

Introducción a ML – Sesión 1

Cesar Garcia

2025

Objetivos de la sesión

- Comprender qué es el aprendizaje automático (Machine Learning).
- Diferenciar entre aprendizaje clásico y aprendizaje profundo.
- Identificar tipos de problemas: regresión, clasificación, clustering.
- Conectar conceptos con aplicaciones del mundo real.
- Entender por qué el ML funciona y qué patrones aprende.
- Construir intuiciones visuales clave.

¿Qué es el Machine Learning?

Definición

El Machine Learning permite a los sistemas aprender patrones a partir de datos sin ser programados explícitamente.



Por qué funciona el Machine Learning

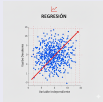



- El ML identifica patrones que no son evidentes ni siquiera al ver los datos.
- Aprende relaciones multivariadas donde cada variable influye en combinación con otras.
- Encuentra fronteras de decisión o reglas matemáticas imposibles de deducir manualmente.
- Aprovecha grandes volúmenes de datos para reducir ruido y generalizar a nuevos casos.

Ejemplo intuitivo

Detección de fraude en tarjetas de crédito:

- No hay una única señal clara de fraude.
- El ML aprende combinaciones sutiles: hora + monto + ubicación + historial + categoría del comercio.
- Los humanos jamás podrían ver todos esos patrones simultáneamente.

Tipos de Problemas

Tarea	Descripción	Imagen
Regresión	Predicción de valores numéricos.	
Clasificación	Asignación de categorías.	
Clustering	Agrupación sin etiquetas.	
Lenguaje Natural (NLP)	Procesamiento de texto y voz.	

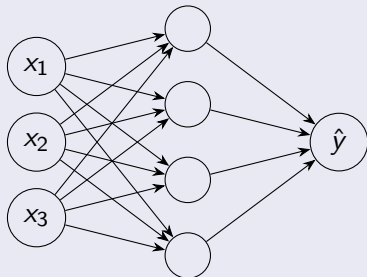
Algoritmos Clásicos

- Regresión lineal/polinomial
- KNN
- Árboles de decisión
- Random Forests
- Support Vector Machines (SVM)

Deep Learning (Redes Neuronales)

- Convolucionales (CNN)
- Recurrentes (RNN, LSTM)
- Variational Autoencoders (VAE)
- Generative Adversarial Networks (GAN)
- Transformers

Diagrama: Red neuronal simple (intuición)



- **Ventajas**

- Muy interpretables: cada coeficiente indica el efecto de una variable.
- Entrenamiento rápido incluso con muchos datos.
- Buena base para entender otros modelos.

- **Desventajas**

- No captura relaciones altamente no lineales.
- Sensible a outliers.
- Supone cierta estructura en los datos (linealidad, homocedasticidad, etc.).

- Idea: si la relación no es lineal, **ampliamos las características**.
- Ejemplo (una variable): $\hat{y} = w_0 + w_1x + w_2x^2 + w_3x^3$
- En la práctica:
 - Creamos nuevas columnas: (x, x^2, x^3, \dots)
 - Aplicamos regresión lineal sobre estas características.
- **Cuidado:**
 - Grado muy alto riesgo de **sobreajuste**.
 - Se suele combinar con **regularización** (Ridge, Lasso).

K-Nearest Neighbors (KNN) – Intuición

- No construye una fórmula explícita, sino que **memoriza los datos**.
- Para predecir un nuevo punto:
 - ① Busca los **k vecinos más cercanos** en el conjunto de entrenamiento.
 - ② Clasificación: votación mayoritaria entre las clases de esos vecinos.
 - ③ Regresión: promedio de los valores numéricos de los vecinos.
- Depende fuertemente de la **noción de distancia** (normalmente Euclídea).

- **Ventajas**

- Sencillo de entender e implementar.
- Puede modelar fronteras de decisión muy complejas.
- No necesita entrenamiento costoso (el “costo” está en la predicción).

- **Desventajas**

- Lento para predecir con muchos datos (hay que buscar vecinos).
- Muy sensible a:
 - Escala de las características (es necesario normalizar).
 - Elección de k .
- Mal rendimiento en espacios de alta dimensión (“maldición de la dimensionalidad”).

- k: número de vecinos.
 - k muy pequeño \rightarrow sobreajuste.
 - k muy grande \rightarrow subajuste.
- Tipo de distancia:
 - Euclídea, Manhattan, Minkowski, etc.
- Peso de los vecinos:
 - Todos iguales.
 - O ponderados por distancia (más peso a vecinos más cercanos).

Árboles de decisión – Intuición

- Modelo que aprende una secuencia de **decisiones tipo “si/entonces”**.
- Estructura en forma de árbol:
 - Nodo interno: condición (ej. “ $\text{edad} > 30?$ ”).
 - Rama: resultado de la condición (sí/no).
 - Hoja: predicción final (clase o valor numérico).
- Objetivo: dividir el espacio de características en regiones cada vez más “puras”.

- En cada nodo:
 - Se prueba dividir los datos por alguna característica y umbral.
 - Se mide qué tan “buena” es la división:
 - Clasificación: Gini, entropía.
 - Regresión: reducción de MSE.
 - Se elige la división que **mejor separa** las clases o reduce el error.
- Se repite recursivamente hasta:
 - Profundidad máxima.
 - Número mínimo de muestras por hoja.
 - O hasta que ya no mejore la pureza.

- **Ventajas**

- Muy interpretables (se puede visualizar el árbol).
- Manejan bien variables categóricas y numéricas.
- Capturan relaciones no lineales y efectos de interacción.

- **Desventajas**

- Un solo árbol profundo tiende a **sobreajustar**.
- Pequeños cambios en los datos pueden generar árboles muy diferentes.
- Menor rendimiento que ensambles como Random Forests o Gradient Boosting.

- Un **ensamble de árboles de decisión**.
- En vez de entrenar un solo árbol:
 - Se entrenan muchos árboles sobre diferentes subconjuntos de datos y características.
 - Clasificación: votación mayoritaria entre árboles.
 - Regresión: promedio de las predicciones.
- Idea clave: muchos modelos débiles \rightarrow un modelo fuerte mediante **promediado**.

- Dos fuentes de aleatoriedad:
 - **Bootstrap**: cada árbol ve una muestra aleatoria (con reemplazo) del conjunto de entrenamiento.
 - **Subconjunto de características**: en cada división del árbol, se considera sólo un subconjunto aleatorio de variables.
- Beneficios:
 - Reduce varianza (menos sobreajuste que un solo árbol).
 - Maneja bien datos ruidosos y características irrelevantes.

- **Ventajas**

- Buen rendimiento “out of the box”.
- Menos sobreajuste que un único árbol.
- Proporciona medidas de **importancia de variables**.

- **Desventajas**

- Menos interpretable que un árbol individual.
- Modelo más pesado y más lento en predicción que un solo árbol.
- Muchos hiperparámetros posibles (número de árboles, profundidad, etc.).

Random Forests – Hiperparámetros clave

- `n_estimators`: número de árboles.
- Profundidad máxima de cada árbol.
- Número mínimo de muestras por hoja o por división.
- Número de características consideradas en cada división.

Support Vector Machines (SVM) – Intuición

- Problema típico: **clasificación binaria**.
- Objetivo: encontrar un **hiperplano** que separe las clases con el **mayor margen posible**.
- Los puntos más cercanos al hiperplano se llaman **vectores de soporte**:
 - Son los puntos “críticos” que definen la frontera.
- Intuición: entre todas las fronteras que separan las clases, SVM elige la que ofrece más “colchón” (margen).

- En datos perfectamente separables:
 - Se busca maximizar la distancia entre las clases y el hiperplano.
- En datos reales (con ruido):
 - Se permite cierta cantidad de errores controlados por el parámetro C .
 - C alto \rightarrow menos errores, margen más ajustado (riesgo de sobreajuste).
 - C bajo \rightarrow más errores permitidos, margen más ancho (mejor generalización).

- Cuando los datos **no son linealmente separables** en el espacio original:
 - Se usan **kernels** para proyectarlos implícitamente a un espacio de mayor dimensión.
- Kernels comunes:
 - Lineal
 - Polinomial
 - RBF (Radial Basis Function / Gaussiano)
- El kernel RBF es muy utilizado por su capacidad de modelar fronteras complejas.

- **Ventajas**

- Buen rendimiento en problemas con pocas muestras y alta dimensión.
- Puede aprender fronteras de decisión muy complejas (con kernels).
- Basado en un principio teórico sólido (maximización del margen).

- **Desventajas**

- Entrenamiento costoso con conjuntos de datos muy grandes.
- Menos interpretable que modelos lineales o árboles.
- Requiere cuidado al elegir kernel, C y parámetros como γ .

- **Regresión lineal/polinomial**

- Cuando importa la interpretabilidad y las relaciones son relativamente simples.

- **KNN**

- Bueno como modelo base; sencillo, pero sufre con muchos datos/dimensiones.

- **Árboles de decisión**

- Interpretables, capturan no linealidades, pero sobreajustan fácilmente.

- **Random Forests**

- Buen rendimiento general; robustos y relativamente fáciles de usar.

- **SVM**

- Potentes en alta dimensión y con datos no lineales (con kernel), pero más costosos.

