Capítulo 1/9

Introducción

**Métodos numéricos**

**Estudiaremos los métodos numéricos e intentaremos crear algoritmos de entrenamiento basados en ellos.**

Aprenderemos a utilizar el descenso y la potenciación del gradiente, así como a analizar los algoritmos de entrenamiento para modelos de regresión lineal y redes neuronales.

**Estructura del curso**

El curso comienza con el análisis de algoritmos de entrenamiento. Aprenderemos a calcular la complejidad computacional de un algoritmo y a diferenciar los métodos iterativos de los directos. También descubriremos cómo resolver ecuaciones utilizando el método de bisección.

A continuación, pasaremos al método de descenso de gradiente. Este es la base de muchos algoritmos de entrenamiento. Entrenaremos un modelo de regresión lineal y una red neuronal mediante el descenso de gradiente.

Por último, volveremos a examinar los modelos de árboles y estudiaremos la técnica de potenciación del gradiente (gradient boosting) que se basa en el descenso del gradiente.

Al final, como siempre, te encontrarás con el proyecto del curso. Haz clic [aquí](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/moved_Descripcin_del_proyecto_DS_Sprint_12_ESP.pdf) si quieres ver la descripción del proyecto.

**Tus objetivos:**

* Escribir un algoritmo de descenso de gradiente.
* Entender cómo se entrenan los modelos de regresión lineal y las redes neuronales mediante el descenso de gradiente.
* Dominar la técnica de la potenciación del gradiente.

En este sprint trabajarás en estas habilidades:

Logotipo

Descripción generada automáticamenteImagen que contiene Gráfico

Descripción generada automáticamenteAnálisis de algoritmos

**Introducción**

Analizaremos los algoritmos de entrenamiento de modelos con el fin de entender por qué necesitamos métodos numéricos.

**Empezarás por:**

* Aprender sobre la complejidad computacional de un algoritmo y cómo calcularla
* Estudiar los algoritmos directos e iterativos
* Dominar el método de bisección.

**¿Cuánto tiempo llevará?**

6 lecciones de 10-15 minutos cada una

Esquemático

Descripción generada automáticamente

Capítulo 2/9

Análisis de algoritmos

**Complejidad computacional**

¿Qué algoritmos de entrenamiento son los más rápidos? ¿Cómo puedes determinar su complejidad computacional?

La cantidad de tiempo durante el cual un PC ejecuta las operaciones depende de su capacidad de procesamiento individual, por lo que el tiempo de ejecución de algoritmos no se mide en segundos. Está determinado por el número de operaciones elementales que realiza el algoritmo: sumas, multiplicaciones y comparaciones.

El tiempo de ejecución en un PC determinado suele llamarse tiempo de ejecución real. También puedes encontrar el concepto de *tiempo de reloj de pared*. Es decir, el tiempo de ejecución del algoritmo determinado no por tiempo del procesador, sino por el tiempo transcurrido en el reloj.

El tiempo de ejecución de un algoritmo también depende de sus argumentos. Por ejemplo, para encontrar un determinado valor en listas de distinta longitud (por ejemplo, 10 y 1000), se necesitará un número distinto de operaciones.

Vamos a calcular el tiempo de ejecución de un algoritmo que incrementa en uno todos los valores de la lista. Vamos a expresar la longitud de la lista como *n*. El tiempo de trabajo es una función de *n,* escrita como *T(n)*.

def increase(elements):

n = len(elements)

for i in range(n):

elements[i] += 1

Supongamos que:

* Para llamar a la función len() o range() se necesita una operación;
* Cada una de estas operaciones se realiza en un solo paso: asignación compuesta de suma +=, asignación simple = o direccionamiento indexado;
* Cada paso del ciclo requiere dos operaciones ya que no solo hay que aumentar *i* en uno, sino también confirmar que el número total de elementos no supera la longitud de la lista.

def increase(elements):

n = len(elements) *# 2 operaciones*

for i in range(n): *# 1 operación + 2 operaciones \* n*

elements[i] += 1 *# 2 operaciones (para cada valor de i)*

Podemos calcular *T(n)* de la siguiente manera: 2 + 1 + *n* \* (2 + 2) *= 4n* + 3 (los términos se escriben en orden descendente de las potencias de n).

Es imposible calcular el tiempo de ejecución de los algoritmos complejos. Por eso determinamos su complejidad computacional, o tiempo de ejecución asintótico. El término se basa en la palabra asíntota, que define una línea recta a la que se aproxima una cierta curva pero sin llegar a cruzarla (del griego antiguo *ἀσύμπτωτος,* "que no cae junto").

El tiempo de ejecución asintótico de un algoritmo muestra cómo *T(n)* crece a medida que *n* se incrementa.

Cuando *T(n)* es un polinomio (la suma de todos los términos, es decir, de los productos de números y variables de diferentes potencias), el tiempo de ejecución asintótico es igual al término de mayor potencia sin coeficiente (por ejemplo, *n²* en lugar de *5n²*). A medida que *n* alcanza valores mayores, los demás términos pierden importancia.

Consideremos estos ejemplos:

1) Si *T(n)* = *4n* + 3, el tiempo de ejecución asintótico *T(n) ~ n.* El algoritmo tiene complejidad lineal. El símbolo de la tilde (~) significa que el tiempo de ejecución asintótico es igual a *n*.

2) Si *T(n) = 5n²* + *3n* - 1, el tiempo de ejecución asintótico *T(n) ~ n².* El algoritmo tiene complejidad cuadrática.

3) Si *T(n)* = *10n³* - 2*n²* + 5, entonces *T(n) ~ n³*. El algoritmo tiene complejidad cúbica.

4) Si *T(n)* = 10, entonces *T(n)* ~ 1.El algoritmo tiene complejidad constante, es decir, no depende de *n*.

Para transponer una matriz de tamaño *n×n*, tendrás que:

* Crear una nueva matriz de tamaño *n×n*, es decir, con *n²* celdas (el número de operaciones es aproximadamente igual a *n*²)
* Copiar cada elemento con índices (*i, j*) de la matriz original en una celda (*j, i*) de la nueva matriz (aproximadamente *n*² operaciones).

Obtenemos el tiempo asintótico *T(n)* ~ *n²*.

Pregunta

Determina la complejidad computacional de la multiplicación escalar de dos vectores n-dimensionales:

10-50 milisegundos

La complejidad computacional es compleja

*T(n)* ~ n

Cada coordenada del primer vector debe multiplicarse por la coordenada correspondiente del segundo. Su total es igual a n.

*T(n)* ~ *n²*

¡Bien hecho!

Pregunta

Determina la complejidad computacional de la multiplicación de la matriz *X* cuyo tamaño es *n×n* por el vector ndimensional *y*:

*T(n)* ~ 1

*T(n)* ~ n

*T(n)* ~ *n²*

Cada fila de la matriz *X* se multiplica escalarmente por el vector *y.* Hay *n* filas en total, por lo que cada multiplicación escalar requiere aproximadamente *n* operaciones. El resultado es *n²*.

*T(n)* ~ *n³*

Capítulo 2/9

Análisis de algoritmos

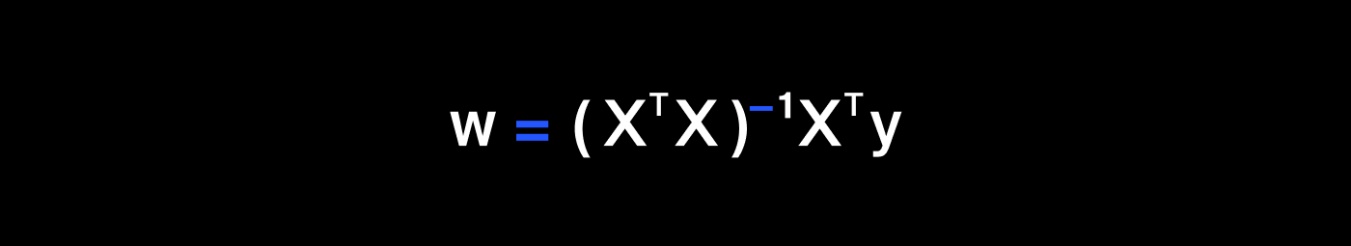
**Tiempo de entrenamiento de un modelo de regresión lineal**

¿Es difícil entrenar modelos de regresión lineal? Vamos a hacer las cuentas.

El objetivo del entrenamiento de una regresión lineal se representa de la siguiente manera:



Los pesos se calculan mediante esta fórmula:



Determinaremos la complejidad computacional del cálculo de pesos, pero primero repasemos algunos aspectos:

* Vamos a expresar el número de observaciones en el conjunto de entrenamiento como *n* y el número de características como *p*;
* El tamaño de la matriz *X* será *n x p* y el tamaño del vector *y* será *n*;
* La complejidad computacional se expresará como *T(n, p)* porque depende de dos parámetros, *n* y *p*.

La primera acción de la fórmula es la multiplicación de la matriz *X* por su transposición (X.T.dot(X)).

Pregunta

¿Cuál es el tamaño de la matriz resultante de la multiplicación de la matriz *X* por su transposición (X.T.dot(X))?

*n* x *n*

*p* x *p*

El tamaño de la transposición de la matriz *X* es *p x p*. Cuando se multiplica por la matriz *X* (*p* x *p.*), la dimensión n se "colapsa" dando como resultado una matriz con el tamaño de *p* x *p.*

*n* x *p*

*n*

*p* x *n*

*p*

¡Perfecto!

Pregunta

Determina la complejidad computacional al multiplicar la matriz *X* por su transposición (X.T.dot(X)):

*T(n, p)* ~ *np*

*T(n, p) ~ np²*

La matriz resultante tiene *p x p* elementos y cada uno de ellos se calcula como un producto escalar de vectores *n-*dimensionales.

*T(n, p)* ~ *n²p*

*T(n, p) ~ n²p²*

¡Lo has entendido bien!

Realicemos la segunda acción y encontremos la inversa de la matriz. El algoritmo para la inversión de una matriz de tamaño *p x p* tiene una complejidad cúbica.

Multipliquemos la matriz inversa por la transposición de la matriz *X*.

Pregunta

¿Cuál es la complejidad computacional de esta multiplicación? opciones:

*T(n, p)* ~ *np*

*T(n, p) ~ np²*

La matriz resultante tiene p x n elementos y cada uno de ellos se calcula como un producto escalar de vectores *p-*dimensionales.

*T(n, p)* ~ *n²p*

*T(n, p) ~ n²p²*

¡Excelente trabajo!

Ahora vamos a multiplicar el resultado de la acción anterior por el vector *y*.

Pregunta

Determina la complejidad computacional de esta multiplicación:

*T(n, p)* ~ *n*

*T(n, p) ~ p*

*T(n, p) ~ np*

Hay *p* filas en la matriz y la longitud de cada una es igual a *n*. Se multiplican por un vector n-dimensional.

*T(n, p) ~ np²*

*T(n, p)* ~ *n²p*

¡Buen trabajo!

Para calcular la complejidad del entrenamiento del algoritmo, hay que sumar las respuestas:

*T(n, p) ~ np² + p³ + np² + np*

Normalmente hay menos características que observaciones, lo que significa que *p* < *n*. Si multiplicamos ambas partes por *p²* obtenemos *p³* < *np²*. Si solo tomamos el término con la mayor potencia, obtenemos: *T(n, p) ~ np²*. Si hay muchas características, el entrenamiento llevará más tiempo.

Relaciona los conceptos con su definición, escribiendo la letra  
correspondiente en el paréntesis.

Termina de escribir el código de la función bisect(). Esta función resolverá la ecuación utilizando el método de bisección, que es uno de los métodos iterativos. La función toma:

* function — una función con los valores cero del objetivo. En Python, las funciones se pueden pasar como argumentos. Así es como se llama: function(x)
* left, right — extremos izquierdo y derecho del segmento
* error — magnitud de error aceptable (la exactitud del algoritmo depende de ella)

La prueba del método para dos funciones ya está en el precódigo.

Termina de escribir el código de la función bisect(). Esta función resolverá la ecuación utilizando el método de bisección, que es uno de los métodos iterativos. La función toma:

* function — una función con los valores cero del objetivo. En Python, las funciones se pueden pasar como argumentos. Así es como se llama: function(x)
* left, right — extremos izquierdo y derecho del segmento
* error — magnitud de error aceptable (la exactitud del algoritmo depende de ella)

La prueba del método para dos funciones ya está en el precódigo.

**Pistas**

Termina el código para seleccionar un nuevo segmento:

middle = (left + right) / 2

if function(left) \* function(middle) < 0:

right = middle

else:

*# < escribe tu código aquí >*

import math

def bisect(function, left, right, error):

# El bucle while repite el código mientras se cumpla el criterio.

# Añadimos el criterio de parada.

while right - left > error:

# asegúrate de que no haya ceros

if function(left) == 0:

return left

if function(right) == 0:

return right

# biseca el segmento y encuentra el nuevo

middle = (left + right) / 2# < escribe tu código aquí >

if function(left) \* function(middle) < 0:# < escribe tu código aquí >

right = middle

else:

left = middle# < escribe tu código aquí >

return left

def f1(x):

return x \*\* 3 - x \*\* 2 - 2 \* x

def f2(x):

return (x + 1) \* math.log10(x) - x \*\* 0.75

print(bisect(f1, 1, 4, 0.000001))

print(bisect(f2, 1, 4, 0.000001))

Resultado

1.999999761581421

3.6952309608459473

¡Es correcto!

1.999999761581421 es casi dos. ¡Hemos encontrado la raíz de *x³ - x² - 2x = 0!* Y estamos satisfechos con la aproximación.

Capítulo 2/9 · Faltan 2 lecciones

Análisis de algoritmos

**Comparación de métodos**

Las raíces de una ecuación también se pueden encontrar de forma directa. ¿Qué método deberías elegir?

La ecuación de la función *f(x) = x³ - x² - 2x* se resuelve así:

* Podemos factorizar una *x*: *(x² - x - 2)x = 0,* y vemos que el cero es la primera raíz de la ecuación.
* El discriminante de la ecuación cuadrática *(x² - x - 2)* es 1 + 4 \* 1 \* 2 = 9.

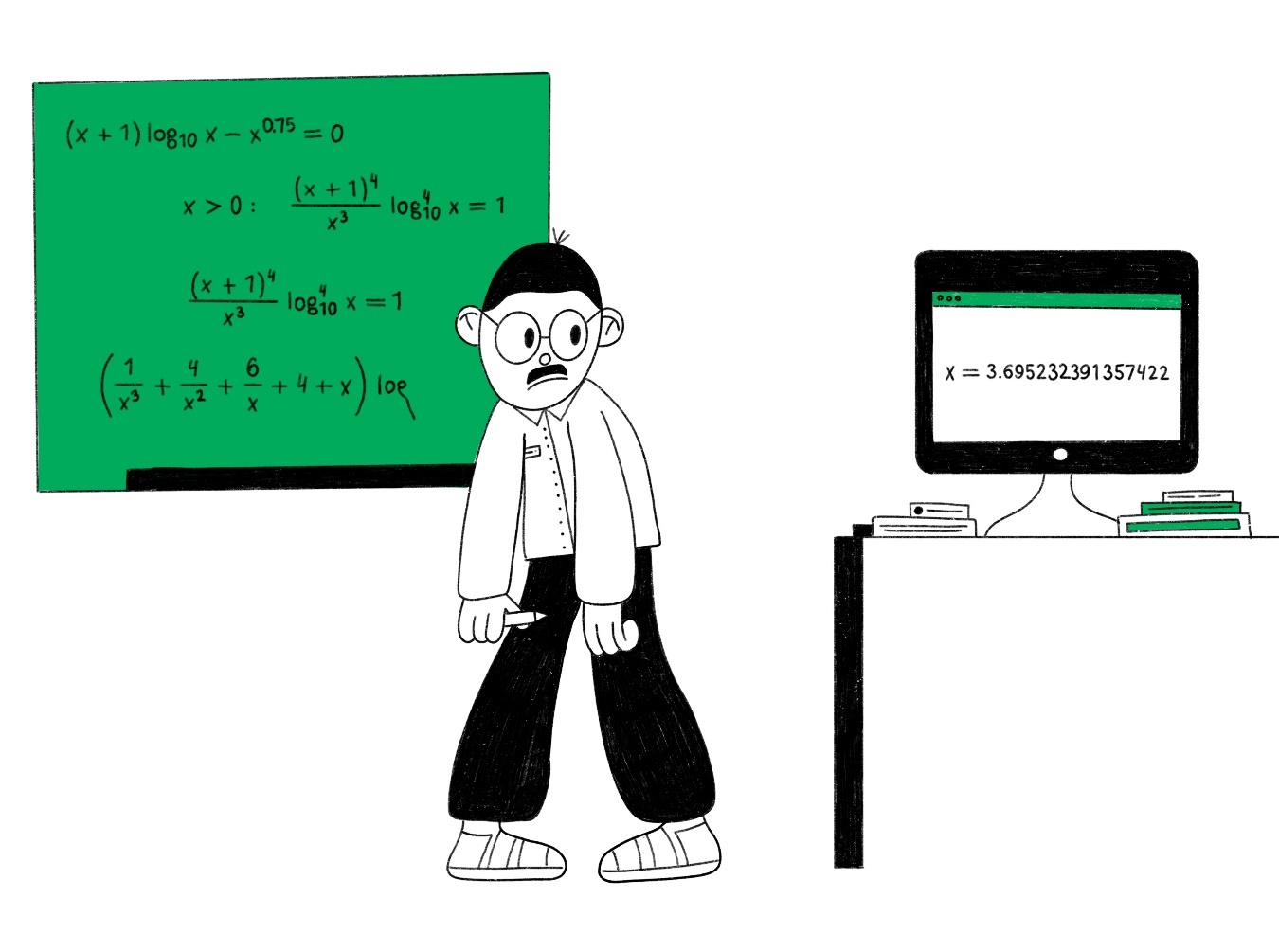
El discriminante (del latín *discrimino*, que significa "distinguir"; *D*) de la ecuación *ax²* + *bx* + *c* = 0 se puede encontrar utilizando la fórmula *b² - 4ac*. Si *D > 0*, la ecuación cuádratica tiene dos raíces si *D* = 0, tiene una raíz, y si *D* < 0, no tiene ninguna.

* Las raíces de la ecuación *(x² - x - 2)* se pueden encontrar mediante la fórmula *(-b ± √D)/2a)*, es decir, (1 ± 3)/2.

Las raíces resultantes son: -1, 0, 2.

Ha sido rápido. Y ni siquiera hemos tenido que ejecutar ningún código.

No se puede aplicar lo mismo a la función f(x)=(x+1)log⁡10(x)−x0.75*f*(*x*)=(*x*+1)log10​(*x*)−*x*0.75. No se puede encontrar su raíz de forma directa, pero el método de bisección puede servir con cualquier función, a condición de que sea continua.



Una gran parte de este curso se centra en el descenso de gradiente, que es el método iterativo clave para el machine learning. Muchos algoritmos de entrenamiento se basan en él porque tiene muchas ventajas en comparación con los métodos directos, por ejemplo:

* Funciona más rápido con grandes conjuntos de datos en regresiones lineales con la función de pérdida *ECM*.
* También es apropiado para regresiones lineales con otras funciones de pérdida (no todas tienen fórmulas).
* Puede utilizarse para entrenar redes neuronales que también carecen de fórmulas directas.

Capítulo 2/9 · Última lección

Análisis de algoritmos

**Conclusión**

En este capítulo aprendiste a:

* Determinar la complejidad computacional de un algoritmo;
* Distinguir los métodos directos de los iterativos;
* Resolver ecuaciones mediante el método de bisección.

**Llévate esto contigo**

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Análisis de algoritmos](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/Resumen_del_captulo_Anlisis_de_algoritmos.pdf)
* [Hoja informativa: Análisis de algoritmos](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/moved_Hoja_informativa_Anlisis_de_algoritmos_DS_Sprint_12_esp.pdf)

En el próximo capítulo, aprenderás sobre el algoritmo de descenso de gradiente.

Capítulo 3/9

Descenso de gradiente

**Introducción**

**Vamos a echar un vistazo al algoritmo de descenso de gradiente.**

**Empezarás por:**

* Aprender sobre las funciones de pérdida de los problemas de clasificación y regresión;
* Descubrir qué es el gradiente de una función;
* Estudiar el algoritmo de descenso de gradiente.

**¿Cuánto tiempo llevará?**

6 lecciones de 10-15 minutos cada una.

Capítulo 3/9

Descenso de gradiente

**Minimización de la función de pérdida**

**Antes de plantear una tarea de descenso de gradiente, veamos el problema de entrenamiento desde un nuevo ángulo.**

Ya conoces la función de pérdida. Esta cuantifica las respuestas incorrectas del modelo y suele utilizarse para su entrenamiento.

En el curso anterior, equiparamos la función de pérdida con el ECM. En este nuevo ejercicio, vamos a expresar la función de pérdida como *L( y, a ),* donde el vector *y* representa las respuestas correctas y el vector *a* representa las predicciones.

Escribe la tarea de entrenamiento del modelo mediante la siguiente función de pérdida:



Pregunta

Tienes que entrenar el modelo utilizando la minimización de la función de pérdida. ¿Qué vas a hacer?

Encontrar los pesos de regresión lineal que minimicen los valores de las predicciones.

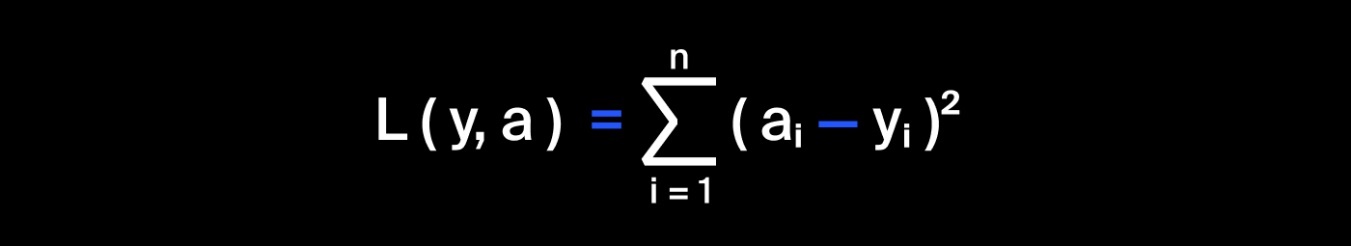
Elegir los pesos de regresión lineal de manera que las predicciones se acerquen lo más posible a las respuestas correctas.

¡El modelo está siendo entrenado! Así es como se hace.

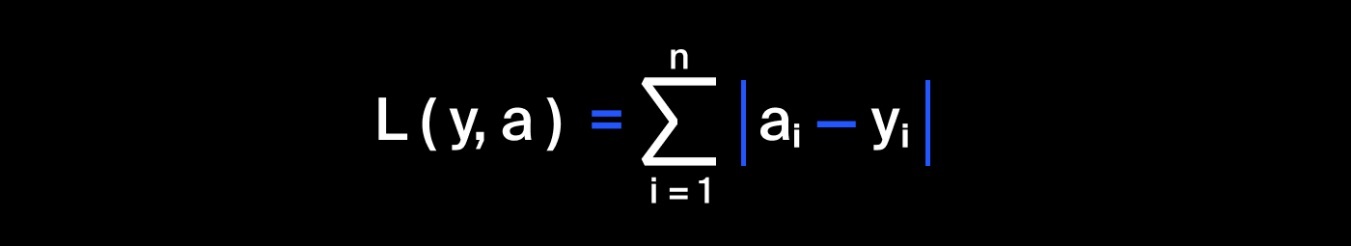
Seleccionar los datos de entrenamiento para minimizar la función de pérdida.

¡Perfecto!

Para calcular el ECM, o la función de pérdida cuadrática, eleva al cuadrado la diferencia entre las respuestas correctas y las predicciones:



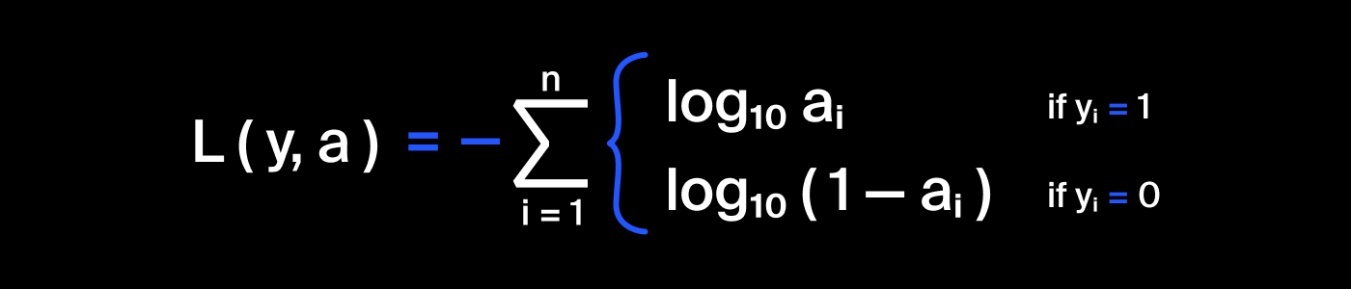
¿Qué otras funciones de pérdida existen para una tarea de regresión? Si queremos que la función de la tarea sea menos sensible a los valores atípicos, entonces en lugar de *ECM*, utilizaremos la función de pérdida absoluta (EAM), que también conoces ya. Al igual que con el *ECM*, la utilizaremos como una función de pérdida, no como una métrica de evaluación.



Cuando se trata de problemas de clasificación, a menudo se utiliza la métrica de *exactitud*. Pero rara vez se puede utilizar como una función de pérdida. Esto se debe a que la función de cálculo de la exactitud no tiene ninguna derivada que muestre el cambio en la función ante pequeños cambios en el argumento.

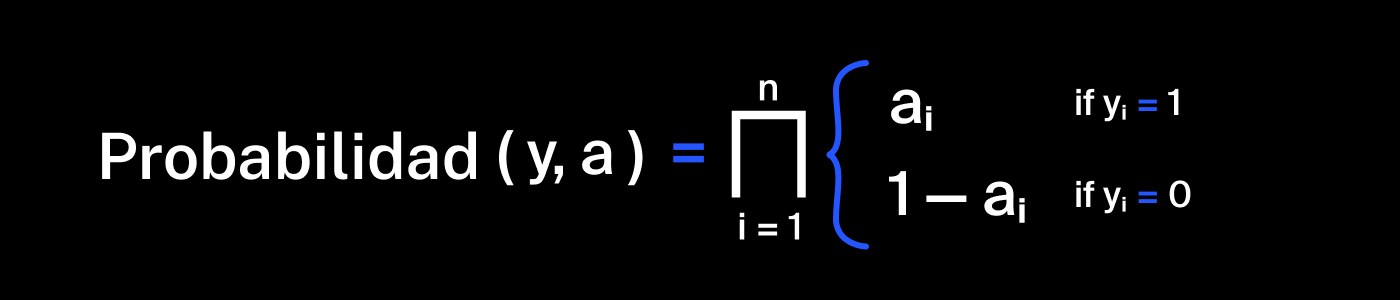
Sustituyamos la *exactitud* por la verosimilitud logarítmica negativa o la función de pérdida logística. La función de pérdida suma los logaritmos de las probabilidades en función de la observación. Si la respuesta correcta es 1, se suma *log₁₀ aᵢ*. Si es cero, se añade *log₁₀* (1 - *aᵢ*). El logaritmo (del griego antiguo *λόγος,* "palabra"; *ἀριθμός*, "número") es el exponente al que hay que elevar una base (en nuestro caso, 10) para obtener el argumento.

Aquí tienes la fórmula:



donde *aᵢ* es la probabilidad de la clase 1 para la observación con índice *i*. Es decir, el valor de *aᵢ* debe ser lo más alto posible para una observación de clase positiva y lo más bajo posible para una observación de clase negativa.

El nombre de la verosimilitud logarítmica negativa proviene de la función de verosimilitud, que calcula la probabilidad de que el modelo dé respuestas correctas para todas las observaciones si los valores *aᵢ* se toman como respuesta:

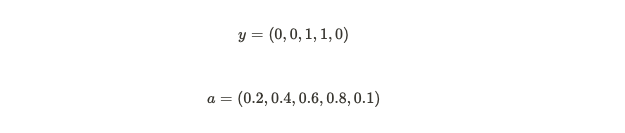


Al tomar el logaritmo, los valores "se expanden" a un rango más amplio. Por ejemplo, el rango de 0 a 1 se convierte en el rango de -∞ a 0. Para este rango, los errores de cálculo no son tan importantes. Multiplicamos por -1 porque la función de pérdida final debe minimizarse.

A diferencia de la *exactitud*, la función de pérdida logarítmica tiene una derivada.

Pregunta

Utilizando la fórmula, encuentra el valor de la función de pérdida logarítmica para los siguientes valores de y y a:



-3.7

2.42

0.68

¡Correcto! ¡Hemos encontrado la pérdida! Echa un vistazo a la solución.

1.57

¡Tu comprensión del material es impresionante!

Calcular el valor de la función logística:

from math import log10

print(

-(

log10(1 - 0.2)

+ log10(1 - 0.4)

+ log10(0.6)

+ log10(0.8)

+ log10(1 - 0.1)

)

)

0.68327501581

Capítulo 3/9

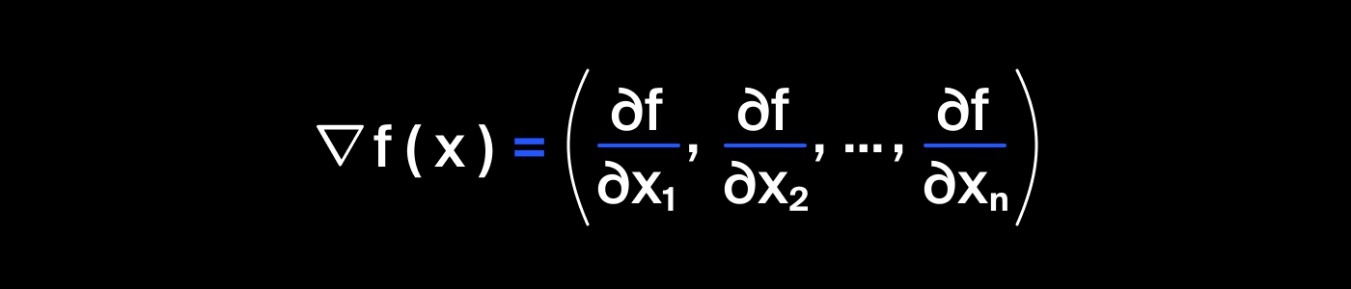
Descenso de gradiente

**Gradiente de una función**

**No siempre podemos encontrar manualmente el mínimo de la función de pérdida.**El gradiente de una función**puede ayudar a encontrar la dirección.**

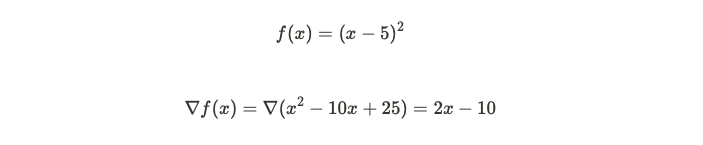
La función de pérdida depende de los parámetros del algoritmo. Esta función es una función de valor vectorial, es decir, toma un vector y devuelve un escalar.

El gradiente de una función de valor vectorial es un vector que está formado por las derivadas de la respuesta para cada argumento. Se expresa como ∇ (nabla - un arpa hebrea de figura similar a la del símbolo). El gradiente de la función *f* de un vector n-dimensional x se calcula de la siguiente manera:

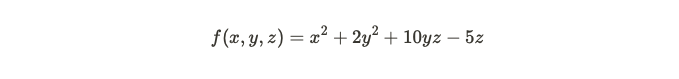


donde *∂f/∂xi* es la derivada parcial de la función *f* para el argumento *xᵢ.*

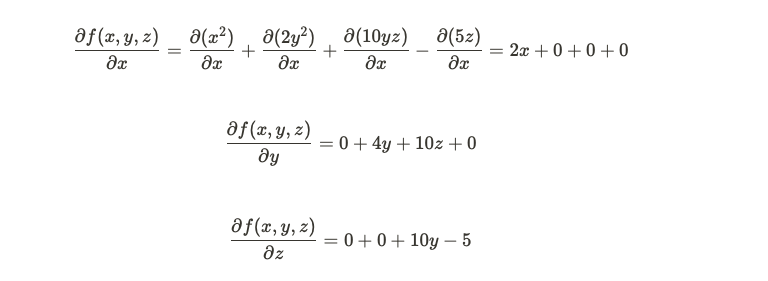
El gradiente de una función para un argumento es la derivada:



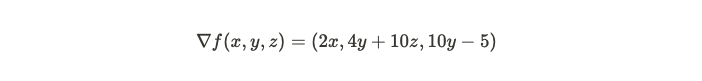
Vamos a encontrar el gradiente de una función de tres argumentos:



Calcula las derivadas parciales para *x*, *y* y *z*:

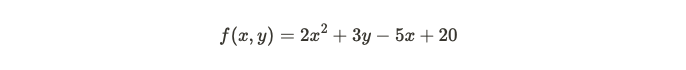


Obtén un vector tridimensional a partir de las derivadas parciales.



Pregunta

Encuentra el gradiente de una función de dos argumentos:



(4x - 5)

(3)

(4x-5, 3)

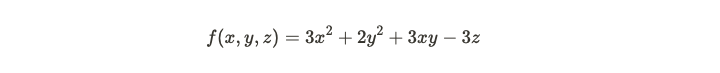
El gradiente es un vector de derivadas parciales con respecto a *x* y *y*.

(4x, 3, 5)

¡Buen trabajo!

Pregunta

Encuentra el gradiente de una función de tres argumentos:



(6x + 3, 4y)

(6x + 3y, 4y + 3x)

(6x + 3y, 4y + 3x, -3)

El gradiente es un vector de cociente de derivadas con respeto a *x*, *y* y *z*.

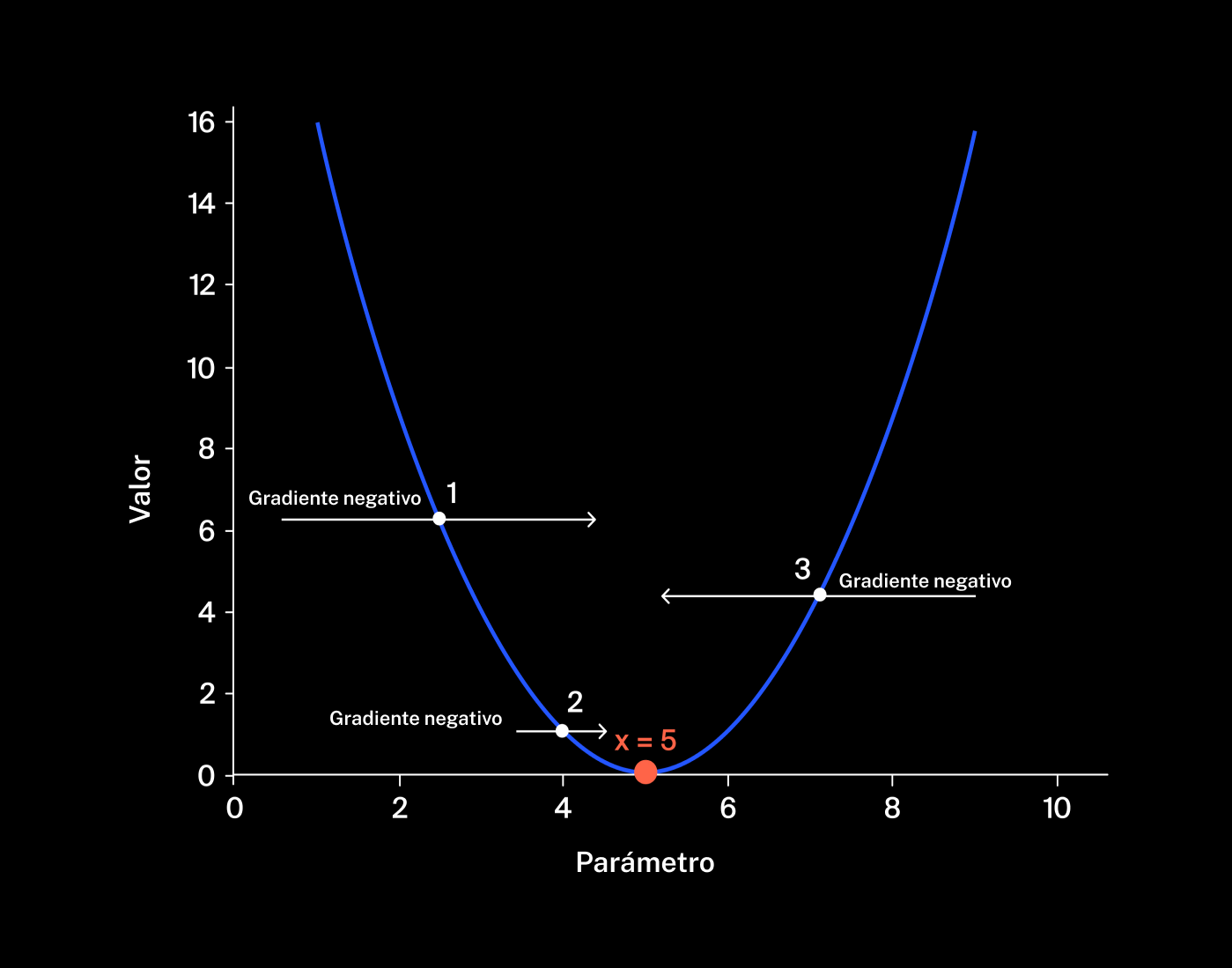
(6x + 3, 4y, 3)

¡Perfecto!

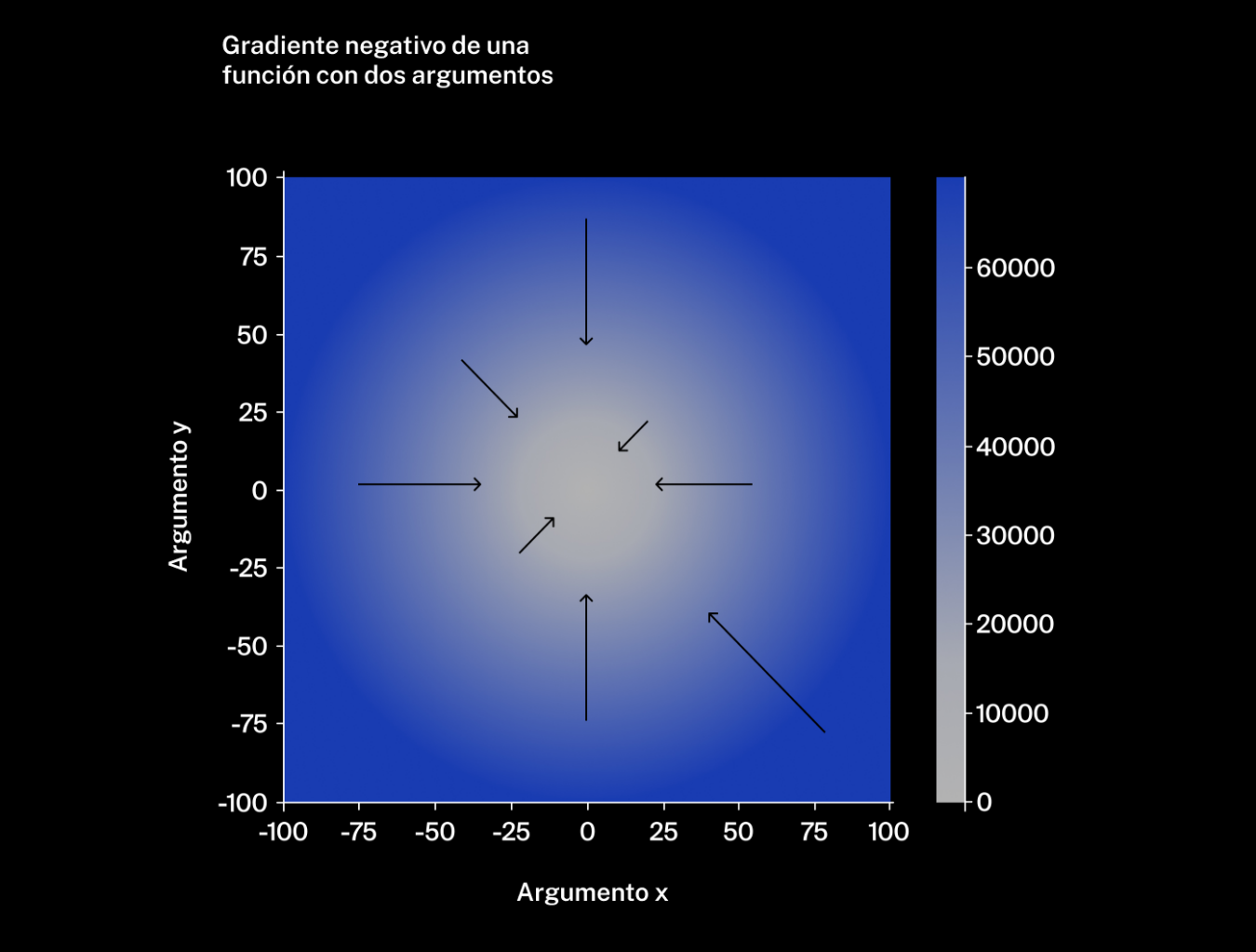
El gradiente indica la dirección en la que la función crece más rápido. Sin embargo, no sirve para resolver el problema de minimización. Necesitamos el gradiente negativo, que es un vector opuesto que muestra el decrecimiento más rápido. Lo podemos determinar de la siguiente manera:



Para la función unidimensional *f(x) = (x - 5)²,* el gradiente negativo es *∇f(x)=-2(x-5)*. Los valores del gradiente negativo situados a la izquierda de *x = 5* son positivos. A la derecha son negativos.



Para la función *f(x, y) = 2x² + 2y² - 8*, el gradiente negativo es *∇f(x)=(-4x, -4y)*. Observa su dirección en cada punto:



Pregunta

Elige todas las afirmaciones correctas:

Elige tantas como quieras

El gradiente negativo indica el punto mínimo de la función.

El gradiente de la función en un punto determinado es igual al vector de las derivadas parciales de los argumentos.

Las derivadas se calculan mediante los coeficientes de la función.

El vector gradiente negativo es opuesto al vector gradiente.

El gradiente se convierte en negativo cuando se multiplica por -1.

El gradiente negativo indica la dirección en la que la función crece más rápido.

El gradiente negativo indica la dirección en la que la función decrece más rápido.

Correcto.

¡Tu comprensión del material es impresionante!

Capítulo 3/9 · Faltan 3 lecciones

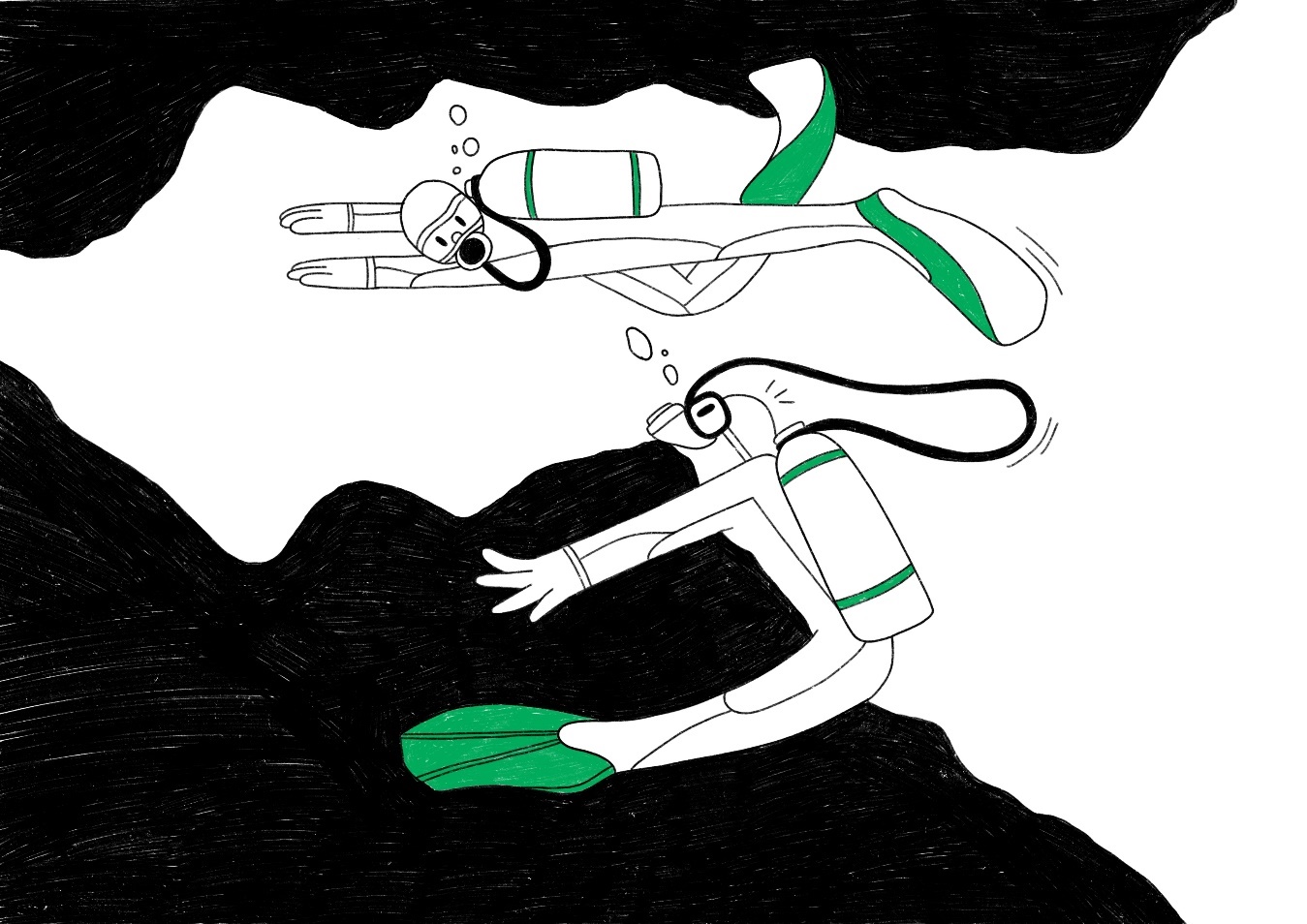
Descenso de gradiente

**Descenso de gradiente**

**Vamos a encontrar el mínimo de la función de pérdida utilizando el algoritmo de descenso de gradiente.**

El descenso de gradiente es un algoritmo iterativo para encontrar el mínimo de la función de pérdida. Sigue la dirección del gradiente negativo y se aproxima gradualmente al mínimo.

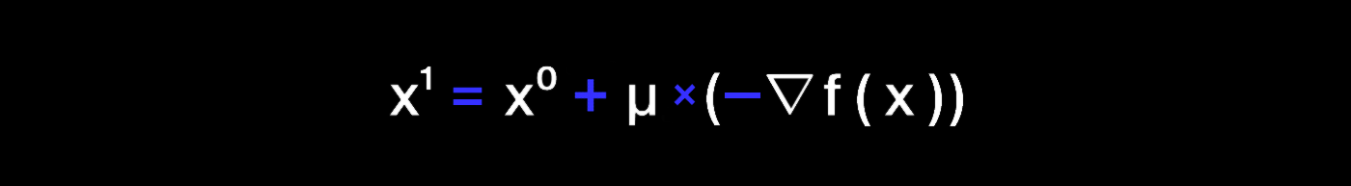
Podríamos comparar el descenso de gradiente con el buceo profundo. Un buceador novato tiene que examinar previamente el lugar de inmersión. En caso contrario, cualquier movimiento, por más ligero que sea, puede provocar un golpe contra una roca. Por lo tanto, la inmersión debe ser lenta y realizarse con mucho cuidado.



El algoritmo de descenso de gradiente funciona de la misma manera, es decir, "se sumerge hasta el fondo" al realizar repetidamente los mismos pasos. Es difícil llegar al mínimo en una sola iteración, porque el vector de gradiente negativo no indica el punto mínimo de la función de pérdida, sino la dirección del decrecimiento.

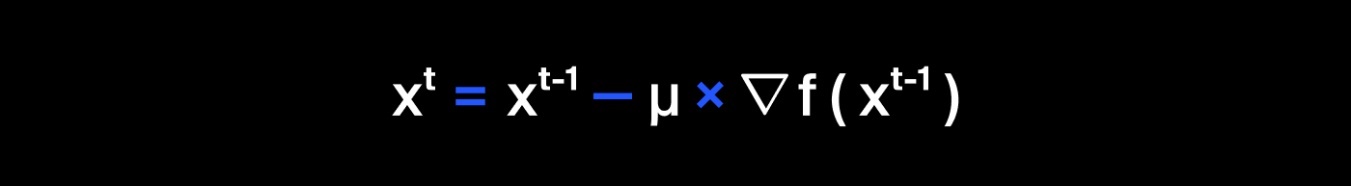
Para comenzar el descenso, vamos a elegir el valor inicial del argumento (vector *x*). Se expresa como x⁰.

A partir de ahí, se realizará el primer paso de descenso de gradiente. El siguiente punto, x¹, se calcula de la siguiente manera: al punto x⁰ se le suma el gradiente negativo multiplicado por el tamaño del paso de descenso del gradiente (*μ*).



El valor μ determina el tamaño del paso de descenso de gradiente. Si el paso es pequeño, el descenso pasará por muchas iteraciones. No obstante, cada una de ellas nos acercará al mínimo de la función de pérdida. Si el paso es demasiado grande, podemos pasar por alto el mínimo (chocaremos contra la roca como un buceador).

Repite la operación para obtener los valores de los argumentos en las siguientes iteraciones. El número de iteraciones se expresa como *t*. Para obtener los nuevos valores de *xᵗ*, hay que multiplicar el gradiente negativo por el tamaño del paso y sumar este producto al valor anterior:



El descenso de gradiente está completado cuando:

* el algoritmo complete el número necesario de iteraciones

o

* el valor de *x* está por debajo de un valor umbral (elegido por el analista según la exactitud requerida).

Pregunta

Elige todas las afirmaciones correctas:

Elige tantas como quieras

Las iteraciones de descenso de gradiente ayudan a acercarse al mínimo de la función de pérdida.

El mínimo se va acercando con cada paso.

Cuanto mayor sea el tamaño del paso, más rápido encontrará el algoritmo el mínimo.

El vector del argumento se actualiza después de cada iteración.

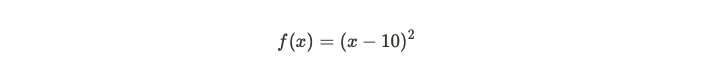
Una vez completado el descenso de gradiente, obtendrás un conjunto de argumentos donde la función de pérdida se acerque al mínimo.

El descenso de gradiente finaliza cuando el valor de la función es cero.

¡Perfecto!

Pregunta

Calcula el valor x y el valor de la función de pérdida tras realizar cuatro iteraciones de descenso de gradiente. Esta es la función:



El tamaño del paso *μ* es 0,4. El valor inicial de *x* es igual a 0.

Elige la respuesta correcta:

*x*=10, *f(x)*=0

*x*=10.08, *f(x)*=0.007

*x*=9.984, *f(x)*=0.000256

¡El descenso de gradiente casi ha alcanzado el mínimo!

*x*=9.92, *f(x)*=0.006

¡Lo has entendido bien!

Se nos presenta una función de pérdida:

f(x) = (x - 10)^2

Se nos pide aplicar el algoritmo de descenso de gradiente para encontrar un mínimo local de esta función.

* **Tasa de aprendizaje (tamaño de paso):** μ = 0.4
* **Valor inicial:** x = 0
* **Número de iteraciones:** 4

**Descendente de Gradiente:**

Es un algoritmo de optimización iterativo que busca encontrar un mínimo local de una función. En cada iteración, se actualiza el valor de x moviéndolo en la dirección opuesta al gradiente de la función.

**Pasos a Seguir:**

1. **Calcular el gradiente:** La derivada de f(x) respecto a x es: f'(x) = 2(x - 10) El gradiente es simplemente f'(x) en este caso unidimensional.
2. **Actualizar x:** En cada iteración, actualizamos x de la siguiente manera: x\_nuevo = x\_actual - μ \* f'(x\_actual)
3. **Repetir:** Repetimos el paso 2 el número de iteraciones especificado.

**Solución:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Iteración** | **x\_actual** | **f'(x\_actual)** | **x\_nuevo** | **f(x\_nuevo)** |
| 1 | 0 | -20 | 0 - 0.4 \* (-20) = 8 | (8-10)^2 = 4 |
| 2 | 8 | -4 | 8 - 0.4 \* (-4) = 9.6 | (9.6-10)^2 = 0.16 |
| 3 | 9.6 | -0.8 | 9.6 - 0.4 \* (-0.8) = 9.92 | (9.92-10)^2 = 0.0064 |
| 4 | 9.92 | -0.16 | 9.92 - 0.4 \* (-0.16) = 9.984 | (9.984-10)^2 = 0.000256 |

Exportar a Hojas de cálculo

**Respuesta:**

Después de 4 iteraciones, el valor aproximado de x es 9.984 y el valor de la función de pérdida es 0.000256.

**Observación:**

* La función f(x) = (x-10)^2 es una parábola con un mínimo global en x = 10. El descenso de gradiente nos está llevando hacia ese mínimo.
* A medida que aumentamos el número de iteraciones, el valor de x se acercará cada vez más a 10 y el valor de la función de pérdida se acercará a 0.

**Importante:**

* La elección de la tasa de aprendizaje (μ) es crucial. Un valor muy pequeño puede hacer que la convergencia sea lenta, mientras que un valor muy grande puede hacer que el algoritmo diverja.
* El descenso de gradiente puede quedar atrapado en mínimos locales si la función tiene múltiples mínimos.

**Conclusión:**

El algoritmo de descenso de gradiente ha encontrado una aproximación al mínimo de la función dada después de 4 iteraciones. Sin embargo, para obtener una precisión aún mayor, se podrían realizar más iteraciones.

**¿Te gustaría realizar más iteraciones o explorar otro escenario?**

**¿Tienes alguna otra pregunta sobre el descenso de gradiente o sobre este problema en particular?**

**¿Te gustaría visualizar gráficamente cómo el algoritmo converge hacia el mínimo?**

**Podemos analizar también el impacto de cambiar el valor inicial o la tasa de aprendizaje.**

**Descenso de gradiente en Python**

**Vamos a escribir un algoritmo de descenso de gradiente en Python.**

Primero, vamos a hacer un resumen de los pasos necesarios para ejecutar un algoritmo de descenso de gradiente:

1. En los argumentos del algoritmo, establece el valor inicial, x⁰.
2. Calcula el gradiente de la función de pérdida (es el vector de derivadas parciales respecto a cada argumento que toma el vector *x* como entrada).
3. Encuentra un nuevo valor mediante la fórmula:

x1=x0+μ×(−∇f(x))*x*1=*x*0+*μ*×(−∇*f*(*x*))

donde *μ* es el tamaño de paso que se establece en el argumento del algoritmo.

1. Realiza el número de iteraciones especificado en los argumentos.

En esta lección, minimiza la siguiente función:

f(x1,x2)=(x1+x2−1)2+(x1−x2−2)2*f*(*x*1​,*x*2​)=(*x*1​+*x*2​−1)2+(*x*1​−*x*2​−2)2

La función está seleccionada de manera que su mínimo esté en el punto (1,5 ; -0,5).

1.

Escribimos la función *f*. En el código, se denomina func(). Escribe la función gradient(), que calculará su gradiente según la fórmula. Prueba la función con varios vectores (en precódigo).

import numpy as np

def func(x):

return (x[0] + x[1] - 1) \*\* 2 + (x[0] - x[1] - 2) \*\* 2

def gradient(x):

return np.array([4 \* x[0] - 6, 2 \* (x[0] + x[1] - 1) - 2 \* (x[0] - x[1] - 2)])# < escribe tu código aquí >)

print(gradient(np.array([0, 0])))

print(gradient(np.array([0.5, 0.3])))

Resultado

[-6 2]

[-4. 3.2]

¡Es correcto!

Puede que algunas personas no cambien nunca... ¡pero el gradiente de la función seguro que lo hace! De un punto a otro.

2.

Escribe la función gradient\_descent() que aplica el algoritmo de descenso de gradiente a la función f(x). Esta función utiliza:

* initialization — valor inicial del vector *x*
* step\_size — tamaño del paso *μ*
* iterations — número de iteraciones

La función devuelve los valores del vector *x* tras haber realizado el número de iteraciones especificado. Prueba la función con un número diferente de iteraciones (en precódigo)

import numpy as np

def func(x):

return (x[0] + x[1] - 1) \*\* 2 + (x[0] - x[1] - 2) \*\* 2

def gradient(x):

return np.array([4 \* x[0] - 6, 4 \* x[1] + 2])

def gradient\_descent(initialization, step\_size, iterations):

x = initialization# < escribe tu código aquí >

for i in range(iterations):

x = x - step\_size \* gradient(x)# < escribe tu código aquí >

return x# < escribe tu código aquí >

print(gradient\_descent(np.array([0, 0]), 0.1, 5))

print(gradient\_descent(np.array([0, 0]), 0.1, 100))

Resultado

[ 1.38336 -0.46112]

[ 1.5 -0.5]

¡Es correcto!

Lo hemos conseguido. El algoritmo no puede encontrar la solución en cinco iteraciones, pero hacerlo en 100 es pan comido.

**Conclusión**

En este capítulo aprendiste a:

* Calcular el gradiente de la función;
* Encontrar la dirección en la que desciende la función mediante el gradiente negativo;
* Programar el algoritmo de descenso de gradiente.

**Llévate esto contigo**

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Descenso de gradiente](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/Resumen_del_captulo_Descenso_de_gradiente.pdf)
* [Hoja informativa: Descenso de gradiente](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/moved_Hoja_informativa_Descenso_de_gradiente_DS_Sprint_12_Esp.pdf)

En el próximo capítulo, entrenaremos un modelo de regresión lineal y luego utilizaremos el descenso de gradiente para entrenar una red neuronal compuesta por dichos modelos.

Entrenamiento de descenso de gradiente

**Introducción**

**Veamos cómo se usa el descenso de gradiente al entrenar modelos de regresión lineal y redes neuronales.**

**Empezarás por:**

* Escribir un algoritmo para descenso de gradiente estocástico;
* Dominar la regularización de modelos;
* Descubrir cómo funcionan las redes neuronales.

**¿Cuánto tiempo llevará?**

8 lecciones de 10-15 minutos cada una.

Capítulo 4/9

Entrenamiento de descenso de gradiente

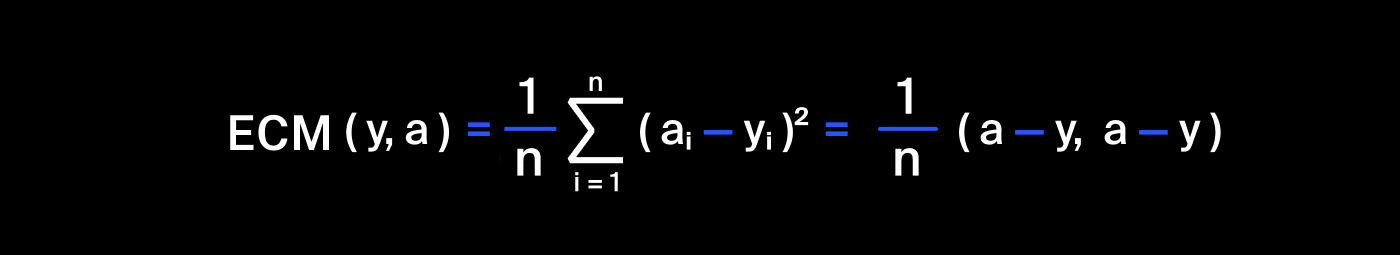
**Descenso de gradiente para regresión lineal**

**Volvamos a la regresión lineal y entrenemos un modelo usando descenso de gradiente.**

Primero, debemos recordar la tarea de entrenamiento de regresión lineal:

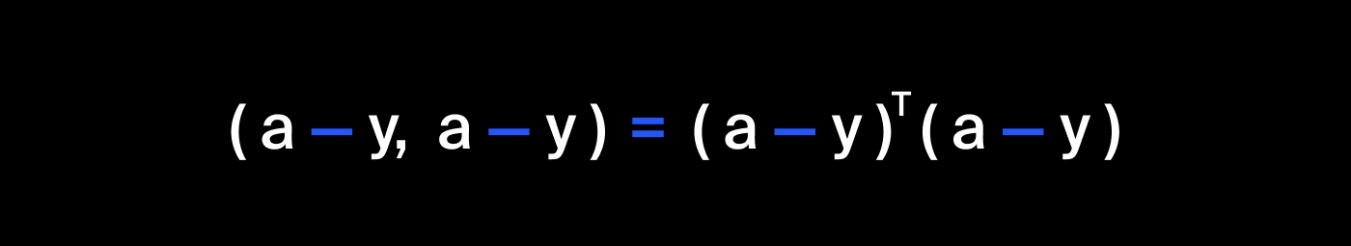


Ahora, vamos a entrenar un modelo usando descenso de gradiente. Pero primero, escribe la función de pérdida en forma de vector para encontrar su gradiente. Expresa el ECM como un producto escalar de la diferencia de vectores:



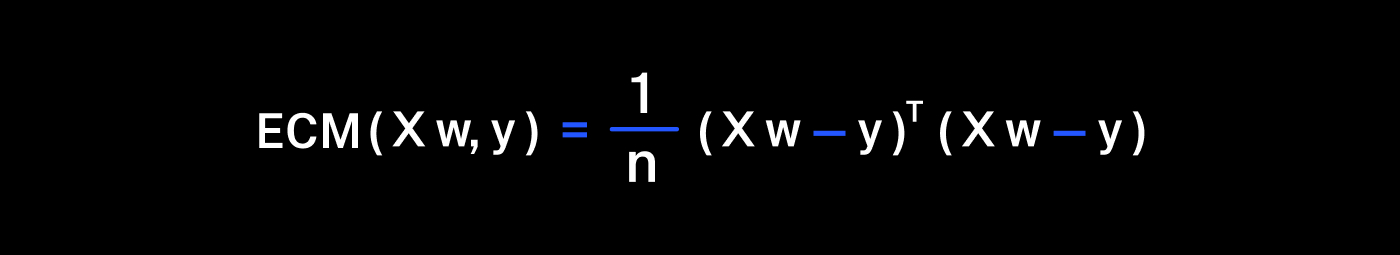
donde *y* es el vector de respuesta correcta y a es el vector de predicción.

Convierte el producto escalar en un producto matricial:

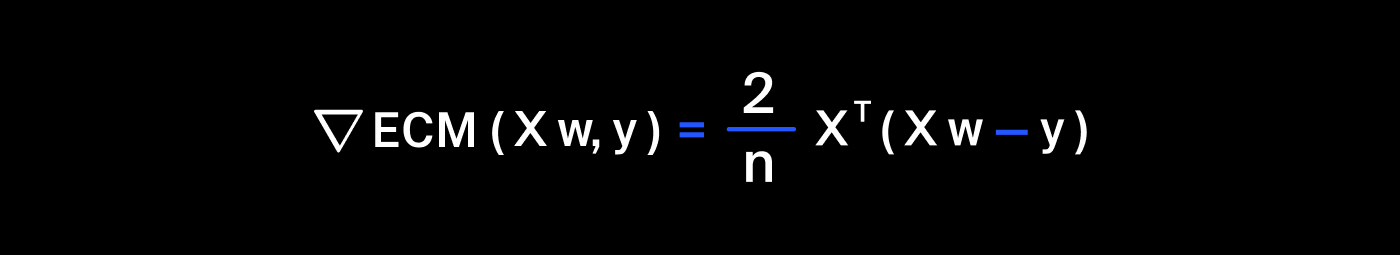


Después de transponer el vector, es decir, convertirlo de una columna a una fila, podemos multiplicarlo por otro vector.

Combina las fórmulas de *ECM* y regresión lineal:



Encuentra la función gradiente para el vector de parámetros *w*. Los gradientes de las funciones con valores vectoriales se calculan de manera similar a las derivadas. Por ejemplo, cuando se trabaja con números, la derivada parcial de *(xw - y)²* para el vector de parámetros w es igual a *2x(xw - y)*. Cuando se trabaja con vectores, solo permanece el factor de w del primer paréntesis ( Xᵀ):



Pregunta

Encuentra la complejidad computacional *T(n, p)* para el gradiente *ECM* en regresión lineal: X.T.dot(X.dot(w) - y). La muestra *X* contiene n observaciones y p características.

*T(n, p) ~ n*

*T(n, p) ~ p*

*T(n, p) ~ np*

¡Correcto!

*T(n, p) ~ np²*

¡Tu comprensión del material es impresionante!

Vamos a explicar la solución.

La fórmula X.T.dot(X.dot(w - y) contiene dos operaciones largas

1. La multiplicación de la matriz *X* por el vector *w*. El tiempo para esta operación es *np.* Para cada observación *n*, se calcula el producto escalar de vectores *p* dimensionales.
2. La multiplicación de la matriz *Xᵀ* por el vector (*Xw - y*). El tiempo para esta operación es *np*: para cada característica *p*, se calcula el producto escalar de vectores *n*dimensionales.

El resultado es *T(n, p) ~ np*.

La complejidad de la solución usando el método directo es *np².* Una iteración del descenso del gradiente es *p* veces más rápida. ¡Pero el número de iteraciones requeridas puede ser más de *p*!

Capítulo 4/9

Entrenamiento de descenso de gradiente

**Descenso de gradiente estocástico**

**Para que el algoritmo de aprendizaje funcione más rápido, podemos reducir el tiempo de una iteración.**

Podemos calcular el gradiente usando pequeñas partes del conjunto de entrenamiento. Estas piezas se conocen como minilotes o lotes. Para que el algoritmo "vea" todo el conjunto de entrenamiento, deben cambiarse sus lotes en cada iteración de forma aleatoria. Aquí necesitamos el descenso de gradiente estocástico en minilotes o descenso de gradiente estocástico, DGE (del griego *στοχαστικός*, "capaz de adivinar"). Ayuda a acelerar el entrenamiento del modelo.

Para obtener lotes, necesitamos barajar todos los datos del conjunto de entrenamiento y dividirlo en partes. Un lote debe contener un promedio de 128 observaciones (u otro valor potencia de 2, e.g., 64, 256, 512, etc). Cuando el algoritmo *DGE* ha pasado por todos los lotes una vez, significa que una época ha terminado. El número de épocas depende del tamaño del conjunto de entrenamiento. Pueden ser una o dos si el conjunto es pequeño o varias docenas si el conjunto es grande. El número de lotes es igual al número de iteraciones para completar una época.



Así es como funciona el algoritmo *DGE*:

1. Hiperparámetros de entrada: tamaño del lote, número de épocas y tamaño del paso.
2. Define los valores iniciales de los pesos del modelo.
3. Divide el conjunto de entrenamiento en lotes para cada época.
4. Para cada lote:

4.1. Calcula el gradiente de la función de pérdida;

4.2. Actualiza los pesos del modelo (agrega el gradiente negativo multiplicado por el tamaño del paso a los pesos actuales).

1. El algoritmo devuelve los pesos finales del modelo.

Encontremos la complejidad computacional del *DGE* con las siguientes definiciones:

* *n* — el número de observaciones en todo el conjunto de entrenamiento;
* *b* — el tamaño del lote;
* *p* — el número de funciones.

Pregunta

¿Cuál es la complejidad computacional del paso para un lote?

*T(n, b, p) ~ nb*

*T(n, b, p) ~ np*

*T(n, b, p) ~ bp*

Es igual a la complejidad computacional del paso para todo el conjunto. Pero el número de observaciones es mucho menor.

*T(n, b, p) ~ nbp*

¡Bien hecho!

Pregunta

¿Cuál es la complejidad computacional de realizar una época si el conjunto se divide integralmente en lotes (es decir, *n/b* es un número entero)?

*T(n, b, p) ~ nb*

*T(n, b, p) ~ np*

¡Correcto! Una época incluye *n/b* iteraciones, *bp* es la complejidad de cada iteración. Solución: \*n/b \* bp = np\*.

*T(n, b, p) ~ bp*

*T(n, b, p) ~ nbp*

¡Bien hecho!

Si el conjunto de datos es grande y el número de épocas es pequeño, la complejidad del entrenamiento es: *T(n, b, p) ~ np*. DGE ayuda a entrenar un modelo de regresión lineal más rápido que usar un método directo

**DGE en Python**

**Escribe un algoritmo *DGE* para un modelo de regresión lineal.**

En esta lección aprenderemos cómo pasar hiperparámetros a un modelo. Necesitamos declarar la clase del modelo y crear el método "inicializador de clase" (\_\_init\_\_):

class SGDLinearRegression:

def \_\_init\_\_(self):

...

Agrega un hiperparámetro step\_size al inicializador de clase:

class SGDLinearRegression:

def \_\_init\_\_(self, step\_size):

...

Ahora podemos pasar el tamaño del paso al modelo al crear una clase:

*# puedes elegir el tamaño del paso de forma arbitraria*

model = SGDLinearRegression(0.01)

Guarda el tamaño del paso como un atributo:

class SGDLinearRegression:

def \_\_init\_\_(self, step\_size):

self.step\_size = step\_size

1.

Inicia el desarrollo del algoritmo DGE con un código ficticio. La descarga de datos y la ejecución de los algoritmos ya está en precódigo. Termina el código de clase del modelo:

* Agrega los hiperparámetros epochs y batch\_size. Agrégalos al inicializador de clase en ese orden y guárdalos en los atributos self.epochs y self.batch\_size.
* En fit(), establece los pesos iniciales (*w*) en cero.
* En predict(), escribe la fórmula para el cálculo de predicción.

Hemos elegido la métrica *R2*. Muestra sus valores en pantalla (en precódigo).

import numpy as np

import pandas as pd

from sklearn.metrics import r2\_score

data\_train = pd.read\_csv('/datasets/train\_data\_n.csv')

features\_train = data\_train.drop(['target'], axis=1)

target\_train = data\_train['target']

data\_test = pd.read\_csv('/datasets/test\_data\_n.csv')

features\_test = data\_test.drop(['target'], axis=1)

target\_test = data\_test['target']

class SGDLinearRegression:

def \_\_init\_\_(self, step\_size,epochs, batch\_size):

self.step\_size = step\_size

self.epochs = epochs

self.batch\_size = batch\_size

# < escribe tu código aquí >)

def fit(self, train\_features, train\_target):

X = np.concatenate(

(np.ones((train\_features.shape[0], 1)), train\_features), axis=1

)

y = train\_target

w = np.zeros(X.shape[1]) # < escribe tu código aquí >)

# escribirás el algoritmo DGE aquí en los próximos ejercicios

self.w = w[1:]

self.w0 = w[0]

def predict(self, test\_features):

return test\_features.dot(self.w) + self.w0

# < escribe tu código aquí >

# ya se han superado los parámetros de entrenamiento adecuados

model = SGDLinearRegression(0.01, 1, 200)

model.fit(features\_train, target\_train)

pred\_train = model.predict(features\_train)

pred\_test = model.predict(features\_test)

print(r2\_score(target\_train, pred\_train).round(5))

print(r2\_score(target\_test, pred\_test).round(5))

Resultado

-0.16938

-0.14579

¡Es correcto!

El valor R2 es menor que cero. Rendimiento de código dummy clásico.

2.

Agrega bucles por épocas y lotes, sin tener en cuenta por ahora el paso de gradiente negativo.

Necesitas:

* Encontrar el número de lotes;
* Encontrar el comienzo del lote *i*: el índice del primer elemento;
* Encontrar el final del lote *i*: el comienzo del siguiente lote.

Muestra en la pantalla los valores *R2* (en precódigo).

import numpy as np

import pandas as pd

from sklearn.metrics import r2\_score

data\_train = pd.read\_csv('/datasets/train\_data\_n.csv')

features\_train = data\_train.drop(['target'], axis=1)

target\_train = data\_train['target']

data\_test = pd.read\_csv('/datasets/test\_data\_n.csv')

features\_test = data\_test.drop(['target'], axis=1)

target\_test = data\_test['target']

class SGDLinearRegression:

def \_\_init\_\_(self, step\_size, epochs, batch\_size):

self.step\_size = step\_size

self.epochs = epochs

self.batch\_size = batch\_size

def fit(self, train\_features, train\_target):

X = np.concatenate(

(np.ones((train\_features.shape[0], 1)), train\_features), axis=1

)

y = train\_target

w = np.zeros(X.shape[1])

for \_ in range(self.epochs):

batches\_count = X.shape[0] // self.batch\_size

for i in range(batches\_count):

begin = i \* self.batch\_size

end = (i + 1) \* self.batch\_size

X\_batch = X[begin:end, :]

y\_batch = y[begin:end]

# escribirás el paso de gradiente negativo aquí en el próximo ejercicio

self.w = w[1:]

self.w0 = w[0]

def predict(self, test\_features):

return test\_features.dot(self.w) + self.w0

model = SGDLinearRegression(0.01, 10, 100)

model.fit(features\_train, target\_train)

pred\_train = model.predict(features\_train)

pred\_test = model.predict(features\_test)

print(r2\_score(target\_train, pred\_train).round(5))

print(r2\_score(target\_test, pred\_test).round(5))

Resultado

-0.16938

-0.14579

¡Es correcto!

En realidad, el resultado no ha cambiado. Solo era una prueba para ver si estabas prestando atención. :) Por ahora solo estábamos observando los lotes.

3.

Termina de escribir el algoritmo DGE:

* Encuentra el gradiente para el lote;
* Crea un paso a lo largo del gradiente negativo para los pesos.

Muestra en la pantalla el resultado del R2 (en precódigo).

import numpy as np

import pandas as pd

from sklearn.metrics import r2\_score

data\_train = pd.read\_csv('/datasets/train\_data\_n.csv')

features\_train = data\_train.drop(['target'], axis=1)

target\_train = data\_train['target']

data\_test = pd.read\_csv('/datasets/test\_data\_n.csv')

features\_test = data\_test.drop(['target'], axis=1)

target\_test = data\_test['target']

class SGDLinearRegression:

def \_\_init\_\_(self, step\_size, epochs, batch\_size):

self.step\_size = step\_size

self.epochs = epochs

self.batch\_size = batch\_size

def fit(self, train\_features, train\_target):

X = np.concatenate(

(np.ones((train\_features.shape[0], 1)), train\_features), axis=1

)

y = train\_target

w = np.zeros(X.shape[1])

for \_ in range(self.epochs):

batches\_count = X.shape[0] // self.batch\_size

for i in range(batches\_count):

begin = i \* self.batch\_size

end = (i + 1) \* self.batch\_size

X\_batch = X[begin:end, :]

y\_batch = y[begin:end]

gradient = 2 \* X\_batch.T.dot(X\_batch.dot(w) - y\_batch) / X\_batch.shape[0]# < escribe tu código aquí >

w -= self.step\_size \* gradient# < escribe tu código aquí >

self.w = w[1:]

self.w0 = w[0]

def predict(self, test\_features):

return test\_features.dot(self.w) + self.w0

model = SGDLinearRegression(0.01, 10, 100)

model.fit(features\_train, target\_train)

pred\_train = model.predict(features\_train)

pred\_test = model.predict(features\_test)

print(r2\_score(target\_train, pred\_train).round(5))

print(r2\_score(target\_test, pred\_test).round(5))

Resultado

0.21882

0.06296

¡Es correcto!

¡R2 es mayor que cero! Colocamos los lotes donde pertenecen y todo comenzó a funcionar. Desafortunadamente, el modelo se sobreajusta, pero lo arreglaremos en la próxima lección.

**Regularización de regresión lineal**

**Vamos a modificar la función de pérdida para eliminar el sobreajuste.**

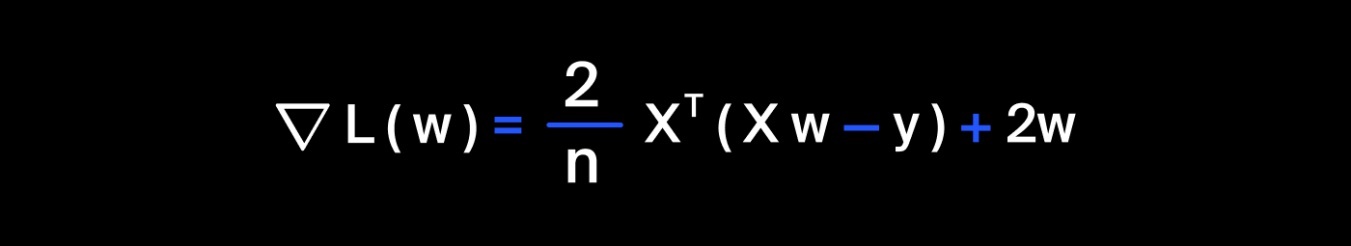
Para reducir el sobreajuste, podemos usar la regularización. Lo que hace es "refinar" el modelo si los valores de los parámetros complican la operación del algoritmo. Para un modelo de regresión lineal, la regularización implica la limitación de pesos.

Generalmente, cuanto más bajos sean los valores de los pesos, más fácil es entrenar el algoritmo. Para averiguar qué tan grandes son los pesos, calculamos la distancia entre el vector de peso y el vector que consta de ceros. Por ejemplo, la distancia euclidiana *d₂*(*w*, 0) es igual al producto escalar de los pesos solos: (*w, w*).

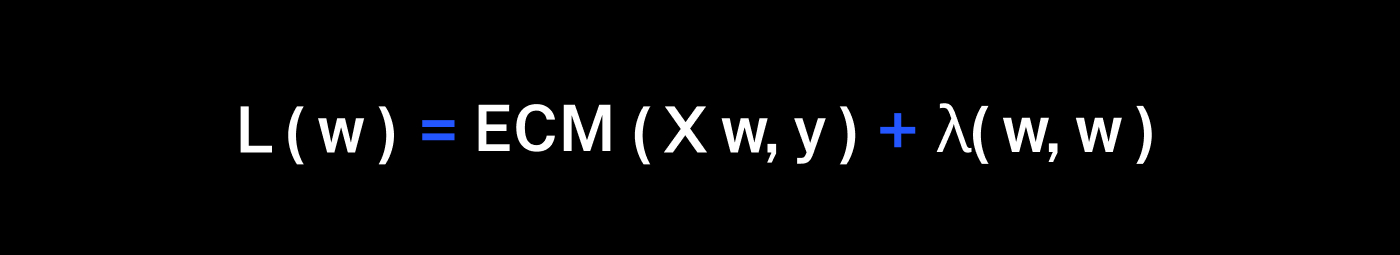
Para limitar los valores de peso, incluye el producto escalar de los pesos en la fórmula de la función de pérdida:

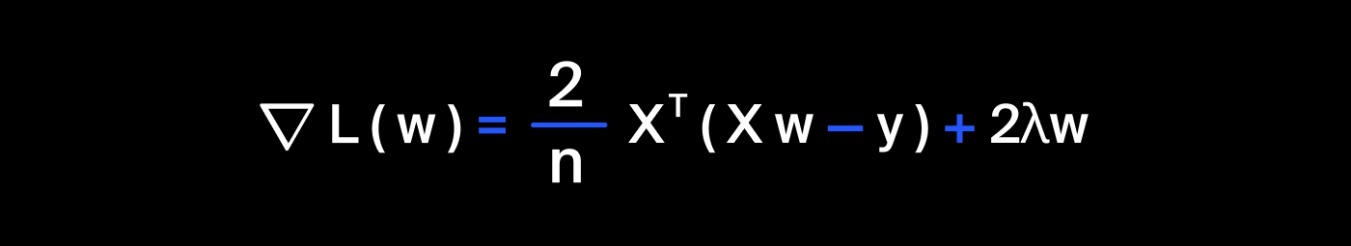


La derivada (w, w) es igual a *2w*. Calcula el gradiente de la función de pérdida:



Para controlar la magnitud de la regularización, agrega el peso de regularización a la fórmula de la función de pérdida. Se denota como *λ*.

  El peso de regularización también se agrega a la fórmula de cálculo de gradiente:



Cuando usamos la distancia euclidiana para la regularización del peso, dicha regresión lineal se denomina regresión de cresta.

Agrega la regularización al modelo.

Finaliza el código de regularización:

* iguala el elemento con el índice cero en el vector reg a cero (por lo general, el cambio no se incluye en la regularización);
* agrega el sumando de regularización a la función gradiente.

Al código del ejercicio anterior le hemos añadido:

* un nuevo hiperparámetro, el peso de regularización (reg\_weight);
* entrenamiento y pruebas con varios valores.

Muestra en la pantalla los valores de peso de regularización y *R2* (en precódigo).

import numpy as np

import pandas as pd

from sklearn.metrics import r2\_score

data\_train = pd.read\_csv('/datasets/train\_data\_n.csv')

features\_train = data\_train.drop(['target'], axis=1)

target\_train = data\_train['target']

data\_test = pd.read\_csv('/datasets/test\_data\_n.csv')

features\_test = data\_test.drop(['target'], axis=1)

target\_test = data\_test['target']

class SGDLinearRegression:

def \_\_init\_\_(self, step\_size, epochs, batch\_size, reg\_weight):

self.step\_size = step\_size

self.epochs = epochs

self.batch\_size = batch\_size

self.reg\_weight = reg\_weight

def fit(self, train\_features, train\_target):

X = np.concatenate(

(np.ones((train\_features.shape[0], 1)), train\_features), axis=1

)

y = train\_target

w = np.zeros(X.shape[1])

for \_ in range(self.epochs):

batches\_count = X.shape[0] // self.batch\_size

for i in range(batches\_count):

begin = i \* self.batch\_size

end = (i + 1) \* self.batch\_size

X\_batch = X[begin:end, :]

y\_batch = y[begin:end]

gradient = (

2

\* X\_batch.T.dot(X\_batch.dot(w) - y\_batch)

/ X\_batch.shape[0]

)

# copia el vector w para evitar cambios

reg = 2 \* w.copy()

reg[0] = 0# < escribe tu código aquí >

gradient += self.reg\_weight \* reg# < escribe tu código aquí >

w -= self.step\_size \* gradient

self.w = w[1:]

self.w0 = w[0]

def predict(self, test\_features):

return test\_features.dot(self.w) + self.w0

# Para comparar la regresión de cresta y la regresión lineal,

# comienza con el peso de regularización = 0.Luego agrega

# el entrenamiento con sus diversos valores.

print('Regularization:', 0.0)

model = SGDLinearRegression(0.01, 10, 100, 0.0)

model.fit(features\_train, target\_train)

pred\_train = model.predict(features\_train)

pred\_test = model.predict(features\_test)

print(r2\_score(target\_train, pred\_train).round(5))

print(r2\_score(target\_test, pred\_test).round(5))

print('Regularization:', 0.1)

model = SGDLinearRegression(0.01, 10, 100, 0.1)

model.fit(features\_train, target\_train)

pred\_train = model.predict(features\_train)

pred\_test = model.predict(features\_test)

print(r2\_score(target\_train, pred\_train).round(5))

print(r2\_score(target\_test, pred\_test).round(5))

print('Regularization:', 1.0)

model = SGDLinearRegression(0.01, 10, 100, 1.0)

model.fit(features\_train, target\_train)

pred\_train = model.predict(features\_train)

pred\_test = model.predict(features\_test)

print(r2\_score(target\_train, pred\_train).round(5))

print(r2\_score(target\_test, pred\_test).round(5))

print('Regularization:', 10.0)

model = SGDLinearRegression(0.01, 10, 100, 10.0)

model.fit(features\_train, target\_train)

pred\_train = model.predict(features\_train)

pred\_test = model.predict(features\_test)

print(r2\_score(target\_train, pred\_train).round(5))

print(r2\_score(target\_test, pred\_test).round(5))

Resultado

/usr/local/lib/python3.9/site-packages/joblib/\_multiprocessing\_helpers.py:46: UserWarning: [Errno 2] No such file or directory. joblib will operate in serial mode

warnings.warn('%s. joblib will operate in serial mode' % (e,))

Regularization: 0.0

0.21882

0.06296

Regularization: 0.1

0.21488

0.07001

Regularization: 1.0

0.16661

0.08061

Regularization: 10.0

0.03945

0.02412

Resultado

/usr/local/lib/python3.9/site-packages/joblib/\_multiprocessing\_helpers.py:46: UserWarning: [Errno 2] No such file or directory. joblib will operate in serial mode

warnings.warn('%s. joblib will operate in serial mode' % (e,))

Regularization: 0.0

0.21882

0.06296

Regularization: 0.1

0.21488

0.07001

Regularization: 1.0

0.16661

0.08061

Regularization: 10.0

0.03945

0.02412

Capítulo 4/9 · Faltan 3 lecciones

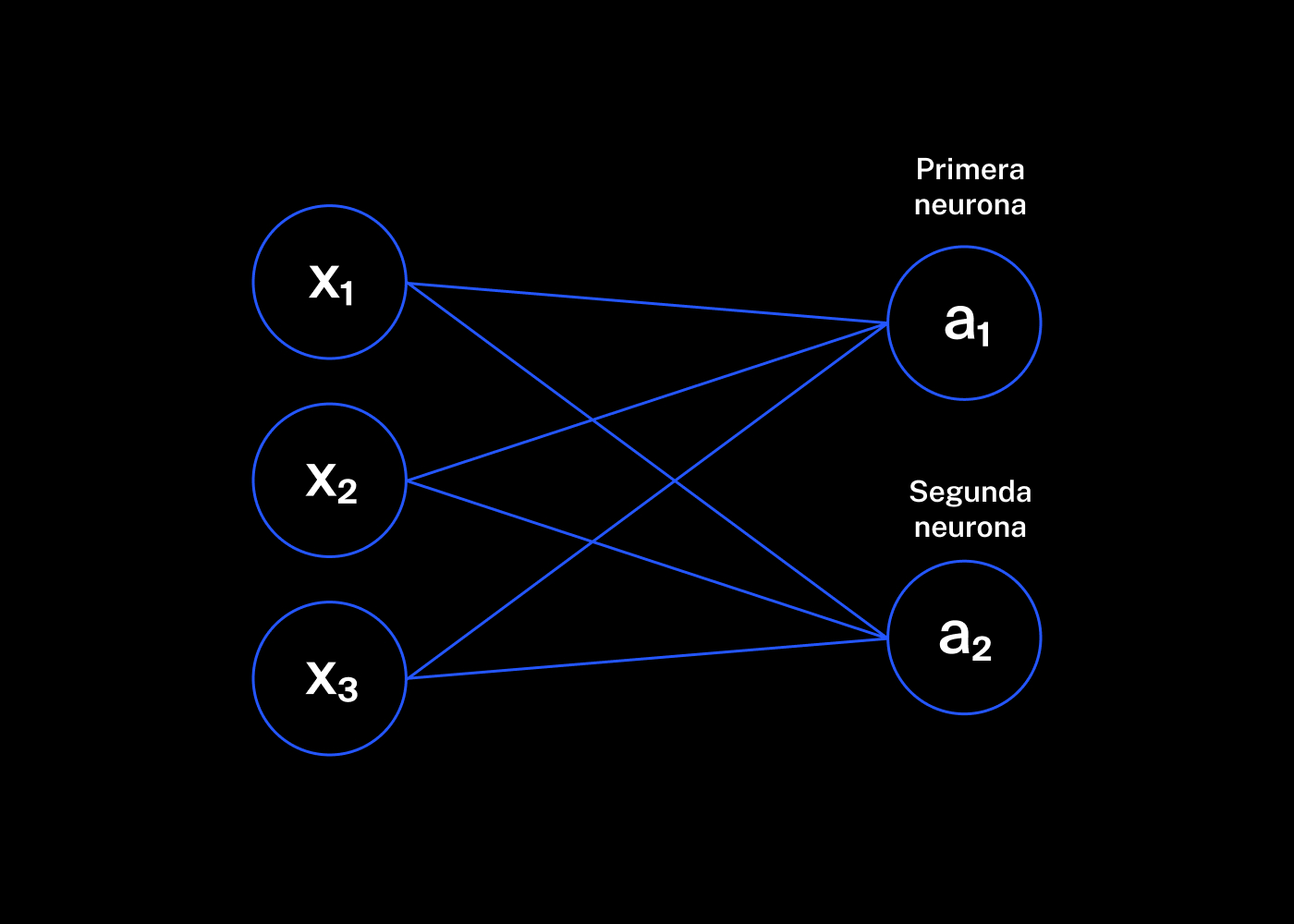
Entrenamiento de descenso de gradiente

**Fundamentos de redes neuronales**

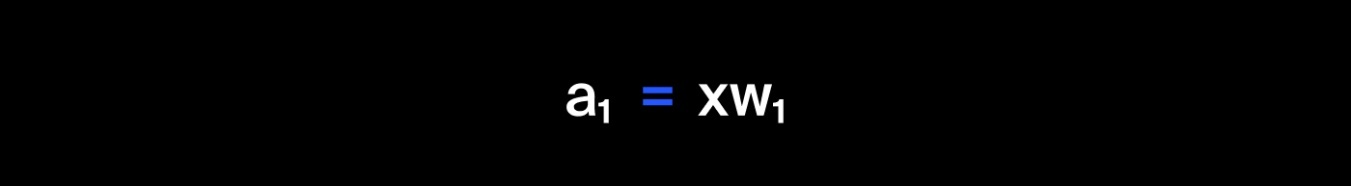
**El algoritmo *DGE* se puede utilizar para entrenar no solo regresiones lineales, sino también redes neuronales.**

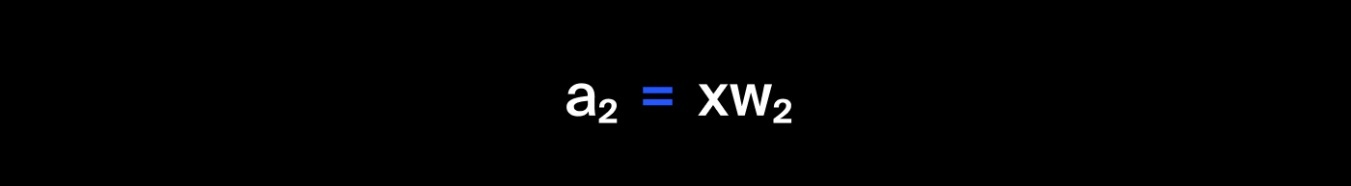
Una red neuronal es un modelo que consta de muchos modelos simples (por ejemplo, modelos de regresión lineal). El nombre proviene de la biología: una red neuronal artificial utiliza el principio de una operación de red de células neuronales donde las neuronas construyen relaciones complejas entre los datos de entrada y salida.

Este es un ejemplo de una red neuronal con tres entradas *x₁, x₂, x₃* y dos salidas *а₁, а₂*:



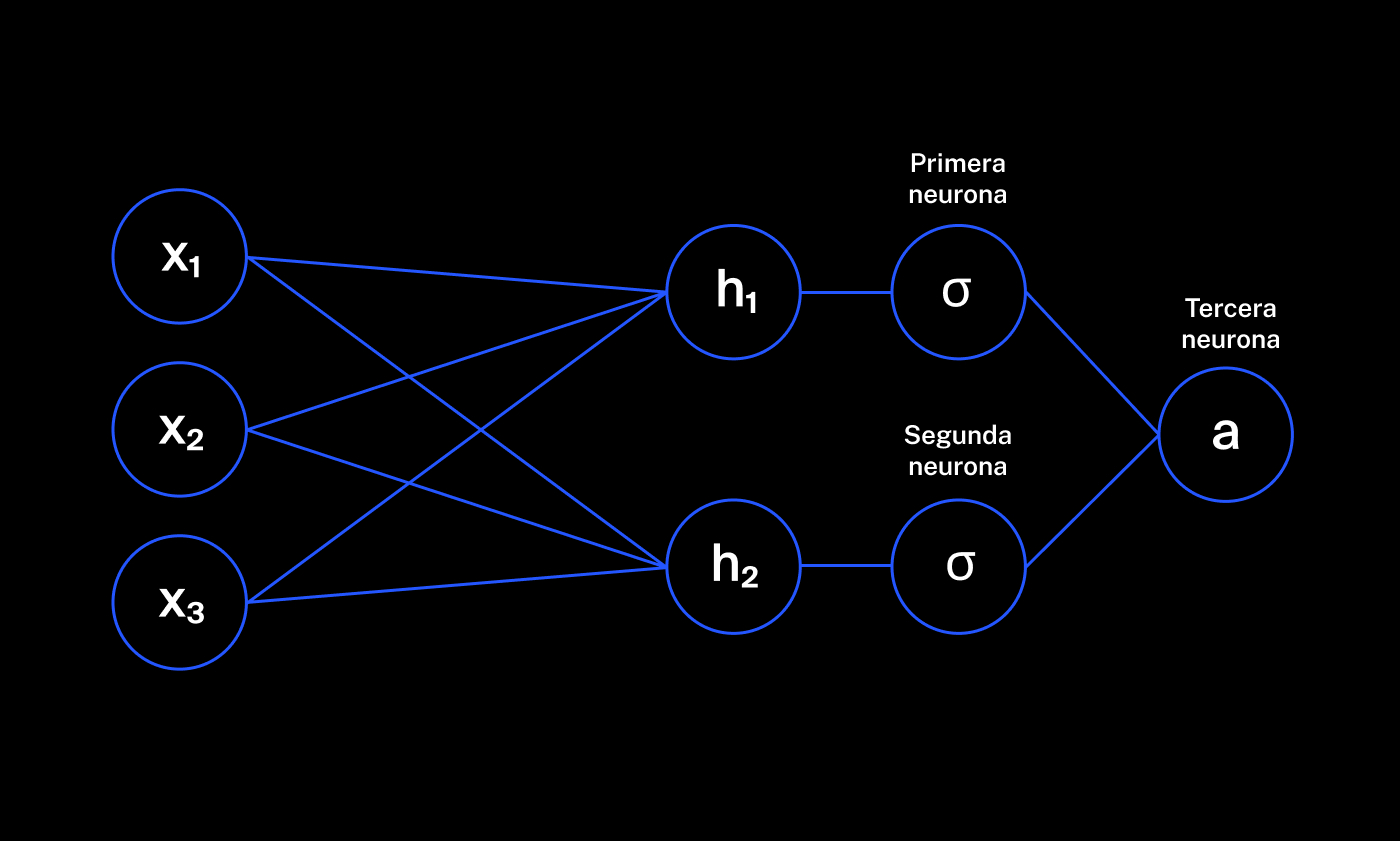
El valor de cada salida, o neurona, se calcula de la misma manera que una predicción de regresión lineal. Aquí, x es un vector, x = (x1, x2, x3):



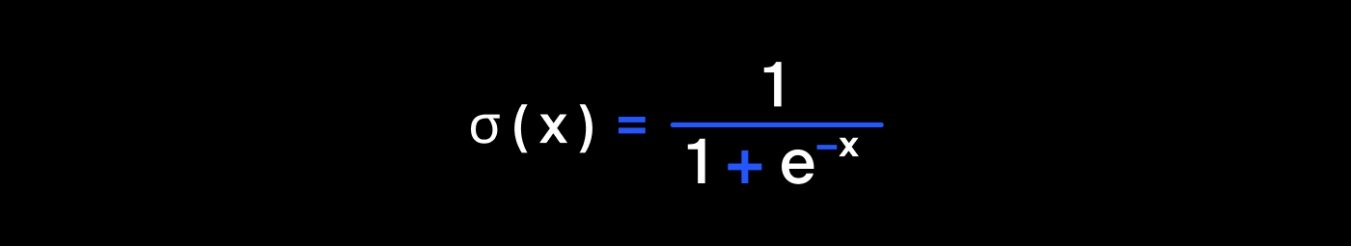


Cada valor de salida tiene sus propios pesos (*w₁* y *w₂*).

Aquí hay otro ejemplo: La red tiene tres entradas *x₁, x₂, x₃*, dos variables ocultas *h₁* y *h₂,* y una salida *а*.



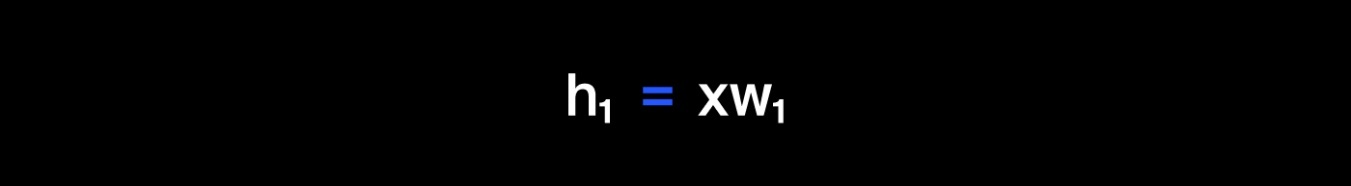
Los valores *h₁* y *h₂* se pasan a la función logística *σ(x)*:

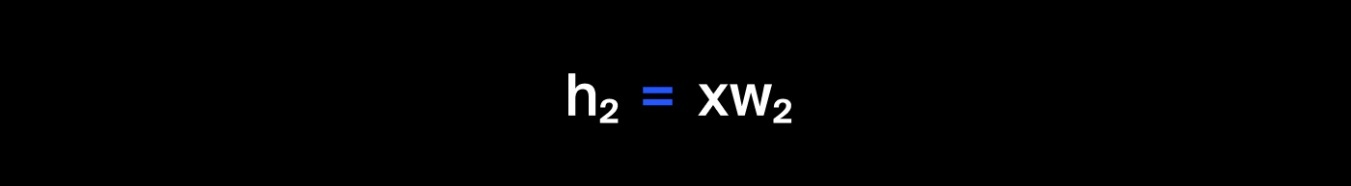


donde e es el número de Euler (aproximadamente 2.718281828).

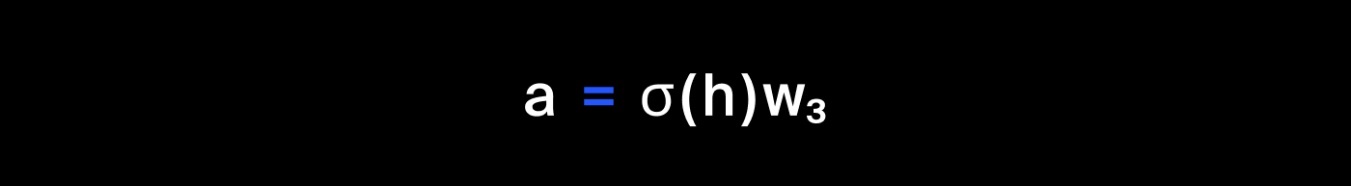
La función logística en una red neuronal se denomina función de activación. Se incluye en la neurona después de multiplicar los valores de entrada por los pesos, cuando las salidas de la neurona se convierten en entradas para otras neuronas. De esta manera, podemos describir dependencias más complejas.

Cada variable oculta *(h₁, h₂)* es igual al valor de entrada multiplicado por un peso:





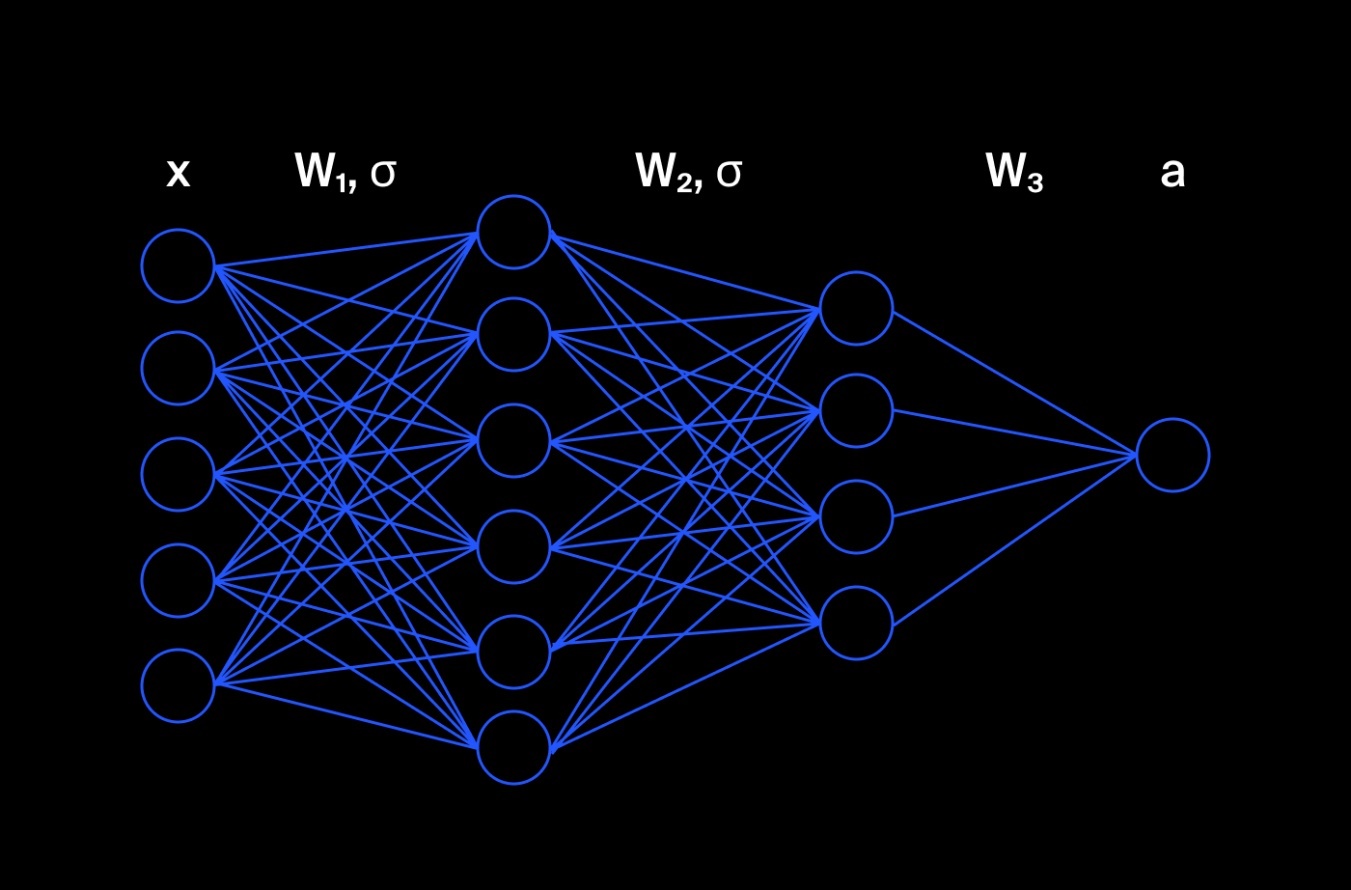
Por comodidad, expresamos las variables ocultas *h₁* y *h₂* como vector *h*. Aquí está la fórmula para calcular la predicción de la red neuronal:



Escribe esta fórmula usando una función con valores vectoriales. Denota los pesos *w₁* y *w₂* como columnas de la matriz *W*. Se obtiene la siguiente fórmula vectorial:



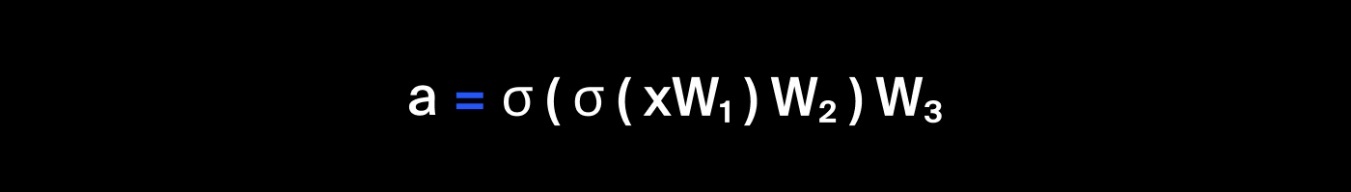
Si ponemos los pesos de varias neuronas en matrices, podemos obtener una red aún más compleja, por ejemplo:



donde:

* *x* — vector de entrada con dimensión *p* (número de características)
* *W₁* — matriz con dimensión *p x m*
* *W₂* — matriz con dimensión *m x k*
* *W₃* — matriz con dimensión *k x 1*
* a — predicción del modelo (número único)

Cuando una red neuronal de este tipo calcula una predicción, realiza todas las operaciones de forma secuencial:



Pregunta

Elige todas las afirmaciones correctas:

Elige tantas como quieras

Una red neuronal no puede tener más de tres matrices.

La regresión lineal es una red neuronal.

Correcto. Tal red tendrá solo una neurona.

Es imposible calcular una predicción de una red neuronal sin valores conocidos de pesos.

Correcto. Los pesos afectan a los modelos y sus predicciones.

¡Lo has entendido bien!

Capítulo 4/9 · Faltan 2 lecciones

Entrenamiento de descenso de gradiente

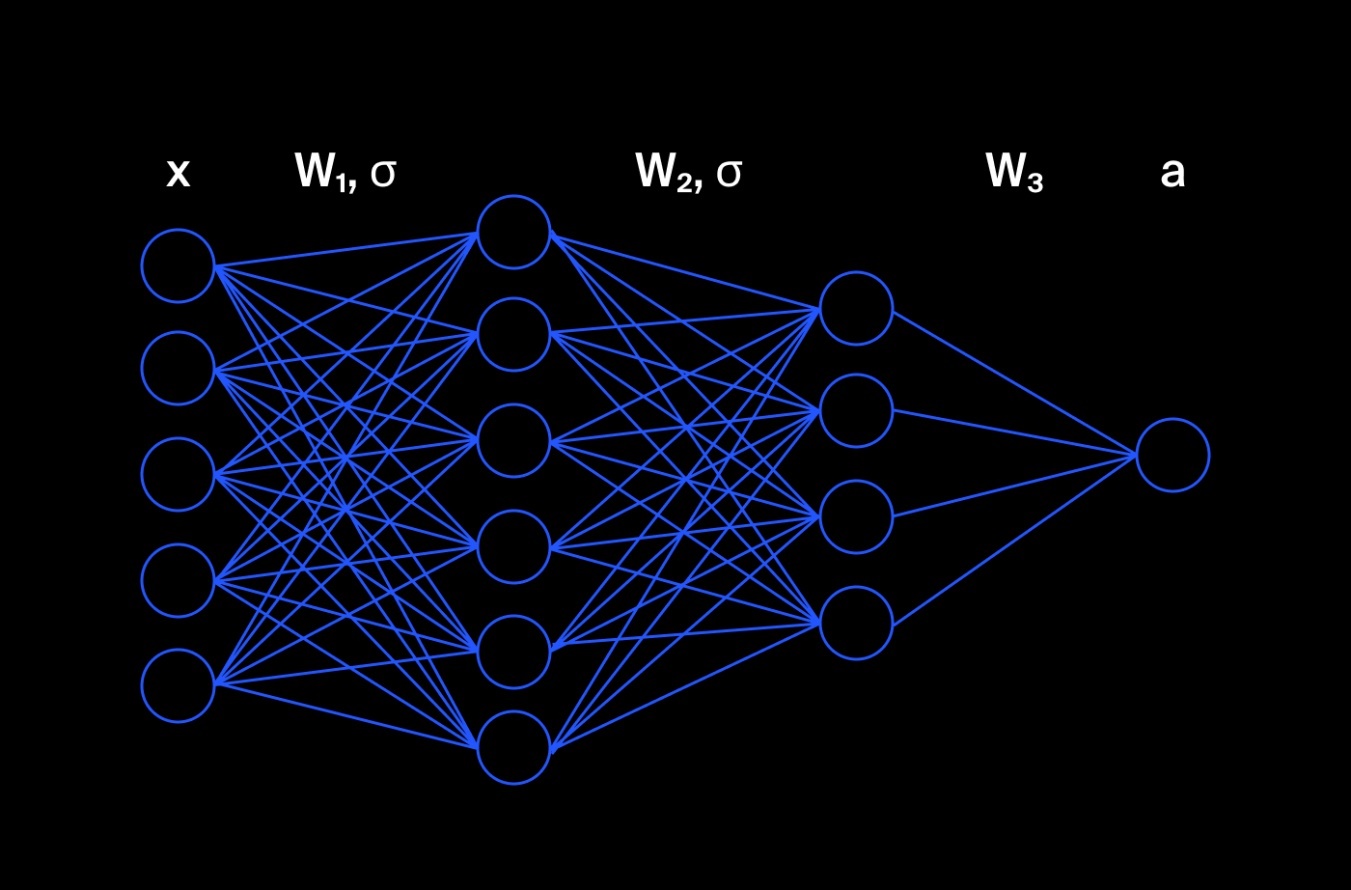
**Entrenamiento de redes neuronales**

**Para entrenar una red neuronal, necesitamos establecer el objetivo de entrenamiento.**

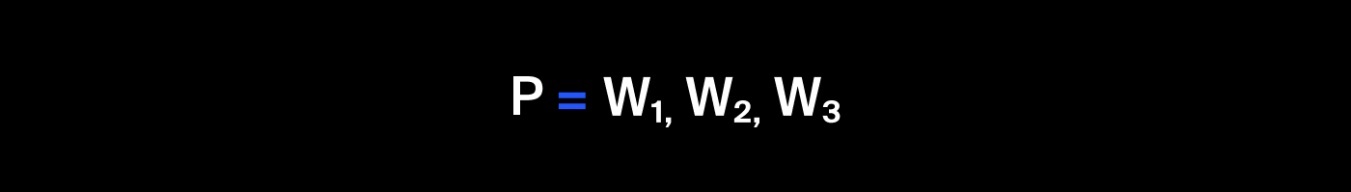
Cualquier red neuronal se puede escribir como una función a partir de su vector de entrada y sus parámetros. Definamos lo siguiente:

* *X* — características del conjunto de entrenamiento
* *P* — conjunto de todos los parámetros de la red neuronal
* *N(X, P) —* función de red neuronal

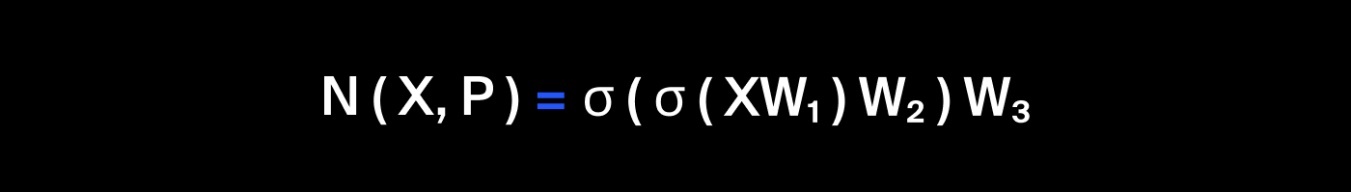
Tomemos esta red neuronal:



Los parámetros de la red neuronal son pesos en las neuronas:



Aquí está la función de red neuronal:



También vamos a definir:

* *y* — respuestas del conjunto de entrenamiento
* *L(a, y)* — función de pérdida (por ejemplo, *ECM)*

Entonces podemos declarar el objetivo del entrenamiento de redes neuronales de la siguiente manera:

Texto

Descripción generada automáticamente

El mínimo de esta función también se puede encontrar usando el algoritmo *DGE*.

El algoritmo de aprendizaje de la red neuronal es el mismo que el algoritmo DGE para la regresión lineal. Solo calculamos el gradiente de la red neuronal, en lugar del gradiente para la regresión lineal.



Ya está. Ahora conoces los conceptos básicos del entrenamiento de redes neuronales. Más adelante habrá un curso completo dedicado a las redes neuronales.

Entrenamiento de descenso de gradiente

**Conclusión**

En este capítulo aprendiste a:

* Programar el algoritmo *DGE*;
* Reducir el sobreajuste mediante la regularización;
* Construir y entrenar redes neuronales.

**Llévate esto contigo**

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Entrenamiento del descenso de gradiente](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/Resumen_del_capi%CC%81tulo_Entrenamiento_del_descenso.pdf)
* [Hoja informativa: Entrenamiento del descenso de gradiente](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/moved_Hoja_informativa_Entrenamiento_del_descenso_de_gradiente_DS_Sprint_12_ESP.pdf)

El siguiente capítulo tratará sobre la potenciación del gradiente.

Potenciación del gradiente

**Introducción**

**En este capítulo, echaremos un vistazo al modelo de potenciación del gradiente.**

**Empezarás por:**

* Descubrir qué es la potenciación del gradiente;
* Aprender a reducir el sobreajuste de un modo de potenciación del gradiente;
* Explorar la librería CatBoost.

**¿Cuánto tiempo llevará?**

6 lecciones de 10-15 minutos cada una.

Capítulo 5/9

Potenciación del gradiente

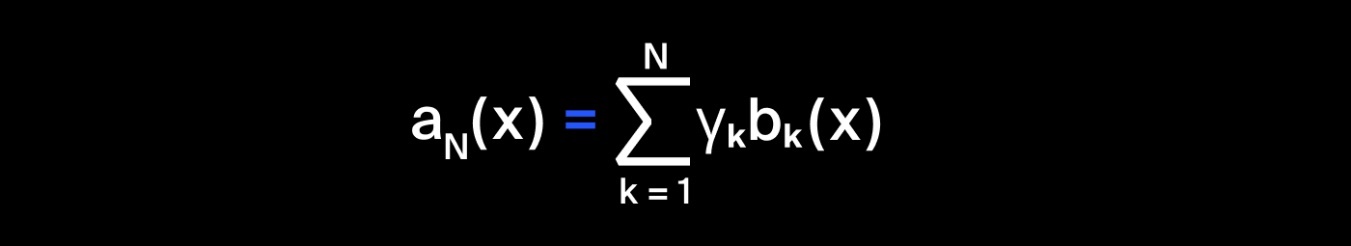
**Ensambles y potenciación**

**Averigüemos qué son los ensambles y cómo ayudan a mejorar los modelos.**

Un ensamble es un conjunto de modelos para resolver el mismo problema. La fortaleza de los ensambles es que el error medio de un grupo de modelos es menos significativo que sus errores individuales.

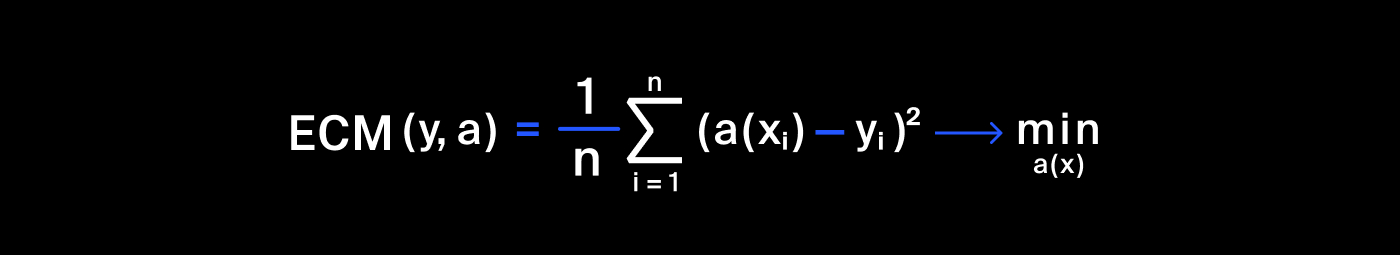
Ya conoces uno de los tipos de modelos de ensamble: el bosque aleatorio. En un bosque aleatorio se promedian los resultados de los alumnos base o débiles (modelos que componen el ensamble). Los clasificadores base para un bosque aleatorio son árboles de decisión.

Otro enfoque para la construcción de ensambles es la potenciación, donde cada modelo posterior considera los errores del anterior y, en la predicción final, los pronósticos de los alumnos básicos. Echa un vistazo:

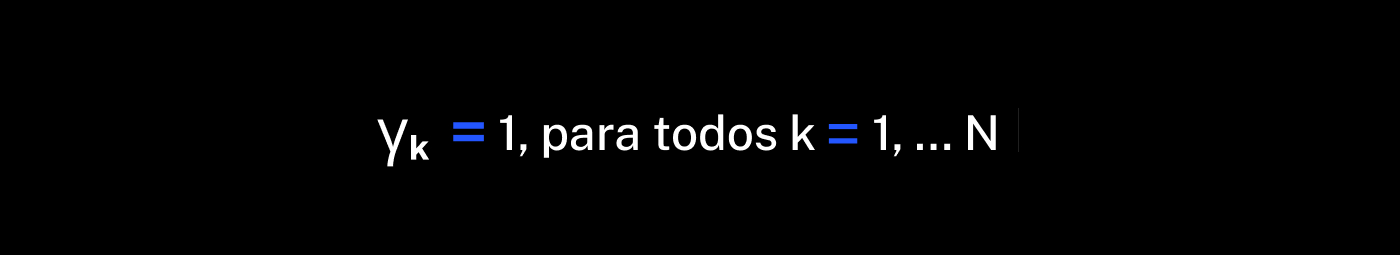


donde aN(x)*aN*(*x*) es la predicción del ensamble, *N* es el número de alumnos base, bk(x)*bk*(*x*) es la predicción del alumno base, y γk*γk* es la ponderación del modelo.

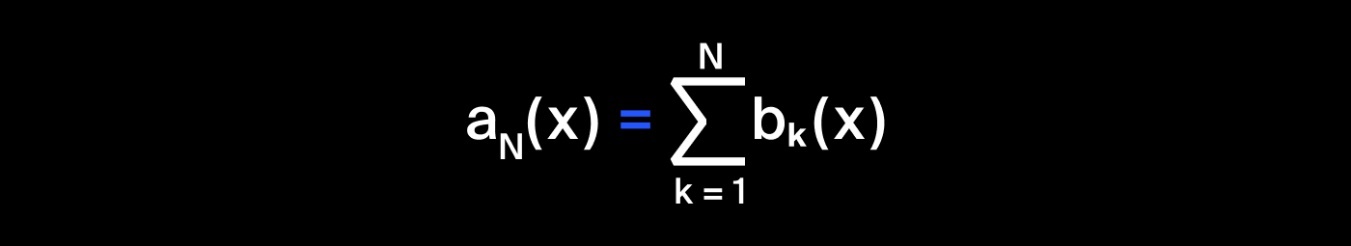
Por ejemplo, estamos tratando con una tarea de regresión. Tenemos *n* observaciones con características *x* y respuestas correctas *y*. Nuestra tarea es minimizar la función de pérdida *ECM*:



Por conveniencia, iguala los pesos del modelo a la unidad:

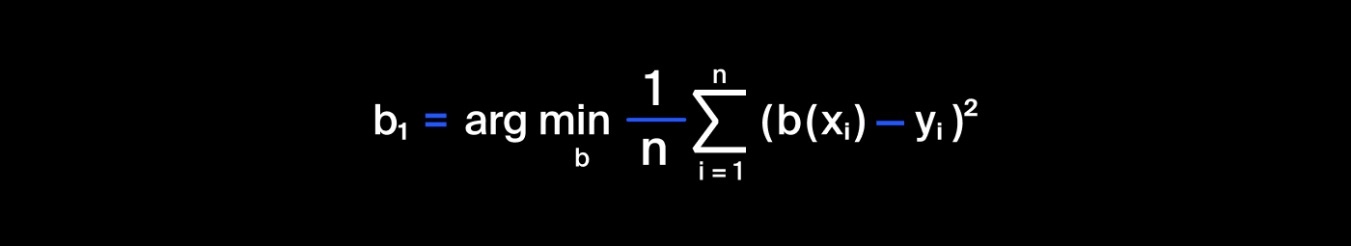


Obtenemos:

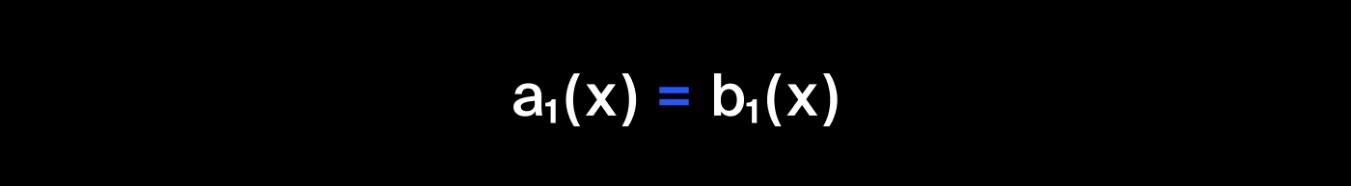


Ahora creamos un conjunto de modelos secuenciales.

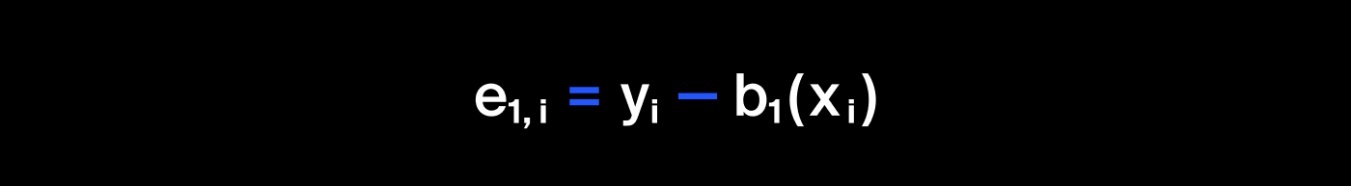
Primero, construye al alumno base *b₁* resolviendo la tarea de minimización:



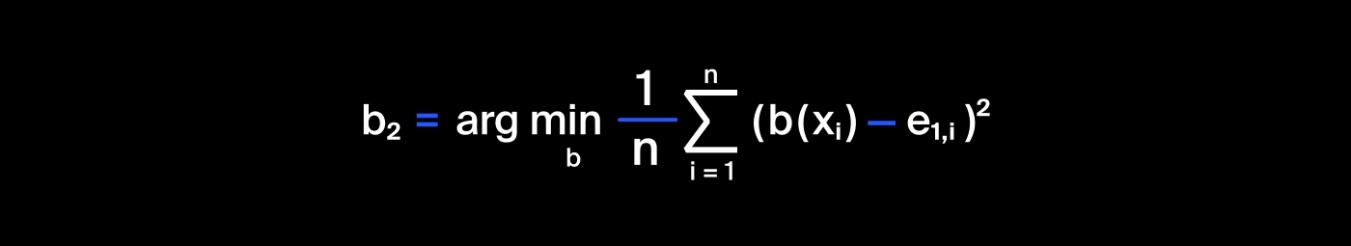
El resultado es este ensamble:



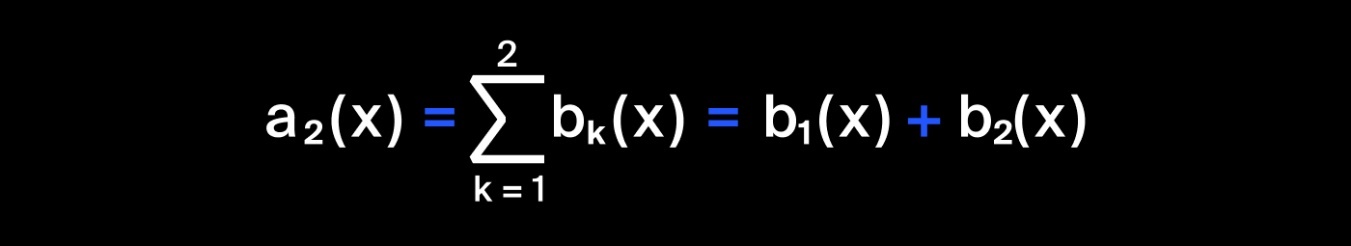
Indica el residuo. Es la diferencia entre la predicción en el primer paso y las respuestas correctas:



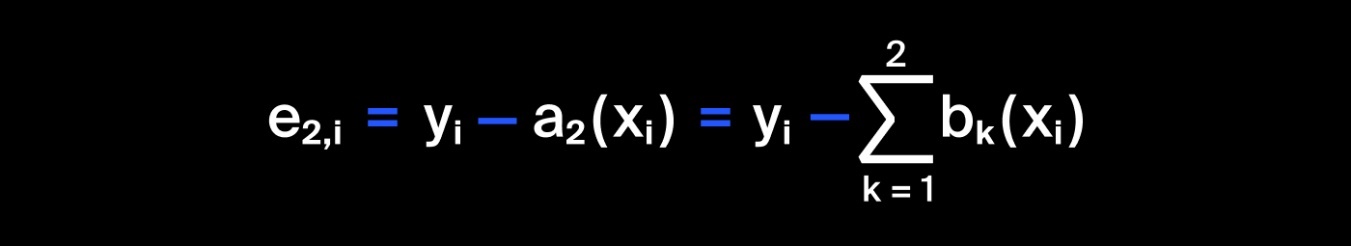
En el segundo paso, construimos el modelo de esta manera:



El ensamble tomará la siguiente forma:

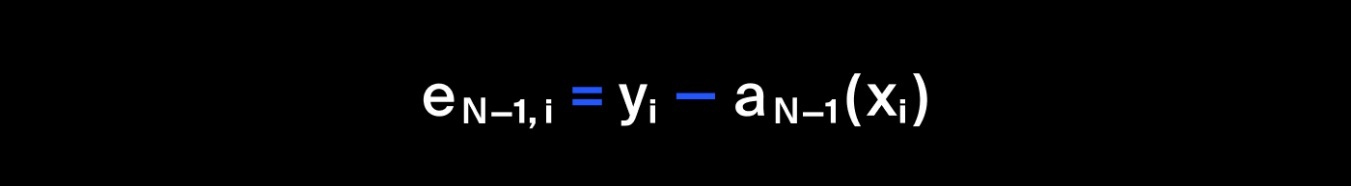


Encuentra el residuo para cada observación *i*:

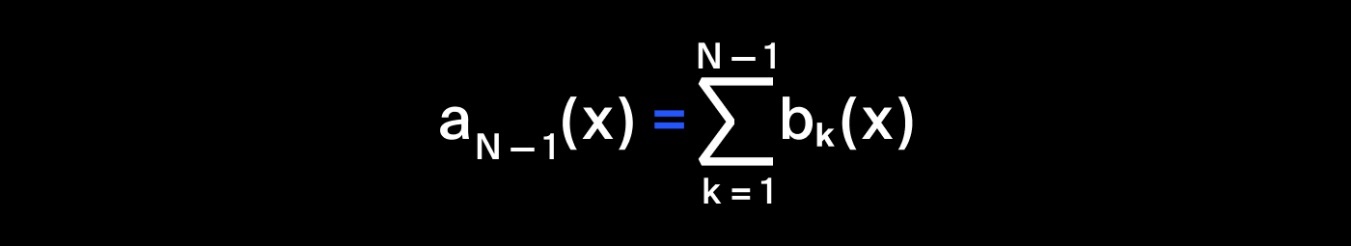


En cada paso siguiente, el algoritmo minimiza el error de ensamble del paso anterior.

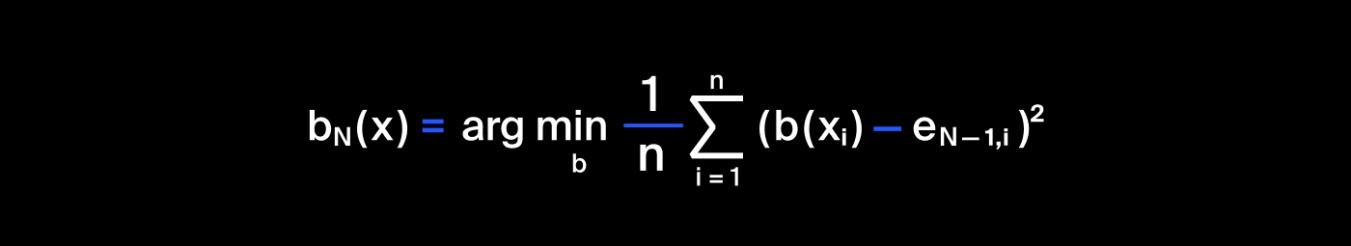
Vamos a resumir las fórmulas. En el paso *N-1*, el residuo se calcula de la siguiente manera:



El ensamble en sí se representa como la suma de las predicciones de todos los alumnos base combinados hasta este paso:



Entonces, en el paso N, el algoritmo elegirá el modelo con el error de ensamble en el paso N-1:



Pregunta

Elige la definición de potenciación:

Minimización de la función de error mediante la actualización secuencial de parámetros

Encontrar la predicción final promediando los resultados de los modelos

Construcción secuencial de un conjunto de modelos a través de la mejora de la predicción en cada paso

¡Correcto!

¡Buen trabajo!

Pregunta

¿Qué predice el alumno base en cada paso de la potenciación del gradiente?

La función de pérdida de ensamble en el paso anterior *N-1*

Residuos en el paso anterior *N-1*

Correcto. Predecimos los residuos del ensamble del paso anterior, minimizando así la función de error.

La diferencia entre la predicción actual del alumno base y la predicción en el paso anterior

¡Bien hecho!

Pregunta

¿Qué son los residuos del ensamble en el paso *N*?

La diferencia entre el ensamble en el paso actual *N* y las respuestas reales

¡Correcto!

La diferencia entre las predicciones base actuales del alumno y las respuestas reales

La diferencia entre la predicción del ensamble actual y la predicción anterior en el paso *N-1*

¡Bien hecho!

Pregunta

¿Qué se minimiza en el paso *N*?

La diferencia entre las predicciones en el paso anterior y las respuestas correctas

La función de error de la predicción del ensamble en el paso anterior

El valor de la función de error para predecir los residuos en el paso anterior

¡Correcto!

¡Excelente trabajo!

Capítulo 5/9

Potenciación del gradiente

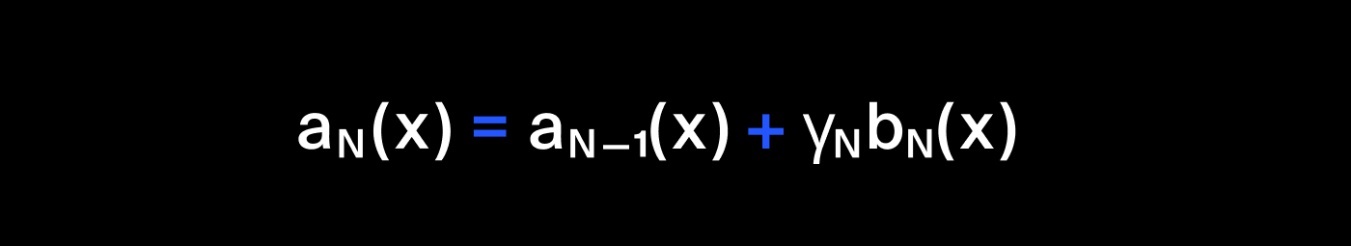
**Potenciación del gradiente**

**Si combinamos la potenciación y el descenso de gradiente, ¡obtenemos una**potenciación del gradiente**!**

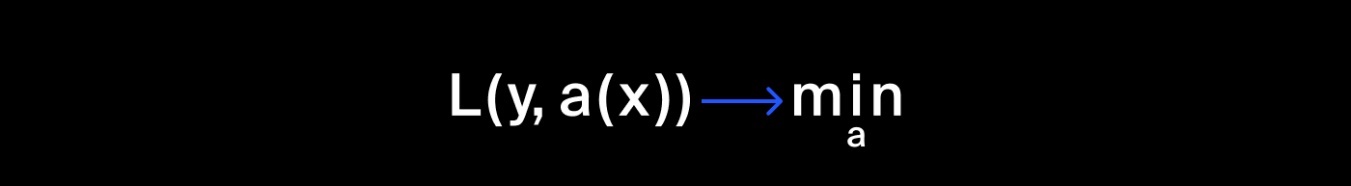
En la potenciación, cada alumno base intenta "empujar" las predicciones del paso anterior hacia las respuestas correctas. Así es como se minimiza la función de pérdida. La potenciación del gradiente hace que el proceso sea aún más eficiente.

Como antes, nuestro objetivo es minimizar la función de pérdida, ¡pero ahora las respuestas del modelo son el argumento a lo largo del cual se realizará el descenso! Y así obtenemos la potenciación del gradiente.

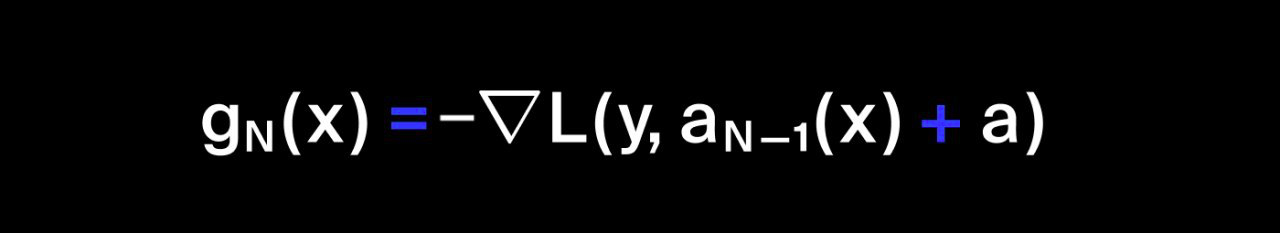
Por ejemplo, nuestra función de pérdida es *L(y, a)* y tiene una derivada. Recordemos la fórmula del ensamble:



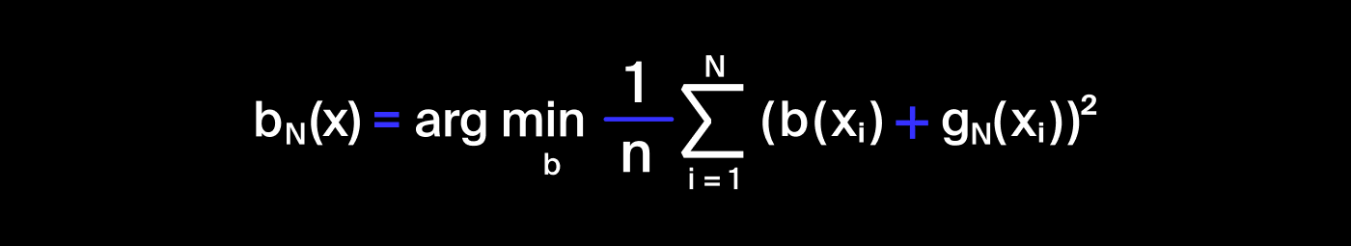
En cada paso, selecciona las respuestas que minimizarán la función:



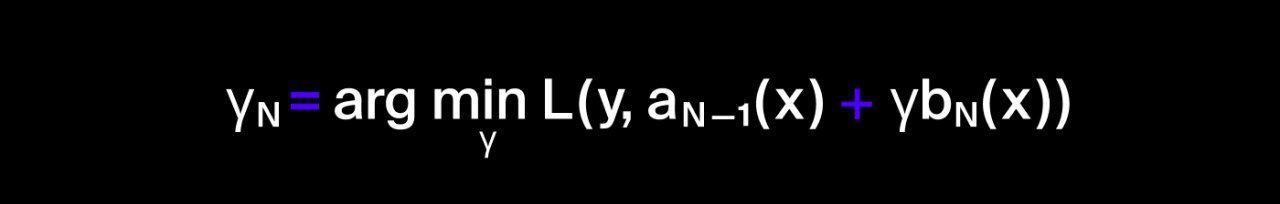
Minimiza la función con descenso de gradiente. Para ello, en cada paso, calcula el gradiente negativo de la función de pérdida para la predicción *gN*:



Para impulsar las predicciones hacia las respuestas correctas, el alumno base aprende a predecir *gN*:



Obtén el peso para *bN* de la tarea de minimización iterando diferentes valores:



Es el coeficiente para el alumno base lo que ayuda a ajustar el ensamble para hacer predicciones lo más precisas posible.

La potenciación de gradiente es adecuada para diferentes funciones de pérdida que tienen derivadas, por ejemplo, el cuadrado medio en una tarea de regresión o logarítmica en una tarea de clasificación binaria.

Pregunta

¿Qué es la potenciación de gradiente?

La actualización de los parámetros del alumno base hacia el gradiente negativo

La construcción secuencial del ensamble a través del cambio de predicciones

Descenso de gradientes funcionales iterativos

¡Correcto!

La actualización secuencial de los pesos del modelo lineal hacia el gradiente negativo

¡Bien hecho!

Pregunta

El objetivo para los alumnos base en la potenciación del gradiente es:

*ECM*

Residuos de predicción

Gradiente de función de pérdida

Gradiente negativo de la función de pérdida

¡Correcto!

¡Buen trabajo!

Capítulo 5/9 · Faltan 3 lecciones

Potenciación del gradiente

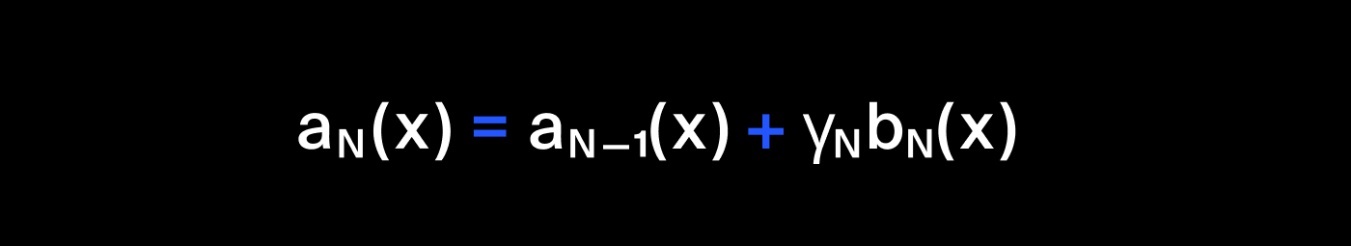
**Regularización de potenciación del gradiente**

**La regularización se puede utilizar para reducir el sobreajuste durante la potenciación del gradiente.**

Si se han reducido los pesos en una regresión lineal, entonces la regularización de potenciación del gradiente es:

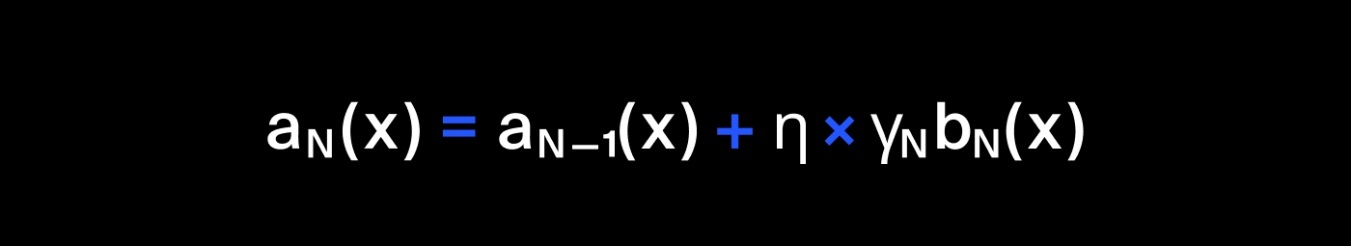
1. reducción del tamaño del paso;
2. ajuste de los parámetros del árbol;
3. aleatorización de submuestras para estudiantes base *bᵢ*.

Reduce el tamaño del paso. Revisa la fórmula para calcular predicciones en el paso *N*:



Cuando un algoritmo toma pasos que son demasiado grandes, este recuerda rápidamente el conjunto de entrenamiento. Esto da como resultado un sobreajuste del modelo.

Introduce el coeficiente *η*, que controla la tasa de aprendizaje y puede usarse para reducir el tamaño del paso:



El valor de este coeficiente se elige iterando diferentes valores en el rango de 0 a 1. Un valor más pequeño significa un paso más pequeño hacia el gradiente negativo y una mayor exactitud del ensamble. Pero si la tasa de aprendizaje es demasiado baja, el proceso de entrenamiento llevará demasiado tiempo.

La segunda forma de regularizar la potenciación del gradiente es ajustar los parámetros del árbol. Podemos limitar la profundidad del árbol o la cantidad de elementos en cada nodo, probar diferentes valores y ver cómo afecta el resultado. Por bresiduos y el modelo no estará demasiado sobreajustado. El tercer método de regularización es trabajar con submuestras. El algoritmo funciona con submuestras en lugar del conjunto completo. Esta versión del algoritmo es similar al DGE y se llama potenciación del gradiente estocástico.

Pregunta

Elige el enunciado correcto:

Cuanto mayor sea la tasa de aprendizaje, menor será la probabilidad de sobreajuste.

Cuanto mayor sea la tasa de aprendizaje, mayor será la probabilidad de sobreajuste.

¡Correcto!

Con una alta tasa de aprendizaje, el algoritmo tardará menos en ajustarse a la muestra y encontrar el modelo correcto.

¡Tu comprensión del material es impresionante!

Pregunta

¿Qué tiene de aleatoria la potenciación del gradiente estocástico?

Los pesos para los alumnos base se seleccionan al azar.

Se seleccionan submuestras aleatorias de características para cada modelo.

Los modelos están entrenados en submuestras aleatorias.

¡Correcto!

¡Lo has entendido bien!

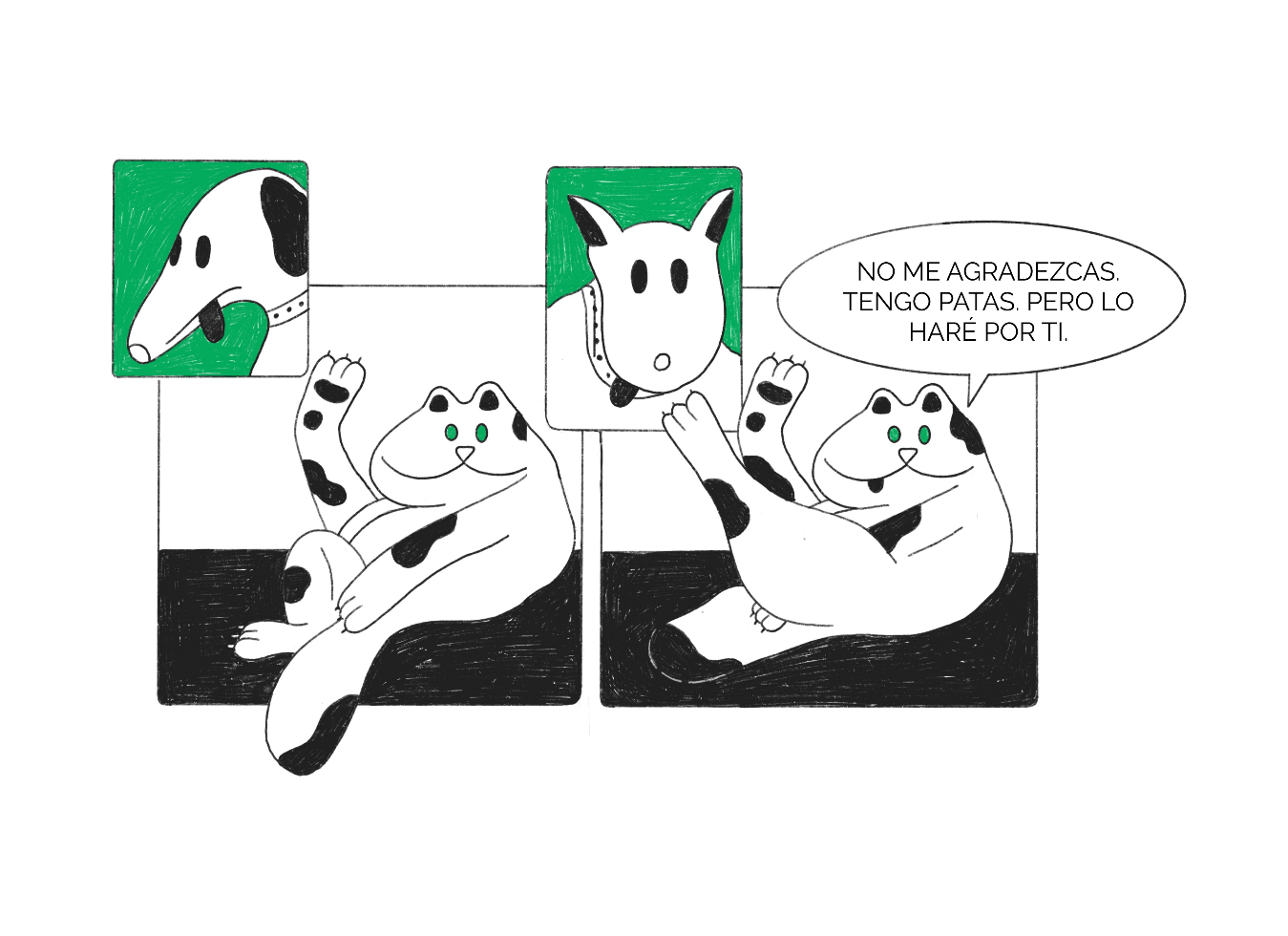
**Librerías para potenciación del gradiente**

**Veamos estas librerías para potencciación del gradiente: XGBoost, LightGBM y CatBoost**

Veamos cada una:

1. XGBoost (potenciación del gradiente extrema) es una librería popular de potenciación del gradiente en Kaggle. Código abierto. Lanzada en 2014.
2. LightGBM (máquina ligera de potenciación del gradiente). Desarrollada por Microsoft. Entrenamiento de potenciación del gradiente rápido y preciso. Funciona directamente con características categóricas. Lanzada en 2017. Comparación con XGBoost: <https://lightgbm.readthedocs.io/en/latest/Experiments.html> *(materiales en inglés)*.
3. CatBoost (potenciación categórica). Desarrollada por Yandex. Superior a otros algoritmos en términos de métricas de evaluación. Aplica varias técnicas de codificación para características categóricas (LabelEncoding, One-Hot Encoding). Lanzada en 2017. Comparación con XGBoost y LightGBM: <https://catboost.ai/#benchmark> *(materiales en inglés)*.

Por ejemplo, así es como CatBoost funciona con características categóricas: un perro salchicha sigue siendo un perro salchicha y un stafford sigue siendo un stafford. No es necesario codificar usando 1 o 0.



Comparemos las librerías en función de dos características:

| **Librería** | **Tratamiento de características categóricas** | **Velocidad** |
| --- | --- | --- |
| XGBoost | No | Baja |
| LightGBM | Sí | Alta |
| CatBoost | Sí | Alta |

Echemos un vistazo a la librería CatBoost. También puedes leer sobre LightGBM cuando tengas oportunidad.

Toma el conjunto de datos de la tarea de seguros (del curso de aprendizaje supervisado).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

data = pd.read\_csv('travel\_insurance\_us.csv')

print(data.head())

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

data.drop('Claim', axis=1), data.Claim, test\_size=0.25, random\_state=12345

)

Agency Agency Type Distribution Channel Product Name Claim \\

0 JZI Airlines Online Value Plan 0

1 EPX Travel Agency Online Cancellation Plan 0

2 EPX Travel Agency Online Cancellation Plan 0

3 EPX Travel Agency Online Cancellation Plan 0

4 JZI Airlines Online Value Plan 0

Duration Destination Net Sales Commission (in value) \\

0 12 TAIWAN, PROVINCE OF CHINA 45.0 15.75

1 50 JAPAN 22.0 0.00

2 251 BRAZIL 80.0 0.00

3 6 INDIA -10.0 0.00

4 5 CHINA 45.0 15.75

Gender Age

0 M 39

1 None 36

2 None 36

3 None 36

4 M 34

Las características categóricas tienen valores perdidos, anotados como valores nuevos. Por ejemplo, None en la columna Género. Este no es un problema para CatBoost: dichos valores son categorías separadas para la librería.

Importa CatBoostClassifier de la librería y crea un modelo. Ya que tenemos un problema de clasificación, especifica la función de pérdida logística. Haz 10 iteraciones para que no tengamos que esperar demasiado.

from catboost import CatBoostClassifier

model = CatBoostClassifier(loss\_function="Logloss", iterations=10)

Entrena el modelo con el método fit(). Además del objetivo y las características, pasa las características categóricas al modelo:

cat\_features = [

'Agency',

'Agency Type',

'Distribution Channel',

'Product Name',

'Destination',

'Gender',

]

model.fit(features\_train, target\_train, cat\_features=cat\_features)

Learning rate set to 0.5

0: learn: 0.3376311 total: 19.2ms remaining: 172ms

1: learn: 0.2057412 total: 36ms remaining: 144ms

2: learn: 0.1413080 total: 49.4ms remaining: 115ms

3: learn: 0.1075173 total: 64.1ms remaining: 96.2ms

4: learn: 0.0894613 total: 81ms remaining: 81ms

5: learn: 0.0795985 total: 93.7ms remaining: 62.5ms

6: learn: 0.0734879 total: 103ms remaining: 44.1ms

7: learn: 0.0699551 total: 119ms remaining: 29.8ms

8: learn: 0.0690939 total: 126ms remaining: 14ms

9: learn: 0.0680472 total: 142ms remaining: 0us

Cuando tienes muchas iteraciones y no quieres generar información para cada una, usa el argumento **verbose**:

model = CatBoostClassifier(loss\_function="Logloss", iterations=50)

model.fit(features\_train, target\_train, cat\_features=cat\_features, verbose=10)

Learning rate set to 0.5

0: learn: 0.3376311 total: 16.7ms remaining: 819ms

10: learn: 0.0671834 total: 155ms remaining: 549ms

20: learn: 0.0654290 total: 322ms remaining: 445ms

30: learn: 0.0644132 total: 491ms remaining: 301ms

40: learn: 0.0636832 total: 654ms remaining: 144ms

49: learn: 0.0627461 total: 787ms remaining: 0us

Calcula la predicción con predict():

pred\_valid = model.predict(features\_valid)

Ya conoces algunos parámetros del modelo que pueden afectar significativamente el resultado del entrenamiento. Dichos parámetros incluyen la profundidad máxima del árbol depth, la tasa de aprendizaje learning\_rate y el número de árboles en las iterations del ensamble.

Selecciona los hiperparámetros del modelo para que el valor AUC-ROC no sea inferior a 0.822 (el mejor resultado para esta métrica del curso de aprendizaje supervisado). Trata de obtener un resultado más alto. Utiliza la semilla (o "seed") aleatoria 12345.

Muestra en la pantalla el valor de la métrica AUC-ROC (en precódigo).

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from catboost import CatBoostClassifier

from sklearn.metrics import roc\_auc\_score

data = pd.read\_csv('/datasets/travel\_insurance\_us.csv')

features\_train, features\_valid, target\_train, target\_valid = train\_test\_split(

data.drop('Claim', axis=1), data.Claim, test\_size=0.25, random\_state=12345

)

cat\_features = [

'Agency',

'Agency Type',

'Distribution Channel',

'Product Name',

'Destination',

'Gender',

]

model = CatBoostClassifier(loss\_function="Logloss",

iterations=150, # Aumentar para mejorar el rendimiento

random\_seed=12345)# < escribe tu código aquí >

model.fit(features\_train, target\_train, cat\_features=cat\_features, verbose=10)

probabilities\_valid = model.predict\_proba(features\_valid)

probabilities\_one\_valid = probabilities\_valid[:, 1]

print(roc\_auc\_score(target\_valid, probabilities\_one\_valid))

Resultado

Learning rate set to 0.277333

0: learn: 0.4688641 total: 39.9ms remaining: 5.94s

10: learn: 0.0803674 total: 339ms remaining: 4.29s

20: learn: 0.0676325 total: 622ms remaining: 3.82s

30: learn: 0.0661601 total: 940ms remaining: 3.61s

40: learn: 0.0655294 total: 1.3s remaining: 3.45s

50: learn: 0.0652151 total: 1.65s remaining: 3.21s

60: learn: 0.0650206 total: 2.01s remaining: 2.94s

70: learn: 0.0645903 total: 2.37s remaining: 2.64s

80: learn: 0.0643558 total: 2.73s remaining: 2.33s

90: learn: 0.0640066 total: 3.09s remaining: 2s

100: learn: 0.0636229 total: 3.45s remaining: 1.67s

110: learn: 0.0634252 total: 3.81s remaining: 1.34s

120: learn: 0.0629009 total: 4.17s remaining: 999ms

130: learn: 0.0623186 total: 4.53s remaining: 657ms

140: learn: 0.0620025 total: 4.89s remaining: 312ms

149: learn: 0.0618044 total: 5.21s remaining: 0us

0.8247370519512989

¡Es correcto!

Mejorar una métrica para CatBoost es muy fácil. De hecho, hasta un gato podría hacerlo. Si seguimos ajustando los hiperparámetros, podemos lograr un AUC-ROC perfecto.

Capítulo 5/9 · Última lección

Potenciación del gradiente

**Conclusión**

En este capítulo aprendiste a:

* Minimizar la función de pérdida utilizando la potenciación del gradiente;
* Reducir el sobreajuste de potenciación del gradiente;
* Trabajar con la librería CatBoost.

**Llévate esto contigo**

Descárgate el resumen del capítulo y la hoja informativa para poder consultarlos cuando los necesites.

* [Resumen del capítulo: Potenciación del gradiente](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/Resumen_del_captulo_Potenciacin_del_gradiente.pdf)
* [Hoja informativa: Potenciación del gradiente](https://practicum-content.s3.us-west-1.amazonaws.com/new-markets/DS_Sprint_12/moved_Hoja_informativa_Potenciacin_del_gradiente_DS_Sprint12_ESP.pdf)

El siguiente capítulo trata sobre habilidades blandas (soft skills).

Capítulo 6/9

Habilidades socioemocionales

**Introducción**

¿Quieres trabajar como junior toda tu vida? Seguramente la respuesta es no. Es hora de hablar sobre cómo los Junior Data Scientists ascienden hasta convertirse en profesionales de nivel medio.

Nunca empezarás a hacer cálculos como lo hace un PC. Las ventajas de un Data Scientist están en sus habilidades de pensamiento crítico y sistémico. El trabajo de un detective no es contar las líneas de una huella dactilar, sino encontrar al culpable.

Cuando llegue el momento, lo que te hará ascender en la escala profesional será tanto la capacidad de profundizar en el contexto como tus habilidades de comunicación.

En este capítulo hablaremos sobre varios algoritmos que vale la pena convertir en hábitos.

Capítulo 6/9

Habilidades socioemocionales

**Contexto de la tarea**

Acostúmbrate a profundizar inmediatamente en el contexto de la tarea.

Pregunta

Trabajas como Data Scientist para la tienda en línea "Hamazon". El departamento de marketing te asigna la tarea de desarrollar un sistema de recomendaciones de productos. ¿De dónde crees que surgió esta tarea?

Todas las tiendas necesitan recomendaciones de productos.

Los empleados del departamento de marketing no tienen nada qué hacer.

Los especialistas en marketing decidieron que hay pocos visitantes debido a la dificultad de buscar un producto.

Los especialistas en marketing están haciendo un experimento.

Así es. Decidieron que el machine learning ayudaría a promocionar la tienda en línea. "La publicidad en redes sociales funciona mejor. Queremos eso”, agregaron los especialistas en marketing.

¡Lo has entendido bien!

Los siguientes puntos te ayudarán a establecer la dirección del desarrollo del modelo:

* Debes seleccionar solo algunos productos para que quepan en una pequeña sección de publicidad;
* No tiene sentido recomendar productos que puedes encontrar en otra tienda porque no atraerán la atención de los visitantes;
* No dispones de datos sobre pedidos anteriores y vistas de productos, pero sí tienes información sobre los perfiles de los visitantes en la red social seleccionada.

El algoritmo para entender el contexto es simple:

1. Responde a la pregunta: ¿de dónde vino la tarea?
2. Determina con el cliente dónde buscar respuestas e hipótesis.

Luego, debes abordar la solicitud de forma correcta. Es decir, averiguar varios detalles sobre la tarea. Ahora hablemos de cómo hacer esto.

Capítulo 6/9 · Faltan 3 lecciones

Habilidades socioemocionales

**Trabajar con solicitudes**

La tarea no siempre es clara y exhaustiva. Para completarla por tu cuenta, necesitas recopilar más información. Acostúmbrate al hecho de que esta es tu responsabilidad.

Te ahorrarás tiempo y esfuerzo si sigues estos 6 pasos:

1. Asegúrate de saber quién es tu cliente final.

Puede ser cualquier persona, incluida la alta dirección. Es poco probable que el director ejecutivo se acerque a ti en persona, pero puede establecer una tarea a través del líder del equipo. Sabiendo quién es el cliente final, puedes comprender exactamente qué escribir en el informe.

Pregunta

Elige la pregunta que te ayudará a entender quién es tu cliente final:

¿Por qué necesitamos completar esta tarea?

¿Cual es tu posicion?

¿Quién necesita los resultados?

¿A quién debemos contactar para completar esta tarea?

¿Está esto por encima de mi salario?

¡Tu comprensión del material es impresionante!

1. Averigua cuál es el objetivo final de tu trabajo.

Te pidieron que vieras qué tan efectivos son los diferentes canales de comunicación. Los objetivos de "encontrar un canal para una mayor inversión" y "decidir por cuál canal optar" determinan cuáles serán las recomendaciones finales.

Pregunta

¿Qué pregunta debes hacerte para comprender el objetivo final de tu trabajo?

¿Por qué trabajo aquí?

¿Cuál es el objetivo principal de un Data Scientist?

¿Cuál es el propósito de completar esta tarea?

¿Qué haremos con los resultados de esta tarea?

¿Recibiré una bonificación por completar esta tarea?

¿No me regañarán por el resultado?

¡Lo has entendido bien!

1. Especifica todos los detalles que el cliente de la tarea desea conocer. ¿Hay inconvenientes ocultos?

El líder de tu equipo te pidió que desarrollaras un modelo que predijera el tiempo de entrega según el clima y el tráfico. Y durante la conversación, resultó que el modelo además debe funcionar en teléfonos inteligentes y ser rápido.

Pregunta

Selecciona todas las preguntas que te ayudarán a perfeccionar tu tarea:

Elige tantas como quieras

¿Cuál es la contraseña del Wi-Fi aquí?

¿Qué restricciones de recursos existen para desarrollar el modelo?

¿En qué variable debo guardar el modelo?

¿Dónde puedo probar el primer prototipo del modelo?

¿Cuándo es la fecha límite?

¡Perfecto!

1. Repasa todos los detalles y escribe cuál debería ser el resultado perfecto.

Según la empresa, eso podría ser escribir un ticket, almacenarlo como un correo electrónico o agregarlo a la aplicación Task Manager. Incluso si el cliente afirma que un acuerdo oral es suficiente, te recomendamos anotar los parámetros de entrada para que tengas algo que consultar.

1. Esbozar un prototipo y consultar con el cliente para ver si este funciona para ellos.

Puede haber varias iteraciones de "prototipo —> retroalimentación —> correcciones" antes de acordar un formulario conveniente.

1. Proporcionar al cliente el resultado final.

Si hubo una comunicación escrita, termínala: cierra los tickets o finaliza la tarea en la aplicación Task Manager.

Mientras completas estos pasos, puede resultar que la tarea no esté relacionada con tu trabajo, o incluso con Data Science. Por ejemplo, es posible que ya haya datos preparados disponibles y solo necesites solicitarlos o tal vez otro departamento se esté ocupando de ellos. Así, les ahorrarás mucho tiempo a tus colegas y también a ti.

Capítulo 6/9 · Faltan 2 lecciones

Habilidades socioemocionales

**Reajuste del resultado**

El trabajo está completo. Pero de repente el cliente dice que el resultado de tu trabajo no cumple con sus expectativas. O de repente quiere cambiar los parámetros de entrada, por ejemplo, las métricas de evaluación. Su argumento es sólido como una roca, dice que debe ser fácil de arreglar (¡aunque sabes que no es nada fácil!).

¿Cómo deberías responder? Hay dos lados del algoritmo: emociones y sustancia.

1. Respecto a las emociones:

Obviamente no estás feliz, especialmente si tienes que hacer grandes mejoras. Sin embargo:

* + No tiene sentido discutir. Es importante que le expliques a tu colega que los dos están desperdiciando tiempo y recursos;
  + Explica de forma amigable que los ajustes son buenos durante el proceso, no en la etapa de resultados.

1. En cuanto a la sustancia:
   * Piensa si abordaste la solicitud lo suficiente. ¿Qué más se podría haber pedido para cumplir con los requisitos técnicos de alta calidad?
   * Incluye siempre la escalabilidad en tu código para que puedas hacer correcciones y ajustes rápidamente.

No tienes escapatoria de algunos ajustes en tu trabajo, pero tienes el poder para hacerlos lo más constructivos posible.

Capítulo 6/9 · Última lección

Habilidades socioemocionales

**Conclusión**

La habilidad de un Data Scientist comienza con la capacidad de comprender el contexto de una tarea y abordar la solicitud de forma adecuada.

Te recomendamos que imprimas estos algoritmos, los tengas a la mano y ejecutes cualquier tarea a través de ellos. Con el tiempo, esto se convertirá en un hábito y funcionará a tu favor en términos de crecimiento profesional.

¡Felicidades! Has completado otro curso de la plataforma de entrenamiento. Ahora es el momento perfecto para probar tus habilidades y resolver un nuevo problema de machine learning. Al realizar este proyecto, trabajarás por tu cuenta.

Cuando termines, envía tu trabajo al revisor del proyecto. Recibirás feedback dentro de las siguientes 48 horas. Después de esto, harás los cambios necesarios a tu trabajo y lo enviarás para una segunda revisión.

Por lo general, este proceso se repetirá varias veces hasta que recibas el visto bueno del revisor y se aprueben todas las correcciones. Todo eso es parte del trabajo.

Tu proyecto se considerará completado una vez que el revisor del proyecto lo apruebe.

**Descripción del proyecto**

Rusty Bargain es un servicio de venta de coches de segunda mano que está desarrollando una app para atraer a nuevos clientes. Gracias a esa app, puedes averiguar rápidamente el valor de mercado de tu coche. Tienes acceso al historial, especificaciones técnicas, versiones de equipamiento y precios. Tienes que crear un modelo que determine el valor de mercado.

A Rusty Bargain le interesa:

* la calidad de la predicción
* la velocidad de la predicción
* el tiempo requerido para el entrenamiento

**Instrucciones del proyecto**

1. Descarga y examina los datos.
2. Entrena diferentes modelos con varios hiperparámetros (debes hacer al menos dos modelos diferentes, pero más es mejor. Recuerda, varias implementaciones de potenciación del gradiente no cuentan como modelos diferentes). El punto principal de este paso es comparar métodos de potenciación del gradiente con bosque aleatorio, árbol de decisión y regresión lineal.
3. Analiza la velocidad y la calidad de los modelos.

Observaciones:

* Utiliza la métrica RECM para evaluar los modelos.
* La regresión lineal no es muy buena para el ajuste de hiperparámetros, pero es perfecta para hacer una prueba de cordura de otros métodos. Si la potenciación del gradiente funciona peor que la regresión lineal, definitivamente algo salió mal.
* Aprende por tu propia cuenta sobre la librería LightGBM y sus herramientas para crear modelos de potenciación del gradiente (gradient boosting).
* Idealmente, tu proyecto debe tener regresión lineal para una prueba de cordura, un algoritmo basado en árbol con ajuste de hiperparámetros (preferiblemente, bosque aleatorio), LightGBM con ajuste de hiperparámetros (prueba un par de conjuntos), y CatBoost y XGBoost con ajuste de hiperparámetros (opcional).
* Toma nota de la codificación de características categóricas para algoritmos simples. LightGBM y CatBoost tienen su implementación, pero XGBoost requiere OHE.
* Puedes usar un comando especial para encontrar el tiempo de ejecución del código de celda en Jupyter Notebook. Encuentra ese comando.
* Dado que el entrenamiento de un modelo de potenciación del gradiente puede llevar mucho tiempo, cambia solo algunos parámetros del modelo.
* Si Jupyter Notebook deja de funcionar, elimina las variables excesivas por medio del operador del:
* del features\_train

**Descripción de los datos**

El dataset está almacenado en el archivo /datasets/car\_data.csv. [descargar dataset](https://practicum-content.s3.us-west-1.amazonaws.com/datasets/car_data.csv).

Características

* *DateCrawled* — fecha en la que se descargó el perfil de la base de datos
* *VehicleType* — tipo de carrocería del vehículo
* *RegistrationYear* — año de matriculación del vehículo
* *Gearbox* — tipo de caja de cambios
* *Power* — potencia (CV)
* *Model* — modelo del vehículo
* Mileage — kilometraje (medido en km de acuerdo con las especificidades regionales del conjunto de datos)
* *RegistrationMonth* — mes de matriculación del vehículo
* *FuelType* — tipo de combustible
* *Brand* — marca del vehículo
* *NotRepaired* — vehículo con o sin reparación
* *DateCreated* — fecha de creación del perfil
* *NumberOfPictures* — número de fotos del vehículo
* *PostalCode* — código postal del propietario del perfil (usuario)
* *LastSeen* — fecha de la última vez que el usuario estuvo activo

Objetivo

*Price* — precio (en euros)

**Evaluación del proyecto**

Hemos definido los criterios de evaluación para el proyecto. Léelos con atención antes de pasar al ejercicio.

Esto es en lo que se fijarán los revisores al examinar tu proyecto:

* ¿Seguiste todos los pasos de las instrucciones?
* ¿Cómo preparaste los datos?
* ¿Qué modelos e hiperparámetros consideraste?
* ¿Conseguiste evitar la duplicación del código?
* ¿Cuáles son tus hallazgos?
* ¿Mantuviste la estructura del proyecto?
* ¿Mantuviste el código ordenado?

Ya tienes tus hojas informativas y los resúmenes de los capítulos, por lo que todo está listo para continuar con el proyecto.

¡Buena suerte!

¡Vamos allá!

O haz el proyecto en tu ordenador y súbelo cuando hayas terminado.

**Conclusión**

Has completado el curso de métodos numéricos.

Has aprendido a:

* Calcular la complejidad computacional de los algoritmos;
* Entrenar modelos utilizando el algoritmo de descenso de gradiente;
* Utilizar la técnica de potenciación del gradiente.

El siguiente curso es sobre la creación de características. Aprenderás a resolver tareas de clasificación y regresión de textos y series temporales.