

5. Aprendizaje Automático (Machine Learning)

Copyright (c) 2025 Adrián Quiroga Linares Lectura y referencia permitidas; reutilización y plagio prohibidos

5.1 Introducción y Cambio de Paradigma

Hasta ahora (Sistemas Expertos), programábamos la IA diciéndole explícitamente **qué hacer** (reglas **SI-ENTONCES**). En el Aprendizaje Automático, cambiamos el enfoque: **la máquina aprende las reglas a partir de los datos.**

Un programa aprende si mejora su rendimiento (**P**) en una tarea (**T**) basándose en la experiencia (**E**).

- **Ejemplo (Damas):**
 - **T (Tarea):** Jugar a las damas.
 - **E (Experiencia):** Jugar miles de partidas contra sí mismo.
 - **P (Rendimiento):** % de partidas ganadas.

5.2 Tipos de Aprendizaje

No todos los problemas son iguales. Según la información que tengamos, clasificamos el aprendizaje en:

5.2.1 Aprendizaje Supervisado

Tenemos los "exámenes resueltos". Entregamos a la máquina datos de entrada (x) junto con su respuesta correcta o etiqueta (y). El objetivo es aprender la relación para predecir y en datos nuevos.

Se divide en dos subtipos clave:

1. **Regresión:** Predice un **valor numérico continuo** (infinitas posibilidades).
 - *Ejemplo:* Predecir el precio de una casa según sus metros cuadrados.
2. **Clasificación:** Predice una **clase o etiqueta discreta** (categorías fijas).
 - *Ejemplo:* ¿Es este tumor maligno o benigno? (0 o 1).
 - *Nota:* Esto es lo que hacía el Perceptrón (clasificar).

5.2.2 Aprendizaje No Supervisado

Solo tenemos los datos (x), sin respuestas. La máquina debe encontrar **estructuras o patrones** por sí sola.

- **Ejemplo (Clustering):** Agrupar noticias similares en Google News o segmentar clientes por comportamiento .

5.3 Regresión Lineal: El Modelo Base

Es el "Hola Mundo" del Machine Learning. Busca dibujar la **Línea recta** que mejor se ajusta a una nube de puntos.

5.3.1 La Hipótesis (h_θ)

Es la fórmula matemática que usamos para predecir. En regresión lineal simple (una variable), es la ecuación de una recta:

$$h_\theta(x) = \theta_0 + \theta_1 x$$

- θ_0 : El punto de corte (Bias).
- θ_1 : La pendiente (Peso).

Conexión Clave: ¿Te suena? Es idéntico a la parte interna de una neurona ($z = w \cdot x + b$).

- Por esto, en el examen, un **Regresor Lineal** es lo mismo que una **Neurona Artificial con función de activación Lineal (Identidad)**.
- Si a esa suma no le pones función de activación (o la función es $f(x) = x$), la neurona se comporta exactamente como este modelo de regresión.

5.3.2 La Función de Coste ($J(\theta)$)

Para aprender, necesitamos medir cuánto nos equivocamos. Usamos el Error Cuadrático Medio (MSE):

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum (h_\theta(x^{(i)}) - y^{(i)})^2$$

- Básicamente: "Calcula la diferencia entre lo que predijiste y el valor real, elévalo al cuadrado (para que sea siempre positivo) y haz la media".
- **Objetivo:** Encontrar los θ (pesos) que hagan que esta J sea **mínima** (cero error idealmente).
- La función de coste de un **regresor lineal** mide lo bien o mal que opera el regresor lineal en un conjunto de datos. Es una función convexa.
- Esto es una función de coste convexa.

5.3.3 Generalización: Regresión Lineal Multivariable

El modelo básico ($h_\theta(x) = \theta_0 + \theta_1 x$) solo sirve si hay una única característica de entrada. Pero el mundo es complejo.

- **Concepto:** si tenemos n características (x_1, x_2, \dots, x_n), la fórmula se expande

- **Nueva Hipótesis:**

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

- **Vectorización:** Para que el ordenador lo calcule rápido, esto se expresa como una multiplicación de matrices: $h_{\theta}(x) = \theta^T x$.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}, x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{metros} \\ \text{habitaciones} \end{bmatrix}$$

$$\text{Si } \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}, \text{ entonces } \theta^T = [\theta_0, \theta_1, \theta_2].$$

$$\theta^T x = [\theta_0, \theta_1, \theta_2] \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$$

5.3.4 Adaptación a Curvas: Regresión Polinómica

A veces, los datos no siguen una línea recta, sino una curva.

- **El Truco:** no necesitamos inventar un algoritmo nuevo. Podemos "engaños" a la regresión lineal creando **nuevas características artificiales** elevando las originales al cuadrado, al cubo, etc.
- **Hipótesis:**

$$\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2$$

- **Peligro:** Al añadir términos polinómicos ($x^2, x^3 \dots$), la línea se vuelve muy flexible y curva. Esto mejora el ajuste, pero si nos pasamos, causaremos **Overfitting** (esto se explica más adelante).

Imagina que quieras predecir cuánto vas a gastar en electricidad (y) basándote en la temperatura de la calle (x).

- **Intento con Regresión Lineal (Línea Recta):**

- La máquina pensaría: "*Cuanto más calor hace, más gastas*" (Línea subiendo)
O BIEN "*Cuanto más calor hace, menos gastas*" (Línea bajando).
- **¿Por qué falla?** Porque la realidad es más compleja:
 - Si hace mucho frío (0°C), gastas mucho (calefacción).
 - Si hace una temperatura media (20°C), gastas poco (ni frío ni calor).
 - Si hace mucho calor (40°C), vuelves a gastar mucho (aire acondicionado).
- **Forma de los datos:** Tienen forma de "**U**" (una parábola).

- **Solución (Regresión Polinómica):**

- Al añadir el término al cuadrado (x^2), la fórmula puede curvarse y crear esa "U".
- Conclusión:** El modelo aprende que el gasto sube en *ambos extremos* de la temperatura.

Nota

En un regresor polinomial, cuanto mayor es el grado del polinomio, mejor podemos ajustar el modelo a su conjunto de entrenamiento. No implica que vaya a funcionar mejor el modelo, seguramente haya alto overfitting.

5.4 Regresión Logística (Para Clasificación)

5.4.1 La hipótesis

(*Importante: Aunque se llame "regresión", sirve para clasificar*). Es el modelo matemático que conecta la Regresión Lineal con las Neuronas Artificiales.

- El Problema:** La regresión lineal predice valores continuos (puede dar 500, -20, 1000). Pero si queremos clasificar (0 o 1, Tumor benigno o maligno), necesitamos que la salida esté acotada entre 0 y 1.
- La Solución (Función Sísmoide):** Envolvemos la hipótesis lineal dentro de una función logística o sísmoide ($g(z)$).

$$h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

- Interpretación:** La salida se convierte en una **probabilidad**.
 - Si $h_{\theta}(x) = 0.7$, significa "Hay un 70% de probabilidad de que sea clase 1".
- Frontera de Decisión:** Es la línea (o curva) que separa la zona donde predecimos 0 de la zona donde predecimos 1. Se produce cuando la probabilidad es exactamente 0.5.

5.4.2 Función de Coste

Se define en dos partes:

- Si la respuesta real es 1 ($y = 1$):

$$\text{Coste} = -\log(h_{\theta}(x))$$

- Interpretación:** Si la máquina predice 1 (acierta), el coste es 0. Si predice 0 (falla estrepitosamente), el coste tiende a infinito (castigo enorme).

- Si la respuesta real es 0 ($y = 0$):

$$\text{Coste} = -\log(1 - h_{\theta}(x))$$

- **Interpretación:** Si predice 0 (acierta), coste 0. Si predice 1 (falla), coste infinito.

Para no escribir dos fórmulas, las combinamos en una sola ecuación elegante:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))]$$

- **Nota:** Solo uno de los dos términos se activa a la vez (porque y es 0 o 1, anulando la otra parte).

5.4.3 Clasificación Binaria vs. Multiclasificación

La Regresión Logística está diseñada por naturaleza para responder "Sí/No" (Binaria). ¿Pero qué pasa si quiero clasificar correos en "Trabajo", "Amigos" y "Spam" (3 clases)?

A. Clasificación Binaria (2 Clases)

- **Datos:** $y \in \{0, 1\}$.
- **Frontera de Decisión:** Una sola línea (o curva) que separa el espacio en dos zonas.
- **Modelo:** Una sola hipótesis $h_\theta(x)$.

B. Clasificación Multiclasificación (Más de 2 grupos)

- **Datos:** $y \in \{0, 1, 2, \dots, n\}$.
- **Estrategia:** Usamos la técnica "**Uno contra Todos**" (**One-vs-All**).

En lugar de entrenar un solo clasificador complejo, entrenamos varios clasificadores binarios:

1. Transformamos el problema:

- Imagina que tienes 3 clases: Triángulos, Cuadrados y Círculos.
- Entrenas el **Clasificador 1**: ¿Es un Triángulo o NO? (Juntas cuadrados y círculos en el grupo "NO").
- Entrenas el **Clasificador 2**: ¿Es un Cuadrado o NO?
- Entrenas el **Clasificador 3**: ¿Es un Círculo o NO?

2. Predicción:

- Cuando llega un dato nuevo, lo pasas por los 3 clasificadores.
- Cada uno te dará una probabilidad (ej. C1: 20%, C2: 80%, C3: 10%).
- **Eliges la clase con la probabilidad más alta** ($\max h_\theta^{(i)}(x)$).

5.4.4 Ejemplo

Quieres que la IA decida si un email entrante es "Spam" o "Correo Normal".

- **Intento con Regresión Lineal:**

- Le das al sistema el número de veces que aparece la palabra "Oferta".
- La regresión lineal te podría devolver un valor como 150 o -20.
- **¿Por qué falla?** ¿Qué significa que un correo sea "150 de Spam"? No tiene sentido. Tú necesitas una probabilidad entre 0 y 1.

- **Solución (Regresión Logística):**

- La función **Sigmoide** "aplasta" cualquier número que salga para que esté siempre entre **0 y 1**.
- **Resultado:** Te dice "*La probabilidad de que esto sea Spam es 0.95*".
- **Decisión:** Como $0.95 > 0.5$ (umbral), lo manda a la carpeta de Spam.

5.5 El Motor de Aprendizaje: Descenso del Gradiente

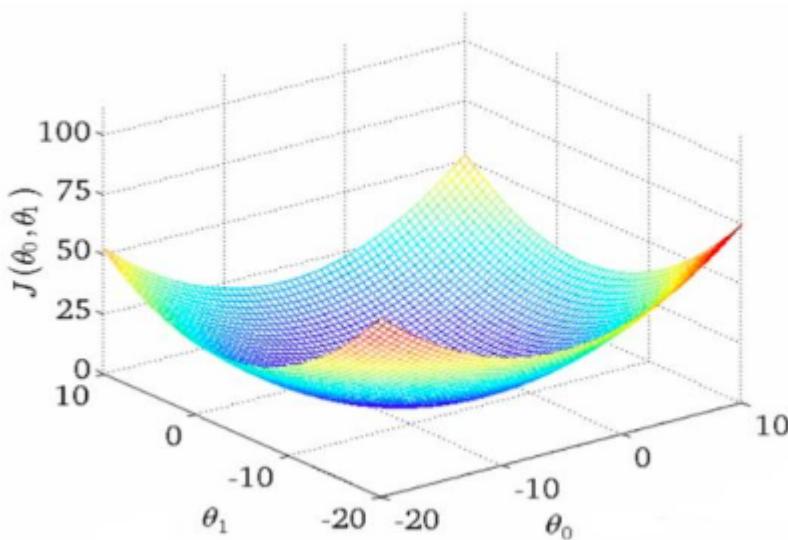
¿Cómo encuentra la máquina los mejores θ sin probar a lo loco? Usando un algoritmo de optimización iterativo, similar a "bajar una montaña a ciegas tanteando el terreno".

Algoritmo: Repetir hasta converger:

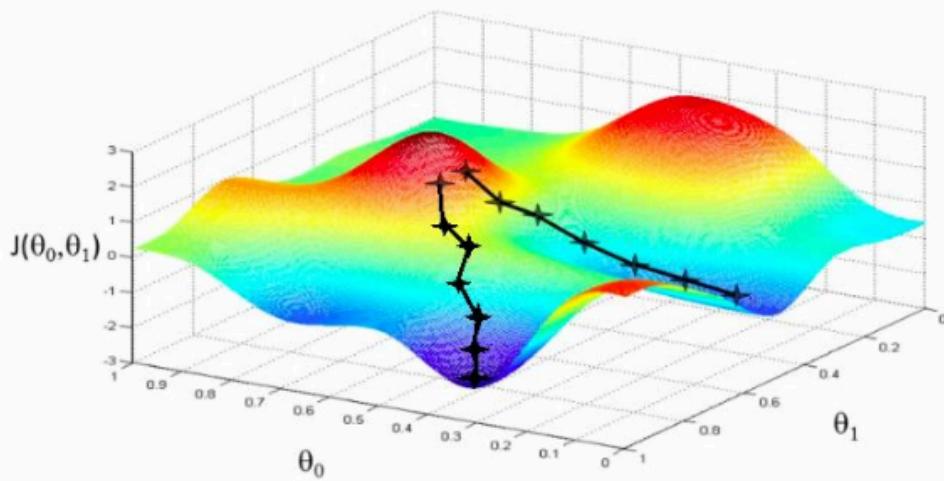
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

- **Derivada ($\frac{\partial}{\partial \theta}$):** Nos dice la pendiente. Si es positiva, debemos bajar (restar); si es negativa, subir. Nos da la **dirección** correcta.
- **Tasa de Aprendizaje (α):** Es el **tamaño del paso**.
 - Si α es muy pequeño: El aprendizaje es lentísimo (pasitos de bebé).
 - Si α es muy grande: Podemos "saltarnos" el mínimo y nunca converger (pasos de gigante).
- **Conexión:** Este es el mismo principio matemático que usaba el Perceptrón para actualizar sus pesos ($w_{nuevo} = w_{viejo} + \Delta w$).

En modelos simples, como el perceptrón, el regresor lineal (también polinómico y multivariable) y el regresor logístico la función de coste es convexa y converge, como en la imagen.



En el descenso por gradiente en modelos no lineales, como las redes neuronales, no asegura alcanzar el mínimo global ya que puede quedarse en un mínimo local. Por lo tanto la inicialización de los parámetros (los pesos) es importante ya que determina el punto inicial del descenso por gradiente. Tal y como se ve en esta imagen.



Ecuación Normal (Alternativa al descenso gradiente)

Es una forma alternativa de encontrar el valor óptimo de θ . La diferencia fundamental está en **cómo** llegan a la solución:

- **Descenso del Gradiente (Iterativo):** Empiezas con valores aleatorios y vas ajustando poco a poco (iteraciones) hasta minimizar el error. Necesitas elegir un "learning rate" (α).
- **Ecuación Normal (Analítica):** Usas álgebra lineal para resolver el problema de golpe. Es como tener una fórmula mágica que te "teletransporta" directamente al

punto más bajo de la función de coste, sin dar pasos. Encuentra la solución exacta matemáticamente.

$$\theta = (X^T X)^{-1} X^T y$$

Aunque parece intimidante, la lógica es similar a resolver una ecuación simple como $5x = 10$. Para despejar x , dividirías por 5 (o multiplicarías por el inverso 5^{-1}). En el mundo de las matrices:

1. Queremos encontrar θ tal que $X\theta \approx y$
2. No podemos simplemente "dividir" por la matriz X .
3. La operación $(X^T X)^{-1} X^T$ es, en términos muy simplificados, la forma matricial de "pasar la X al otro lado dividiendo" para despejar la θ .

Característica	Descenso del Gradiente	Ecuación Normal (La de tu imagen)
Learning Rate (α)	Necesitas elegir uno y ajustarlo.	No necesitas elegir nada.
Iteraciones	Muchas iteraciones.	Ninguna . Es un solo cálculo directo.
Complejidad	Funciona bien incluso con millones de características (n grande). $O(kn^2)$	Se vuelve muy lenta si tienes muchas características (p.ej., $n > 10,000$) porque calcular la inversa de una matriz $(X^T X)^{-1}$ es costoso computacionalmente. $O(n^3)$
Resultado	Aproximado (muy cercano al óptimo).	Exacto (el óptimo matemático). Aunque a veces puede no funcionar porque no toda matriz es invertible.

5.6 Problemas Fundamentales: Sesgo vs. Varianza

Cuando entrenamos un modelo, podemos fallar por dos extremos opuestos:

5.6.1 Underfitting (Subajuste / Alto Sesgo)

Occurre cuando el modelo es **demasiado simple** e ignora la información de los datos porque "ya cree saber la respuesta"

- **Ejemplo:** Intentar ajustar una línea recta a datos que forman una curva parabólica.
- **Síntoma:** Tiene mucho error tanto en el entrenamiento como en la prueba. No aprende bien.

5.6.2 Overfitting (Sobreajuste / Alta Varianza)

Occurre cuando el modelo es **demasiado complejo** y sensible. Presta tanta atención a los detalles y al ruido de los datos de entrenamiento que, si le cambias un poco los datos, cambia totalmente su predicción. **No aprende, memoriza**.

- **Ejemplo:** Un polinomio de grado 10 que pasa por **todos** los puntos haciendo zig-zag.
- **Síntoma:** Error bajísimo en entrenamiento, pero altísimo en datos nuevos (falla al generalizar).

5.7 Solución: Regularización

¿Cómo evitamos el Overfitting si queremos usar modelos complejos (muchas variables)?

La **Regularización** consiste en "castigar" al modelo si usa pesos (θ) muy grandes. Modificamos la Función de Coste añadiendo un término extra.

Imagina una función matemática (un polinomio) que hace muchas curvas locas para pasar por todos los puntos (Overfitting). Para que una curva suba y baje bruscamente en distancias cortas, necesita multiplicar las variables por números muy grandes (pesos θ altos).

- Si obligamos a que esos números (θ) sean **pequeños**, la curva se vuelve más **suave, plana y sencilla**, evitando esos zig-zags extremos.

La función de coste $J(\theta)$ es lo que el ordenador intenta minimizar (hacer lo más pequeña posible).

$$J(\theta) = \underbrace{\text{Error de Predicción}}_{\text{"Qué tan bien acierto"}} + \underbrace{\lambda \sum \theta_j^2}_{\text{"Impuesto por complejidad"}}$$

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^m \theta_j^2 \right]$$

Imagina que eres un profesor evaluando a un alumno (el modelo):

1. **Error Original:** Es la nota del examen. Quieres que falle lo menos posible.
2. **Término de Regularización ($\lambda \sum \theta^2$):** Es una multa por usar "chuletas" o respuestas demasiado complicadas.

El papel de Lambda (λ): El tamaño de la multa. El parámetro λ es el "juez" que decide cuánto nos importa la simplicidad frente a la precisión exacta:

- Si λ es muy ALTO (Multas enormes):
El modelo dice: "¡Oye, usar pesos grandes me sale carísimo en la fórmula! Mejor pongo casi todos los pesos (θ) cercanos a cero para que el coste total sea bajo".
 - **Consecuencia:** El modelo se vuelve una línea casi plana. Es **demasiado simple** (riesgo de **Underfitting**).
- Si λ es 0 (Sin multa):
El modelo dice: "Solo me importa acertar los puntos, no me importa si uso ecuaciones complicadísimas".

- **Consecuencia:** El modelo hace zig-zags locos para tocar todos los puntos. Es **demasiado complejo** (riesgo de *Overfitting*).

La regularización busca el equilibrio perfecto (un λ intermedio) donde el modelo acierte lo suficiente (bajo error) pero manteniendo una curva suave (pesos bajos) para poder generalizar bien con datos nuevos. Reducir varianza, atenua el efecto de la correlación entre predictores y minimiza la influencia en el modelo de los predictores menos relevantes. Por lo general, aplicando regularización se consiguen modelos con mayor poder predictivo (generalización).