboost, gradiente boosting) y stacking.

Entre las nociones que quedaron de lado podemos mencionar por ejemplo la técnica de evaluación Out-Of-Bag para bagging o también el muy popular "XGBoost" (que significa eXtrem Gradient Boosting) que es una librería que implementa métodos de Gradient Boosting junto con una gran cantidad de trucos adicionales que hacen que el aprendizaje sea mucho más eficiente (y manejable para grandes conjuntos de datos). Finalmente, nos gustaría concluir recordando que el aprendizaje de conjunto consiste en combinar algunos modelos base para obtener un modelo de conjunto con mejores prestaciones/propiedades. Por lo tanto, incluso si el bagging, el boosting y el stacking son los métodos de conjunto más utilizados, las variantes son posibles y pueden diseñarse para adaptarse mejor a algunos problemas específicos. Esto requiere principalmente dos cosas: comprender completamente el problema al que nos enfrentamos.