Facsímil 01

Los cimientos

Capítulo 11

Machine learning clásico: el mapa antes de los LLM

686f6c61 · 2026

Capítulo 11: Machine learning clásico: el mapa antes de los LLM

Entrando en el tema

Hemos pasado diez capítulos construyendo los cimientos: neuronas, redes, retropropagación, embeddings. Todo eso es *deep learning*. Pero antes del *deep learning*, y todavía hoy para la mayoría de problemas del mundo real, existe el *machine learning* clásico.

No es un primo pobre. Es un conjunto de herramientas más simples, más rápidas y más interpretables que resuelven la mayoría de problemas que te encontrarás. Clasificar correos, predecir ventas, detectar fraude, agrupar clientes: todo esto se hacía con ML clásico décadas antes de que existiera ChatGPT. Y se sigue haciendo.

Este capítulo es tu mapa. No te convertirá en experto en cada técnica, pero te dará el vocabulario para saber qué buscar cuando te enfrentes a un problema real.

El paisaje del ML clásico

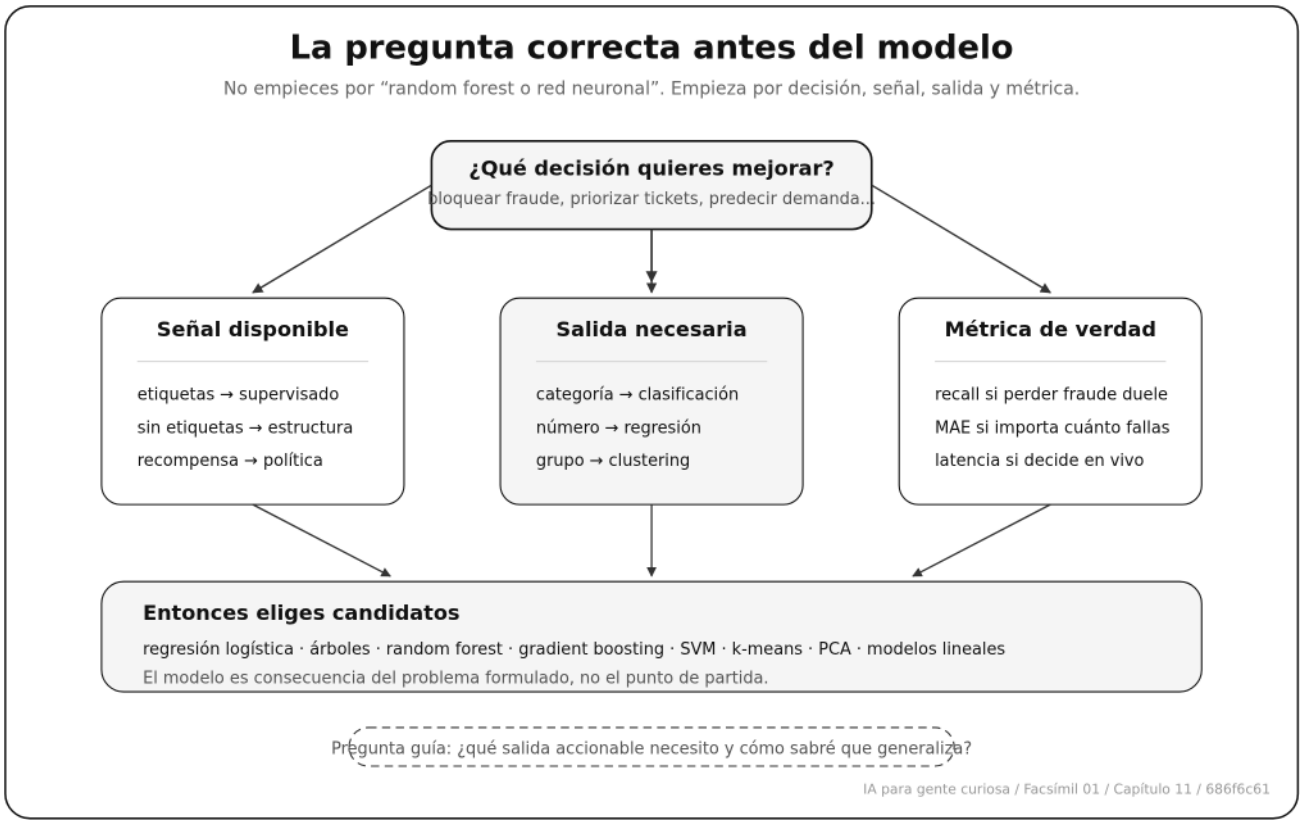
El *machine learning* se organiza alrededor de una pregunta fundamental: ¿qué tipo de señal de aprendizaje tienes?[1]

Supervisado. Tienes ejemplos etiquetados: cada entrada viene con su respuesta correcta. «Este *email* es *spam*», «esta imagen contiene un gato», «esta transacción es fraudulenta». El modelo aprende a imitar esas etiquetas. Es el paradigma más común y el que mejor funciona cuando tienes datos etiquetados de calidad.

No supervisado. No tienes etiquetas. Buscas estructura oculta: agrupar clientes por comportamiento, detectar anomalías, reducir la dimensionalidad de tus datos. El modelo encuentra patrones, pero tú tienes que interpretarlos. Que el algoritmo encuentre tres grupos no significa que esos tres grupos signifiquen algo útil para tu negocio.

Refuerzo. No tienes ejemplos etiquetados, pero sí una señal de recompensa. Un agente actúa en un entorno, recibe *feedback* y aprende a maximizar la recompensa acumulada. Es el paradigma de los juegos, la robótica y, cada vez más, del post-entrenamiento de LLMs (RLHF).

Semi-supervisado y auto-supervisado. Variantes híbridas. El semi-supervisado combina unos pocos ejemplos etiquetados con muchos sin etiquetar. El auto-supervisado genera sus propias etiquetas a partir de los datos: predecir la siguiente palabra en un texto, predecir la zona recortada de una imagen. Es el paradigma que hizo posibles los LLMs.



Anatomía de un dataset

Antes de elegir algoritmo, necesitas entender tus datos.[2]

- **Instancia (fila):** un ejemplo concreto. Un cliente, una transacción, una imagen, un *email*.
- **Feature (columna, variable, atributo):** una característica medible de la instancia. Edad, importe, número de caracteres, color promedio.
- **Label / target (etiqueta):** lo que quieres predecir. «Fraude / No fraude», «150 000 €», «Categoría A».
- **Dataset:** el conjunto de instancias con sus *features* y, si es supervisado, sus etiquetas.

Regla práctica: si no puedes explicar qué representa cada fila, qué significa cada columna y qué decisión habilita la predicción, no tienes un problema de ML. Tienes datos sueltos. Resolver esa ambigüedad es el primer paso de cualquier proyecto.

Clasificación vs regresión

La primera decisión técnica es el tipo de salida.[3]

Clasificación. La salida es una categoría discreta. «*Spam* / No *spam*», «Gato / Perro / Pájaro», «Fraude / No fraude». El modelo predice una clase o una distribución de probabilidad sobre las clases. Con esa probabilidad, tú decides el umbral: ¿bloqueo toda transacción con probabilidad de fraude > 0.3, o solo las que superan 0.9?

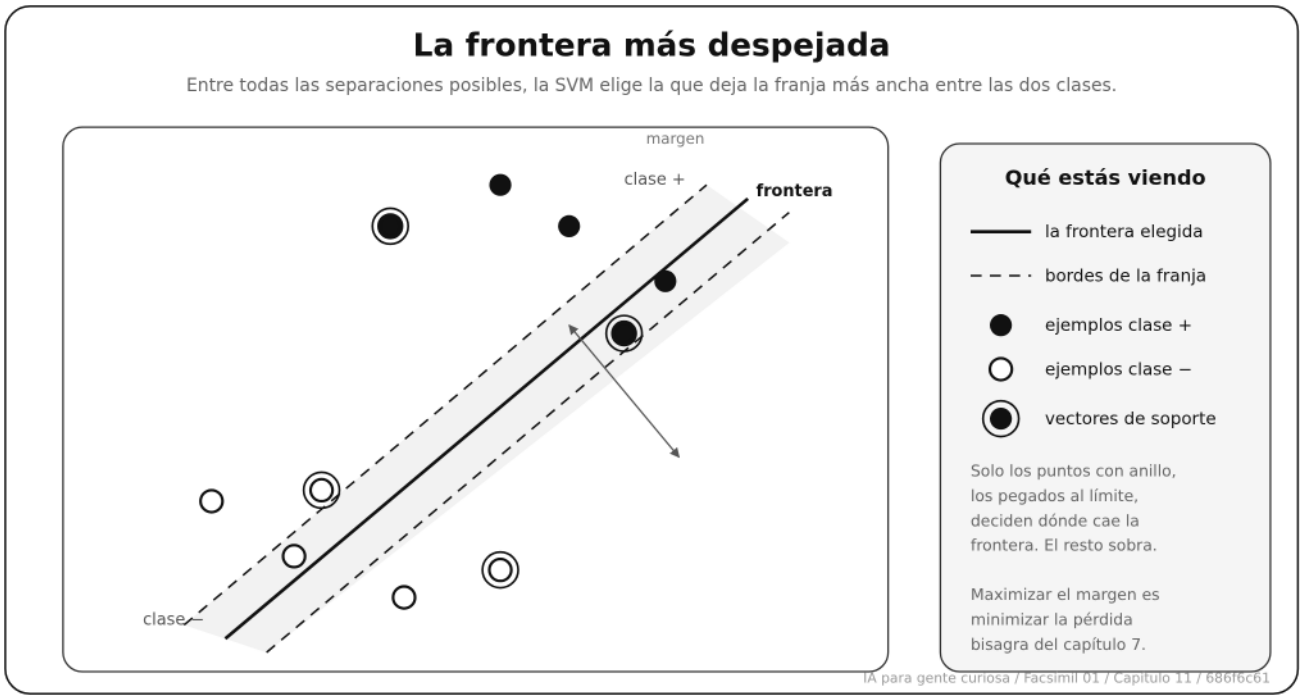
Regresión. La salida es un valor continuo. El precio de una casa, la temperatura de mañana, los ingresos del próximo trimestre. Aquí no existe «acertar o fallar» de forma binaria: importa cuánto te alejas del valor real. Un error de 1 000 € en una predicción de 300 000 € es aceptable; un error de 100 000 € no lo es.

Error común: convertir todo en clasificación porque es más cómodo. Si el valor numérico importa para decidir, usa regresión. Si solo importa el orden relativo, quizá necesitas *ranking*. Si una probabilidad dispara una acción, necesitas calibración, no solo clasificación.

Un modelo con criterio: las máquinas de vectores de soporte

Entre los modelos supervisados clásicos hay uno que merece una parada propia, porque encarna una idea elegante que sigue viva mucho después de su época dorada: la **máquina de vectores de soporte** (en inglés *support vector machine*, SVM).[4] Para una clasificación binaria con dos clases bien separadas, hay infinitas rectas (o planos) que las dividen sin error. La pregunta que se hace la SVM es cuál de todas elegir, y su respuesta tiene sentido: la que deja **la franja más ancha posible** entre las dos clases, es decir, la frontera de **margen máximo**.

La intuición es la de un camino. Si tienes que trazar una raya entre dos grupos de casas, la más robusta no es la que pasa rozando una de ellas, sino la que deja el mayor espacio libre a ambos lados; así, un dato nuevo que caiga un poco descolocado todavía quedará del lado correcto. De ese planteamiento sale el nombre del modelo: la frontera no depende de todos los ejemplos, sino solo de los pocos que tocan los bordes de la franja, los más difíciles, los pegados al límite. Esos ejemplos son los **vectores de soporte**, y son los únicos que sostienen la frontera; podrías borrar todos los demás y la solución no se movería.



Entrenar una SVM es, por debajo, exactamente lo que viste en el capítulo de funciones de pérdida: minimizar la **pérdida bisagra** (*hinge*), la que no se conforma con caer del lado correcto y exige hacerlo con holgura. Esa conexión cierra el círculo entre el modelo y su pérdida: el deseo geométrico de «la franja más ancha» y la fórmula $\max(0,\, 1 - m)$ son la misma idea contada de dos maneras.

Dos piezas más completan el modelo y explican por qué fue tan dominante en los años dos mil. La primera es el **margen blando** (*soft margin*): los datos reales no se separan limpiamente, sus clases se solapan, así que la SVM permite que algunos ejemplos invadan la franja o caigan del lado equivocado, a cambio de un coste. Un parámetro, llamado habitualmente C, regula ese equilibrio entre dejar la franja ancha y tolerar pocos errores; es la palanca de regularización del modelo. La segunda es el **truco del núcleo** (*kernel trick*): cuando ninguna recta separa bien las clases, la SVM puede medir la similitud entre ejemplos como si los hubiera proyectado a un espacio de muchas más dimensiones, donde sí existe una frontera plana, sin llegar a construir ese espacio de forma explícita. Con un núcleo adecuado, como el RBF, una frontera recta en ese espacio invisible se ve como una curva flexible en el original. Eso le dio a las SVM una potencia notable con pocos datos, antes de que el *deep learning* tomara el relevo.

¿Cuándo elegirla hoy? Las SVM rinden bien con conjuntos de datos pequeños o medianos y con muchas *features*, como la clasificación de texto, y cuando quieres una frontera robusta y bien fundamentada. En datos tabulares grandes, los *ensembles* de árboles (*random forest*, *gradient boosting*) suelen ganarles en la práctica, y cuando hay muchísimos datos y señal compleja, las redes neuronales se imponen. Pero la idea del margen máximo no se ha jubilado: sigue latiendo en la pérdida bisagra, en el modo en que pensamos la robustez de una frontera, y en la intuición de que los ejemplos difíciles, los del borde, son los que de verdad definen un clasificador.

Validación: cómo saber si generaliza

Entrenar un modelo es fácil. Lo difícil es saber si funcionará con datos que no ha visto nunca.[5]

Train / validation / test. Divide tus datos en tres conjuntos antes de tocar nada. Entrenas con *train*, ajustas hiperparámetros con *validation* y mides el rendimiento final con *test*. Si usas *test* para tomar decisiones, ya no es *test*: es *validation* y necesitas uno nuevo.

Cross-validation. Cuando tienes pocos datos, el *k-fold* entrena el modelo K veces, cada una con una partición distinta de train/validation. Te da una estimación más robusta del rendimiento y su variabilidad. No te fíes de una sola métrica con suerte.

Data leakage. El error más insidioso. Una columna que contiene información del futuro, un identificador que correlaciona con la etiqueta, un duplicado entre train y test. El modelo aprende una señal que no existirá en producción. Y tu métrica de validación será excelente... hasta que despliegues.

Data drift. El mundo cambia. Tus clientes cambian, los precios cambian, las campañas de *marketing* cambian. Un modelo entrenado con datos de 2023 puede fallar en 2026 aunque el código no haya cambiado. Monitoriza.

Overfitting y underfitting

Overfitting. El modelo memoriza los datos de entrenamiento. La pérdida de entrenamiento es mínima, pero la de validación es alta. Es el estudiante que se aprende las respuestas del examen de memoria: saca un 10 en el simulacro y suspende el examen real.[6] Solución: más datos, regularización, *dropout*, modelos más simples.

Underfitting. El modelo es demasiado simple para capturar los patrones de los datos. Tanto la pérdida de entrenamiento como la de validación son altas. Es el estudiante que ni siquiera ha abierto el libro. Solución: modelo más complejo, más *features*, más tiempo de entrenamiento.

El diagnóstico rápido: si ambas pérdidas son altas → *underfitting*. Si la de entrenamiento es baja pero la de validación es alta → *overfitting*. Si ambas son bajas y cercanas → buen trabajo.

Matriz de confusión, precision y recall

Cuando clasificas, la exactitud (*accuracy*) no cuenta toda la historia. Imagina un detector de fraude: el 99 % de las transacciones son legítimas. Un modelo que diga «no fraude» siempre tendrá un 99 % de *accuracy*. Y será completamente inútil.

La **matriz de confusión** desglosa lo que realmente ocurre:

	PREDICHO: POSITIVO	**PREDICHO: NEGATIVO**
Real: positivo	TP (verdadero positivo)	FN (falso negativo)
Real: negativo	FP (falso positivo)	TN (verdadero negativo)

De aquí salen las tres métricas que de verdad importan. La **precision** mide, de todo lo que el modelo marcó como positivo, qué parte lo era en realidad. Si predice fraude, responde cuántas de esas alertas eran fraudes reales:

$$\text{Precision} = \frac{TP}{TP + FP}$$

El **recall** (sensibilidad) mide, de todo lo que era realmente positivo, qué parte detectó el modelo. De todos los fraudes que ocurrieron, cuántos llegaste a capturar:

$$\text{Recall} = \frac{TP}{TP + FN}$$

El **F1** resume ambas en una sola cifra con su media armónica, útil cuando necesitas un equilibrio entre las dos:

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

SÍMBOLO	SIGNIFICADO	EJEMPLO
TP	Verdaderos positivos: casos positivos que el modelo acertó	fraudes detectados
FP	Falsos positivos: negativos que el modelo marcó como positivos	compras legítimas bloqueadas
FN	Falsos negativos: positivos que el modelo dejó escapar	fraudes no detectados
$\mathrm{Precision}$	Fiabilidad de las alertas positivas, entre 0 y 1	0,90 significa que nueve de cada diez alertas son fraude real
Recall	Cobertura de los positivos reales, entre 0 y 1	0,70 significa que capturas siete de cada diez fraudes
F_1	Media armónica de precision y recall, entre 0 y 1	equilibra ambas en un único número

En palabras: la precision responde «cuando aviso, ¿acierto?» y el recall responde «de todo lo que debía cazar, ¿cuánto cacé?». Subir una suele bajar la otra, porque bajar el umbral captura más positivos reales (más recall) a costa de más falsas alarmas (menos precision). El F1 resume ese equilibrio, y como la media armónica tira hacia el valor más bajo, basta con que precision o recall sea pequeño para que el F1 se hunda. Eso evita quedarte contento con un modelo que brilla en una métrica y fracasa en la otra.

La elección entre *precision* y *recall* depende del coste del error. En detección de fraude, un FN (dejar pasar un fraude) puede costar miles de euros; un FP (bloquear una compra legítima) puede costar un cliente. En diagnóstico médico, un FN (no detectar una enfermedad) puede costar una vida. No optimices *accuracy* a ciegas: optimiza la métrica que refleja el coste real del error en tu dominio.

Baseline, umbral y calibración

Un modelo no se evalúa en el vacío. Se evalúa contra una **baseline**: una regla simple que representa “lo mínimo que habría que superar”. Puede ser predecir siempre la clase mayoritaria, una regla de negocio escrita a mano o el sistema actual. Si tu modelo complejo no mejora claramente esa baseline en la métrica que importa, no has ganado ingeniería; has añadido complejidad.

Después viene el **umbral**. Muchos clasificadores no devuelven directamente una clase, sino una puntuación o probabilidad: “este ticket tiene 0,73 de probabilidad de ser urgente”. Convertir 0,73 en una acción exige una política: con umbral 0,5 quizá escalas demasiados tickets; con 0,9 quizá dejas pasar incidencias graves. El umbral no se elige por estética, sino por coste de FP y FN.

Y si vas a usar la probabilidad como probabilidad, necesitas **calibración**. Un modelo está bien calibrado si, entre todos los casos donde dice 0,8, aproximadamente el 80 % acaba siendo positivo.[7] Esto importa cuando una puntuación dispara una acción real: revisión manual, bloqueo, reembolso, alerta médica, priorización de SLA. Un modelo puede ordenar bien los casos y aun así mentir en la escala de sus probabilidades.

Clustering: agrupar sin etiquetas

A veces no tienes etiquetas, pero sospechas que tus datos tienen estructura. El *clustering* agrupa instancias por similitud. El algoritmo más sencillo y usado es **k-means**:[8]

1. Elige K (el número de grupos que quieres encontrar).
2. Coloca K centroides aleatoriamente.
3. Asigna cada punto al centroide más cercano.
4. Recalcula los centroides como la media de los puntos asignados.
5. Repite hasta que los centroides se estabilizan.

Es simple, rápido y funciona sorprendentemente bien. Pero tiene trampas:

- **Tienes que elegir K.** Si no sabes cuántos grupos hay, toca probar. Los métodos del codo (*elbow*) y la silueta (*silhouette*) ayudan, pero no sustituyen el criterio.
- **La distancia importa.** Normalmente se usa distancia euclídea. Si una *feature* tiene valores en millones y otra entre 0 y 1, la primera domina completamente la agrupación. Normaliza.
- **La semilla importa.** Distintas inicializaciones dan distintos resultados. No te cases con la primera agrupación que veas.

Cuándo no confiar en clusters. Que el algoritmo encuentre grupos no significa que esos grupos signifiquen algo. Un *cluster* puede ser un artefacto de la métrica de distancia, de una *feature* mal escalada o del azar. Valida siempre contra conocimiento del dominio: ¿tienen sentido esos grupos para quien conoce el negocio? Si no puedes explicar qué tienen en común los miembros de un *cluster*, probablemente no sea un hallazgo, sino ruido.

Más allá de k-means: jerárquico y por densidad

k-means es el martillo, pero no todo es un clavo. Arrastra dos supuestos fuertes: que sabes cuántos grupos hay (la K) y que esos grupos son más o menos esféricos y de tamaño parecido. Cuando alguna de esas dos cosas no se cumple, conviene tener a mano otras dos familias clásicas.

Clustering jerárquico (aglomerativo). En vez de fijar K de antemano, empieza tratando cada punto como su propio grupo y, paso a paso, funde los dos grupos más cercanos, hasta que queda uno solo. El resultado no es una partición, sino un árbol de fusiones —el **dendrograma**— que puedes cortar a la altura que quieras para obtener 2, 5 o 20 grupos. La decisión central es cómo medir la distancia entre dos grupos, lo que se llama el *enlace*: el mínimo entre sus puntos (*single*, que tiende a encadenar), el máximo (*complete*), la media, o el criterio de Ward, que funde los grupos que menos aumentan la varianza interna.[9] Su ventaja es doble: no te obliga a elegir K y te da una jerarquía legible. Su coste es que escala mal (cuadrático o peor en memoria y tiempo), así que no es para millones de puntos.

DBSCAN (por densidad). Define un *cluster* como una zona densa de puntos separada de otras por zonas vacías. Solo necesita dos parámetros: un radio `eps` y un mínimo de vecinos `minPts` para considerar un punto «interior». A partir de ahí, expande cada región densa hasta donde llega la densidad.[10] Tiene tres virtudes que a k-means le faltan: **descubre K por sí solo**, encuentra **grupos de forma arbitraria** (no solo esferas, también lunas o anillos) y **marca como ruido** los puntos aislados, en vez de forzarlos dentro de un grupo. Su talón de Aquiles es elegir `eps`: si los grupos tienen densidades muy distintas, un único radio no sirve para todos.

Evaluar agrupaciones sin etiquetas: la silueta

Sin etiquetas no existe el «acierto», así que ¿cómo sabes si una agrupación es buena? Una medida muy usada es el **coeficiente de silueta**. Para cada punto compara lo cerca que está de los suyos con lo cerca que está del grupo vecino más próximo. Si a es la distancia media a su propio grupo y b la distancia media al grupo ajeno más cercano, la silueta del punto es:

$$s = \frac{b - a}{\max(a, b)} \in [-1, 1]$$

Cerca de 1 significa que el punto está bien dentro de su grupo y lejos de los demás; cerca de 0, que cae justo en la frontera; negativa, que probablemente está en el grupo equivocado.[11] Promediar la silueta de todos los puntos da una nota global que, entre otras cosas, sirve para tantear cuántos grupos tiene tu *dataset*: pruebas varios valores de K y te quedas con el que mejor silueta da. Eso sí, sigue siendo una pista, no una verdad: una silueta alta sobre una métrica mal escalada también engaña.

Reducir dimensiones: PCA y la maldición de la dimensionalidad

Cuando cada instancia tiene cientos o miles de *features*, aparecen problemas que no se ven con dos o tres columnas: las distancias se vuelven poco informativas (casi todo queda «lejos» de casi todo), los modelos sobreajustan con facilidad y, además, no hay forma de dibujar los datos para mirarlos. Es la **maldición de la dimensionalidad**. La respuesta clásica es **reducir la dimensión**: describir cada punto con muchas menos coordenadas perdiendo lo mínimo.

El método de referencia es el **análisis de componentes principales (PCA)**. La idea geométrica es buscar las direcciones en las que los datos más varían y proyectar sobre ellas. Esas direcciones —las **componentes principales**— son los vectores propios de la matriz de covarianza de los datos, ordenados por su valor propio, que mide cuánta varianza captura cada uno.[12] [13] Si X son los datos centrados y $\Sigma = \frac{1}{n} X^\top X$ su covarianza, PCA resuelve $\Sigma v = \lambda v$: cada vector propio v es una componente y su λ dice qué fracción de la varianza explica. Quedándote con las primeras componentes —las de mayor λ— conservas casi toda la información con muchas menos columnas. El **gráfico de sedimentación** (*scree plot*), que dibuja la varianza explicada por componente, ayuda a decidir cuántas conservar.

PCA es una proyección **lineal**, así que cuando la estructura interesante es curva conviene otra herramienta solo para *visualizar*: **t-SNE** y **UMAP** colocan los puntos en dos dimensiones intentando preservar la vecindad local, y producen esos mapas donde los grupos saltan a la vista.[14] Una advertencia importante: estos mapas deforman las distancias globales, así que sirven para *ver* estructura, no para medirla ni para alimentar otro modelo con sus coordenadas. Esta reducción de dimensión es, además, el puente con el resto del facsímil: los *embeddings* del facsímil 4 y las representaciones del facsímil 8 son exactamente esto — describir cada cosa con un vector corto que conserva lo que importa.

Cómo funciona por dentro

Hasta aquí hemos hablado de clasificación, métricas y validación desde fuera. Conviene abrir uno de estos algoritmos y mirar su engranaje. Elijo el árbol de decisión porque conecta de forma directa con la clasificación y porque es el ejemplo canónico de modelo interpretable. La idea de partida es sencilla. Un árbol de decisión es una secuencia de preguntas sobre las *features* de cada instancia. Cada pregunta divide el grupo de ejemplos en dos partes y la meta es que esas partes queden lo más puras posible, es decir, que dentro de cada parte casi todos los ejemplos pertenezcan a la misma clase. Predecir consiste después en recorrer el árbol desde la raíz, respondiendo cada pregunta, hasta llegar a una hoja que asigna una clase.

El corazón del algoritmo es cómo elige cada pregunta. En cada nodo el árbol evalúa muchos cortes posibles del tipo «edad menor que 30», «importe mayor que 500», «tipo de cliente igual a premium» y se queda con el corte que más reduce la impureza de los grupos resultantes. La impureza se mide con un criterio numérico. El más usado es el índice de Gini, que cuantifica la probabilidad de etiquetar mal un ejemplo elegido al azar si le asignamos una clase según la proporción del grupo. Su fórmula es:

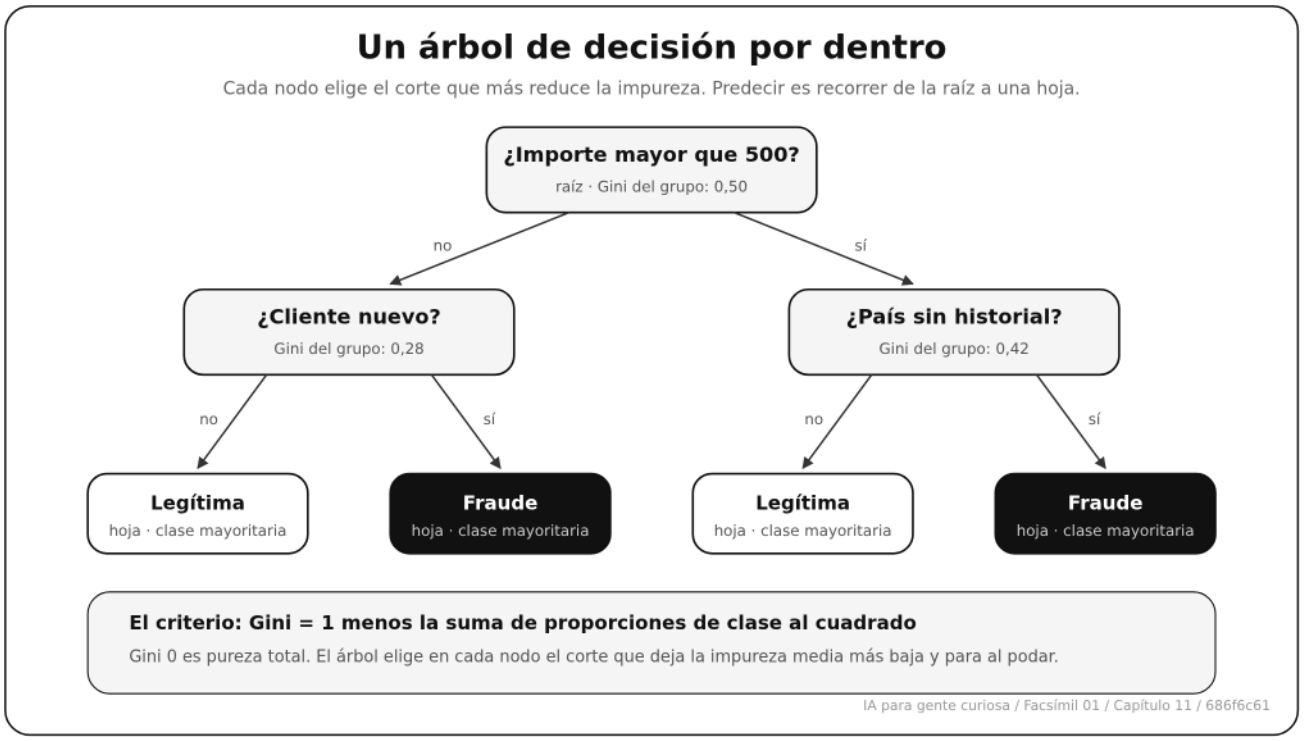
$$G = 1 - \sum_{k=1}^{C} p_k^2$$

SÍMBOLO	SIGNIFICADO
G	Índice de Gini del grupo, entre 0 y un valor máximo según el número de clases
C	Número de clases posibles
p_k	Proporción de ejemplos del grupo que pertenecen a la clase k
$\sum$	Suma sobre todas las clases, de la primera a la última

En palabras: si todos los ejemplos del grupo son de la misma clase, una proporción vale 1 y el resto 0, la suma de cuadrados vale 1 y el Gini queda en 0, que es la pureza total. Si las clases están muy mezcladas, las proporciones se reparten, la suma de cuadrados baja y el Gini sube. El árbol prueba cada corte candidato, calcula el Gini de los dos grupos que genera, los pondera por su tamaño y elige el corte que deja la impureza media más baja. Una alternativa equivalente en espíritu es la entropía, que mide la misma idea de mezcla con logaritmos. Ambas premian los cortes que separan limpiamente las clases.

El proceso es recursivo. Una vez elegido el mejor corte de la raíz, el árbol repite exactamente el mismo razonamiento dentro de cada rama, buscando ahora el mejor corte para el subgrupo que ha quedado en esa rama. Y vuelve a repetirlo en los subgrupos de los subgrupos. Este crecimiento continúa hasta una condición de parada: que el grupo sea puro, que queden muy pocos ejemplos para seguir dividiendo, que se alcance una profundidad máxima fijada de antemano o que ningún corte mejore la impureza. Aquí reaparece una idea de este capítulo. Si dejas crecer el árbol sin freno, acabará memorizando el conjunto de entrenamiento con ramas hechas a medida de cada ejemplo, que es justo el *overfitting* descrito antes. Por eso se podan los árboles o se limita su profundidad, lo que equivale a buscar el modelo más simple que aún explica los datos.

La razón por la que este algoritmo es interpretable está en su propia mecánica. La predicción de un árbol es un camino de preguntas legibles que cualquier persona del negocio puede seguir: «se marcó como fraude porque el importe superaba 500, el cliente era nuevo y la operación venía de un país sin historial». No hay nada oculto. Puedes imprimir el árbol entero y auditar cada decisión. Una red neuronal profunda, en cambio, distribuye su criterio entre millones de pesos sin un significado individual claro, de modo que sabes qué predice pero no puedes explicar fácilmente por qué. Esta transparencia es la que conecta con el espíritu del *machine learning* clásico que recorre todo el capítulo: modelos que no solo aciertan, sino que permiten justificar la decisión ante quien la sufre o se beneficia de ella. Por eso, cuando necesitas defender una clasificación delante de un cliente, un regulador o un equipo de negocio, un árbol de decisión sigue siendo una de las herramientas más valiosas del catálogo.



En el día a día

Cuando te enfrentes a un problema real, la primera pregunta es si tienes etiquetas. Si las tienes, es un problema supervisado: clasificación cuando la salida es una categoría y regresión cuando es un número. En ambos casos conviene empezar por lo simple, una regresión logística o un árbol de decisión, antes de saltar a una red neuronal. Si el modelo sencillo ya resuelve el problema, te has ahorrado complejidad, coste y un sistema difícil de explicar.

Si no tienes etiquetas, el terreno cambia. El clustering sirve para explorar y ver si los datos esconden grupos, la detección de anomalías para encontrar lo raro y la reducción de dimensionalidad para visualizar y comprimir. Son herramientas de descubrimiento, no de predicción: te muestran estructura, pero la interpretación corre de tu cuenta.

El volumen de datos también manda. Con cientos o miles de ejemplos, el ML clásico es tu mejor apuesta, porque los árboles, las SVMs y los *ensembles* rinden bien en ese rango. El *deep learning* solo empieza a brillar cuando hay decenas de miles de ejemplos o más. Y si necesitas justificar tus decisiones ante un cliente, un regulador o un compañero, un árbol de decisión te dice exactamente por qué clasificó algo como lo hizo, mientras que una red de cien capas no te da esa cuenta.

Por qué debería importarte

El *deep learning* no ha dejado obsoleto al ML clásico. Lo ha complementado. Para muchos problemas, en especial con datos tabulares y pocos ejemplos, un *random forest* o una regresión logística siguen siendo más rápidos, más baratos y más interpretables que cualquier red neuronal.

Además, el ML clásico te da el vocabulario para medir. *Precision*, *recall*, *overfitting*, *cross-validation*: estos conceptos no desaparecen cuando usas LLMs. Se vuelven más importantes. Un agente de IA que clasifica tickets de soporte necesita una matriz de confusión igual que un clasificador de *spam* de 2005.

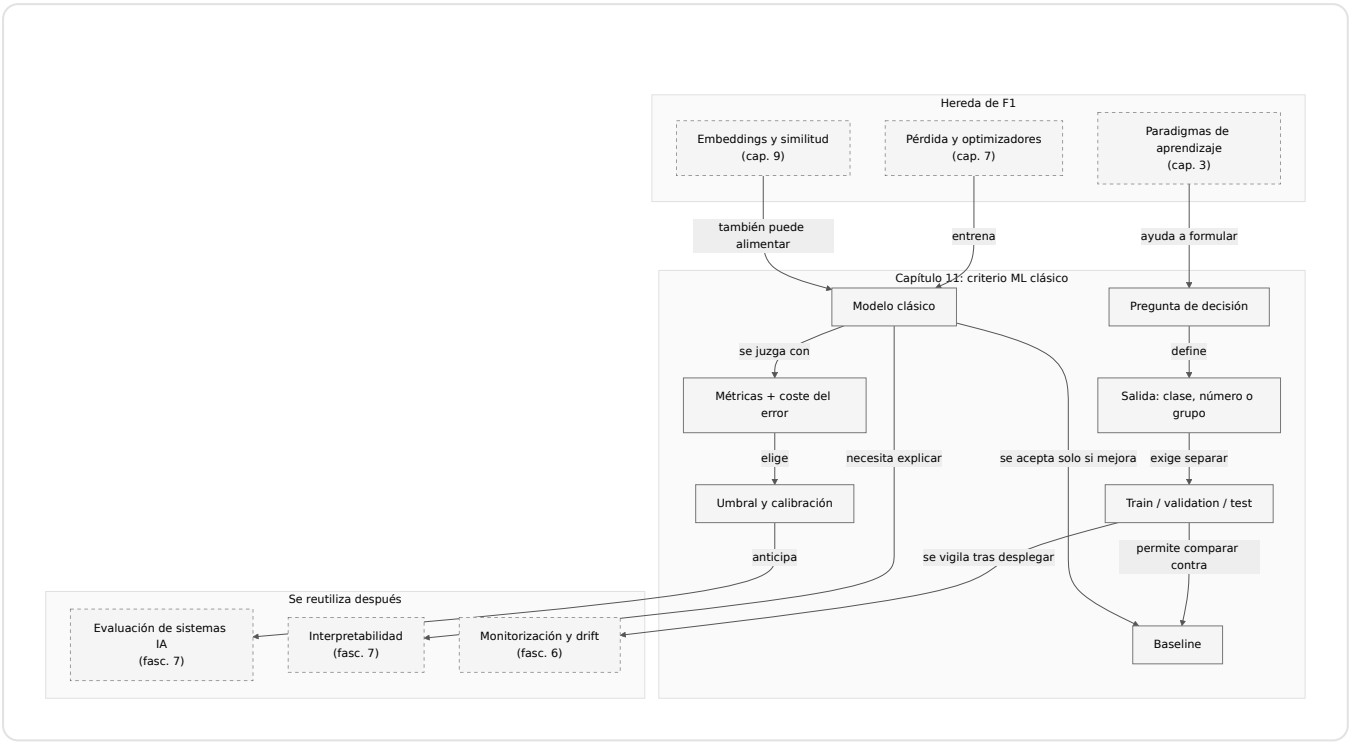
Dónde solía tropezar yo

ERROR	POR QUÉ ES UN ERROR	ANTÍDOTO
Medir solo *accuracy*	En conjuntos desbalanceados (1 % de fraude, 99 % legítimo), un modelo que siempre dice «legítimo» tiene 99 % de *accuracy* y es completamente inútil.	Mira *precision*, *recall* y F1. Si las clases están desbalanceadas, *accuracy* no te dice nada.
No separar train/validation /test	Si usas los mismos datos para entrenar y evaluar, el modelo parece perfecto... hasta que lo pones en producción.	Separa antes de tocar nada. *Test* se toca una sola vez, al final.
Ignorar la escala de las features	En k-means (y en cualquier algoritmo basado en distancia), una *feature* con valores grandes domina a las demás.	Normaliza o estandariza tus *features* antes de entrenar.
Interpretar clusters como realidades objetivas	El algoritmo siempre encuentra grupos, aunque los datos sean ruido puro.	Valida contra conocimiento del dominio. Si los grupos no tienen sentido para el negocio, no los uses para decidir.

Cómo encaja todo

Este mapa se lee desde la decisión, no desde el algoritmo. Venimos de los principios del capítulo 3 y de las pérdidas del capítulo 7, pero aquí aparece la disciplina que seguirá en todo el facsímil: definir salida, baseline, validación y métrica antes de enamorarse de una arquitectura.

Lo que se aprende en este capítulo vuelve después en evaluación de LLMs, calibración, interpretabilidad y operación. Un sistema con agentes o RAG también necesita train/test mental: qué casos pruebo, qué métrica acepto, qué coste tiene cada error y cuándo prefiero una regla simple.



Vocabulario aprendido

TÉRMINO	DEFINICIÓN
Clasificación	Tarea supervisada donde la salida es una categoría discreta entre un conjunto predefinido de clases.
Regresión	Tarea supervisada donde la salida es un valor numérico continuo.
Overfitting	El modelo memoriza los datos de entrenamiento y no generaliza a datos nuevos.
Underfitting	El modelo es demasiado simple para capturar los patrones de los datos.
Matriz de confusión	Tabla que cruza predicciones con valores reales, desglosando aciertos, falsos positivos y falsos negativos.
Precision	Proporción de predicciones positivas que son realmente correctas.
Recall	Proporción de casos positivos reales que el modelo detectó.
F1	Media armónica de precision y recall; solo es alta cuando ambas lo son.
Árbol de decisión	Modelo que clasifica mediante una secuencia de preguntas sobre las *features*, legible de la raíz a la hoja.
Índice de Gini	Medida de impureza de un grupo; vale 0 cuando todos los ejemplos son de la misma clase.
Máquina de vectores de soporte (SVM)	Clasificador que separa dos clases con la frontera de margen máximo. Se entrena minimizando la pérdida bisagra.
Margen máximo	La franja más ancha posible entre las clases; la SVM elige la frontera que la maximiza.
Vectores de soporte	Los ejemplos pegados al borde de la franja; los únicos que determinan dónde cae la frontera.
Kernel trick	Truco que permite trazar fronteras no lineales midiendo similitudes como en un espacio de más dimensiones, sin construirlo.
Baseline	Regla o sistema simple que el modelo debe superar para justificar su complejidad.
Calibración	Grado en que las probabilidades predichas se corresponden con frecuencias reales observadas.

TÉRMINO	DEFINICIÓN
Clustering	Técnica no supervisada que agrupa instancias por similitud sin etiquetas predefinidas.

Antes de pasar página

- [ ] ¿Puedo distinguir entre clasificación y regresión con ejemplos concretos? (Si no, vuelve a «Clasificación vs regresión».)
- [ ] ¿Sé interpretar una matriz de confusión? (Si no, vuelve a «Matriz de confusión».)
- [ ] ¿Entiendo la diferencia entre *precision* y *recall*? (Si no, vuelve a esa sección.)
- [ ] ¿Puedo explicar qué baseline usaría y qué umbral elegiría si el coste de FP y FN cambia? (Si no, vuelve a «Baseline, umbral y calibración».)
- [ ] ¿Puedo explicar qué es el *overfitting* y cómo detectarlo? (Si no, vuelve a «Overfitting y underfitting».)
- [ ] ¿Entiendo qué es el margen máximo de una SVM y por qué solo importan los vectores de soporte? (Si no, vuelve a «Un modelo con criterio».)
- [ ] ¿Sé interpretar precision, recall, F1 y falsos negativos? (Si no, vuelve a «Matriz de confusión, precision y recall».)

En resumen

IDEA FUERZA	DETALLE
El ML clásico no ha muerto.	Para datos tabulares, pocos ejemplos o necesidad de explicabilidad, los árboles y las regresiones siguen siendo la mejor opción.
La pregunta no es «qué modelo», sino «qué salida, qué señal, cómo mido».	Define el problema antes de elegir la herramienta. Clasificación, regresión, clustering: cada uno tiene sus métricas.
Las métricas correctas dependen del coste del error.	No optimices *accuracy* a ciegas. Un FN en diagnóstico médico no vale lo mismo que un FP en recomendaciones de películas.
Una probabilidad no es una decisión hasta que eliges baseline, umbral y calibración.	Si el modelo no supera una regla simple o sus probabilidades no son fiables, todavía no tienes un sistema defendible.
El *clustering* encuentra estructura, no significado.	Valida siempre contra conocimiento del dominio. Un grupo no es un hallazgo hasta que alguien que conoce el negocio dice «esto tiene sentido».

Para saber más

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

Cortes, C. y Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273-297. https://doi.org/10.1007/BF00994018

Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org

Guo, C., Pleiss, G., Sun, Y. y Weinberger, K. Q. (2017). On calibration of modern neural networks. En *Proceedings of the 34th International Conference on Machine Learning* (pp. 1321-1330). PMLR. https://proceedings.mlr.press/v70/guo17a.html

Hastie, T., Tibshirani, R. y Friedman, J. (2009). *The elements of statistical learning* (2.ª ed.). Springer. https://web.stanford.edu/~hastie/ElemStatLearn/

LeCun, Y., Bengio, Y. y Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444. https://doi.org/10.1038/nature14539

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. En *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability* (Vol. 1, pp. 281-297). https://projecteuclid.org/euclid.bsmsp/1200512992

Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson.

Notas

1. Russell, S. y Norvig, P. (2021). *Artificial intelligence: a modern approach* (4.ª ed.). Pearson. Los capítulos 18 a 21 cubren los paradigmas de aprendizaje y establecen el marco conceptual para clasificar cualquier problema de ML según el tipo de datos y retroalimentación disponible. ↩

2. Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer. El capítulo 1 establece los fundamentos de la teoría de la probabilidad y la decisión, y define la terminología básica de *datasets*, *features* y variables objetivo. ↩

3. Hastie, T., Tibshirani, R. y Friedman, J. (2009). *The elements of statistical learning* (2.ª ed.). Springer. https://web.stanford.edu/~hastie/ElemStatLearn/. El capítulo 2 ofrece una visión general de aprendizaje supervisado y establece la distinción fundamental entre problemas de clasificación y regresión. ↩

4. Cortes, C. y Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273-297. https://doi.org/10.1007/BF00994018. El artículo introduce las SVM, que buscan la frontera de separación de máximo margen y popularizaron el uso de *kernels* para fronteras no lineales. ↩

5. Goodfellow, I., Bengio, Y. y Courville, A. (2016). *Deep learning*. MIT Press. https://www.deeplearningbook.org. El capítulo 5.2 aborda las técnicas de validación y la importancia de separar correctamente los conjuntos de entrenamiento, validación y prueba para obtener estimaciones fiables del rendimiento. ↩

6. Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324. Breiman demostró que los *ensembles* de árboles de decisión, como los *random forests*, mitigan el *overfitting* al promediar múltiples modelos entrenados con subconjuntos distintos de datos. ↩

7. Guo, C., Pleiss, G., Sun, Y. y Weinberger, K. Q. (2017). On calibration of modern neural networks. En *Proceedings of the 34th International Conference on Machine Learning* (pp. 1321-1330). PMLR. https://proceedings.mlr.press/v70/guo17a.html. El artículo muestra que modelos modernos pueden tener alta exactitud y estar mal calibrados, por lo que sus probabilidades no deben interpretarse sin comprobación. ↩

8. MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. En *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability* (Vol. 1, pp. 281-297). https://projecteuclid.org/euclid.bsmsp/1200512992.

MacQueen introdujo el algoritmo k-means, que sigue siendo uno de los métodos de *clustering* más utilizados por su simplicidad y eficiencia computacional. ↩

9. Ward, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, 58(301), 236-244. https://doi.org/10.1080/01621459.1963.10500845. Ward propuso el criterio de enlace que minimiza el incremento de varianza intragrupo al fundir, uno de los más usados en *clustering* aglomerativo. ↩

10. Ester, M., Kriegel, H.-P., Sander, J. y Xu, X. (1996). A density-based algorithm for discovering clusters in large spatial databases with noise. En *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)* (pp. 226-231). AAAI Press. DBSCAN encuentra el número de grupos por sí solo, descubre formas arbitrarias y marca como ruido los puntos que no encajan en ninguna región densa. ↩

11. Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20, 53-65. https://doi.org/10.1016/0377-0427(87)90125-7. Rousseeuw introdujo la silueta como ayuda gráfica para interpretar y validar agrupaciones. ↩

12. Pearson, K. (1901). On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(11), 559-572. https://doi.org/10.1080/14786440109462720. Pearson planteó por primera vez el ajuste de líneas y planos que mejor representan una nube de puntos, semilla de PCA. ↩

13. Jolliffe, I. T. (2002). *Principal Component Analysis* (2.ª ed.). Springer. El texto de referencia moderno sobre PCA, su álgebra, sus variantes y sus límites. ↩

14. van der Maaten, L. y Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9, 2579-2605. t-SNE preserva las distancias locales para revelar estructura de grupos en alta dimensión, pero deforma las distancias globales: úsalo para mirar, no para medir. ↩

Cuadernos para practicar

Escanea el código con el móvil para abrir el cuaderno en Google Colab y ejecutarlo sin instalar nada.

Machine learning clásico que decide de verdad

https://colab.research.google.com/github/686f6c61/iaparagentecuriosa-notebooks/blob/main/Facs%C3%ADmil%2001/04-ml-clasico-que-decide.ipynb