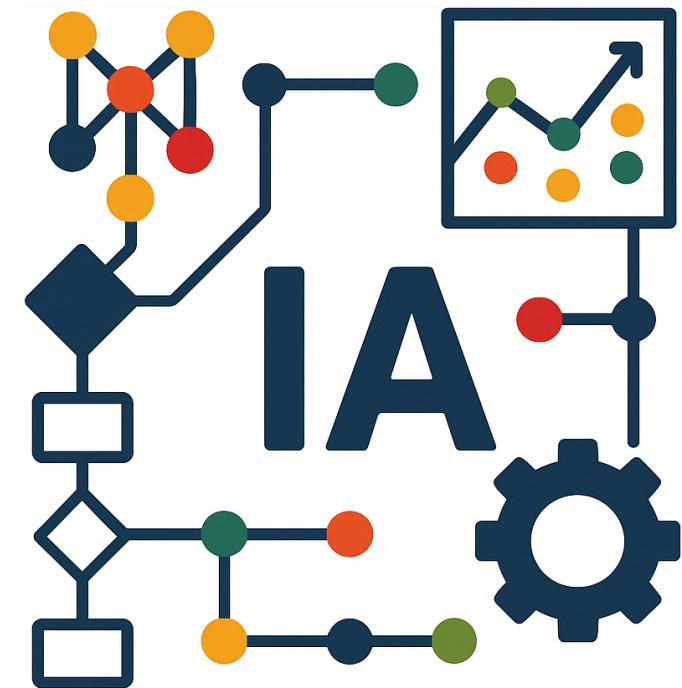


Inteligencia Artificial

Departamento de Computación - UNRC



¿Qué es la Inteligencia?

Autor/es	Año	Enfoque	Definición (resumen)
David Wechsler	1958	Psicología clásica	Capacidad global para actuar con propósito, pensar racionalmente y enfrentarse al entorno.
Howard Gardner	1983	Inteligencias múltiples	Inteligencias distintas para resolver problemas o crear productos valiosos culturalmente.
Robert Sternberg	1985	Psicología cognitiva	Inteligencia como adaptación, selección o modificación del entorno.
Raymond Cattell	1963	Psicometría	Inteligencia fluida (resolver nuevos problemas) vs cristalizada (conocimiento adquirido).
Newell & Simon	1972	Ciencias cognitivas	Sistema simbólico físico como base de la inteligencia general.
Jung & Haier	2007	Neurociencia	Hipótesis P-FIT: eficiencia en conexiones entre áreas frontales y parietales predice inteligencia.
Deary, Penke & Johnson	2010	Neurociencia	La inteligencia tiene una base biológica visible en la estructura y función cerebral.
John Searle	1980	Filosofía de la mente	Estados mentales tienen intencionalidad: están dirigidos a objetos o estados de cosas.
Daniel Dennett	1991	Filosofía / Ciencia Cognitiva	La inteligencia emerge de procesos mentales distribuidos sin centro de control.

¿Qué es la Inteligencia?

nature

Explore content ▾ About the journal ▾ Publish with us ▾ Subscribe

[nature](#) > [news](#) > [article](#)

NEWS | 12 October 2022

Neurons in a dish learn to play Pong – what's next?

Cellular version of computer game challenges assumptions about intelligence.

By [Heidi Ledford](#)

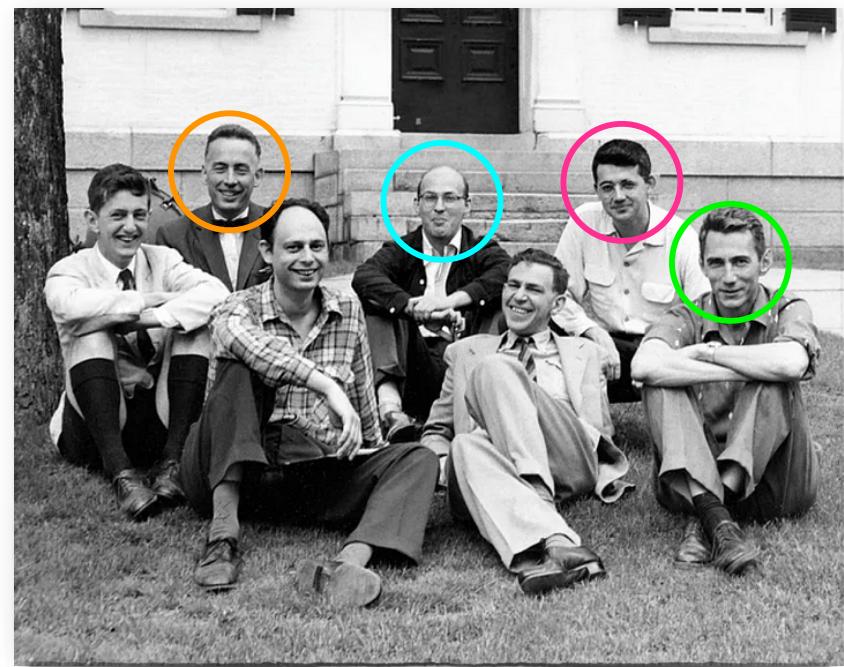
Hundreds of thousands of human neurons growing in a dish coated with electrodes have been taught to play a version of the classic computer game *Pong*¹.

Inteligencia Artificial

A PROPOSAL FOR THE
DARTMOUTH SUMMER RESEARCH PROJECT
ON ARTIFICIAL INTELLIGENCE

- J. McCarthy, Dartmouth College
- M. L. Minsky, Harvard University
- N. Rochester, I.B.M. Corporation
- C.E. Shannon, Bell Telephone Laboratories

August 31, 1955

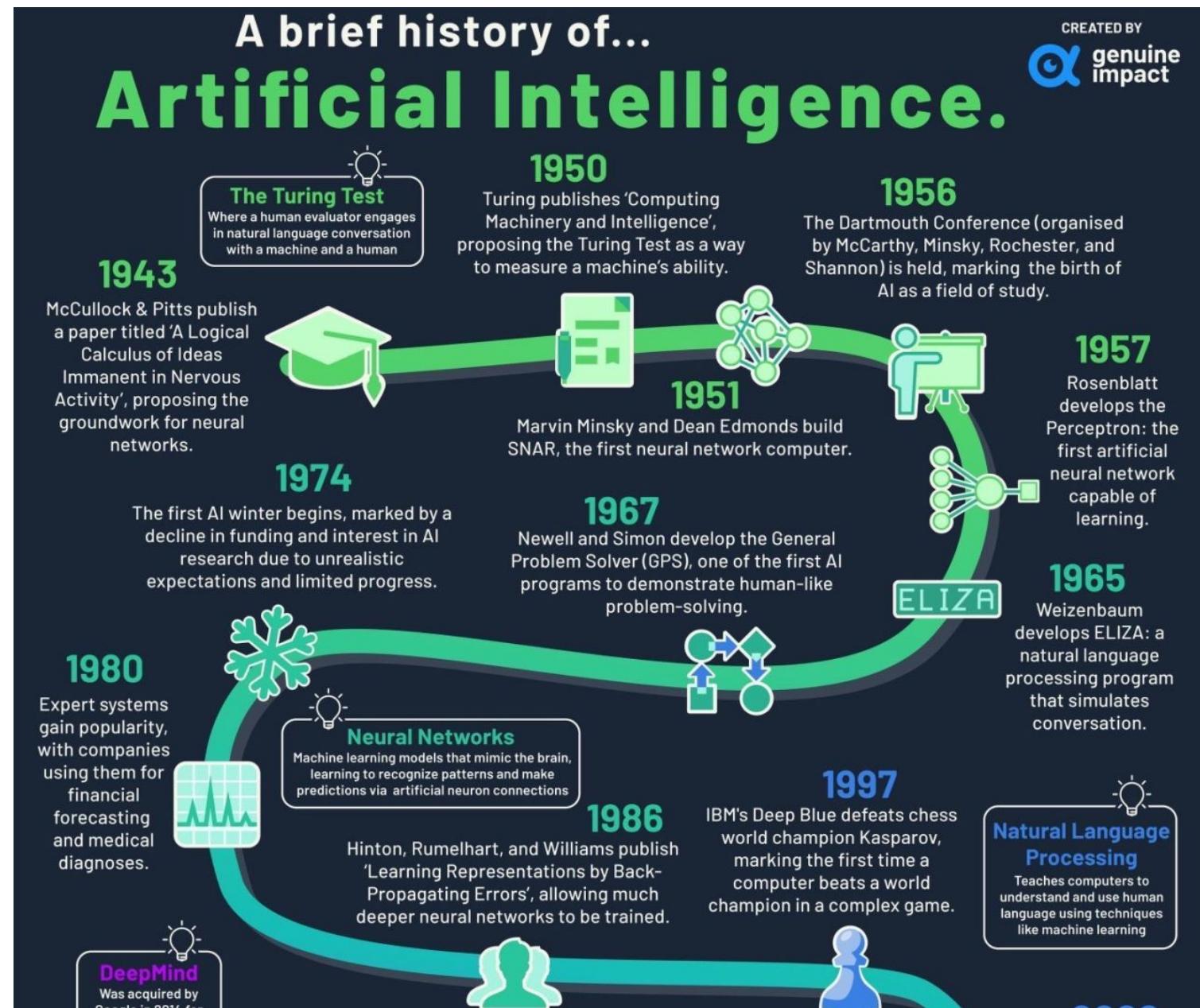


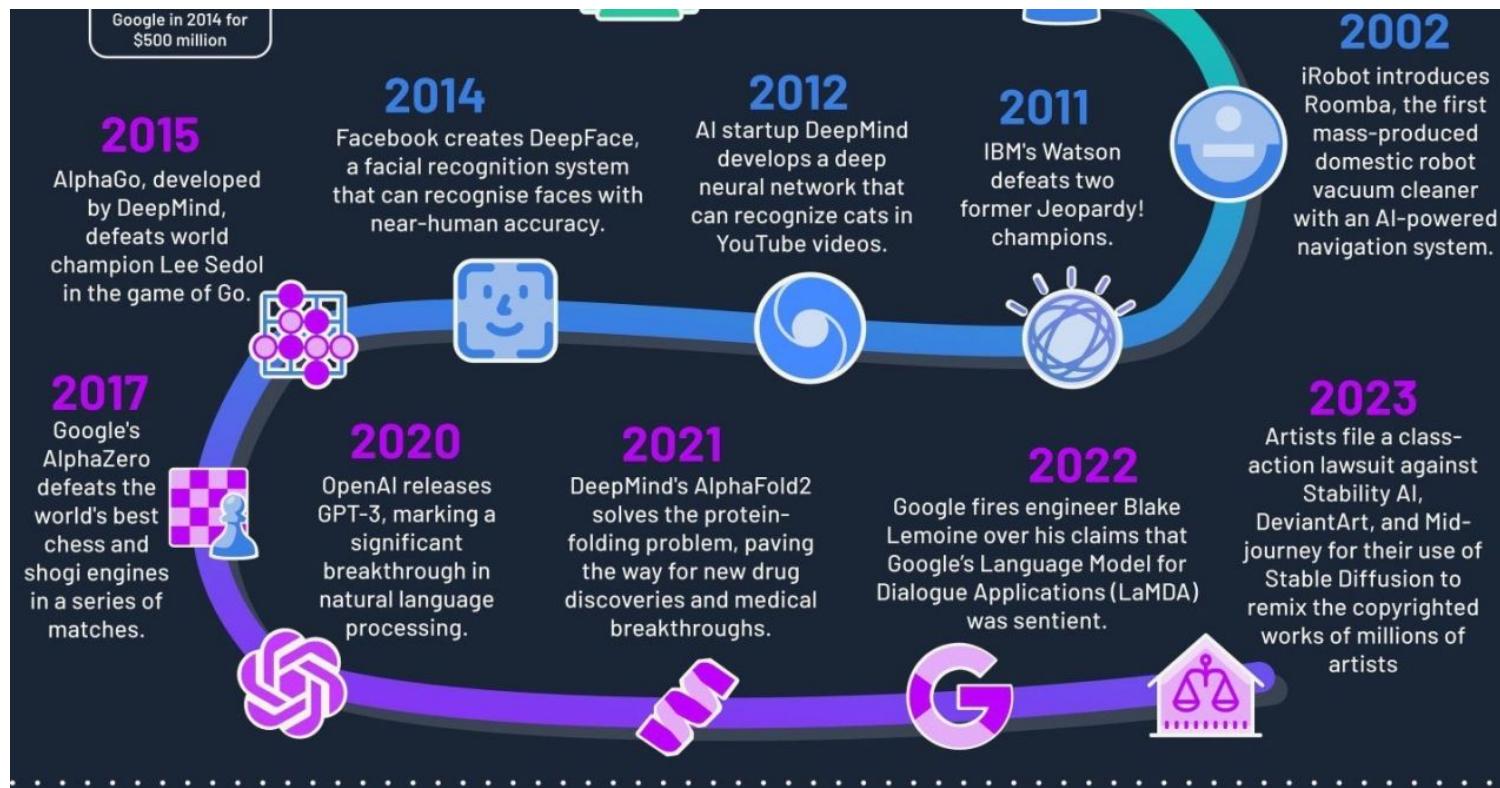
“...every aspect of learning or any other feature of intelligence can in principle be so precisely described that a machine can be made to simulate it. An attempt will be made to find how to make machines use language, form abstractions and concepts, solve kinds of problems now reserved for humans, and improve themselves. We think that a significant advance can be made in one or more of these problems if a carefully selected group of scientists work on it together for a summer.”

— McCarthy et al., Proposal for the Dartmouth Summer Research Project on Artificial Intelligence (1955)

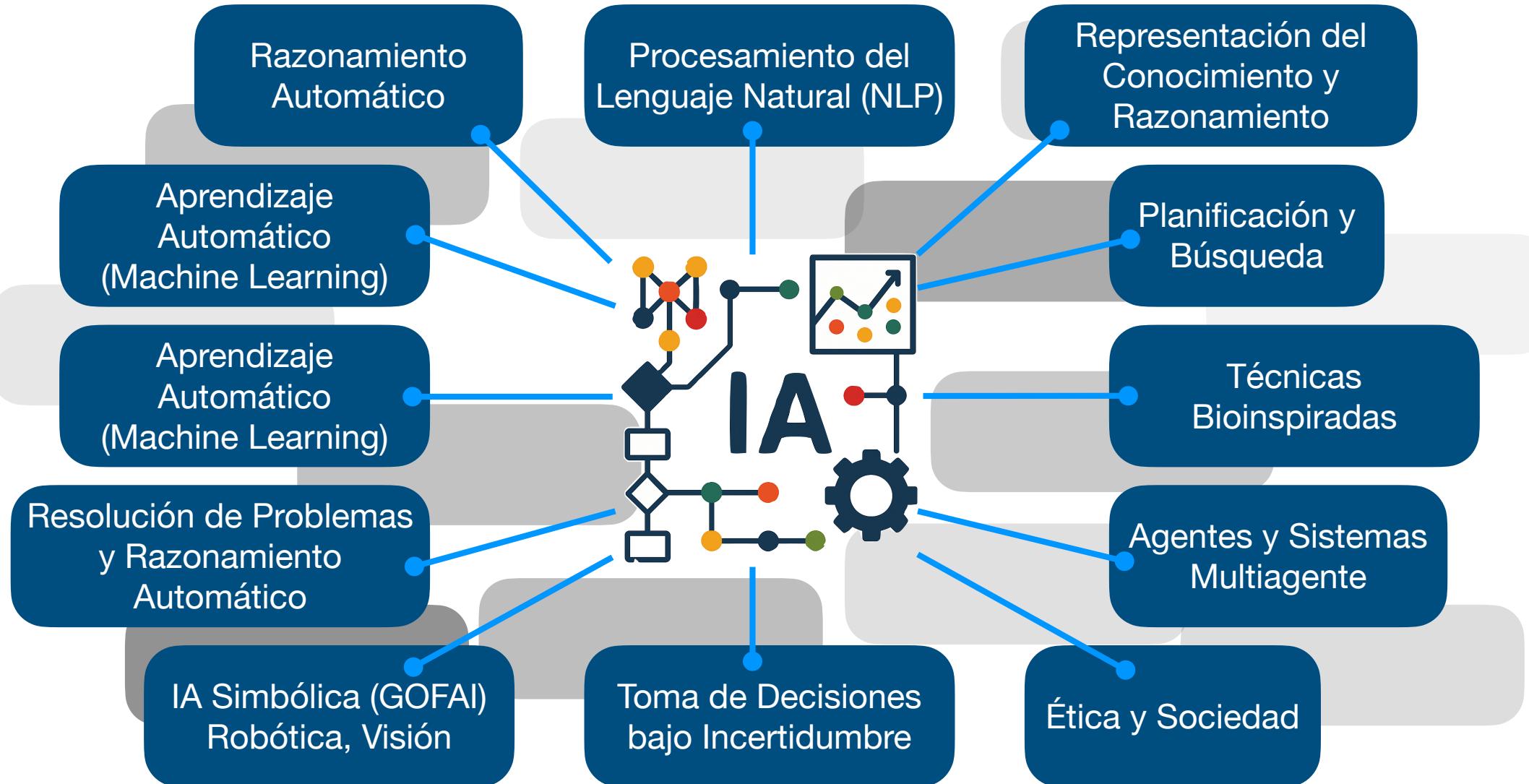
A brief history of... Artificial Intelligence.

CREATED BY
 genuine
impact

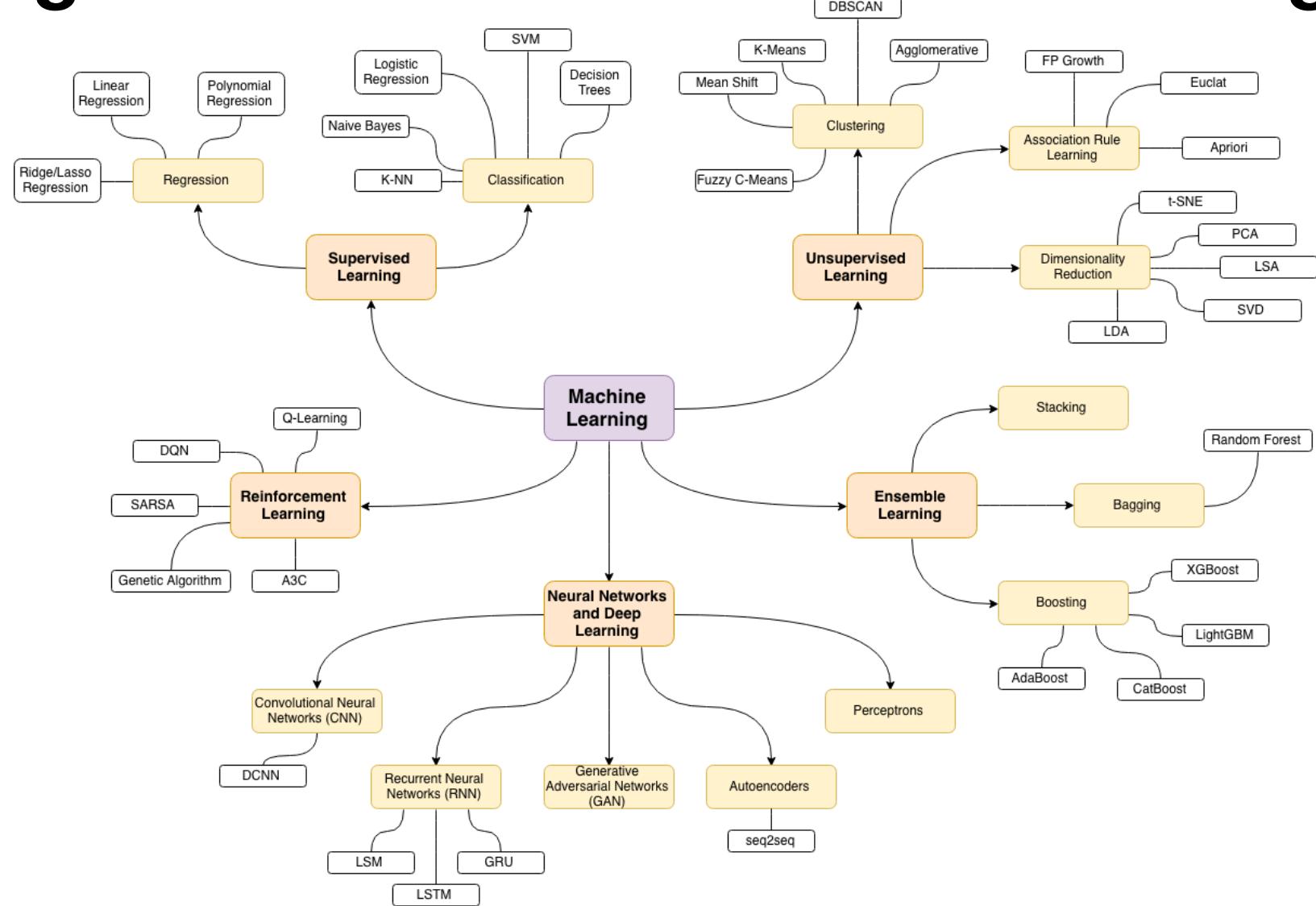




Inteligencia Artificial - Diferentes áreas



Inteligencia Artificial - Machine Learning

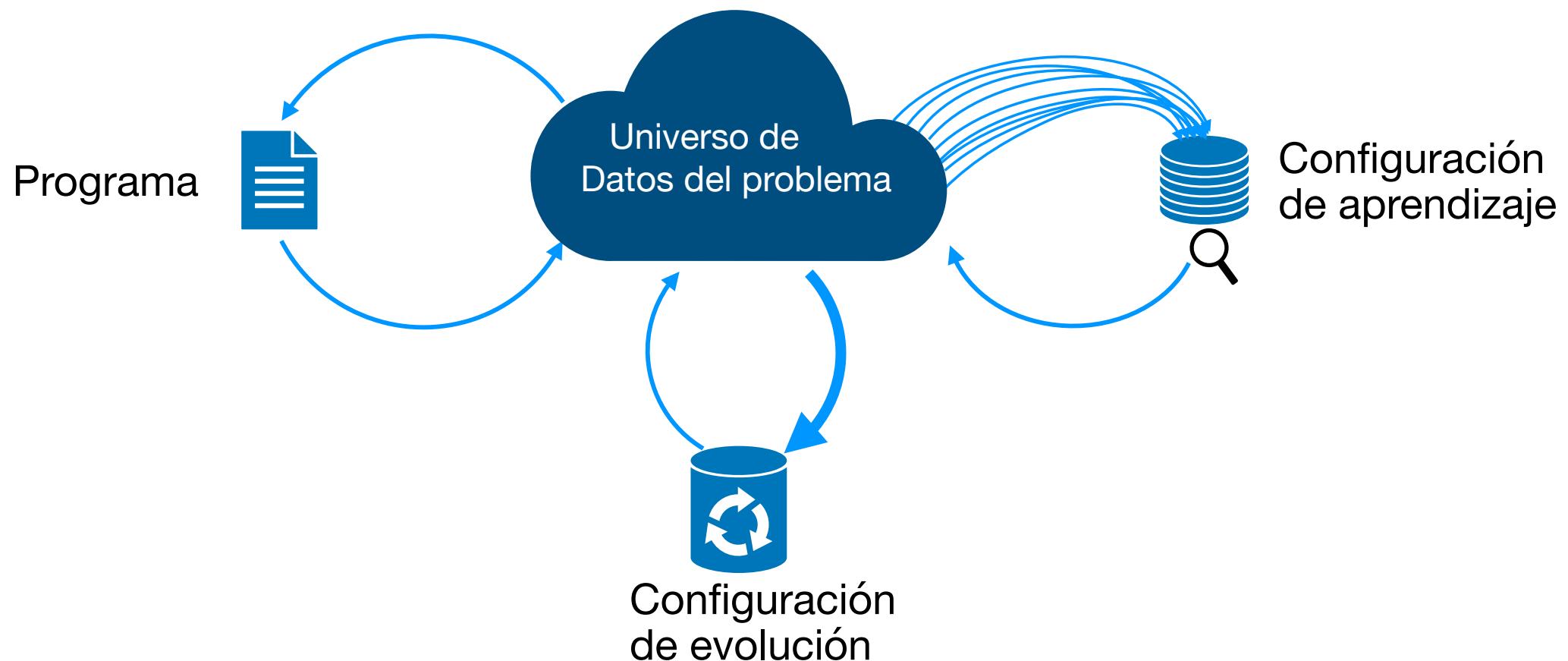


Inteligencia Artificial - Machine Learning



Solución de Problemas con:

Programación “tradicional” - Machine Learning - Algoritmos Genéticos



Solución de Problemas con:

Programación “tradicional” - Machine Learning - Algoritmos Genéticos

Característica	Programación Tradicional	Machine Learning	Algoritmos Genéticos
Paradigma	Instrucciones explícitas	Aprendizaje a partir de datos	Optimización evolutiva
Entrada	Reglas + Datos	Datos (con etiquetas o no)	Población inicial de soluciones
Salida	Resultados (según reglas definidas)	Modelo aprendido (predictor, clasificador)	Solución óptima o aproximada
Tipo de solución	Determinística	Probabilística / Estadística	Estocástica / Heurística
Proceso de desarrollo	El programador define la lógica paso a paso	El sistema aprende la lógica a partir de ejemplos	El sistema evoluciona soluciones mediante operadores genéticos
Capacidad de adaptación	Baja (requiere reprogramar si cambian los datos)	Alta (puede generalizar a nuevos datos)	Alta (explora múltiples soluciones y se adapta)
Ámbito de aplicación	Problemas bien definidos con reglas claras	Problemas con patrones ocultos y grandes datos	Problemas de búsqueda y optimización complejos

Agenda

Algoritmos Genéticos

Machine Learning

Modelos de Lenguajes

Inteligencia Artificial

Algoritmos Genéticos



Algoritmos Genéticos

Los **algoritmos genéticos** son una familia de algoritmos de búsqueda inspirados en los **principios de la evolución** en la naturaleza. Al imitar el proceso de selección natural y reproducción, los algoritmos genéticos pueden generar soluciones de alta calidad para diversos problemas que involucran búsqueda, optimización y aprendizaje.

Los algoritmos genéticos, desarrollados por John Holland (University of Michigan, 1970's), utilizan un mecanismo de búsqueda informada.

Algoritmos Genéticos

Los principios de la teoría de la evolución darwiniana pueden resumirse mediante los siguientes conceptos:

Principio de variación: Los rasgos o atributos de los individuos que forman parte de una población pueden variar. Como resultado, los individuos difieren entre sí.

Principio de herencia: Algunos rasgos se transmiten de manera consistente de los individuos a su descendencia. Como consecuencia, la descendencia se parece más a sus progenitores que a otros individuos no relacionados.

Principio de selección: Las poblaciones suelen competir por los recursos en su entorno. Los individuos *que poseen rasgos mejor adaptados* al entorno tendrán más probabilidades de sobrevivir y, además, contribuirán con un mayor número de descendientes a la siguiente generación.

Algoritmos Genéticos

Los algoritmos genéticos son algoritmos de **búsqueda informada**, y como tales se aplican a problemas que pueden expresarse como una búsqueda de **configuraciones exitosas** a partir de ciertas **configuraciones iniciales**, mediante la aplicación de **reglas predefinidas de reconfiguración o avance**.

En los algoritmos genéticos, una **población de soluciones candidatas evoluciona de forma iterativa**: las mejores soluciones tienen más probabilidades de ser seleccionadas y transmitir sus características a la siguiente generación, logrando así **mejorar progresivamente la calidad de las soluciones al problema**.

Algoritmos Genéticos

Genotipo

Población

Evaluación (Fitness)

Evolución

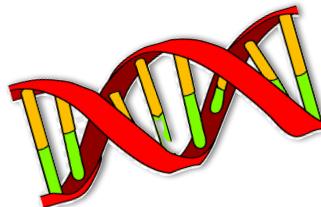
Selección

Mutación

Cruzamiento

Algoritmos Genéticos - Genotipo

En la naturaleza, la reproducción, el cruce y la mutación se realizan a través del **genotipo**, que es una colección de **genes** agrupados en **cromosomas**. Cuando dos individuos se reproducen para crear descendencia, cada cromosoma de la descendencia contiene una combinación de genes de ambos progenitores.



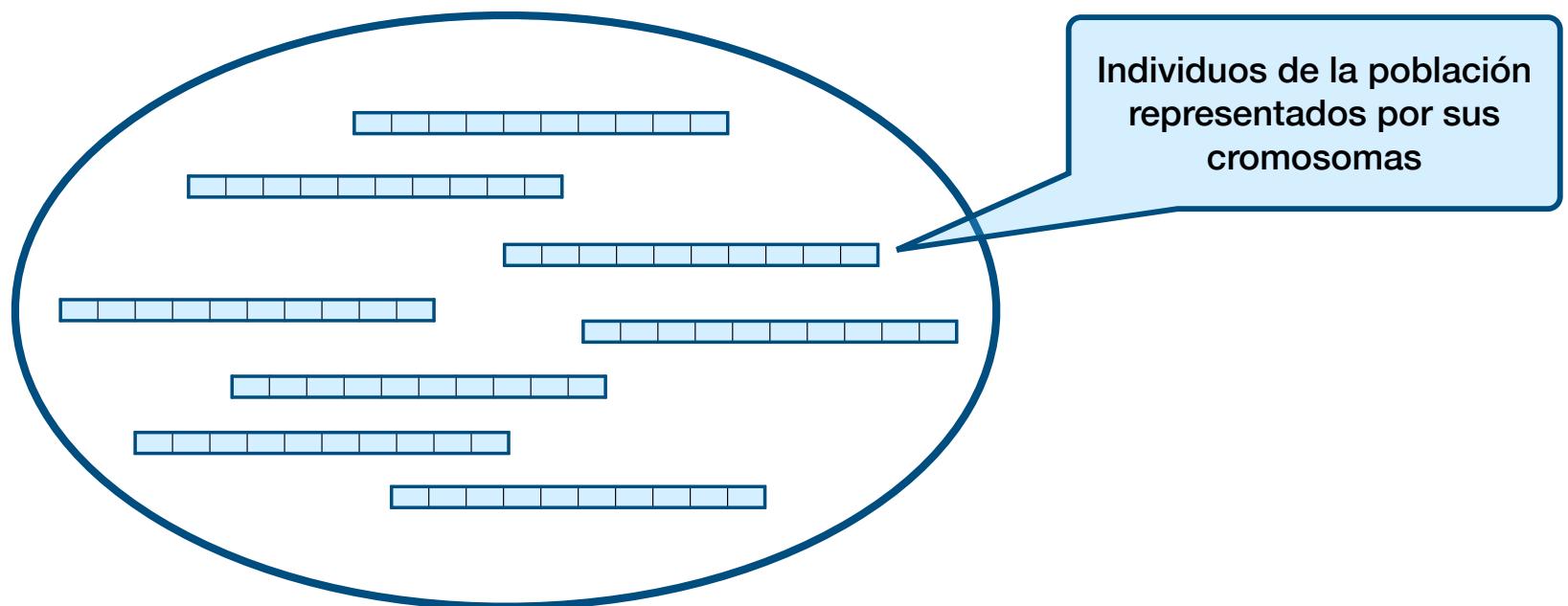
Imitando este concepto, en los algoritmos genéticos cada individuo se representa mediante un **cromosoma que agrupa una colección de genes**. Por ejemplo, un cromosoma puede expresarse como una cadena binaria, donde cada bit representa un gen.

0	1	1	0	1	1	0	1	0	1
---	---	---	---	---	---	---	---	---	---

A los posibles valores que puede tomar un **Gen** se lo denomina **ALELO**

Algoritmos Genéticos - Población

Los algoritmos genéticos mantienen una **población de individuos**, es decir, un *conjunto de soluciones candidatas para el problema*. Como cada individuo se representa mediante un cromosoma, la población puede verse como una **colección de cromosomas**.



Algoritmos Genéticos - Evaluación (Fitness)

Los individuos se **evalúan mediante una función de aptitud** (también llamada función objetivo), que es la función que buscamos optimizar o el problema que intentamos resolver.

Los individuos con una **mejor puntuación** de aptitud representan mejores soluciones y **tienen más probabilidades de ser seleccionados** para cruzarse (combinarse) y **formar parte de la siguiente generación**. Con el tiempo (iteraciones), la calidad de las soluciones mejora, los valores de aptitud aumentan y **el proceso puede detenerse cuando se encuentra una solución con un valor de aptitud satisfactorio**.

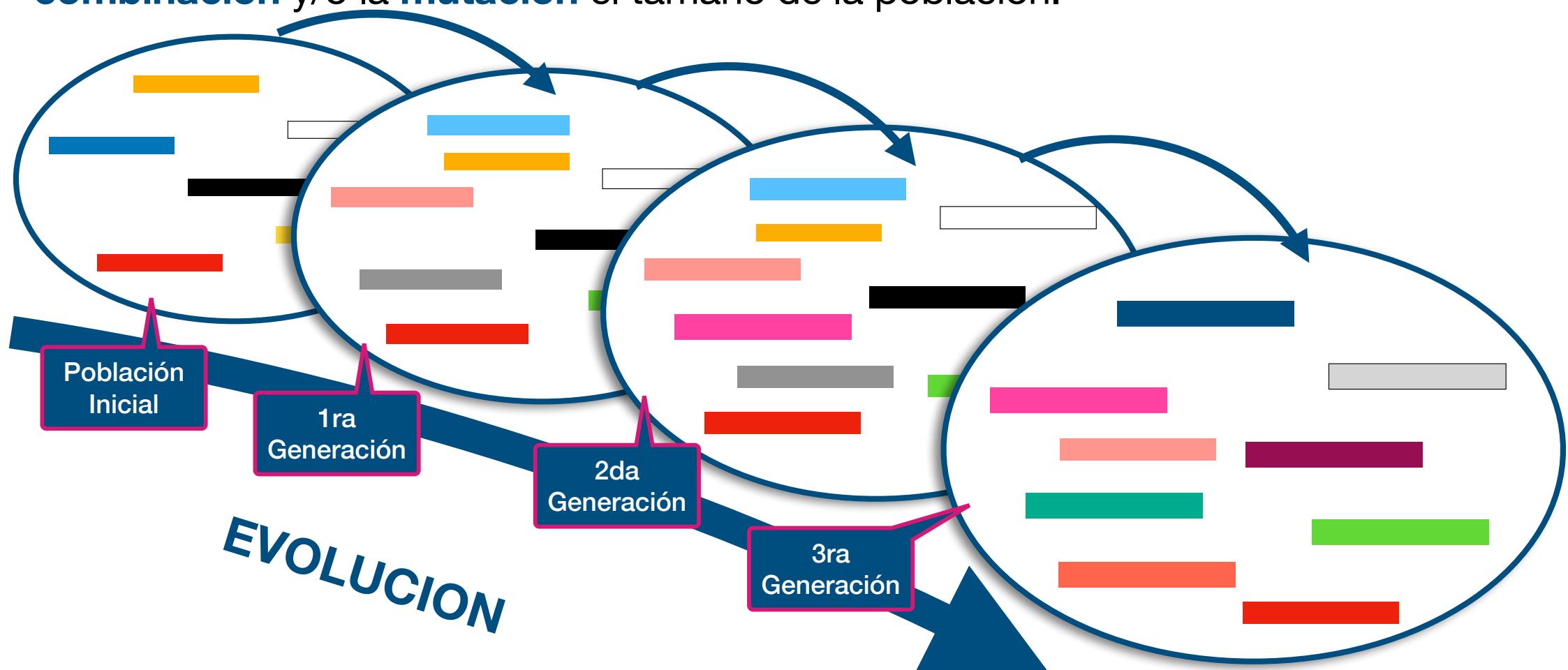
La función de Fitness debe calificar a un individuo (solución candidata) en términos de del objetivo (solución)



Es deseable que la función sea lo menos ambigua posible, es decir, que permita distinguir a cada individuo de la manera más clara y precisa posible

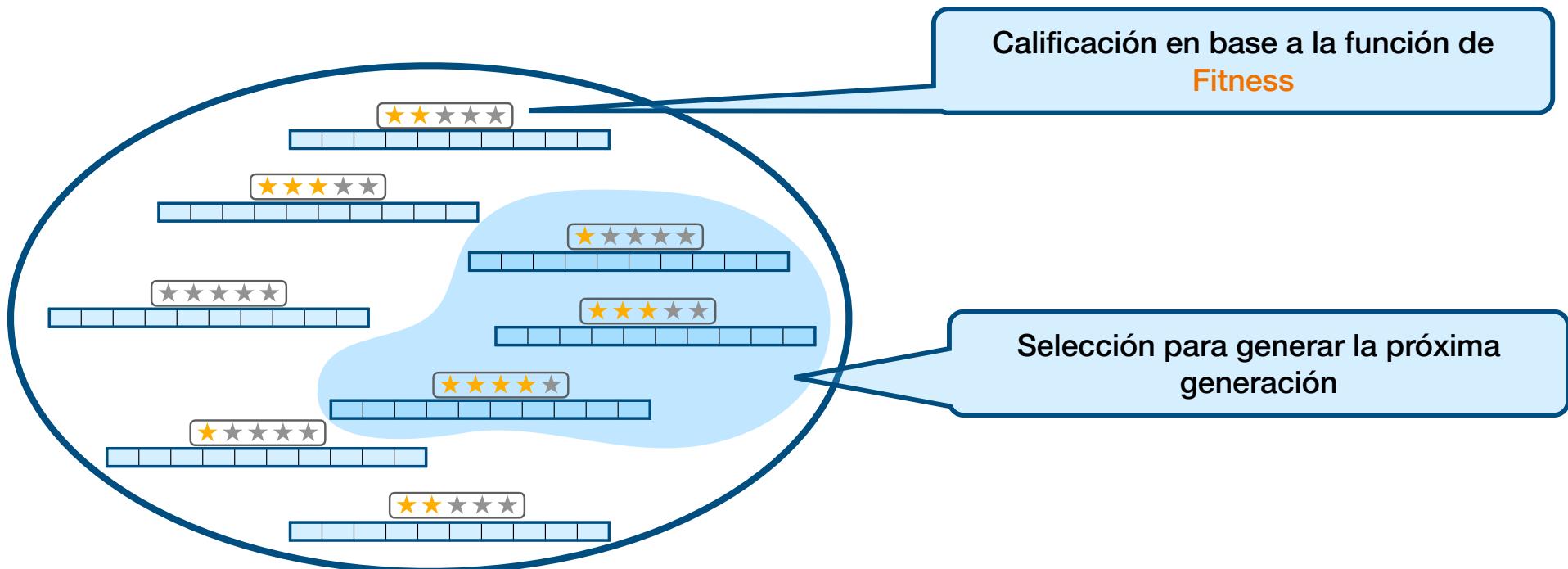
Algoritmos Genéticos - Evolución

Para hacer evolucionar una población, se **eligen** individuos para realizar la **combinación** y/o la **mutación** el tamaño de la población.



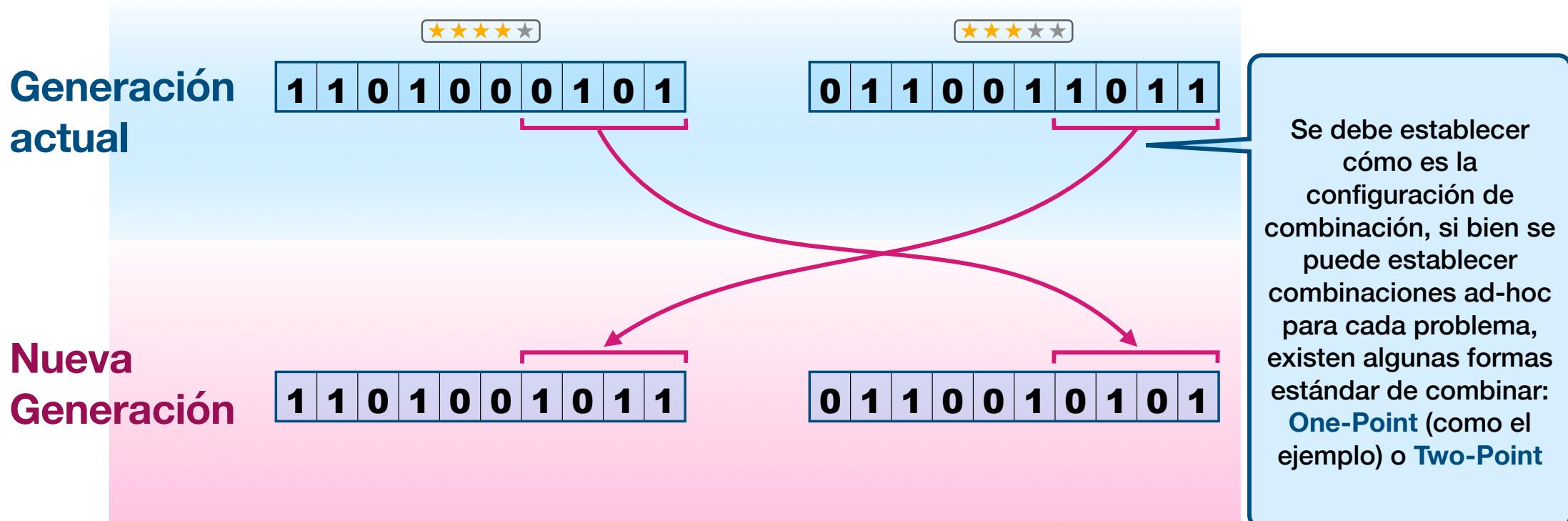
Algoritmos Genéticos - Selección

Luego de evaluar la aptitud de cada individuo, se **seleccionan aquellos con mayor puntuación para combinarse y producir la siguiente generación**. Los *individuos con baja aptitud también pueden ser elegidos, aunque con menor probabilidad*, para no excluir completamente su material genético.



Algoritmos Genéticos - Combinación (crossover)

Para crear una nueva pareja de individuos, normalmente se eligen dos padres de la generación actual y se intercambian partes de sus cromosomas (combinación) para generar dos nuevos cromosomas que representan a la **descendencia**. Esta operación se denomina **crossover**.

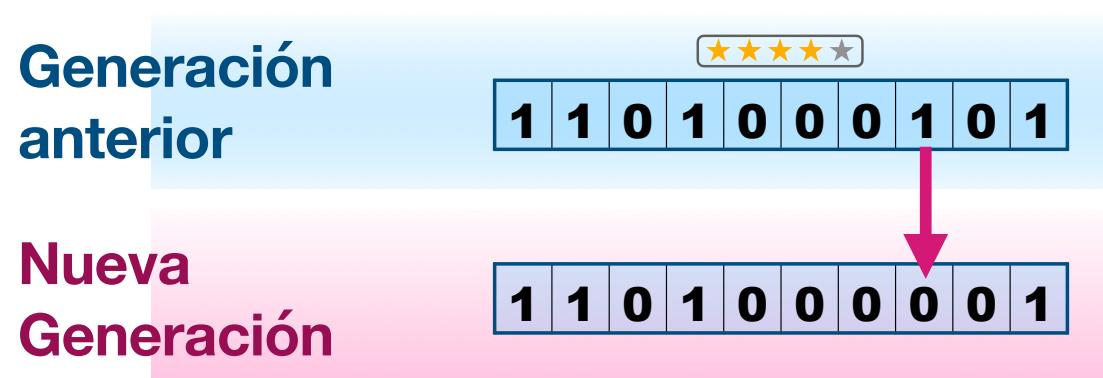


Algoritmos Genéticos - Mutación

La mutación es uno de los ingredientes más interesantes de los algoritmos genéticos. Básicamente, una **mutación produce un cambio imprevisto** en algún individuo de la población.

La **mutación ayuda a mantener la diversidad en la población**, y en muchos casos a recuperar información perdida en la evolución de la población.

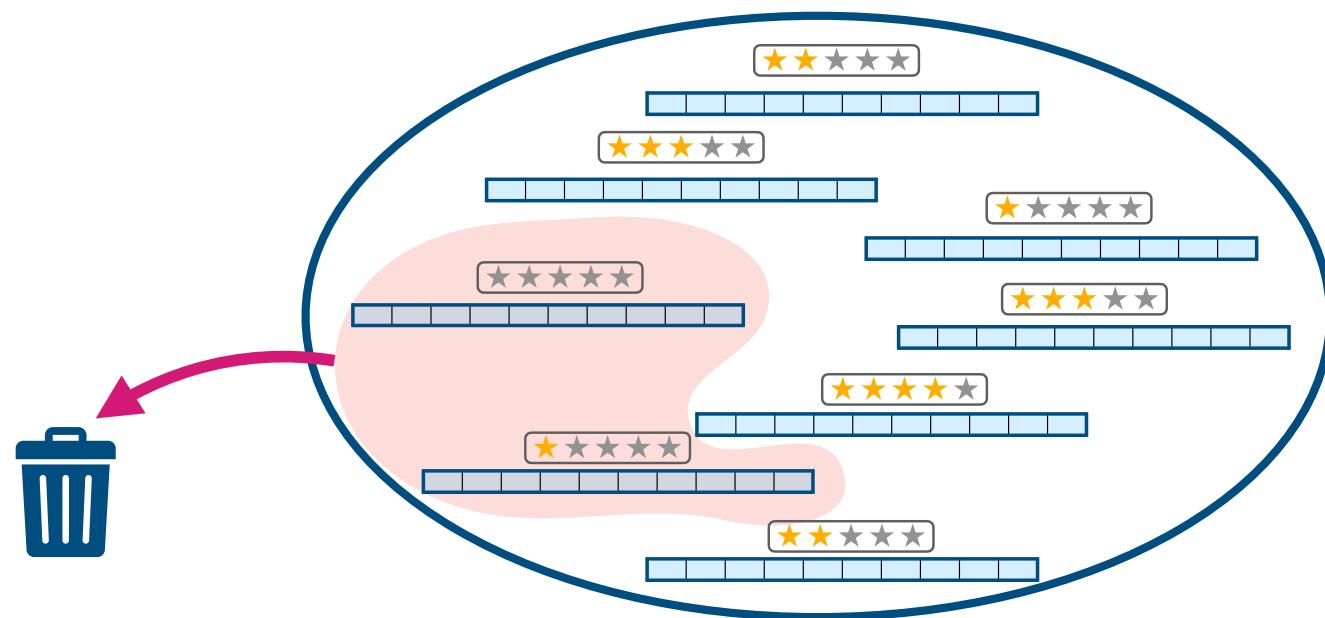
Una de las formas más comunes de mutación es el **cambio en el valor de un gen de un cromosoma**, por otro de sus **alelos**.



Algoritmos Genéticos - Eliminación

Evaluar la aptitud de una población depende, evidentemente, del **tamaño** de la misma. Es por esto esencial **mantener acotado el tamaño de las poblaciones**.

Por esta razón, durante la evolución se deben **eliminar individuos** de la población. Los individuos **elegidos para eliminar son aquellos “menos aptos”** (de acuerdo a la función de fitness), basados claramente en la idea de que los individuos más aptos tienen más chances de sobrevivir (selección natural)



Algoritmos Genéticos vs Algoritmos de Búsqueda

Existen varias diferencias importantes entre los algoritmos genéticos y los algoritmos de búsqueda u optimización tradicionales, como los basados en gradientes.

Las características clave que distinguen a los algoritmos genéticos son:

- Mantener una **población de soluciones**.
- Usar una **representación genética de las soluciones**.
- Emplear una **función de aptitud para evaluar** los resultados.
- Presentar un **comportamiento probabilístico**.

Algoritmos Genéticos - Ventajas

Las características únicas de los algoritmos genéticos les otorgan varias ventajas frente a los algoritmos de búsqueda tradicionales.

Las principales ventajas son:

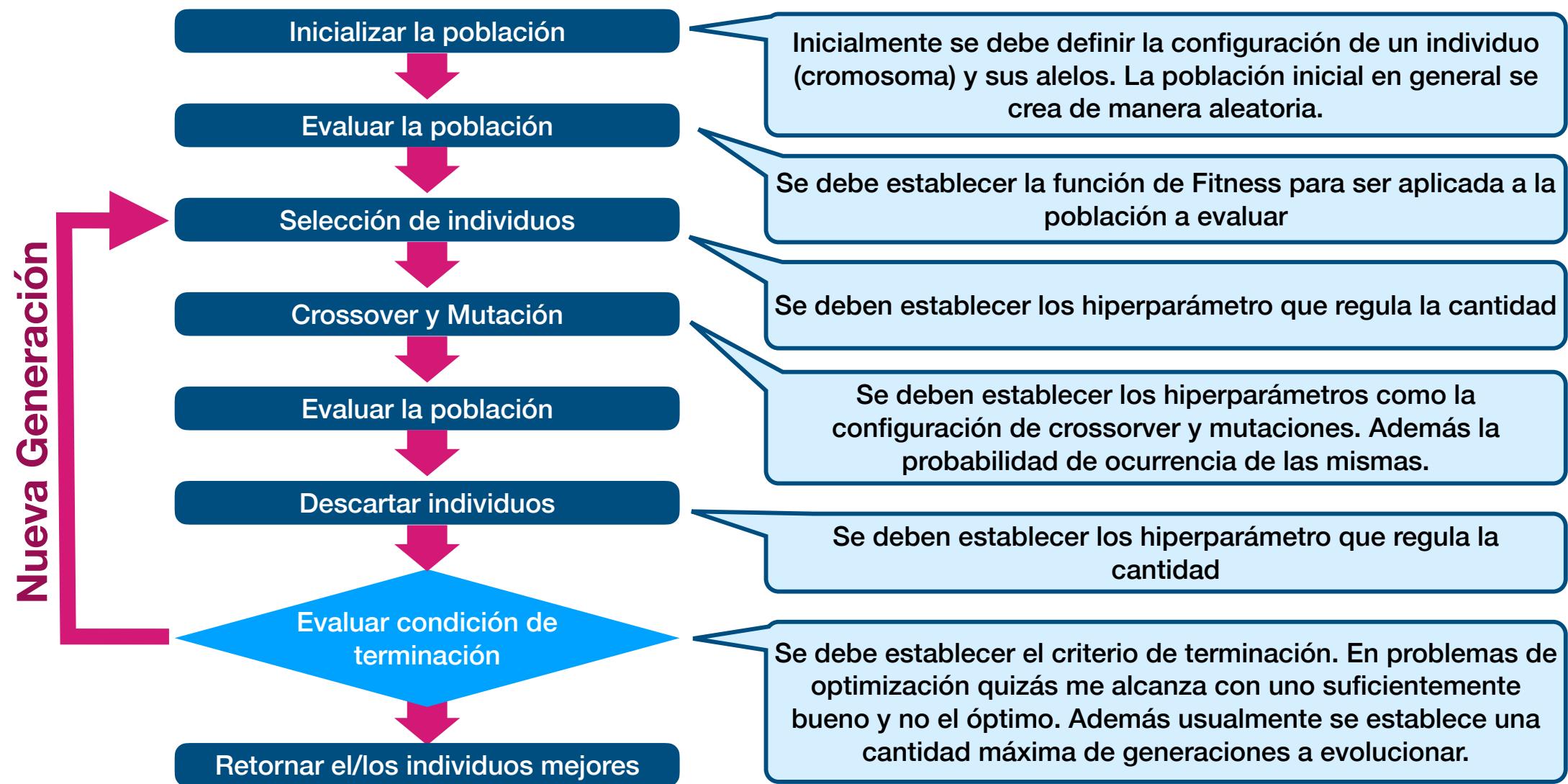
- Capacidad de **optimización global**.
- Manejo de problemas con representaciones matemáticas complejas.
- **Resiliencia al ruido**.
- Compatibilidad con el **paralelismo y el procesamiento distribuido**.
- Adecuación para el **aprendizaje continuo**.

Algoritmos Genéticos - Limitaciones

Las principales limitaciones de los algoritmos genéticos son:

- Necesidad de **definiciones específicas** del problema.
- Requieren ajuste de **hiperparámetros**.
- Operaciones **computacionalmente intensivas**.
- Riesgo de **convergencia prematura**.
- **No garantizan encontrar una solución óptima.**

Estructura de un algoritmo genético



Estructura de un algoritmo genético - Selección

La selección se realiza al inicio de cada ciclo del algoritmo genético para elegir los individuos que serán los padres de la siguiente generación. Este proceso es probabilístico y la probabilidad de ser elegido depende del valor de aptitud, favoreciendo a los individuos con mayor puntaje.

Los métodos de selección más usados son:

- En la **selección por ruleta (FPS)**, cada individuo tiene una probabilidad de ser elegido proporcional a su aptitud, como si ocupara una porción de una ruleta cuyo tamaño depende de su valor de aptitud.
- En el **muestreo universal estocástico (SUS)**, se hace un solo giro de la ruleta y se seleccionan varios individuos a la vez usando puntos de selección equidistantes.
- El **escalado de aptitud** transforma los valores originales de aptitud mediante una función lineal $\text{fitness escalada} = a * \text{fitness} + b$ para ajustarlos a un rango deseado.
- En la **selección por torneo**, se eligen al azar dos o más individuos de la población, y el que tenga la mayor aptitud es el que resulta seleccionado.

Estructura de un algoritmo genético - Combinación

El operador de *crossover* combina la información genética de dos padres para generar descendientes, aplicándose con alta probabilidad. Si no se aplica, los padres se clonian en la siguiente generación.

Los métodos de *crossover* más usados son:

- En la **combinación de un punto**, se intercambian los genes a la derecha de un punto seleccionado al azar entre dos padres para crear dos descendientes con información genética combinada.
- En la **combinación de dos o más (k) puntos**, se intercambian los genes situados entre dos o más puntos aleatorios seleccionados en los cromosomas de ambos padres.
- En la **combinación uniforme**, cada gen del descendiente se elige de forma independiente, seleccionando al azar uno de los padres. Cuando la probabilidad es del 50%, ambos padres tienen la misma posibilidad de aportar cada gen.

Estructura de un algoritmo genético - Mutación

La mutación es el último operador genético que se aplica al crear una nueva generación. Se aplica a los descendientes resultantes de la selección y la combinación, y **ocurre con una probabilidad generalmente muy baja**. En algunas versiones del algoritmo genético, la probabilidad de mutación aumenta gradualmente con las generaciones para evitar la estancación y mantener la diversidad de la población. Sin embargo, si la tasa de mutación es demasiado alta, el algoritmo se vuelve similar a una búsqueda aleatoria.

Los métodos de *mutación* más usados son:

- En la mutación por **inversión de gen**, se selecciona y cambia un gen al azar
- En la **mutación por intercambio**, se seleccionan al azar dos genes y se intercambian sus valores
- En la **mutación por inversión**, se selecciona una secuencia aleatoria de genes en cromosomas y se invierte el orden de los genes en esa secuencia
- En la **mutación por reorganización**, una subsecuencia de genes se selecciona y se reordena aleatoriamente.

Estructura de un algoritmo genético - Elitismo

El **elitismo** es una estrategia opcional en los algoritmos genéticos que garantiza que los mejores individuos de una generación pasen directamente a la siguiente.

Consiste en **copiar los n mejores individuos** (según una cantidad predefinida) antes de llenar el resto de la población con descendientes creados mediante selección, cruce y mutación.

Estos individuos élite también pueden participar como padres. Esta técnica evita la pérdida de buenas soluciones y puede mejorar significativamente el rendimiento del algoritmo.

Estructura de un algoritmo genético - Implementación en Python (MaxONES)

```
import random

# -----
# Parámetros
# -----
POP_SIZE = 20      # Tamaño de la población
GENS = 30          # Número de generaciones
CXPB = 0.8         # Probabilidad de cruce
MUTPB = 0.05       # Probabilidad de mutación
GENOME_LENGTH = 20 # Longitud de cada individuo (número de bits)

# -----
# Función de fitness
# -----
def fitness(individual):
    return sum(individual) # Cuenta la cantidad de 1s

# -----
# Crear población
# -----
def create_individual():
    return [random.randint(0, 1) for _ in range(GENOME_LENGTH)]

def create_population():
    return [create_individual() for _ in range(POP_SIZE)]
```

Estructura de un algoritmo genético - Implementación en Python (MaxONES)

```
# -----
# Operadores genéticos
# -----
def selection(population):
    # Selección por torneo
    k = 3
    selected = []
    for _ in range(POP_SIZE):
        aspirants = random.sample(population, k)
        winner = max(aspirants, key=fitness)
        selected.append(winner)
    return selected

def crossover(p1, p2):
    # Combinación de un punto
    if random.random() < CXPB:
        point = random.randint(1, GENOME_LENGTH - 1)
        return p1[:point] + p2[point:], p2[:point] + p1[point:]
    return p1[:], p2[:]

def mutate(individual):
    for i in range(GENOME_LENGTH):
        if random.random() < MUTPB:
            individual[i] = 1 - individual[i] # Flip bit
    return individual
```

Estructura de un algoritmo genético - Implementación en Python (MaxONES)

```
# Algoritmo principal
# -----
def genetic_algorithm():
    population = create_population()
    for gen in range(GENS):
        # Evaluar y mostrar el mejor
        population.sort(key=fitness, reverse=True)
        print(f"Gen {gen}: Mejor = {population[0]} Fitness = {fitness(population[0])}")

        # Selección
        selected = selection(population)

        # Reproducción
        next_gen = []
        for i in range(0, POP_SIZE, 2):
            offspring1, offspring2 = crossover(selected[i], selected[i+1])
            next_gen.append(mutate(offspring1))
            next_gen.append(mutate(offspring2))

        population = next_gen
    return max(population, key=fitness)

# -----
# Ejecutar
# -----
best = genetic_algorithm()
print(f"Mejor individuo encontrado: {best}, Fitness = {fitness(best)}")
```

Estructura de un algoritmo genético - Implementación en Python (DEAP)

DEAP es una librería de Python diseñada para crear algoritmos genéticos y otras meta-heurísticas de forma modular, flexible y eficiente.

DEAP se organiza en cuatro módulos principales:

- **Creator**: permite crear clases personalizadas para individuos y fitness
- **Base**: contiene la caja de herramientas (Toolbox) donde registras funciones
- **Tools**: incluye operadores genéticos y funciones utilitarias, como: **cruce o combinación** (uno y dos puntos, etc.), **mutación** (oneFlip, swap, gaussiana, etc.) y **selección** (torneo, etc), entre otras
- **Algorithms**: proporciona implementaciones listas de algoritmos evolutivos

Estructura de un algoritmo genético - Implementación en Python (DEAP)

Flujo básico de un algoritmo con DEAP:

1. Definir clases para fitness e individuos con creator.
2. Registrar funciones para generar atributos, individuos y poblaciones con Toolbox.
3. Definir operadores genéticos (cruce, mutación, selección).
4. Definir función de evaluación (fitness).
5. Inicializar la población.
6. Evolucionar usando un algoritmo predefinido o personalizado

Estructura de un algoritmo genético - Implementación MaxOnes con DEAP

Fitness de maximización = 1.0,
(minimización = -1.0)

Establecemos un individuo que tiene una lista de genes

Definimos un gen como un booleano (0 o 1)

Registraremos que un individuo tiene 20 genes booleanos

Registraremos que la población es una lista de individuos

Definimos y registramos la función de evaluación de la Fitness

Registraremos los operadores de combinación, mutación y selección

Inicializamos la Población

Evolucionamos

```
from deap import base, creator, tools, algorithms
# 1. Crear clase de fitness y clase de individuo
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)
# 2. Toolbox: atributos, individuos, población
toolbox = base.Toolbox()
# Atributo binario: 0 o 1
toolbox.register("attr_bool", random.randint, 0, 1)
# Un individuo es una lista de 20 bits
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, 20)
# Una población es una lista de individuos
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
# 3. Función de evaluación: contar 1's
def eval_maxones(individual):
    return sum(individual),
toolbox.register("evaluate", eval_maxones)

# 4. Operadores genéticos
toolbox.register("mate", tools.cxTwoPoint)          # Cruce de 2 puntos
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05) # Mutación bit flip
toolbox.register("select", tools.selTournament, tournsize=3) # Selección torneo

# 5. Algoritmo principal
def main():
    random.seed(42)
    pop = toolbox.population(n=50) # Población inicial
    algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=20, verbose=True)
```

Algoritmos Genéticos MultiObjetivo - MOGA

Un **algoritmo genético multiobjetivo (MOGA)** es una extensión de los algoritmos genéticos tradicionales diseñada para optimizar varias funciones objetivo **simultáneamente**, que suelen estar en **conflicto entre sí**.

Ejemplo típico: minimizar el costo y maximizar la calidad de un producto.

En vez de encontrar una única solución óptima, un MOGA **busca un conjunto de soluciones** conocido como **frente de Pareto que representa diferentes compromisos entre los objetivos**.

Algoritmos Genéticos MultiObjetivo - MOGA

Diferencias con un algoritmo genético (GA) clásico:

- En un GA clásico, hay una única función de aptitud (fitness).
- En un MOGA, **hay M funciones objetivo** que deben evaluarse para cada individuo: $F(x) = (f_1(x), f_2(x), \dots, f_M(x))$.
- No siempre es posible ordenar las soluciones de mejor a peor de forma directa, ya que algunas soluciones **son no dominadas**.

MOGA - Dominancia de Pareto

Una **solución A domina a otra solución B** si:

- **A** es mejor o igual que **B** en todos los objetivos.
- **A** es estrictamente mejor que **B** en al menos un objetivo.

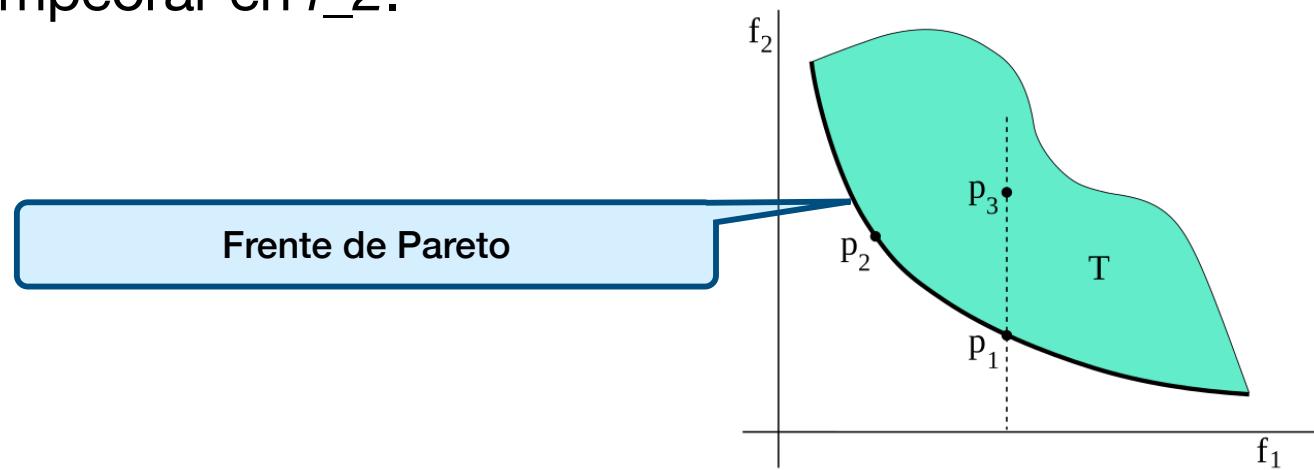
Si ninguna de las dos domina a la otra, se dice que son ***no dominadas***.

MOGA - Frente de Pareto

El **frente de Pareto** es el **conjunto de soluciones no dominadas** en el espacio de búsqueda.

En la práctica, **el objetivo de un MOGA es aproximar este frente** para ofrecer al decisor un conjunto de soluciones con diferentes compensaciones.

Ejemplo gráfico: En un problema *biobjetivo* (minimizar f_1 y f_2), el frente de Pareto se representa como una curva en el plano donde ningún punto puede mejorar en f_1 sin empeorar en f_2 .



MOGA - Frente de Pareto - Ejemplo discreto

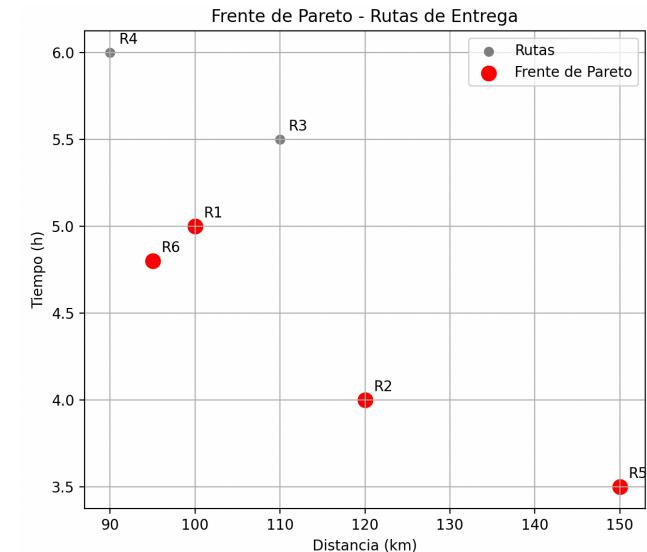
Una empresa de logística quiere optimizar la ruta de un camión que debe entregar mercancía en varias ciudades. Se analizan dos objetivos:

- Distancia total (minimizar) – asociada al costo de combustible.
- Tiempo total de entrega (minimizar) – considerando tráfico y velocidad en cada tramo.

R3 está dominada por R1 (peor en distancia y tiempo)

R4 está dominada por R6 (peor en distancia y tiempo)

Ruta	Distancia (km)	Tiempo (h)
R1	100	5.0
R2	120	4.0
R3	110	5.5
R4	90	6.0
R5	150	3.5
R6	95	4.8



El frente de Pareto está formado por R1, R2, R5, R6, porque ninguna puede mejorar un objetivo sin empeorar el otro.

MOGA - Frente de Pareto - Ejemplo continuo

Producción Industrial (Costo vs Contaminación)

- **Costo de Producción** $F_1(x) = x^2 + 2$ (El costo disminuye con un cierto valor de x , pero luego empieza a aumentar)
- **Nivel de Contaminación** $F_2(x) = 10 / (x+1)$ (La contaminación disminuye con valores altos de x , pero nunca llega a cero)

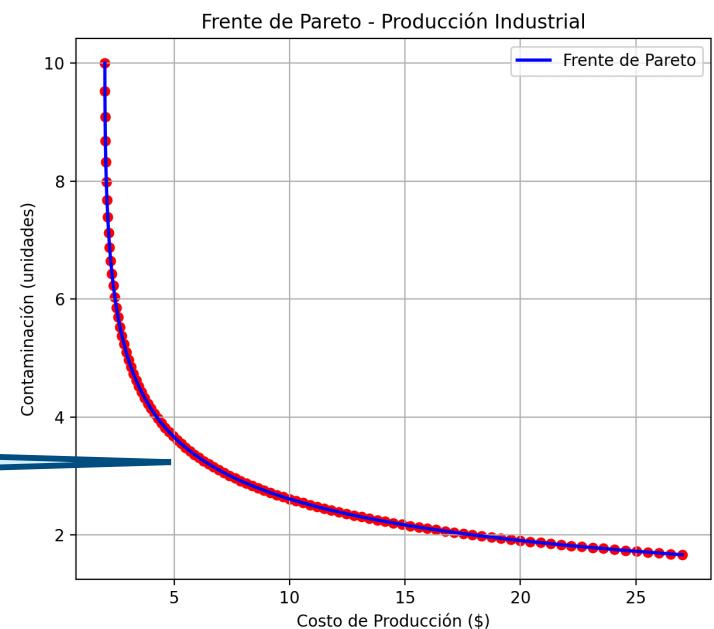
Ambos objetivos están en conflicto:

Un x pequeño implica bajo costo pero alta contaminación.

Un x grande reduce la contaminación, pero el costo sube.

El frente de Pareto muestra las opciones no dominadas.

Posibles mejores soluciones

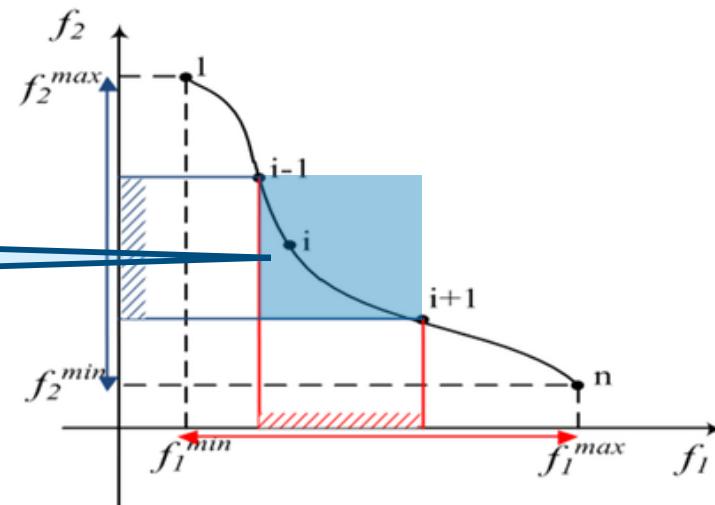


MOGA - Algoritmo Non-dominated Sorting Genetic II (NSGA-2)

El principal aporte NSGA-2 en el contexto de algoritmos genéticos es que NSGA-II no solo busca soluciones óptimas, sino que intenta **mantener una distribución uniforme a lo largo del frente de Pareto**.

Durante la etapa de selección de individuos se clasifican las soluciones a tener en cuenta para la reproducción aquellas no dominadas y luego las soluciones dominadas según diferentes frentes. Para mantener uniformes la distribución se calcula la **distancia de densidad o hacinamiento (Crowding Distance)**, para elegir la más distantes.

Se eligen candidatos mas
aislados
(crowding distance mayor)



1- Find the first and the last member of a front (first front is considered here)

$$2- CD_1 = CD_n = \infty$$

3- considering i

a) for the first objective:

$$d_i^1 = \frac{|f_1^{i+1} - f_1^{i-1}|}{f_1^{\max} - f_1^{\min}}$$

b) for the second objective:

$$d_i^2 = \frac{|f_2^{i+1} - f_2^{i-1}|}{f_2^{\max} - f_2^{\min}}$$

$$4- CD_i = d_i^1 + d_i^2$$

5- This should be done for each member of any front

MOGA - Ejemplo de cálculo de Crowding Distance

- Se le asigna a cada **solución (individuo)** distancia 0
- Para cada **objetivo f_k** se ordenan las soluciones según f_k
- A los **extremos** se le asigna **distancia infinita** para que siempre se tengan en cuenta en la selección
- Para cada **individuo i** se calcula su distancia $d(i)$:

$$d(i) += \frac{f_k(i+1) - f_k(i-1)}{f_k^{\max} - f_k^{\min}}$$

donde $f_k(i+1)$ y $f_k(i-1)$ son los vecinos adyacentes de i en el orden del objetivo

- La distancia final de cada individuo es la suma de las distancias sobre todos los objetivos.

Solución	f_1	f_2
A	2	9
B	4	6
C	6	3
D	8	1

- Para f_1 , ordenamos: A(2), B(4), C(6), D(8). A y D tienen distancia infinita

$$d_B = \frac{6 - 2}{8 - 2} = \frac{4}{6} = 0.67 \quad d_C = \frac{8 - 4}{8 - 2} = \frac{4}{6} = 0.67$$

- Para f_2 , ordenamos: D(1), C(3), B(6), A(9). D y A tienen distancia infinita

$$d_B+ = \frac{9 - 3}{9 - 1} = \frac{6}{8} = 0.75 \quad d_C+ = \frac{6 - 1}{9 - 1} = \frac{5}{8} = 0.625$$

Resultado:

A = ∞

B = $0.67 + 0.75 = 1.42$

C = $0.67 + 0.625 = 1.29$

D = ∞

Se preferirían A y D (extremos) y luego B sobre C.

MOGA - Implementación del ejemplo continuo usando DEAP y NSGA-2

Selección utilizándose NSGA2

```
# 1. Definir el tipo de fitness y el individuo (multiobjetivo,  
minimización)  
creator.create("FitnessMin", base.Fitness, weights=(-1.0, -1.0)) #  
minimizamos ambos objetivos  
creator.create("Individual", list, fitness=creator.FitnessMin)  
  
# 2. Toolbox: generar individuos y población  
toolbox = base.Toolbox()  
toolbox.register("attr_float", random.uniform, 0, 10) # valores entre  
0 y 10  
toolbox.register("individual", tools.initRepeat, creator.Individual,  
toolbox.attr_float, 2) # [x, y]  
toolbox.register("population", tools.initRepeat, list, toolbox.individual)  
  
# 3. Función de evaluación: costo vs contaminación  
def eval_cost_pollution(ind):  
    x, y = ind  
    cost = x**2 + y**2 # Costo ~ cuadrático  
    pollution = (x-5)**2 + (y-5)**2 # Contaminación ~ centro en (5,5)  
    return cost, pollution  
  
toolbox.register("evaluate", eval_cost_pollution)  
  
# 4. Operadores genéticos  
toolbox.register("mate", tools.cxSimulatedBinaryBounded, low=0, up=10, eta=20.0)  
toolbox.register("mutate", tools.mutPolynomialBounded, low=0, up=10, eta=20.0, indpb=0.2)  
toolbox.register("select", tools.selNSGA2)  
  
# 5. Algoritmo principal  
def main():  
    random.seed(42)  
    pop = toolbox.population(n=50)  
    hof = tools.ParetoFront()  
    stats = tools.Statistics(lambda ind: ind.fitness.values)  
    stats.register("avg", lambda fits: tuple(sum(f)/len(f) for f in zip(*fits)))  
    stats.register("min", lambda fits: tuple(min(f) for f in zip(*fits)))  
    stats.register("max", lambda fits: tuple(max(f) for f in zip(*fits)))  
  
    # Evolucionar con NSGA-II  
    algorithms.eaMuPlusLambda(pop, toolbox, mu=50, lambda_=100, cxpb=0.7, mutpb=0.3,  
                             ngen=40, stats=stats, halloffame=hof, verbose=True)  
    print("\n--- Frente de Pareto ---")  
    for ind in hof:  
        print(ind, ind.fitness.values)
```

Inteligencia Artificial

Machine Learning - (Aprendizaje Automático)

Introducción



Machine Learning

“The field of study that gives computers the ability to learn without being explicitly programmed.”

Samuel, A. L. (1959). Some Studies in Machine Learning Using the Game of Checkers. IBM Journal of Research and Development, 3(3), 210–229.

“A computer program is said to learn from **experience E** with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.”

Mitchell, T. M. (1997). Machine Learning. McGraw-Hill.

Machine Learning

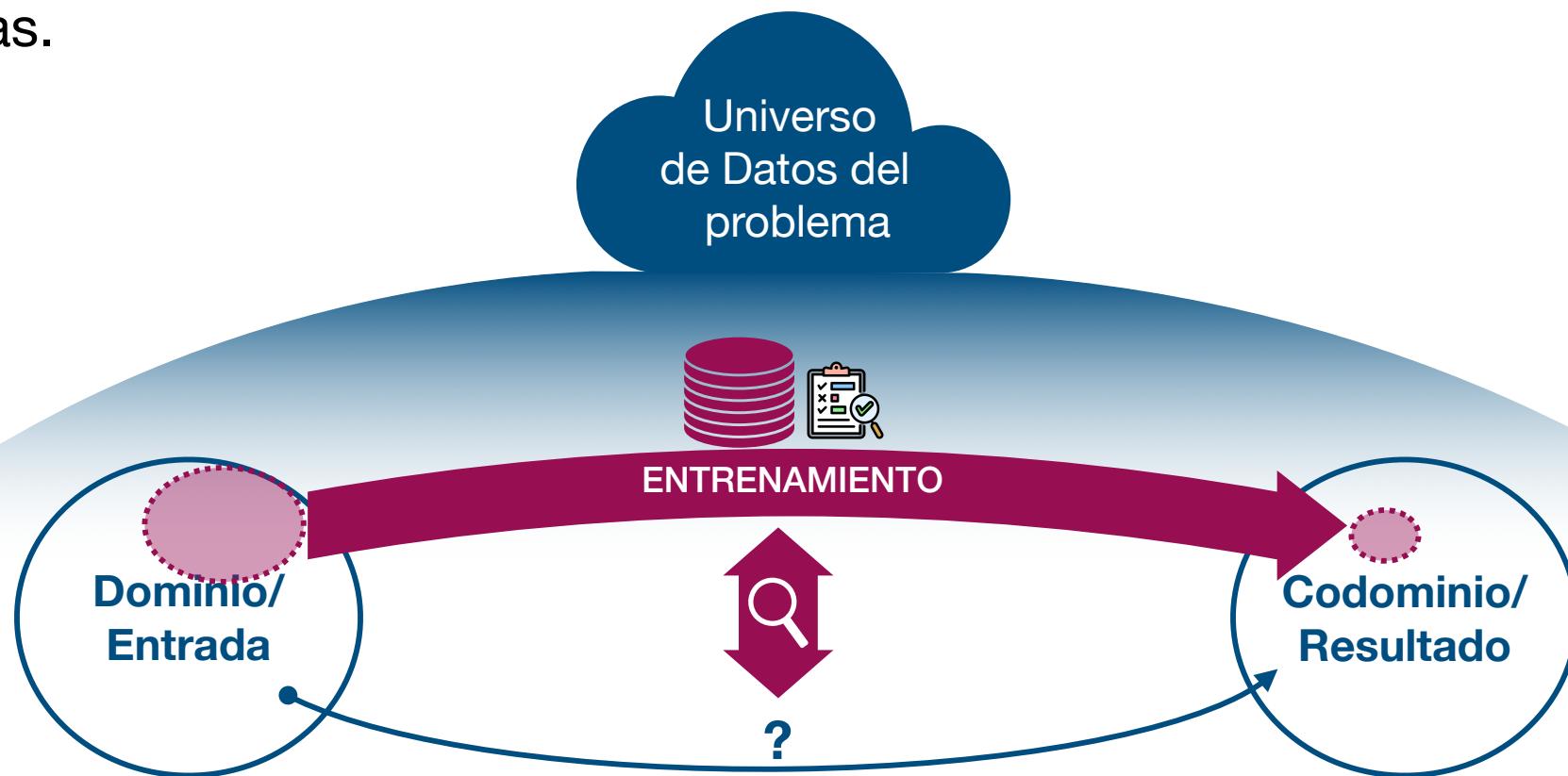
En otras palabras podemos definir el aprendizaje como “**capacidad de mejorar en una tarea a través de la práctica**”.

Esto plantea dos preguntas fundamentales: **¿cómo sabe la computadora si está mejorando?** y **¿cómo sabe qué debe cambiar para mejorar?**. Existen diferentes respuestas posibles, que dan lugar a distintos tipos de aprendizaje automático:

- **Aprendizaje supervisado**: El algoritmo recibe un conjunto de entrenamiento con las respuestas correctas y aprende a generalizar para responder correctamente a nuevas entradas.
- **Aprendizaje no supervisado**: No se proporcionan respuestas correctas. El algoritmo busca similitudes entre los datos para agruparlos.
- **Aprendizaje por refuerzo (reinforcing)**: El algoritmo sabe si su respuesta fue correcta o no, pero no cómo mejorarla. Aprende explorando distintas opciones.
- **Aprendizaje evolutivo**: La evolución biológica puede interpretarse como un proceso de aprendizaje (aptitud de adaptación). Podemos modelar este proceso en una computadora usando el concepto de aptitud, que representa una puntuación que indica qué tan buena es una solución actual.

Machine Learning - Aprendizaje Supervisado

El algoritmo recibe un conjunto de entrenamiento **con las respuestas correctas** y aprende a generalizar para responder correctamente a nuevas entradas.



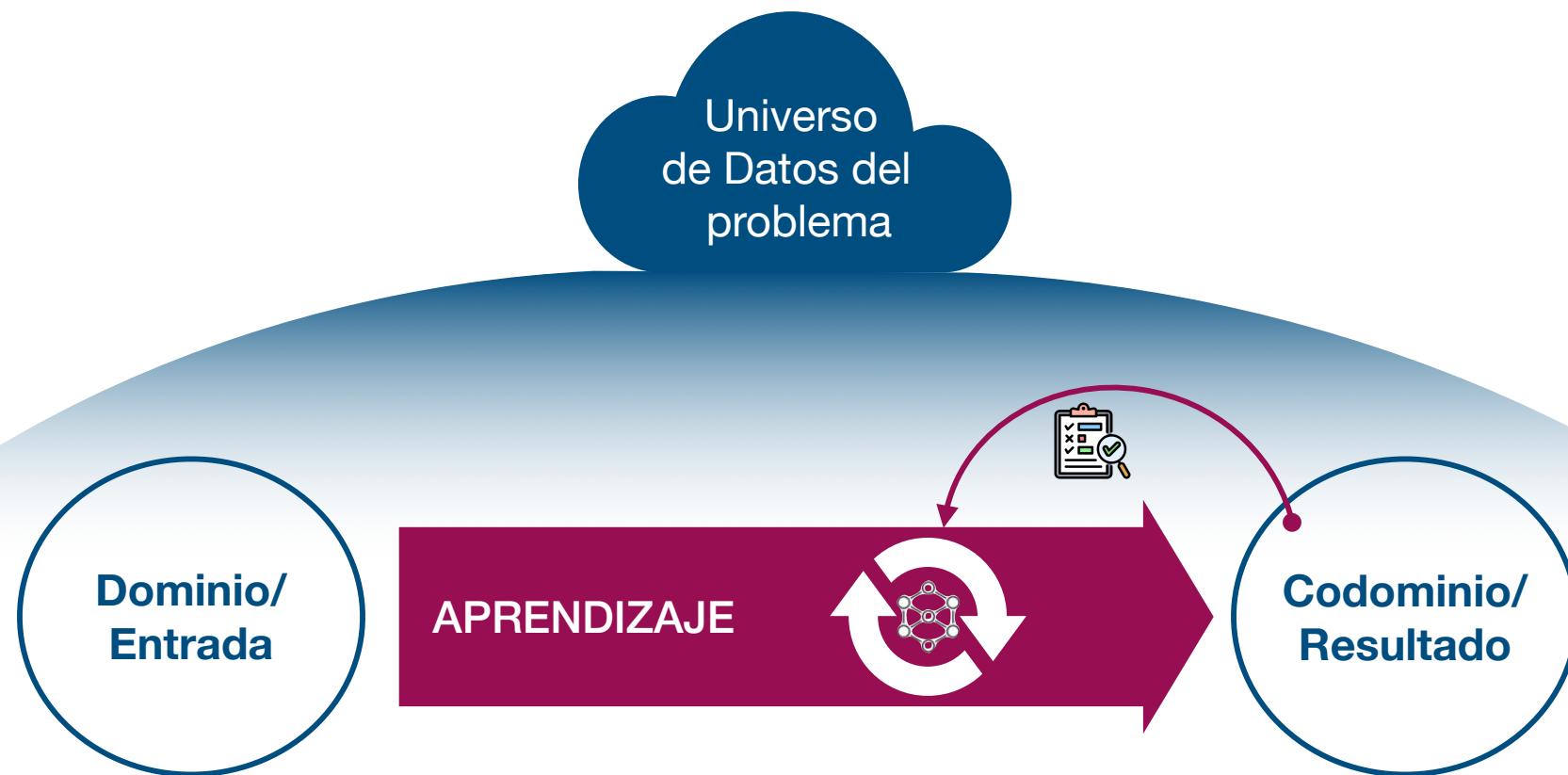
Machine Learning - Aprendizaje No Supervisado

No se proporcionan respuestas correctas. El algoritmo busca similitudes entre los datos para agruparlos



Machine Learning - Aprendizaje por refuerzo (reinforcing)

El algoritmo sabe si su respuesta fue correcta o no, pero no cómo mejorarla. Aprende explorando distintas opciones.



Machine Learning - ¿ Cuándo utilizarlo ?

Aunque el uso de ML ha crecido exponencialmente, su aplicación se **concentra en contextos con grandes volúmenes de datos** para extraer información relevante

- ML es útil en problemas que requieren la definición de un **número muy grande de reglas**, o de reglas muy complejas
- ML puede dar buenas soluciones a problemas complejos, para los que no conocemos una buena solución algorítmica
- Un sistema de ML puede adaptarse a ambientes cambiantes entrenándolo periódicamente con nuevos datos
- ML puede ayudar a las personas a ganar entendimiento sobre problemas difíciles
- Puede ayudar a analizar y extraer conclusiones a partir de grandes cantidades de datos

Machine Learning - Qué esperamos como resultado

Tipo de Aprendizaje	Tipo de Resultado Esperado	Ejemplo
Aprendizaje Supervisado	Clasificación	Predecir categorías. Ej: detectar si un email es spam o no.
	Regresión	Predecir valores numéricos. Ej: precio de una casa.
Aprendizaje No Supervisado	Agrupamiento (Clustering)	Descubrir grupos. Ej: segmentación de clientes por comportamiento.
	Detección de anomalías	Identificar valores atípicos. Ej: fraude bancario.
	Reducción de dimensionalidad	Simplificar datos. Ej: visualización de datos de alta dimensión.
	Reglas de asociación	Encontrar relaciones frecuentes. Ej: productos comprados juntos.
Aprendizaje por Refuerzo	Secuencias óptimas de acciones (reglas)	Tomar decisiones en ambientes dinámicos. Ej: robots o videojuegos.
Aprendizaje Auto-supervisado	Representaciones de datos	Aprender estructuras internas sin etiquetas explícitas. Ej: embeddings.
Aprendizaje Semi-supervisado	Clasificación / Regresión	Mezcla de datos etiquetados y no etiquetados. Mejora generalización.

Machine Learning - Problemas y desventajas

Tipo de Problema	Problema / Desventaja	Descripción	Ejemplos
Datos	Insuficiencia de datos	ML requiere grandes volúmenes de datos diversos para generalizar bien.	Sobreajuste, mala precisión.
	Datos sesgados o desequilibrados	Si los datos no representan todas las clases o contextos, el modelo será parcial o injusto.	Discriminación algorítmica.
	Mala calidad de datos	Datos erróneos, incompletos afectan negativamente el aprendizaje.	Predicciones erróneas, comportamiento impredecible.
	Dificultad para etiquetar datos	El etiquetado supervisado puede ser costoso o lento.	Retrasa el entrenamiento y limita la precisión.
Modelado	Sobreajuste (overfitting)	El modelo se ajusta demasiado a los datos de entrenamiento y falla en generalizar.	Buen rendimiento en entrenamiento, pobre en producción.
	Subajuste (underfitting)	El modelo es demasiado simple y no captura la complejidad del problema.	Baja precisión general.
	Alta complejidad computacional	Modelos avanzados como redes neuronales profundas requieren mucha computación y energía.	Costos elevados de entrenamiento y mantenimiento.
	Interpretabilidad limitada	Muchos modelos son cajas negras difíciles de explicar.	Problemas de confianza, auditoría y cumplimiento normativo.
Ética y Sociedad	Discriminación y sesgo	El modelo puede reproducir o amplificar sesgos presentes en los datos.	Injusticia en decisiones
	Falta de transparencia	No siempre se entiende por qué el modelo toma una decisión.	Dificultad para explicar decisiones ante usuarios o reguladores.
	Riesgo de automatización	Puede reemplazar tareas humanas sin considerar impacto social o laboral.	Pérdida de empleos, brechas digitales.
Implementación	Requiere infraestructura	Se necesita hardware especializado, almacenamiento, y despliegue robusto.	Costos de infraestructura.
	Mantenimiento constante	Los modelos necesitan ser actualizados con nuevos datos.	Degrado del rendimiento con el tiempo.
	Riesgo de sobreconfianza	Creer que el modelo siempre tiene razón puede llevar a malas decisiones automatizadas.	Falta de supervisión humana.

Inteligencia Artificial

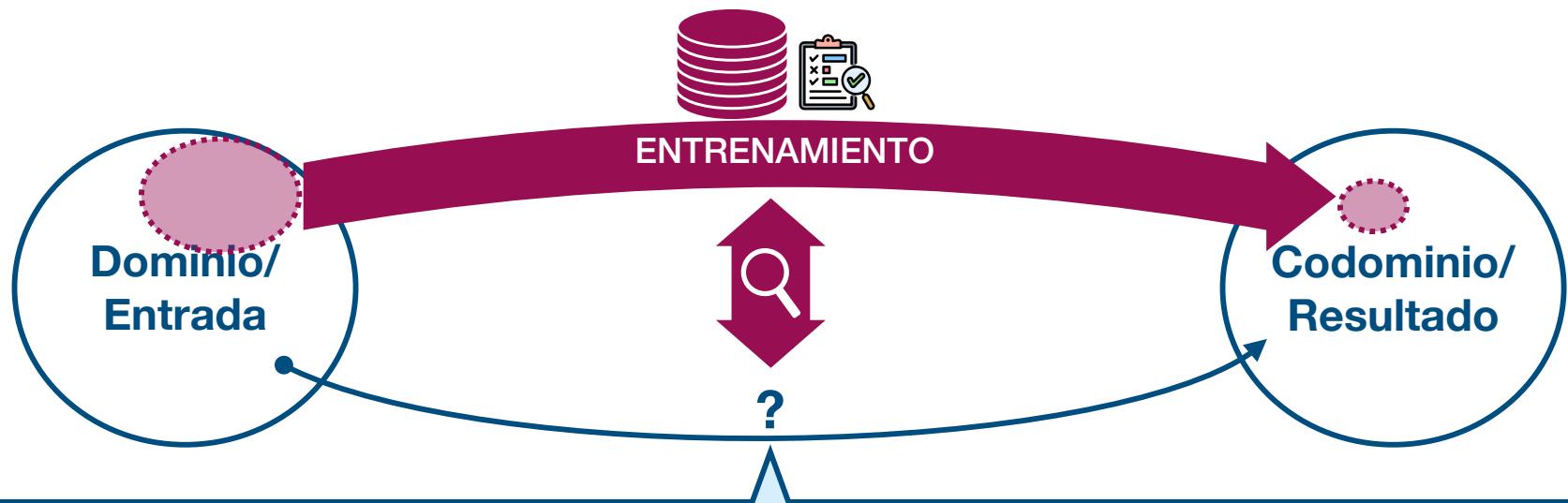
Machine Learning - (Aprendizaje Automático)

Aprendizaje Supervisado



Machine Learning - Aprendizaje Supervisado

El algoritmo recibe un conjunto de entrenamiento **con las respuestas correctas** y aprende a generalizar para responder correctamente a nuevas entradas.



De manera general, los problemas se dividen en dos categorías, de acuerdo a la salida esperada del algoritmo:

- Problemas de **clasificación**: La salida del algoritmo es un valor tomado de un conjunto de finito de valores
- Problemas de **predicción (regresión)**: La salida es un número entero o real

ML Supervisado - Desafíos del aprendizaje

El objetivo del aprendizaje automático es conseguir **un modelo que se ajuste lo mejor posible** (aprenda) a los datos de entrenamiento. La **calidad** del modelo aprendido será determinante para la certeza de las respuestas cuando lo utilicemos para clasificar o predecir datos ante nuevas consultas (no estudiadas durante el entrenamiento).

Los problemas que conllevan los extremos del ajuste impactan negativamente en la calidad del modelo:

- **Subajuste (Underfitting)**: modelo es demasiado simple para captar los patrones de los datos.
- **Sobreajuste (Overfitting)**: cuando el modelo aprende excesivamente de los datos de entrenamiento, incluyendo el ruido o las particularidades específicas, y por eso no generaliza bien a nuevos dato.

ML Supervisado - ¿Cómo medimos la calidad?

Según el tipo de problema a resolver (clasificación o regresión), se emplean distintas métricas que permiten evaluar la calidad del modelo. Estas métricas son fundamentales durante el desarrollo, ya que orientan el ajuste del modelo hacia una solución adecuada.

Por ejemplo, en problemas de **clasiﬁcación** se utilizan métricas como la **precisión** (accuracy), la **sensibilidad** (recall) o **F1-Score** que tiene en cuenta ambas anteriores.

En cambio, en problemas de **regresión** (predicción), se aplican métricas como el **error absoluto medio** (MAE) o el **error cuadrático medio** (MSE), que indican qué tan lejos están las predicciones de los valores reales

ML Supervisado - Proceso general de desarrollo e implementación

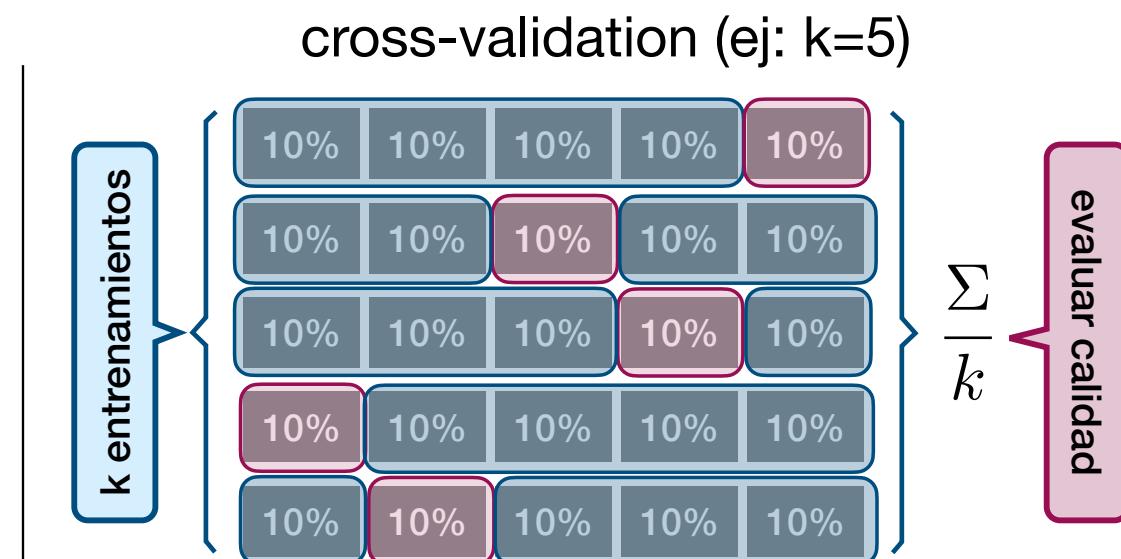
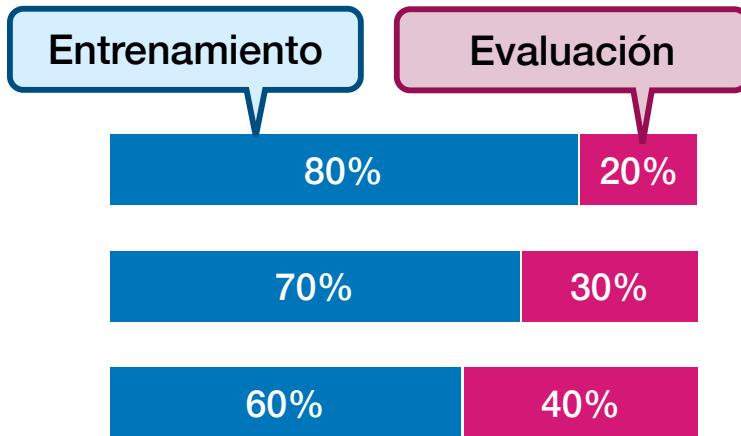
Para aplicar las métricas que guían el desarrollo del modelo, y dado que este se entrena a partir de un conjunto de datos, es necesario **particionar** dicho conjunto en diferentes subconjuntos, destinados a las **distintas etapas** del proceso: uno para el **aprendizaje (entrenamiento)** y otro para la **evaluación (prueba)**.

La elección de cómo seleccionar estas particiones y sus elementos tiene un alto impacto en el proceso de desarrollo. Aunque en general depende fuertemente de cada problema a resolver y el tipo de modelo, se destina un porcentaje mayor de los datos a la etapa de entrenamiento.

ML Supervisado - Partición de los datos durante el desarrollo

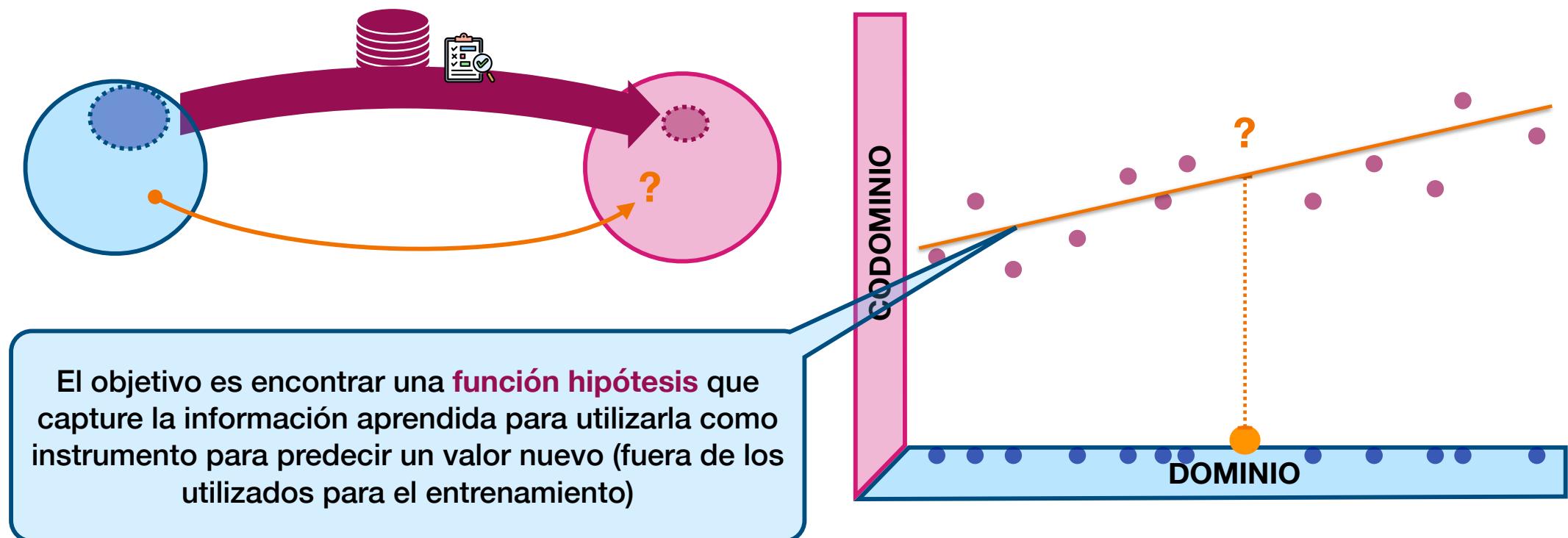
Una práctica común es dividir los datos en un 80 % para entrenamiento y un 20 % para evaluación, aunque también se utilizan otras proporciones como 70/30 o 60/40, dependiendo del tamaño y la calidad del conjunto de datos disponible.

Cuando se dispone de **pocos datos** o para fortalecer la evaluación, se utiliza una técnica llamada **validación cruzada (cross-validation)**. La variante más simple (**k-fold cross-validation**), propone dividir los datos en **k subconjuntos (folds)**. Se entrena el modelo **k** veces utilizando **k-1** folds para entrenamiento y el restante para prueba. Al final, se promedian los resultados.



ML Supervisado - Regresión (predicción) Lineal (RL)

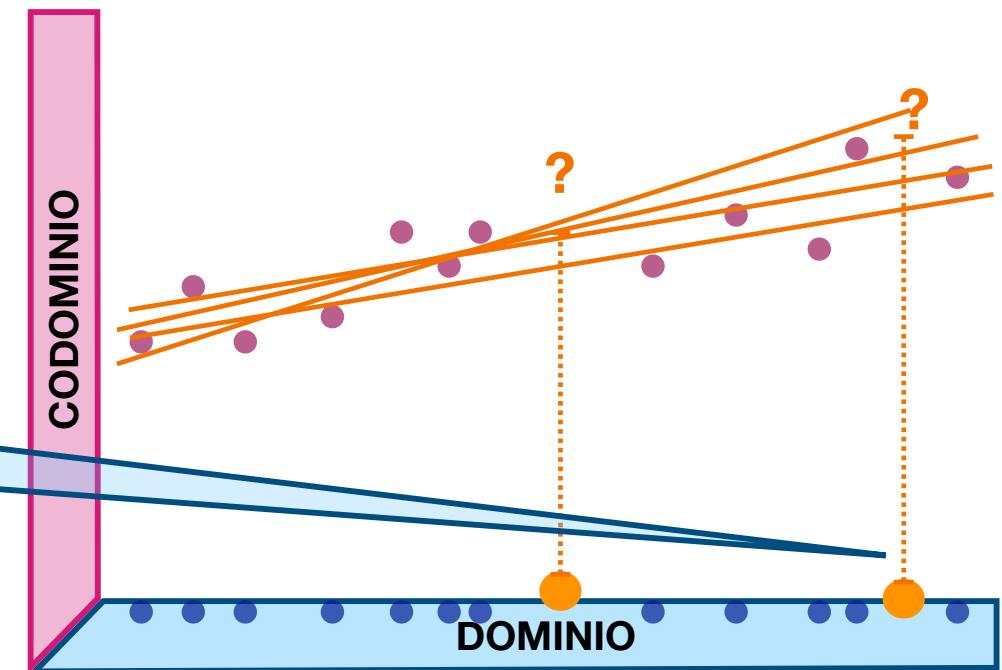
Comenzaremos el abordaje de técnicas de ML Supervisado explorando una técnica simple de predicción. Esto es, dado un sub-conjunto de datos que representan cierta información (entrada y resultado), nuestro objetivo es aprender de este sub-conjunto para poder predecir el resultado para cualquier otro dato de entrada.



ML Supervisado - Regresión Lineal (RL)

Pueden existir varias **funciones lineales** que capturen la información, lo que buscamos es la **más precisa**. Es decir, aquella que minimice el error entre los datos (puntos) del aprendizaje y la función.

La función a encontrar debe poder predecir de la mejor manera posible cualquier otro dato que no forme parte del entrenamiento



ML Supervisado - Regresión Lineal (RL)

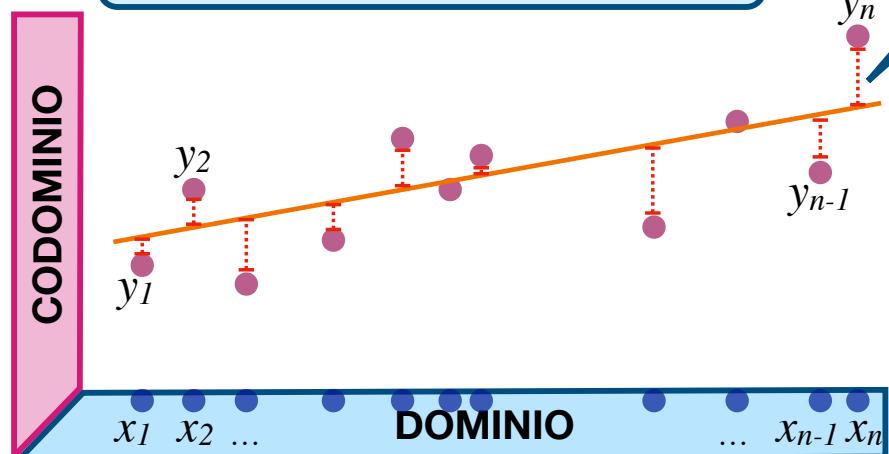
Una métrica posible para alcanzar el objetivo es el **Error Mínimo Cuadrado (MSE)** que computa **el promedio de los errores** entre los puntos de entrenamiento y la función hipótesis (h) a encontrar. Recordemos que la forma normal de una función lineal es:

$$h(x) = \theta_0 + \theta_1 x$$

θ_0 y θ_1 son los **parámetros** del modelo a aprender

Podemos calcular al diferencia como:
 $(y_i - h(x_i))$

$$\text{MSE} = \frac{\cancel{+} + \cancel{+} + \cancel{+} + \cancel{+} + \cancel{+} + \cancel{+}}{n}$$



Reemplazando h por su definición y tomando el cuadrado obtenemos :
 $(y_i - (\theta_0 + \theta_1 x_i))^2$

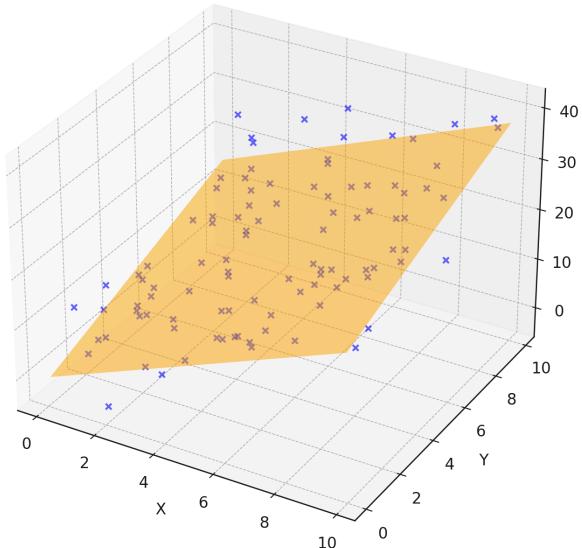
$$\frac{\sum_{i=1}^n (y_i - (\theta_0 + \theta_1 x_i))^2}{n}$$

ML Supervisado - RL - Generalización

Los modelos a aprender y predecir pueden tener más de una dimensión (características a predecir)

Ejemplo podrían estar en el espacio
(en caso de 2 características)

$$h(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$



Para el caso general de m características

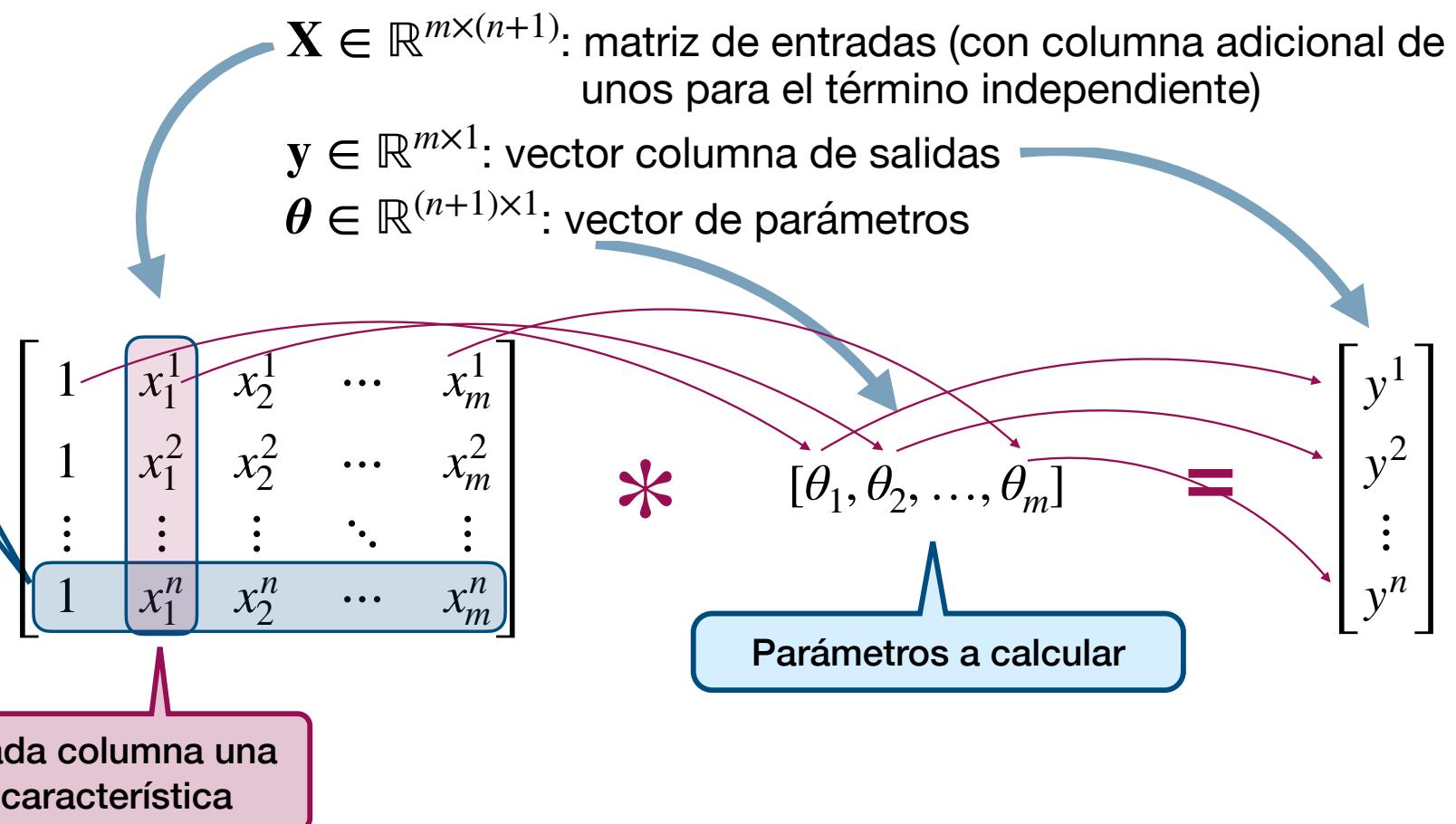
$$h(x_1, \dots, x_m) = \theta_0 + \theta_1 x_1 + \dots + \theta_m x_m$$

MSE

$$\frac{\sum_{i=1}^n (y_i - (\theta_0 + \theta_1 x_1^i + \dots + \theta_m x_m^i))^2}{n}$$

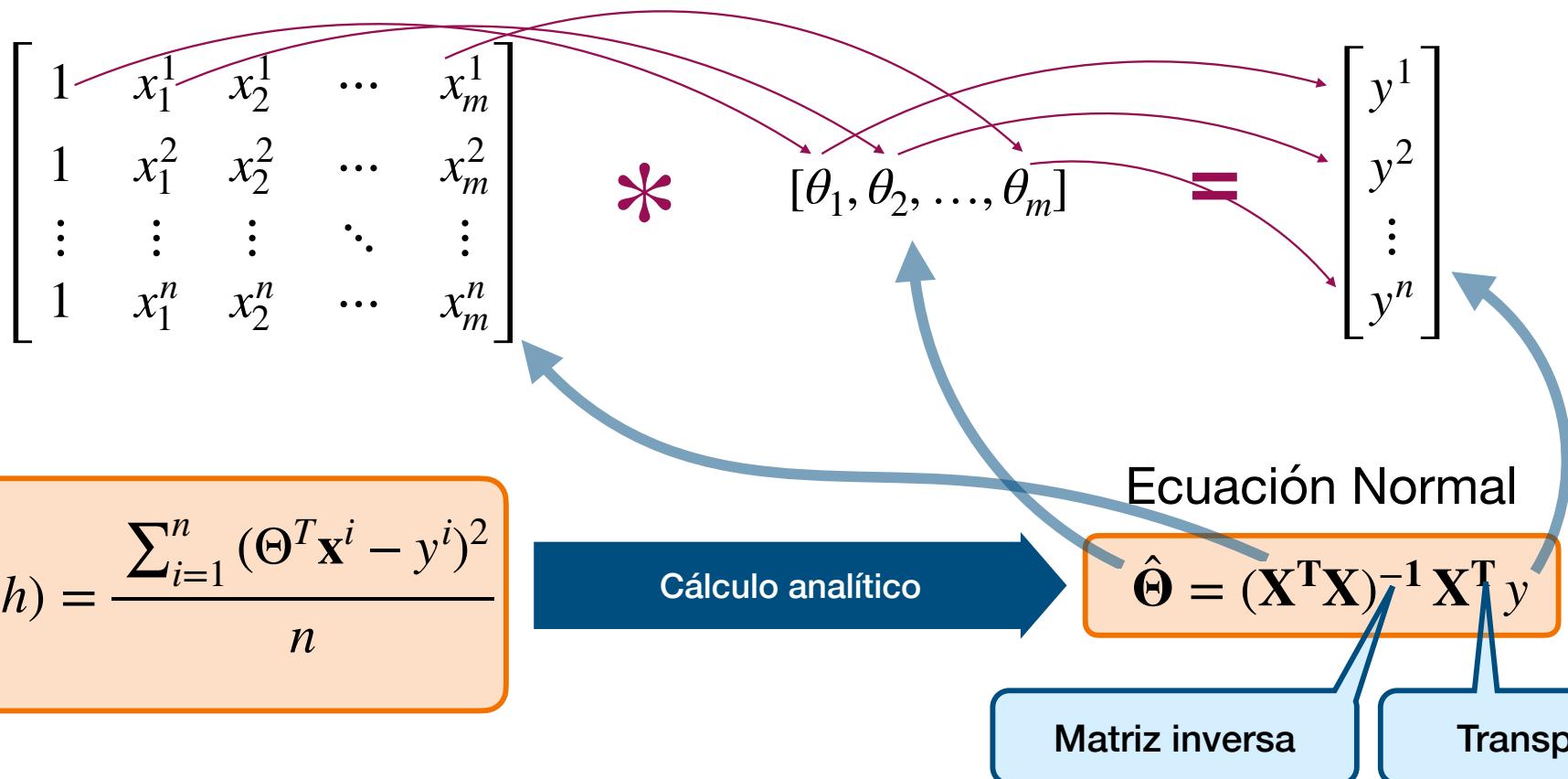
ML Supervisado - RL - Notación Matricial

Si tenemos un conjunto de datos de entrenamiento de n instancias de m características, anotamos:



ML Supervisado - RL - Ecuación Normal y MSE

Con esta representación matricial, utilizando la definición de MSE, podemos computar de manera analítica (propiedades de álgebra) el vector de parámetros que ajusta la función **hipótesis** a obtener.



ML Supervisado - RL - Ejemplo

$$\hat{\Theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T y$$

x1	x2	y
1	2	10
2	1	12
3	4	20
4	3	22
5	5	30

$$X = \begin{bmatrix} 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & 3 & 4 \\ 1 & 4 & 3 \\ 1 & 5 & 5 \end{bmatrix}, \quad y = \begin{bmatrix} 10 \\ 12 \\ 20 \\ 22 \\ 30 \end{bmatrix}$$

Para calcular la **inversa** debe ser una matriz cuadrada y el **determinante** distinto de cero.

$$X^{-1} = \frac{1}{\det(X)} * \text{adj}(X)$$

DEBE coincidir la cantidad de **filas** con la cantidad de **columnas** de los multiplicandos

$$X^T = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 4 & 3 & 5 \end{bmatrix} \quad \sum *$$

$$X^T X = \begin{bmatrix} 5 & 15 & 15 \\ 15 & 55 & 49 \\ 15 & 49 & 55 \end{bmatrix}$$

$$(X^T X)^{-1} = \begin{bmatrix} 1.2 & -0.1667 & -0.1667 \\ -0.1667 & 0.2778 & -0.2222 \\ -0.1667 & -0.2222 & 0.2778 \end{bmatrix}$$

$$\hat{\Theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} \quad h(x_1, x_2) = 2 + 3x_1 + 1x_2$$

ML Supervisado - RL - Costo computacional

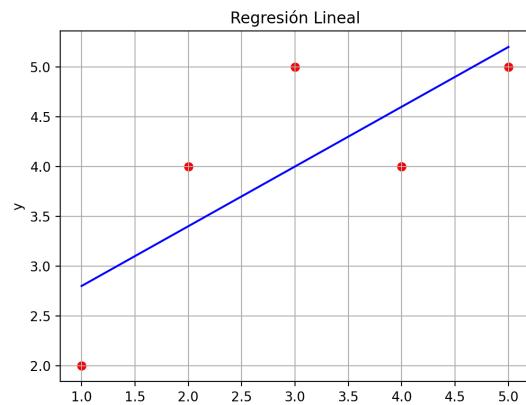
Como se puede observar, el costo computacional no es simple. Los algoritmos para calcular la ecuación normal para n instancias teniendo en cuenta m características es:

$$\mathcal{O}(nm^2 + m^3)$$

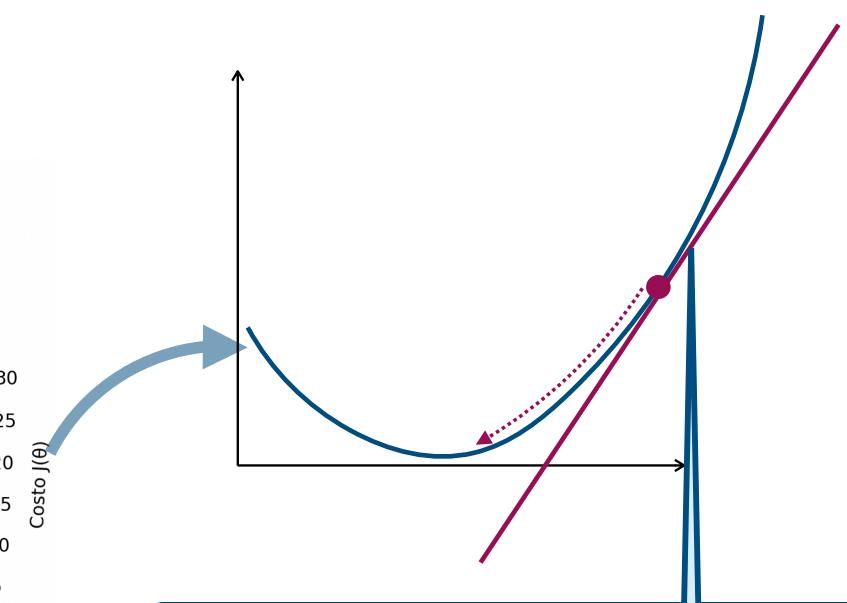
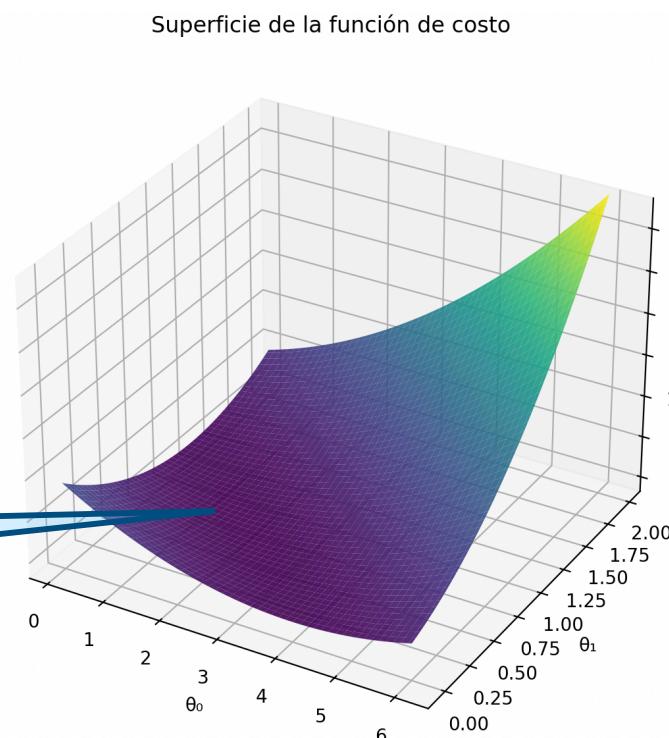
Recordemos que la precisión de nuestro modelo se encuentra altamente relacionado con la cantidad de datos de entrenamiento, por ejemplo, si tenemos 100.000 instancias (un ejemplo de muy simple), aún con pocas características, deja de ser una opción viable. Además, del tiempo, debemos mantener las matrices en memoria para poder operar con ellas.

ML Supervisado - RL - Cómputo con gradientes

Una alternativa es utilizar las propiedades de la función de costo para guiar la búsqueda de los parámetros.



Para regresión lineal, el espacio de pesos es una función convexa con un único mínimo

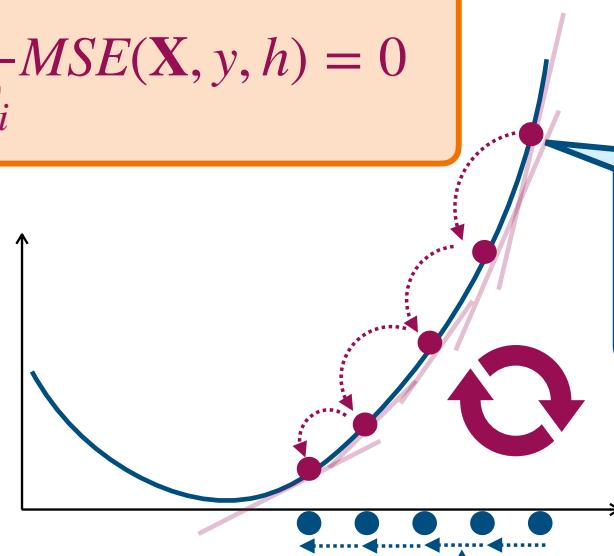


Podemos utilizar la derivada de la función de costo para encontrar el mínimo, es decir, los **parámetros** de nuestra función de hipótesis

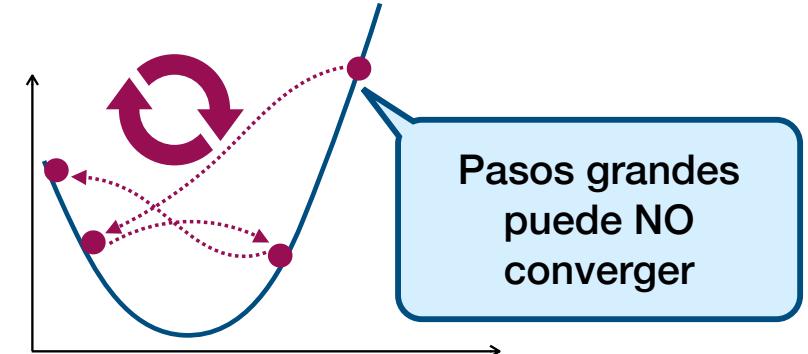
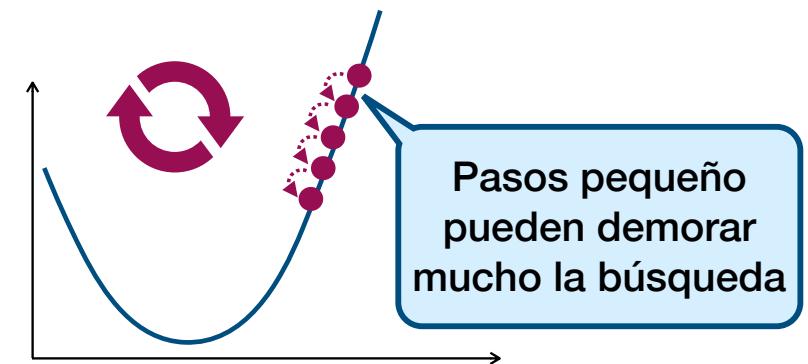
ML Supervisado - RL - Cómputo con gradientes

La idea es ir recorriendo iterativamente la derivada de la función de costo en búsqueda del mínimo. Esto es, los valores de los parámetros que mejor ajustan nuestra función de hipótesis, **cuando la derivada de la función de error es cero.**

$$\frac{\partial}{\partial \theta_i} MSE(\mathbf{X}, y, h) = 0$$



Vamos buscando de a **pasos** en búsqueda del mínimo
(similar “*hill climbing*” pero con mínimo en vez de máximo)



ML Supervisado - RL - Repaso de reglas de derivación

Constante	$\frac{\partial}{\partial x} k = 0$
Constante por función	$\frac{\partial}{\partial x} k f(x) = k \frac{\partial}{\partial x} f(x)$
Regla de la suma	$\frac{\partial}{\partial x} f(x) + g(x) = \frac{\partial}{\partial x} f(x) + \frac{\partial}{\partial x} g(x)$
Regla de la potencia	$\frac{\partial}{\partial x} x^k = k x^{k-1} \quad (r \neq 0)$
Regla de la cadena	$\frac{\partial}{\partial x} f(g(x)) = f'(g(x)) \frac{\partial}{\partial x} g(x)$

ML Supervisado - RL - Cómputo con gradientes

Comencemos con un modelo simple de una sola característica: $\theta_0 1 + \theta_1 x$

$$\frac{\partial}{\partial \theta_j} MSE(\mathbf{x}, y, h_{\theta}) = 0$$

$$\begin{aligned}\frac{\partial}{\partial \theta_j} MSE(x, y, h_{\theta}) &= \frac{\partial}{\partial \theta_j} (h_{\theta}(x) - y)^2 \\ &= \frac{\partial}{\partial \theta_j} ((\theta_0 1 + \theta_1 x) - y)^2 \\ &= 2((\theta_0 1 + \theta_1 x) - y) * \frac{\partial}{\partial \theta_j} ((\theta_0 1 + \theta_1 x) - y) \\ &= 2(h_{\theta}(x)) - y) * \frac{\partial}{\partial \theta_j} ((\theta_0 1 + \theta_1 x) - y)\end{aligned}$$

ML Supervisado - RL - Cómputo con gradientes

$$\frac{\partial}{\partial \theta_j} MSE(x, y, h_{\theta}) = 2(h_{\theta}(x)) - y) * \frac{\partial}{\partial \theta_j} ((\theta_0 1 + \theta_1 x) - y)$$

$$\frac{\partial}{\partial \theta_0} MSE(x, y, h_{\theta}) = 2(h_{\theta}(x)) - y) * 1$$

$$\frac{\partial}{\partial \theta_1} MSE(x, y, h_{\theta}) = 2(h_{\theta}(x)) - y) * x$$

Para movernos en la dirección contraria al gradiente debemos restar el gradiente a los pesos actuales:

- $\theta_0 = \theta_0 - \eta * 2(h_{\theta}(x)) - y)$
- $\theta_1 = \theta_1 - \eta * 2(h_{\theta}(x)) - y) * x$

η es la tasa de aprendizaje (learning rate), que permite regular la velocidad del aprendizaje

Notar que las actualizaciones son intuitivas, si $h_{\theta}(x) = \theta_0 + \theta_1 x > y$ debemos:

- Reducir θ_0
- Si x es positivo reducir θ_1 ; si x es negativo agrandar θ_1

ML Supervisado - RL - Multiples gradientes (características)

$$\nabla_{\theta} MSE(\mathbf{X}, y, h_{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} MSE(\mathbf{X}, y, h_{\theta}) \\ \frac{\partial}{\partial \theta_1} MSE(\mathbf{X}, y, h_{\theta}) \\ \dots \\ \frac{\partial}{\partial \theta_n} MSE(\mathbf{X}, y, h_{\theta}) \end{pmatrix} = \begin{pmatrix} 2 \sum_{i=1}^n (h_{\theta}(\mathbf{x}_i) - y_i) * 1 \\ 2 \sum_{i=1}^n (h_{\theta}(\mathbf{x}_i) - y_i) * x_{i,1} \\ \dots \\ 2 \sum_{i=1}^n (h_{\theta}(\mathbf{x}_i) - y_i) * x_{i,n} \end{pmatrix} = 2\mathbf{X}^T(\mathbf{X}\theta - y)$$

Para simplificar la lectura omitimos la división por n

Para movernos en la dirección contraria al gradiente:

$$\theta = \theta - \eta \nabla_{\theta} MSE(\mathbf{X}, y, h_{\theta}) = \theta - \eta 2\mathbf{X}^T(\mathbf{X}\theta - y)$$

Este método utiliza todo \mathbf{X} para entrenar, con los cuales con un conjunto grande de entrenamiento puede tardar mucho. Funciona bien en cuanto el número de características, es mucho más rápido que la ecuación normal.

ML Supervisado - RL - Estructura del Algoritmo

```
# Parámetros iniciales
theta = np.zeros((2, 1)) # θ₀ y θ₁
eta = 0.01                # tasa de aprendizaje
n_iter = 1000              # número de iteraciones

for iteration in range(n_iter):
    gradients = (1/m) * X_b.T @ (X_b @ theta - y) # derivadas parciales
    theta = theta - eta * gradients
```

Cantidad de instancias
(Datos de Entrenamiento)

ML Supervisado - RL - Cómputo estocástico con gradientes

Una desventaja del aprendizaje por gradientes, es que se utilizan **todos** los datos de entrenamiento para la búsqueda de los parámetros del modelo.

$$\theta = \theta - \eta \nabla_{\theta} MSE(\mathbf{X}, y, h_{\theta}) = \theta - \eta 2\mathbf{X}^T(\mathbf{X}\theta - y)$$

Una alternativa es el **aprendizaje estocástico por gradientes** utiliza un subconjunto elegido de forma aleatoria de los datos para la búsqueda.

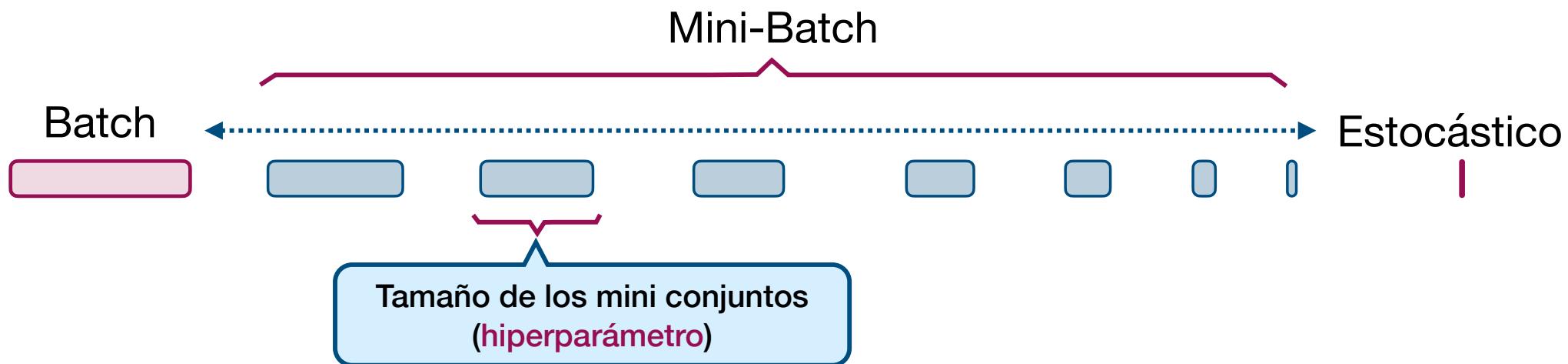
- La principal ventaja que ofrece el cómputo estocástico con gradientes es que **no se necesitan cargar todas las instancias** para realizar la búsqueda.
- La desventaja es que **puede tardar más tiempo** en converger a un buen mínimo local y no necesariamente siempre encontramos el valor óptimo para todo el conjunto de datos.
- Existen técnicas para ayudar la convergencia como **scheduling learning rate (Momentum)**, que ajusta la tasa de aprendizaje a medida que se acerca a un mínimo, similar a la técnica de **simulated annealing**.

ML Supervisado - RL - Cómputo estocástico con gradientes (**mini-batch**)

Una alternativa que pretende tomar lo mejor de las técnicas anteriores es la de **mini-batch**.

La idea es parecida que aprendizaje por gradientes, pero en vez de tomar una sola instancia se toman conjuntos chicos de forma aleatoria.

- Es menos errática que el aprendizaje estocástico.
- No utiliza todo el conjunto de entrenamiento.



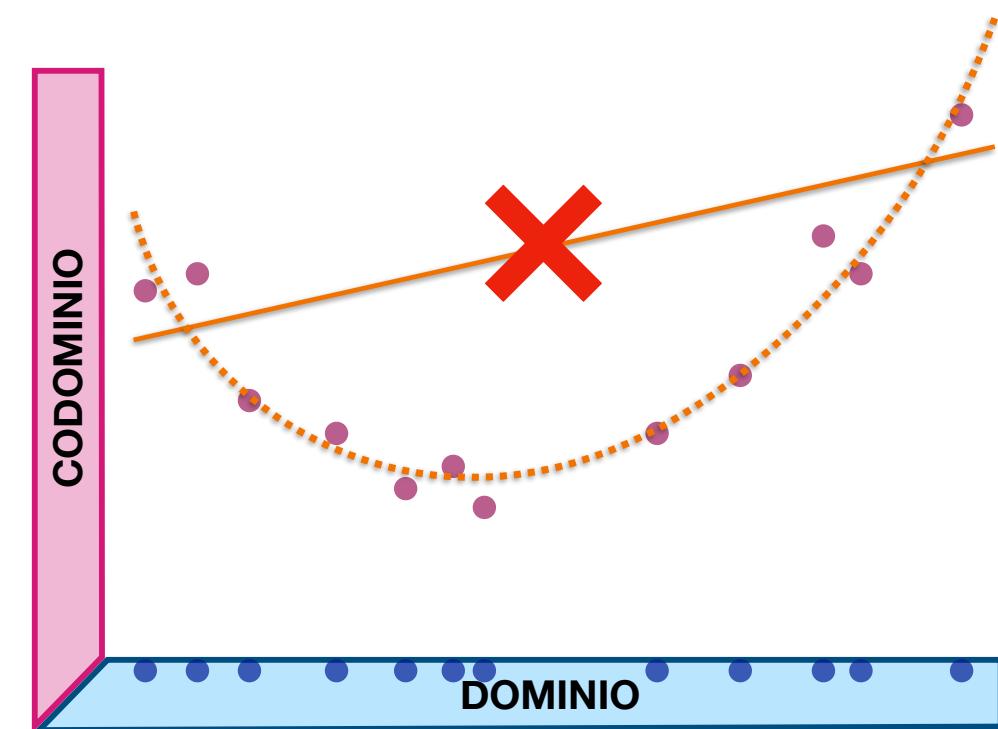
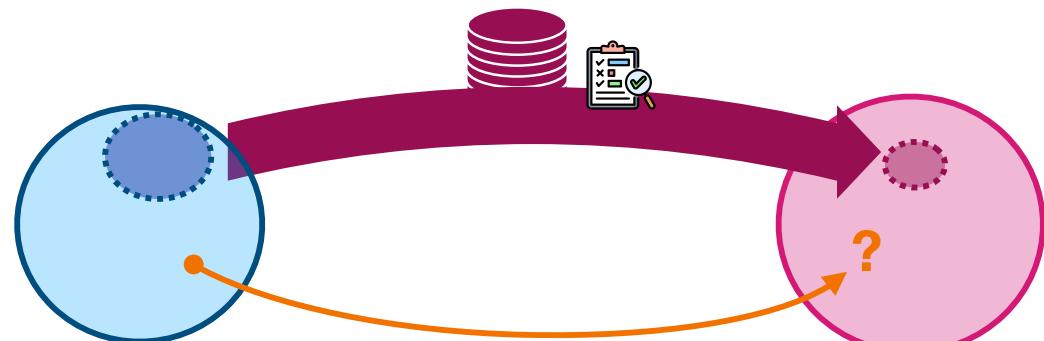
ML Supervisado - RL - Comparación



Convergencia	Inmediata	Gradual directa (poca varianza entre iteraciones)	Gradual (varianza media entre iteraciones)	Gradual (alta varianza entre iteraciones)
Hiperparámetros	Ninguno	Tasa de aprendizaje	Tasa de aprendizaje, tamaño de batch, etc.	Tasa de aprendizaje, etc.
Uso de datos de entrenamiento	exhaustivo	exhaustivo	reducido (ajustable)	mínimo
Comportamiento con muchas características	Lento	Rápido	Rápido	Rápido
Comportamiento con muchas instancias	Rápido	Lento	Rápido	Rápido

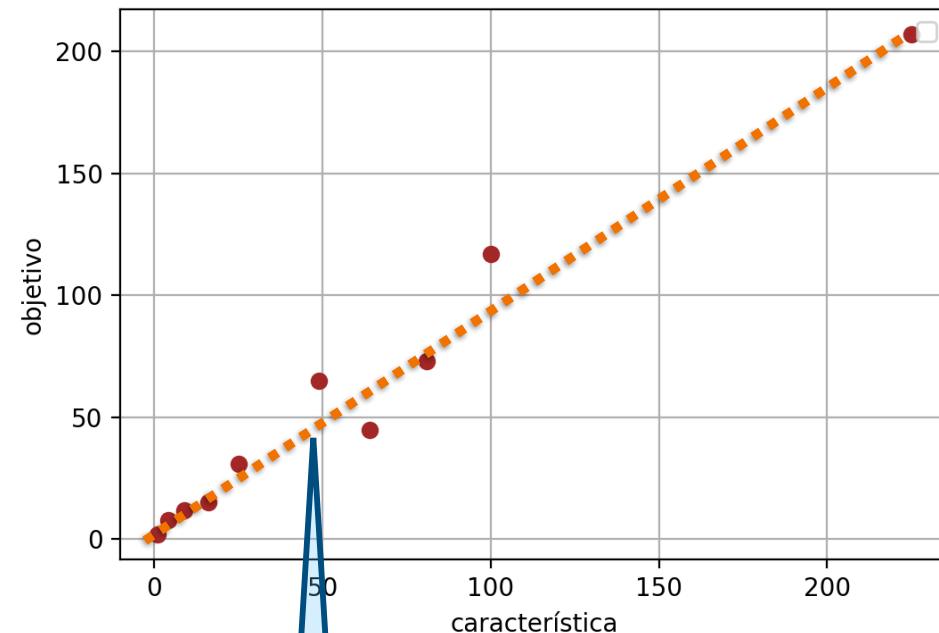
ML Supervisado - Regresión Polinomial

¿ Qué sucede si la función hipótesis a encontrar no tiene el comportamiento de una función lineal ?



ML Supervisado - Regresión Polinomial

Objetivo (y)	Característica (x)	x^2
65	-7	49
12	-3	9
8	2	4
2	1	1
8	2	4
15	4	16
31	5	25
45	7	49
73	9	81
117	10	100
207	15	225



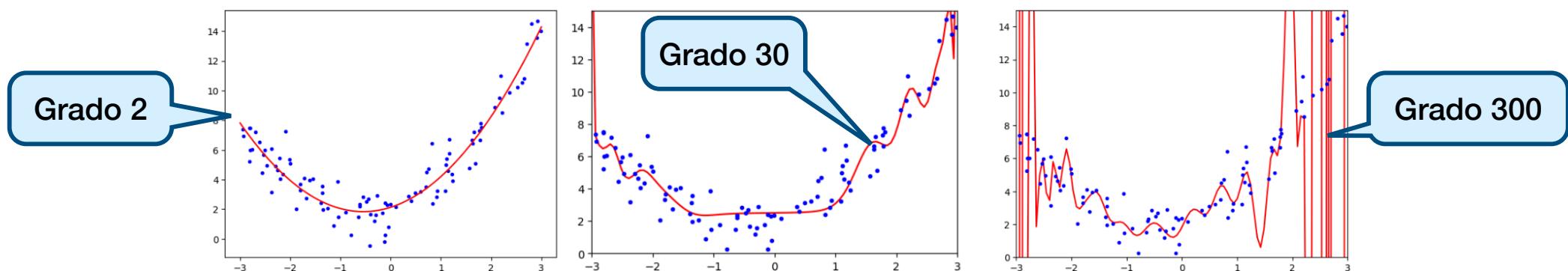
Teniendo en cuenta la característica agregada, el modelo parece ajustarse a una función hipótesis cuadrática.

ML Supervisado - Regresión Polinomial

La idea para poder ajustar modelos polinomiales es agregar más datos (características) con el valor de las características originales elevados al polinomio correspondiente. Luego buscamos, mediante **la técnica de regresión lineal el modelo** que mejor ajuste estos datos.

Si hay más de una característica propia del modelo a ajustar, **se agregan combinaciones** de las mismas. Por ejemplo, para 2 caract. y grado 3, agregan $a^2, a^3, b^2, b^3, ab, a^2b, ab^2$.

De manera general para n características y grado d , se agregan $\frac{(n + d)!}{d!n!}$ características adicionales. Se debe **tener en cuenta el problema la explosión de la cantidad de características** a incorporar.

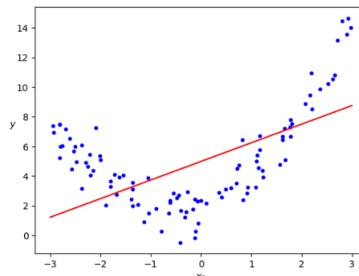


ML Supervisado - Underfitting y Overfitting

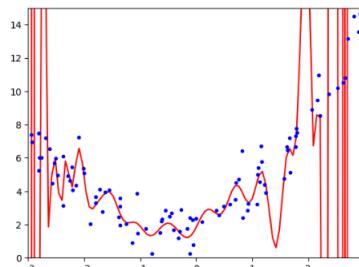
Como podemos apreciar, la elección de polinomio para el modelo a ajustar son parte de los **hiperparámetros**. La elección de los mismos puede derivar en la obtención de modelos ajustados serios problemas de predicción.

Se denomina **underfitting** cuando el modelo obtenido es demasiado simple y no logra aprender los patrones importantes de los datos.

Se denomina **overfitting** cuando el modelo obtenido aprende demasiados detalles y el ruido del conjunto de entrenamiento.



Underfitting: la función aprendida no refleja el comportamiento de los datos de entrenamiento

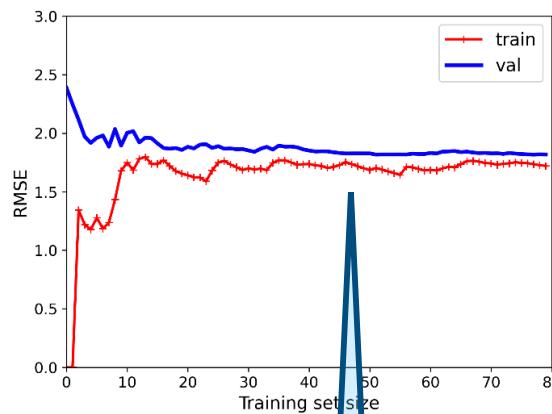


Overfitting: la función se ajusta muy bien a los datos de entrenamiento, pero no va a funcionar bien para nuevos datos

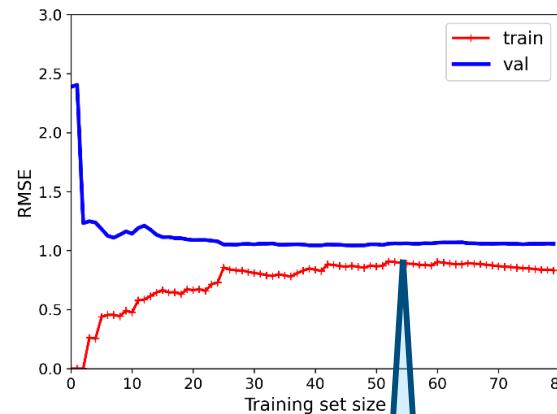
ML Supervisado - Curvas de aprendizaje

Una manera de evaluar la calidad de nuestra función hipótesis (modelo aprendido) es mediante la denominadas **curvas de aprendizaje**.

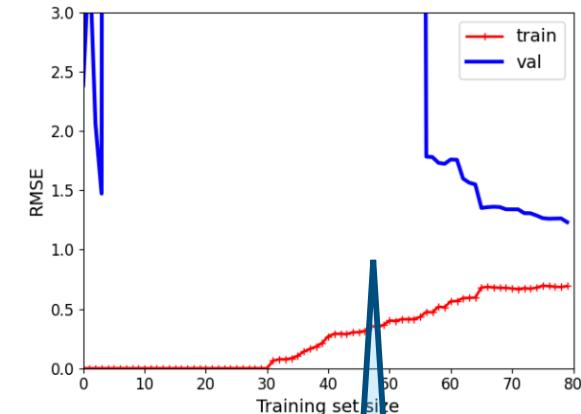
Estas **curvas representan gráficamente el error mínimo cuadrado (RMSE)** contrastándolo entre el **conjunto de datos de entrenamiento y el de validación**.



Si ambas curvas convergen en un error alto, estamos en presencia de underfitting. Generaliza de acuerdo al aprendizaje pero con un alto grado de error



Si ambas curvas convergen en un error moderado/bajo, estamos en ajustando correctamente el modelo



Si las curvas presentan mucha distancia entre ellas, estamos en presencia de overfitting. Se ajusta muy bien a los datos del entrenamiento pero no generaliza

ML Supervisado - SESGO vs VARIANZA

Un resultado teórico importante de la estadística y el aprendizaje automático es que el error de generalización de un modelo puede expresarse como la suma de tres tipos de errores muy diferentes:

Sesgo (Bias): Esta parte del error de generalización se debe a suposiciones incorrectas, como asumir que los datos son lineales cuando en realidad son cuadráticos. Un modelo con alto sesgo probablemente presentará underfitting en los datos de entrenamiento.

Varianza (Variance): Esta parte se debe a la excesiva sensibilidad del modelo ante pequeñas variaciones en los datos de entrenamiento. Un modelo con muchos grados de libertad seguramente tendrá alta varianza y, por lo tanto, overfitting.

Error irreducible (Irreducible error): Esta parte se debe al ruido inherente de los datos. La única manera de reducir este error es sanitizar los datos (por ejemplo, eliminando valores atípicos).

En general, aumentar la complejidad de un modelo suele incrementar su varianza y reducir su sesgo. Reducir la complejidad tiende a aumentar el sesgo y disminuir la varianza.

ML Supervisado - Regularización

Una buena manera de reducir el **overfitting** es mediante la **regularización** del modelo, es decir, imponerle restricciones. Una forma sencilla de regularizar un modelo polinómico es reducir el número de grados del polinomio. La regularización se logra normalmente restringiendo los pesos del modelo:

Ridge Regression: Penaliza la suma de los cuadrados de los pesos del modelo.

Fórmula de penalización: $cost(h_{\theta}) = MSE(h_{\theta}) + \alpha \sum_i \theta_i^2$

Los pesos grandes se reducen hacia cero, pero nunca se vuelven exactamente cero. No elimina variables irrelevantes, solo las reduce.

Lasso Regression (Least Absolute Shrinkage and Selection Operator) : Penaliza la suma de los valores absolutos de los pesos.

Fórmula de penalización: $cost(h_{\theta}) = MSE(h_{\theta}) + \alpha \sum_i |\theta_i|$

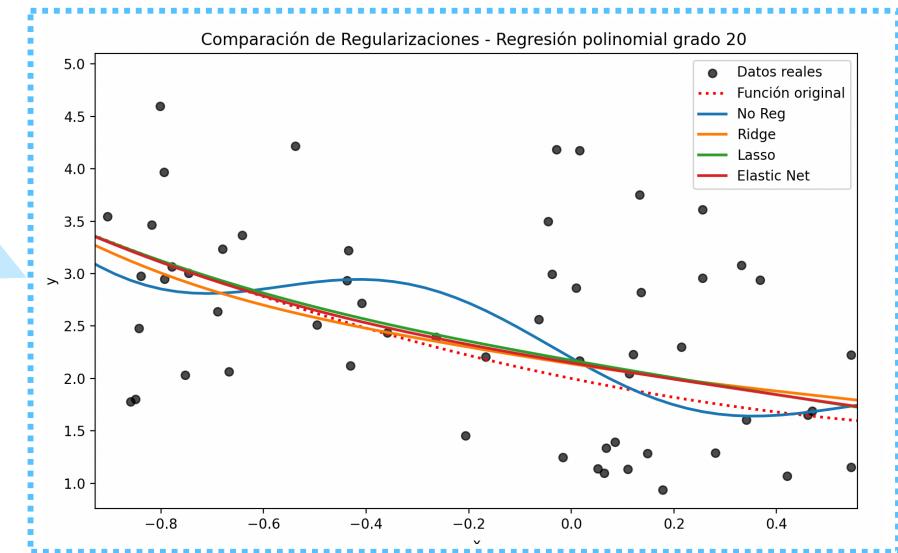
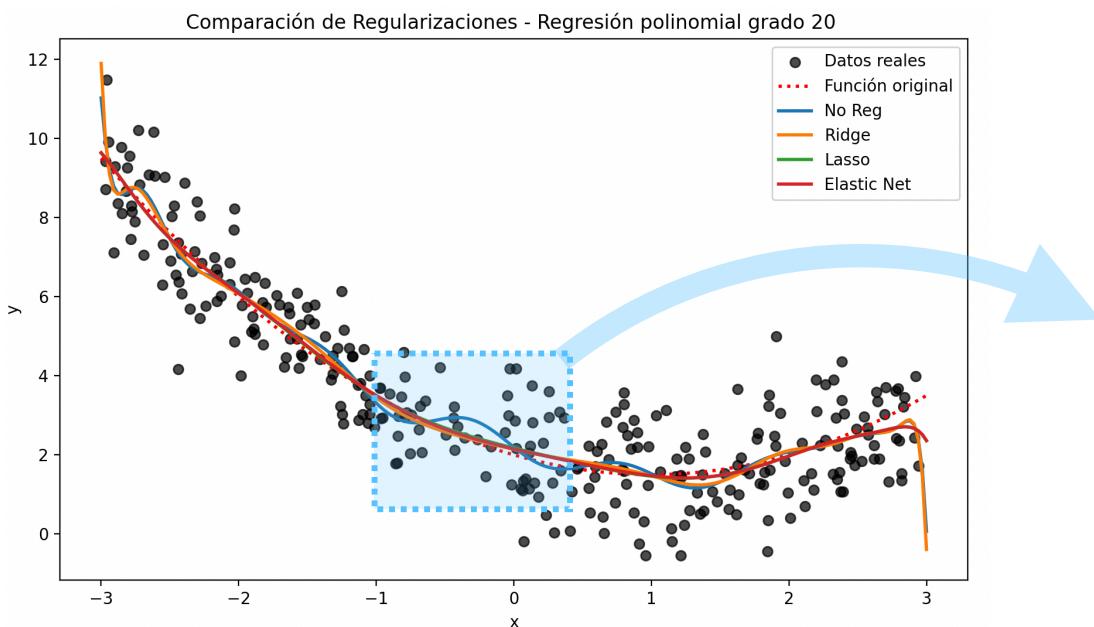
Tiende a reducir algunos pesos exactamente a cero, quita la variable asociada del modelo. Hace selección automática de variables y simplifica el modelo. Menos estable cuando hay muchas variables correlacionadas.

ML Supervisado - Regularización

Una técnica de **regularización** que combina **Ridge** y **Lasso** es **Elastic Net Regression**:

Fórmula de penalización: $cost(h_\theta) = MSE(h_\theta) + r\alpha \sum_{i=1}^n |\theta_i| + (1 - r)\alpha \sum_{i=1}^n \theta_i^2$

La primera componente de la fórmula (como Lasso) elimina variables irrelevantes. La segunda (como Ridge) maneja bien la multicolinealidad. Funciona bien cuando hay muchas variables correlacionadas y cuando queremos al mismo tiempo regularizar y seleccionar variables.

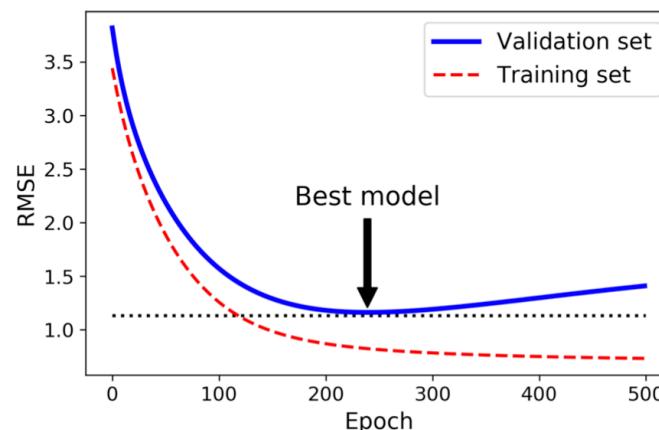


ML Supervisado - Regularización - Uso

- Casi siempre es preferible usar algún grado de regularización
- **Ridge** suele ser una buena opción por defecto
- Si se sospecha que pocas características son útiles, se prefiere **Lasso** o **Elastic Net**, ya que reducen los pesos de las características innecesarias a cero.
- **Elastic Net** suele ser preferible respecto Lasso. Lasso NO suele funcionar bien cuando el número de características es mayor al número de instancias de entrenamiento, o cuando hay una correlación fuerte entre algunas características
- De todos modos, como es típico en ML, en la práctica usualmente se pueden **probar distintas alternativas** experimentalmente, y elegir la que funcione mejor para el problema.

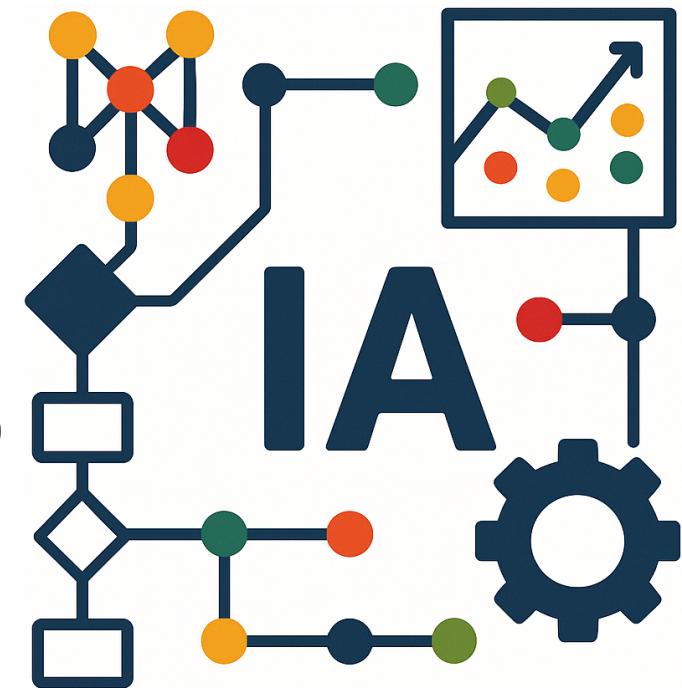
ML Supervisado - Finalización Temprana (Early Stopping)

- Cuando el entrenamiento es iterativo (Estocástico por Gradiente), el modelo que da mejores resultados en el conjunto de validación *puede* aparecer en **iteraciones intermedias** durante el entrenamiento.
- Esto se debe a que en las últimas iteraciones se suele hacer **overfitting** sobre el conjunto de entrenamiento.
- **Early stopping** consiste en guardar el modelo con mejores resultados contrastados con el conjunto de datos de validación.



Inteligencia Artificial

Machine Learning - (Aprendizaje Automático)
Un caso de estudio



Machine Learning - La importancia de los datos



The Unreasonable Effectiveness of Data

Alon Halevy, Peter Norvig, and Fernando Pereira, Google

Eugene Wigner's article "The Unreasonable Effectiveness of Mathematics in the Natural Sciences"¹ examines why so much of physics can be neatly explained with simple mathematical formulas

such as $f = ma$ or $e = mc^2$. Meanwhile, sciences that involve human beings rather than elementary particles have proven more resistant to elegant mathematics. Economists suffer from physics envy over

behavior. So, this corpus could serve as the basis of a complete model for certain tasks—if only we knew how to extract the model from the data.

Learning from Text at Web Scale

The biggest successes in natural-language-related machine learning have been statistical speech recognition and statistical machine translation. The reason for these successes is not that these tasks are easier than other tasks; they are in fact much harder

En muchos problemas de IA, tener más y mejores datos es más importante que usar modelos más complejos

A. Halevy, P. Norvig and F. Pereira, "The Unreasonable Effectiveness of Data," in *IEEE Intelligent Systems*, vol. 24, no. 2, pp. 8-12, March-April 2009, doi: 10.1109/MIS.2009.36.

Machine Learning - La importancia de los datos

Quizás el ejemplo más famoso de sesgo de muestreo ocurrió durante las elecciones presidenciales de EE. UU. en 1936, en las que se enfrentaron Landon y Roosevelt: la revista **Literary Digest** realizó una encuesta muy grande, enviando cuestionarios por correo a unas **10 millones de personas**. Obtuvo **2,4 millones de respuestas** y predijo con gran confianza que **Landon obtendría el 57 %** de los votos. Sin embargo, **Roosevelt ganó con el 62 %**.

El error estuvo en el método de muestreo de Literary Digest:

Fuentes de direcciones sesgadas: Para obtener las direcciones a las que enviar las encuestas, Literary Digest usó directorios telefónicos, listas de suscriptores de revistas, listas de miembros de clubes, etc. Todas estas listas tendían a favorecer a personas con cierta posición económica, quienes tenían mayor probabilidad de votar al Partido Republicano (y, por ende, a Landon).

Baja tasa de respuesta (menos del 25 %): Menos de una cuarta parte de las personas encuestadas respondió. Esto introdujo un sesgo adicional, conocido como sesgo por no respuesta (**nonresponse bias**), ya que se excluía potencialmente a personas que no estaban interesadas en la política, que no simpatizaban con Literary Digest u otros grupos clave.

Machine Learning - Acerca de los modelos

El Teorema **No Free Lunch (NFL)** enunciado por *David Wolpert* en 1996, establece que no existe un modelo de aprendizaje automático que sea el mejor para todos los problemas. Cada modelo funciona bien solo en ciertos tipos de datos y mal en otros. Por ejemplo, un modelo lineal es adecuado si la relación entre las variables es lineal, pero no lo será si la relación es no lineal y compleja.

En la práctica, esto significa que **siempre estamos haciendo suposiciones implícitas sobre la naturaleza de los datos al elegir un modelo**. Como no es posible evaluar todos los modelos posibles, lo habitual es probar solo algunos que resulten razonables para el problema.

Machine Learning - Proceso de modelado y utilización

- 1. Problem Definition:** Definir claramente qué problema se quiere resolver, los objetivos y las métricas a utilizar.
- 2. Data Collection:** Obtener los datos necesarios desde las fuentes disponibles.
- 3. Data Cleaning and Preprocessing:** Corregir errores, manejar valores faltantes, normalizar y transformar los datos para que el modelo los pueda usar.
- 4. Exploratory Data Analysis:** Analizar visual y estadísticamente los datos para entender patrones, tendencias y posibles problemas.
- 5. Feature Engineering and Selection:** Crear nuevas características relevantes y seleccionar las más útiles para mejorar el rendimiento del modelo.
- 6. Model Selection:** Elegir el tipo de modelo que probablemente funcione mejor según el problema y los datos.
- 7. Model Training:** Entrenar el modelo con los datos de entrenamiento para que aprenda patrones.
- 8. Model Evaluation and Tuning:** Evaluar el rendimiento en datos de prueba y ajustar hiperparámetros para optimizarlo.
- 9. Model Deployment and Maintenance:** Poner el modelo en producción para que empiece a usarse.

Machine Learning - Recolección de Datos

En esta etapa nos concentraremos en reunir de forma sistemática los conjuntos de datos que servirán como materia prima para entrenar el modelo. Durante la recolección, es fundamental garantizar que los datos **sean relevantes para el problema**, y que contengan **todas las características necesarias**.

Aspectos clave de la Recolección de Datos:

Relevancia: Reunir datos directamente relacionados con el problema incluyendo las características esenciales.

Calidad: Mantener la precisión, consistencia y cumplir con **estándares éticos**.

Cantidad: Asegurar un volumen suficiente de datos para entrenar un modelo.

Diversidad: Incluir conjuntos de datos variados para capturar una amplia gama de escenarios y patrones.

Machine Learning - Sanitización de Datos

Una vez recolectados los datos es necesario un proceso de sanitización, que involucra su estructuración, limpieza (descartando partes irrelevantes para el modelo), proveer consistencia, detectar datos faltantes, casos atípicos (outliers), etc.

Limpieza de Datos: Abordar problemas como valores faltantes, valores atípicos e inconsistencias en los datos.

Pre-procesamiento de Datos: Estandarizar formatos, escalar valores y codificar variables categóricas para lograr coherencia.

Calidad de Datos: Asegurar que los datos estén bien organizados y listos para un análisis significativo.

Machine Learning - Exploración de Datos

Para descubrir información y comprender la estructura del conjunto de datos, por ejemplo, encontrar patrones y características ocultas en los datos, se utiliza el Análisis Exploratorio de Datos. Las visualizaciones ayudan a presentar la información de forma comprensible.

Exploración: Utilizar herramientas estadísticas y visuales para explorar patrones en los datos.

Patrones y Tendencias: Identificar patrones subyacentes, tendencias y posibles desafíos dentro del conjunto de datos.

Perspectivas: Obtener información valiosa para la toma de decisiones en etapas posteriores, por ejemplo la selección de modelos.

Machine Learning - Selección e ingeniería de Características

Esta etapa consiste en seleccionar únicamente las características relevantes para la predicción del modelo. La ingeniería de características implica seleccionar características relevantes o crear nuevas características transformando las existentes para la predicción con el fin de mejorar la precisión y minimizar la complejidad computacional.

Ingeniería de Características: Crear nuevas características o transformar las existentes para capturar mejor patrones y relaciones.

Selección de Características: Identificar el subconjunto de características más relevantes para el modelo.

Optimización: Equilibrar el conjunto de características para maximizar la precisión y minimizar la complejidad computacional.

Machine Learning - Selección del modelo

La selección del modelo es una decisión clave que determina el marco algorítmico para la predicción. La elección depende de la naturaleza de los datos, la complejidad del problema y los resultados deseados.

Alineación: Elegir un modelo que se ajuste al problema definido y a las características del conjunto de datos.

Complejidad: Considerar la complejidad del problema y la naturaleza de los datos al seleccionar un modelo.

Factores de decisión: Evaluar aspectos como el rendimiento y la escalabilidad al elegir un modelo.

Experimentación: Probar diferentes modelos para encontrar el que mejor se adapte al problema.

Machine Learning - Entrenamiento

El entrenamiento del modelo es un **proceso iterativo** en el que el algoritmo ajusta sus parámetros para minimizar errores y mejorar la precisión predictiva. Un proceso de entrenamiento exitoso asegura que el modelo funcione bien con datos nuevos, garantizando predicciones confiables en escenarios reales.

Datos de entrenamiento: Determinar estrategias y porciones de datos para el entrenamiento.

Proceso iterativo: Entrenar el modelo de forma iterativa, ajustando parámetros para minimizar errores y mejorar la precisión (control de underfitting y overfitting).

Validación: Contrastar rigurosamente el modelo para asegurar su precisión con datos nuevos (generalización).

Machine Learning - Evaluación

La evaluación es fundamental para obtener información sobre las fortalezas y debilidades del modelo. Si el modelo no alcanza los niveles de rendimiento deseados, puede ser necesario volver a ajustarlo y modificar sus hiperparámetros para mejorar su precisión predictiva. Este ciclo iterativo de evaluación y ajuste es crucial para lograr el nivel de robustez y fiabilidad deseado en el modelo.

Métricas de evaluación: Utilizar métricas como MSE, RMSE, precisión, exhaustividad, etc. para evaluar el rendimiento del modelo, identificando fortalezas y debilidades.

Mejora iterativa: Realizar ajustes en los hiperparámetros para mejorar el modelo hasta alcanzar los niveles de confiabilidad deseados.

Machine Learning - Implementación (Deploy)

Una vez obtenido el modelo con una evaluación exitosa, el modelo está listo para su despliegue en aplicaciones del mundo real. El despliegue del modelo implica integrarlo en los sistemas existentes, permitiendo que las organizaciones utilicen sus predicciones para tomar decisiones informadas.

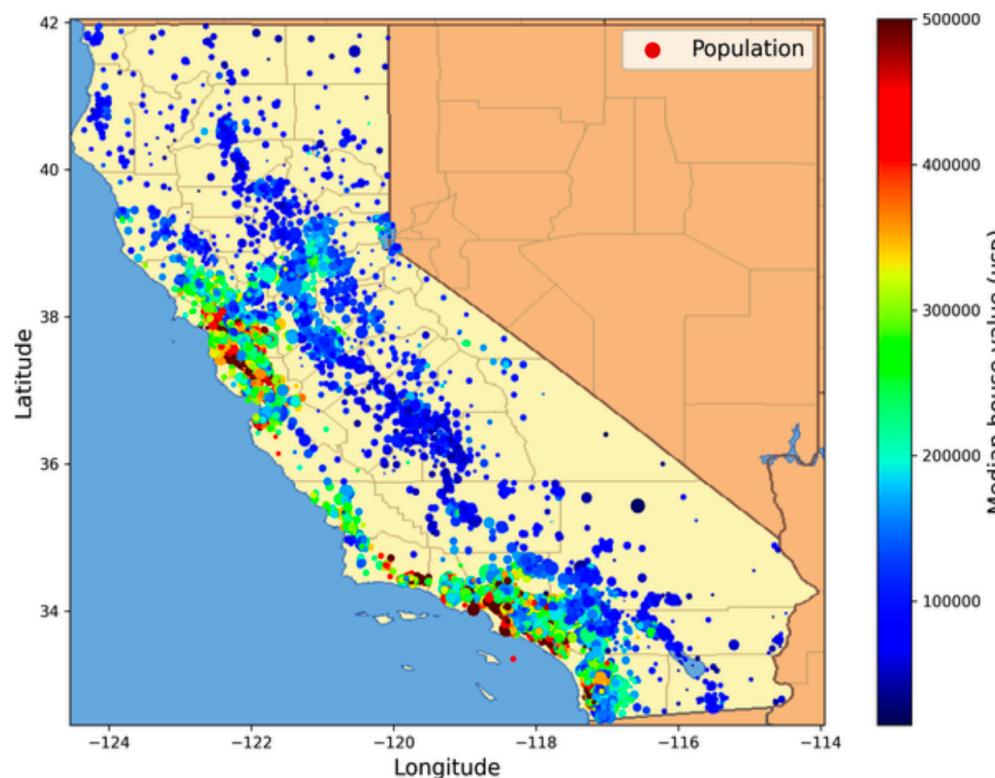
Integración: Incorporar el modelo entrenado en sistemas o procesos existentes para su uso en entornos reales.

Uso del modelo: Utilizar las predicciones del modelo para fundamentar decisiones informadas.

Mejora continua: Supervisar el rendimiento del modelo y realizar ajustes según sea necesario para mantener su eficacia a lo largo del tiempo.

Caso de Estudio: California Housing Prices

Se dispone de datos acerca del costo de propiedades (casas) en California y se espera crear un modelo que pueda predecir los valores de las casas según sus características, como lugar (geo-posición), salario promedio, dimensiones, valor, etc.



Caso de Estudio: California Housing Prices

Es un problema de **regresión múltiple** ya que el sistema utilizará **múltiples características** para realizar una predicción. También es un problema de regresión **univariante**, ya que **solo intentamos predecir el valor** para cada distrito. Si intentáramos predecir múltiples valores por distrito, sería un problema de regresión multivariante.

Una decisión a tomar es cuál será la métrica para determinar el aprendizaje:

Root Mean Squared Error (RMSE): También denominada norma euclídea o norma l_2

$$RMSE(X, h) = \sqrt{\frac{1}{m} \sum_1^m (h(X_i) - y_i)^2}$$

Mean Absolute Error (MAE): También denominada norma de manhattan o norma l_1

$$MAE(X, h) = \frac{1}{m} \sum_1^m |h(X_i) - y_i|$$

En general, la norma l_k de un vector se define como $l_k = (|v_0|^k + \dots + |v_n|^k)^{1/k}$. Normas más grandes ponen más énfasis en los valores grandes y menos en los valores pequeños (más propenso a ser afectado por valores atípicos).

Caso de Estudio: California Housing Prices

Comencemos las etapas de obtención de datos, su análisis, exploración y sanitización. Vamos utilizar un repositorio que cuenta con la información en formato csv (comma separated values).

```
from pathlib import Path
import pandas as pd
import tarfile
import urllib.request

def load_housing_data():
    tarball_path = Path("datasets/housing.tgz")
    if not tarball_path.is_file():
        Path("datasets").mkdir(parents=True, exist_ok=True)
        url = "https://github.com/ageron/data/raw/main/housing.tgz"
        urllib.request.urlretrieve(url, tarball_path)
    with tarfile.open(tarball_path) as housing_tarball:
        housing_tarball.extractall(path="datasets")
    return pd.read_csv(Path("datasets/housing/housing.csv"))

housing = load_housing_data()
```

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY

Caso de Estudio: California Housing Prices

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY

Cada fila es la información de un distrito que contiene 10 características:

- **longitude**: Longitud geográfica del distrito.
- **latitude**: Latitud geográfica del distrito.
- **housing_median_age**: Mediana sobre la edad de las viviendas.
- **total_rooms**: Número total de habitaciones en el distrito.
- **total_bedrooms**: Número total de dormitorios en el distrito.
- **population**: Población total del distrito.
- **households**: Número total de hogares en el distrito.
- **median_income**: Ingreso medio por hogar en el distrito.
- **median_house_value**: **Valor medio de las viviendas (objetivo de la predicción)**.
- **ocean_proximity**: Proximidad al océano (característica por categorías).

Mediana: se ordenan los valores y se toma el central o el promedio de los valores centrales. Es más representativa en presencia de valores atípicos (outliers)

Caso de Estudio: California Housing Prices

longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY

Algunas observaciones:

- **Información faltante**: Por ejemplo para la cantidad de dormitorios
- **ocean_proximity**: Proximidad al océano (característica por categorías) no tiene valores numéricos, sino un **enumerado**: *1H OCEAN, INLAND, NEAR OCEAN, NEAR BAY, ISLAND*. En estas situaciones podemos codificarlos para un modelo usando:
 - **One-hot encoding** (crear variables binarias para cada categoría).
 - **Ordinal encoding** (si hubiera un orden lógico, que aquí no lo hay).

Caso de Estudio: California Housing Prices

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

Promedio (media): Suma de todos los valores dividido entre el número de datos. La media es sensible a la distribución es simétrica, pero es sensible a outliers.

Desviación estándar: Mide cuánto se dispersan los datos alrededor de la media. La desviación estándar es sensible a outliers.

Mínimo y máximo: Valores más pequeño y más grande en el conjunto de datos.

Cuartiles (25, 50, 75): Dividen los datos ordenados en cuartiles.

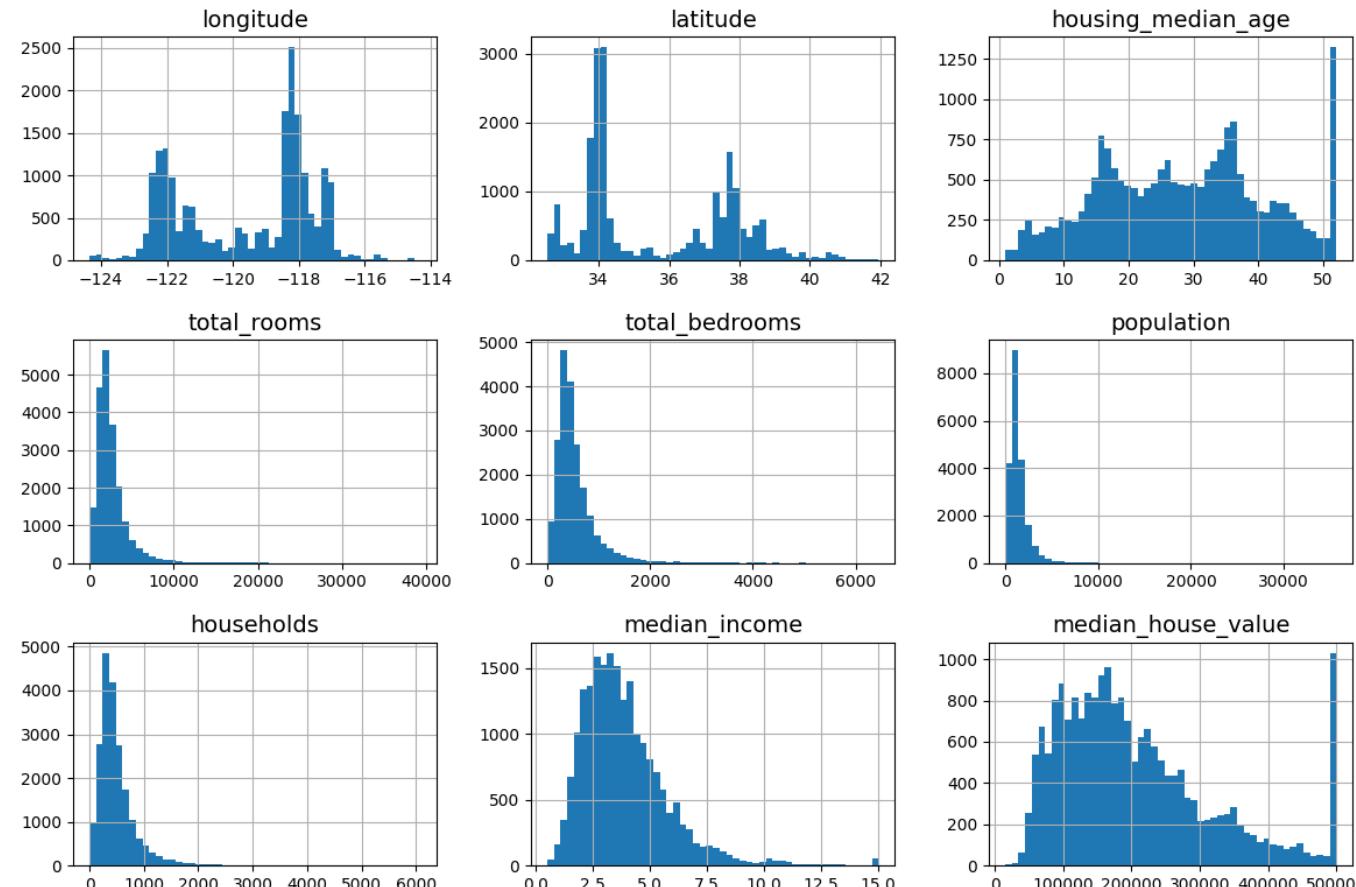
Q1 (25%): Valor por debajo del cual está el 25% de los datos.

Q2 (50%): Valor central que separa en dos mitades los datos.

Q3 (75%): Valor por debajo del cual está el 75% de los datos.

- Faltan algunos datos para la feature total_bedrooms
- *median_income*: está expresado en decenas de miles de dólares, y restringido a un rango entre 0,5 y 15
- *median_age* y *median_house_value* también están restringidos
 - *median_house_value* es lo que queremos predecir y nuestro algoritmo podría aprender que nunca puede superar los \$500.000
- Si queremos predecir mejor valores > \$500.000, tenemos dos opciones:
 - Mejorar la recolección de datos
 - Eliminar los distritos con valor \$500.000 del conjunto de datos
- Las características tienen escalas muy diferentes

Caso de Estudio: California Housing Prices



```
import matplotlib.pyplot as plt  
housing.hist(bins=50, figsize=(12, 8))  
plt.show()
```

- Las características tienen escalas muy diferentes
- Los histogramas se extienden más hacia la derecha de la media que hacia la izquierda. Esto podría hacer que algunos algoritmos de ML tengan dificultad en aprender patrones
- Podemos **escalar** las características para que tengan una distribución normal

Caso de Estudio: California Housing Prices

Una vez analizados los datos y sus características, dependiendo de la metodología de aprendizaje evaluación, debemos separar aquellos datos que vamos a utilizar para evaluar el/los modelos aprendidos.

Un error común es inspeccionar o seleccionar “manualmente” dicho conjunto. Esto derivará seguramente en un sesgo ya que podemos suponer ciertos patrones o criterios por conocer los datos de evaluación (**data snooping bias**).

Para este caso de estudio, podemos utilizar por ejemplo un **muestreo aleatorio**:

```
import numpy as np

def shuffle_and_split_data(data, test_ratio):
    shuffled_indices = np.random.permutation(len(data))
    test_set_size = int(len(data) * test_ratio)
    test_indices = shuffled_indices[:test_set_size]
    train_indices = shuffled_indices[test_set_size:]
    return data.iloc[train_indices],
           data.iloc[test_indices]

train_set, test_set = shuffle_and_split_data(housing, 0.2)
len(train_set)
len(test_set)
```

Una forma similar pero con algunas mejoras es utilizar:

```
from sklearn.model_selection import
train_test_split

train_set, test_set =
train_test_split(housing,
                test_size=0.2, random_state=42)
```

Caso de Estudio: California Housing Prices

Si el conjunto de datos no es lo suficientemente grande, corremos el riesgo de cometer **sesgo por muestreo** en nuestra selección, por ejemplo que en la selección aleatoria, no se corresponda adecuadamente (con cierta proporción) la cantidad de muestras respecto a ciertas características. Para minimizar los riegos de sesgo por muestreo (selección de datos) podemos utilizar **muestreo estratificado**.

Es una técnica utilizada para garantizar que una muestra (como el conjunto de prueba) refleje la composición general de la población en relación con ciertas características importantes.

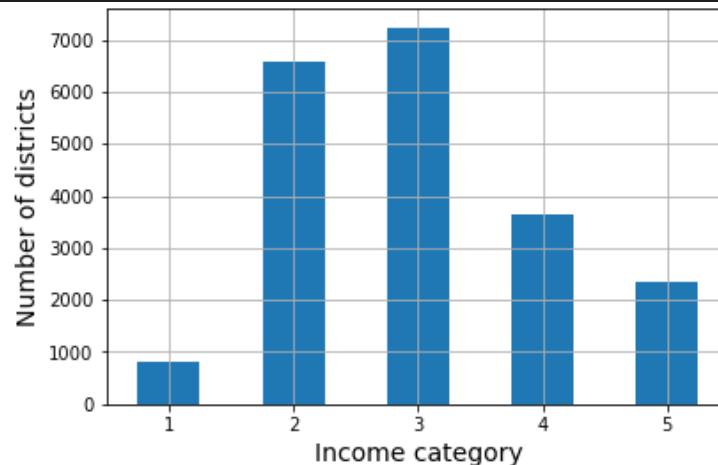
- La población se divide en **subgrupos homogéneos** llamados **estratos**.
- De cada estrato, se toma un **número proporcional de instancias**, de manera que la muestra conserve la misma distribución que el conjunto de datos completo.

Caso de Estudio: California Housing Prices

Por ejemplo, respecto de la categoría de sueldo promedio (relevante para nuestro problema), podemos categorizar (estratificar) en 5 categorías:

```
housing["income_cat"] = pd.cut(housing["median_income"],  
                                bins=[0., 1.5, 3.0, 4.5, 6., np.inf],  
                                labels=[1, 2, 3, 4, 5])
```

```
housing["income_cat"].value_counts().sort_index().plot.bar(rot=0,  
grid=True)  
plt.xlabel("Income category")  
plt.ylabel("Number of districts")  
plt.show()
```



Caso de Estudio: California Housing Prices

Para utilizar estas categorías en la selección del conjunto de prueba podemos utilizar:

```
strat_train_set, strat_test_set = train_test_split(  
    housing, test_size=0.2, stratify=housing["income_cat"], random_state=42)
```

Comparamos los dos tipos de selecciones focalizados en las categorías creadas

Income Category	Overall %	Stratified %	Random %	Strat. Error %	Rand. Error %
1	3.98	4.00	4.24	0.36	6.45
2	31.88	31.88	30.74	-0.02	-3.59
3	35.06	35.05	34.52	-0.01	-1.53
4	17.63	17.64	18.41	0.03	4.42
5	11.44	11.43	12.09	-0.08	5.63

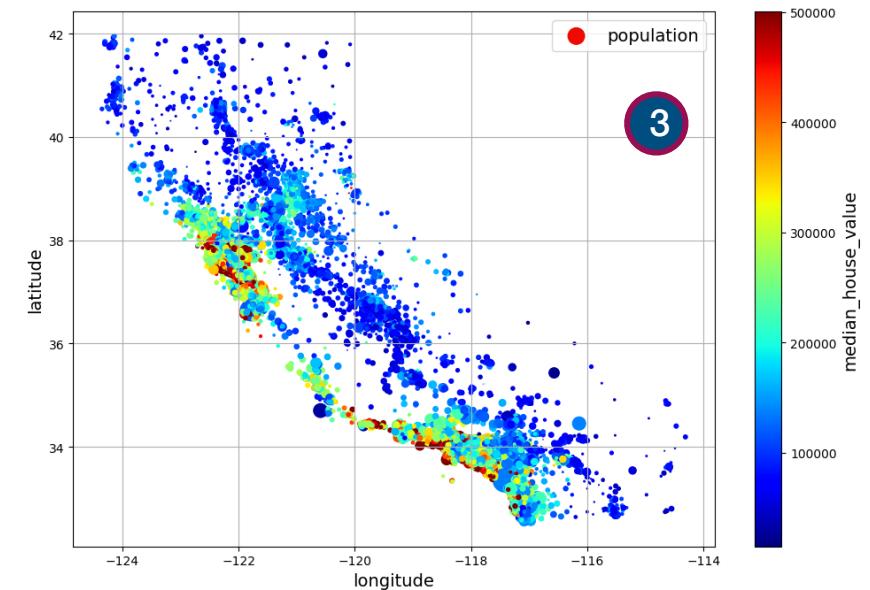
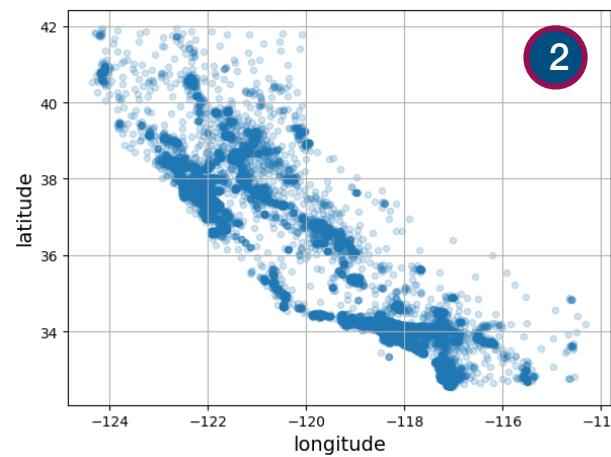
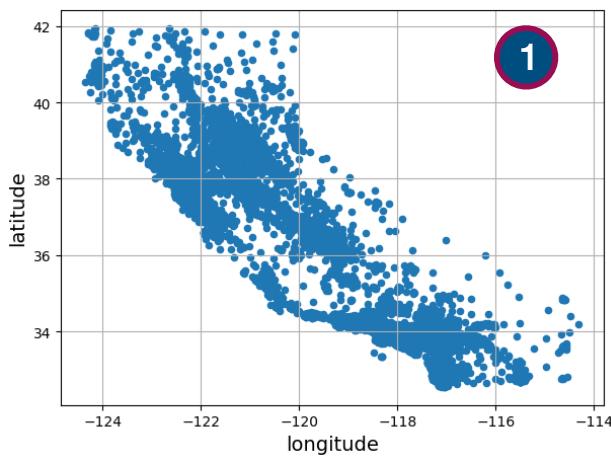
Una vez elegidos organizados ambos conjuntos, debemos quitar esa característica creada sólo al efecto de dividir los conjuntos de aprendizaje y prueba.

```
for set_ in (strat_train_set, strat_test_set):  
    set_.drop("income_cat", axis=1, inplace=True)
```

Caso de Estudio: California Housing Prices

En todo momento podemos visualizar la información:

- 1 housing.plot(kind="scatter", x="longitude", y="latitude", grid=True)
- 2 housing.plot(kind="scatter", x="longitude", y="latitude", grid=True, alpha=0.2)
- 3 housing.plot(kind="scatter", x="longitude", y="latitude", grid=True,
s=housing["population"] / 100, label="population",
c="median_house_value", cmap="jet", colorbar=True,
legend=True, sharex=False, figsize=(10, 7))
plt.show()

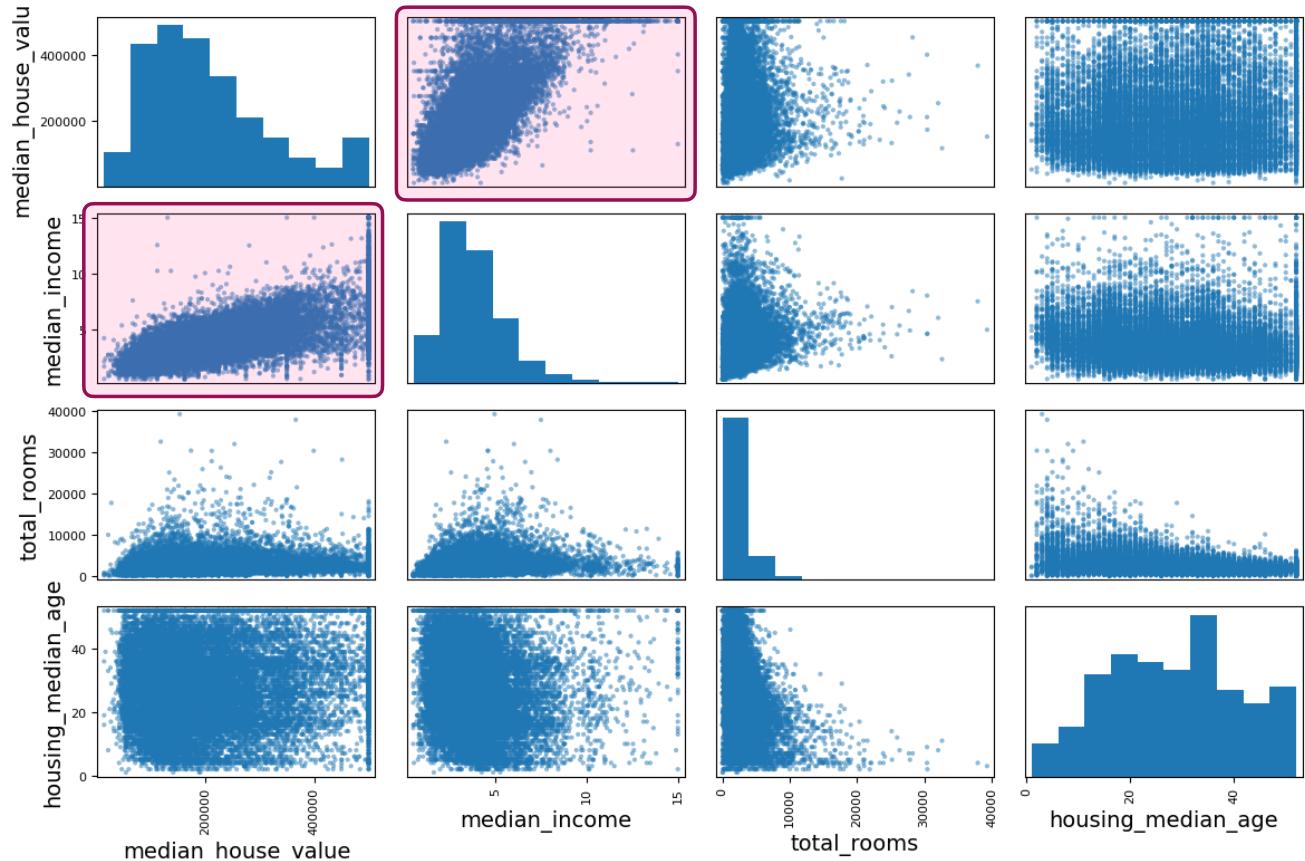


Caso de Estudio: California Housing Prices

```
from pandas.plotting import scatter_matrix

attributes = ["median_house_value", "median_income", "total_rooms", "housing_median_age"]
scatter_matrix(housing[attributes], figsize=(12, 8))
plt.show()
```

Otra mirada exploratoria es la búsqueda de **correlaciones**, por ejemplo, **Pearson's r** (coeficiente de correlación de Pearson) es una medida estadística que indica la fuerza y dirección de la relación lineal entre dos variables cuantitativas. Podemos contrastar esta medida entre las características de los datos a modelar.



Caso de Estudio: California Housing Prices

Esta medida nos da valores entre **-1** y **1** con la siguiente interpretación:

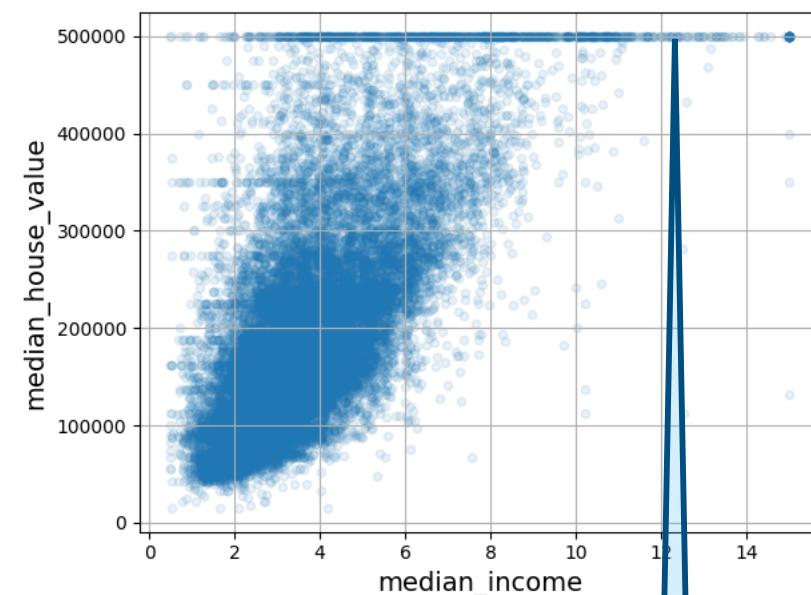
r = 1: correlación positiva perfecta (una sube, la otra sube proporcionalmente).

r = -1: correlación negativa perfecta (una sube, la otra baja proporcionalmente).

r = 0: no hay correlación lineal.

```
corr_matrix = housing.corr(numeric_only=True)
corr_matrix[ "median_house_value" ].sort_values(ascending=False)
```

	median_house_value
median_house_value	1.000000
median_income	0.688075
total_rooms	0.134153
housing_median_age	0.105623
households	0.065843
total_bedrooms	0.049686
population	-0.024650
longitude	-0.045967
latitude	-0.144160



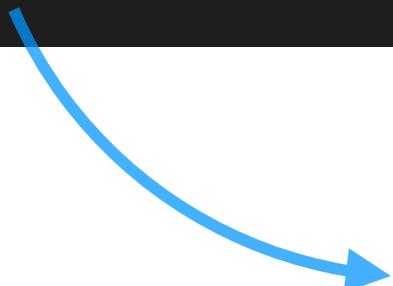
Los datos presentan límites y valores repetidos (p. ej., \$500.000, \$450.000, \$350.000) que evidencian ruido y podrían requerir la eliminación de distritos afectados o la mejora en su recolección para evitar que el modelo aprenda estos patrones.

Caso de Estudio: California Housing Prices

Durante el proceso de exploración, podemos derivar más características, por ejemplo:

```
housing[ "rooms_per_house" ] = housing[ "total_rooms" ] / housing[ "households" ]
housing[ "bedrooms_ratio" ] = housing[ "total_bedrooms" ] / housing[ "total_rooms" ]
housing[ "people_per_house" ] = housing[ "population" ] / housing[ "households" ]

corr_matrix = housing.corr(numeric_only=True)
corr_matrix[ "median_house_value" ].sort_values(ascending=False)
```



	median_house_value
median_house_value	1.000000
median_income	0.688380
rooms_per_house	0.143663
total_rooms	0.137455
housing_median_age	0.102175
households	0.071426
total_bedrooms	0.054635
population	-0.020153
people_per_house	-0.038224
longitude	-0.050859
latitude	-0.139584
bedrooms_ratio	-0.256397

Caso de Estudio: California Housing Prices

Una vez explorados los datos, comenzamos su preparación previa al entrenamiento.

Habíamos notado la ausencia del valor **total_bedrooms** en algunos distritos.

Tenemos tres opciones:

- Eliminar los distritos correspondientes
- Eliminar el atributo completo
- Asignarle algún valor a los distritos con problemas (cero, la media, la mediana, etc.)

```
housing.dropna(subset=[ "total_bedrooms" ], inplace=True)      # option 1
housing.drop("total_bedrooms", axis=1)                      # option 2
median = housing[ "total_bedrooms" ].median()    # option 3
housing[ "total_bedrooms" ].fillna(median, inplace=True)
```

Otra forma más eficiente es utilizando las implementaciones provistas por **Scikit-Learn**, que facilitan el reuso de las *transformaciones*, y sus sucesivas aplicaciones a distintos datasets (ej. a partes del dataset, al test set, etc.)

Caso de Estudio: California Housing Prices

Podemos usar el *transformer* Imputer de Scikit-Learn para llenar los datos faltantes con la **media**.

Primero eliminaremos el atributo *ocean_proximity* porque la media no se puede computar para atributos enumerados, luego creamos un Imputer que compute las medias del dataset con el método *fit()*. Finalmente aplicamos la transformación reemplazando los valores faltantes por la **media**.

```
from sklearn.impute import SimpleImputer

housing_num = housing.select_dtypes(include=[np.number]) #drop ocean_proximity

imputer = SimpleImputer(strategy="median")

imputer.fit(housing_num)

imputer.statistics_

X = imputer.transform(housing_num)
```

Scikit-Learn: API

Estimadores: Son objetos que aprenden parámetros a partir de un conjunto de datos usando el método `fit()`. Si es aprendizaje supervisado, reciben también las etiquetas. Los hiperparámetros de configuración (por ejemplo, `strategy` en `SimpleImputer`) y se definen al crear el objeto.

Transformadores: Estimadores que además pueden transformar datos mediante `transform()`, usando los parámetros aprendidos con `fit()`. Tienen el método `fit_transform()` para hacer ambos pasos de forma optimizada.

Predictores: Estimadores que pueden generar predicciones con `predict()` y evaluar su desempeño con `score()`.

Buenas prácticas de diseño:

- Los hiperparámetros y parámetros aprendidos son accesibles como atributos públicos (los aprendidos llevan guion bajo, ej. `statistics_`).
- Los datos se representan con arrays NumPy o matrices SciPy, no con clases personalizadas.
- Se fomenta la composición, como crear **pipelines** que encadenen transformadores y un estimador final.
- Se ofrecen valores por defecto razonables para facilitar la creación rápida de modelos funcionales.

Caso de Estudio: California Housing Prices

Dado que todos los algoritmos de Machine Learning funcionan mejor con números, debemos tratar la característica *ocean_proximity*, que es un enumerado.

En general tenemos dos opciones:

- Asignarle un valor entero diferente a cada valor enumerado, esto viene implementado por el transformador **OrdinalEncoder** de Scikit-Learn. Debemos tener precaución de no introducir alguna correlación o alteración a los algoritmos.
- Codificar los valores del atributo enumerado como **bitsets**, esto viene implementado por el transformador **OneHotEncoder** de Scikit-Learn.

```
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import OneHotEncoder

ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)

cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
```

Caso de Estudio: California Housing Prices

Una de las transformaciones más importantes en el preprocesamiento de datos es la normalización o escalado de características (**feature scaling**). La mayoría de los algoritmos de ML funcionan mal si las variables numéricas tienen **escalas muy diferentes**.

En el ejemplo de datos de viviendas, el número total de habitaciones varía de aproximadamente 6 a 39.320, mientras que el ingreso medio solo va de 0 a 15 (**Sin escalado, muchos modelos tenderán a ignorar el ingreso medio y dar más peso al número de habitaciones**).

Las dos técnicas más comunes para igualar la escala de todas las variables son:

- **Min-max scaling (normalización)**: ajusta los valores para que queden dentro de un rango fijo, normalmente [0, 1]. Mantiene la forma de la distribución original y es útil para algoritmos que asumen rangos acotados (ej. redes neuronales). Sin embargo, es muy sensible a *outliers*: un solo valor extremo cambia la escala de todo.
- **Estandarización (standardization)**: transforma los datos para que tengan media 0 y desviación estándar 1, restando a cada valor la media del conjunto de datos y lo divide por la desviación estándar. Si bien su rango no es fijo, no se ve afectada por la escala original, es más robusta ante valores atípicos y es más útil para algoritmos que asumen datos centrados (SVM, PCA, regresión logística, redes neuronales).

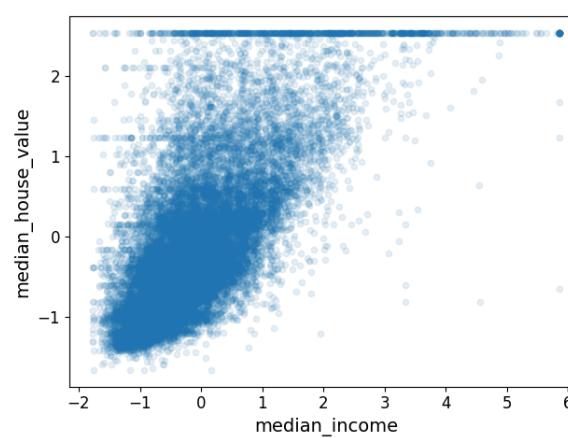
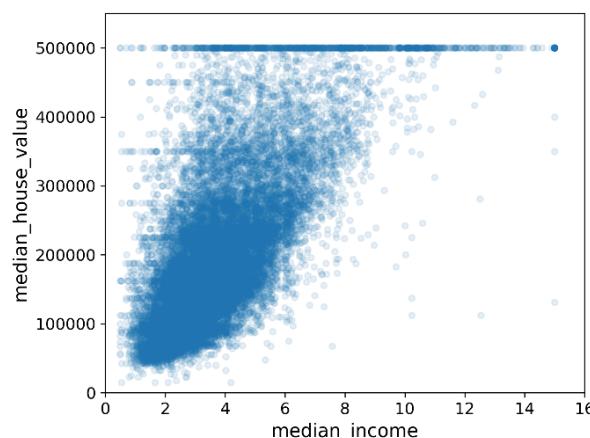
Caso de Estudio: California Housing Prices

Podemos utilizar los métodos provistos por Scikit-Learn para aplicar diferentes tipos de escalados necesario en nuestro conjunto de datos:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler

min_max_scaler = MinMaxScaler(feature_range=(-1, 1))
housing_num_min_max_scaled = min_max_scaler.fit_transform(housing_num)

std_scaler = StandardScaler()
housing_num_std_scaled = std_scaler.fit_transform(housing_num)
```



Scikit-Learn: Pipelines de Transformación

Como vimos, hay muchos pasos de transformación de datos que deben ejecutarse en el orden correcto. Scikit-Learn proporciona la clase **Pipeline** para ayudar con este tipo de secuencias de transformaciones.

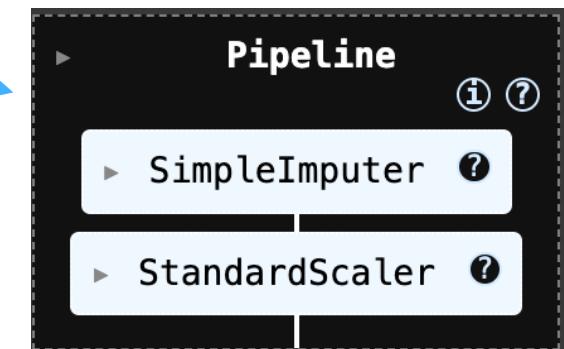
```
from sklearn.pipeline import make_pipeline
from sklearn import set_config
from sklearn.compose import ColumnTransformer

num_pipeline = make_pipeline(SimpleImputer(strategy="median"), StandardScaler())
set_config(display='diagram')
num_pipeline

num_attribs = ["longitude", "latitude", "housing_median_age", "total_rooms",
              "total_bedrooms", "population", "households", "median_income"]
cat_attribs = ["ocean_proximity"]

cat_pipeline = make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OneHotEncoder(handle_unknown="ignore"))

preprocessing = ColumnTransformer([
    ("num", num_pipeline, num_attribs),
    ("cat", cat_pipeline, cat_attribs),
])
housing_prepared = preprocessing.fit_transform(housing)
```



Datos listos para el aprendizaje

Caso de Estudio: California Housing Prices

Una vez preparados los datos, es hora del entrenamiento:

```
from sklearn.linear_model import LinearRegression  
  
lin_reg = make_pipeline(preprocessing, LinearRegression())  
lin_reg.fit(housing, housing_labels)
```

Podemos probar sus predicciones contrastándolas con los valores de entrenamiento:

```
housing_predictions = lin_reg.predict(housing)  
housing_predictions[:5].round(-2) # -2 = rounded to the nearest hundred  
housing_labels.iloc[:5].values  
  
lin_rmse = root_mean_squared_error(housing_labels, housing_predictions)  
lin_rmse
```

```
array([242800., 375900., 127500., 99400., 324600.])
```

```
array([458300., 483800., 101700., 96100., 361800.])
```

```
-47.0%, -22.3%, 25.4%, 3.4%, -10.3% !!
```

```
68647.95 !!
```

Demasiado “lejos” seguramente
estamos en presencia de
underfitting

Caso de Estudio: California Housing Prices

Probemos con otro modelo:

```
from sklearn.tree import DecisionTreeRegressor  
  
tree_reg = make_pipeline(preprocessing, DecisionTreeRegressor(random_state=42))  
tree_reg.fit(housing, housing_labels)
```

Podemos probar sus predicciones contrastándolas con los valores de entrenamiento:

```
housing_predictions = tree_reg.predict(housing)  
tree_rmse = root_mean_squared_error(housing_labels, housing_predictions)  
tree_rmse
```

0.0 !!

Demasiado “cerca” seguramente
estamos en presencia de **overfitting**

Caso de Estudio: California Housing Prices

Probemos entrenando y validando con **Cross-Validation**:

```
from sklearn.model_selection import cross_val_score

tree_rmses = -cross_val_score(tree_reg, housing, housing_labels, scoring="neg_root_mean_squared_error", cv=10)
lin_rmses = -cross_val_score(lin_reg, housing, housing_labels, scoring="neg_root_mean_squared_error", cv=10)
pd.Series(tree_rmses).describe()
pd.Series(lin_rmses).describe()
```

En Scikit-Learn, el RMSE se devuelve como valor negativo porque la validación cruzada espera que valores mayores sean mejores. Por eso, hay que invertir el signo para obtener el RMSE real.

$k = 10$

	count	10.000000	count	10.000000
mean	66366.983603		mean	69847.923224
std	1976.844743		std	4078.407329
min	63557.655007		min	65659.761079
25%	65004.623899		25%	68088.799156
50%	65886.897085		50%	68697.591463
75%	68129.026040		75%	69800.966364
max	69530.301101		max	80685.254832

!!

Sigue siendo un modelo demasiado impreciso. Además confirma que el modelo “tree” entendido estaba en presencia de overfitting

Caso de Estudio: California Housing Prices

Una más y no jodemos más:

Analizaremos en detalles estos modelos

```
rom sklearn.ensemble import RandomForestRegressor  
  
forest_reg = make_pipeline(preprocessing,  
                           RandomForestRegressor(random_state=42))  
forest_rmses = -cross_val_score(forest_reg, housing, housing_labels,  
                               scoring="neg_root_mean_squared_error", cv=10)  
  
pd.Series(forest_rmses).describe()
```

count	10.000000
mean	46938.209246
std	1018.397196
min	45522.649195
25%	46291.334639
50%	47021.703303
75%	47321.521991
max	49140.832210

```
forest_reg.fit(housing, housing_labels)  
housing_predictions = forest_reg.predict(housing)  
forest_rmse = root_mean_squared_error(housing_labels, housing_predictions)  
forest_rmse
```

17521.56

Claramente este modelo es mejor que los anteriores

Caso de Estudio: California Housing Prices

Como lo mencionamos, entrenar un modelo para que funcione adecuadamente puede requerir de una gran cantidad de pruebas en búsqueda del mismo. Más aún, para cada modelo bajo prueba, pueden existir diferentes hiperparámetros.

Una vez seleccionados algunos modelos potencialmente adecuados, es hora del **Fine Tuning**. Esto es ajustar los hiperparámetros para lograr el mejor ajuste posible. Una forma de sistematizar este proceso de búsqueda que provee Scikit-Learn es GridSearchCV.

```
from sklearn.model_selection import GridSearchCV

full_pipeline = Pipeline([
    ("preprocessing", preprocessing),
    ("random_forest", RandomForestRegressor(random_state=42)),
])
param_grid = [
    {'preprocessing_geo_n_clusters': [5, 8, 10],
     'random_forest_max_features': [4, 6, 8]},
    {'preprocessing_geo_n_clusters': [10, 15],
     'random_forest_max_features': [6, 8, 10]},
]
grid_search = GridSearchCV(full_pipeline, param_grid, cv=3,
                           scoring='neg_root_mean_squared_error')
grid_search.fit(housing, housing_labels)
grid_search.best_params_
```

Valores para los **hiperparámetros**.

Esta funcionalidad realiza una **validación cruzada con todas las combinaciones de los valores propuestos y devuelve los hiperparámetros que obtuvieron el mejor rendimiento**

{'preprocessing_geo_n_clusters': 15,
 'random_forest_max_features': 6}

Caso de Estudio: California Housing Prices

Una vez obtenido un modelo, debemos probar su funcionamiento con los datos de prueba.

```
x_test = strat_test_set.drop("median_house_value", axis=1)
y_test = strat_test_set["median_house_value"].copy()

final_predictions = final_model.predict(X_test)

final_rmse = root_mean_squared_error(y_test, final_predictions)
print(final_rmse)
```

41556.05

Otra medida interesante a analizar es el **Intervalo de Confianza**, que nos informa el rango del valor bajo una cierta probabilidad (expectativa) de **confianza**.

```
from scipy import stats

def rmse(squared_errors):
    return np.sqrt(np.mean(squared_errors))

confidence = 0.95
squared_errors = (final_predictions - y_test) ** 2
boot_result = stats.bootstrap([squared_errors], rmse,
                               confidence_level=confidence, random_state=42)
rmse_lower, rmse_upper = boot_result.confidence_interval
```

[39629.60, 43816.66]

Caso de Estudio: California Housing Prices

Monitoreo del modelo

- Luego del deployment, idealmente se deben monitorear las predicciones que hace el modelo de ML sobre nuevos datos, y detectar rápidamente degradaciones en el rendimiento.
- Es posible que los datos con los que se consulta el modelo vayan cambiando con el paso del tiempo (data drift). Por ejemplo, una aplicación que detecta patrones en fotografías puede degradar su performance por diversos motivos: nuevas cámaras de mayor resolución, nuevos filtros, nuevos objetos de moda, etc.

Gestión y calidad de datos

- A medida que van ingresando nuevos datos para hacer consultas al modelo, almacenarlos y usarlos para reentrenar periódicamente.
- Si el rendimiento se degrada con los sucesivos entrenamientos, esto puede indicar problemas con la recolección de datos. También pueden indicar problemas en la recolección: detección de datos faltantes, outliers cada vez más frecuentes, etc.

Reentrenamiento y validación

- Antes del deployment de un modelo entrenado con nuevos datos, evaluar el rendimiento del modelo previo y del nuevo con el dataset completo. Sólo publicar si el nuevo modelo supera en performance al anterior.
- Crear scripts para automatizar esta tarea.

Seguridad y respaldo

- Hacer backups de los modelos anteriores para restaurarlos rápidamente si se detectan problemas (ej. una degradación) en el modelo actual.