



UNIVERSIDAD DE CUENCA  
desde 1867

# Introducción: ¿Qué es el aprendizaje automático?

Andres Auquilla  
2020



La gran variedad de términos y sus usos hacen que este campo luzca intimidante y muchas veces repetitivo

# Content

**Logística del curso**

Sílabo, reglas, etc.

**Concepto de Aprendizaje Automático**

Concepto, relaciones con otras áreas

**Tipos de aprendizajes**

Aprendizajes en Machine Learning

**Aplicaciones**

Estado del arte

# Content

Logística del curso  
Sílabo, reglas, etc.

Concepto de Aprendizaje Automático  
Concepto, relaciones con otras áreas

Tipos de aprendizajes  
Aprendizajes en Machine Learning

Aplicaciones  
Estado del arte

# Aspectos logísticos

## Instructor: Andrés Auquilla

- Email: [andres.auquilla@ucuenca.edu.ec](mailto:andres.auquilla@ucuenca.edu.ec) (incluir *Aprendizaje Automático* en el subject)
- Oficina: 307 edificio Promas (solo con cita previa)

## Clases:

- Lunes de 09:00 a 11:00
- Jueves de 15:00 a 17:00
- Si por alguna razón se atrasa, **solo entre sin hacer ruido**

## Prácticas:

- Habrán varias sesiones prácticas
- Ejercicios a papel y lápiz
- Ejercicios en la computadora (weka, Python, Knime)

# Mi Background

## Títulos:

- Ingeniero de sistemas – Universidad de Cuenca
- Master of Science of Artificial Intelligence – KU Leuven, Bélgica
- PhD in Engineering (c) – KU Leuven, Bélgica “*Personalized Products Through Anticipative Methods Based on Historical Information and System Profiles*”

## Background en Machine Learning:

- Aplicaciones industriales que hagan uso de energía
- Extracción de patrones en series de tiempo
- Creación de métodos predictivos con series de tiempo
- Patente: sistema de control adaptativo para un sistema de climatización inteligente

# ¿Cuál es su background?

En 5 minutos responder lo siguiente:

- ¿Alguna vez ha creado alguna aplicación inteligente?, ¿Cuál?
- ¿Qué aspectos del Aprendizaje Automático le interesan más?
- ¿Qué aplicación/investigación que ha leído, le ha parecido la más interesante?, ¿Por qué?

# Materiales de ayuda

Todos los materiales estarán disponibles en el evirtual

- **Machine Learning and Inductive Inference**
- The hundred-page Machine Learning Book
- Machine Learning Yearning

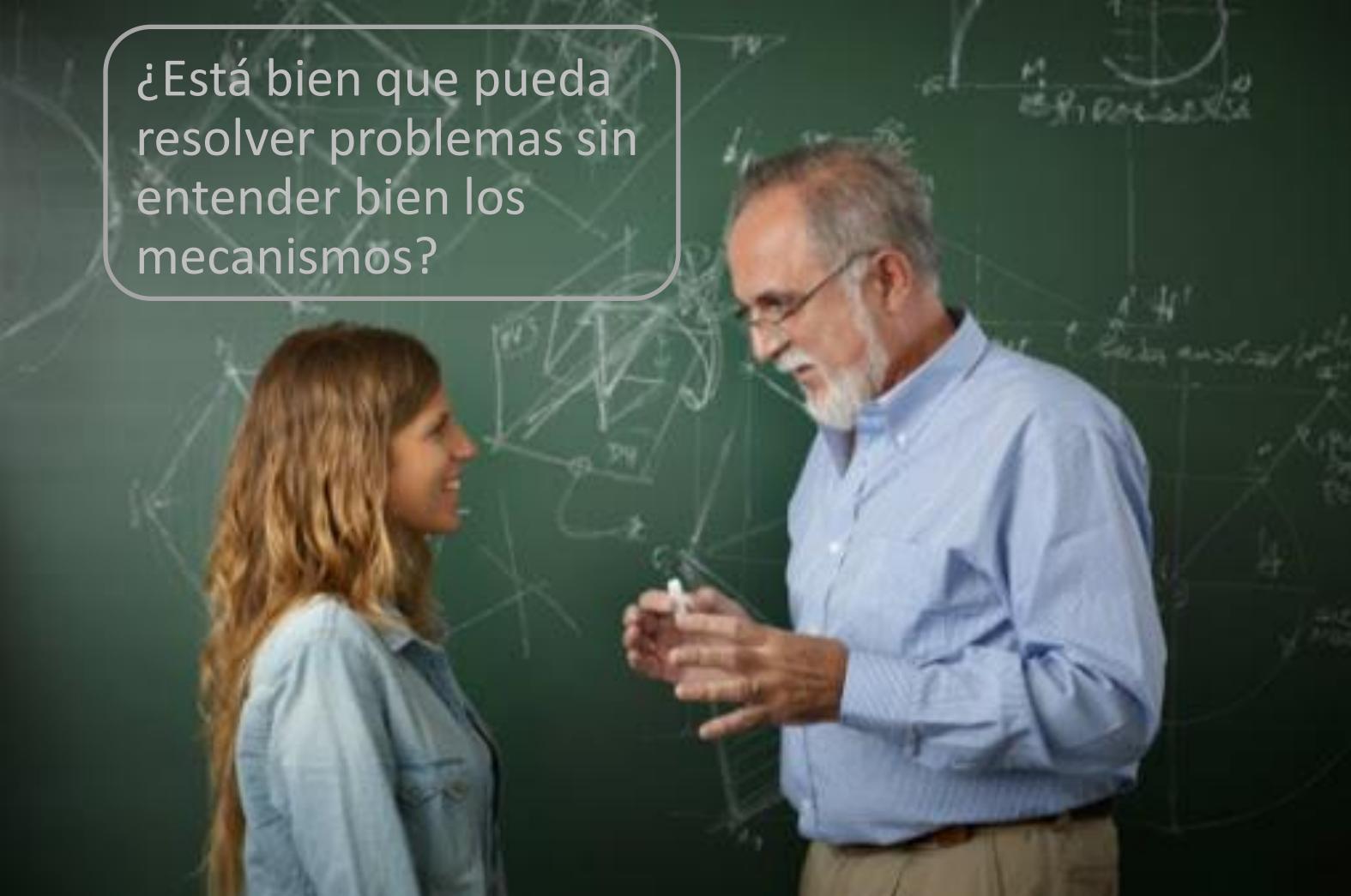
Los slides son material complementario, no principal

# Estilo del curso

El curso es mayormente teórico (especial énfasis en los tipos de aprendizaje)

Es importante entender los fundamentos matemáticos básicos detrás de las técnicas a utilizar

Analizar situaciones reales prácticas y proponer arquitecturas para solucionar problemas puntuales

A photograph of a teacher and a student in a classroom setting. The teacher, an older man with a beard and glasses, is gesturing with his hands while speaking. The student, a young woman with long hair, is smiling and looking at him. They are standing in front of a chalkboard covered with various diagrams and mathematical equations.

¿Está bien que pueda resolver problemas sin entender bien los mecanismos?



Si, si estudiase en un instituto  
técnico. Como esto es  
ingeniería, la respuesta es NO

# Estilo del curso

El curso es mayormente teórico (especial énfasis en los tipos de aprendizaje)

Es importante entender los fundamentos matemáticos básicos detrás de las técnicas a utilizar

Analizar situaciones reales prácticas y proponer arquitecturas para solucionar problemas puntuales

# Cómo tener éxito en el curso

## La clave del éxito

- Asistir a todas las sesiones de clases
- Asistir y resolver las sesiones de ejercicios
- Práctica y más práctica
- No tener miedo a probar técnicas o herramientas por su cuenta

## En clases

- No tenga miedo de hacer preguntas
- **Sea proactivo!**

# Contenido del curso

1. Introducción
2. Conceptos generales del aprendizaje
3. Aprendizaje supervisado
4. Aprendizaje no supervisado
5. Reinforcement learning
6. Métodos ensemble
7. Evaluación de la hipótesis
8. Aprendizaje probabilístico

# Content

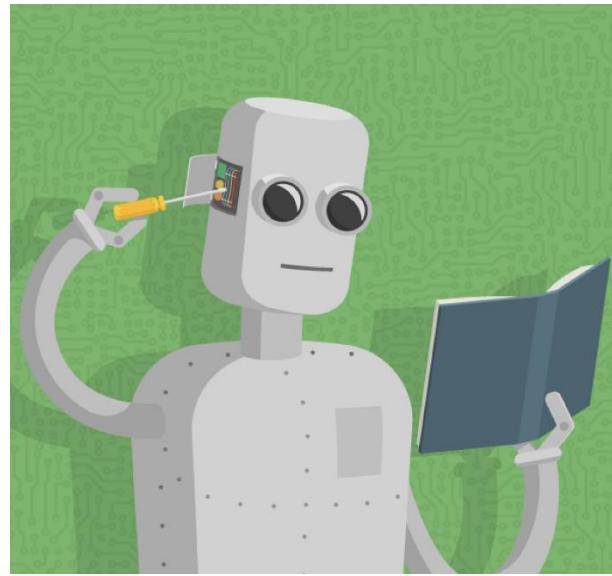
Logística del curso  
Sílabo, reglas, etc.

Concepto de Aprendizaje Automático  
Concepto, relaciones con otras áreas

Tipos de aprendizajes  
Aprendizajes en Machine Learning

Aplicaciones  
Estado del arte

# Inteligencia → Habilidad para aprender sin haber sido programado para ello



- Arthur Samuel (1959): Machine learning – campo de estudio que da a las computadoras la habilidad de aprender sin haber sido explícitamente programadas para eso
- Tom Mitchell (1998): un programa aprende por experiencia  $E$  con respecto a tareas  $T$  y medidas de rendimiento  $P$ , si su rendimiento en la tarea  $T$  mejora con la experiencia  $E$

# No todo lo Smart es realmente Smart



Luz + motion sensor



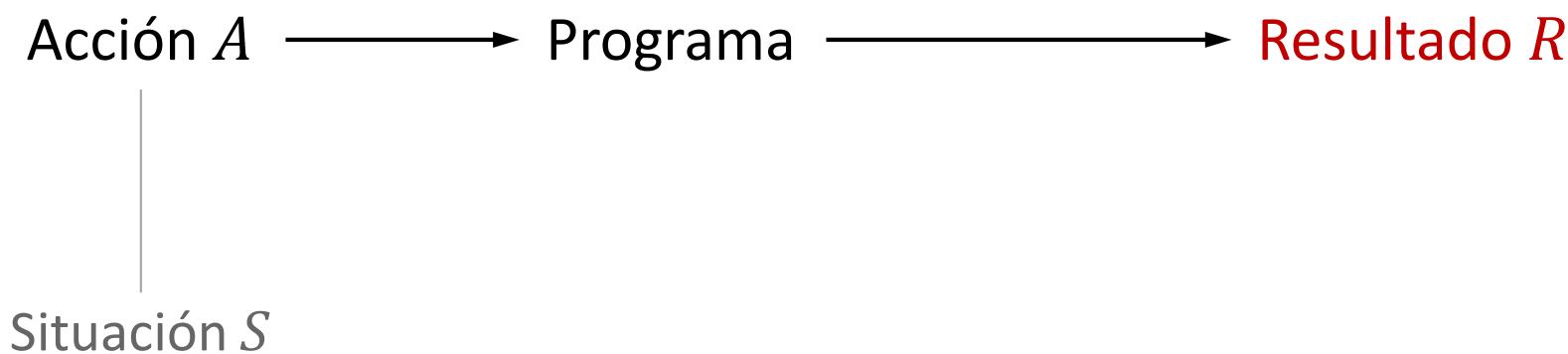
Smart watch



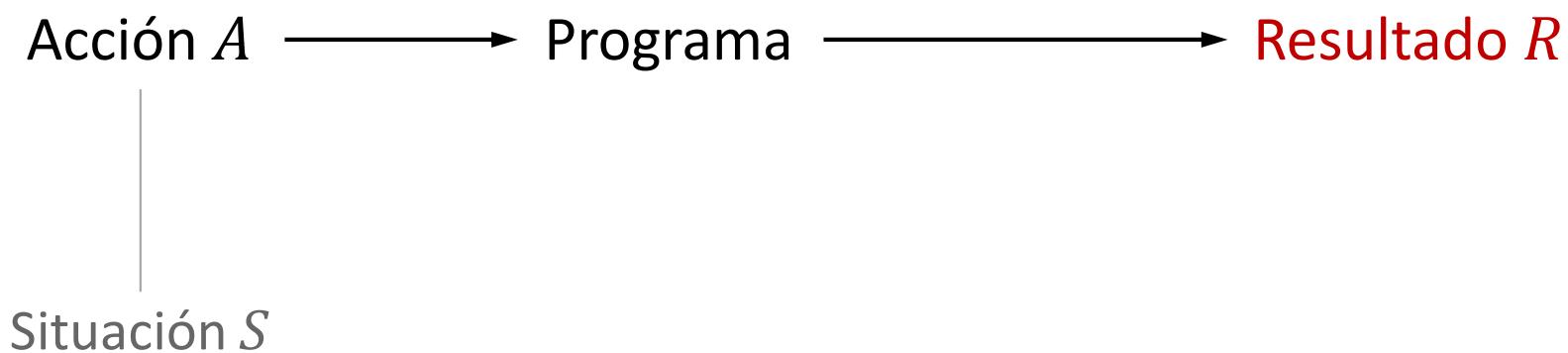
Aspirador inteligente

No son inteligentes por que fueron diseñados para cumplir tareas determinadas

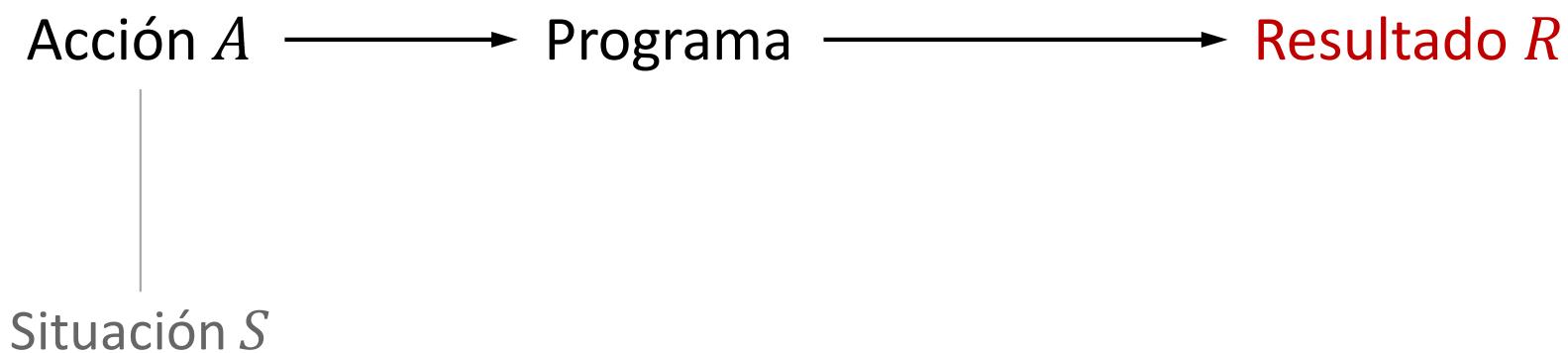
No aprenden a través de las interacciones/experiencias



Experiencia: Ejecutar una acción  $A$  en una situación  $S$ , hace que se dé un resultado  $R$



Aprendizaje: si  $R$  es deseable, realice la acción  $A$  nuevamente



Aprendizaje: si  $R$  no es deseable, realice otra acción

No siempre las acciones se realizan  
bajo las mismas situaciones

¿Qué sucede cuando existe una situación similar?  
Se necesita una **generalización**

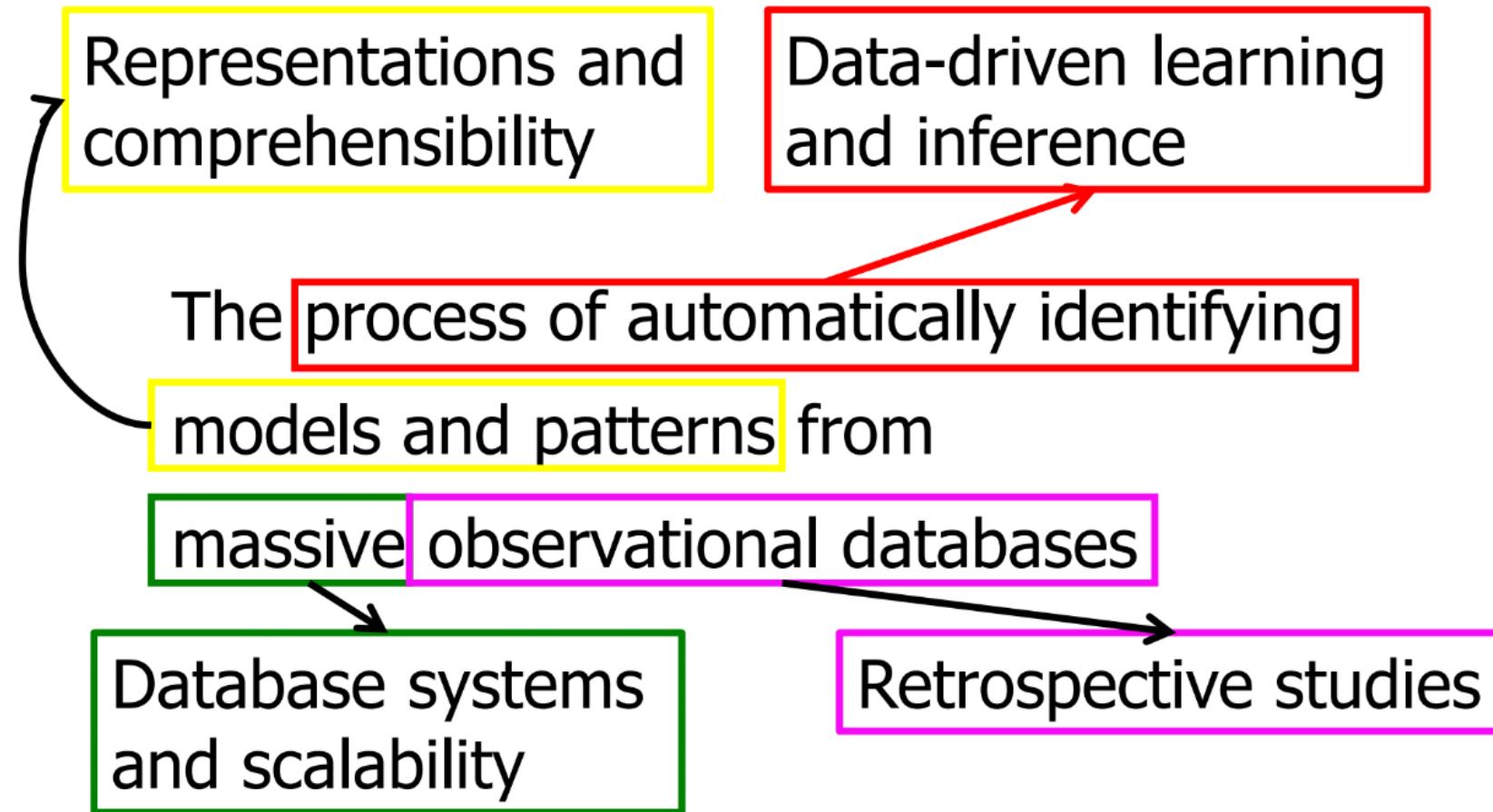
¿Y si se escoge otra acción, incluso si la previa era ya buena?  
Se necesita **exploración**

Este curso se centra mayormente en la **generalización**, a.k.a. **inferencia inductiva**

Data Mining es el proceso automático de identificar modelos y patrones de bases de datos observacionales

Las bases de datos son:

- Válidas: obtienen nuevos datos con cierta certeza
- Nuevas: no triviales
- Útiles: pueden ser utilizadas para ciertas tareas
- Entendibles: los humanos deberían poder interpretar sus patrones



# Data Mining tiene tres objetivos principales

## Entender los datos

Encontrar patrones útiles y no triviales

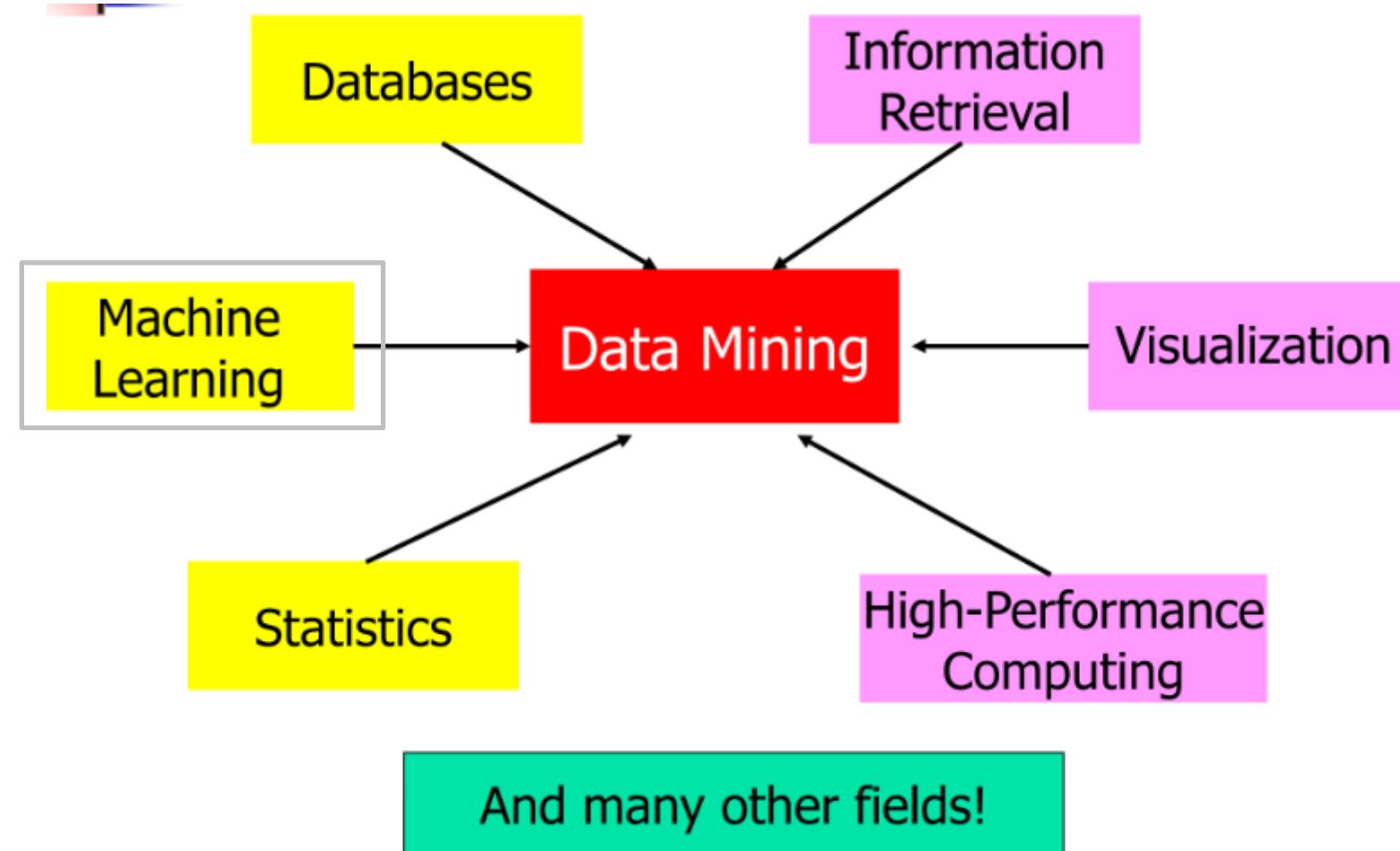
## Extraer conocimiento de los datos

Que variables son útiles, existen correlaciones, etc.

## Realizar predicciones del futuro

Series de tiempo, clasificaciones, regresiones, etc.

# Data Mining se cimenta en varias otras disciplinas



# Machine Learning vs Data Mining

Desde un punto de vista de alto nivel, son muy parecidas!

Data Mining se enfoca más en:

- Escalabilidad: los datos residen en bases de datos
- Aplicaciones: datos disponibles, necesidades
- Algoritmos: como se los utiliza dentro de la aplicación

Machine Learning se enfoca en:

- Cuestiones más teóricas
- Terminología usada mayormente en la academia/investigación
- Todo el ciclo necesario para que una aplicación aprenda a través de experiencias

# Machine Learning vs Estadística

La estadística tradicional se enfoca en:

- Generar una hipótesis, luego obtener datos, luego realizar pruebas y analizar
- Orientado al modelo

Machine Learning es diferente:

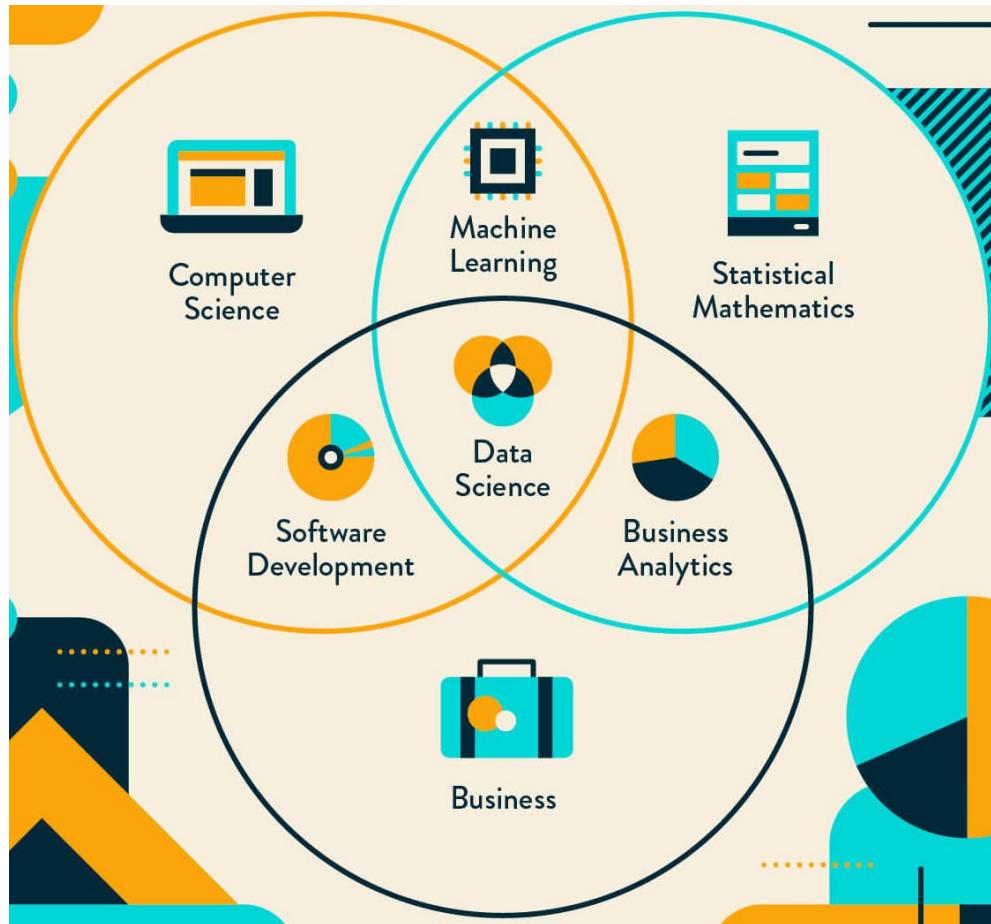
- Usualmente no se tiene una hipótesis
- Centrada en el análisis de datos ya existentes
- Orientado a los datos

En cuestiones de evaluación, las ideas estadísticas son muy importantes

# #10 years challenge

2009	2019
$Y = \beta X + \epsilon$	$Y = \beta X + \epsilon$
STATISTICS	MACHINE LEARNING
※ 10 YEARS CHALLENGE	

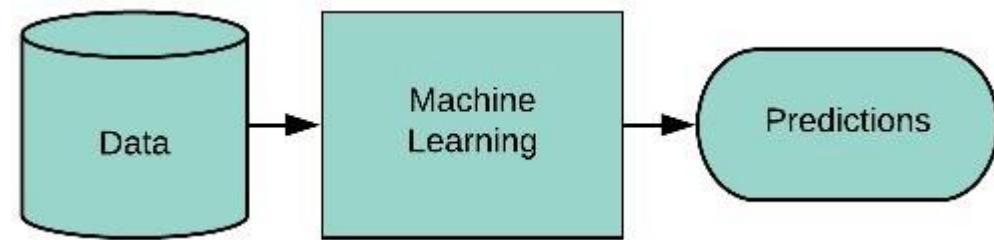
# Aún más complejidad: Data Science



Data Science: revelar y proveer conocimiento sobre datos disponibles

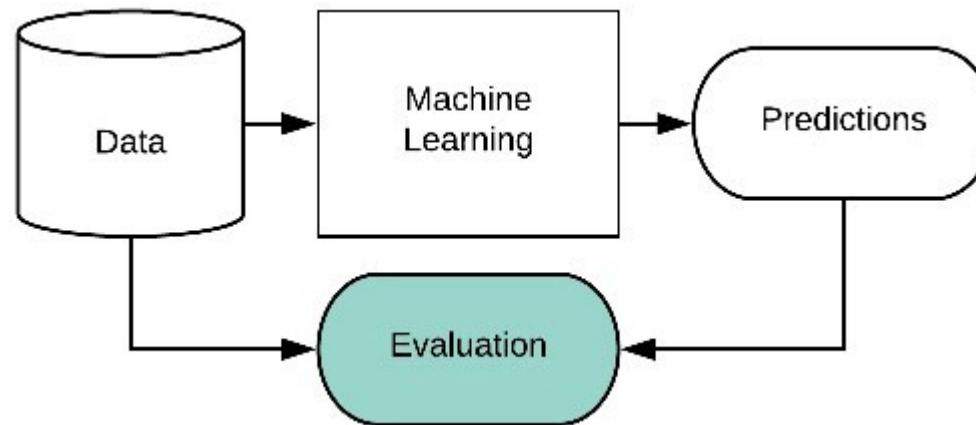
- Centrado en objetivos de usuarios
- Hace uso de varias otras disciplinas
- Sus resultados deben ser altamente interpretables

# Machine Learning es mucho más que crear un modelo que funcione



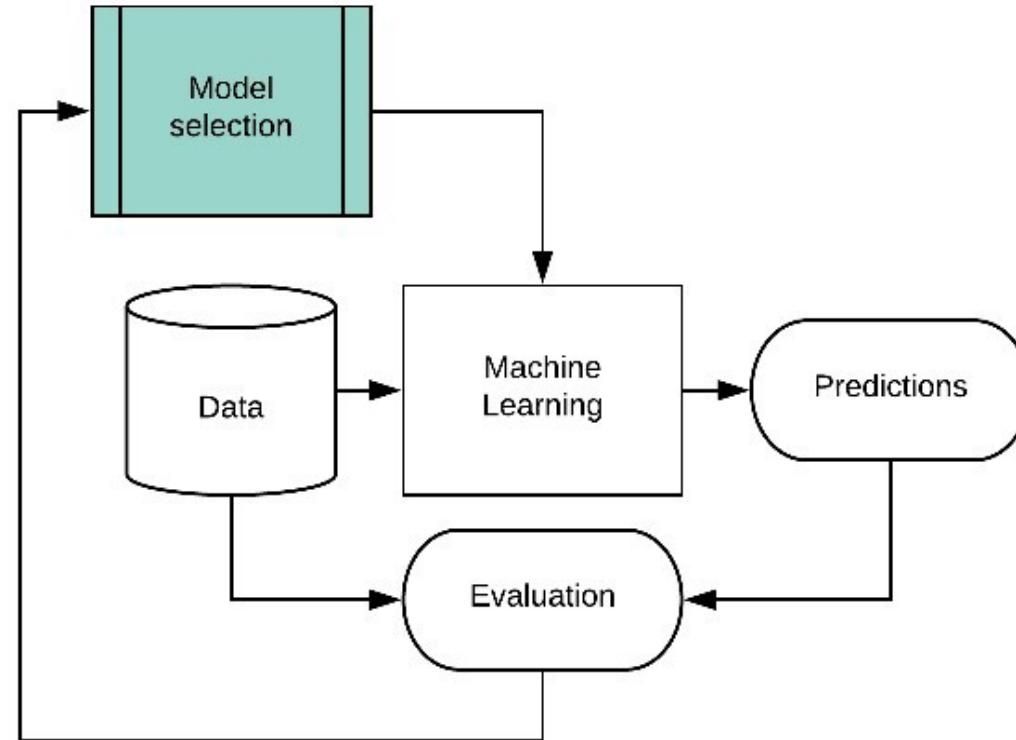
- Flujo simplista
- No hay garantía de que funcione
- Nivel: noob!

# Machine Learning es mucho más que crear un modelo que funcione



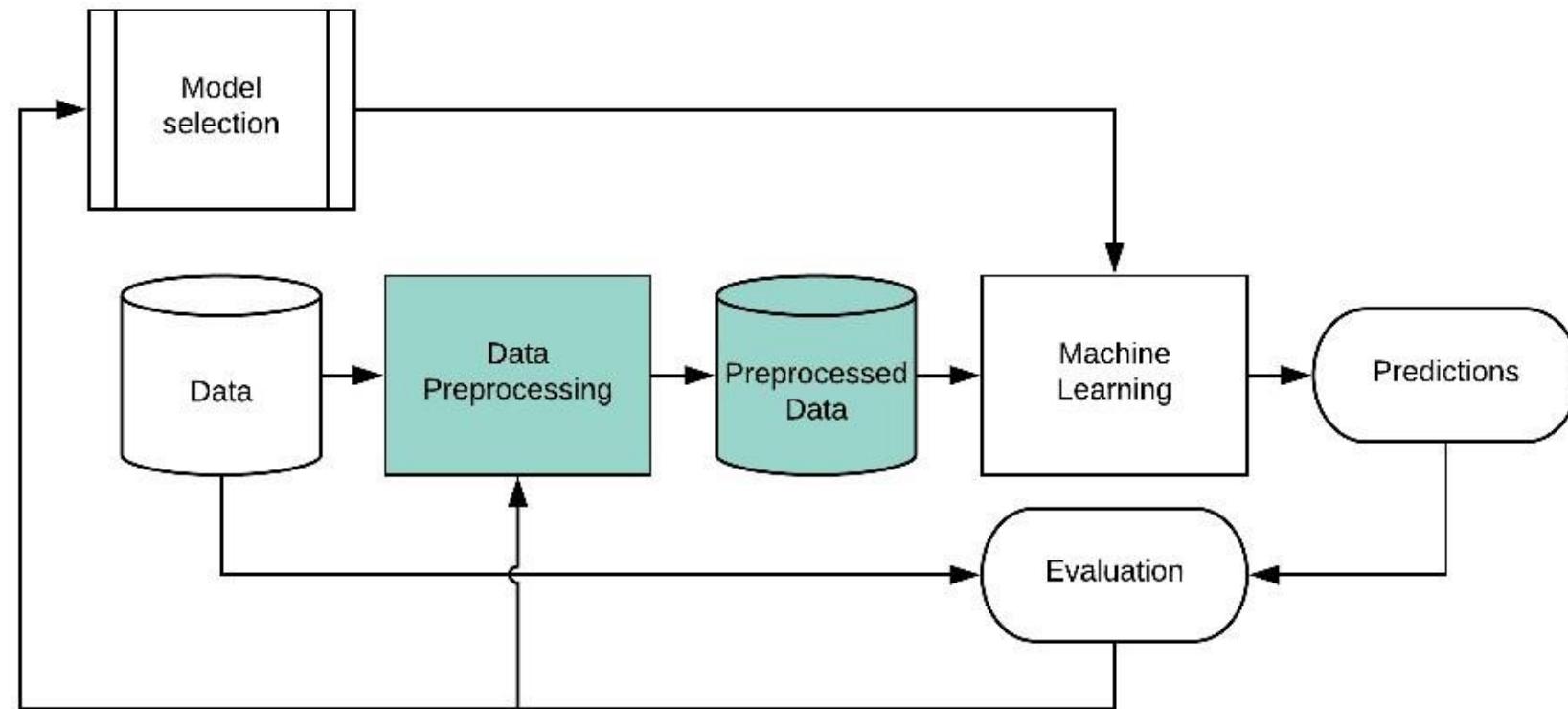
- Existe una certeza del nivel de generalización
- Mucho espacio para mejorar
- La mayoría de personas conoce ML a este nivel
- Nivel: todavía noob

# Machine Learning es mucho más que crear un modelo que funcione



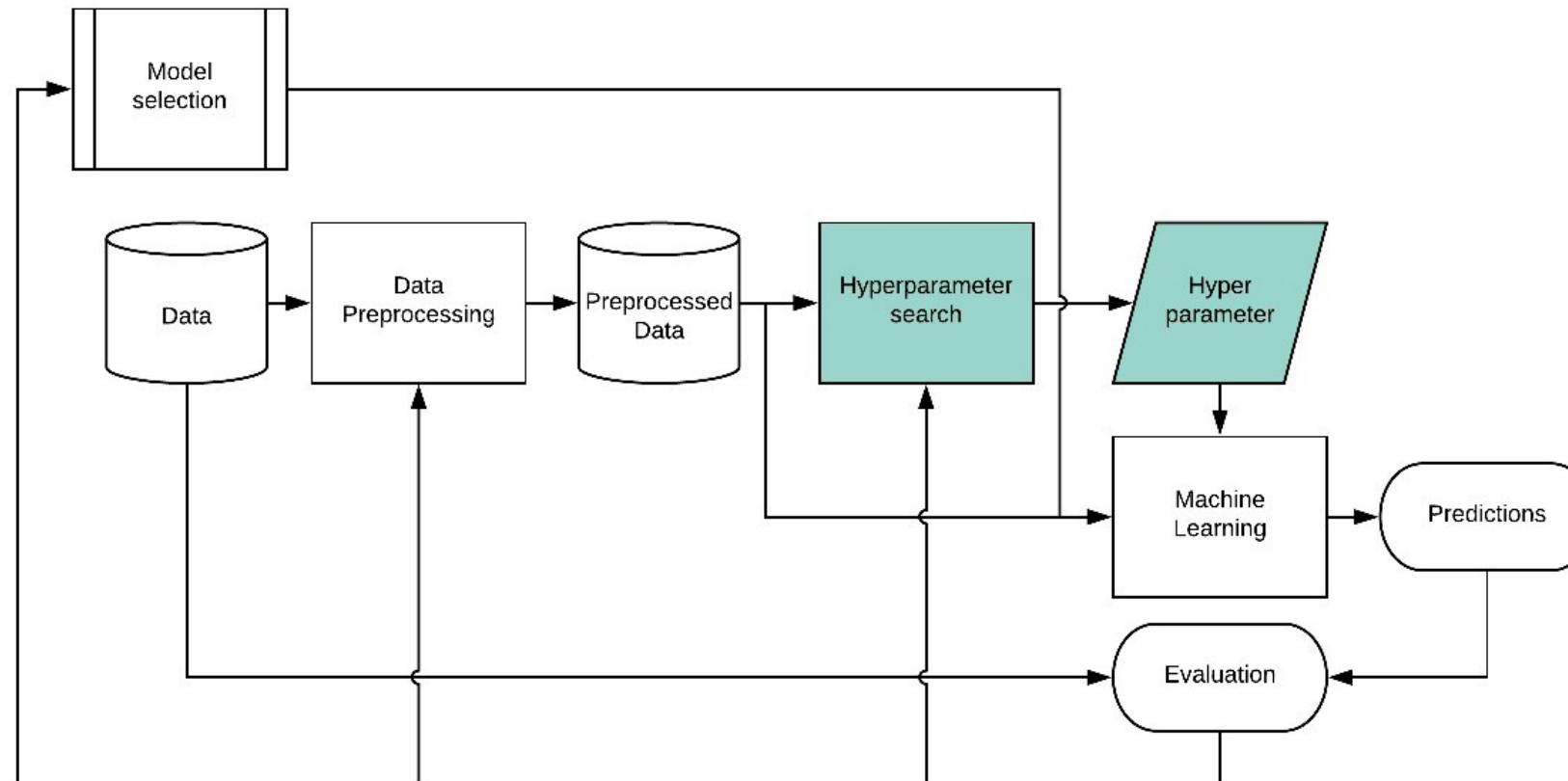
- Se analizan varios otros modelos para seleccionar el mejor
- El nivel de evaluación no es el más adecuado
- Todavía mucho por mejorar
- Nivel: intermedio bajo

# Machine Learning es mucho más que crear un modelo que funcione



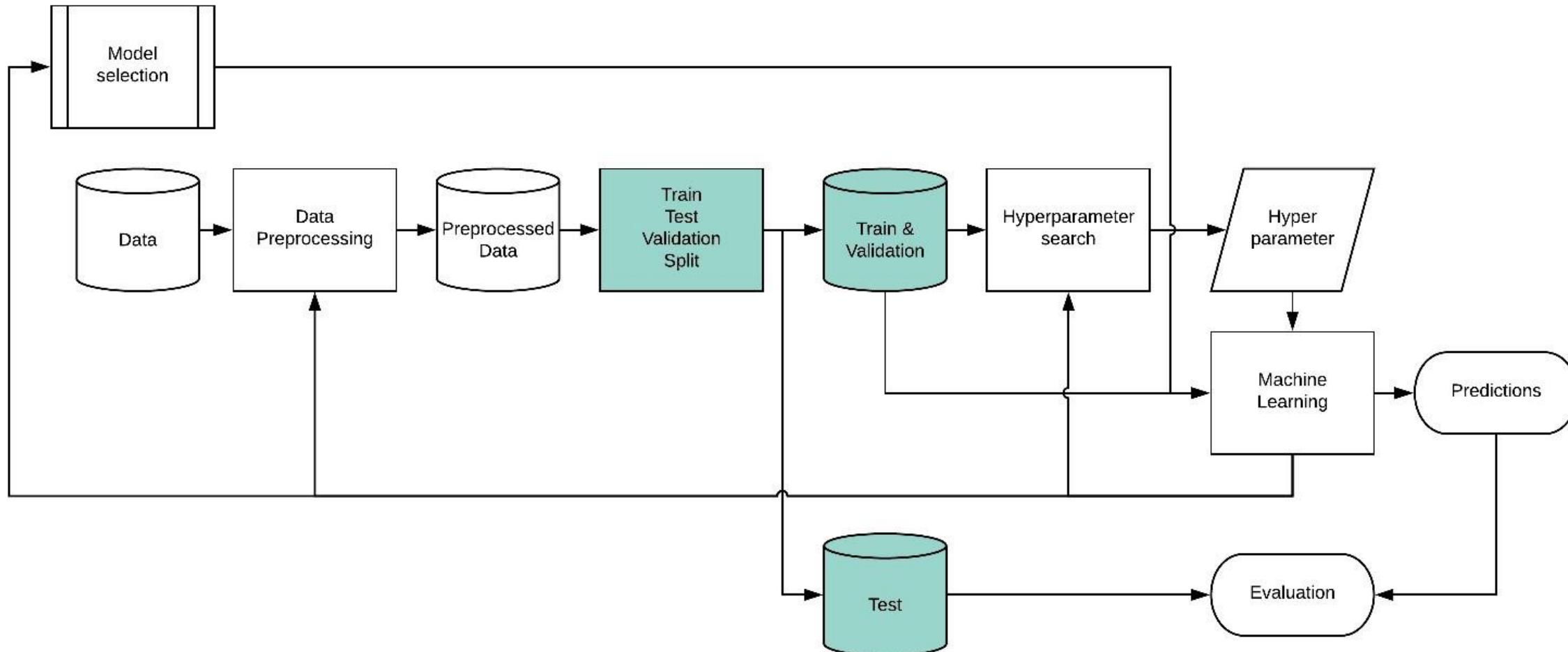
- La calidad de los datos es mejorada
- Los modelos mejoran su rendimiento
- Nivel: intermedio

# Machine Learning es mucho más que crear un modelo que funcione



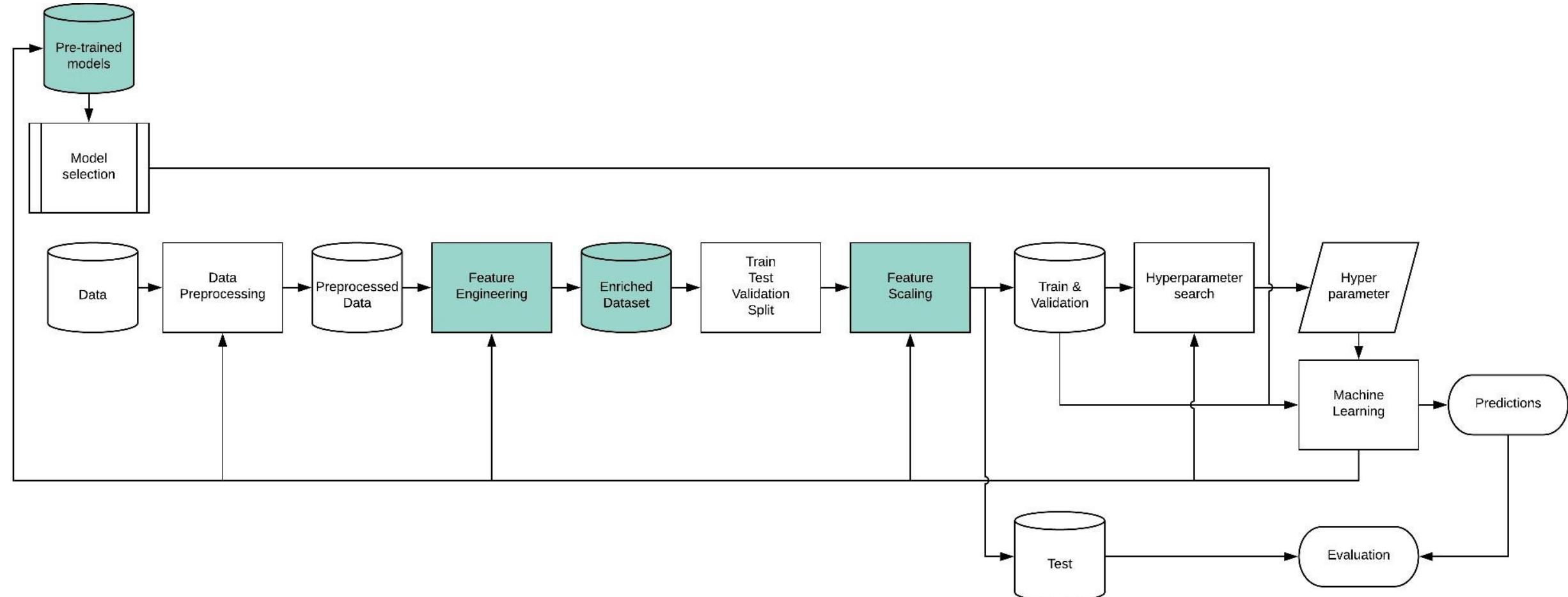
- Los modelos mejoran más con el tuning de parámetros
- Modelos robustos
- Nivel: intermedio/avanzado

# Machine Learning es mucho más que crear un modelo que funcione



- La generalización está garantizada
- Nivel: avanzado

# Machine Learning es mucho más que crear un modelo que funcione



- Nuevas variables creadas para mejorar el problema
- Si el problema ya fue solucionado, se hace uso de esto
- Nivel: Pro!

# Content

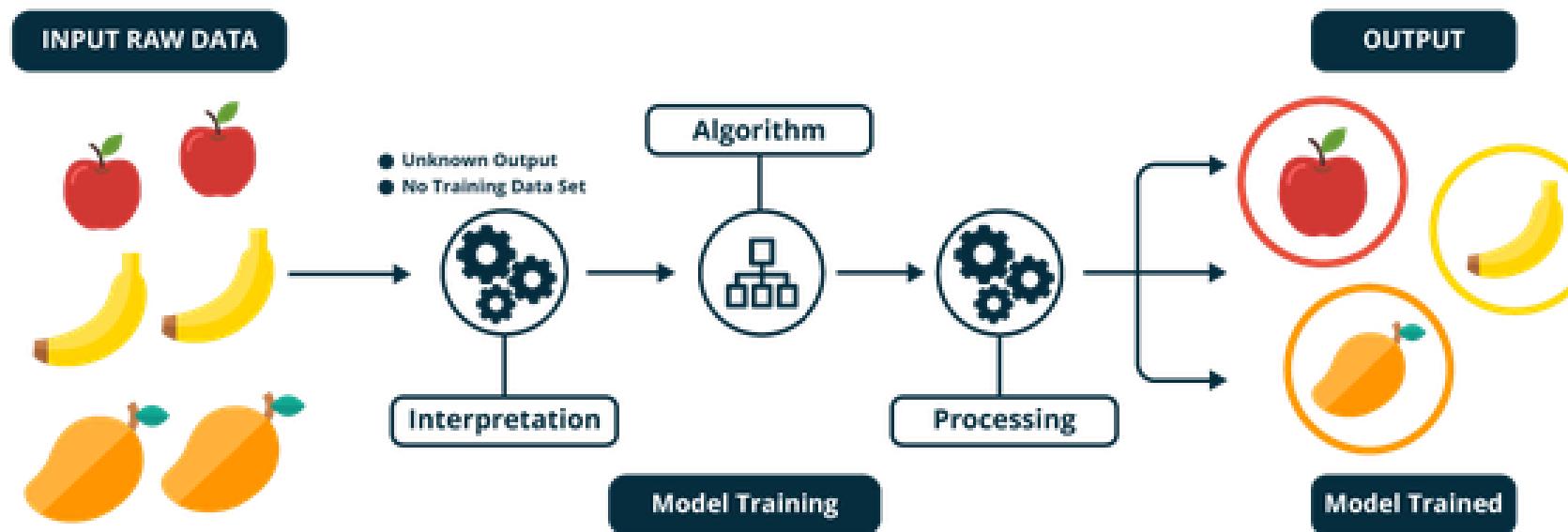
Logística del curso  
Sílabo, reglas, etc.

Concepto de Aprendizaje Automático  
Concepto, relaciones con otras áreas

Tipos de aprendizajes  
Aprendizajes en Machine Learning

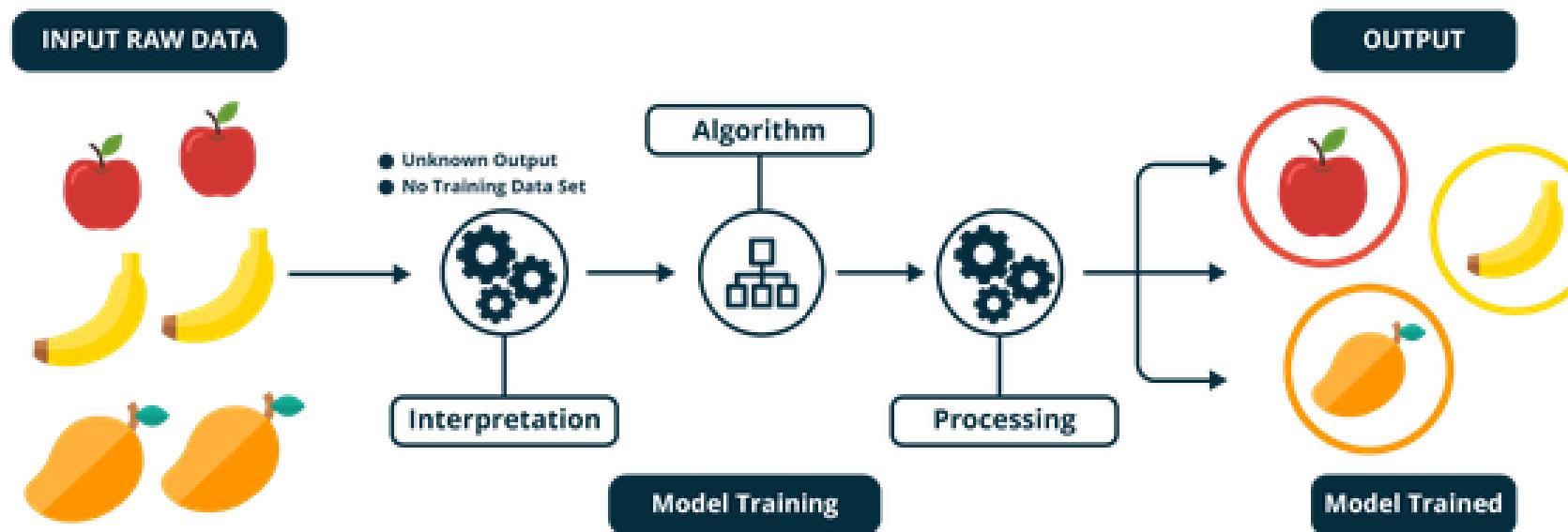
Aplicaciones  
Estado del arte

El **aprendizaje supervisado** se basa en entradas etiquetadas y un modelo que deduce una etiqueta para una nueva entrada



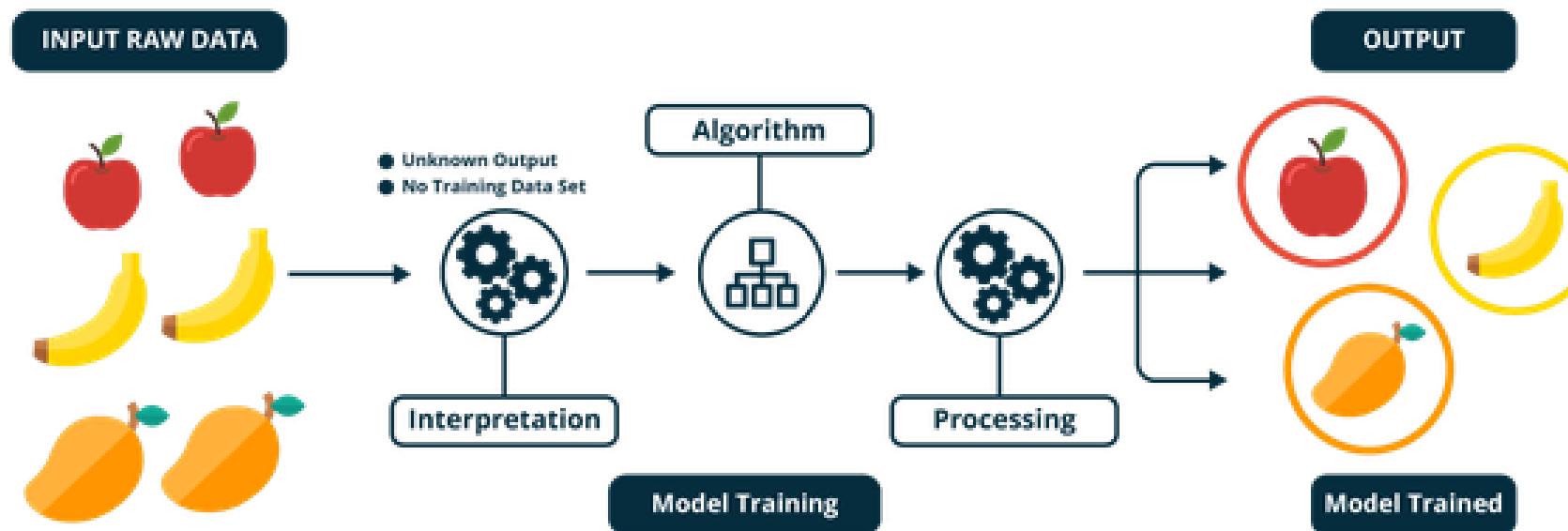
El modelo aprende a reconocer tres tipos de frutas en base a ejemplos

El **aprendizaje supervisado** se basa en entradas etiquetadas y un modelo que deduce una etiqueta para una nueva entrada



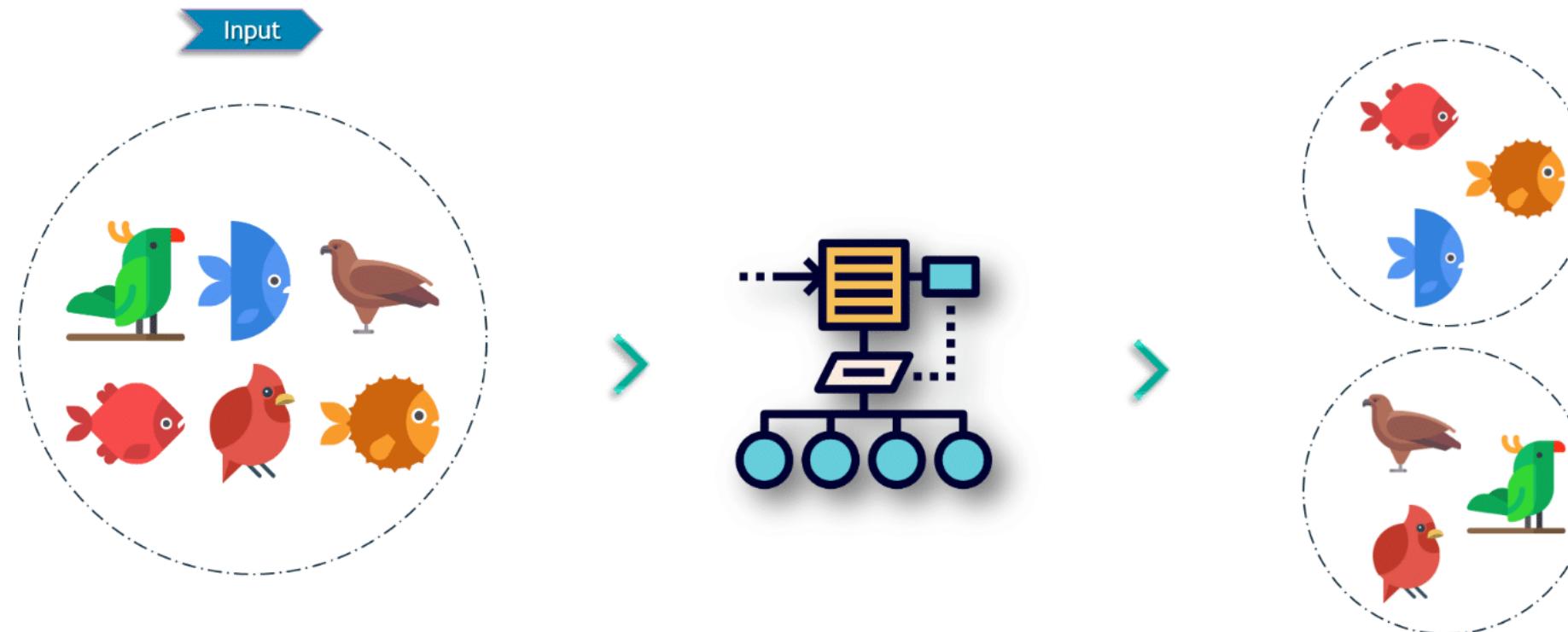
Cada ejemplo tiene asociada una etiqueta (tipo de fruta)

El **aprendizaje supervisado** se basa en entradas etiquetadas y un modelo que deduce una etiqueta para una nueva entrada



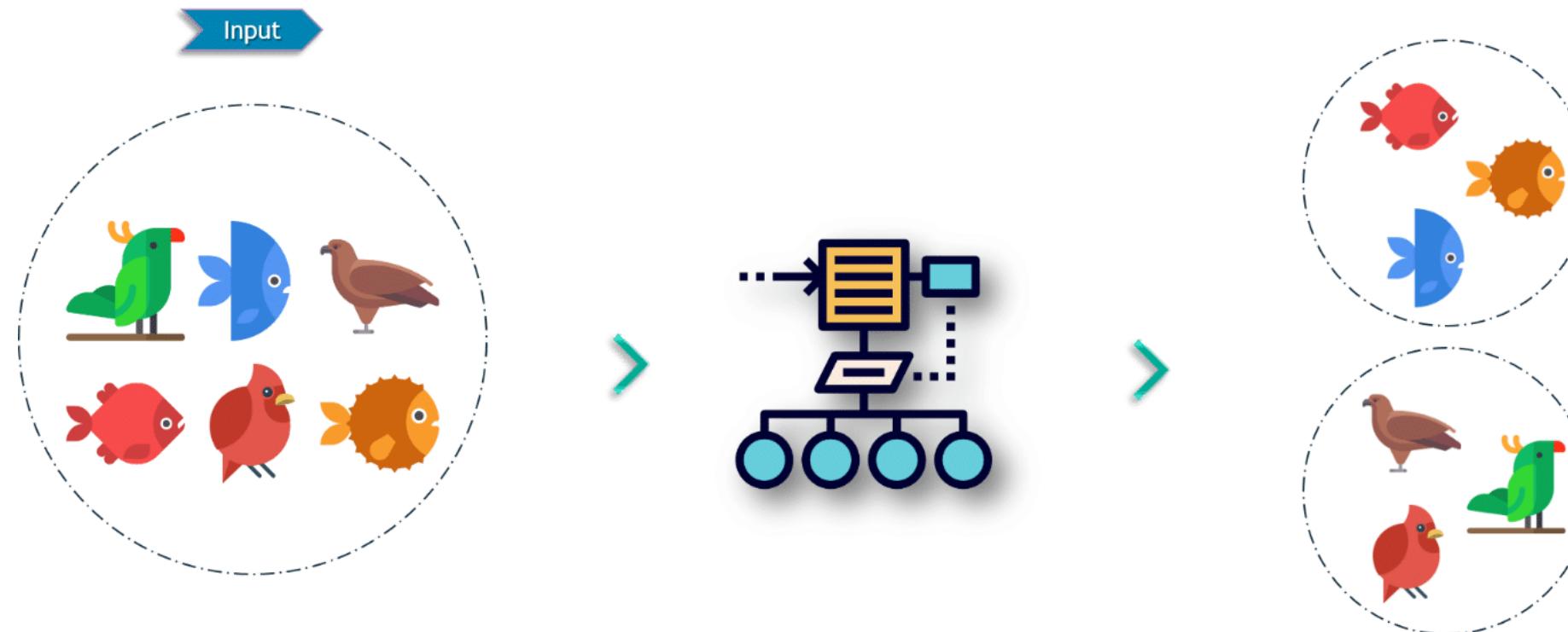
Cuando se le presenta una nueva fruta al modelo, este lo clasifica como uno de los tipos de fruta

El **aprendizaje no supervisado** se basa en entradas sin etiquetas, donde el modelo intenta crear categorías



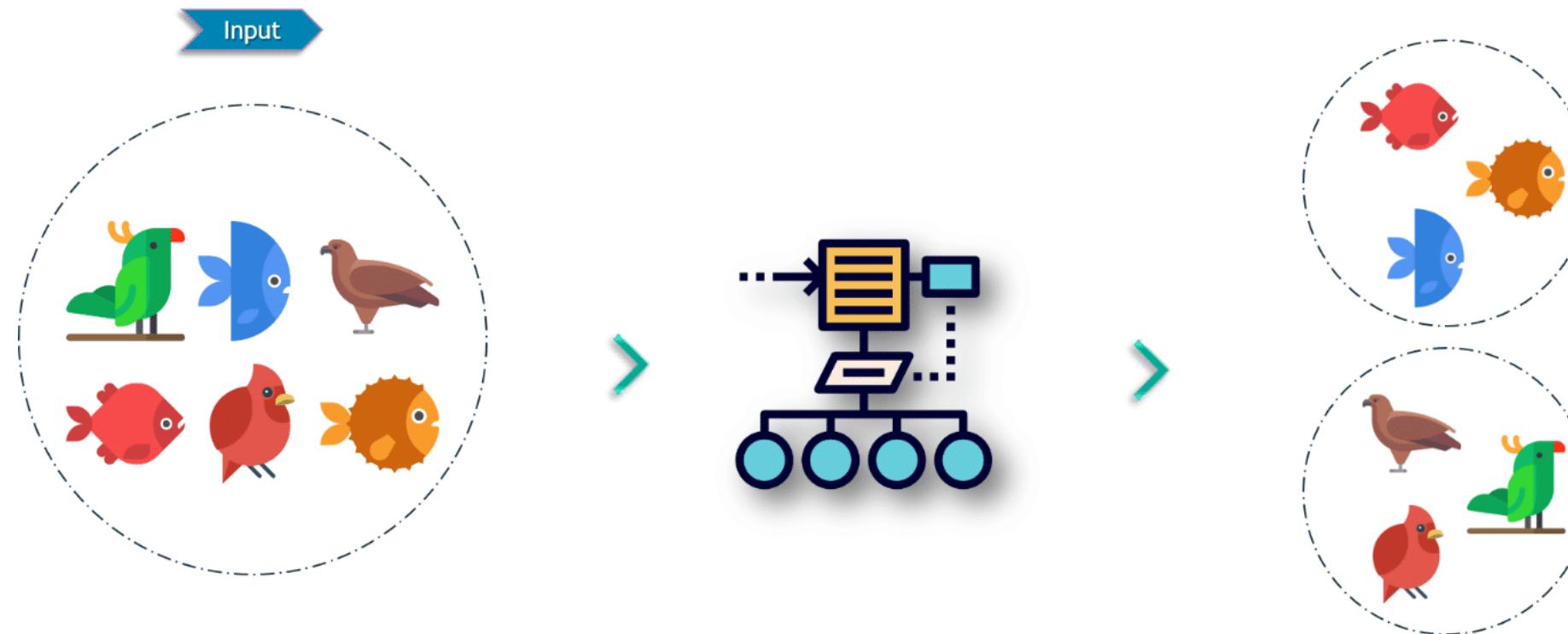
En este tipo de aprendizaje las entradas no están etiquetadas (situación muy común)

El **aprendizaje no supervisado** se basa en entradas sin etiquetas, donde el modelo intenta crear categorías



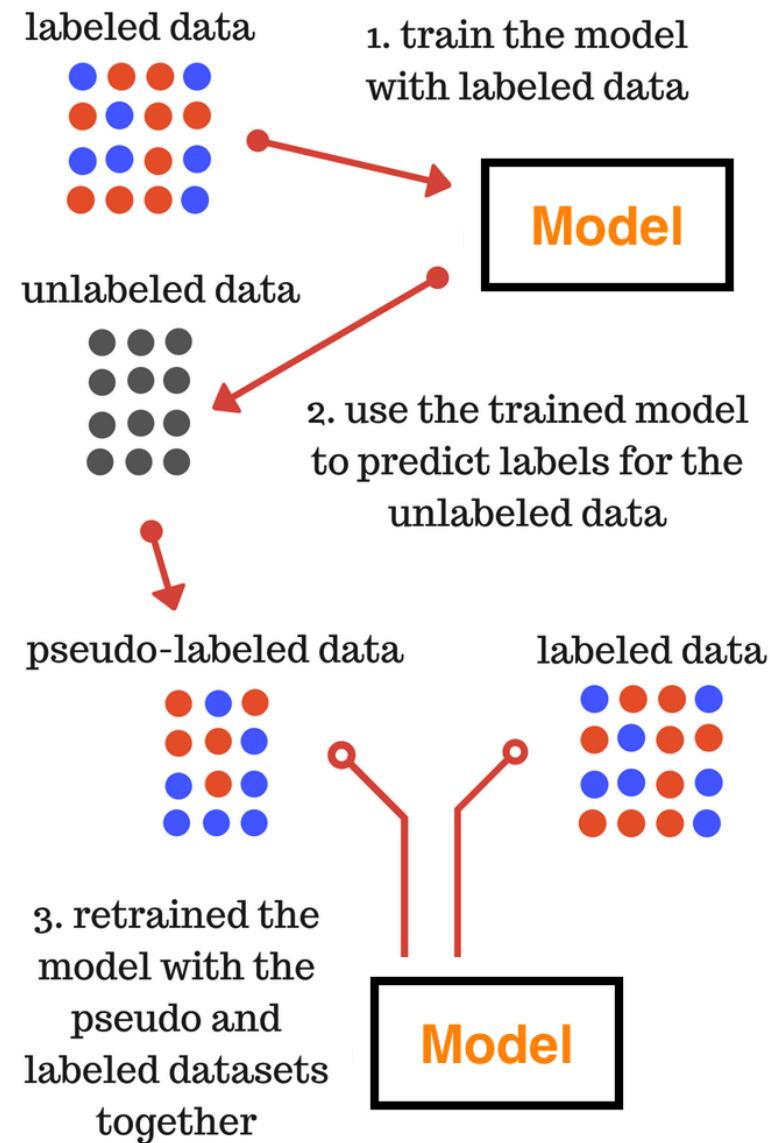
El modelo intenta agrupar las entradas en base a  
a alguna métrica de similaridad

El **aprendizaje no supervisado** se basa en entradas sin etiquetas, donde el modelo intenta crear categorías



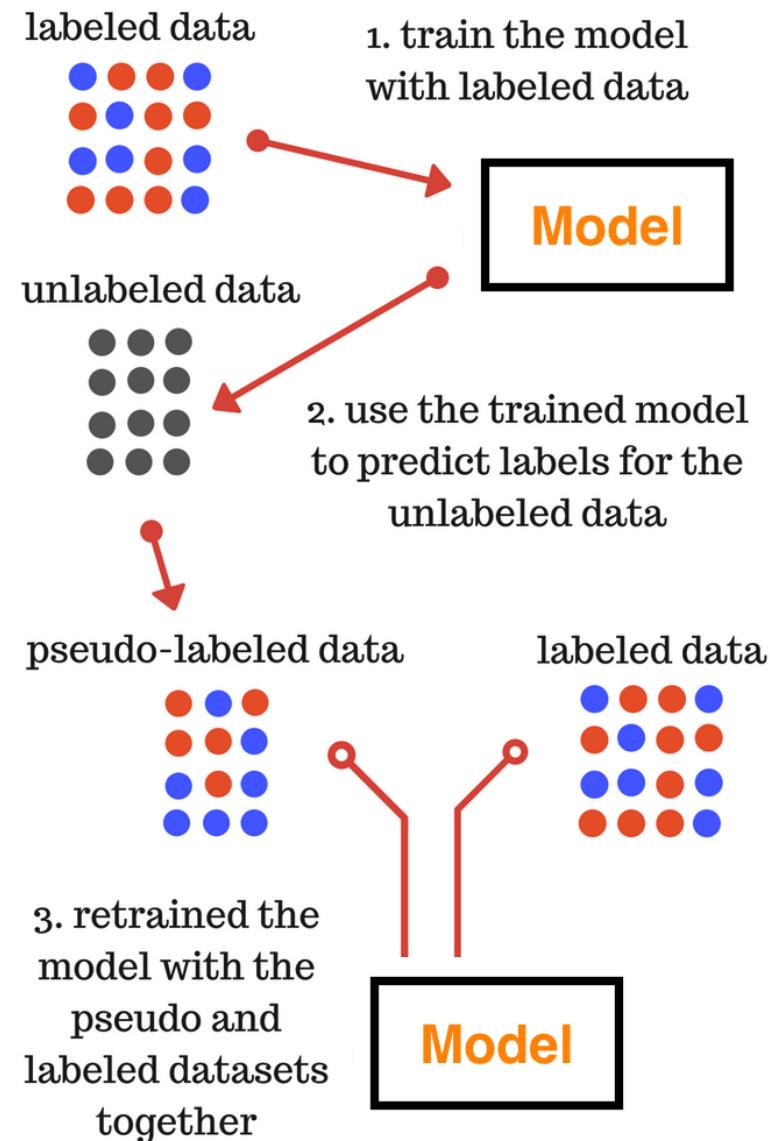
Finalmente el modelo crea grupos de elementos similares

# El aprendizaje semi supervisado se basa en usar las entradas etiquetadas para categorizar las que no tienen etiquetas



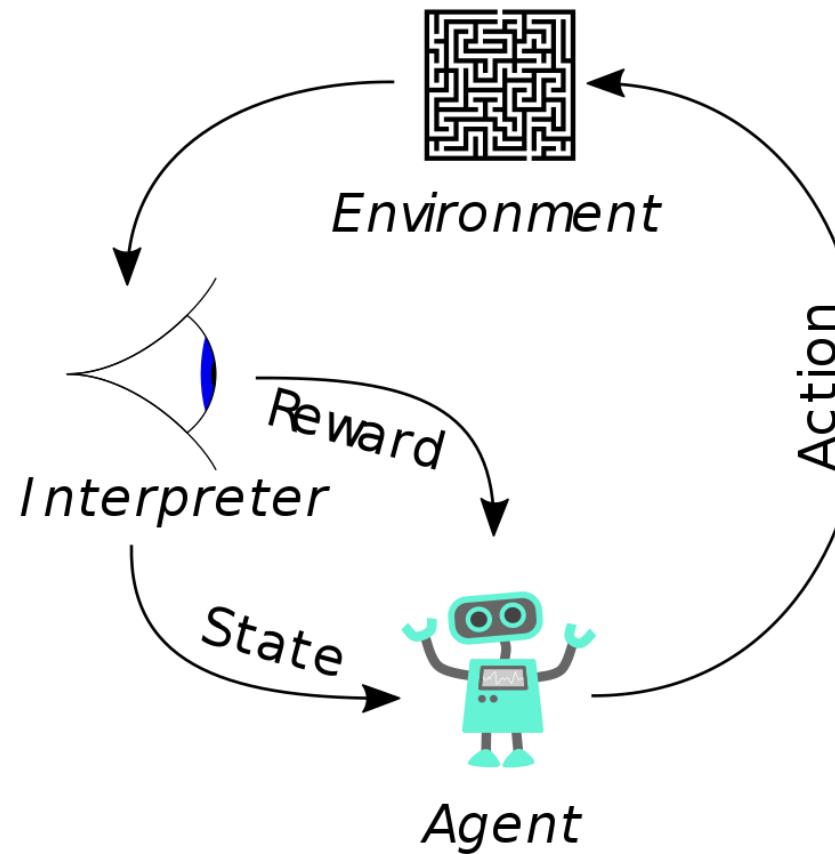
Existen entradas con y sin etiquetas  
(mayormente sin etiquetas)

# El aprendizaje semi supervisado se basa en usar las entradas etiquetadas para categorizar las que no tienen etiquetas



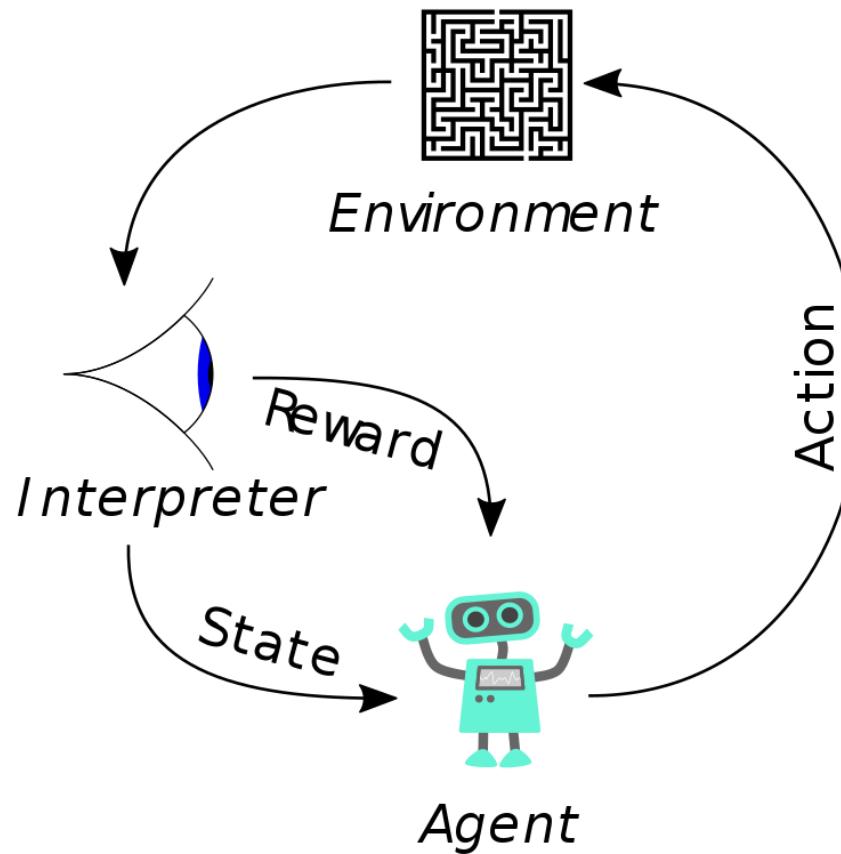
El objetivo es usar las entradas etiquetadas para categorizar las que no tienen

En **Reinforcement Learning**, el modelo percibe su ambiente y aprende en base a premios



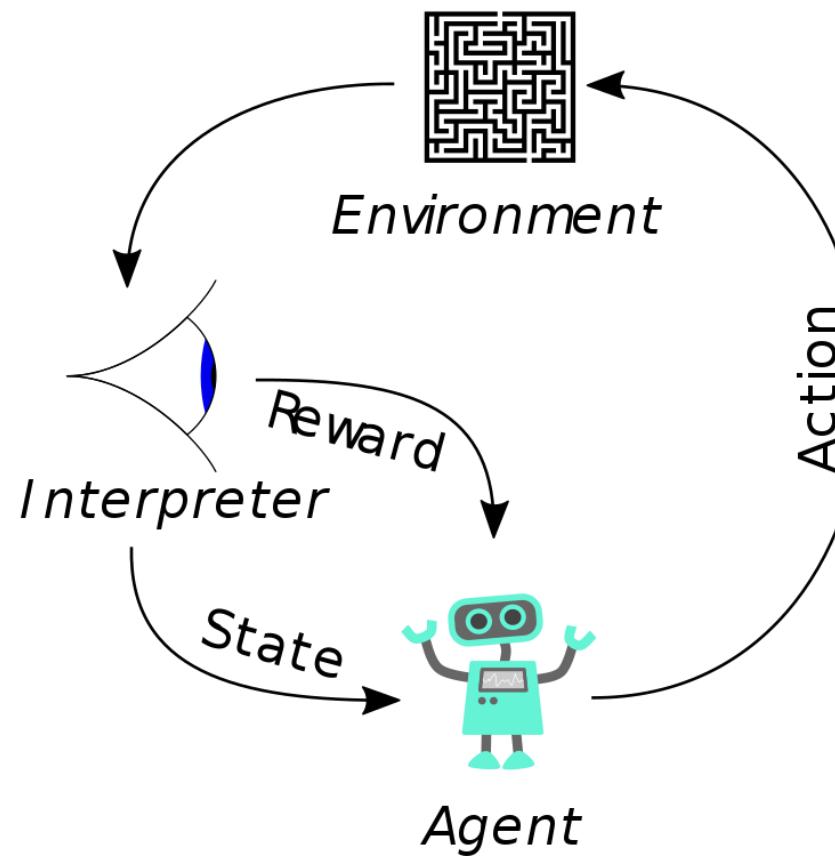
Un agente (e.j. robot) debe realizar una tarea  $T$   
(encontrar la salida de un laberinto)

En **Reinforcement Learning**, el modelo percibe su ambiente y aprende en base a premios



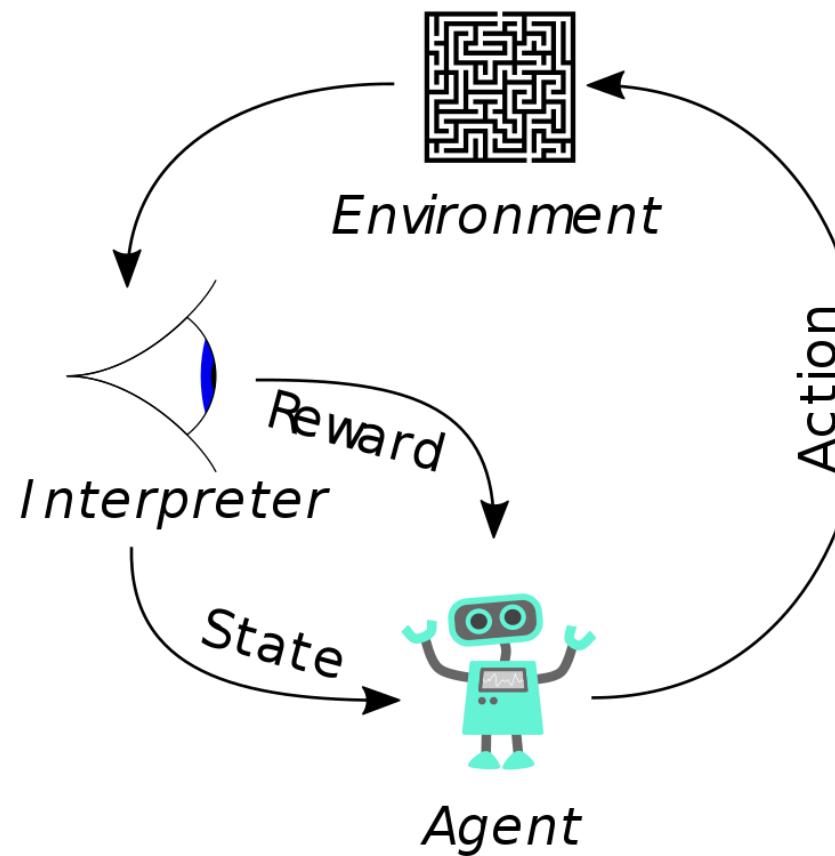
El agente realizar alguna acción  $A$  que es interpretada por un intérprete

En **Reinforcement Learning**, el modelo percibe su ambiente y aprende en base a premios



El intérprete decide si la acción  $A$  fue útil o no.  
Dependiendo de esto, devuelve un premio  $R$

En **Reinforcement Learning**, el modelo percibe su ambiente y aprende en base a premios



El agente actualiza su estado y genera una nueva acción  $A'$

# Content

Logística del curso  
Sílabo, reglas, etc.

Concepto de Aprendizaje Automático  
Concepto, relaciones con otras áreas

Tipos de aprendizajes  
Aprendizajes en Machine Learning

Aplicaciones  
Estado del arte

# Encontrando patrones en el genoma humano

Los genomas humanos (también otros) están codificados en el DNA

- Una larga secuencia de strings (nucleoides): (A, C, G, T)

Existen métodos para determinar esta cadena

Siguiente paso: entender el genoma

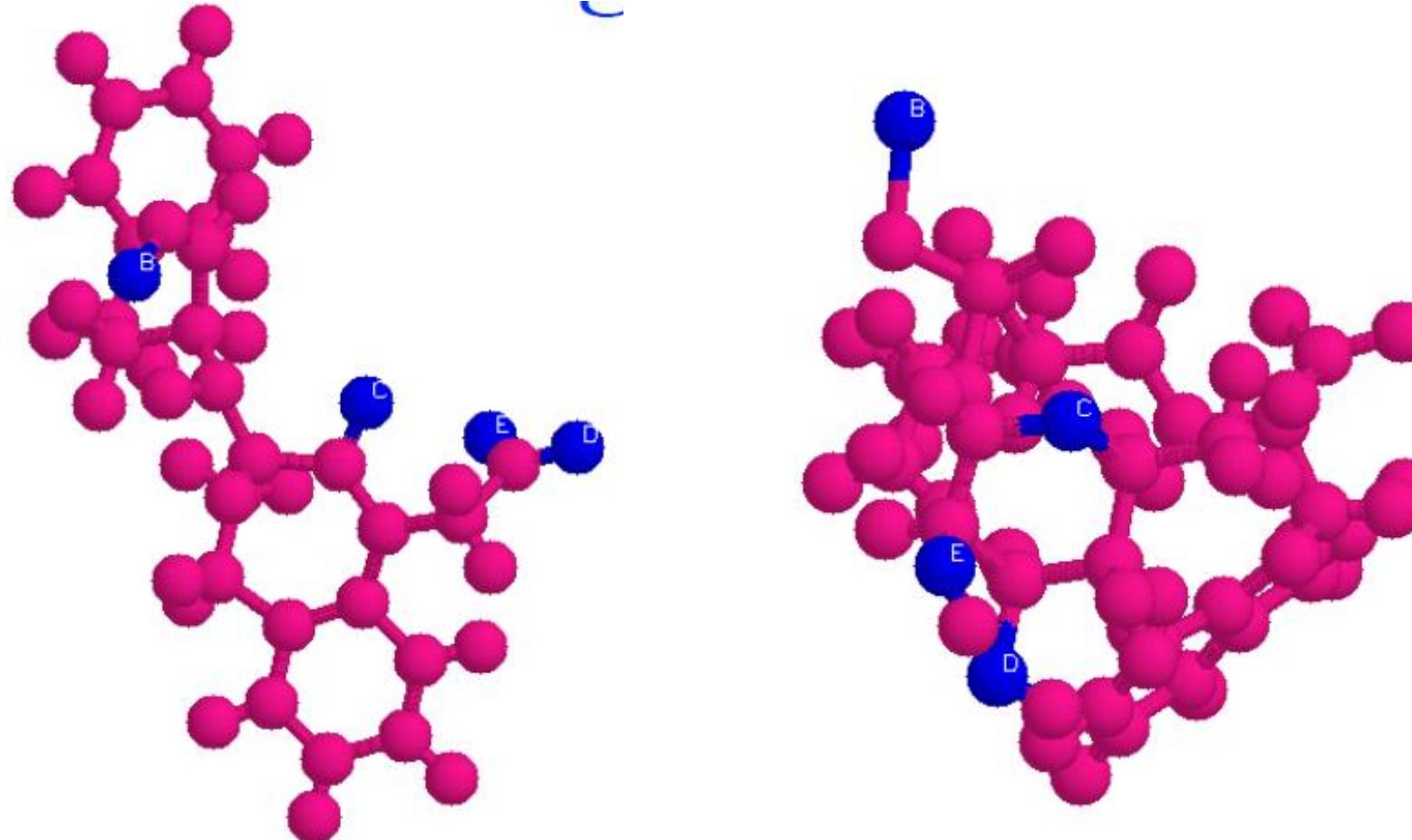
- Encontrar que partes realizan que función
- Técnicas de Machine Learning/Data Mining fueron usadas para esta tarea

... TGAAGAGACTTAACCTATTACCGGACATCGACT...



Gene encoding for ...

# Descubriendo patrones en moléculas para crear nuevos tratamientos



Analizar moléculas y como actúan contra alguna enfermedad para encontrar que tienen en común; esta podría ser la razón para su actividad

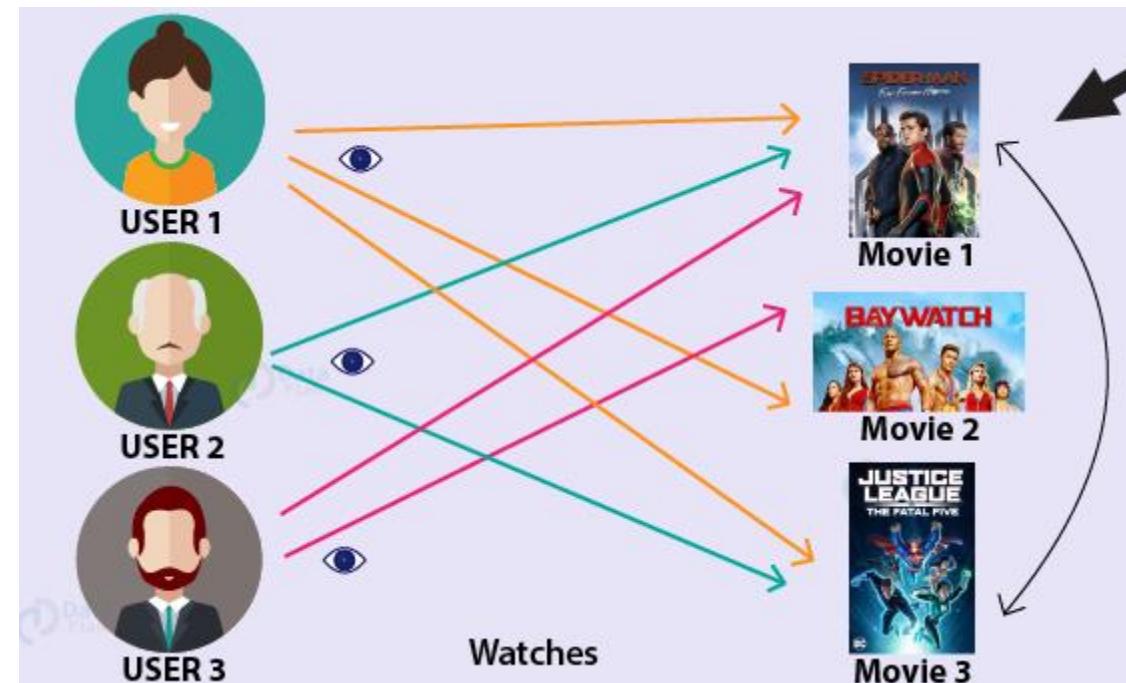
# EVE el robot científico, King (2015)



**La investigación científica es iterativa**  
Generar hipótesis, probar, analizar

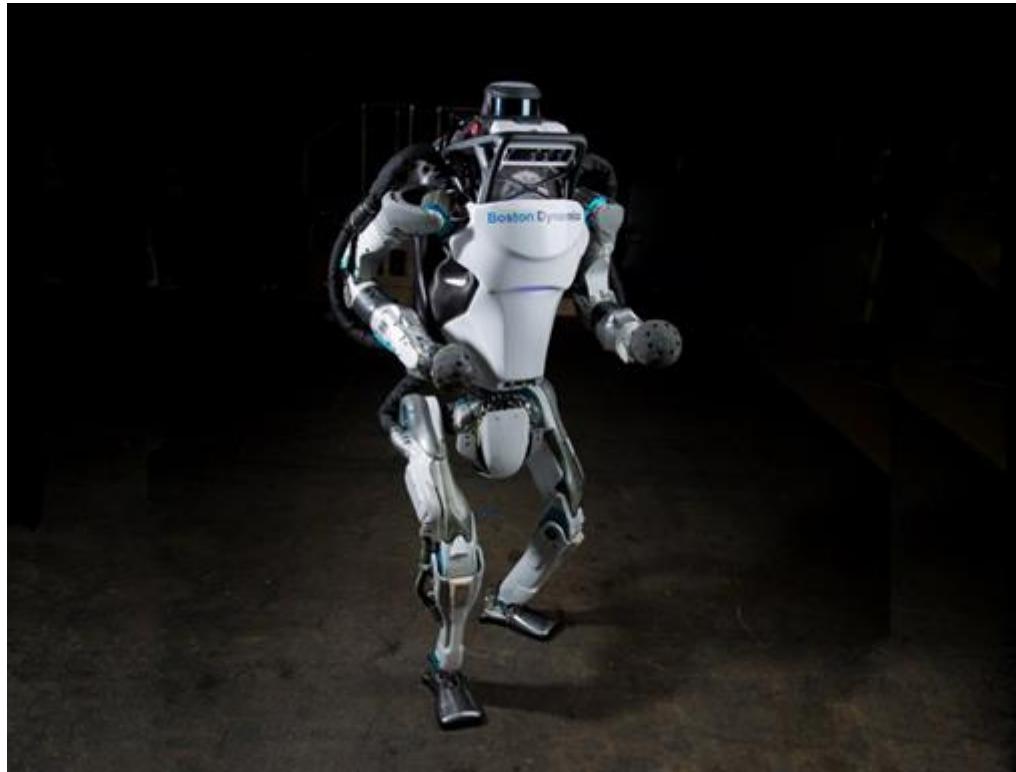
**EVE automatiza todo el proceso**  
Genera todo el proceso sin necesidad de humanos

# Sistemas de recomendación basados en perfiles de usuario



El sistema recomienda elementos en base a la similitud del perfil de la persona con otros en el sistema

# Robots que aprenden de la experiencia



El robot aprende en base a estímulos y recompensas. Cuando una acción genera un resultado correcto, aprende de la experiencia

# Termostatos inteligentes que aprenden de los patrones de las personas

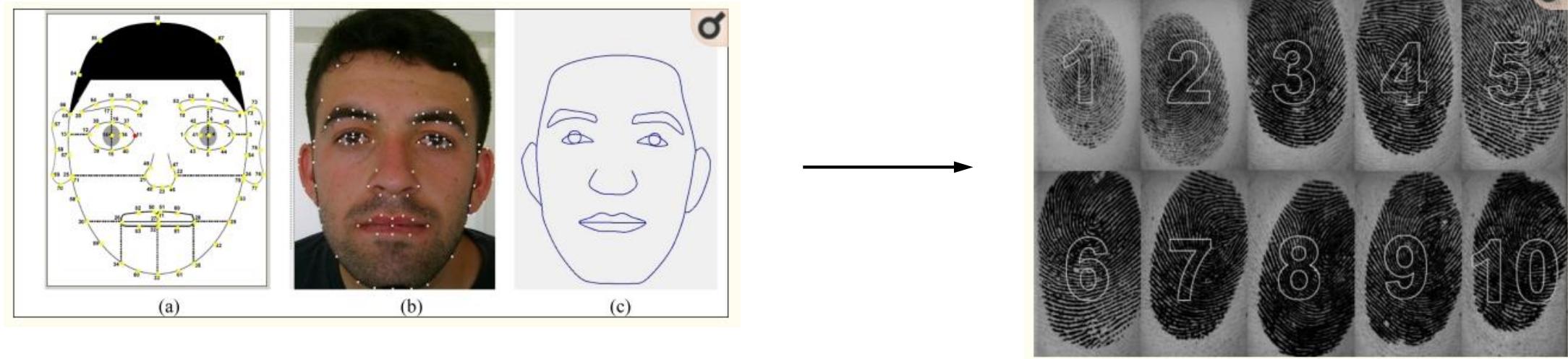


**El termostato aprende de patrones**  
Las personas tienen patrones de ocupación

**El sistema provee comfort**  
El sistema se activa si se espera presencia

**El sistema genera ahorro**  
El sistema se desactiva si no habrá presencia

# Generar rostros en base de huellas: ¿Cómo lucía el ladrón?



Se extraen características de los rostros y huellas para correlacionarlos y crear un modelo que pueda predecir como luce una persona en base a su huella



UNIVERSIDAD DE CUENCA  
*desde 1867*

# Introducción: ¿Qué es el aprendizaje automático?

Andres Auquilla  
2020

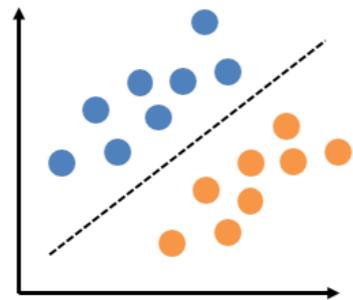


UNIVERSIDAD DE CUENCA  
*desde 1867*

# Conceptos generales del aprendizaje

Andres Auquilla  
2020

# Content



**Notación y definiciones**

Sets, listas, matrices, etc.

**Aprendizaje basado en instancias**

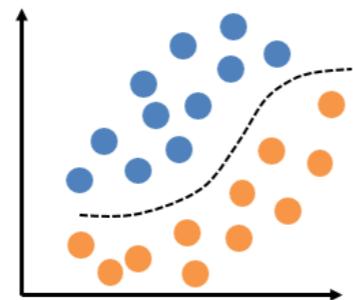
Instancias y mapeos

**Version spaces**

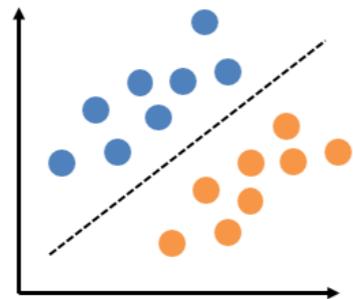
Hipótesis, conceptos, aprendizaje

**Bias**

Suposiciones que hacen los modelos

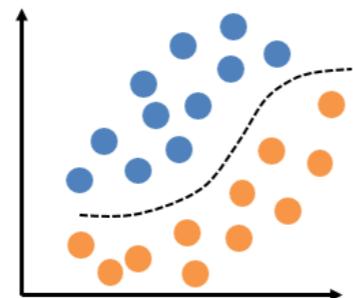


# Content



Notación y definiciones  
Sets, listas, matrices, etc.

Aprendizaje basado en instancias  
Instancias y mapeos



Version spaces  
Hipótesis, conceptos, aprendizaje

Bias  
Suposiciones que hacen los modelos

## Estructuras de datos:

- Escalares: 15, -3.8
- Vectores:  $a=[12, 45, 67]$ ,  $b=[1, 0]$
- $a^j$  donde  $j$  denota una dimensión específica del vector  $a$
- $a^{(2)} = 45$
- Se pueden tener más índices:  $x_i^{(j)}$  denota la dimensión  $j$  del elemento  $i$
- Un conjunto es una colección de datos no ordenados, e.j.  $S = \{1, 3, 18\}$ ,  $S = \{x_1, x_2, \dots, x_n\}$
- Un conjunto puede expresarse como un intervalo entre  $a$  y  $b$ , e.j.  $[a, b]$ . En este caso,  $a$  y  $b$  están incluidos en el conjunto
- Cuando  $a$  y  $b$  no están incluidos, se lo expresa como  $(a, b)$
- $S_3 \leftarrow S_1 \cap S_2$ ,  $S_1 = \{1, 3, 5, 8\}$ ,  $S_2 = \{1, 8, 4\}$ ,  $S_3 = \{1, 8\}$

## Sumatorias:

- Dada una colección  $X = \{x_1, \dots, x_n\}$ , la sumatoria de elementos se define como  $\sum_{i=1}^n x_i = x_1 + x_2 + \dots + x_n$
- Dado un vector  $x = [x^{(1)}, x^{(2)}, \dots, x^{(n)}]$ , la sumatoria de elementos se define como  $\sum_{i=1}^n x^{(i)} = x^{(1)} + \dots + x^{(n)}$

## Multiplicaciones:

- Analogamente, la multiplicación de elementos de un conjunto  $X$  se define como  $\prod_{i=1}^n x_i = x_1 \times x_2 \times \dots \times x_n$

## Operaciones con conjuntos:

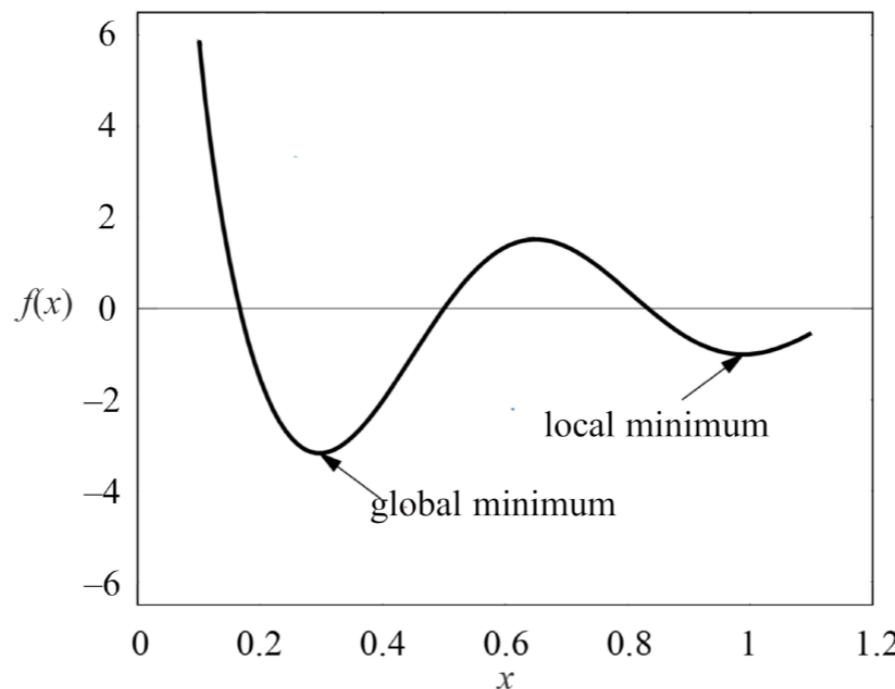
- $S' \leftarrow \{x^2 | x \in S, x > 3\}$  significa que se crea un nuevo conjunto  $S'$  con los valores al cuadrado del conjunto  $S$  siempre y cuando sus valores sean mayores a 3

## Funciones:

- Es una relación que asocia un elemento  $x \in X$  (dominio de la función) a un elemento  $y \in Y$  (codominio de la función)
- Si una función tiene un nombre  $f$ , esta relación se denota como  $y = f(x)$

## Mínimo local y global:

- $c$  es un mínimo local en un intervalo  $[a, b]$  de  $f$  si  $f(x) \geq f(c), c \neq x, x \in [a, b]$
- $c$  es un mínimo global de  $f$  si  $f(x) \geq f(c), c \neq x$



## Max y Argmax:

- Dado  $A = \{a_1, a_2, \dots, a_n\}$ , el operador  $\max_{a \in A} f(a)$  devuelve el valor más alto de  $f(a), \forall a \in A$
- El operador  $\arg\max_{a \in A} f(a)$  retorna el valor de  $a$  que hace que la función  $f$  sea la más alta
- Se aplica la misma analogía para  $\min$  y  $\arg\min$

## Derivadas:

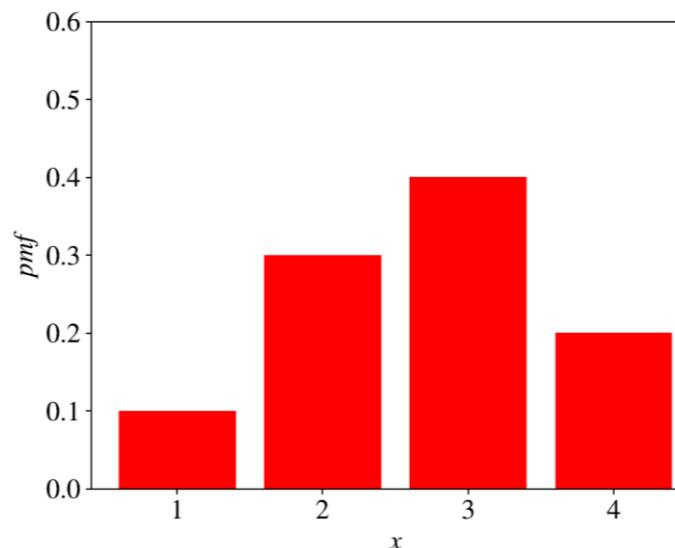
- La derivada describe cuan rápido una función  $f$  crece (o decrece)
- Regla de la cadena:  $F(x) = f(g(x)), F'(x) = f'(g(x))g'(x)$
- Ejemplo:  $F(x) = (5x + 1)^2$ , donde  $g(x) = 5x + 1$  y  $f(g(x)) = (g(x))^2$ . Aplicando la regla de la cadena,  $F'(x) = 2(5x + 1)g'(x) = 2(5x + 1)5 = 50x + 10$

## Gradiente:

- La gradiente de una función  $f(x, y)$  ( $\nabla f$ ) está dada por el vector  $\left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$
- Revisar: <https://www.youtube.com/watch?v=sDv4f4s2SB8>

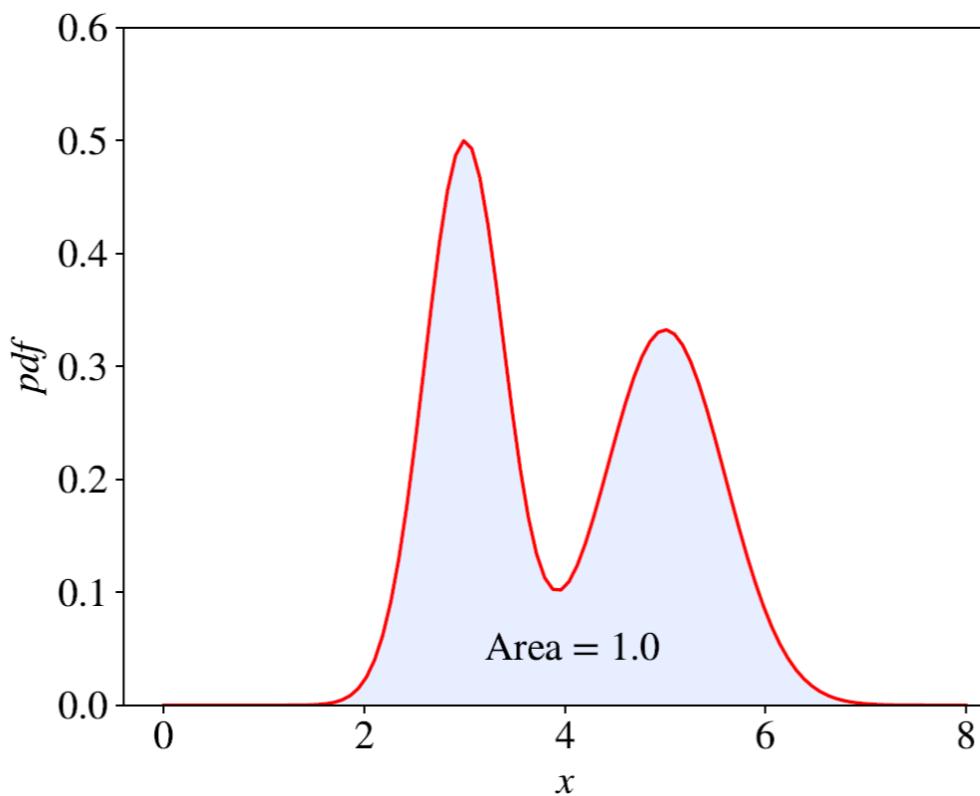
## Variables aleatorias:

- Es una variable cuyos posibles valores son derivados de un fenómeno aleatorio
- Ejemplos: lanzamiento de una moneda  $X = \{1, 0\}$ , donde 1 corresponde a cara y 0 corresponde a sello
- Estas variables pueden ser discretas o continuas
- La distribución de probabilidad de una variable aleatoria discreta se describe como una lista de probabilidades (probability mass función - pmf). Ejemplo:  $\Pr(X = 1) = 0.25$ ,  $\Pr(X = 2) = 0.25$ ,  $\Pr(X = 3) = 0.5$
- $\sum \Pr(X = x) = 1, x \in X$



## Variables aleatorias continuas:

- Continuous random variable (CRV) puede tomar un infinito número de valores en un intervalo. Ejemplos: peso, altura, tiempo, etc.
- Sigue una función de densidad (probability density function - pdf)
- El área bajo la curva de una pdf es siempre 1



## Valor esperado de $X$

- Dada una variable  $X$  con  $k$  posibles valores  $\{x_i\}_{i=1}^k$ , la expectation de  $X$  ( $E[X]$ ) se define como  $E[X] \stackrel{\text{def}}{=} \sum_{i=1}^k [x_i \times \Pr(X = x_i)]$
- También se la denomina media, promedio o valor esperado
- La varianza se define como  $\sigma^2 = E[(X - \mu)^2]$  donde  $\mu = E(X)$
- La expectation de una variable aleatoria continua  $X$  se define como

$$E[X] \stackrel{\text{def}}{=} \int_R x f_X(x) dx$$

- Donde  $f_X$  es la pdf de la variable  $X$  y  $\int_R$  es la integral de la función  $xf_x$
- Normalmente  $f_X$  no es conocido, pero se puede observar (mediante algunos valores de  $X$ )

## Estimadores sin sesgo

- Ya que  $f_X$  es usualmente desconocida, pero se tiene una muestra  $S_X = \{x_i\}_{i=1}^N$ , se pueden calcular valores estadísticos en base a estimadores sin sesgo
- $\hat{\theta}(S_X)$  es un estimador sin sesgo de algún estadístico  $\theta$  calculado usando una muestra  $S_X$  si  $\hat{\theta}(S_X)$  tiene la siguiente propiedad  $E[\hat{\theta}(S_X)] = \theta$
- La media de la muestra se calcula de la siguiente manera  $\frac{1}{N} \sum_{i=1}^N x_i$

# Parámetros vs hyper-parámetros

Parámetros son variables que definen un modelo de ML. Estos son modificados directamente por el algoritmo en función de los datos de entrenamiento

Los hyper-parámetros son propiedades del algoritmo, normalmente tiene valores numéricos. Estos valores no son aprendidos por el algoritmo, sino deben ser configurados por el analista

# Clasificación vs Regresión



## Regression

What is the temperature going to be tomorrow?

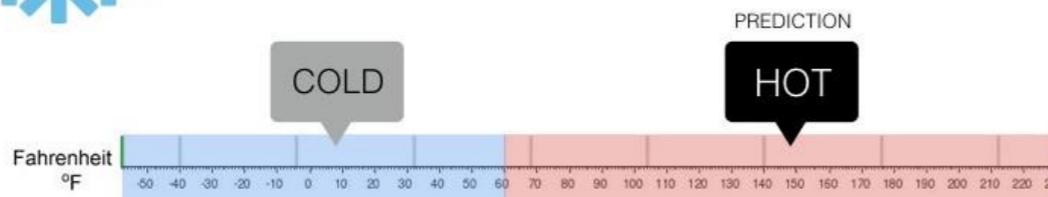


El valor de predicción es continuo



## Classification

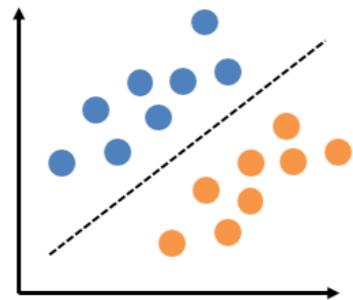
Will it be Cold or Hot tomorrow?



El valor de predicción es discreto. El valor representa a alguna categoría  $k \in K$ . En el ejemplo:  $K = \{cold, hot\}$

En ambos casos, el modelo se generó con datos etiquetados

# Content

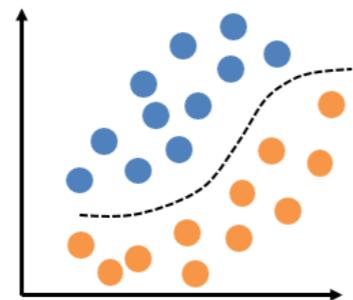


**Notación y definiciones**

Sets, listas, matrices, etc.

**Aprendizaje basado en instancias**

**Instancias y mapeos**



**Version spaces**

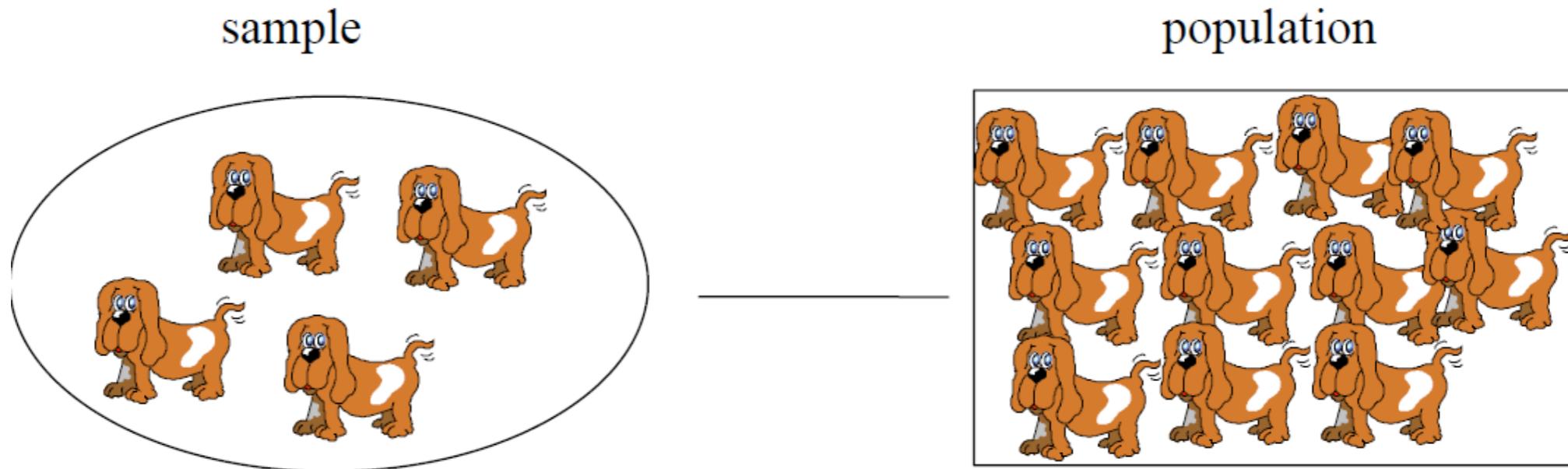
Hipótesis, conceptos, aprendizaje

**Bias**

Suposiciones que hacen los modelos

# La **inferencia inductiva** es razonar de lo específico a lo general

En la estadística, de una muestra inferir propiedades de una población



Observación: “estos perros son de color café”

Hipótesis: “todos los perros son de color café”

# La inferencia inductiva es más general que las estadísticas

Las estadísticas mayormente consisten en métodos numéricos para inferencia

- Inferir: media, distribución de probabilidad, varianza, etc.

## Otras técnicas

- Encontrar la definición simbólica de un concepto (concept learning)
- Encontrar las leyes complejas que gobiernan los datos
- Inducción desde un punto lógico, filosófico, etc.

# Construyendo un sistema para jugar Damas



Aprendizaje = mejorando en la tarea  $T$ , con respecto a la medida de desempeño  $P$ , basado en la experiencia  $E$

En este ejemplo

- $T$  : jugar damas
- $P$  : porcentaje de partidos ganados en un torneo
- $E$  : juegos jugados contra otros jugadores
- ¿Es la experiencia lo suficientemente representativa?

Se debe especificar precisamente los datos de entrada y lo que se aprende

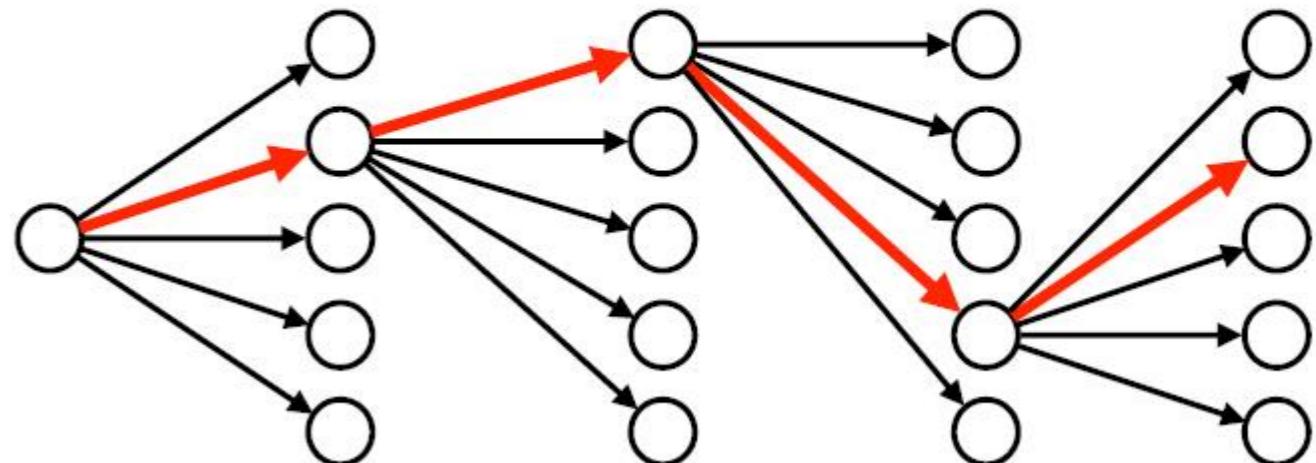
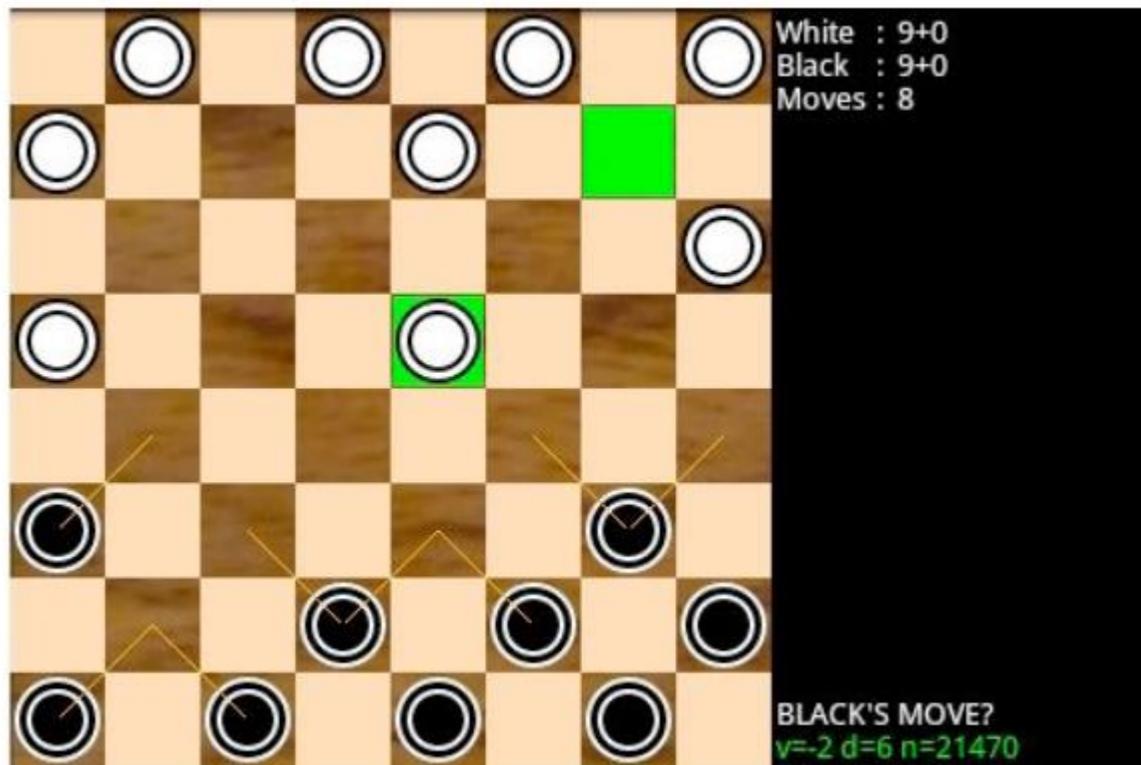
También la representación de datos y el algoritmo a usar

Lo que se provee de entrada

- ¿Evidencia directa o indirecta?
- Directa: e.j. que movimientos fueron positivos, cuales malos
- Indirecta: e.j. jugar todo el juego y ver el resultado

Para este caso, asumir evidencia indirecta

Cada movimiento genera una posible nueva cadena de movimientos que pueden ser explorados



Aprendizaje: dado el estado del juego,  
decidir que movimiento realizar

Realizado de forma directa:

EscogerMovimiento: Tabla    —————>    Movimiento

# Aprendizaje: dado el estado del juego, decidir que movimiento realizar

Realizado de forma indirecta:

$V$ : Tabla  $\longrightarrow R$

- Cuando se juega, considerar todos los posibles movimientos. Escoger el movimiento que lleva al mejor estado
- $V(\text{gana}) = 100; V(\text{ pierde}) = -100; V(\text{empata}) = 0$
- Usualmente no se pueden probar todos los estados (se debe aproximar  $V$ )
- Algoritmo minmax para lógica de juegos

Aprendizaje: dado el estado del juego,  
decidir que movimiento realizar

Como representar V: reglas if/then, red neuronal, función no lineal, etc.

$V$ : Tabla  $\longrightarrow R$

- E.j.  $V = w_1 bp + w_2 rp + w_3 bk + w_4 rk + w_5 bt + w_6 rt$
- $bp, rp$ : número de piezas negras y rojas
- $bk, rk$ : número de reyes negros y rojos
- $bt, rt$ : número de piezas negras y rojas en peligro
- $w_i$ : constantes que serán aprendidas por experiencia (generalización)

Aprendizaje: dado el estado del juego,  
decidir que movimiento realizar

¿Como obtener ejemplos de entrenamiento?.

$V$ : Tabla  $\longrightarrow R$

- Ejemplos de la forma  $\{bp, rp, bk, rk, bt, rt, V\}$
- Determinar  $V$  no es trivial, ¿usar información de partidos previos?
- Las constantes  $w_i$  pueden ser obtenidas por una regresión least-squares

# Averiguar

- Averiguar como funciona el algoritmo minimax
- ¿Que algoritmo fue implementado por la computadora DeepBlue cuando ganó sus partidos contra Kasparov?

Existen varios aspectos que influencian la selección de algoritmos/funciones para crear un modelo

Que algoritmo utilizar

Que funciones son útiles con el algoritmo seleccionado

Factores que influencian el aprendizaje

# de ejemplos, complejidad de la función, ruido, etc.

¿Se puede incluir conocimiento previo?

Este podría contribuir a un mejor rendimiento del modelo

¿Un sistema podría alterar su propio estado?

El sistema está aprendiendo con los ejemplos

# Recordando las tareas de aprendizaje

## Aprendizaje de un concepto

- Se aprende una definición de un concepto
- Aprendizaje supervisado vs no supervisado

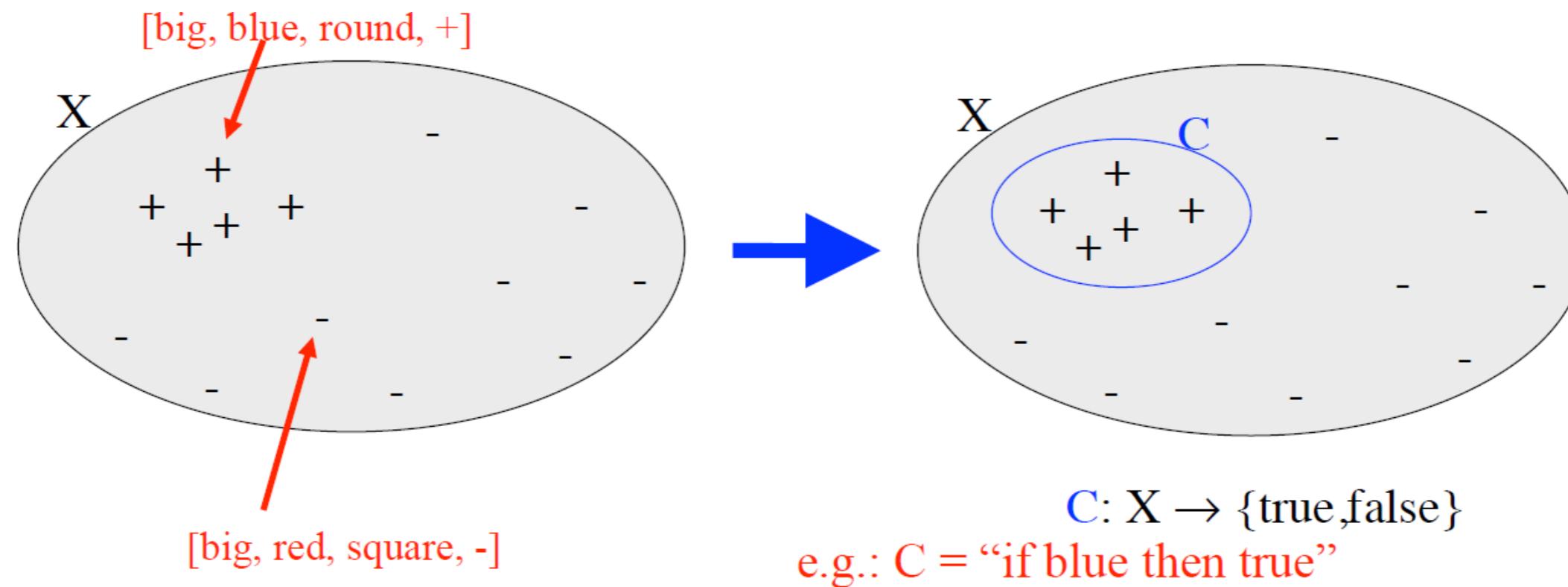
## Aprendizaje de funciones

- Discretas: clasificación
- Contínuas: regresión
- Concepto:  $f(x) \in \{1, 0\}$  función booleana

## Clustering

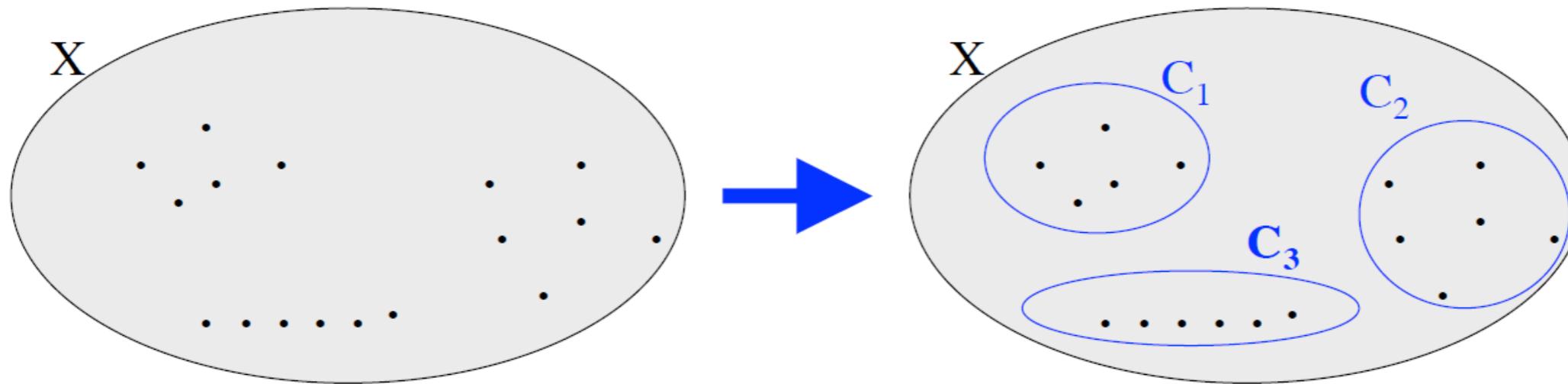
## Encontrar patrones descriptivos

En un **aprendizaje de conceptos supervisado** se infieren propiedades de las instancias etiquetadas



Dados ejemplos positivos y negativos, inferir propiedades que causen que las instancias sean positivas o negativas

En un **aprendizaje de conceptos no supervisado** se determinan conceptos razonables (clustering)



Dados ejemplos no etiquetados, encontrar las definiciones para estos conceptos.  
E.j. taxonomía de animales, tipos de clientes, anomalías, etc.

## Una definición de concepto puede ser **discriminante**

- Definición simple que discrimina un concepto de otras instancias
- E.j. si ladra es un perro, de lo contrario es otra cosa

## Una definición de concepto puede ser **caracterizante**

- Definición compleja que caracteriza un concepto tan precisamente como sea posible
- E.j. los perros tiene cuatro patas, ladran, tienen cola, etc.

Algunos modelos buscan por el *concepto discriminante más simple*, otros por el *concepto caracterizante más complejo*

# El aprendizaje de funciones implicar aprender el mapeo de valores etiquetados

El objetivo es generalizar un nuevo concepto

- Aprender la función  $f: X \rightarrow S$
- Clasificación:  $f(x) = s \in S$ , donde  $S$  es un conjunto finito de valores
- Regresión:  $f(x) = s \in S$ , donde  $S$  es un rango continuo de valores

**Patrones descriptivos** son cualquier tipo de patrón (incluso si no se lo usa para predicción)

El objetivo es encontrar nuevos patrones

Ejemplos:

- Los autos más rápidos usualmente cuestan más que los más lentos
- Una persona no puede estar casado con más de una persona al mismo tiempo
- En un juego de futbol, gana quien haga más goles (Manchester City dataset)

# Existen diversas técnicas para representar datos de entrada (experiencias)

## Representación numérica

- Sea  $x$  un vector de  $k$  dimensiones,  $x = [x^{(1)}, \dots, x^{(k)}]$ , donde  $x_i \in R^k$
- Para representaciones anotadas, se agrega un nuevo elemento al vector  $x$ :  $x = [x^{(1)}, \dots, x^{(k)}, y]$ , donde  $x_i \in R^k, y \in S$
- Si  $S$  es finito: clasificación
- Si  $S$  es un rango continuo: regresión

Existen diversas técnicas para representar datos de entrada (experiencias)

Representación numérica

$x_i$	id	peso	edad	estatura	rendimiento	estado
	1	65	25	175	0.78	1
	2	72	33	180	0.41	1
	3	80	38	162	0.12	0

Existen diversas técnicas para **representar datos** de entrada (experiencias)

Representación numérica

<b>id</b>	<b>peso</b>	<b>edad</b>	<b>estatura</b>	<b>rendimiento</b>	<b>estado</b>
1	65	25	175	0.78	1
2	72	33	180	0.41	1
3	80	38	162	0.12	0

$$X = \{x_i : i \in N\}$$

Existen diversas técnicas para **representar datos** de entrada (experiencias)

Representación numérica

<b>id</b>	<b>peso</b>	<b>edad</b>	<b>estatura</b>	<b>rendimiento</b>	<b>estado</b>
1	65	25	175	0.78	1
2	72	33	180	0.41	1
3	80	38	162	0.12	0

$$Y = \{y_i : i \in N\}$$

# Existen diversas técnicas para representar datos de entrada (experiencias)

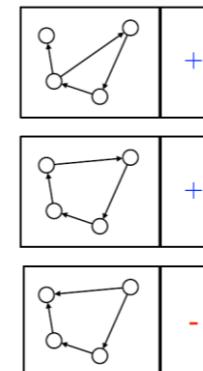
## Representación simbólica

- En este caso,  $x^j, j \in K$  o  $y_i, i \in N$  pueden ser conjuntos del tipo  $\{True, False\}, \{alto, medio, bajo\}$ ; es decir, datos categóricos
- Muchos modelos no pueden hacer uso de estos datos directamente → hay que transformarlos a numéricos

# Existen diversas técnicas para **representar datos** de entrada (experiencias)

## Representación estructural o relacional

- Las instancias tienen una estructura interna: secuencias, grafos, conjuntos. E.j. moléculas, tours, etc.
- Convertirlos en un formato más simple es muy improbable
- No muchas técnicas pueden manejar este tipo de representación



La tarea de aprendizaje y el algoritmo seleccionado, dependen fuertemente del tipo de estructura que tendrán los datos

En este curso: Aprender de datos del tipo “atributo-valor”

# Revisión de algunas técnicas para aprendizaje

## Técnicas simbólicas

- Version Spaces, inducción de árboles de decisión, inducción de conjuntos de reglas, programación lógica inductiva, etc.

## Técnicas numéricas

- Redes neuronales, random forest, support vector machines, etc.

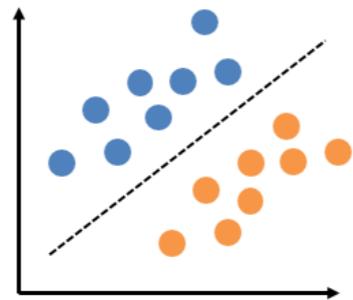
## Técnicas probabilísticas

- Aprendizaje bayesiano

## Técnicas misceláneas

- Algoritmos genéticos, reinforcement learning

# Content

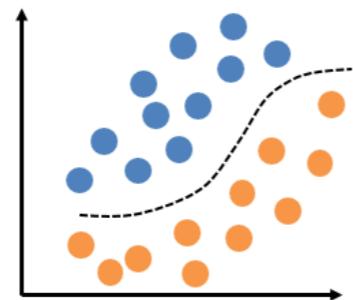


Notación y definiciones

Sets, listas, matrices, etc.

Aprendizaje basado en instancias

Instancias y mapeos



Version spaces

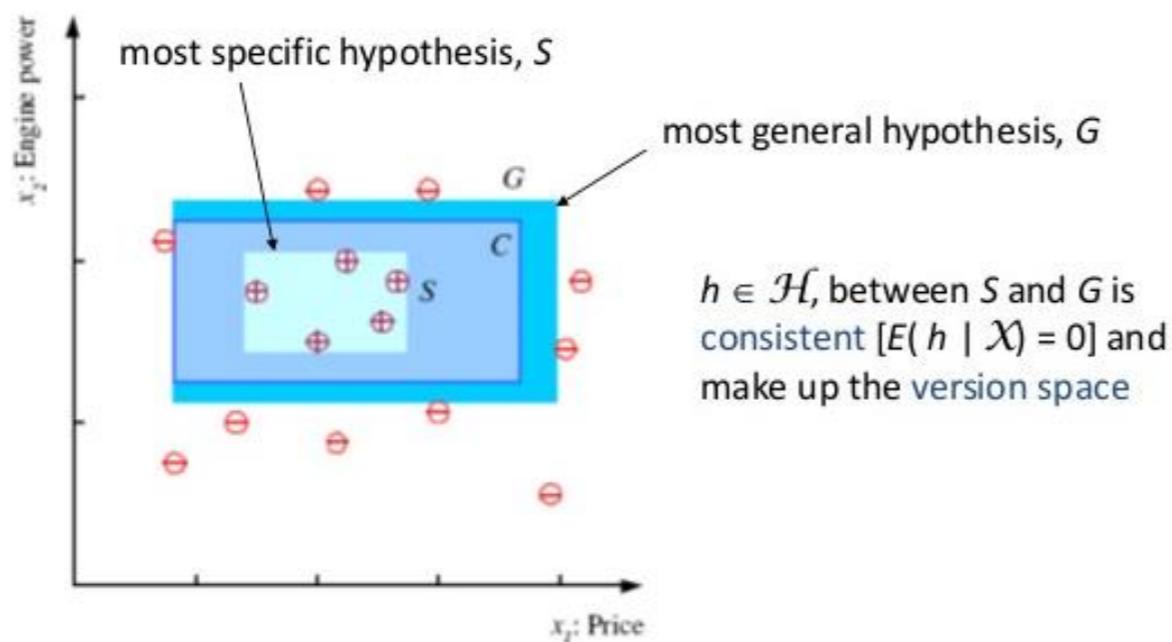
Hipótesis, conceptos, aprendizaje

Bias

Suposiciones que hacen los modelos

**Version Spaces** es una técnica básica  
que introduce conceptos importantes

## S, G, and the Version Space



Introduce conceptos como hipótesis y generalidad que son útiles para otras técnicas más avanzadas

Un **concepto** es un subconjunto de elementos de un conjunto más amplio

Sea  $X$  un conjunto de objetos

$$X = \{Scooby Doo, El oso Yogui, Tin Tin, Lassie, Transformers\}$$

Un concepto  $C$  se define como un subconjunto (lógico) de  $X$

$$C = \{Scooby Doo, Lassie\}, \text{ donde } C \text{ representa perros}$$

Así mismo, este concepto  $C$  retorna 1 si un elemento existe

$$C(Lassie) = 1, C(Tin Tin) = 0$$

Ejemplo: el concepto es “días en los que Juan disfruta de su deporte favorito”

INPUT

OUTPUT

Sky	Temp	Humid	Wind	Water	Forecast	C(x)
sunny	warm	normal	strong	warm	same	1
sunny	warm	high	strong	warm	same	1
rainy	cold	high	strong	warm	change	0
sunny	warm	high	strong	cool	change	1

El **espacio de hipótesis** es el subconjunto de todos los conceptos posibles

Una hipótesis  $h$  es un conjunto de constraints en atributos

- Puede ser un valor: Water=Warm
- Un “cualquier” valor: Water=?
- Un valor no permitido: Water=∅

Si una instancia  $x$  satisface los constraints de  $h$ :  $h(x) = 1$ ; de otro modo  $h(x) = 0$

Ejemplo de hipótesis:

Sky	Temp	Humid	Wind	Water	Forecast
sunny	?	?	?	?	?
?	warm	?	?	?	same

# Concept Learning como una búsqueda

Dado un espacio de hipótesis  $H$  y un conjunto de datos  $S$

- $H$  es un conjunto de definiciones de conceptos posibles
- $S$  consiste en ejemplos positivos y negativos de algún concepto

Encontrar todos los  $h \in H$  consistentes con  $S$

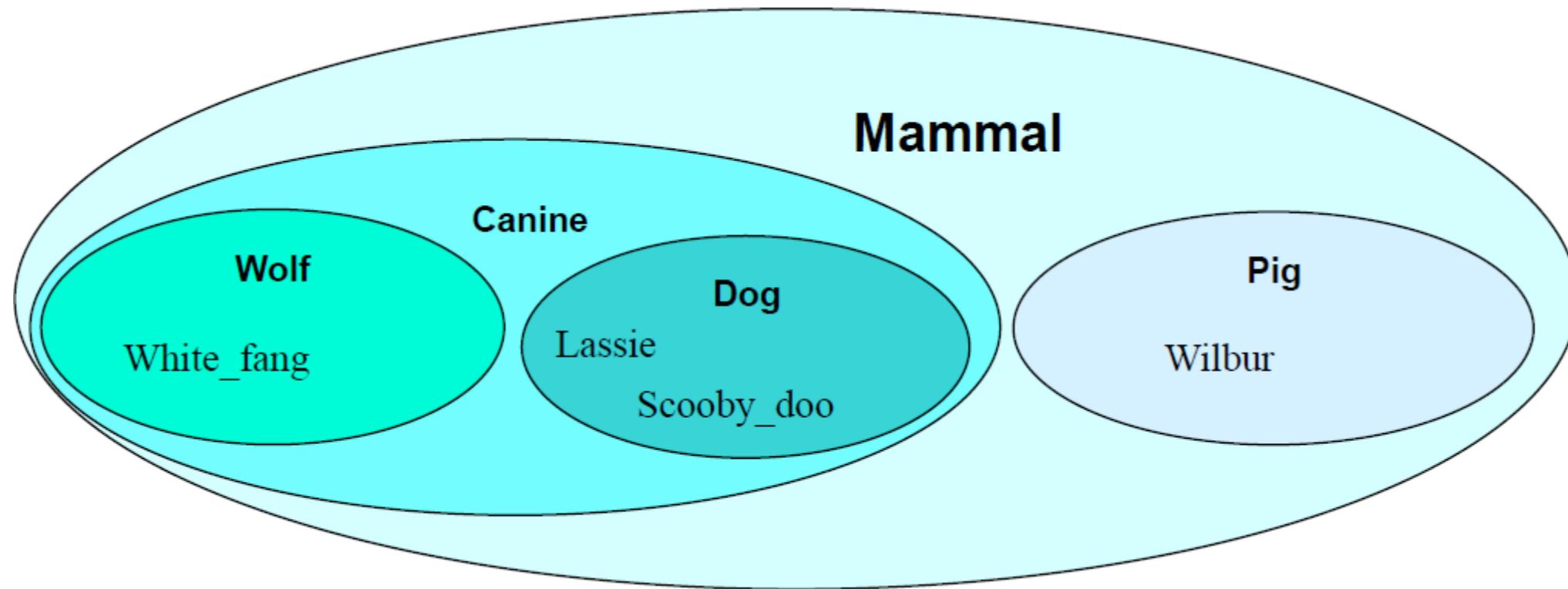
- Consistente significa que cubre todos los ejemplos positivos y negativos

Búsqueda: enumerar todos los  $h \in H$  (no es posible, son muchos!)

Prune search con un orden basado en generalidad

- $h_1$  es más general que  $h_2 \leftrightarrow (x \in h_2 \rightarrow x \in h_1)$
- Dicho de otra manera  $\text{cobertura}(h_1) \subseteq \text{cobertura}(h_2)$

Un concepto  $P$  es más general que otro  $Q$ , si las instancias representadas en  $P$  están también en  $Q$



# El concepto de generalidad produce un orden implícito

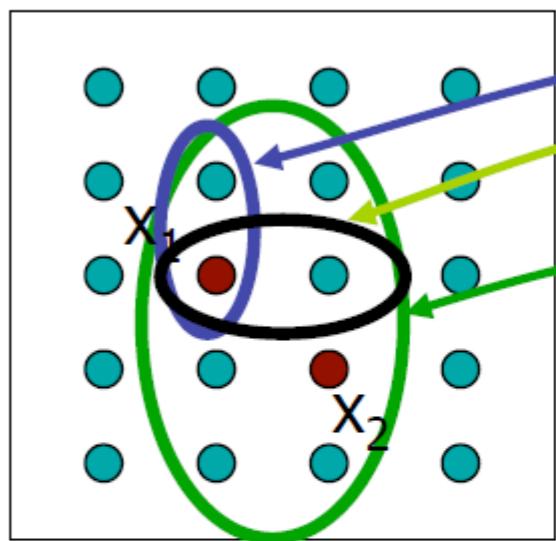
$h_1 = \langle Sunny, ?, ?, Strong, ?, ? \rangle$

$h_2 = \langle Sunny, ?, ?, ?, ?, ?, ? \rangle$

Definición:  $h_j$  es más genera que  $h_k$  iff

$$h_j \geq_g h_k \equiv \forall x(h_k(x) = 1 \rightarrow h_j(x) = 1)$$

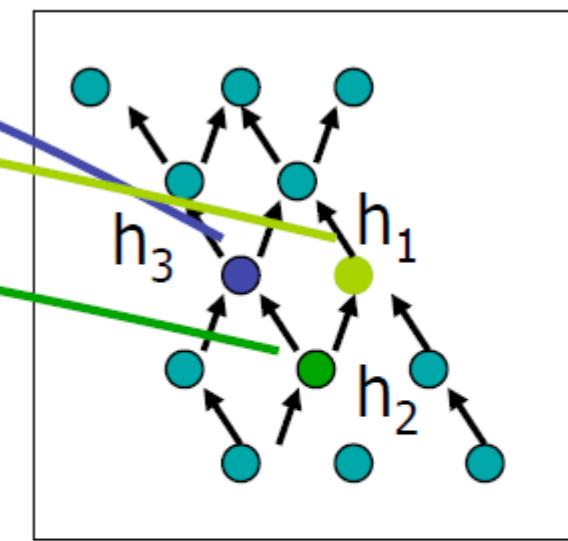
## Instances



$x_1 = < \text{Sunny}, \text{Warm}, \text{High}, \text{Strong}, \text{Cool}, \text{Same} >$

$x_2 = < \text{Sunny}, \text{Warm}, \text{High}, \text{Light}, \text{Warm}, \text{Same} >$

## Hypotheses



$h_1 = < \text{Sunny}, ?, ?, \text{Strong}, ?, ? >$

$h_2 = < \text{Sunny}, ?, ?, ?, ?, ? >$

$h_3 = < \text{Sunny}, ?, ?, ?, \text{Cool}, ? >$

specific  
↑  
↓ general

**Version spaces** es el conjunto de todas las hipótesis que contiene ejemplos positivos únicamente

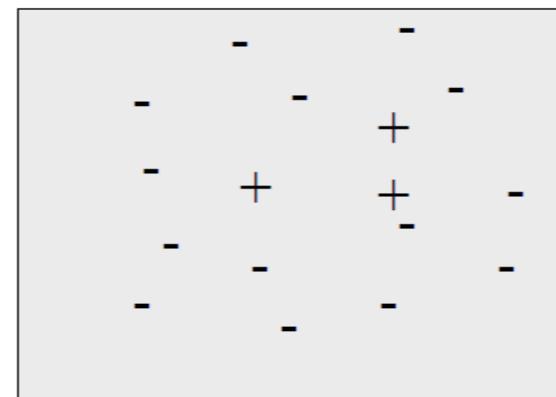
$+$ : instancias que pertenecen al concepto

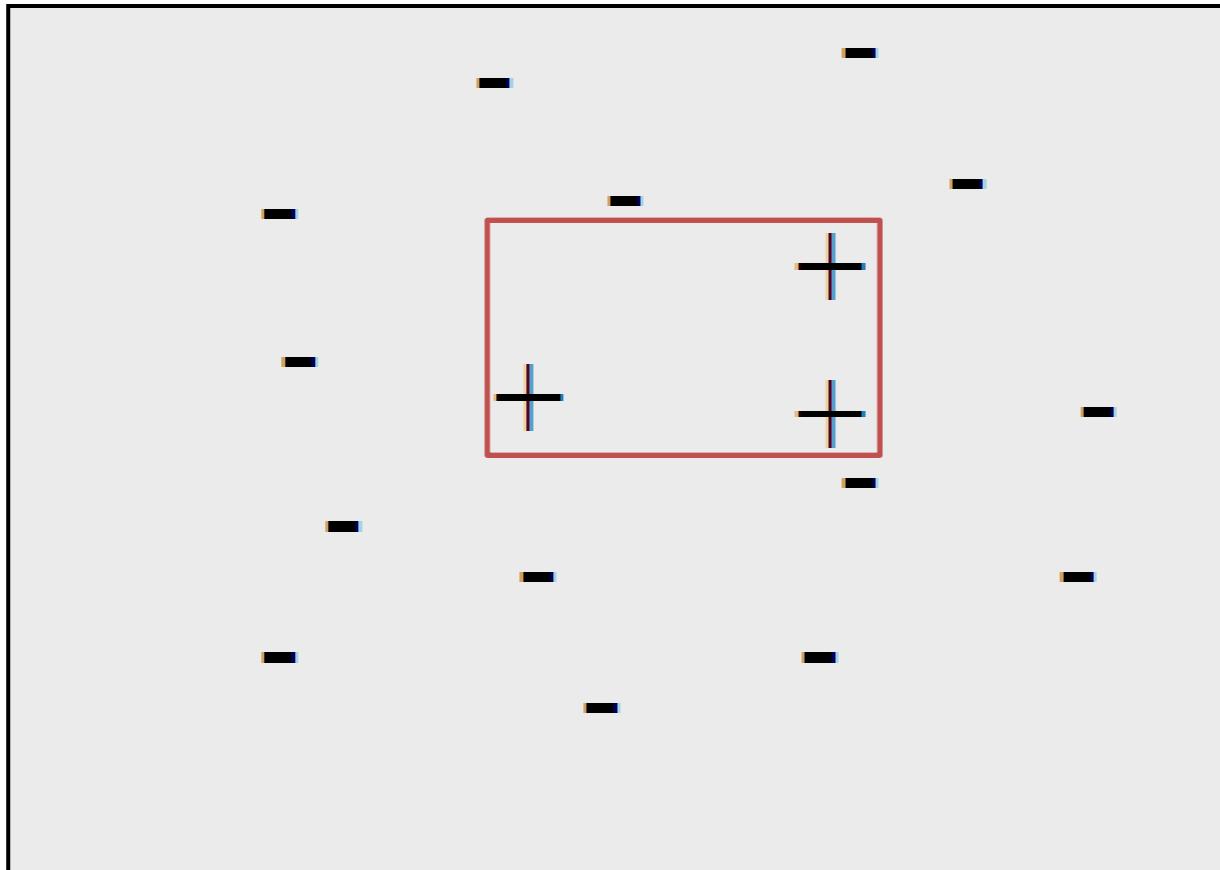
- $S$  consiste en ejemplos positivos y negativos del concepto
- $S$  consiste en ejemplos positivos y negativos del concepto

Asumir que las hipótesis son rectángulos, i.e.  $H$  es el conjunto todos los rectángulos

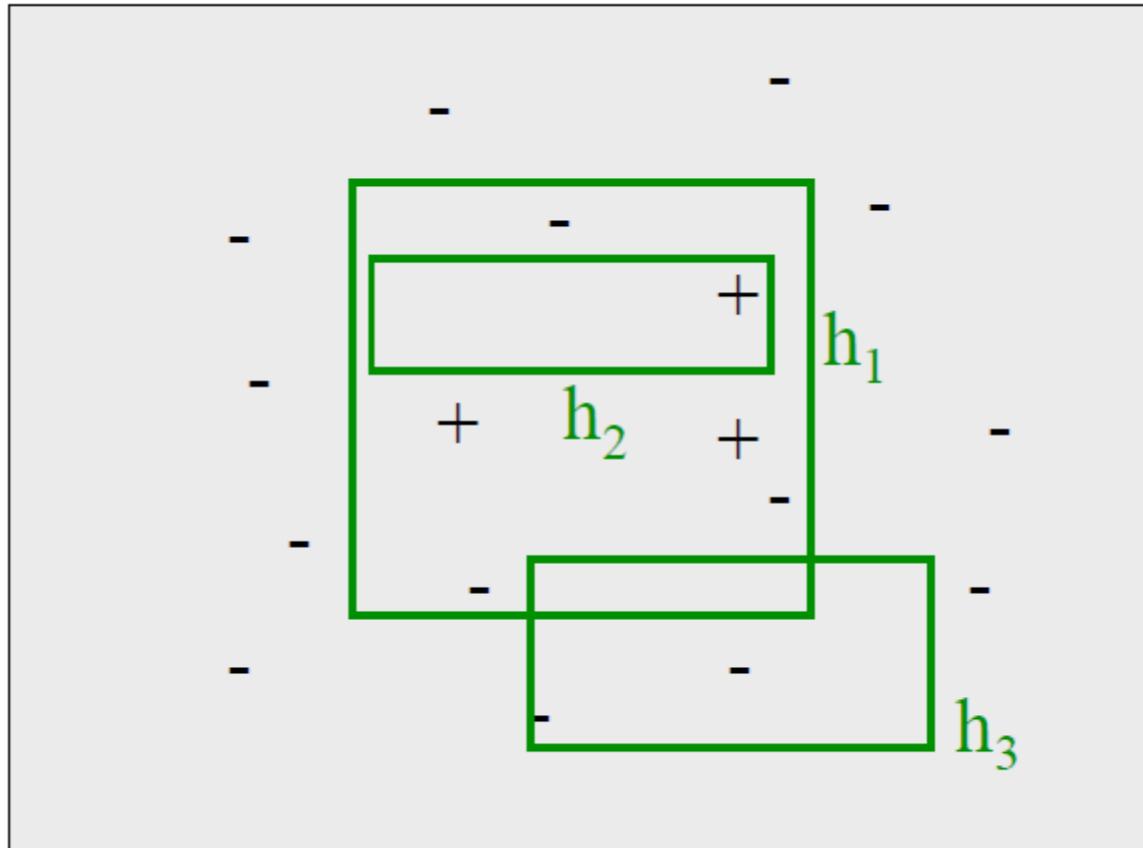
- Rectángulo: concepto. Representado por
- $\langle [x_l, x_u], [y_l, y_u] \rangle = \{(x, y) | x_l < x < x_u \text{ and } y_l < y < y_u\}$

$VS(H, S)$ : conjunto de todos los rectángulos que contiene  $+$  únicamente





Ejemplo de hipótesis consistente



$h_1$  more general than  $h_2$   
=  $h_2$  more specific than  $h_1$

$h_3$  incomparable with  $h_1$

Con la generalidad se pueden construir  
redes de hipótesis en un versión space

Bordes/fronteras de un versión space

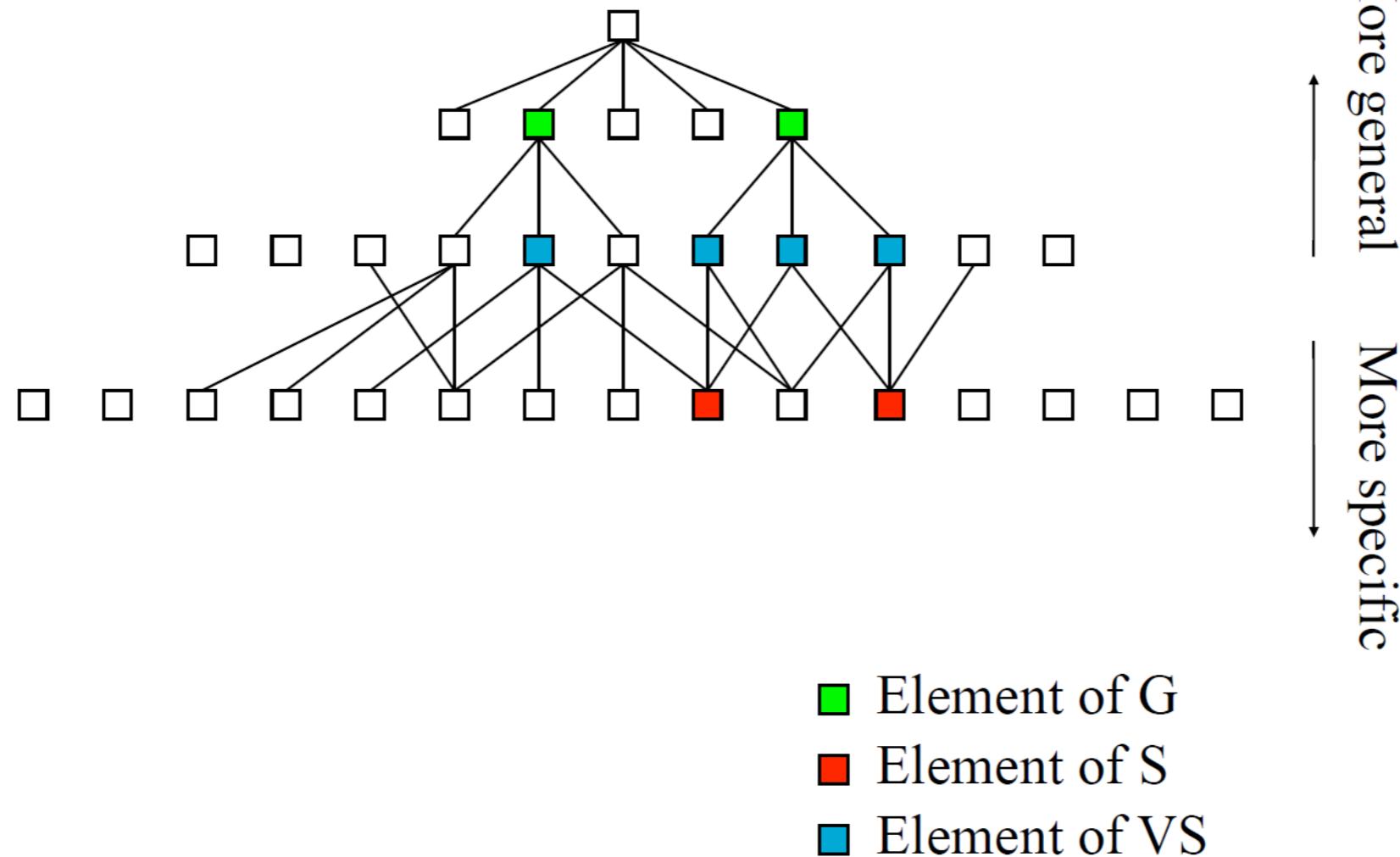
- mas\_especifico(S)
- mas\_general(G)

Cualquier hipótesis  $h$  consistente debe ser

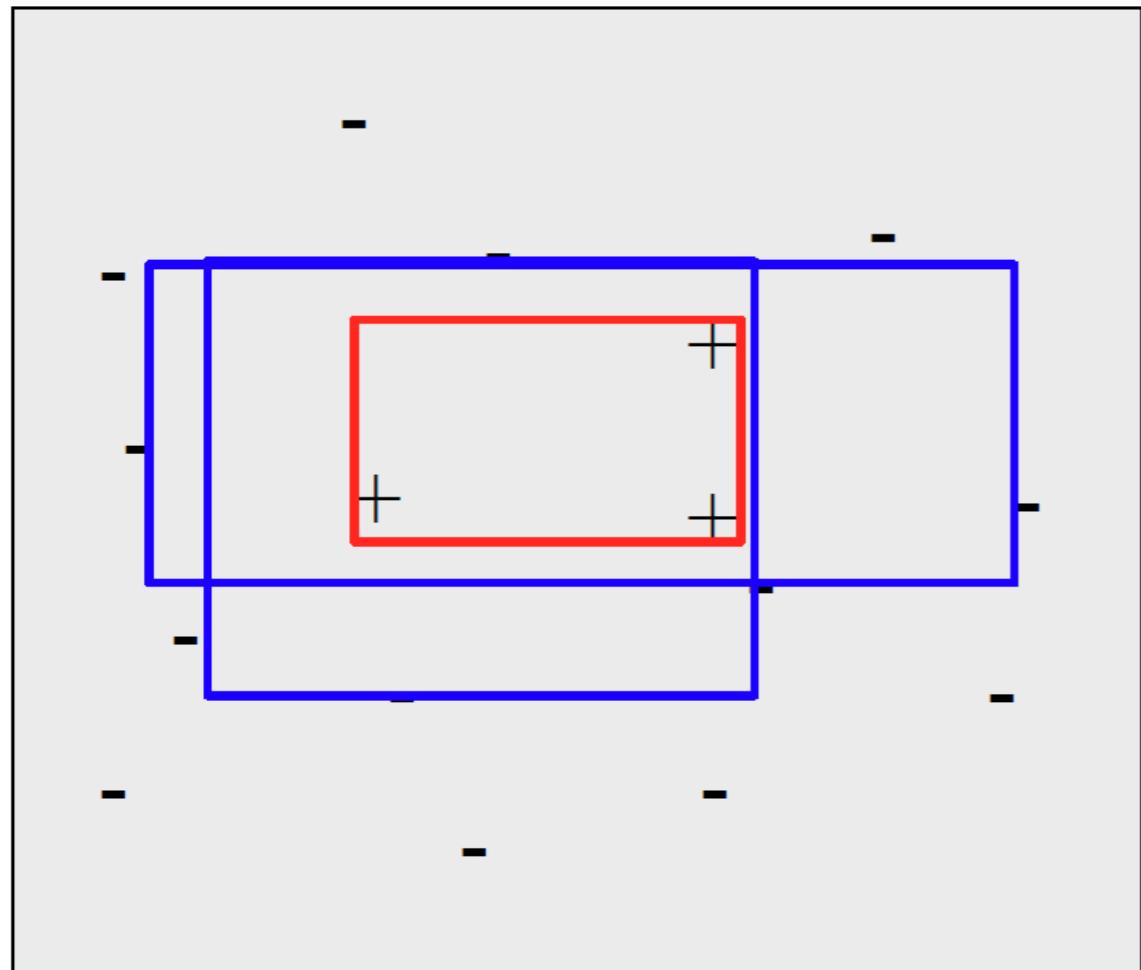
- Mas general que algún elemento en S
- Más específica que algún elemento en G

$G$  y  $S$  especifican completamente el versión space VS

# *Generality lattice*



En donde se encuentran  $S$  y  $G$  en el gráfico inicial



$h_1$ : most specific hypothesis

$h_2$ : most general hypothesis

$h_3$ : another most general hyp.

$$S = \{h_1\}, G = \{h_2, h_3\}$$

Si se calcula  $G$  y  $S$ , es suficiente para conocer  $VS$

Algoritmos populares para  $VS$

- *FindS*: calcula únicamente el conjunto  $S$
- *Candidate Elimination*: calcula  $S$  y  $G$

No se ahondarán en estos algoritmos, por que  $VS$  es cubierta en el curso mayormente como fuente de conceptos importantes

VS es impráctico por que  
no es eficiente y robusto

**VS provee un framework teórico muy útil**

Conceptos como hipótesis y generalidad son compartidos por otros métodos

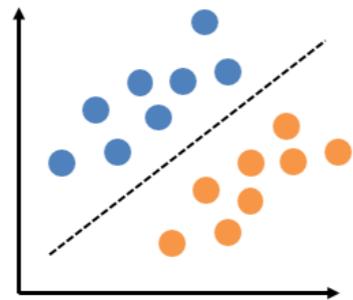
**No es práctico para la mayoría de problemas**

VS colapsa cuando no existen hipótesis consistentes

**No es robusta a ruido**

Ruido=instancias mal etiquetadas (problema común en ML)

# Content



**Notación y definiciones**

Sets, listas, matrices, etc.

**Aprendizaje basado en instancias**

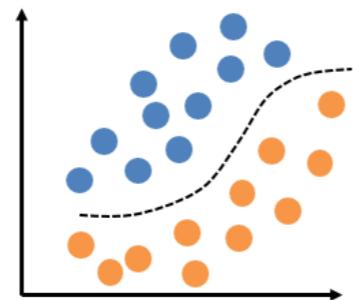
Instancias y mapeos

**Version spaces**

Hipótesis, conceptos, aprendizaje

**Bias**

**Suposiciones que hacen los modelos**



**Inductive Bias** es el conjunto mínimo de supuestos que garantiza el correcto salto inductivo

Después de haber visto los ejemplos

¿Se pueden hacer predicciones con ejemplos nuevos? – generalización!

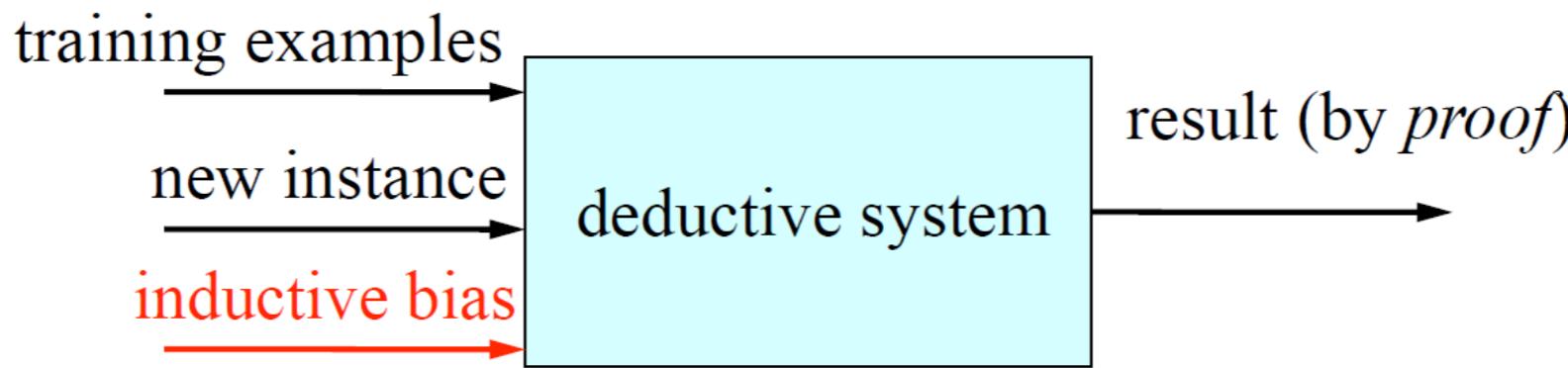
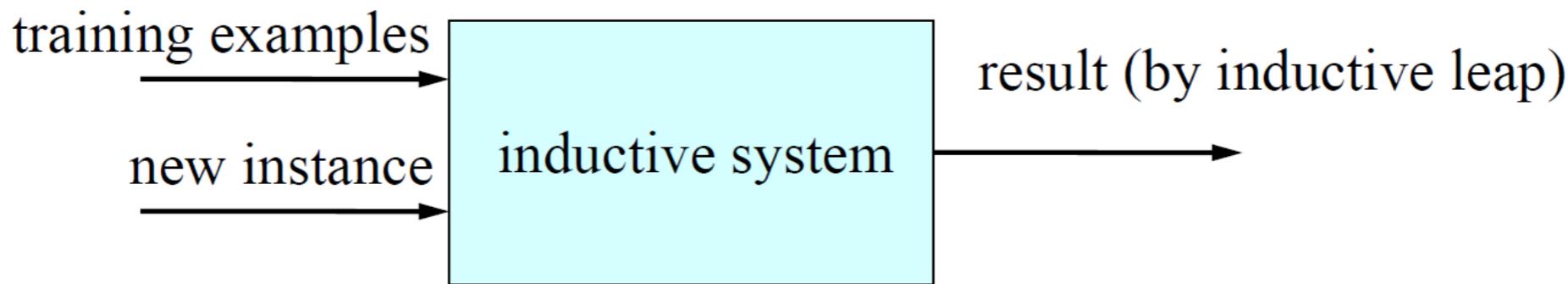
¿Hay alguna garantía del aprendizaje?

Bajo qué circunstancias se puede garantizar

El salto inductivo

Es el ir de ejemplos conocidos a ejemplos no conocidos

# Sistema inductivo vs Sistema Deductivo



**Inductive Bias** es el conjunto mínimo de supuestos que garantiza el correcto salto inductivo

Formalmente (Mitchell)

- $L(x, D)$  denota una clasificación asignada a una instancia  $x$  por el modelo  $L$  luego de haber sido entrenado con los datos  $D$
- $|-$  denota algún método de prueba

El inductive bias de  $L$  se define como el conjunto mínimo de supuestos  $B$  tal que para cualquier concepto  $c$  y datos de entrenamiento  $D_c$ :

- $\forall x \in X: B \wedge D_c \wedge x | - L(x, D_c)$

El conjunto mínimo de supuestos para probar que el modelo aprenderá modelos únicamente correctos

# Diferentes algoritmos de aprendizaje pueden tener diferente bias

Esto explica por que diferentes algoritmos proveen distintos resultados

Mayor bias implica menos aprendizaje

- El bias implica que existen mayores supuestos → menos generalización

¿Se puede aprender algo sin bias?

- NO

Cuando se escoge un algoritmo, el bias es un criterio importante

- Sin embargo, el bias en muchos algoritmos no es un concepto muy bien entendido

# Teorema del No Free Lunch



La imposibilidad de aprender sin bias

- Teoremas del “no free lunch”

Si existen un problema  $P$  para el cual el algoritmo  $L_1$  tiene mejores resultados que  $L_2$ , existe un problema  $P'$  para el cual  $L_2$  tiene mejores resultados que  $L_1$

No existe “el mejor” algoritmo para cualquier problema!



UNIVERSIDAD DE CUENCA  
*desde 1867*

# Introducción: ¿Qué es el aprendizaje automático?

Andres Auquilla  
2020

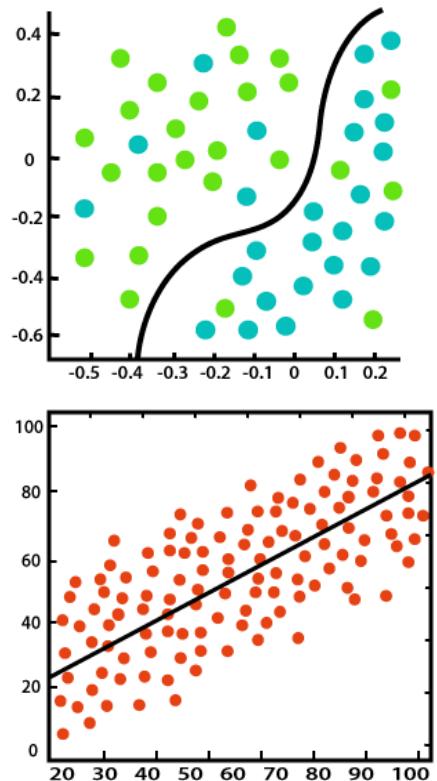


**UNIVERSIDAD DE CUENCA**  
*desde 1867*

# Aprendizaje Supervisado

Andres Auquilla  
2020

# Content



**Arboles de decisión**

Como se crean, conceptos

**Inducción de conjuntos de reglas**

Generalización a través de reglas

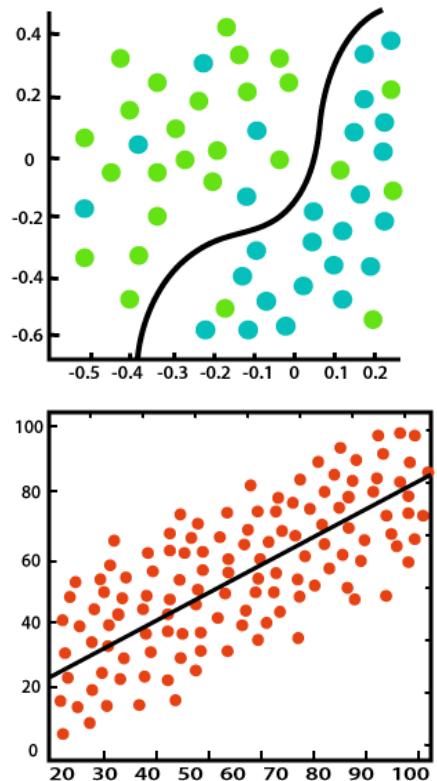
**Support Vector Machines**

Kernels, hyperparámetros

**Artificial Neural Networks**

Nuevas arquitecturas y aplicaciones

# Content



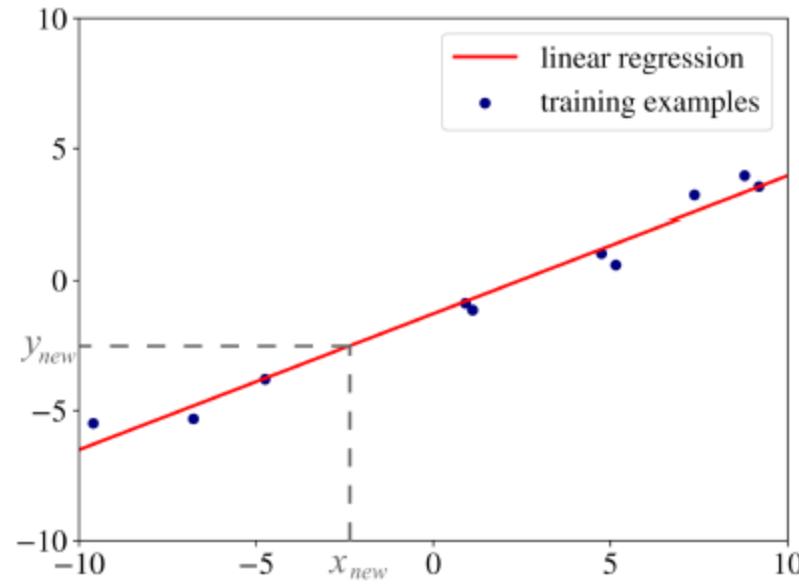
Arboles de decisión  
Como se crean, conceptos

Inducción de conjuntos de reglas  
Generalización a través de reglas

Support Vector Machines  
Kernels, hyperparámetros

Artificial Neural Networks  
Nuevas arquitecturas y aplicaciones

# Regresión lineal es un algoritmo que aprende un modelo como una combinación de variables de entrada



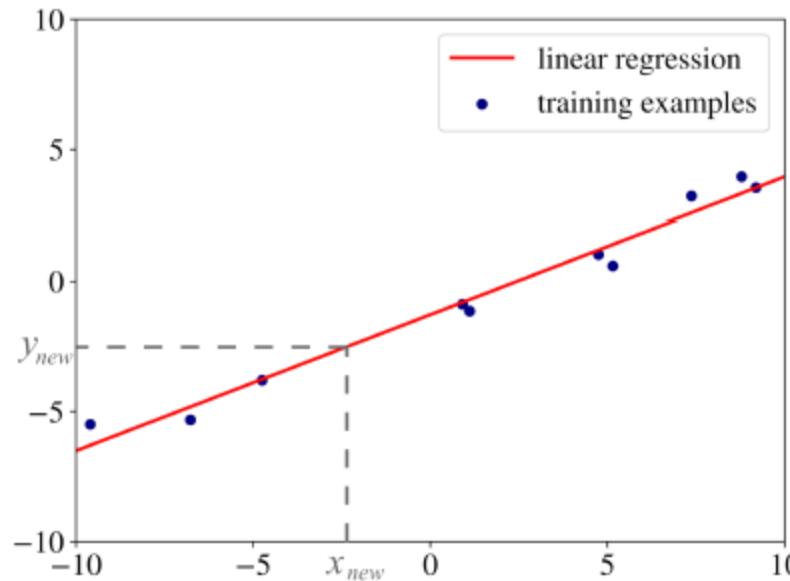
Entrada:

$\{(x_i, y_i)\}_{i=1}^N, y_i \in \mathbb{R}$ , donde  $N$  es el número de ejemplos y el vector de características  $x_i^{(j)} \in \mathbb{R}, j = 1, \dots, D$ , donde  $D$  es el número de dimensiones del vector

Modelo:

$f_{w,b}(x) = wx + b$ , donde  $w$  es un vector de  $D$ -dimensiones y  $b \in \mathbb{R}$

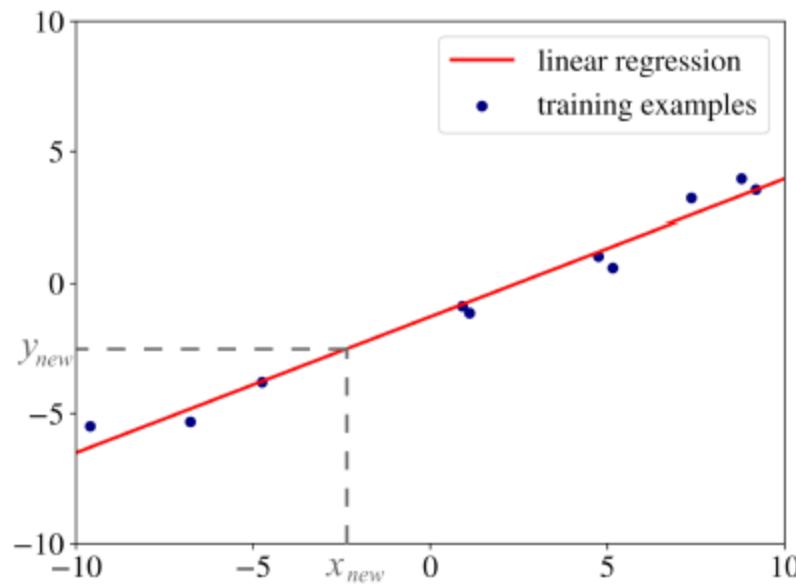
# Regresión lineal es un algoritmo que aprende un modelo como una combinación de variables de entrada



Predicción:  
 $y \leftarrow f_{w,b}(x)$

Optimizar:  
Los parámetros  $w$  y  $b$  deben ser calculados mediante alguna técnica de optimización

# Regresión lineal es un algoritmo que aprende un modelo como una combinación de variables de entrada

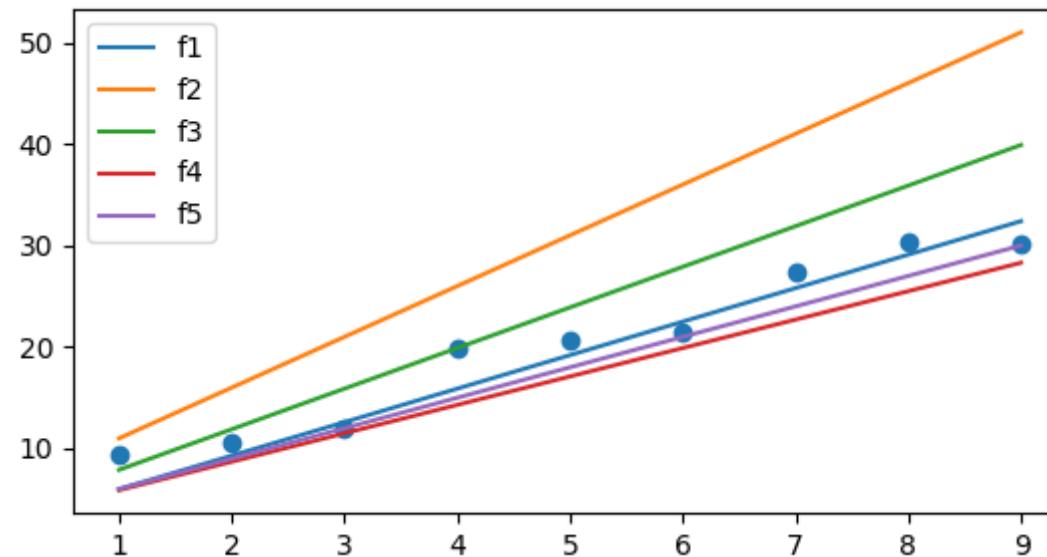


En este caso D=1, producto de la regresión se tiene una línea

Cuando D=2, se tiene un plano

Cuando D>3, se tiene un hyper-plano

Dependiendo del valor de **w** y **b**, pueden existir varias rectas que pasen cerca de los puntos



Por esta razón,  $w$  y  $b$  deben ser estimados de la forma más optima posible

En la gran mayoría de algoritmos de ML, los parámetros se encuentran mediante **optimización**

Para el algoritmo de regresión lineal se debe minizar:

$$\frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2$$

— Función objetivo

También conocida como cost function

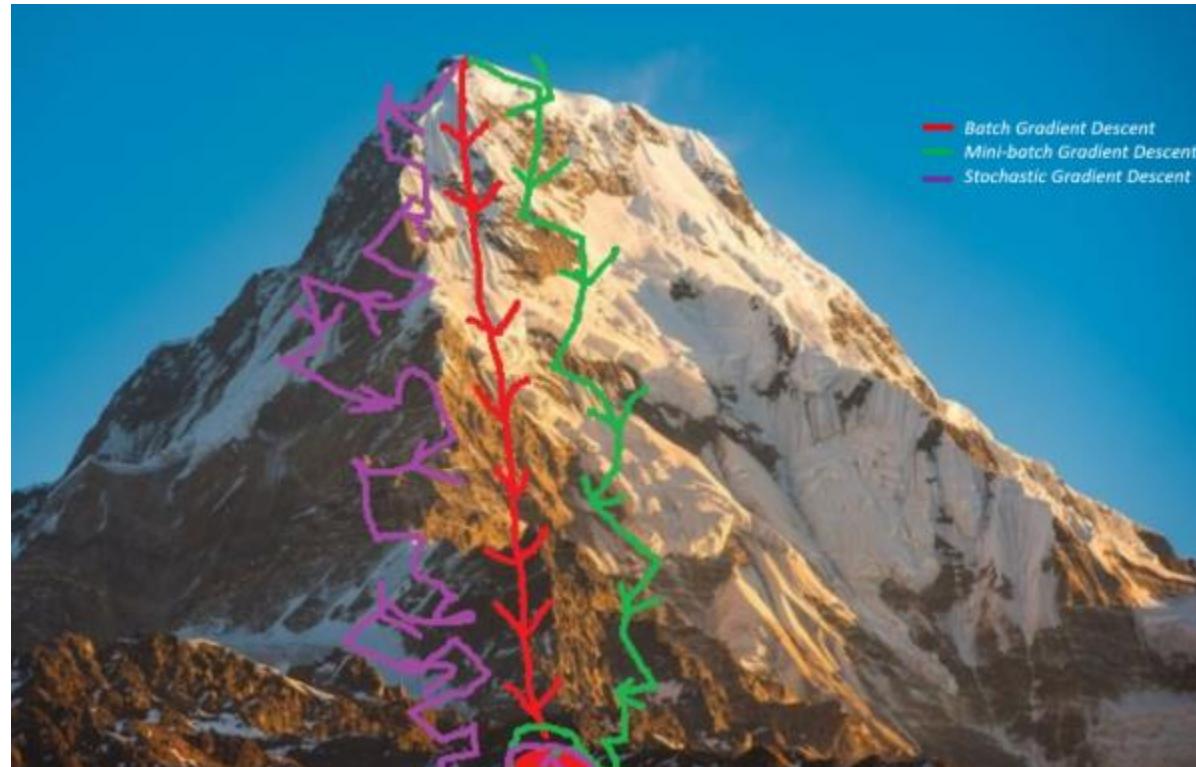
En la gran mayoría de algoritmos de ML, los parámetros se encuentran mediante **optimización**

Para el algoritmo de regresión lineal se debe minizar:

$$\frac{1}{N} \sum_{i=1}^N (f_{w,b}(x_i) - y_i)^2 \quad \text{—— Loss function}$$

¿Porque se utiliza una función cuadrática?

# Gradient Descent y sus variantes es el algoritmo de optimización más comúnmente utilizado en ML

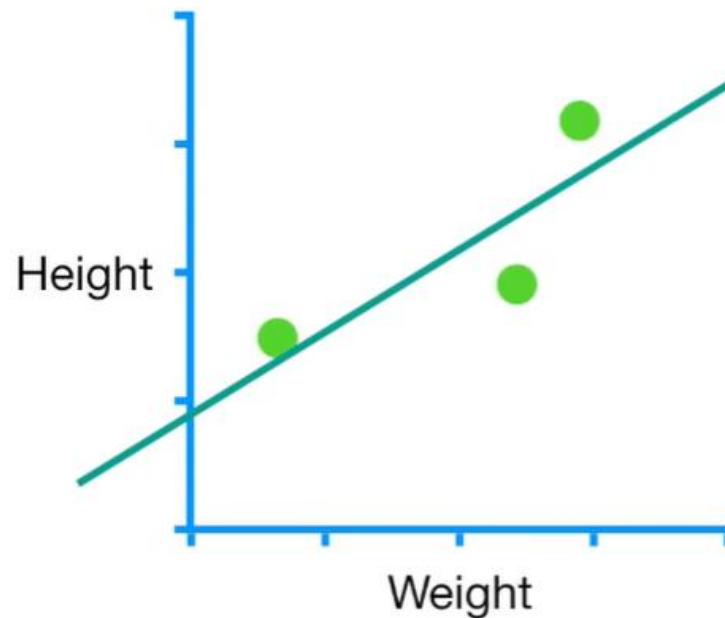


Algoritmo para estimar parámetros en un algoritmo

En base a la función de gradiente, encuentra el valor óptimo

Función iterativa

# Gradiente Descent utiliza la gradiente de la función para encontrar el óptimo global



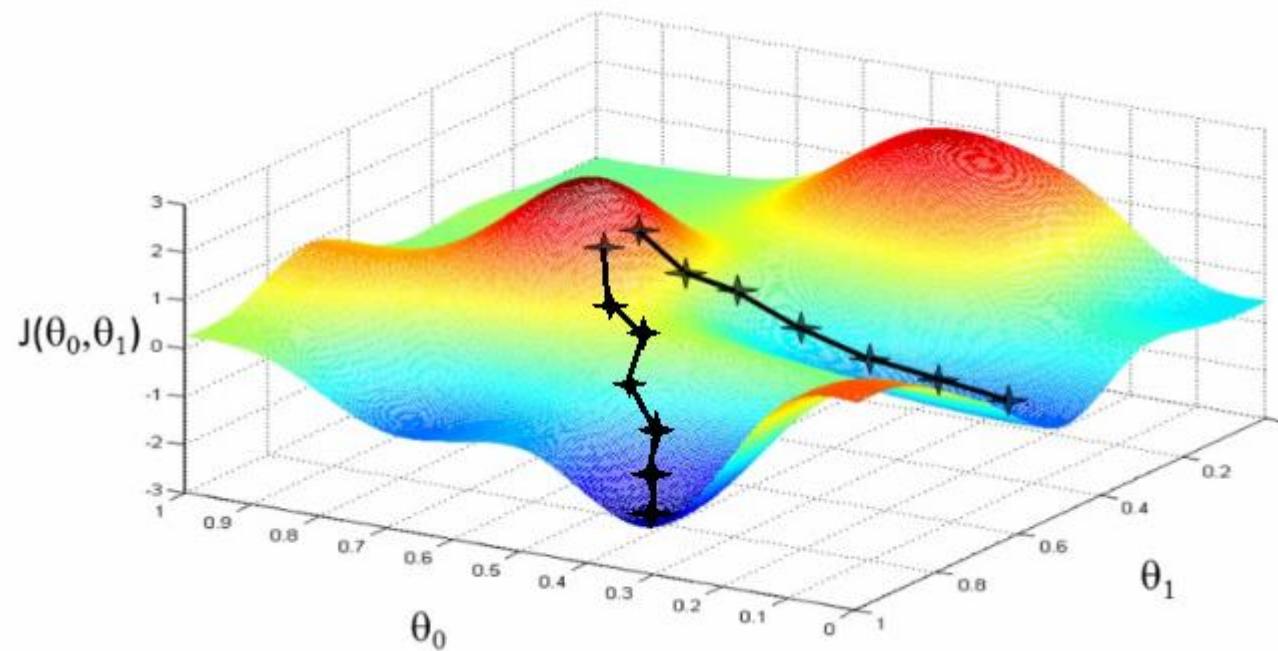
Calcula la gradiente de la loss function para cada parámetro

Inicializa aleatoriamente los parámetros

Calcula los step sizes en cada iteración

Actualiza los valores de los parámetros estimados

Cuando se deben estimar dos parámetros y la función objetivo es compleja, pueden haber varios optimos locales



Gradient Descent encuentra la mejor dirección para descender

# Averiguar

- Mini-Batch & Stochastic Gradient Descent
- Least squares regression line

# Logistic Regression es otro modelo lineal ampliamente utilizado en ML

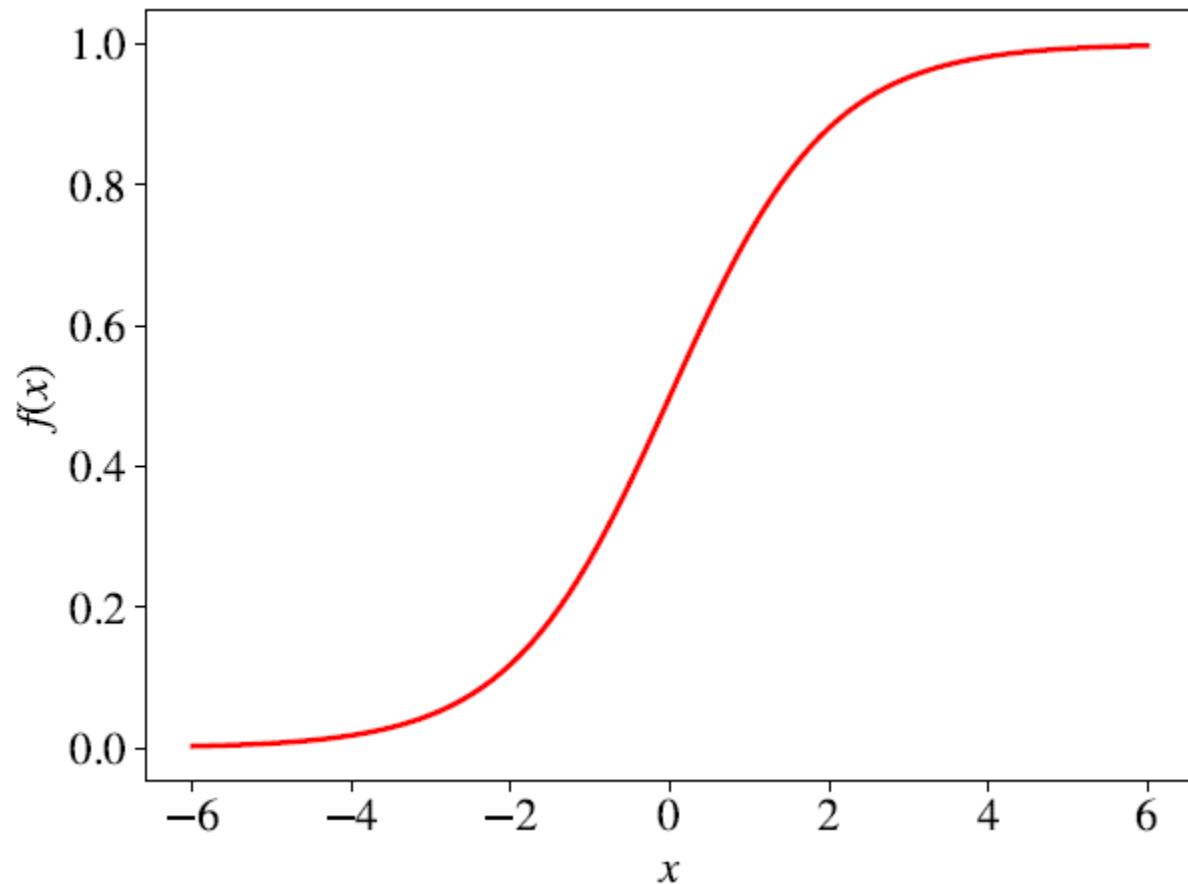
Se analiza en términos de clasificación binaria:

- La idea es encontrar una función para mapear  $x_i$  a  $y_i \in \{0, 1\}$
- Para ello se puede utilizar una logistic function (sigmoid)

$$f(x) = \frac{1}{1 + e^{-x}}$$

- El modelo de regresión logística se define como:
- $f_{w,b} \stackrel{\text{def}}{=} \frac{1}{1+e^{-(wx+b)}}$
- Nuevamente, se deben estimar los parámetros  $w$  y  $b$

$f(x)$  está en el rango de  $[0, 1]$  y esta salida se interpreta como probabilidad de pertenecer a una u otra categoría



Si  $f(x) \sim 1$ ,  $x$  tiene alta probabilidad de pertenecer a la clase 1

Si  $f(x) \sim 0$ ,  $x$  tiene alta probabilidad de pertenecer a la clase 0

Se debe escoger un threshold para determinar la pertinencia a una clase u otra

En logistic regresión se encuentran  $w$  y  $b$  maximizando el likelihood de los datos de acuerdo al modelo

Dados los datos de entrenamiento  $\{(x_i, y_i)\}_1^N, y_i \in \{0, 1\}$

$$L_{w,b} \stackrel{\text{def}}{=} \prod_{i=1}^N f_{w,b}(x_i)^{y_i} \left(1 - f_{w,b}(x_i)\right)^{(1-y_i)}$$

El criterio de optimización en LR se denomina  
**maximum likelihood**

En logistic regresión se encuentran  $w$  y  $b$  maximizando el likelihood de los datos de acuerdo al modelo

Dados los datos de entrenamiento  $\{(x_i, y_i)\}_1^N, y_i \in \{0, 1\}$

$$L_{w,b} \stackrel{\text{def}}{=} \prod_{i=1}^N f_{w,b}(x_i)^{y_i} \left(1 - f_{w,b}(x_i)\right)^{(1-y_i)}$$

Se utiliza la función  $\prod$  por que se trabaja con probabilidades y eventos independientes

En logistic regresión se encuentran  $w$  y  $b$  maximizando el likelihood de los datos de acuerdo al modelo

Dados los datos de entrenamiento  $\{(x_i, y_i)\}_1^N, y_i \in \{0, 1\}$

$$LogL_{w,b} \stackrel{\text{def}}{=} \ln(L_{w,b}) = \sum_{i=1}^N [y_i \ln f_{w,b}(x) + (1 - y_i) \ln(1 - f_{w,b}(x))]$$

Para evitar problemas de cálculo computacional, se suele maximizar el **log-likelihood**

Los modelos lineales imponen muchas limitaciones cuando existe alta dimensionalidad

Una regresión lineal:

$$y(x, w) = w_0 + w_1 x^{(1)} + \cdots + w_D x^{(D)}$$

Una regresión no lineal:

$$y(x, w) = w_0 + \sum_{j=1}^{M-1} w_j \theta_j(x)$$

$$y(x, w) = \sum_{j=0}^{M-1} w_j \theta_j(x) = w^T \theta(x), \theta_0(x) = 1$$

La función  $\theta(x)$  (basis function) es no lineal y tiene parámetros que deben ser estimados

Existen muchas opciones  
para las basis functions

Gausiana

$$\theta(x) = \exp\left(-\frac{(x - \mu_j)^2}{2s^2}\right)$$

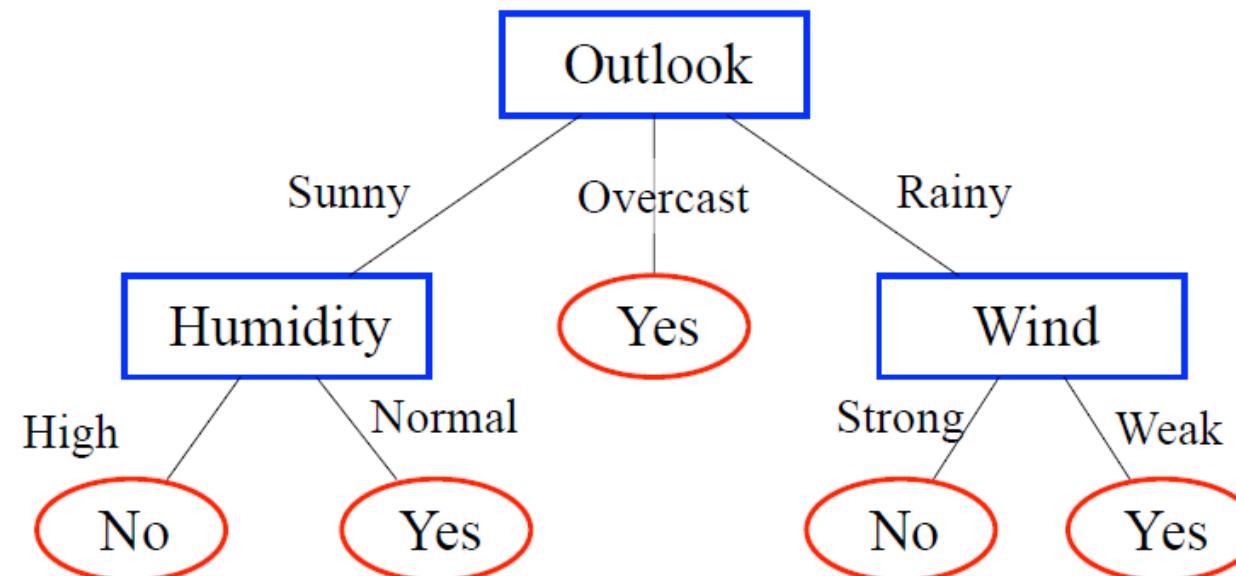
Sigmoid

$$\theta(x) = \sigma\left(\frac{x - \mu_j}{S}\right), \quad \sigma(a) = \frac{1}{1 + \exp(-a)}$$

Un **árbol de decisión** hace preguntas dependiendo de la respuesta anterior

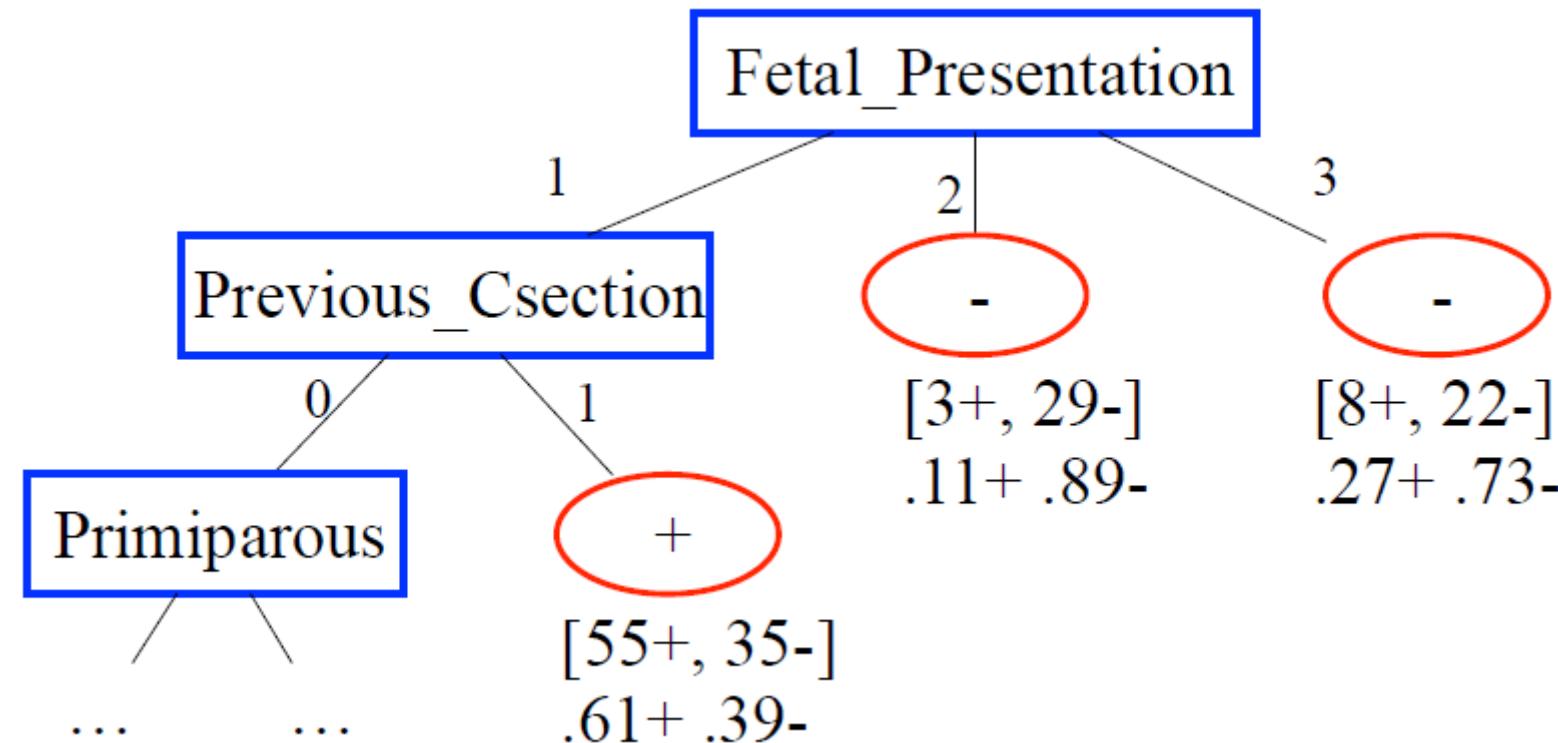
La respuesta se la encuentra al final (hoja)

- Usualmente se utilizan para clasificación



Muchas veces los resultados en  
las hojas no son puros

Ejemplo: ¿Es necesaria una cesárea?



Un árbol de decisión puede representar cualquier operación booleana y permite ruido

Cada nodo representa un atributo

Un atributo puede ser numérico o categórico

Un nodo tiene tantos hijos como posible salidas

Más visible con datos categóricos. Con datos numéricos: binning

Los nodos hoja asignan la clasificación

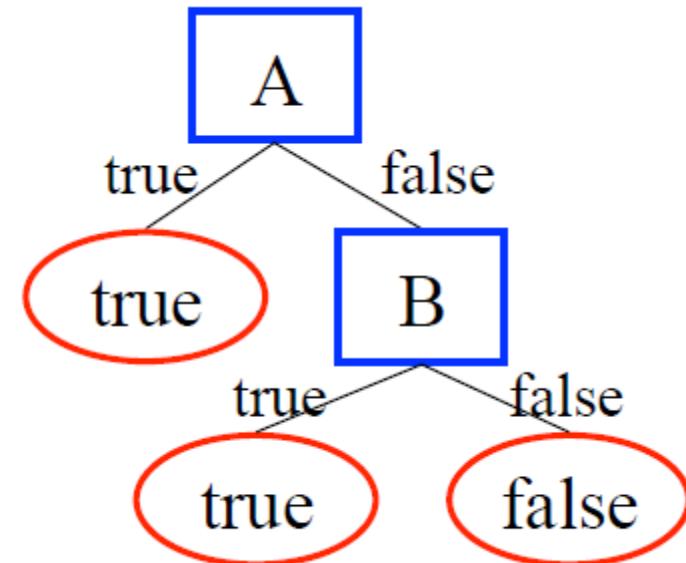
Estos nodos permiten determinar que tan probable es una categoría u otra

Los árboles permiten ruido

Las hojas pueden ser no-puras, i.e. contener valores positivos o negativos

Un árbol puede representar  
cualquier operación booleana

Ejemplo:  $A \vee B$



Así mismo puede representar conceptos  
más complejos

¿Complejidad del árbol vs complejidad de  
la fórmula?

Los árboles pueden ser usados para clasificación, regresión, y clustering

### **Clasificación**

Cada nodo hoja provee una clase

### **Regresión**

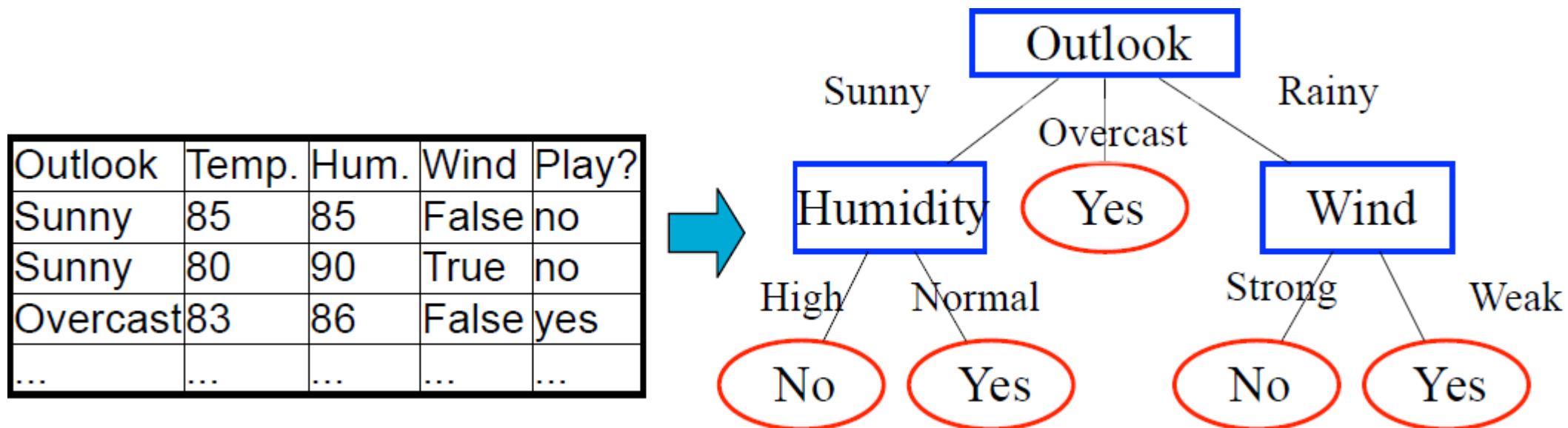
Cada nodo hoja provee un valor

### **Clustering**

Cada nodo hoja agrupa los ejemplos

Un árbol de decisión debería dar una respuesta con el menor número de preguntas posibles

Los árboles pueden ser construidos de forma automática



# Construcción (inducción) de arboles de decisión de forma top-down

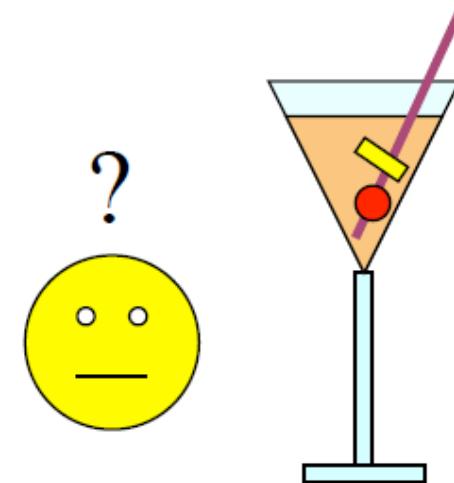
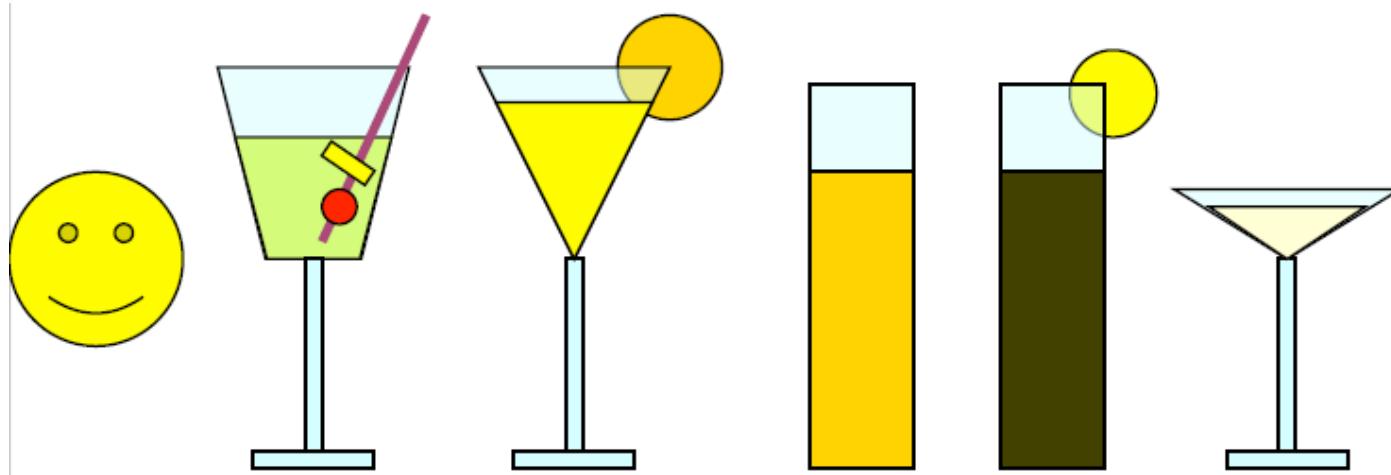
Algoritmo básico de **top down induction of decision trees TDIDT** (basado en el ID3):

- Encontrar buenas particiones: buena= ejemplos puros o similares
- Para cada salida, crear un nodo hijo
- Mover los ejemplos al nodo hijo de acuerdo a la salida
- Repetir el proceso para cada nodo que no sea puro

Preguntas:

- Como decidir que prueba es la mejor (partición)
- Cuando parar el proceso

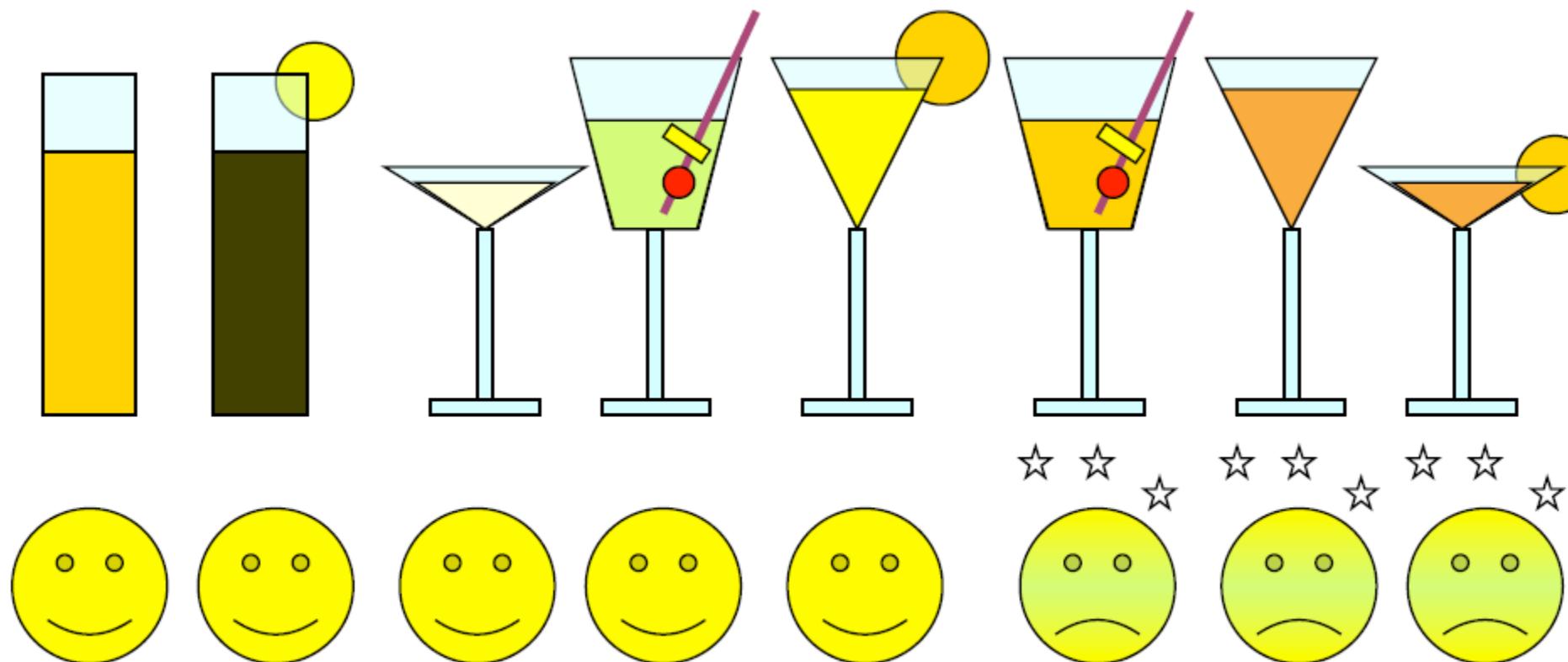
Ejemplo: ¿Este trago me  
sienta bien o no?



Is this drink going to  
make us ill, or not?

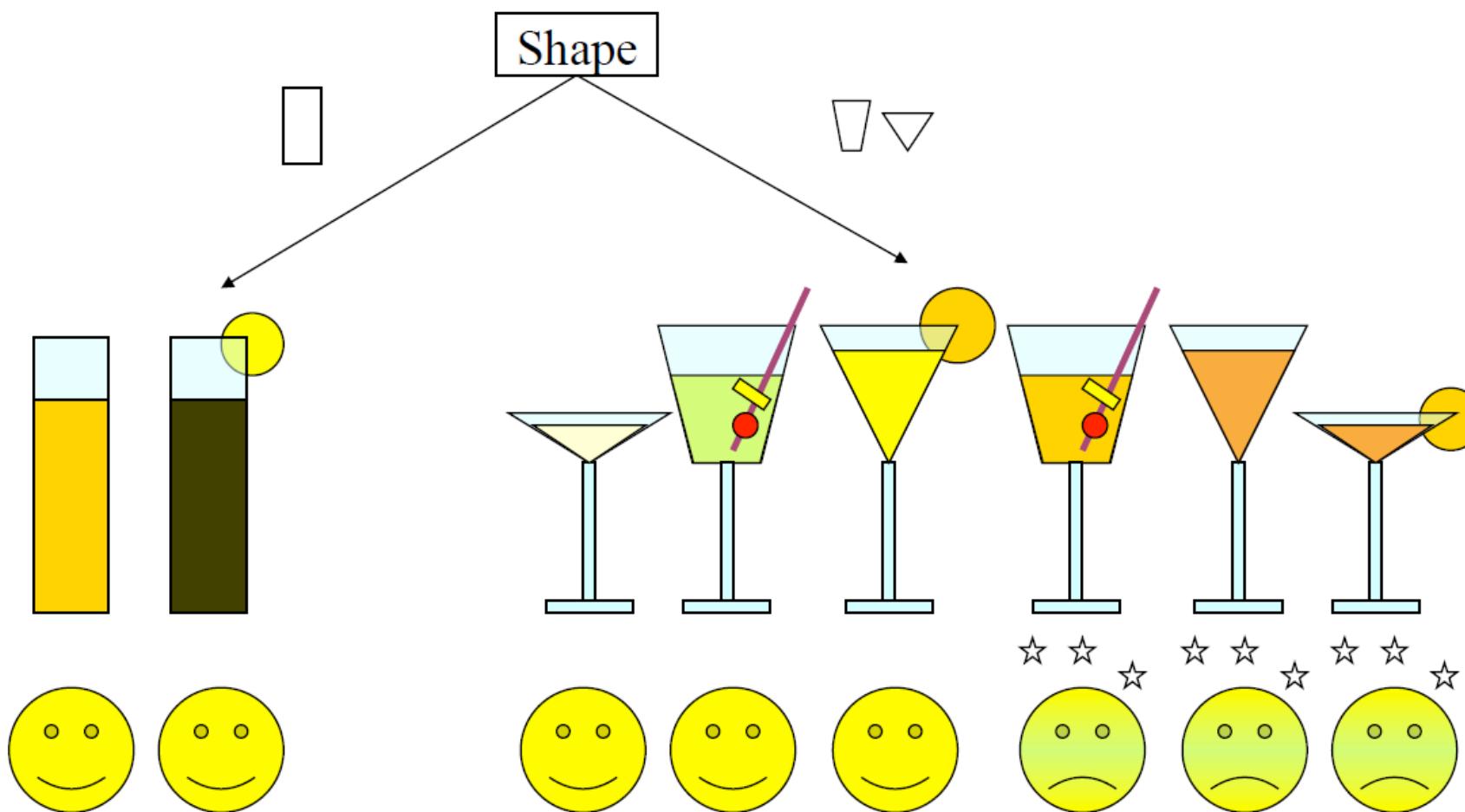
Ejemplo: ¿Este trago me sienta bien o no?

Se parte de 8 ejemplos (5 positivos, 3 negativos)



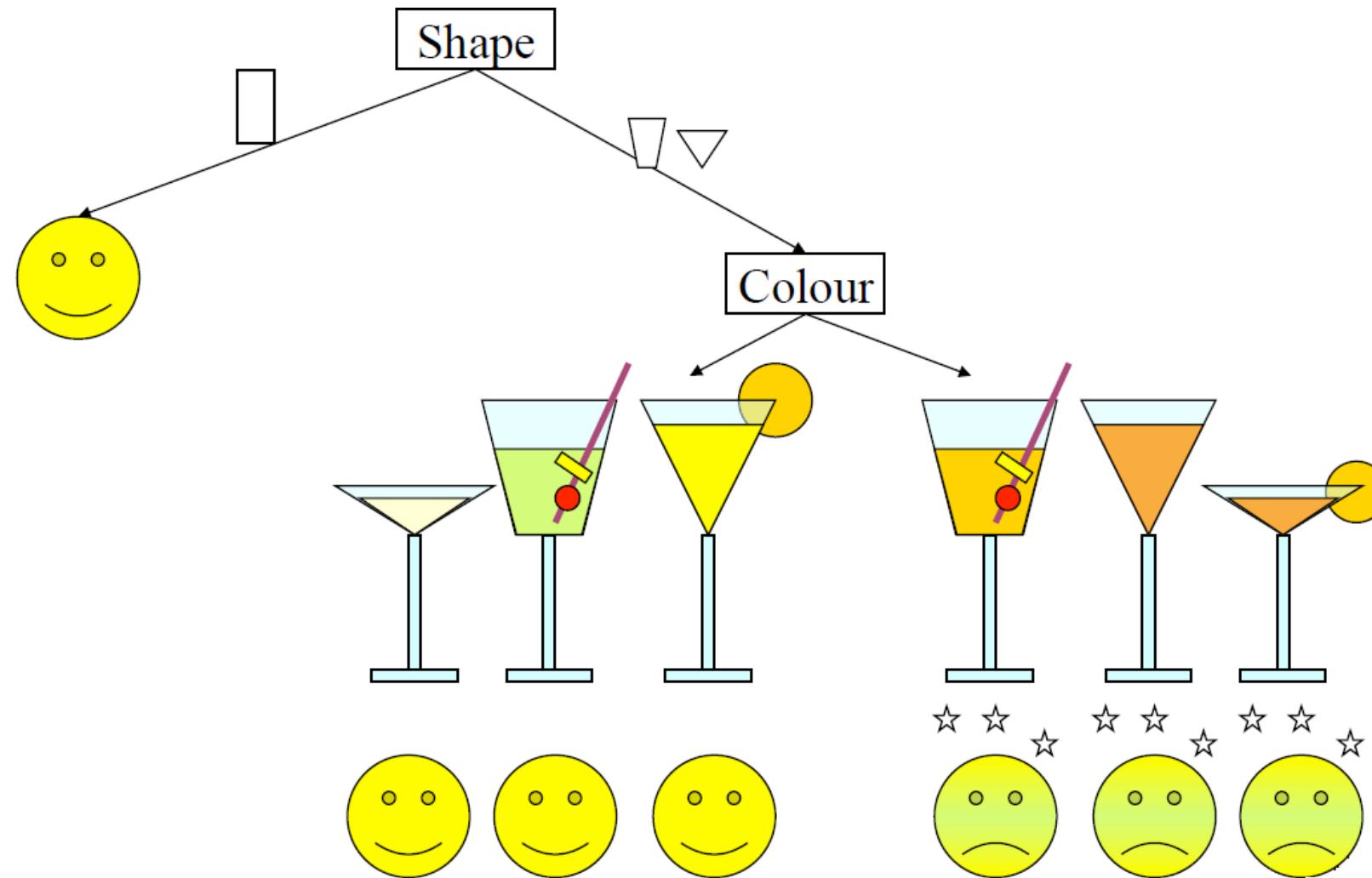
Ejemplo: ¿Este trago me  
sienta bien o no?

La forma es importante



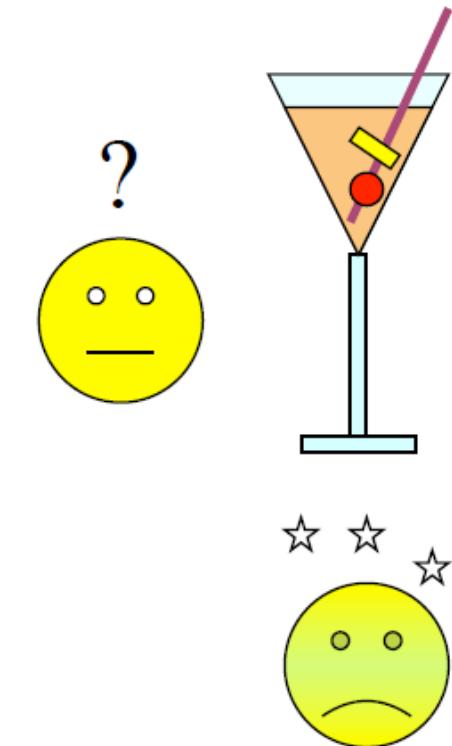
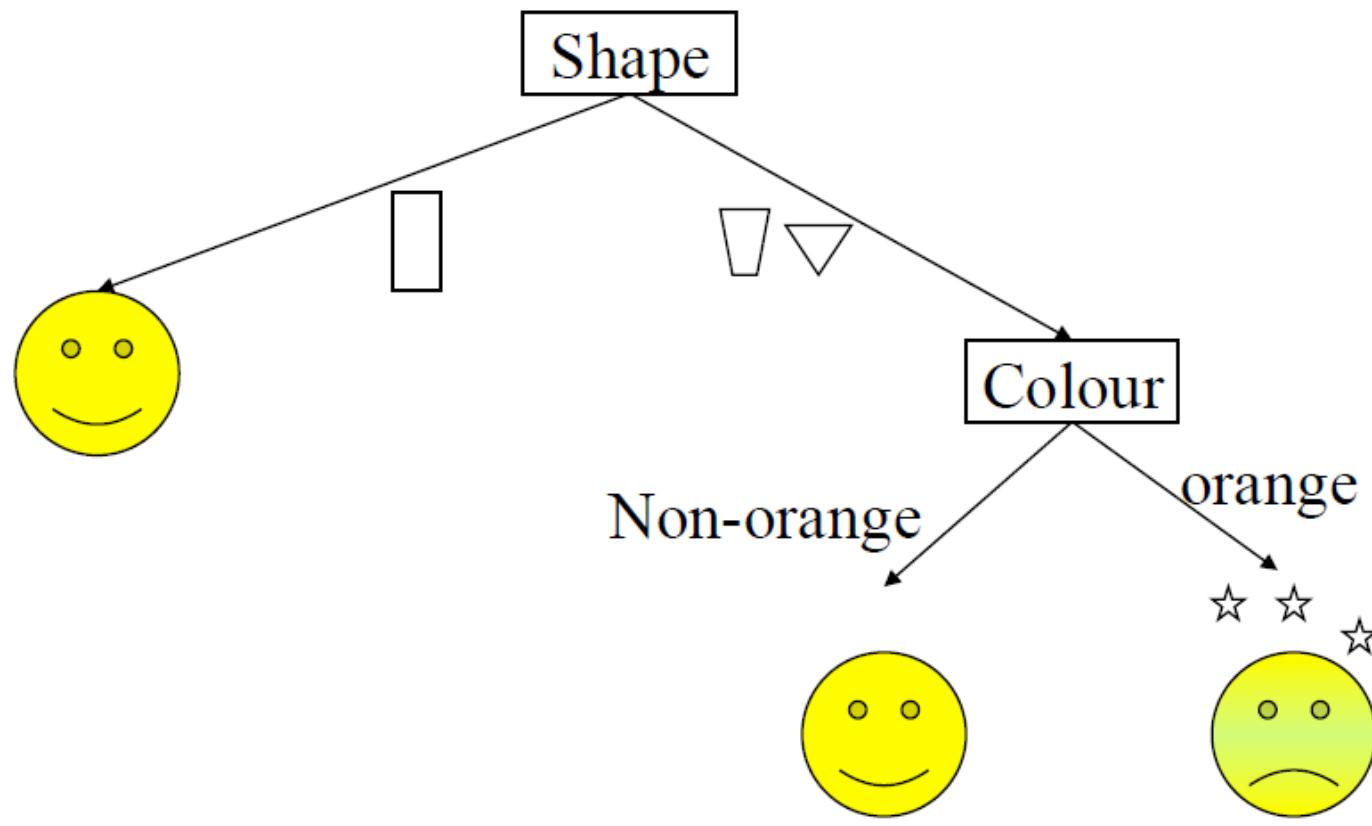
Ejemplo: ¿Este trago me  
sienta bien o no?

Para algunas copas, el color es importante



Ejemplo: ¿Este trago me sienta bien o no?

Ese trago no me sentará bien!



# El objetivo es encontrar los mejores tests (particiones) en cada nodo

Encontrar el mejor test por nodo (el que genere la hoja más pura)

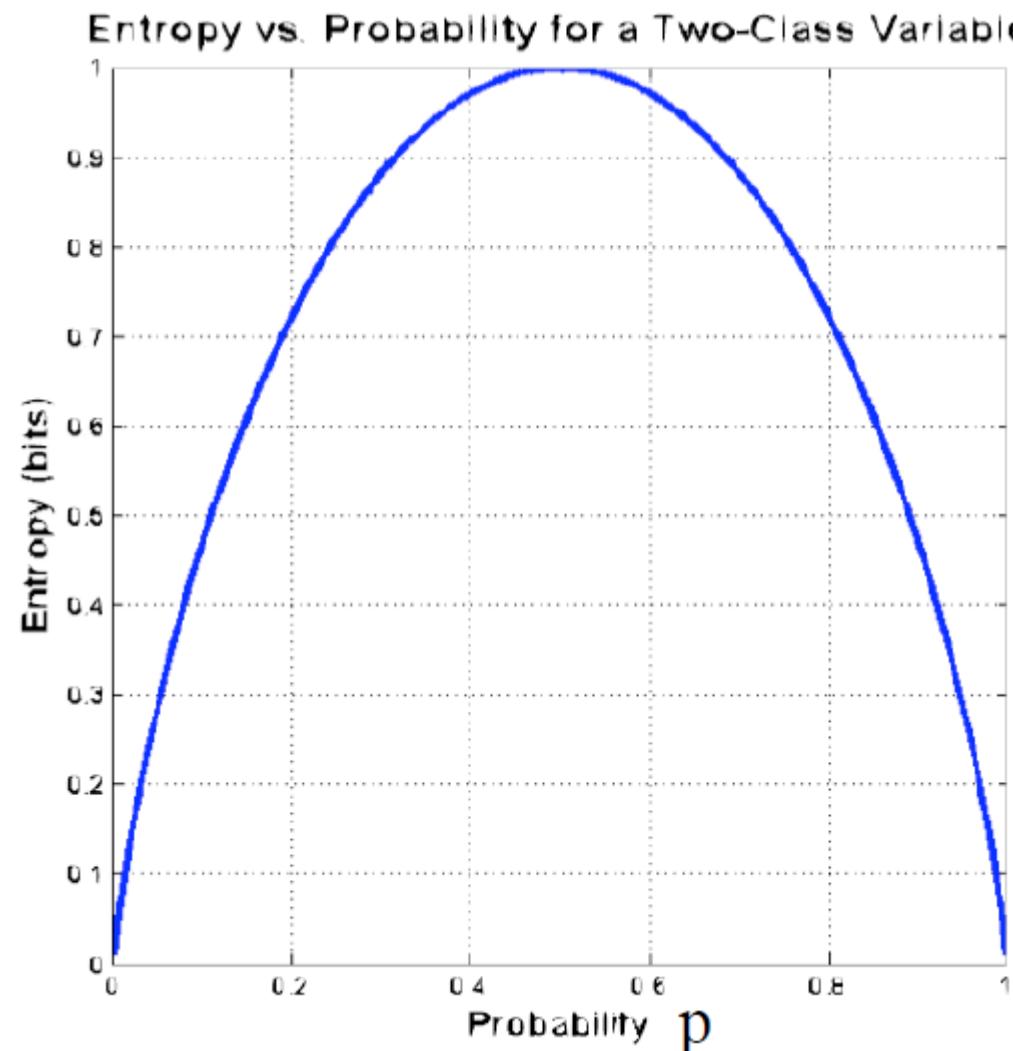
Medida común para medir pureza: **Entropía**

- La entropía mide “información perdida”
- El número de bits que se necesitan para representar información perdida

Dado un conjunto  $S = \{(x_i, y_i)\}_1^N, y_i \in \{0, 1\}$

- $Entropy(S) = -\sum p_i \log_2(p_i)$
- Donde  $p_i = \Pr(y = 1|x)$
- $Entropy(S) = 0 \leftrightarrow \exists i: p_i = 1$

La entropía para dos clases es menor cuando la hoja es pura:  $\Pr(y = 1|x) = 0$  o  $\Pr(y = 1|x) = 1$



# Una heurística para encontrar un test en un nodo: **Information Gain**

Seleccionar el test, que en promedio, provea la mayor información sobre la clase

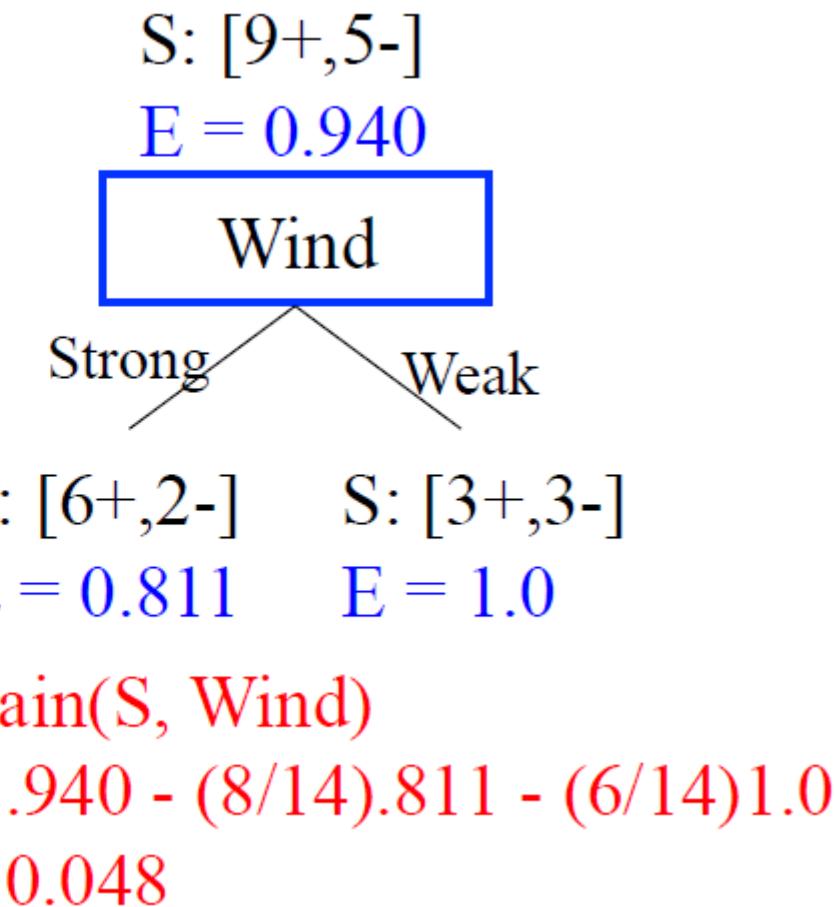
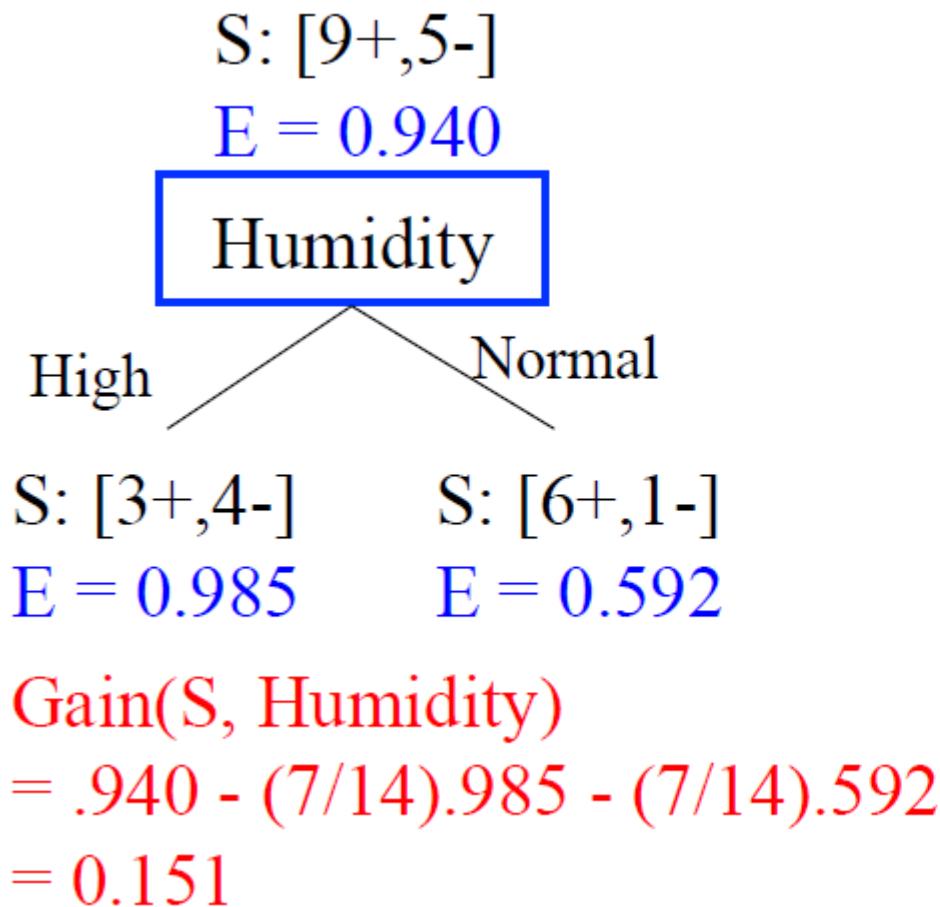
Este test, en promedio, reduce la entropía de la clase

Reducción de entropía esperada = **information gain**

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum \frac{|S_v| \text{Entropy}(S_v)}{|S|}$$

# Ejemplo: ¿Qué atributo usar para particionar el árbol?

En S existen 9+ y 5-. ¿Usar Humidity o Wind? Para particionar



# Espacio de hipótesis e inductive bias en TDIDT

## Espacio de hipótesis

- $H = \text{conjunto de todos los árboles posibles}$

## Inductive Bias

- Preference bias: algunas hipótesis en  $H$  se prefieren sobre otras
- Se prefieren árboles más cortos con atributos informativos en los niveles iniciales

**Occam's Razor:** se prefieren modelos más simples sobre complejos en base a una métrica de rendimiento

Principio similar en la ciencia

No hacer cosas más complicadas de lo necesario

Arboles más complejos producen menos generalización

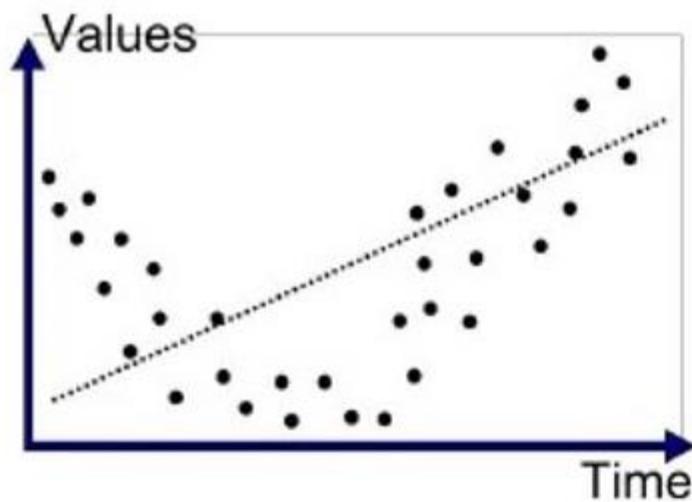
Mayor probabilidad de overfitting

# **Overfitting** es el fenómeno de crear modelos altamente complicados que no generalizan bien en nuevos datos

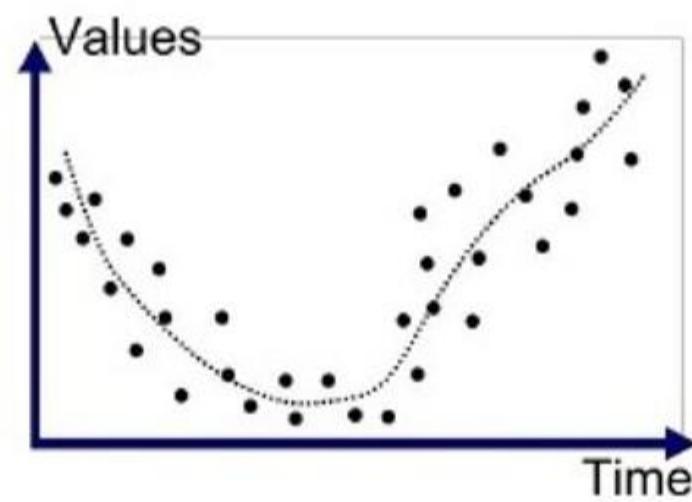
El overfitting se da cuando el modelo deja de aprender y solo memoriza los ejemplos provistos en el entrenamiento

- Normalmente, cuando el modelo se vuelve más complicado (más parámetros, mas capas en una ANN, más niveles en un árbol, etc.)
- Se incrementa el riesgo de modelar también el ruido
- El poder predictivo se reduce

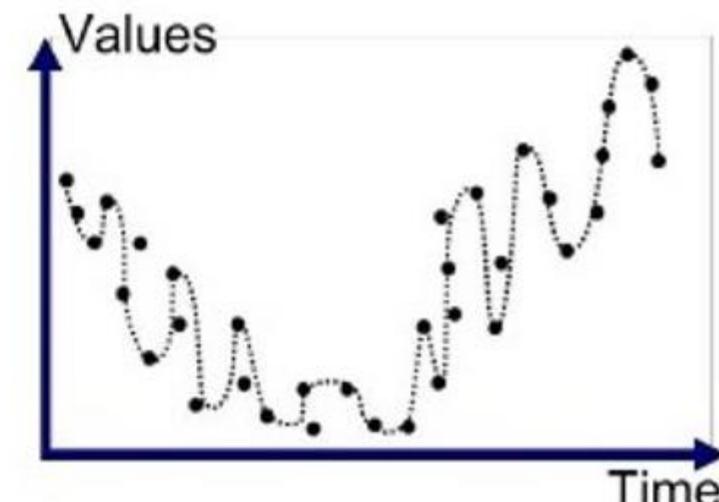
Así mismo, se podría tener **underfitting** si el modelo es muy simple



Underfitted

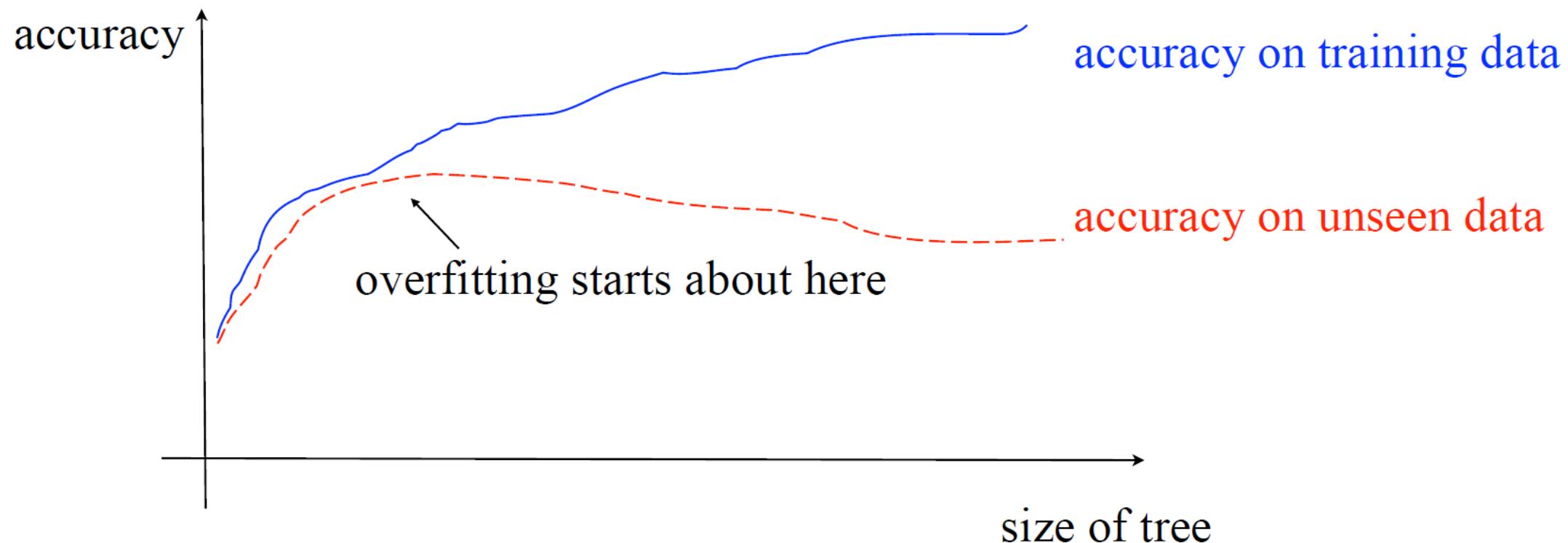


Good Fit/R robust



Overfitted

Este fenómeno se da cuando el rendimiento en los datos de entrenamiento sigue creciendo, pero no en los de prueba



En arboles de decisión, hay dos formas de evitar el overfitting

### Opción 1: Stopping Criterion

- Dejar de agregar más nodos cuando el overfitting comience a manifestarse

### Opción 2: Prunning

- No verificar overfitting cuando se cree el árbol
- Luego, comenzar a “podar” de reversa

El stopping criteria debe ser detectado mediante un set de validación o una prueba estadística

### Validation set

- Validation set = datos no vistos por el modelo para su creación
- Cuando el accuracy disminuye -> parar! (no crear más nodos)

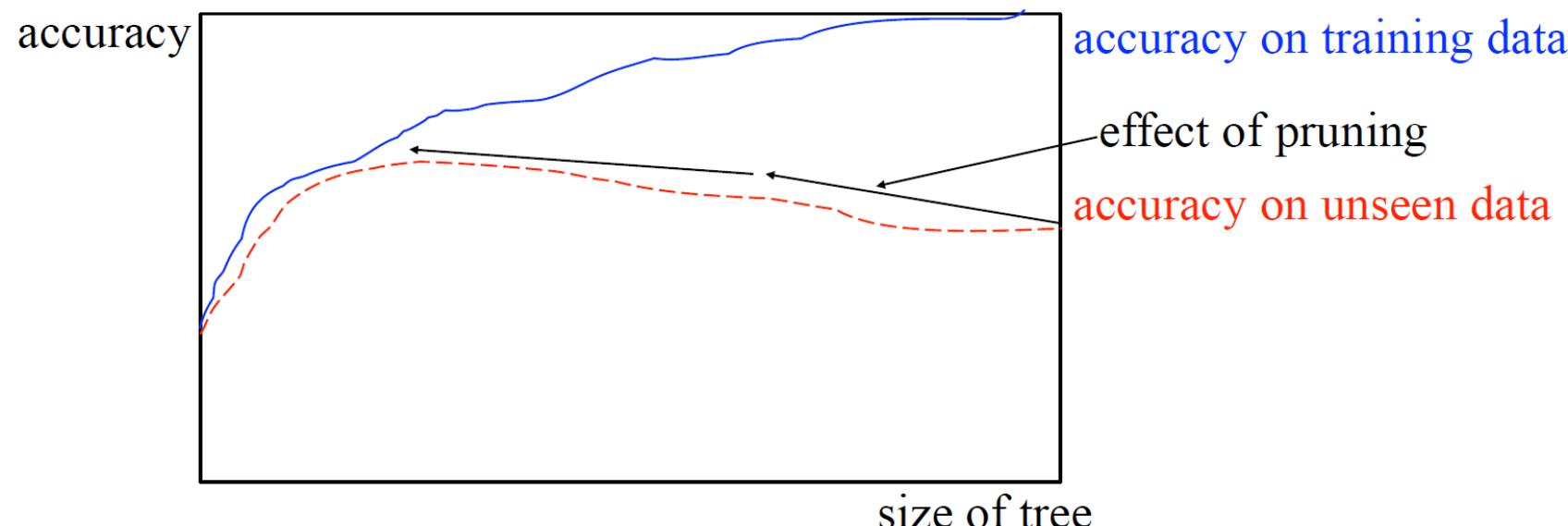
### Usar un test estadístico

- Test de significancia: hay un cambio en la distribución de la clase (chi-cuadrado)
- MDL: minimal description lenght principle. Minimizar  $\text{size}(\text{theory}) = \text{size}(\text{tree}) + \text{size}(\text{misclassifications(tree)})$

# Luego de aprender el árbol, comenzar a “podar” los nodos

Para todos los nodos del árbol

- Estimar el efecto de podar el árbol a ese nivel en términos de poder predictivo. E.j. calcular el accuracy en el test de validación
- Podar el nodo que provee el mejor incremento de rendimiento
- Continuar hasta que no existan más ganancias



Con “stopping criterium” el test puede resultar engañoso

- Es posible que el accuracy decrezca solo momentáneamente

Por esta razón, “pruning” es el método preferido

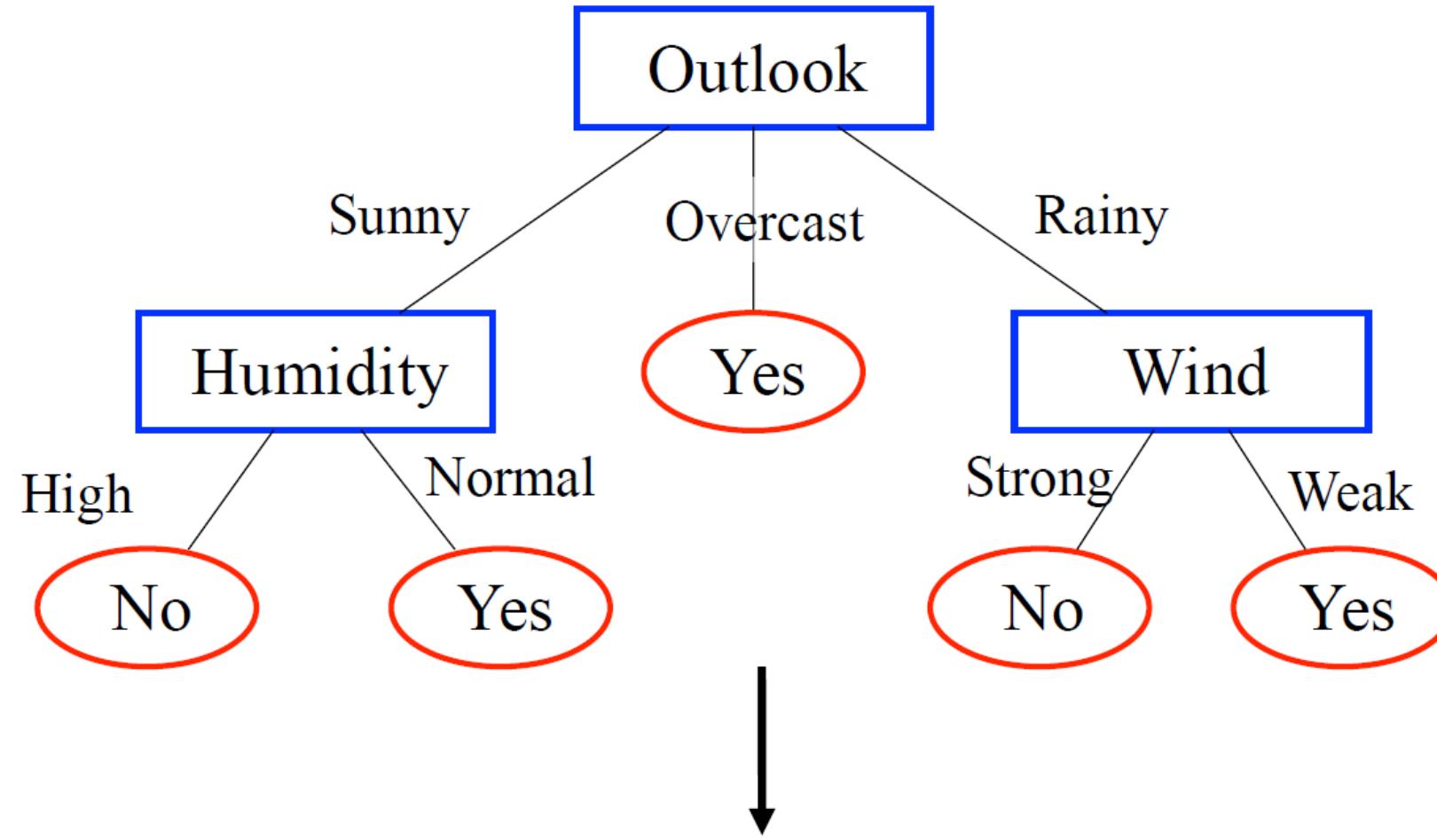
- Aunque sea más costoso computacionalmente

# Un árbol puede ser altamente interpretable con la ayuda de reglas

**Transformar un árbol a un conjunto de reglas**  
Desde la raíz hasta las hojas -> reglas if then

**Incrementa la comprensibilidad**  
Normalmente los modelos de ML son cajas negras

**Permite mayor flexibilidad al momento de podar**  
En una regla se puede eliminar una única condición sin tener que remover una rama



**if** Outlook = Sunny **and** Humidity = High **then** No

**if** Outlook = Sunny **and** Humidity = Normal **then** Yes

...

# Existen más heurísticas para seleccionar los tests en los arboles

## Atributos con valores continuos

- Se pueden crear pruebas binarias: e.j. Temp < 20
- Probar con valores razonables

## Atributos con muchos valores discretos

- Mayor ventaja sobre atributos con menos valores
- Para compensar, dividir la ganancia para “máximo potencial de ganancia” (SI)
- Gain ration  $GR(S, A) = Gain(S, A)/SI(S, A)$
- $SI(S, A) = - \sum \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$
- Donde  $i$  son todos los posibles valores del atributo  $A$

# ¿Qué heurísticas usan algoritmos populares para creación de arboles?

## ID3

- Information gain o gain ratio

## CART

- Gini criterium

TDIDT es una familia de algoritmos  
que cubren CART, ID3, C4.5

```
function TDIDT( $E$ : set of examples) returns tree;  
     $T' := \text{grow\_tree}(E);$   
     $T := \text{prune}(T');$   
    return  $T$ ;
```

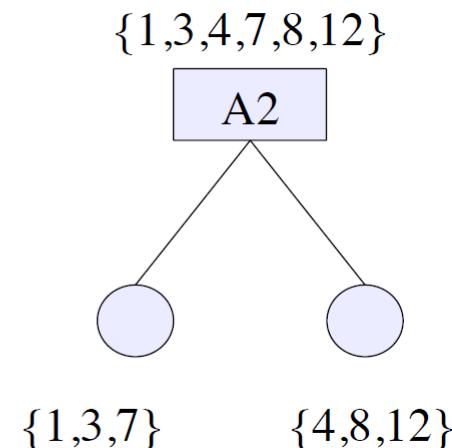
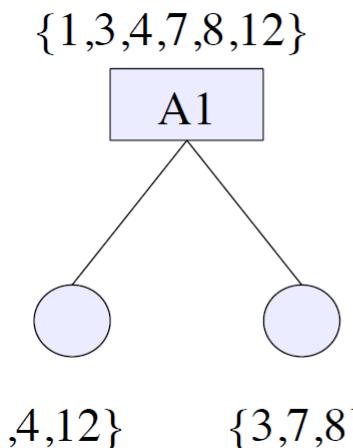
```
function grow_tree( $E$ : set of examples) returns tree;  
     $T := \text{generate\_tests}(E);$   
     $t := \text{best\_test}(T, E);$   
     $P := \text{partition induced on } E \text{ by } t;$   
    if stop_criterion( $E, P$ )  
    then return leaf(info( $E$ ))  
    else  
        for all  $E_j$  in  $P$ :  $t_j := \text{grow\_tree}(E_j);$   
        return node( $t, \{(j, t_j)\}$ );
```

Para calcular regresiones, el árbol crea subsets con varianza mínima

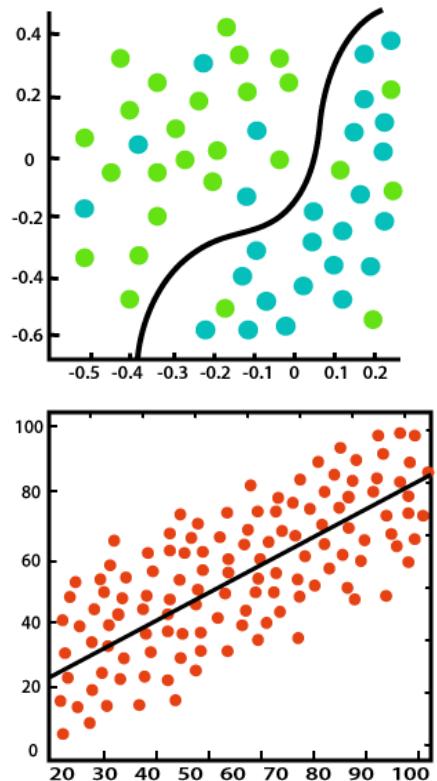
Intentar generar subsets con varianza mínima para la variable objetivo  
(en promedio)

$$Var(S) = \frac{\sum_{x_i \in S} (x_i - \bar{x})^2}{|S| - 1}$$

$$\bar{x} = \frac{\sum_{x_i \in S} x_i}{|S|}$$



# Content



Arboles de decisión

Como se crean, conceptos

Inducción de conjuntos de reglas

Generalización a través de reglas

Support Vector Machines

Kernels, hyperparámetros

Artificial Neural Networks

Nuevas arquitecturas y aplicaciones

# Otra forma de representar teorías es usando **Reglas de Decisión**

Anteriormente revisado: Arboles de decisión

Puede representar conceptos conjuntivos

Otra forma popular es reglas if-then

IF <condición> THEN “pertenece al concepto”

¿Como se pueden aprender esas reglas?

Usando árboles, algoritmos genéticos, algoritmos especiales

# Un método popular para generar reglas es el **Sequential Covering**

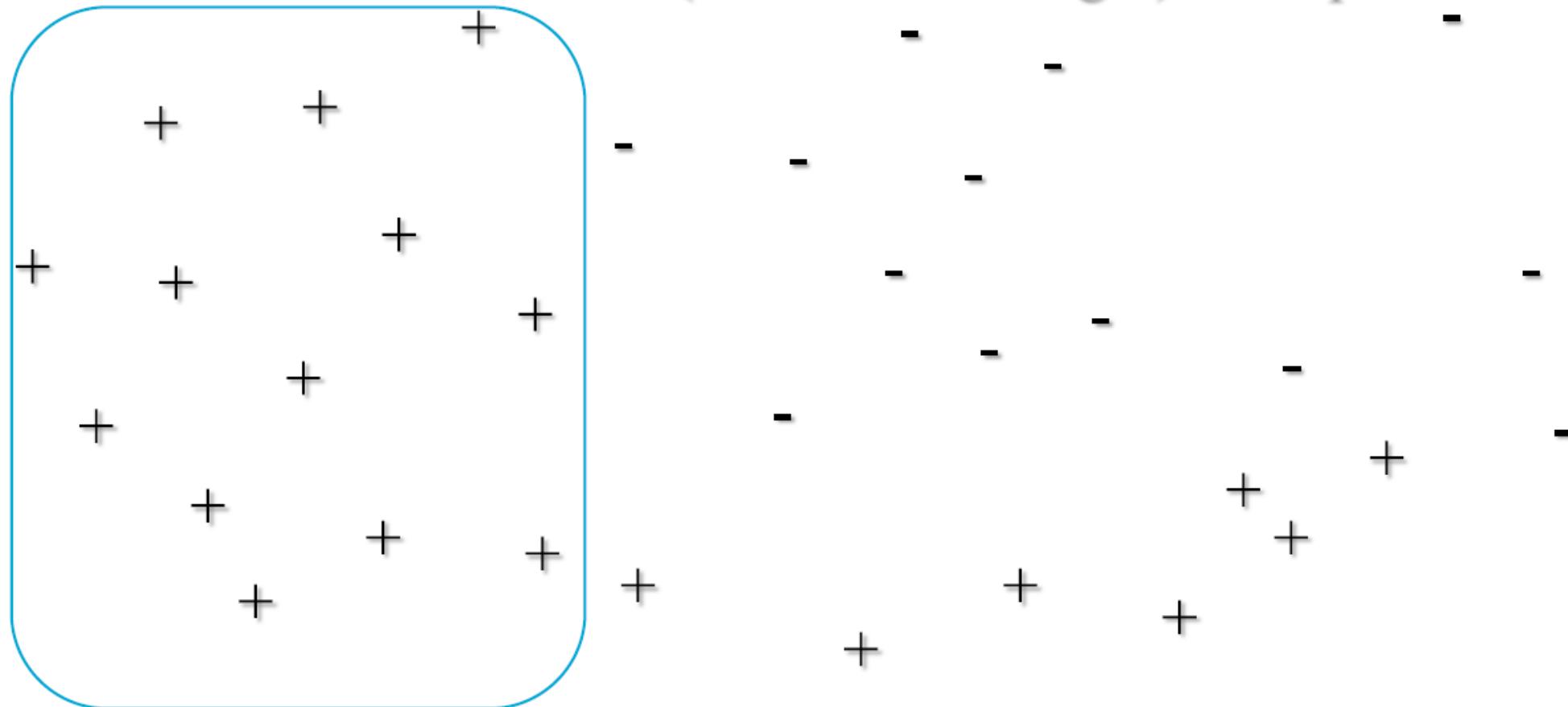
También denominado “separate and conquer”

- Analogía con árboles -> “divide and conquer”

Principio general: aprender un conjunto de reglas, una a la vez

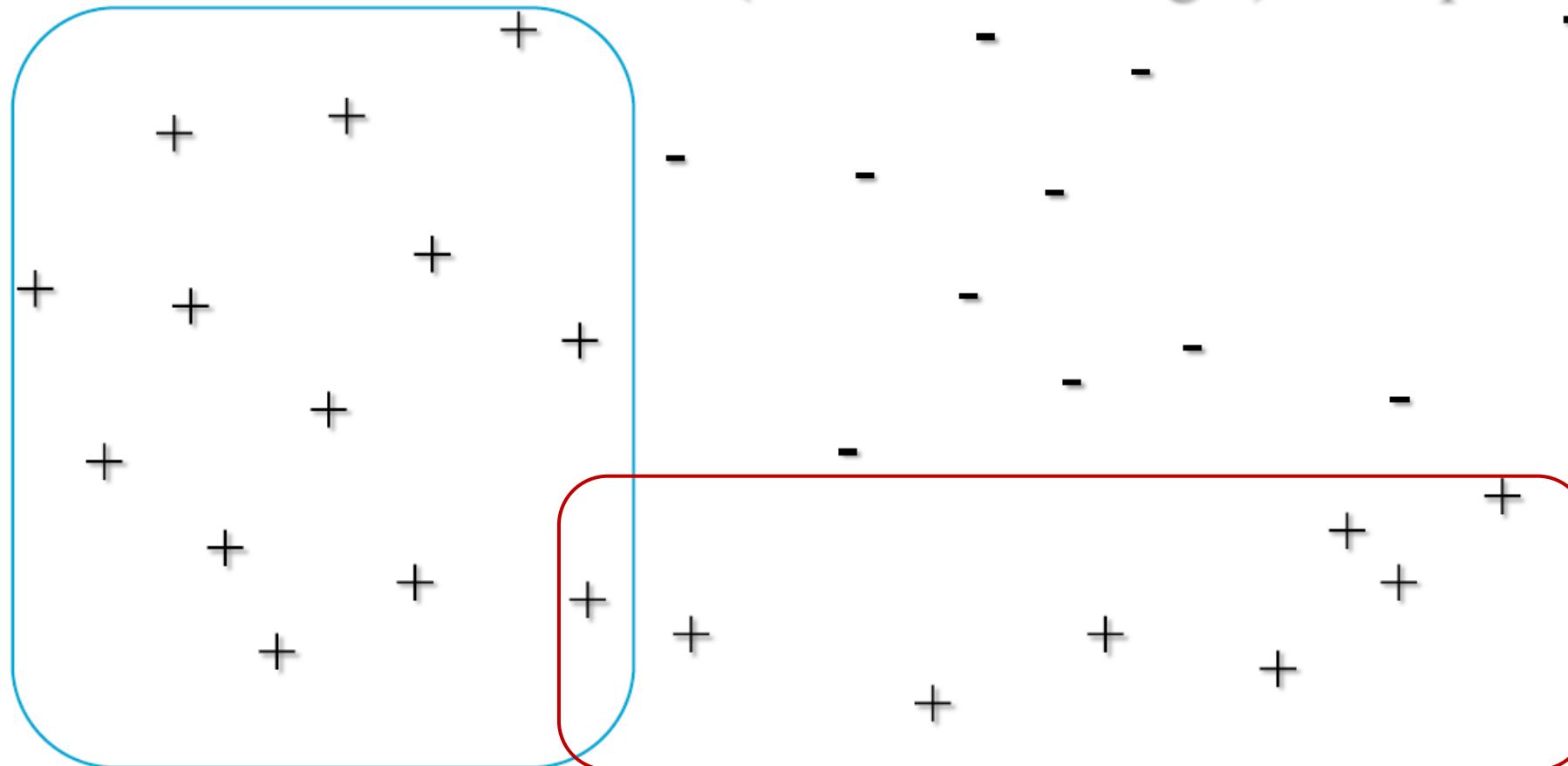
- Aprender una regla con alta precisión
- Puede hacer predicciones sobre algunos ejemplos
- Una vez que los ejemplos fueron cubiertos, centrarse en los otros
- Repetir el proceso hasta que se hayan cubierto todos los ejemplos

**if a & b & c & ... (=in blue rectangle) then positive**



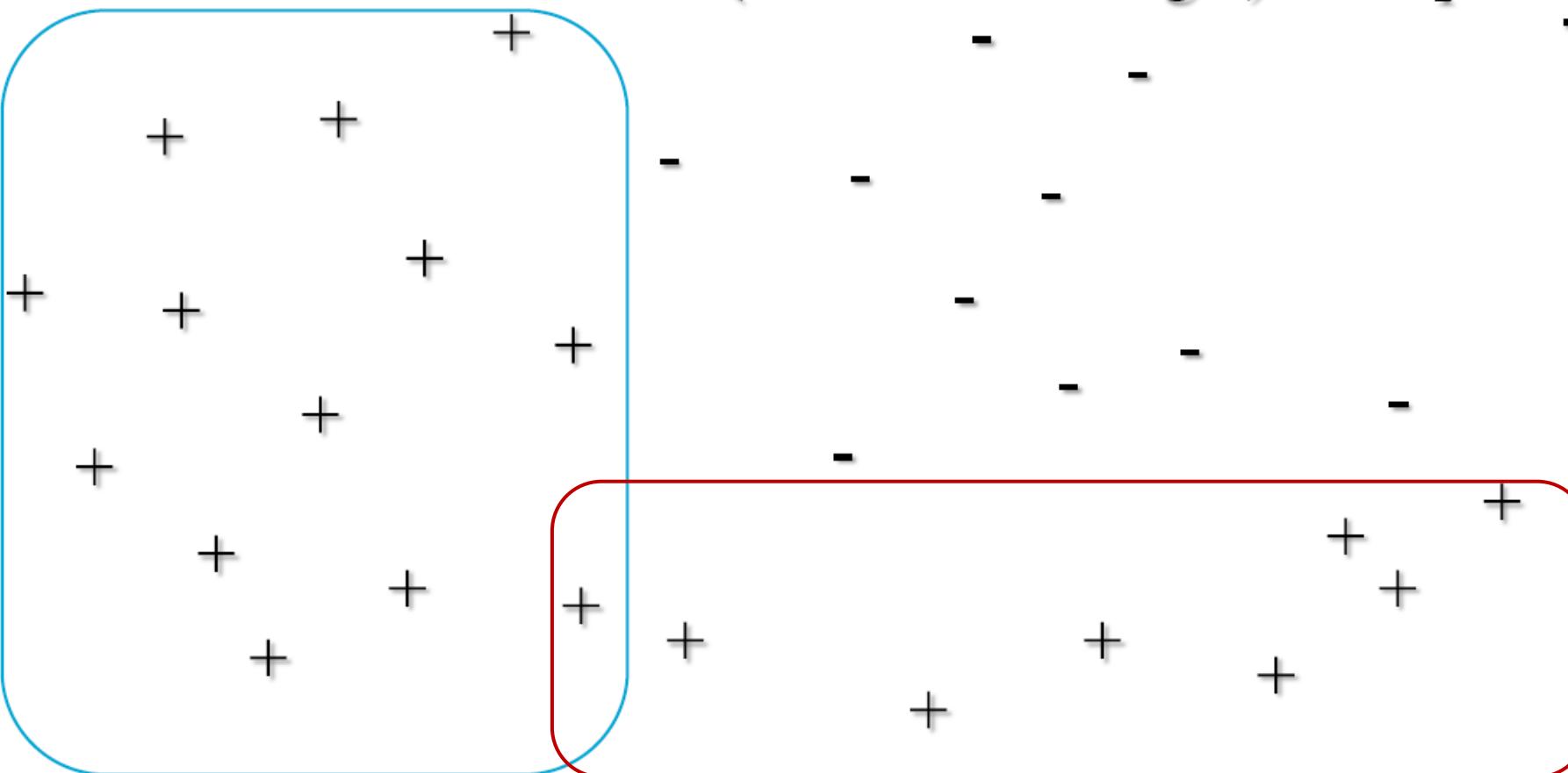
Estos ejemplos pueden  
ser olvidados ahora

**if a & b & c & ... (=in blue rectangle) then positive**



Se adjuntan los nuevos  
ejemplos

**if a & b & c & ... (=in blue rectangle) then positive**



Todos los ejemplos fueron  
incluidos: Terminar

# Algoritmo general para aprender conjuntos de reglas

```
function LearnRuleSet(Target, Attrs, Examples, Threshold):
    LearnedRules :=  $\emptyset$ 
    Rule := LearnOneRule(Target, Attrs, Examples)
    while performance(Rule, Examples) > Threshold, do
        LearnedRules := LearnedRules  $\cup$  {Rule}
        Examples := Examples \ {examples classified correctly by Rule}
        Rule := LearnOneRule(Target, Attrs, Examples)
    sort LearnedRules according to performance
    return LearnedRules
```

# Aprender una regla individualmente

Se utiliza greedy search

Top-down

- Comenzar con la regla más general (máxima cobertura, pero baja presición)
- Agregue literales uno a uno
- Gradualmente maximice la precisión sin sacrificar cobertura

Botton-up

- Comenzar con una regla muy específica (cobertura mínima pero máxima presición)
- Remover literales uno a uno
- Gradualmente maximizar la cobertura sin sacrificar la presición

```

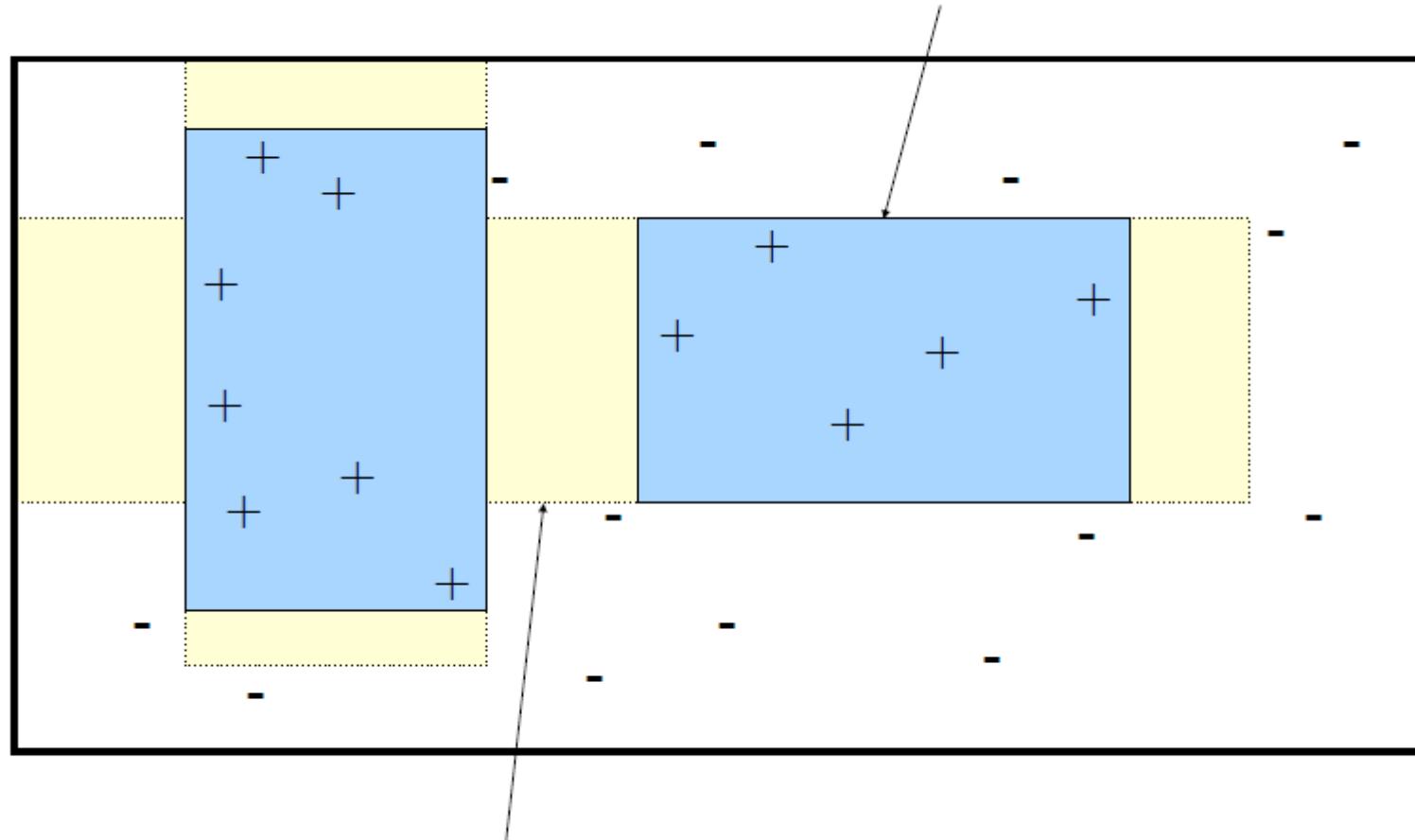
function LearnOneRule(Target, Attrs, Examples):
    NewRule := “IF true THEN pos”
    NewRuleNeg := Neg
    while NewRuleNeg not empty, do
        // add a new literal to the rule
        Candidates := generate candidate literals
        BestLit := argmaxL ∈ Candidates performance(Specialise(NewRule,L))
        NewRule := Specialise(NewRule, BestLit)
        NewRuleNeg := {x ∈ Neg | x covered by NewRule}
    return NewRule

function Specialise(Rule, Lit):
    let Rule = “IF conditions THEN pos”
    return “IF conditions and Lit THEN pos”

```

# Bottom-up vs. Top-down

Bottom-up: typically more specific rules



Top-down: typically more general rules

# La determinación de la calidad de una regla se realiza mediante heurísticas

Como se determina si una regla es “buena”

- Alta precisión y covertura

Funciones de evaluación

- Precisión:  $p/(p + n)$
- Entropía: más simetría entre  $p$  y  $n$

Post-pruning

- Misma idea que en árboles de decisión

# Algoritmo AQ (Michalski et al.): top-down

Para una clase C (hasta que se cubran todos los ejemplos)

- Seleccionar un ejemplo  $e$
- Considerar  $H_e = \{reglas\ que\ cubren\ el\ ejemplo\}$
- Buscar top-down en  $H_e$  para encontrar la mejor regla

Es una búsqueda eficiente:  $H_e$  es mucho más pequeña q  $H$

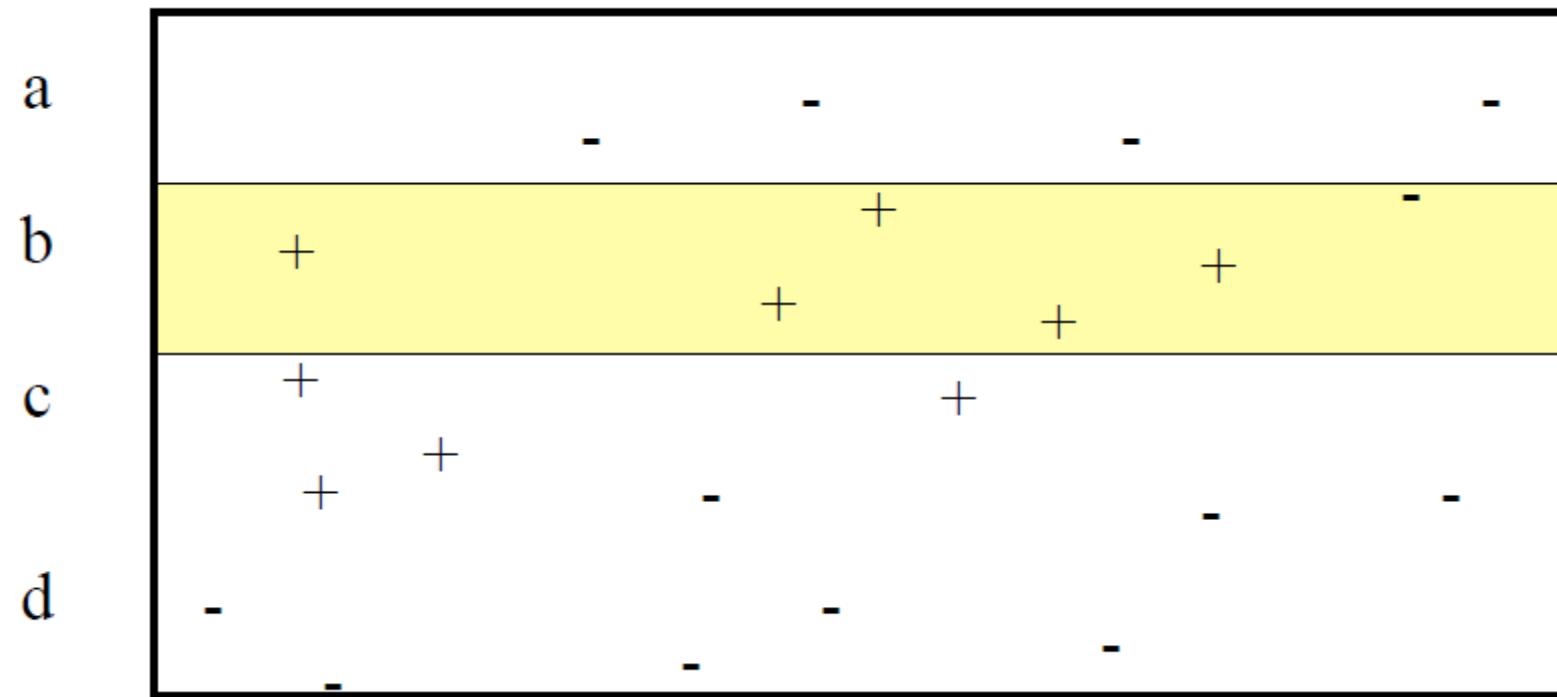
No es robusto con respecto al ruido

Value  
of A:

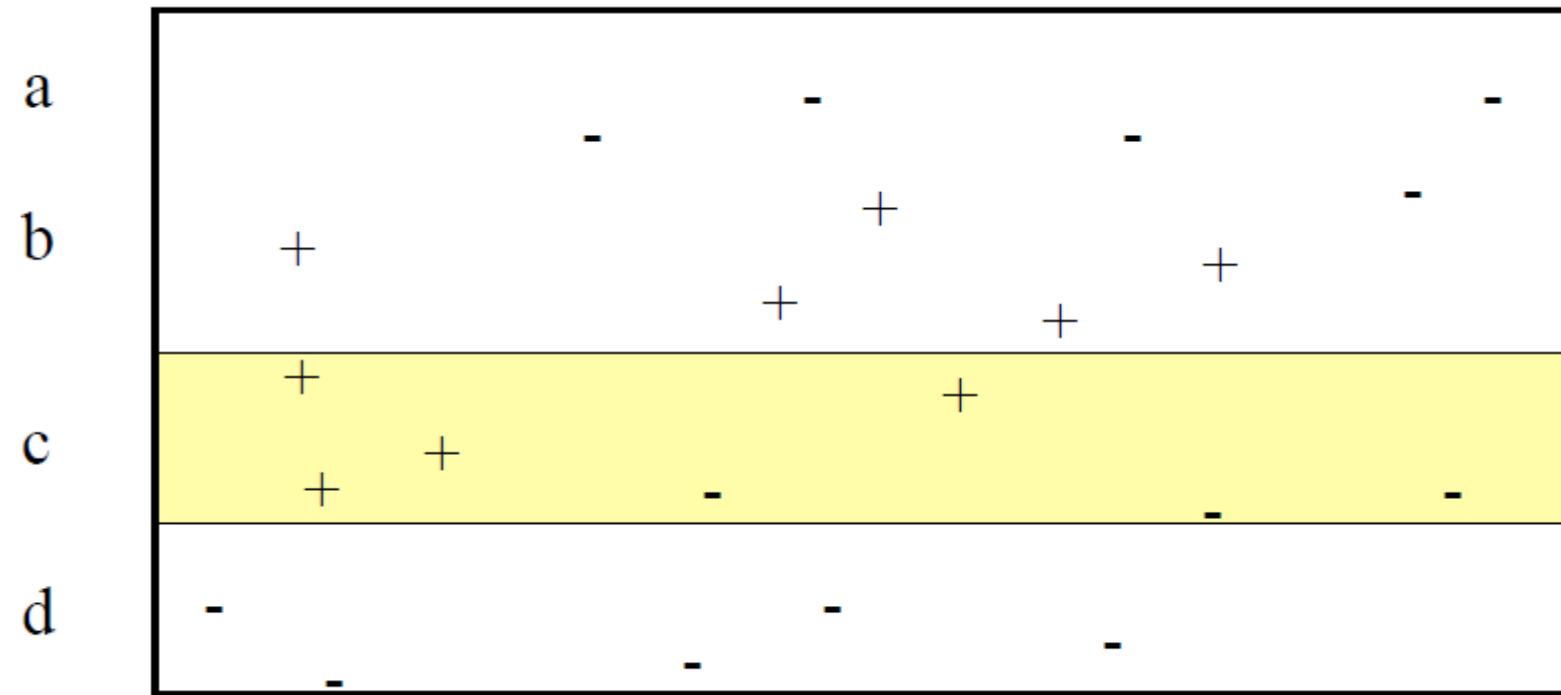
a	-	-	-	-	-
b	+		+	+	+
c	+		+	+	
d	-	+	-	-	-

If A=a then pos

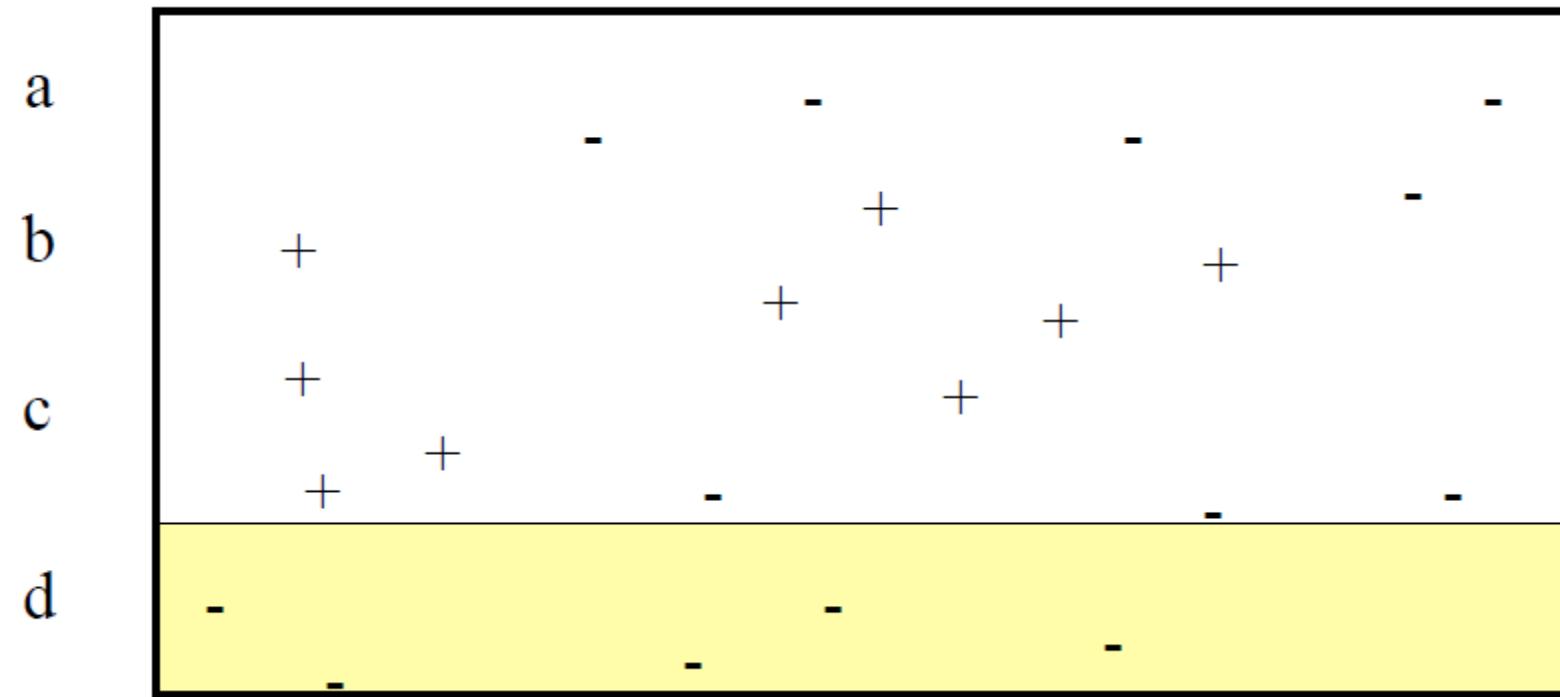
Looking for a good rule in the format “IF A=... THEN pos”



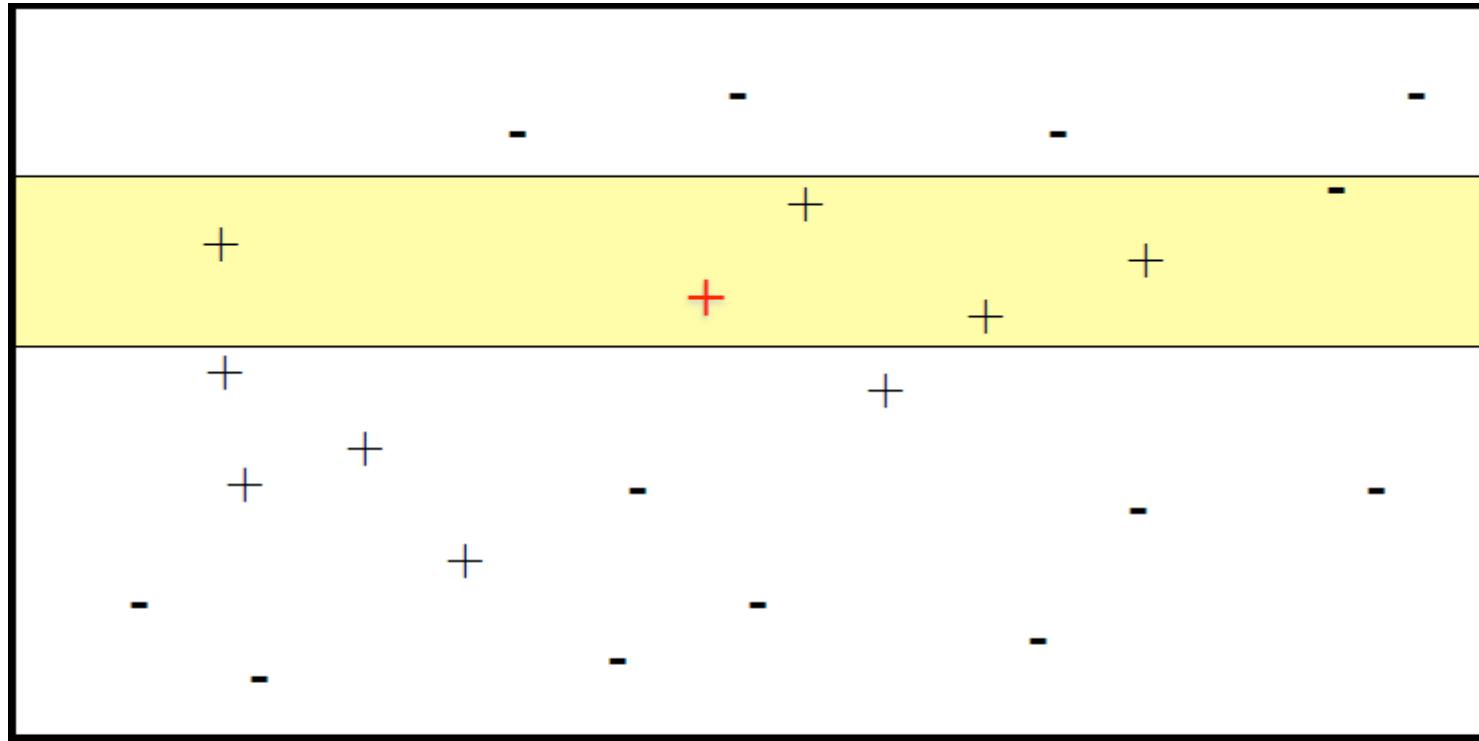
If  $A=b$  then pos



If A=c then pos



If  $A=d$  then pos



If  $A=b$  then pos

Try only rules that cover the seed “ $\textcolor{red}{+}$ ” which has  $A=b$ .  
Hence,  $A=b$  is a reasonable test,  $A=a$  is not.  
*We do not try all 4 alternatives in this case!* Just one.

Las **reglas de asociación** describen relaciones entre conjuntos de atributos boléanos

Muy utilizadas en “basket analysis”, e.j. que productos se compran juntos

Client	cheese	bread	butter	wine	jam	ham
1	yes	yes	yes	yes	no	yes
2	yes	no	yes	no	no	no
3	no	yes	yes	no	no	yes
...	...	...	...	...	...	...

IF bread & butter THEN cheese  
confidence: 50%  
support: 5%

Las reglas de asociación se caracterizan  
por *support* y *confidence*

Support: % de clientes que compraron  $a_1, \dots, a_{n+m}$

Confidence: % de compradores de  $a_1, \dots, a_n$  que también compraron  
 $a_{n+1}, \dots, a_{n+m}$

# El algoritmo para buscar reglas de asociación debe ser eficiente

## Algoritmo APRIORI (Agrawal et al., 1993)

- Parámetros: min support, min confidence
- Encuentra todas las asociaciones basadas en los parámetros
- Disponible en Weka

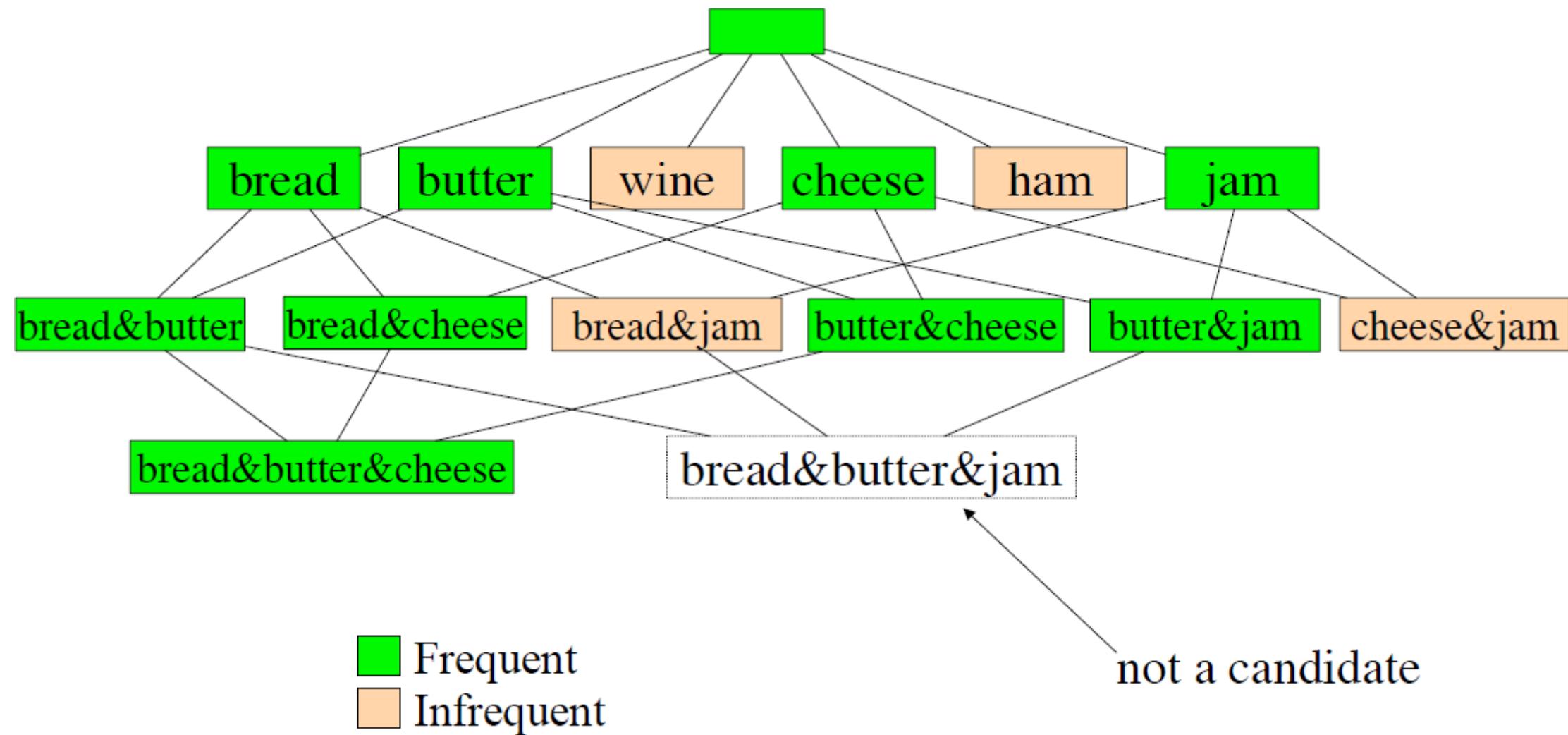
# Observaciones importantes

- Sea  $freq(S)$  = numero de ejemplos conteniendo  $S$
- Considerar IF  $a_1 \dots a_n$  THEN  $a_{n+1} \dots a_{n+m}$
- $Support = freq(\{a_1, \dots, a_{n+m}\})/freq(\{\})$
- $Confidence = freq(\{a_1, \dots, a_{n+m}\})/freq(\{a_1 \dots a_n\})$
- Las asociaciones con suficiente *confidence* y *support* pueden ser derivadas de la lista de “conjuntos frecuentes”
- $S$  es frecuente iff  $freq(S) > \text{min\_support} * freq(\{\})$

# Items frecuentes se encuentran utilizando breadth first

## Paso 1: encontrar todos los conjuntos frecuentes

- Si  $a_1 \dots a_j$  no es frecuente, tampoco lo es  $a_1 \dots a_{j+1}$
- Encontrar todos los ítems frecuentes de cardinalidad 1
- Encontrar todos los ítems frecuentes de cardinalidad 2
- $\{a_1, a_2\}$  es frecuente solo si  $\{a_1\}$  y  $\{a_2\}$  lo son
- Encontrar todos los ítems frecuentes de cardinalidad 3
- ...



# Encontrando conjuntos frecuentes

```
min_freq := min_support * freq( $\emptyset$ );
d := 0;
Q0 = { $\emptyset$  }; /*  $Q_i$  = candidates for level i */
F :=  $\emptyset$  ; /* F = frequent sets */
while Qd  $\neq \emptyset$  do
    for all S in Qd: find freq(S); /* data access */
    delete those S in Qd with freq(S) < min_freq;
    F := F  $\cup$  Qd;
    compute Qd+1;
    d := d+1
return F
```

# Computación offline

compute  $Q_{d+1}$  from  $Q_d$  and  $F$ :

$Q_{d+1} := \emptyset$ ;

**for each**  $S$  in  $Q_d$ :

**for each** item  $x$  not in  $S$ :

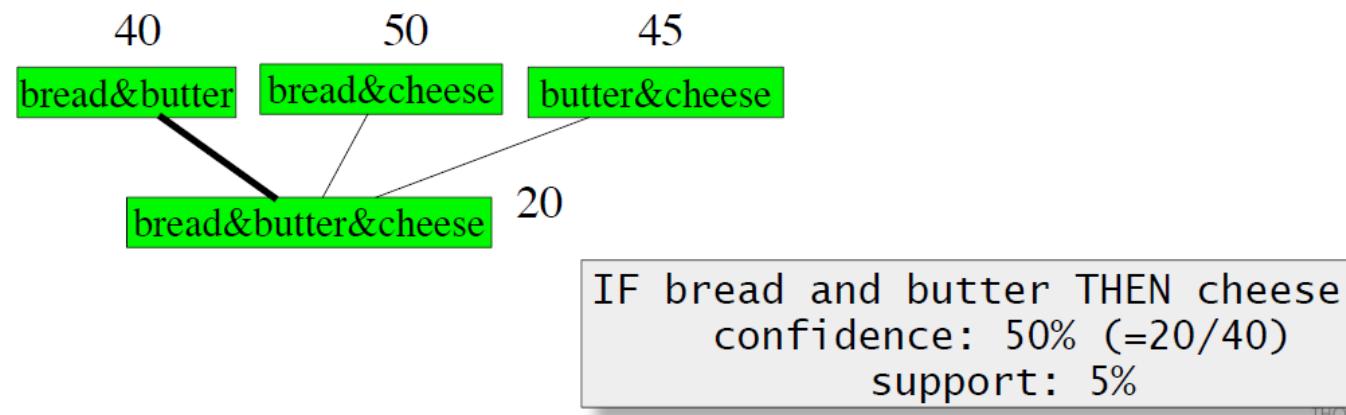
$S' := S \cup \{x\}$ ;

**if** each subset of  $S'$  obtained by removing 1 element of  $S'$  is in  $F$

**then** add  $S'$  to  $Q_{d+1}$ ;

## Paso 1: inferir reglas de asociación con conjuntos frecuentes

- Si  $S \cup \{a\}$  in  $F$  and  $freq(S \cup \{a\}) > \text{min\_confidence}$
- Retornar la regla IF  $S$  THEN  $a$



Las reglas encontradas deben ser  
post-procesadas para tener un orden

A veces se encuentran muchas reglas  
¿Cómo procesarlas?

Ordenar bajo algún criterio  
Support, confidence, significancia estadística

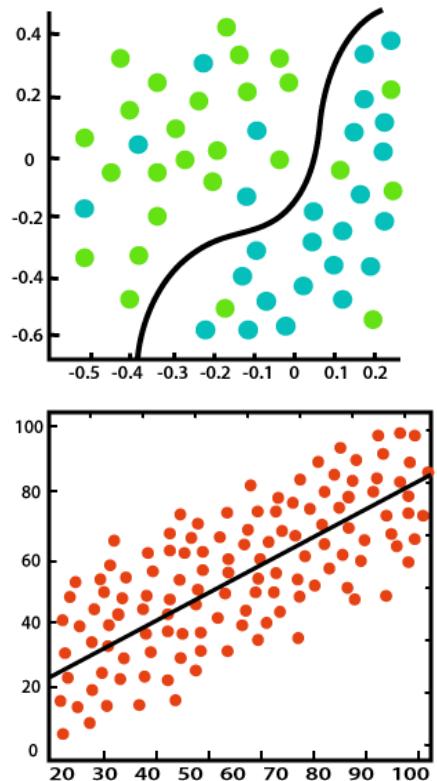
Otros métodos para post-procesado  
Lenguaje de búsqueda

# Investigar

De que se trata el algoritmo GABIL

- Idea básica de un algoritmo genético
- Como se formatean las entradas

# Content



Arboles de decisión

Como se crean, conceptos

Inducción de conjuntos de reglas

Generalización a través de reglas

Support Vector Machines

Kernels, hyperparámetros

Artificial Neural Networks

Nuevas arquitecturas y aplicaciones

**Instance Based Learning** almacena ejemplos para compararlos cuando se presenta uno nuevo

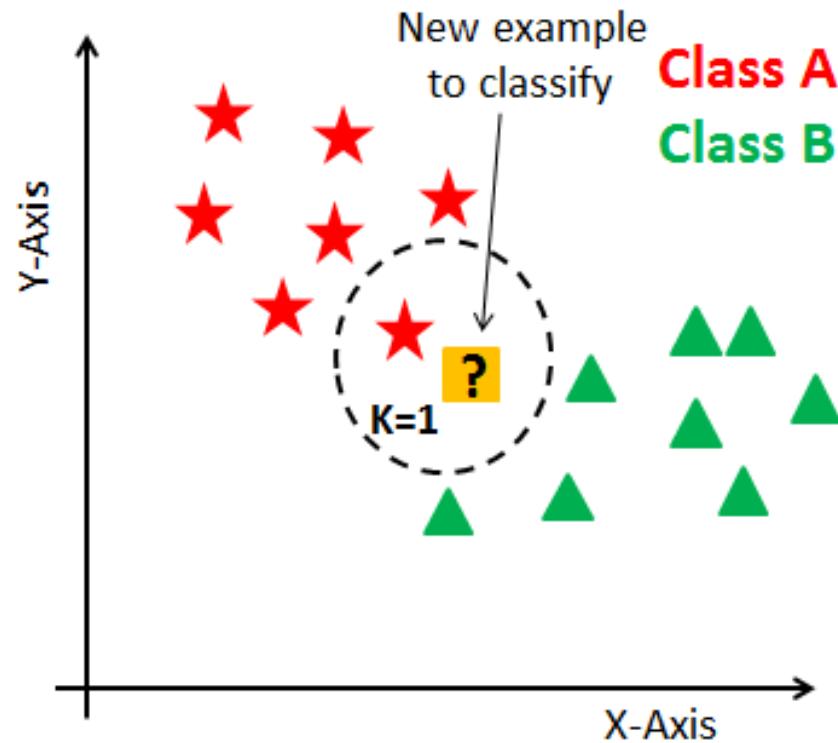
### Idea básica

Buscar por ejemplos similares cuando se presente uno nuevo

### Predictión

Se basa en los ejemplos similares encontrados

**k-nearest-neighbor** es un ejemplo de este tipo de aprendizaje

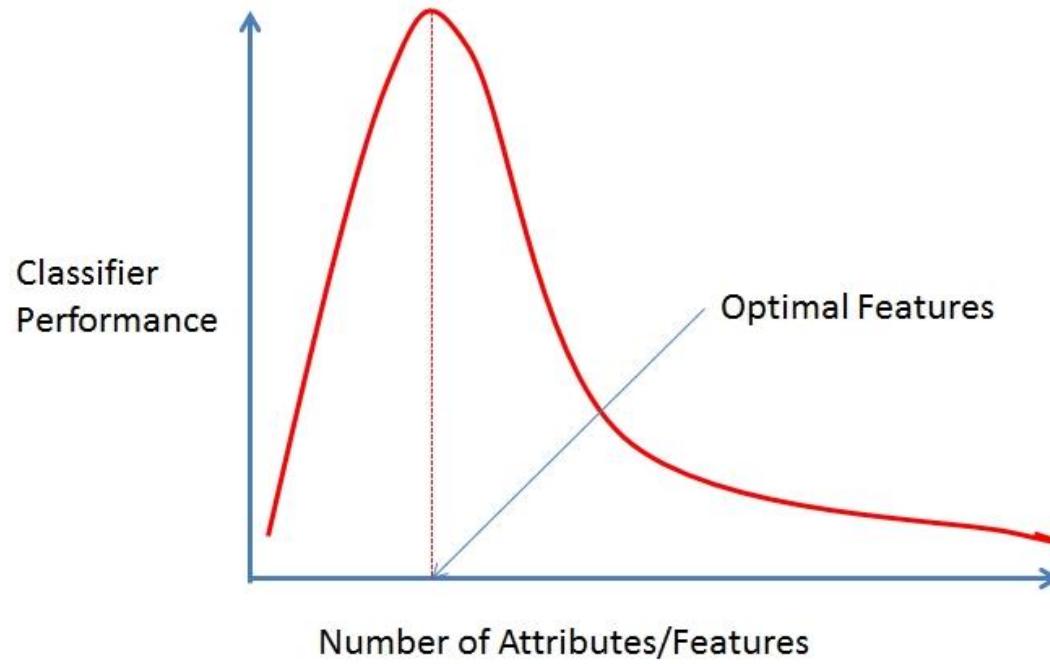


Encuentra las  $k$  instancias más similares (distancia/similaridad) al nuevo ejemplo

Usa la clase más frecuente para clasificación

Usa el valor promedio para regresión

# The curse of dimensionality



Un espacio de dimensiones grande (muchos atributos) tiene un efecto negativo en el aprendizaje

Especialmente negativo para IBL

Si se tienen 20 atributos y solo 2 son representativos, 18 atributos dominarán el modelo

Arboles de decisión/reglas son menos propensos a estos efectos negativos (¿por que?)

Una de las formas de lidiar con la alta dimensionalidad es **re-escalar los atributos** de los ejemplos

$$X = \{(x_i, y_i)\}_1^N, x_i \in R^D$$

$$x^{(1)} = [0, 1] \qquad \qquad x^{(2)} = [-10, 10]$$

La distancia en el primer atributo siempre será menor

Una de las formas de lidiar con la alta dimensionalidad es **re-escalar los atributos** de los ejemplos

$$X = \{(x_i, y_i)\}_1^N, x_i \in R^D$$

$$x^{(1)} = [0, 1] \qquad \qquad \qquad x^{(2)} = [-10, 10]$$

Eso hace que el segundo atributo sea el más dominante para el modelo

Possible solución: **re-escalar**

Una de las formas de lidiar con la alta dimensionalidad es **re-escalar los atributos** de los ejemplos

$$X = \{(x_i, y_i)\}_1^N, x_i \in R^D$$

$$x^{(1)} = [0, 1]$$

$$x'^{(2)} = [0, 1]$$

De hecho:  $\forall j \in D \ x^{(j)} = [0, 1]$

KNN le da la misma importancia a instancias independientemente de si están cerca o no del ejemplo

### Asignar más importancia a instancias cercanas

- Se pueden usar todos los ejemplos, no solo K

$$\hat{f}(x_q) = \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}, f(x_i) \in \{0,1\}, \text{ con } w_i = \frac{1}{d(x_q, x_i)^2}$$

Donde  $d(x_q, x_i)$  es una medida de distancia entre las dos instancias

En alta dimensionalidad, se requiere una  $f(d)$  que vaya rápidamente a 0 (curse of dimensionality)

KNN le da la misma importancia a instancias independientemente de si están cerca o no del ejemplo

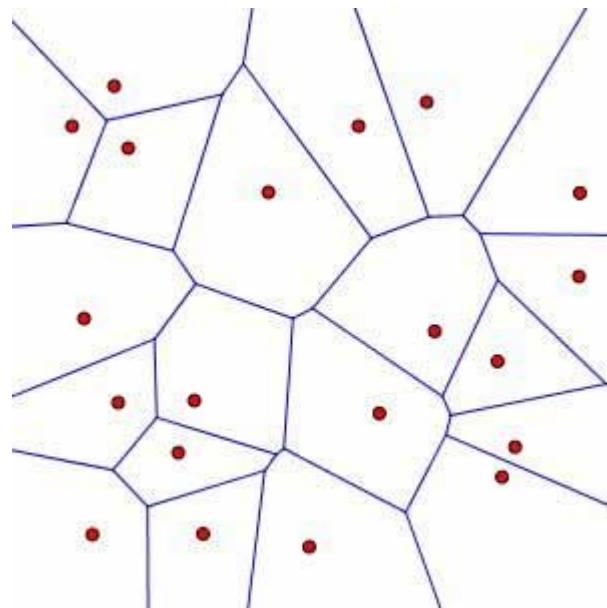
### Asignar más importancia a instancias cercanas

- Se pueden usar todos los ejemplos, no solo K

$$\hat{f}(x_q) = \frac{\sum_{i=1}^k w_i f(x_i)}{\sum_{i=1}^k w_i}, \text{ con } w_i = \exp(-d(x_q, x_i)^2 / \sigma^2)$$

Esta basis function es una mejor alternativa para cuando exista alta dimensionalidad

Un **diagrama de Voronoi** indica las áreas donde una predicción es influenciada por un conjunto de ejemplos



## Ventajas

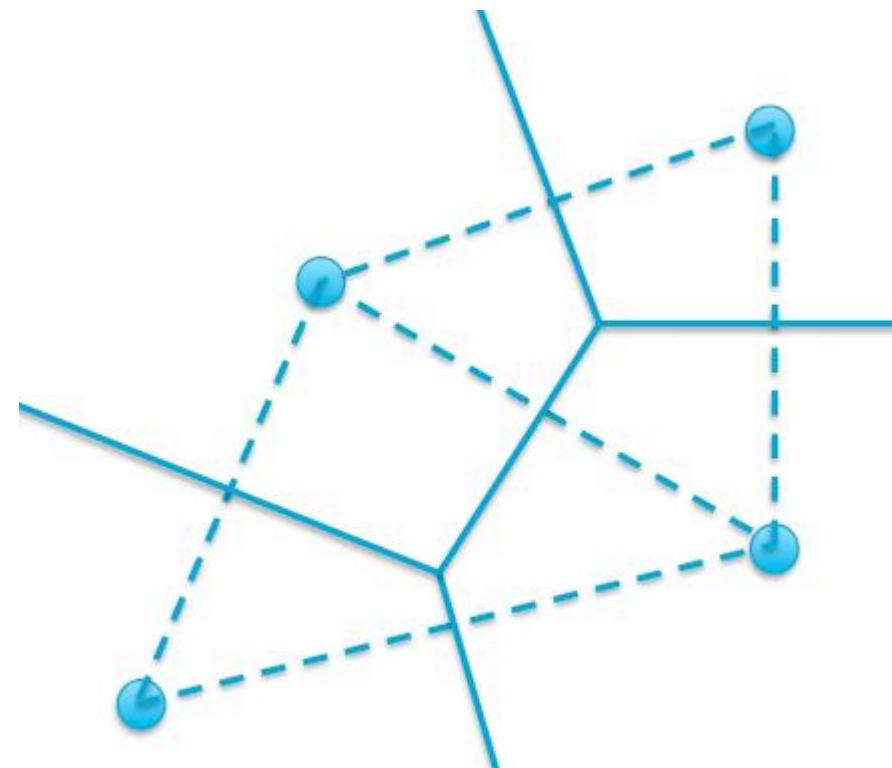
- El aprendizaje es rápido
- La información no se pierde

## Desventajas

- Lento en tiempo de ejecución
- Atributos irrelevantes pueden gobernar la decisión
- Se necesita encontrar una buena medida de similaridad

# Construcción de un diagrama de Voronoi

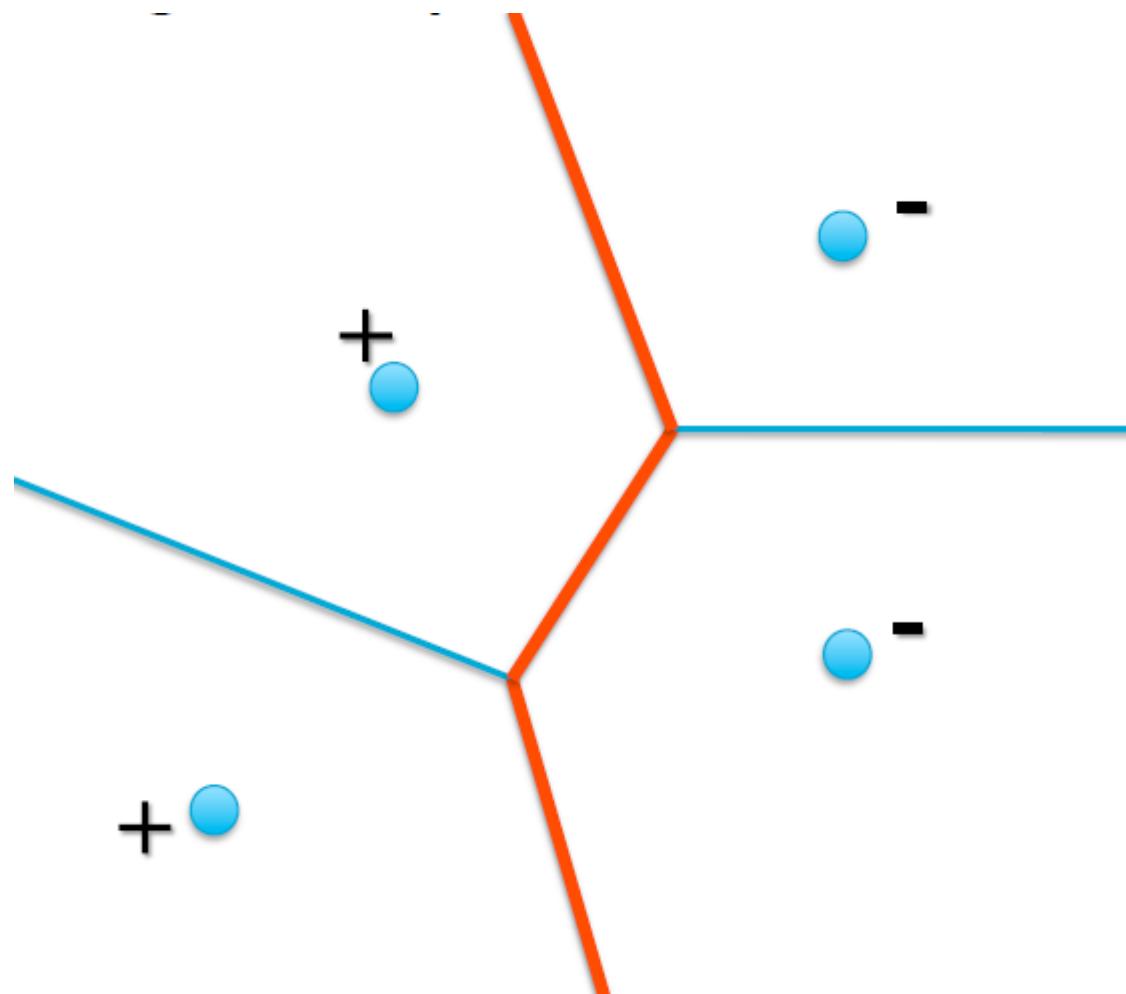
1 nearest neighbour ( $k=1$ ) usando distancia euclídea



Interconectar todas las instancias vecinas

Dibujar una línea recta exactamente en la mitad de todas las instancias

La superficie de decisión separa las regiones de positivos y negativos



# Support Vector Machines resuelven clasificaciones y regresiones no lineales

Sus parámetros se obtienen resolviendo un problema convexo  
Convex optimization problem

Encuentra automáticamente el número de unidades internas  
A diferencia de una red neuronal tradicional

Conceptualmente interesante para espacios de dimensiones gigantes  
Puede aprender y generalizar en estos espacios

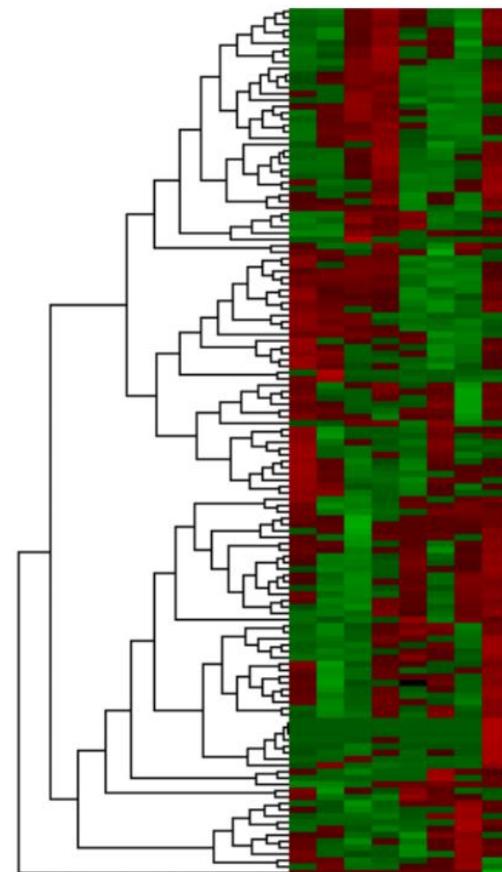
Hace uso de kernels  
Los kernels posibilitan la no linealidad en los problemas

La alta dimensionalidad existe  
en muchas aplicaciones actuales

Ejemplos:

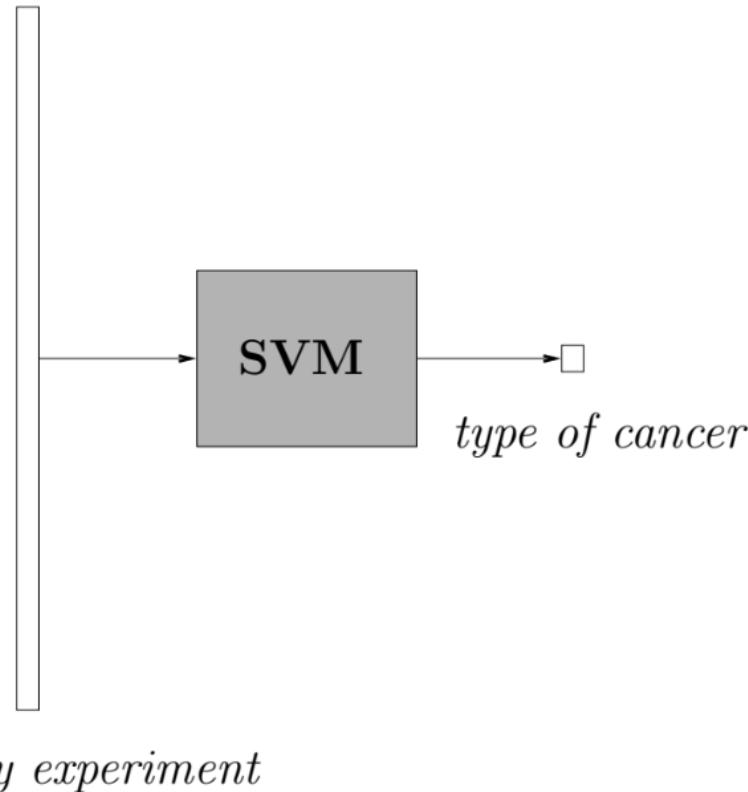
- Experimentos con microarrays en bioinformática
- Clasificación de tumores cerebrales
- Text categorization
- Clasificación de texto escrito

Microarray data – filas: niveles de expresiones  
de los genes; columnas: experimentos



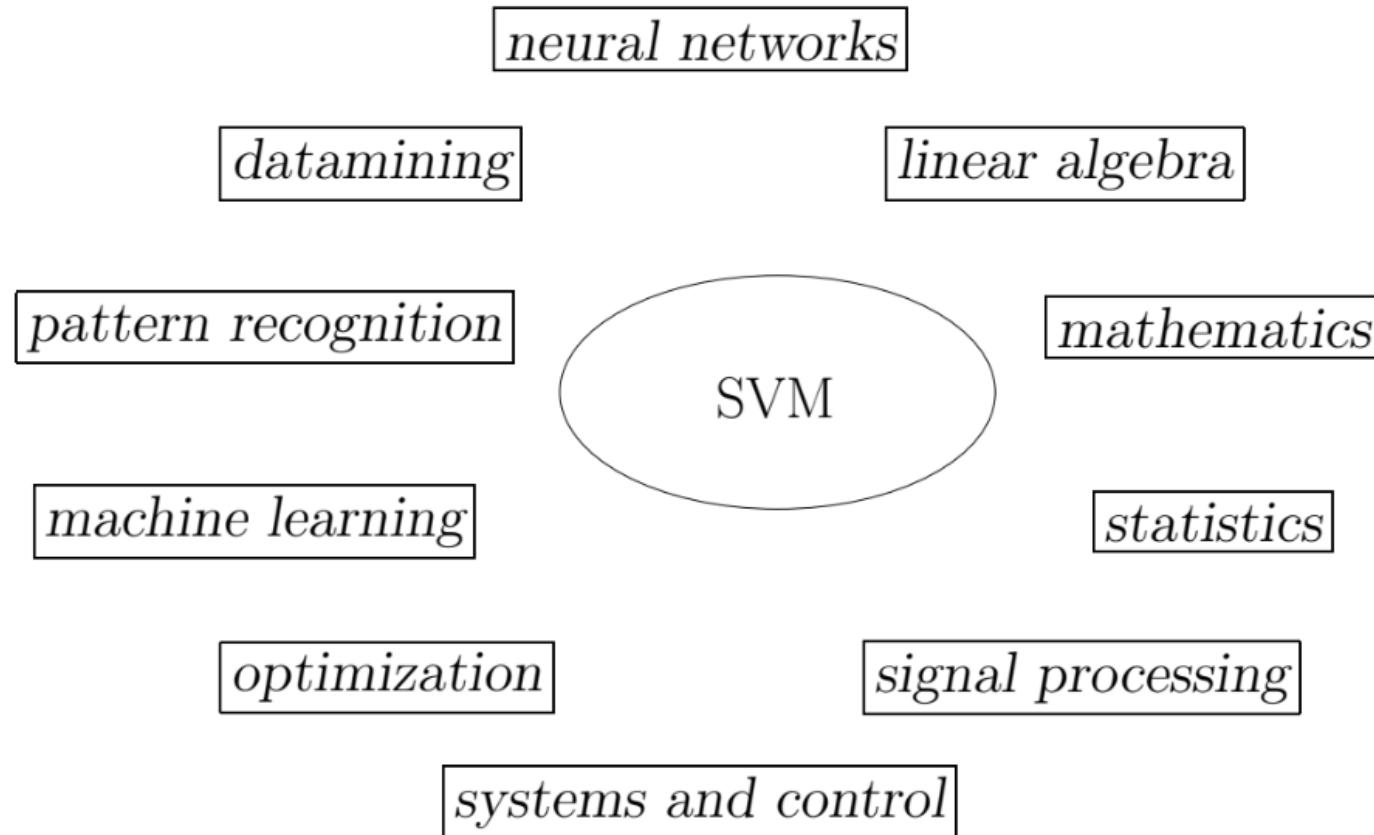
Típicamente: miles de genes, pero solo 50-100  
experimentos

SVM puede lidiar con alta dimensionalidad, MLP no (necesita un proceso de reducción de dimensiones)

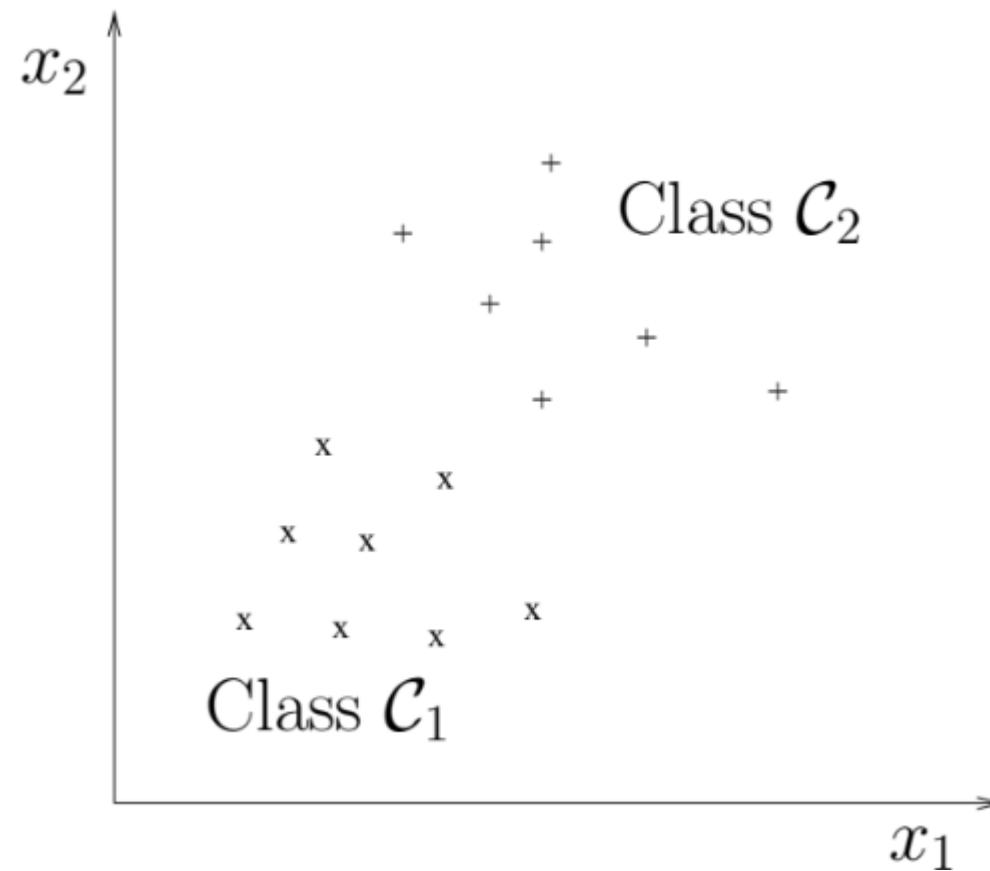


MIT Leukemia dataset  
(7,129 gene expressions; 38 training and 34 test samples):

# SVM tiene nexos con muchas otras áreas del conocimiento

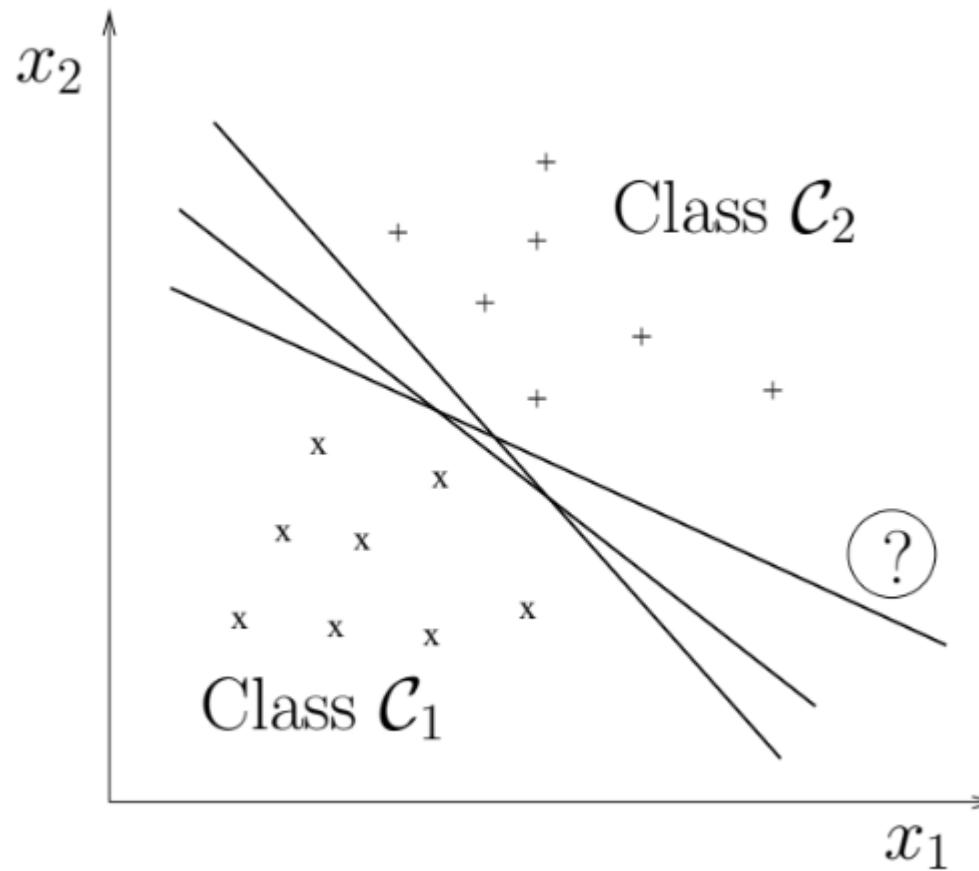


La tarea de clasificación consiste en encontrar el hyper-plano de separación óptimo



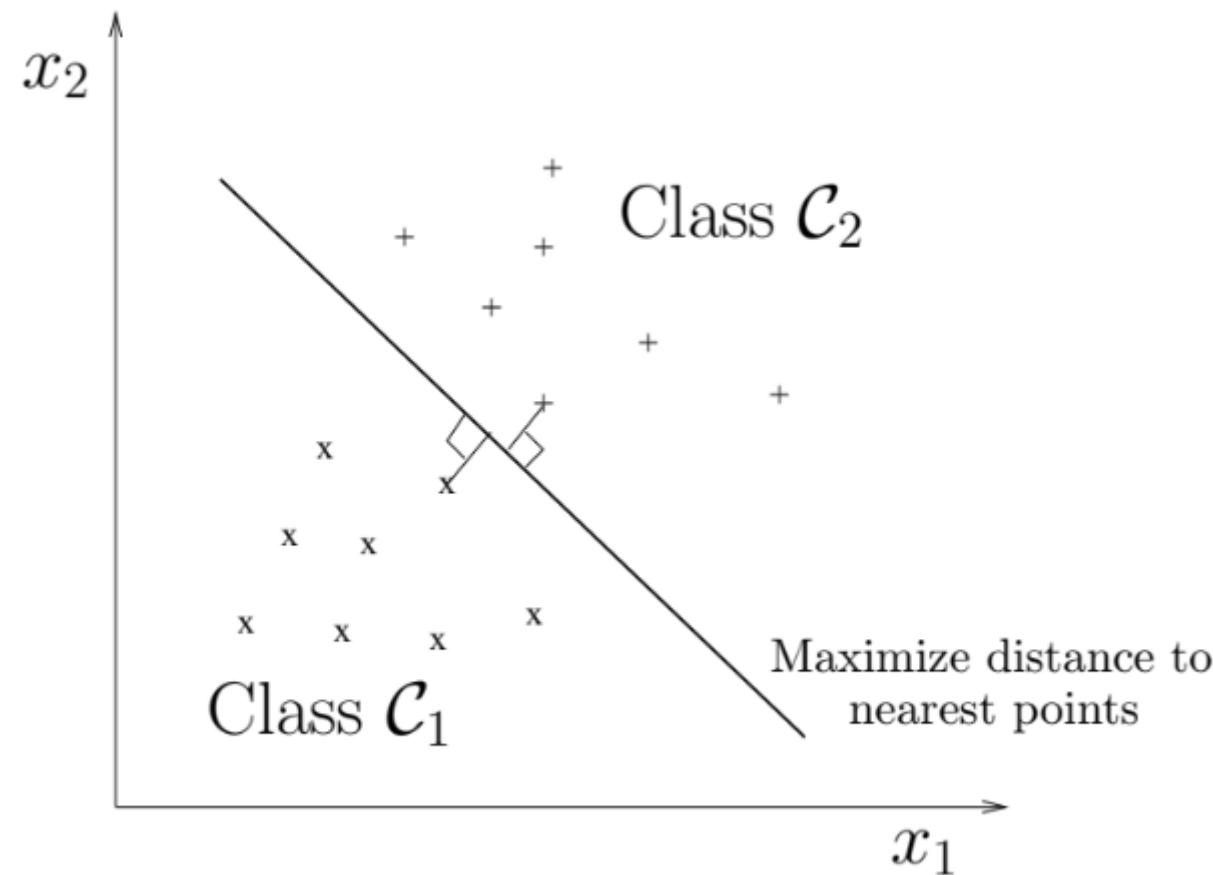
¿Cómo se pueden separar estas dos clases?

La tarea de clasificación consiste en encontrar el hyper-plano de separación óptimo



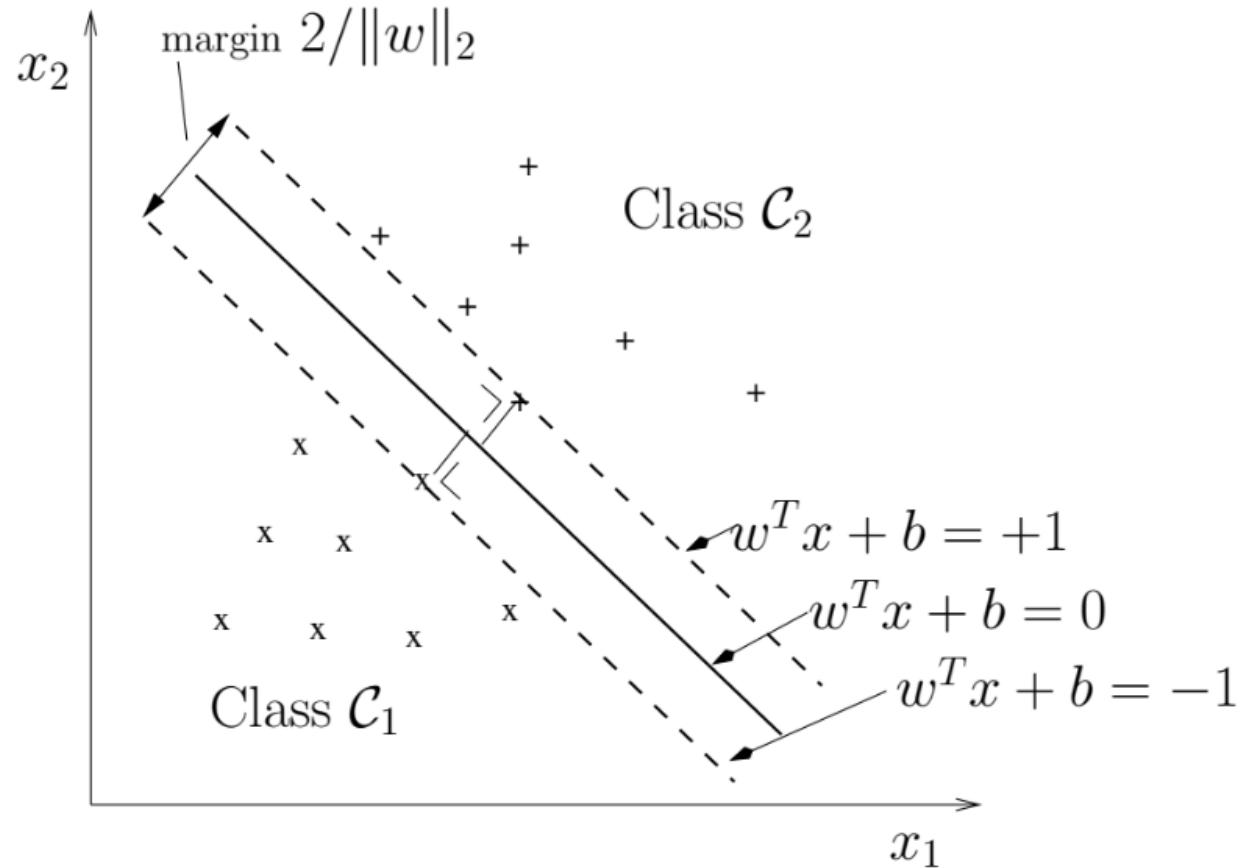
Existen varias posibilidades. ¿Cuál es la mejor?

La tarea de clasificación consiste en encontrar el hyper-plano de separación óptimo



La que maximize la distancia entre los puntos más cercanos (tips of support vectors)

El hyper-plano optimo se obtiene maximizando la distancia entre los puntos más cercanos al mismo



$w^T x_1^* + b = 1$  and  $w^T x_2^* + b = -1$   
Donde,  $x_1^*$  y  $x_2^*$  son los puntos más cercanos al hyperplano

El problema es **minimizar  $\|w\|$** . De esta forma se maximiza el margen

## SVM - linear separable case

Given a training set  $\{x_k, y_k\}_{k=1}^N$

Input patterns  $x_k \in \mathbb{R}^n$

Output patterns  $y_k \in \mathbb{R}$  where  $y_k \in \{-1, +1\}$

Assume

$$\begin{cases} w^T x_k + b \geq +1 & , \text{ if } y_k = +1 \\ w^T x_k + b \leq -1 & , \text{ if } y_k = -1 \end{cases}$$

This is equivalent to

$$y_k [w^T x_k + b] \geq 1, \quad k = 1, \dots, N$$

(i.e. require that **all** training data are **correctly** classified)

Optimization problem:

$$\min_w \frac{1}{2} w^T w \quad \text{s.t.} \quad y_k [w^T x_k + b] \geq 1, \quad k = 1, \dots, N$$

# SVM tiene dos representaciones: primal y dual

in the **primal space**:

$$y(x) = \text{sign}[w^T x + b]$$

with unknowns  $w, b$ .

in the **dual space**:

$$y(x) = \text{sign}\left[\sum_{k=1}^N \alpha_k y_k x_k^T x + b\right]$$

with unknowns  $\alpha, b$

(this follows from the fact that  $w = \sum_{k=1}^N \alpha_k y_k x_k$ )

# SVM utiliza quadratic programming para resolver el problema de optimización

$$\max_{\alpha}$$

$$\alpha^T \begin{array}{|c|} \hline \text{matrix} \\ \hline \end{array} \alpha + \begin{array}{|c|} \hline \text{matrix} \\ \hline \end{array} \alpha$$

such that

$$\begin{array}{|c|} \hline \text{matrix} \\ \hline \end{array} = 0$$

$\alpha$

Note that:

- the size of the matrix grows with the number of training data points (if  $N = 10^6$  then the size of the matrix is  $10^6 \times 10^6$  !)
- the size of the solution vector  $\alpha$  is only determined by the number of training data  $N$  and not by the dimension of the input space

Los valores  $\alpha_k$  dan el nombre a los support vectors

La matriz en el problema QP es “positive definite” o “positive semidefinite”:

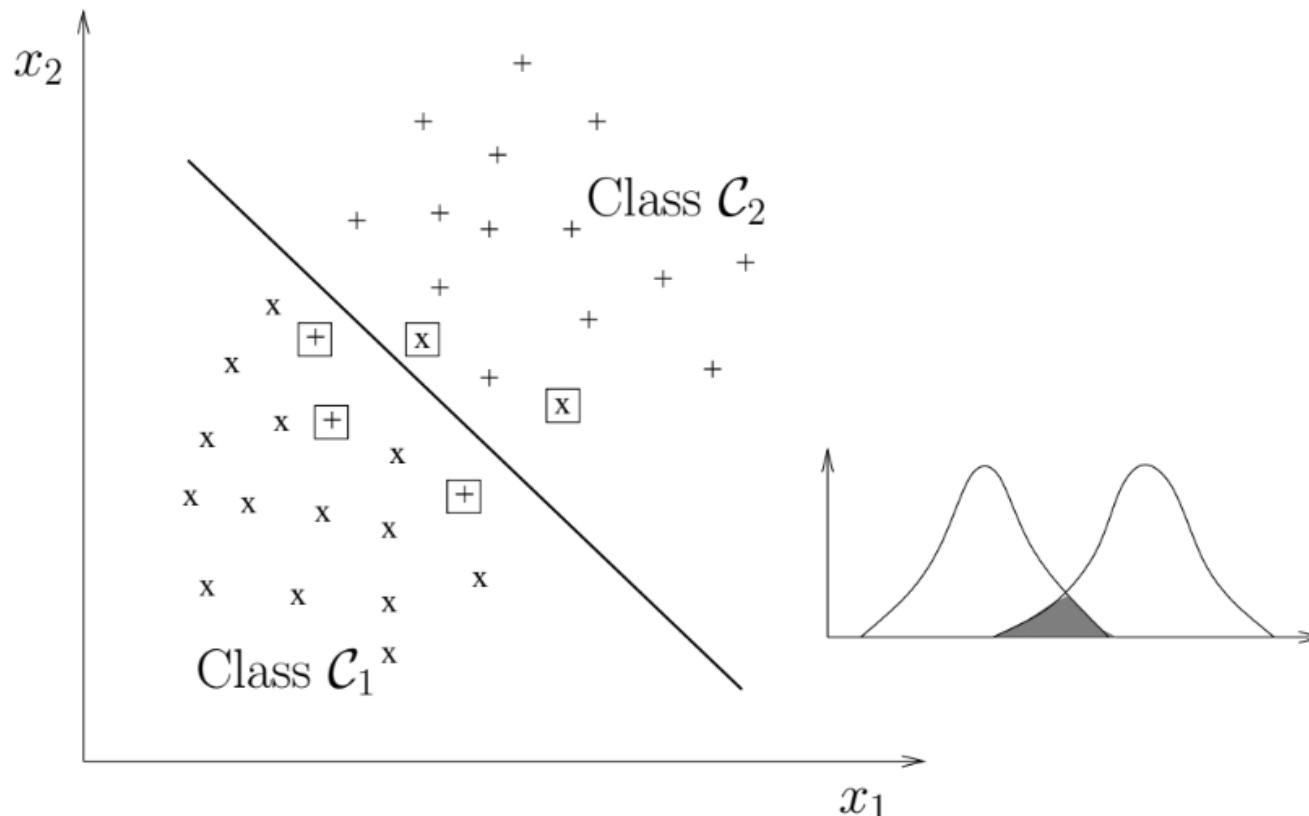
- Pos. Def. – solución para  $\alpha$  única
- Pos. Semidef. – posiblemente muchas soluciones al problema, pero  $w, b$  tiene solución única

La solución es dispersa

- Muchos elementos de  $\alpha$  son cero
- $y(x) = \text{sign}[\sum_{k \in S_{SV}} \alpha_k y_k x_k^T x + b]$ , donde  $S_{SV}$  denota el conjunto de  $\alpha_k \neq 0$

Support values: todos los  $\alpha_k \neq 0$ . Support vectors: datos de entrenamiento que corresponden a  $\alpha_k \neq 0$

Muchas veces las instancias no pueden ser separadas de forma lineal



En este caso existen instancias que están en el lado incorrecto del hyperplano

Para solucionar el problema, se introducen *slack* variables al problema de optimización

Modify the inequalities into

$$y_k[w^T x_k + b] \geq 1 - \xi_k, \quad k = 1, \dots, N$$

with **slack variables**  $\xi_k > 0$  such that the original inequalities can be violated for certain points if needed.

A misclassified point corresponds to  $\xi_k > 1$ .

Para solucionar el problema, se introducen *slack* variables al problema de optimización

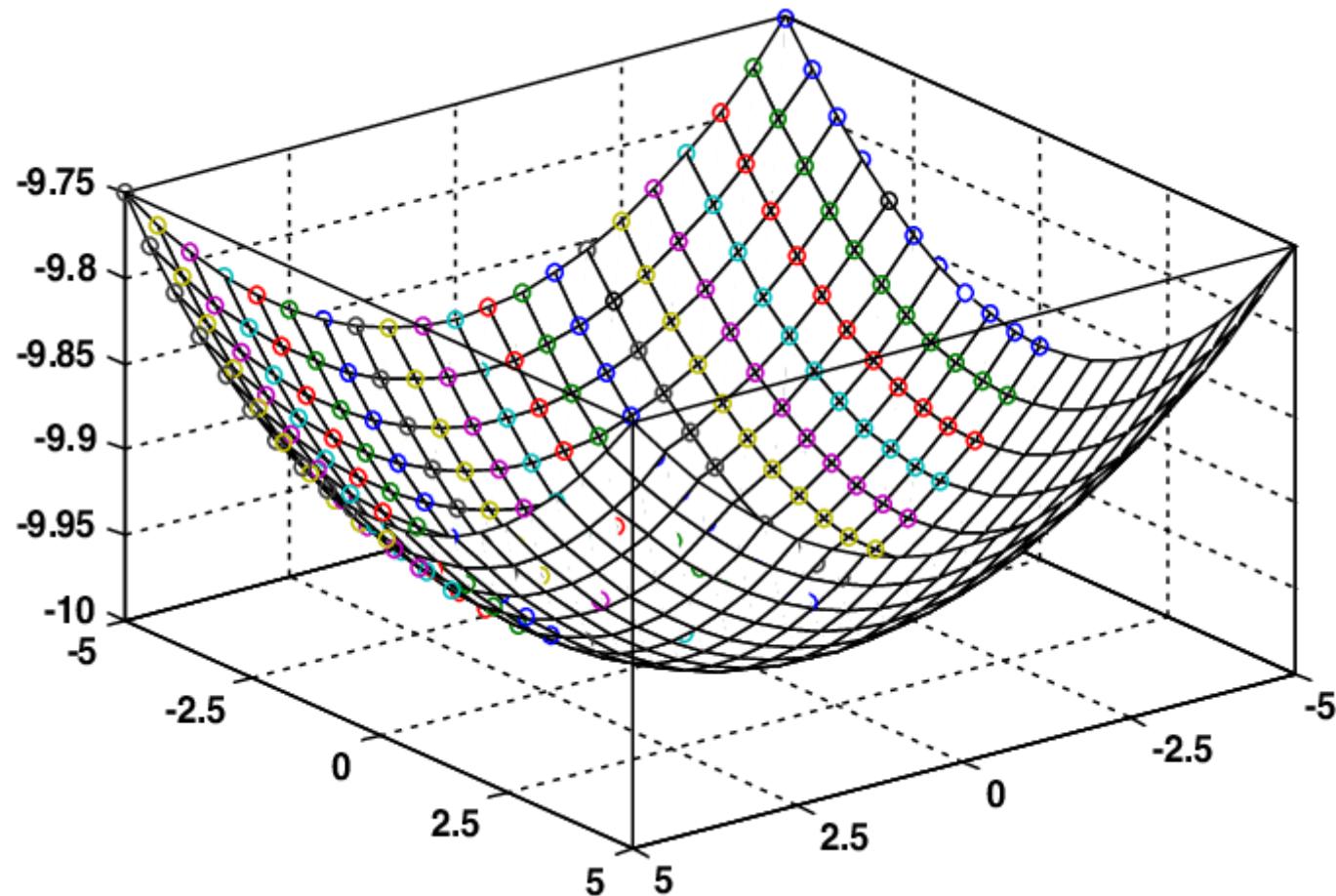
Optimization problem

$$\min_{w,b,\xi} \mathcal{J}(w, \xi) = \frac{1}{2} w^T w + c \sum_{k=1}^N \xi_k$$

subject to

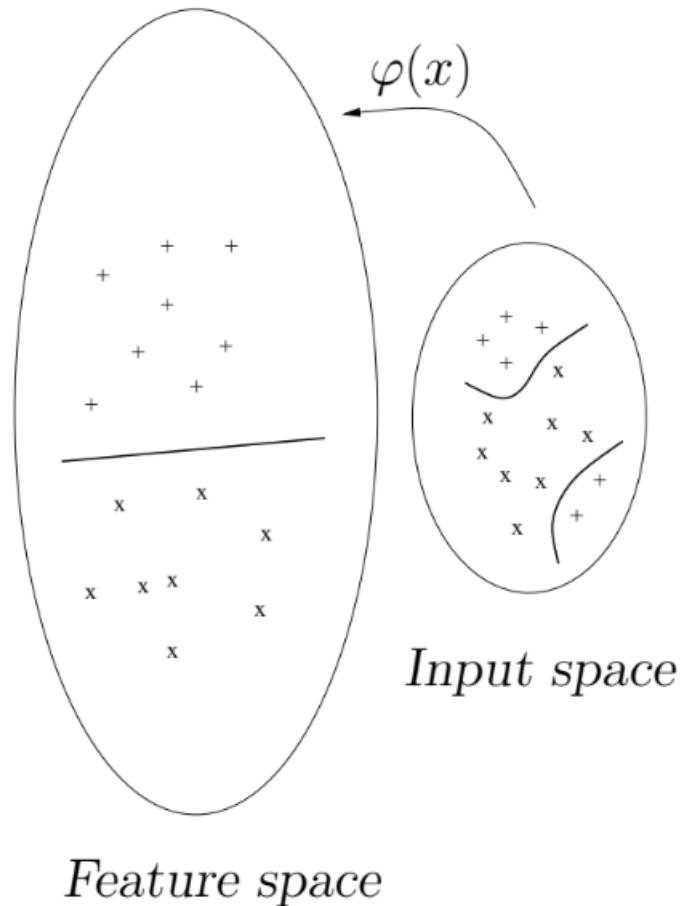
$$\begin{cases} y_k [w^T x_k + b] \geq 1 - \xi_k, & k = 1, \dots, N \\ \xi_k \geq 0, & k = 1, \dots, N. \end{cases}$$

Como el problema es cuadrático, la superficie de la parábola tiene un único mínimo global



Esta es una de las características más importantes de SVM

# Cuando los datos no son separables, se hace uso de transformaciones (**nonlinear SVM**)



En el espacio original (input space) los datos no son separables

Los datos deben ser mapeados a un nuevo espacio (feature space). Este espacio tiene alta dimensionalidad, pero existe separabilidad

Existen muchas posibilidades de funciones  $\varphi(x)$  para hacer este mapeo

# Nonlineal SVM classifier

Given a training set  $\{x_k, y_k\}_{k=1}^N$

Input patterns  $x_k \in \mathbb{R}^n$

Class labels  $y_k \in \mathbb{R}$  where  $y_k \in \{-1, +1\}$

Classifier:

$$y(x) = \text{sign}[w^T \varphi(x) + b]$$

with  $\varphi(\cdot) : \mathbb{R}^n \rightarrow \mathbb{R}^{n_h}$  mapping to high dimensional feature space (can be infinite dimensional)

# Nonlineal SVM classifier

For separable data, assume

$$\begin{cases} w^T \varphi(x_k) + b \geq +1 & , \text{ if } y_k = +1 \\ w^T \varphi(x_k) + b \leq -1 & , \text{ if } y_k = -1 \end{cases}$$

which is equivalent to

$$y_k [w^T \varphi(x_k) + b] \geq 1, \quad k = 1, \dots, N$$

Optimization problem (**non-separable case**):

$$\min_{w,b,\xi} \mathcal{J}(w, \xi) = \frac{1}{2} w^T w + c \sum_{k=1}^N \xi_k$$

subject to

$$\begin{cases} y_k [w^T \varphi(x_k) + b] \geq 1 - \xi_k, & k = 1, \dots, N \\ \xi_k \geq 0, & k = 1, \dots, N. \end{cases}$$

Existen muchas opciones de mapeo (**kernels**) que pueden ser utilizados para construir un modelo de clasificación

Some possible **kernels**  $K(\cdot, \cdot)$ :

$$K(x, x_k) = x_k^T x \text{ (linear SVM)}$$

$$K(x, x_k) = (x_k^T x + \tau)^d \text{ (polynomial SVM of degree } d\text{)}$$

$$K(x, x_k) = \exp\{-\|x - x_k\|_2^2/\sigma^2\} \text{ (RBF SVM)}$$

$$K(x, x_k) = \tanh(\kappa x_k^T x + \theta) \text{ (MLP SVM)}$$

In the case of RBF and MLP kernel, the number of hidden units corresponds to the number of support vectors.

# Formulaciones para non-linear SVM

**Primal problem:**

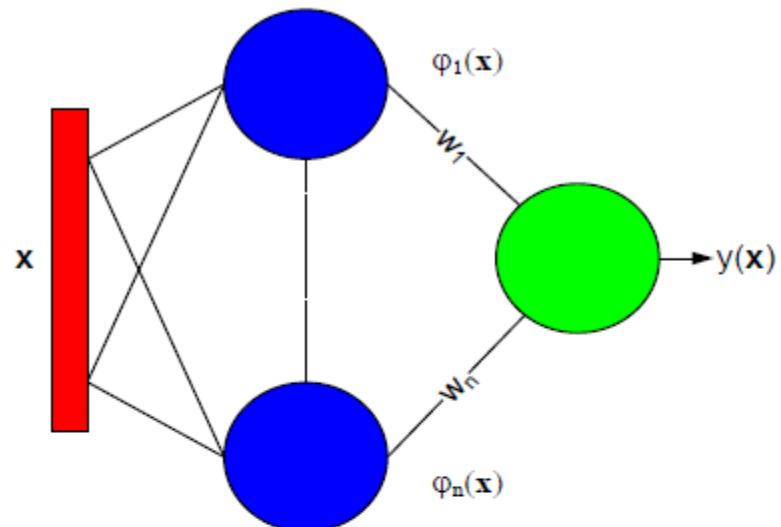
$$\begin{cases} \min_{w,b,\xi} \| w \|_2^2 + C \sum_{i=1}^N \xi_i \\ s.t. \\ \forall i, y_i (w^T \varphi(x_i) + b) \geq 1 - \xi_i \\ \forall i, \xi_i \geq 0 \end{cases}$$

*K(x<sub>i</sub>, x<sub>j</sub>) = φ(x<sub>i</sub>)<sup>T</sup>φ(x<sub>j</sub>) ("Kernel trick")*

**Dual problem:**

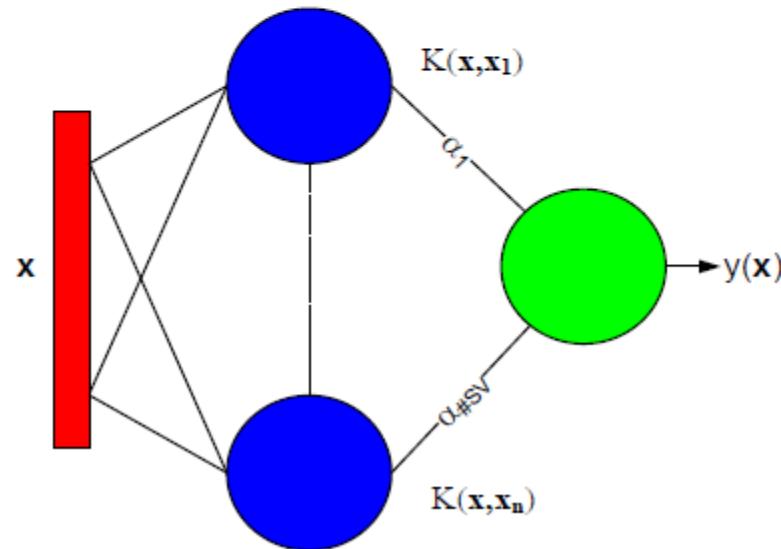
$$\begin{cases} \max_{\alpha_k} -\frac{1}{2} \sum_{k,l=1}^N y_k y_l K(x_k, x_l) \alpha_k \alpha_l + \sum_{k=1}^N \alpha_k \\ s.t. \\ \sum_{k=1}^N \alpha_k y_k = 0 \\ 0 \leq \alpha_k \leq C, k = 1, \dots, N \end{cases}$$

# Formulaciones para non-linear SVM



*Primal problem*

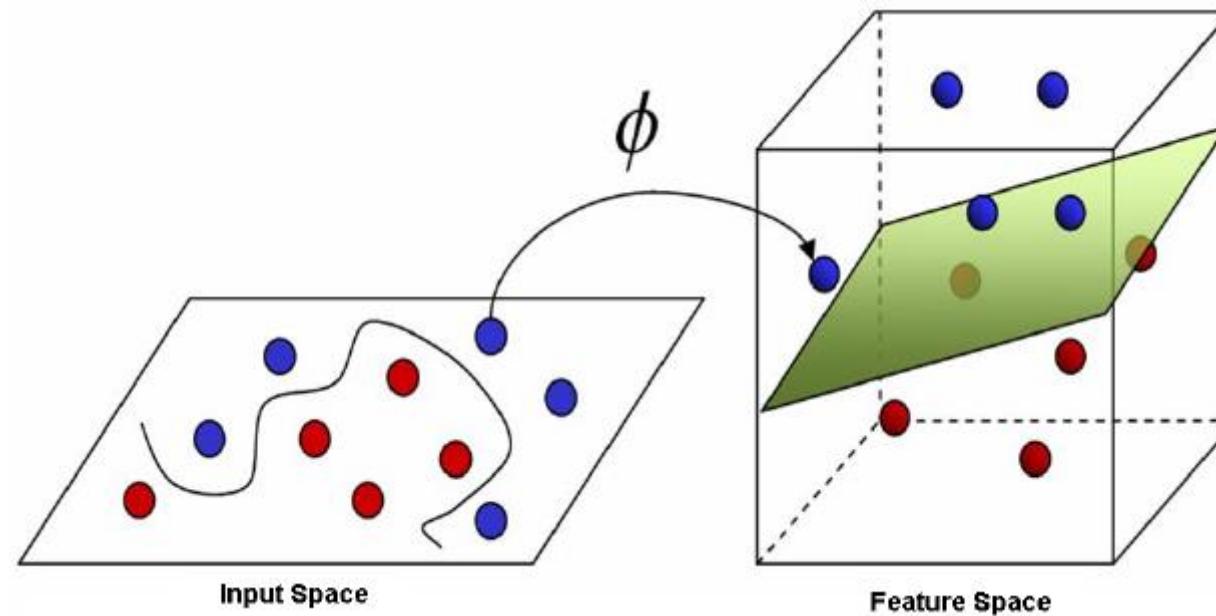
$$y(x) = \text{sign}(w^T \varphi(x) + b)$$



*Dual problem*

$$y(x) = \text{sign}\left[\sum_{k=1}^{\#SV} \alpha_k y_k K(x, x_k) + b\right]$$

El mapeo de instancias a un feature space  
hace que existe un hyperplano que separe las clases



#hidden units = #support vectors

# Lazy vs eager learning

## Lazy learning:

- Espera que se haga una pregunta
- Solo entonces, comienza a procesarla
- Ejemplos: KNN, IBL

## Eager learning:

- Generaliza antes de que se le presenten preguntas
- Ejemplos: TDIDT, Naive Bayes, ANN, SVM

Para que un modelo de SVM generalice correctamente, se deben determinar correctamente sus hyperparámetros

Dado un clasificador SVM con un kernel RBF

$$y(x) = \text{sign} \left[ \sum_{k=1}^N \alpha_k y_k \exp \left( -\frac{\|x - x_k\|_2^2}{\sigma^2} \right) + b \right]$$

Se debe estimar:

- $\alpha, b$  mediante el problema de optimización QP (quadratic problem)
- $\sigma \rightarrow \text{¿Como?}$
- ¿Existen otros parámetros?

Para que un modelo de SVM generalice correctamente, se deben determinar correctamente sus hyperparámetros

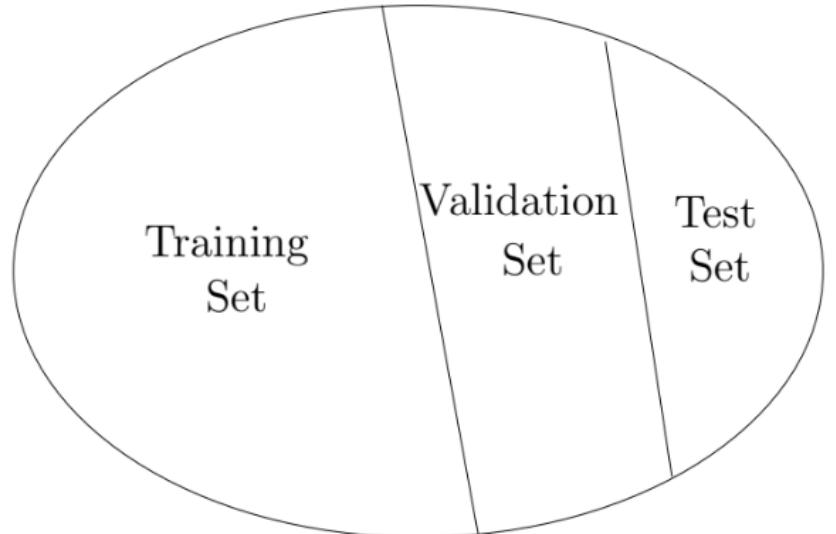
Dado un clasificador SVM con un kernel RBF

$$y(x) = \text{sign} \left[ \sum_{k=1}^N \alpha_k y_k \exp \left( -\frac{\|x - x_k\|_2^2}{\sigma^2} \right) + b \right]$$

El parámetro  $\sigma$  es muy importante para la generalización

- Si es muy grande, los límites de decisión tienden a ser lineales -> underfitting
- Si es muy pequeño, los límites de decisión tienden a ser altamente no-lineales -> overfitting

Para estimar  $\sigma$  se realizan divisiones en los datos de entrenamiento



Encontrar  $\alpha$  y  $b$  con los datos de entrenamiento

Luego probar varios valores de  $\sigma$

Seleccionar el valor de sigma para el cuál el error de validación sea el mínimo

Finalmente chequear el modelo con los datos de prueba

# Existen varios métodos para seleccionar un parámetro

## Optimización con un test de validación

- Se debe realizar esta partición de forma manual
- La generalización debe chequearse con un test de prueba

## Cross-validation

- Los datos de entrenamiento son partidos varias veces
- Se selecciona el parámetro con mejor rendimiento entre varias ejecuciones

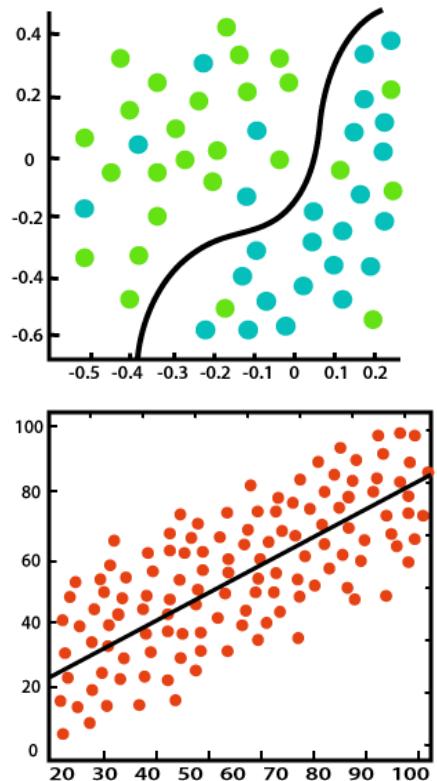
## Inferencia bayesiana

## Fronteras de generalización (VC theory)

# Averiguar

¿Cuál es la formulación de SVM para realizar estimaciones de funciones?

# Content



Arboles de decisión

Como se crean, conceptos

Inducción de conjuntos de reglas

Generalización a través de reglas

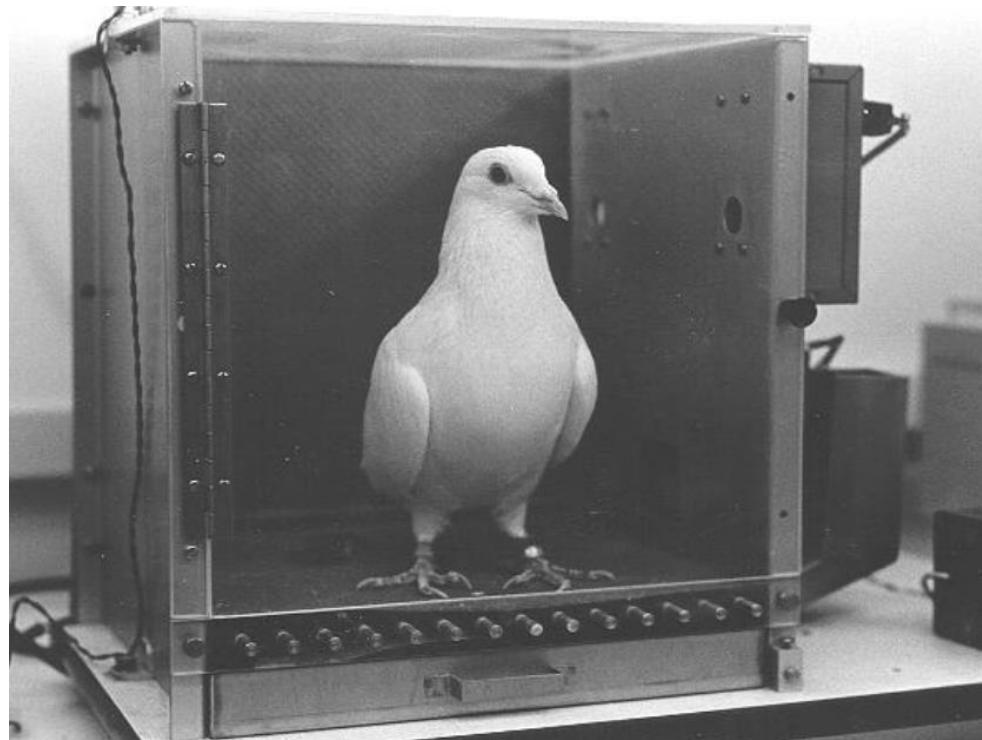
Support Vector Machines

Kernels, hyperparámetros

Artificial Neural Networks

Nuevas arquitecturas y aplicaciones

# Las redes neuronales artificiales (ANN) se inspiran en sistemas neuronales biológicos



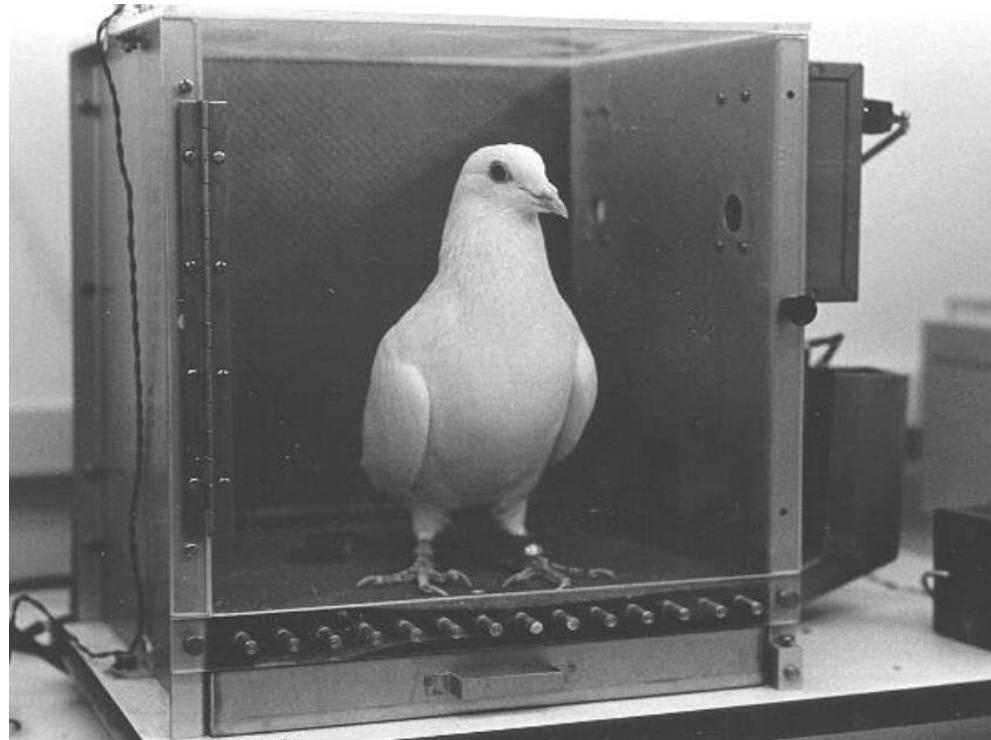
Experimento: Palomas expertas

Una paloma fue puesta en una jaula

Se le presentaron pinturas de diferentes artistas, e.j.  
Chagall/Van Gogh

Se le premiaba cuando picoteaba al presentarle  
pinturas de un artista definido: Van Gogh

# La paloma tuvo una efectividad del 95% discriminando pinturas de varios artistas



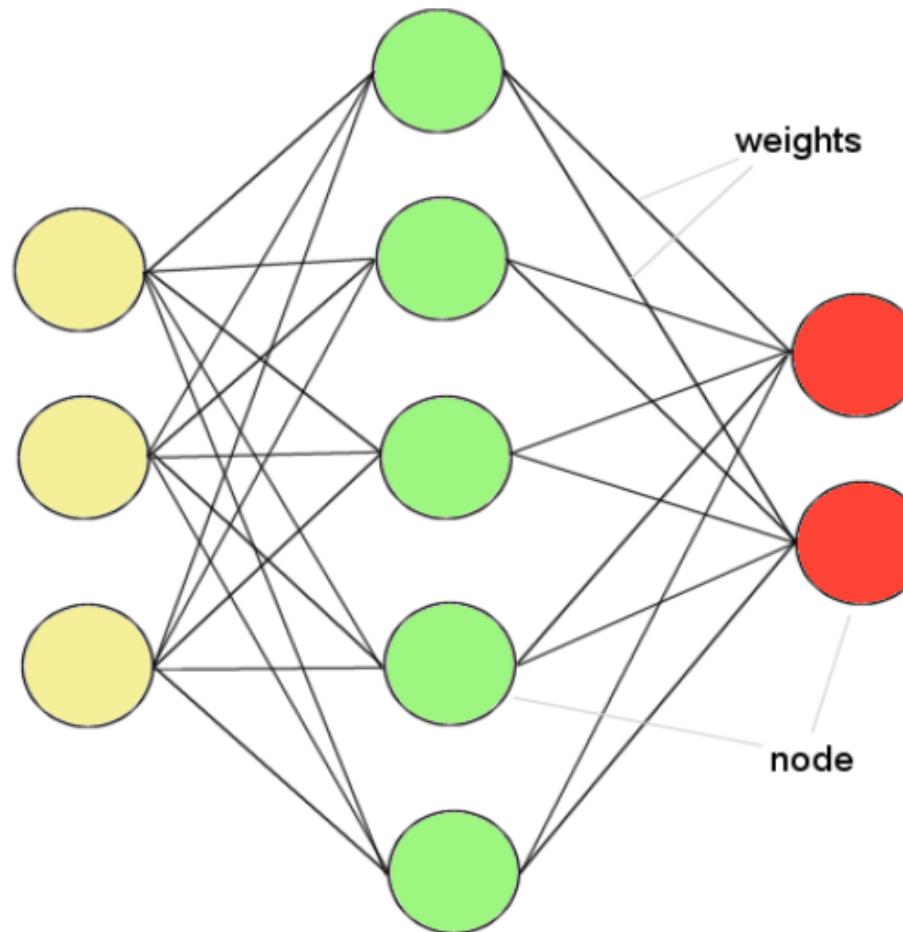
Efectividad del 85% cuando se le presentaron nuevas  
pinturas de Van Gogh (generalización)

La paloma no memoriza, aprende

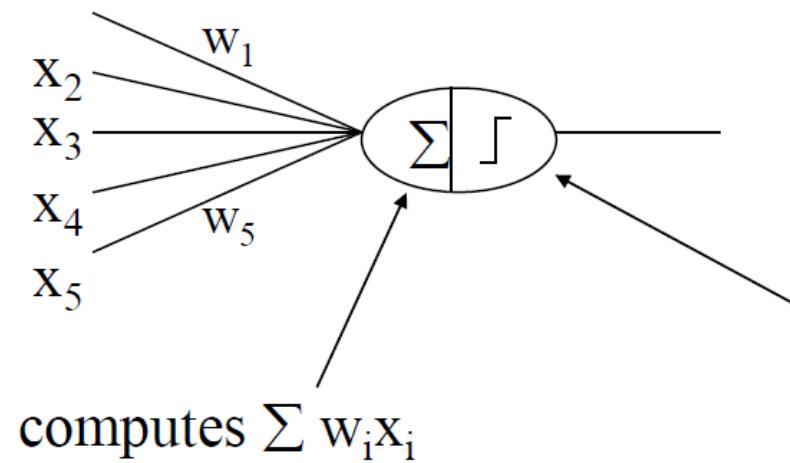
Pudo extraer exitosamente el patrón (estilo)

Las redes neuronales (biológicas y artificiales) son  
buenas en estas tareas

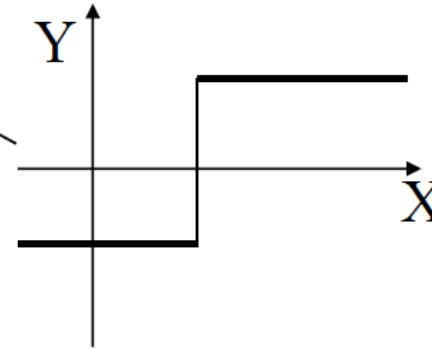
ANN's se componen de **neuronas** (nodos)  
y **sinapsis** (pesos)



# La ANN más simple es el Perceptrón



threshold function:  
 $Y = -1 \text{ if } X < t, Y = 1 \text{ otherwise}$



El perceptrón simula una neurona

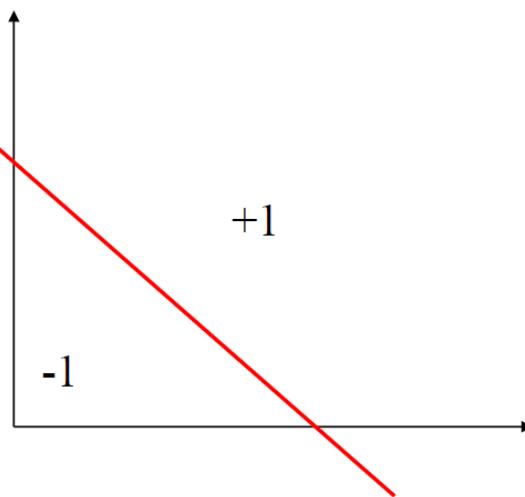
La neurona se dispara si la suma de entradas  $\sum w_i x_i > t$ , donde  $t$  es el umbral (threshold)

# Un perceptrón aprende una función para separar linealmente dos clases

Funcionalmente parecido a SVM lineal

El perceptrón aprende una función de la forma  $ax + by$

Si  $ax+by>c$  entonces la clase es +1; caso contrario, -1



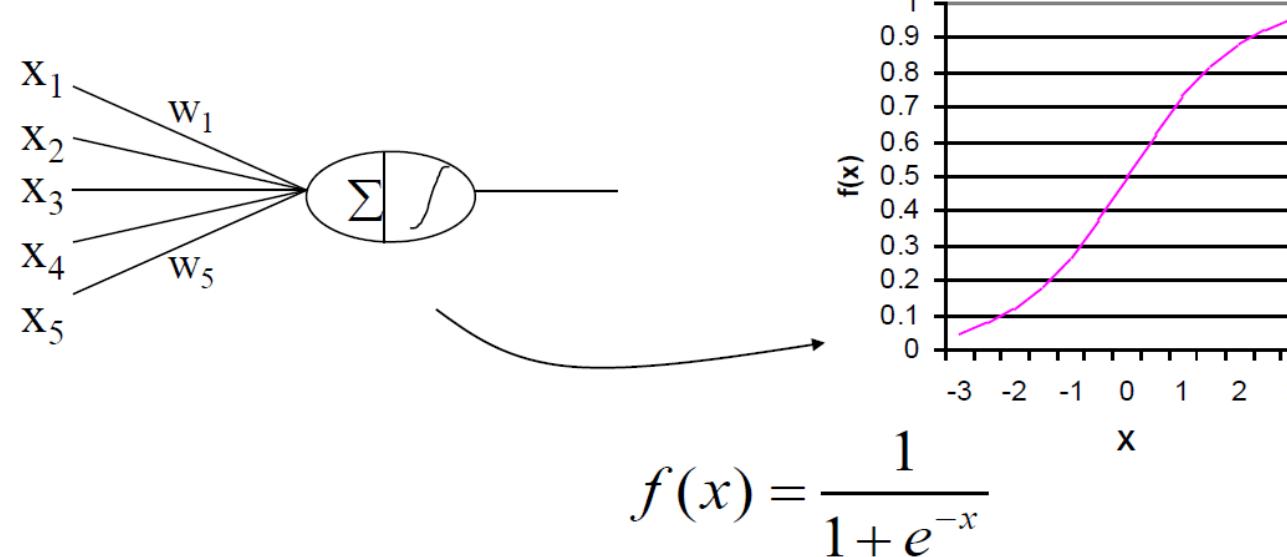
Cuando existen n-entradas, el perceptrón construye un hiperplano en un espacio n-dimensional

La idea del hiperplano es similar a SVM lineal

Las clases deben ser linealmente separables

- Serias limitaciones con problemas complejos
- Su poder de generalización es muy limitado

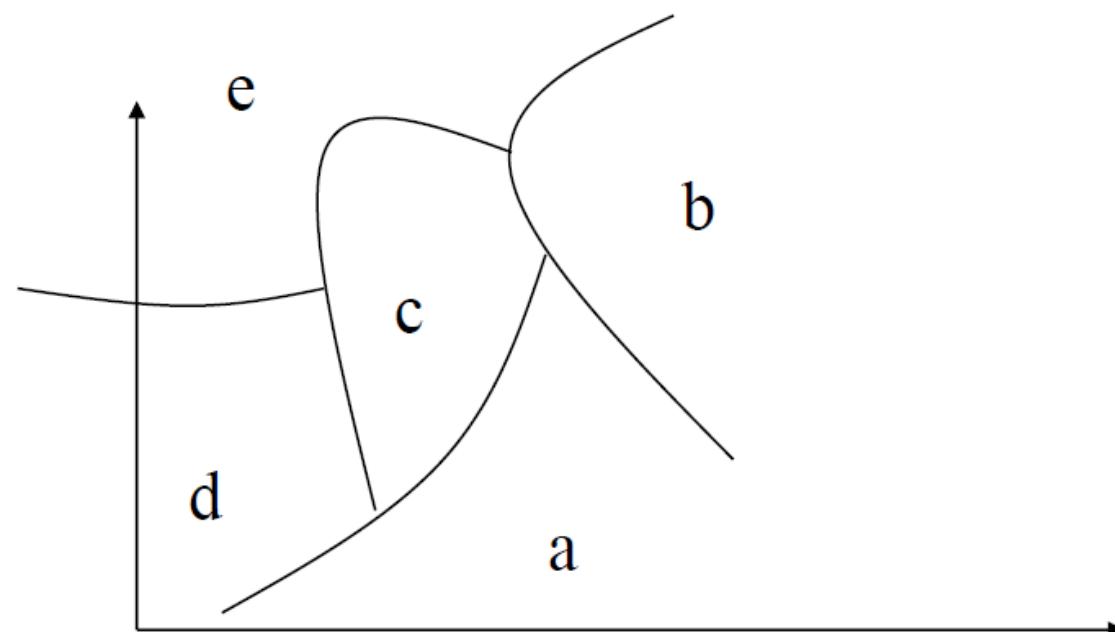
Muchas veces se prefiere usar una función de activación sigmoid en vez de un threshold



Sus valores cambian de forma continua

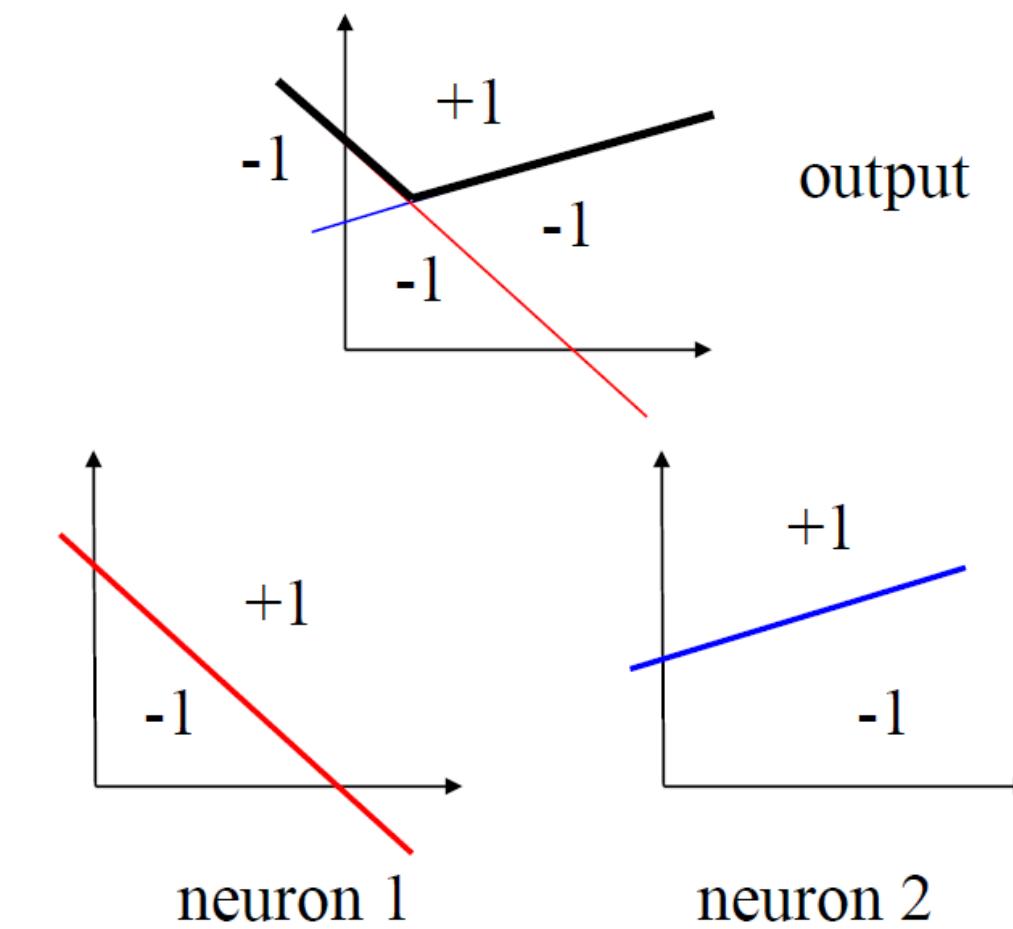
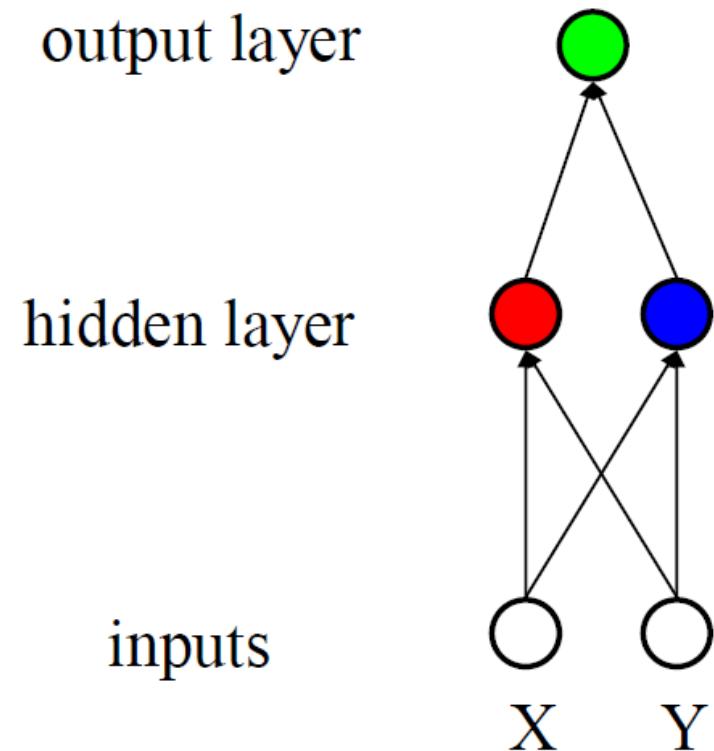
Su uso tiene ventajas matemáticas al momento de entrenar la red

Esta función mejora la capacidad de representación:  
genera superficies de decisión no lineales



Para cada una de las 5 clases, existen 5  
áreas de decisión

Las redes multi-capas incrementan el poder de representación combinando varias neuronas en una red



En una red **feedforward**, cada capa tiene como entrada la salida de la capa previa

Las neuronas de capas sucesivas están interconectadas

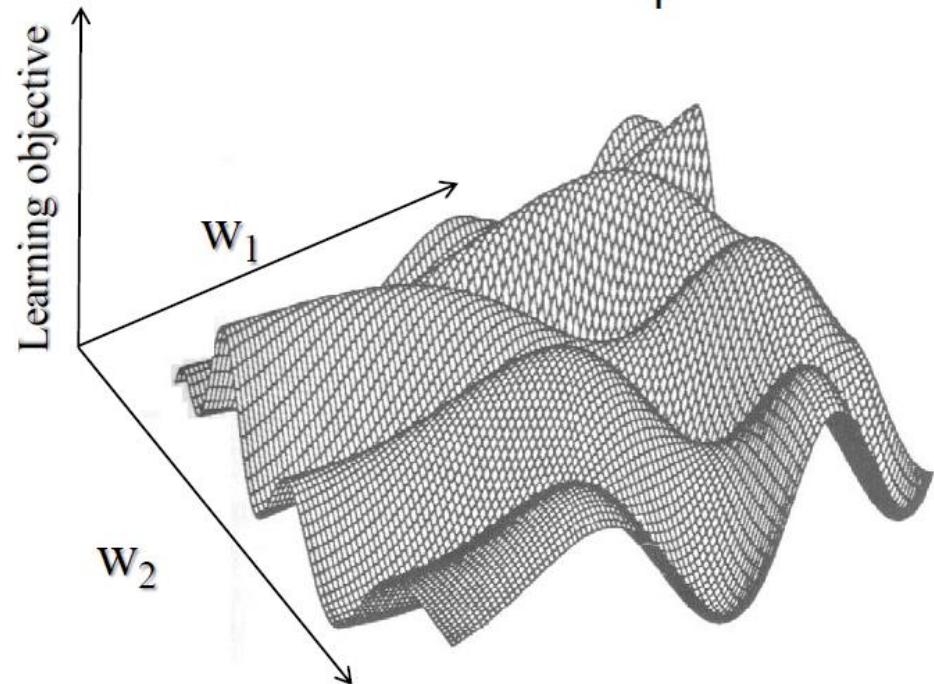
Capas sucesivas = diferentes representaciones de entradas

Redes Feedforward de 2 capas son muy populares todavía

Las configuraciones de los pesos de interconexiones determinan el comportamiento de la red

¿Como se determinan los pesos?

Tradicionalmente se usa **backpropagation** como  
técnica de entrenamiento en redes neuronales



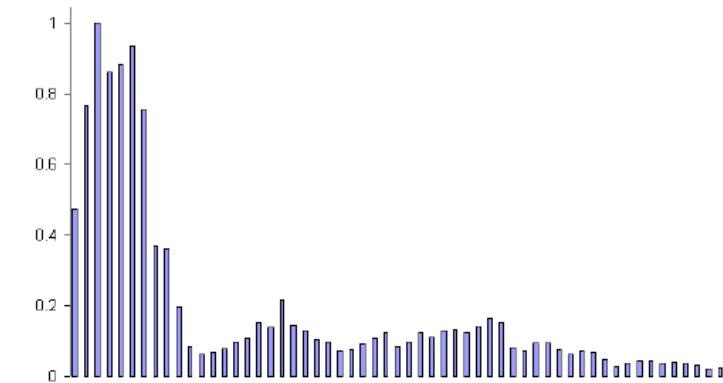
- Se requieren datos de entrenamiento
- Se comienzan con valores pequeños y aleatorios para inicializar los pesos
- El error en la salida es usado para ajustar los pesos (backpropagation)
- Se utiliza gradiente descent para determinar los pesos
- La superficie del error tiene varios mínimos locales

**Problema!**

Ejemplo: aprender a discriminar entre dos diferentes voces diciendo “hola”

## Data

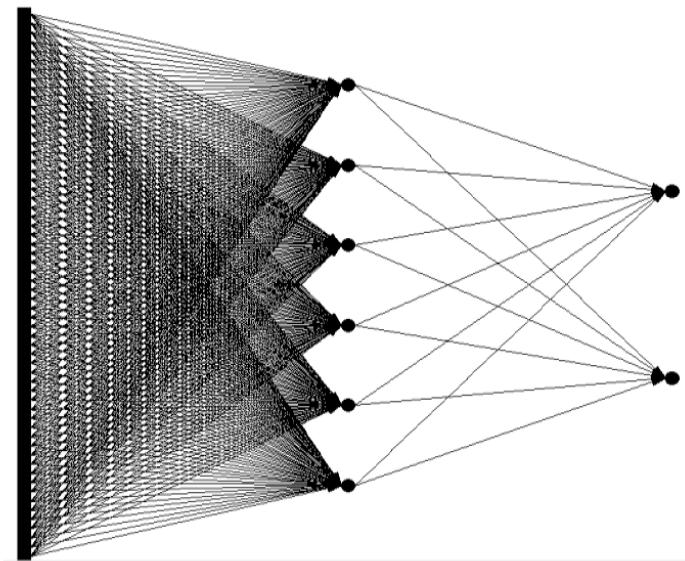
- Sources
  - Steve Simpson
  - David Raubenheimer
- Format
  - Frequency distribution (60 bins)
  - Analogy: cochlea



Ejemplo: aprender a discriminar entre dos diferentes voces diciendo “hola”

### Network architecture

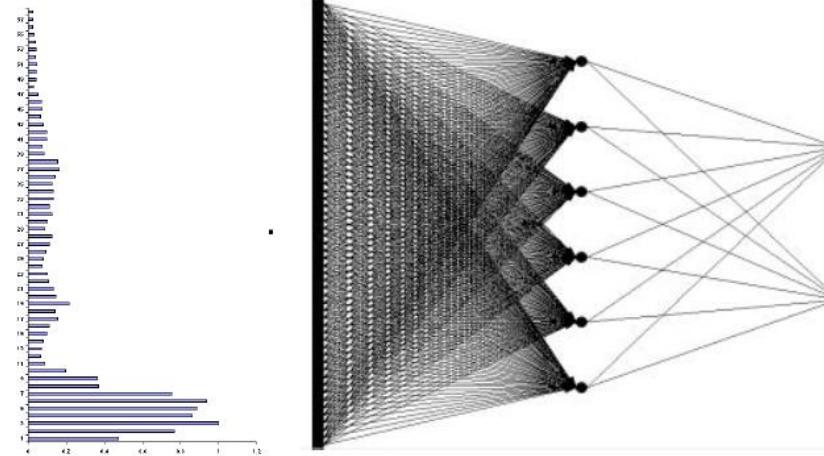
- Feed forward network
  - 60 input (one for each frequency bin)
  - 6 hidden
  - 2 output (0-1 for “Steve”, 1-0 for “David”)



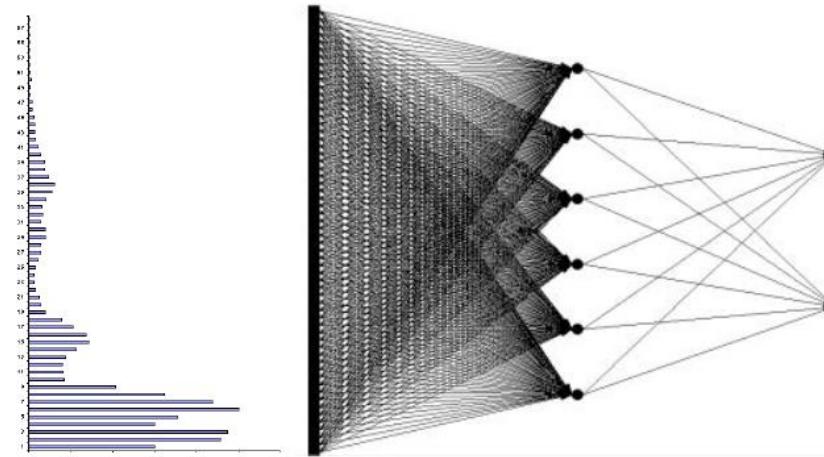
# Ejemplo: aprender a discriminar entre dos diferentes voces diciendo “hola”

Los datos se presentan a la red

Steve



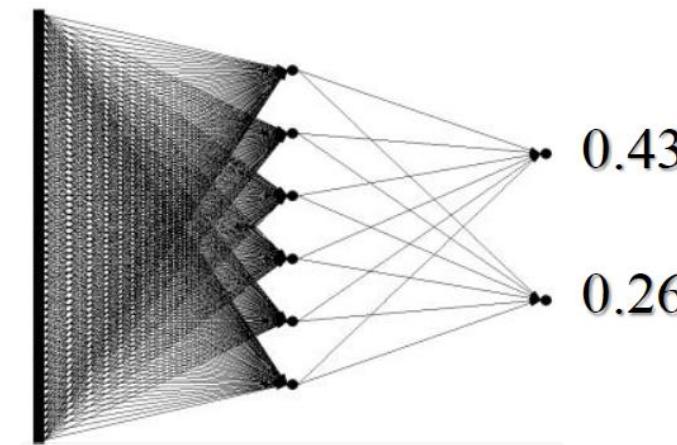
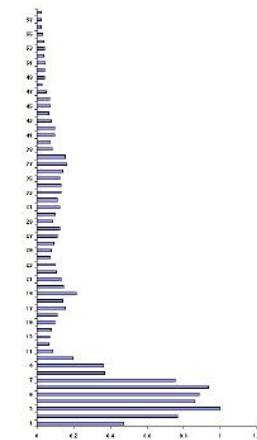
David



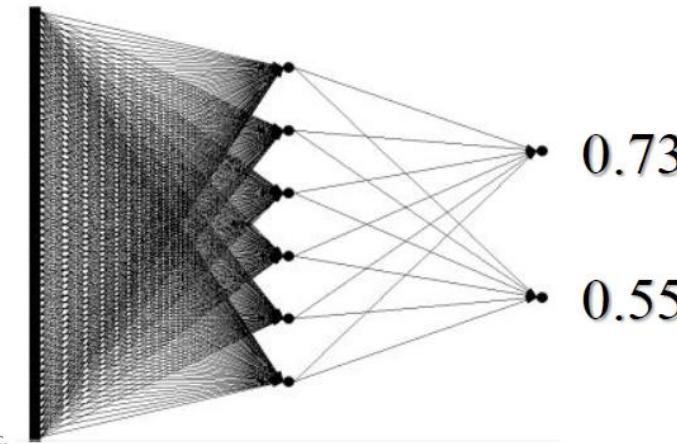
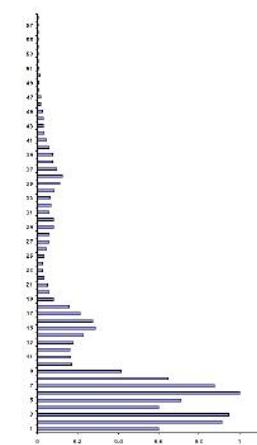
# Ejemplo: aprender a discriminar entre dos diferentes voces diciendo “hola”

Cuando la red no está entrenada, los resultados son incorrectos

Steve



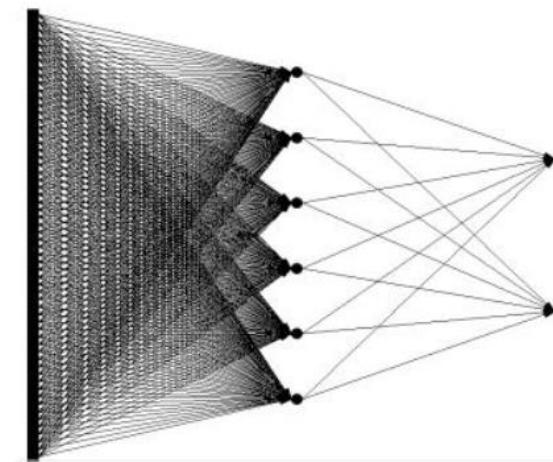
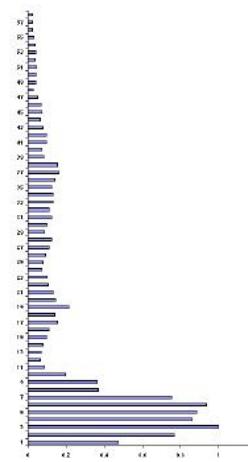
David



# Ejemplo: aprender a discriminar entre dos diferentes voces diciendo “hola”

Se calcula el error de la red

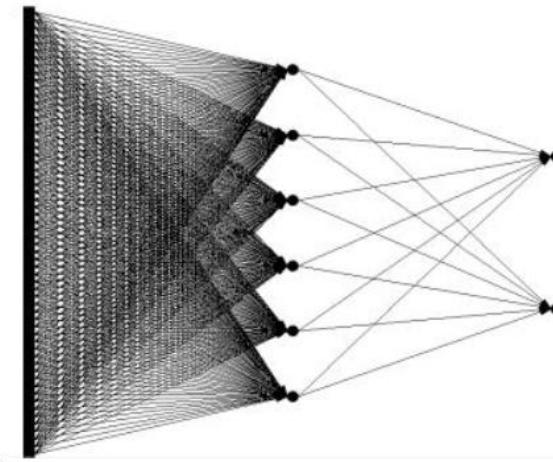
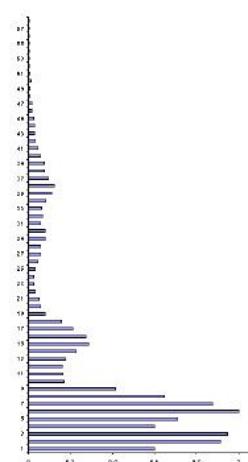
Steve



$$\bullet \quad 0.43 - 0 = 0.43$$

$$\bullet \quad 0.26 - 1 = 0.74$$

David



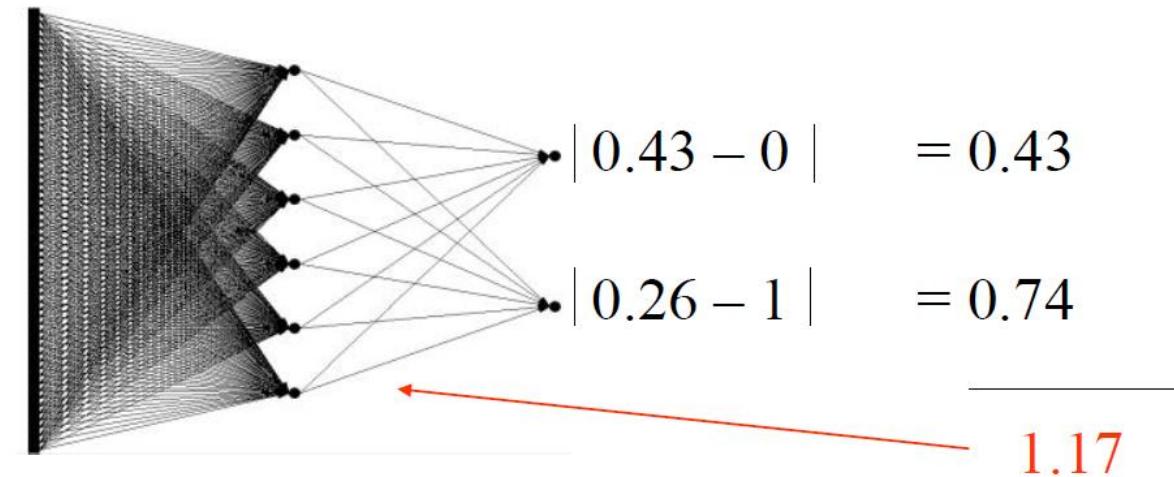
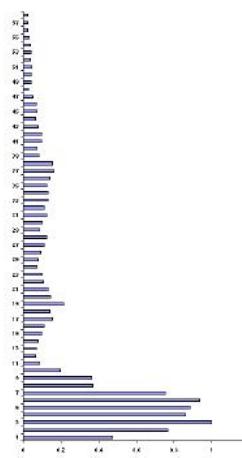
$$\bullet \quad 0.73 - 1 = 0.27$$

$$\bullet \quad 0.55 - 0 = 0.55$$

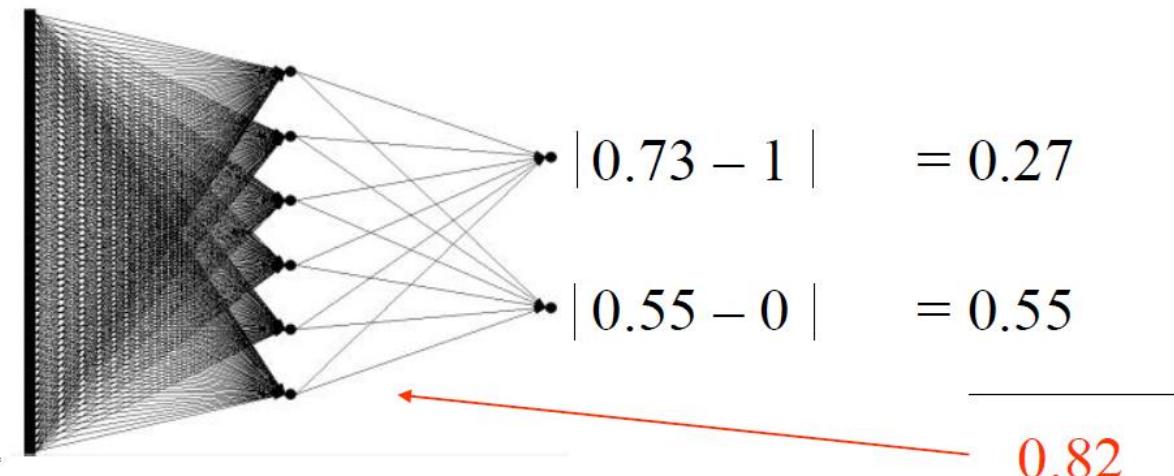
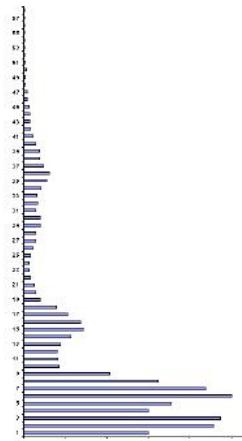
# Ejemplo: aprender a discriminar entre dos diferentes voces diciendo “hola”

Se propaga hacia atrás el error y se ajustan los pesos

Steve



David



# Ejemplo: aprender a discriminar entre dos diferentes voces diciendo “hola”

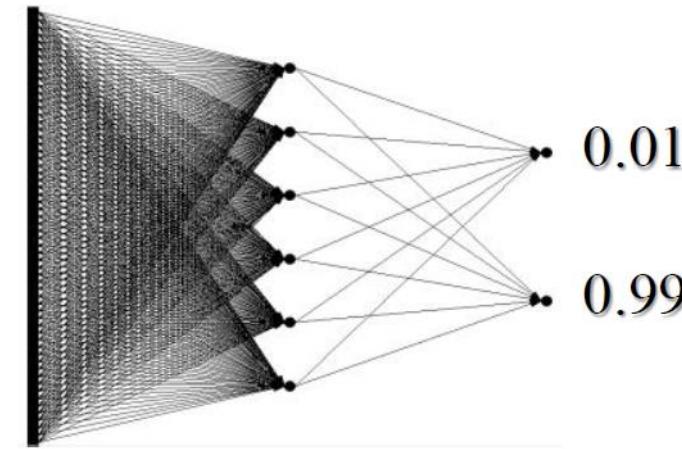
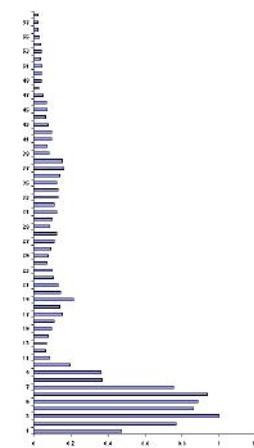
El proceso se repite para todos los ejemplos de entrenamiento

- Se presentan los datos
- Se calcula el error
- El error se propaga hacia atrás
- Se ajustan los pesos

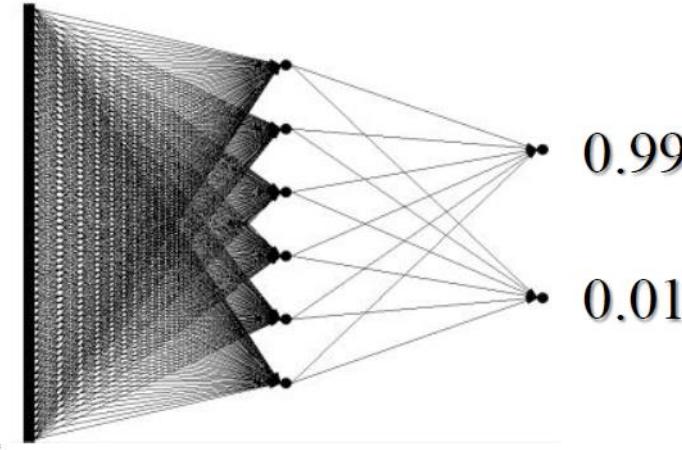
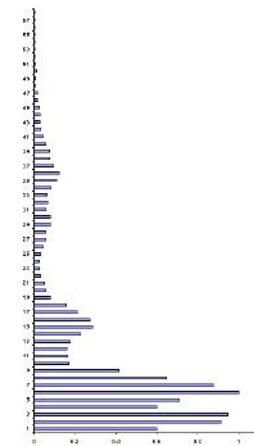
Ejemplo: aprender a discriminar entre dos diferentes voces diciendo “hola”

Una vez la red haya sido entrenada, provee resultados correctos

Steve



David



En las **redes recurrentes** la información fluye de forma bidireccional

En redes Feed forward

- Información fluye en una sola dirección
- Un patrón de entrada produce una salida
- No hay memoria del estado previo

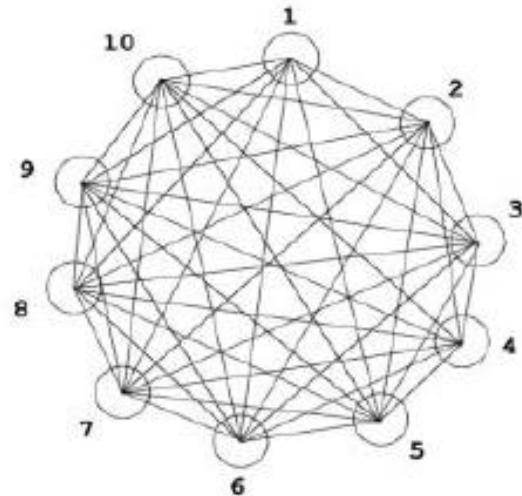
Recurrencia

- Nodos se conectan con la capa previa o consigo mismo
- Existe memoria del estado(s) previo(s)

En redes biológicas existen ambas formas de interconexiones

# Las redes de Hopfield son un subtipo de redes recurrentes

## Redes de Hopfield



- Completamente recurrente
- Los pesos son simétricos  $w_{i,j} = w_{j,i}$
- Los nodos pueden estar activos o inactivos
- La actualización es aleatoria

Hebb rule: los nodos que se activan juntos, se conectan

Puede acceder a una memoria si se le presenta un ejemplo parcial o corrupto

# ¿Por qué funcionan las redes neuronales artificiales?

**Son aproximadores universales**

Una red feedforward puede aproximar cualquier función continua no-lineal

**Aprendizaje y adaptación**

Pueden generalizar correctamente y se adaptan en línea

**Procesamiento paralelo**

Hardware especializado para esta tarea

**Sistema multivariable**

ANN manejan de forma natural múltiples entradas y salidas

Pero no todo es rosa, las ANN tienen  
varios problemas de implementación

**Escoger la arquitectura no es una tarea trivial**  
Cuantas capas, cuantas neuronas, como se inicializan

**Existencia de múltiples óptimos locales**  
Es improbable que se halle el optimo global durante el entrenamiento

**No tiene un fundamento matemático sólido**  
A diferencia de SVM

**Su capacidad de interpretación es muy limitada**  
Es una caja negra que funciona, pero no es interpretable

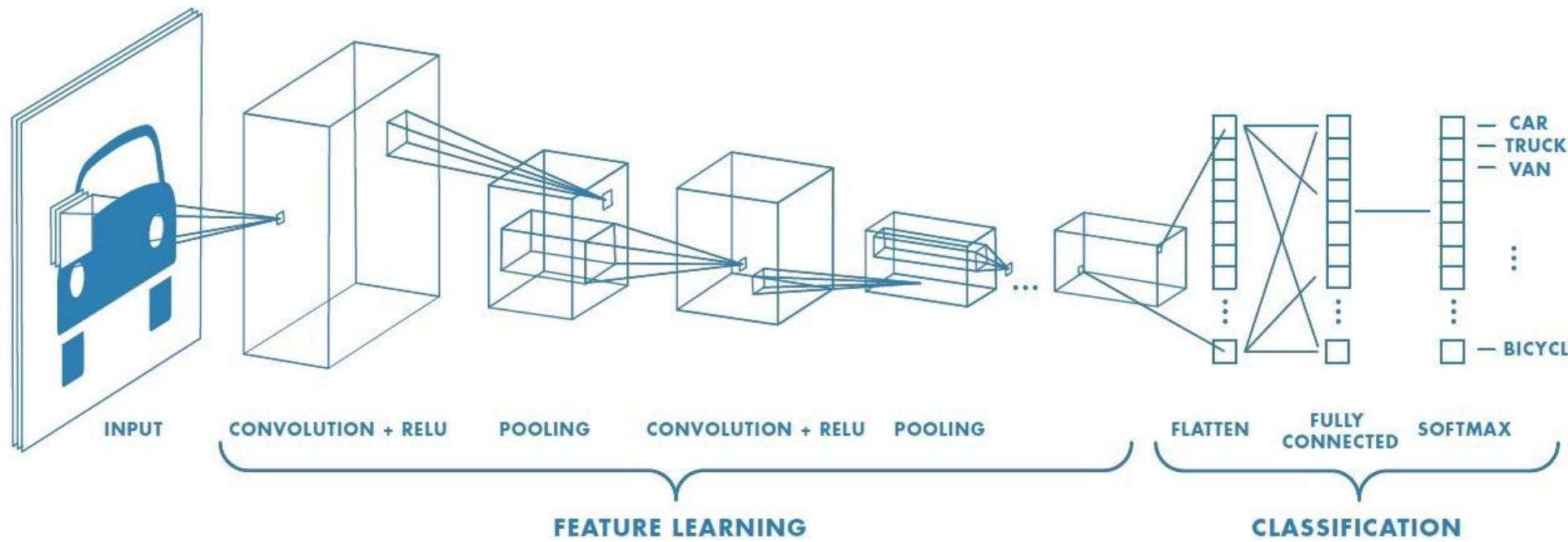
# Nuevos avances en optimización y ciencias de la computación propiciaron el nacimiento de **Deep Learning**

Deep Learning nace de las ANN, pero con nuevas características

- Resuelve el problema del “vanishing gradient” (cuando se tienen varias capas y los pesos no se actualizan)
- Mejoras en el algoritmo de gradient descent
- Presenta nuevas arquitecturas: convolutional NN (CNN), long short term memory (LSTM), generative adversary (GAN)

Debido a estas mejoras las redes pueden tener muchos niveles de capas internas

**CNN** utiliza varias capas internas que ejecutan convoluciones y filtros a los datos de entrada



Divide la entrada en patches mediante una técnica denominada sliding window

El problema con MLP cuando se agregan varias capas es que el número de parámetros crece exponencialmente

CNN reduce el número de parámetros necesarios

- Sus capas internas se auto-regulan
- Aprovecha las capacidades computacionales actuales (paralelización)

Muy utilizado en aplicaciones de procesamiento de imágenes y texto

- Lidia correctamente con la alta dimensionalidad
- Las entradas tienen mucha información local correlacionada

CNN tiene capas internas especializadas en tareas particulares para poder generar una salida

### Representación básica de una CNN

Entrada ————— Filtro local ————— Filtro volumen ————— Pooling ————— output



- E.j. imágenes (varias capas)
- Información relevante mayormente local (focalizada en regiones)

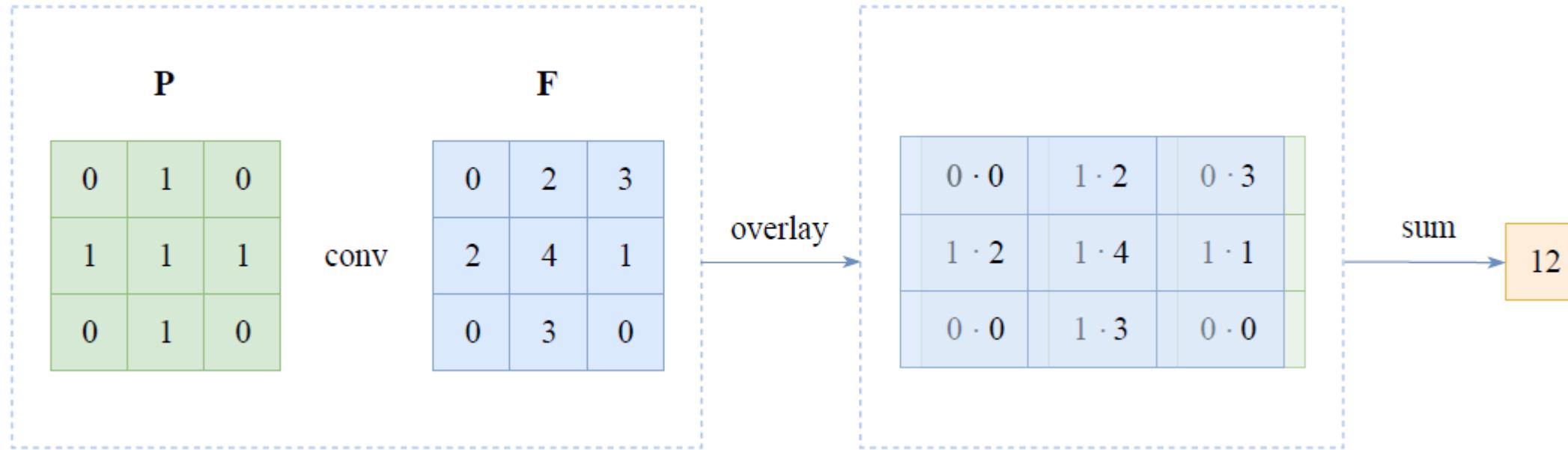
CNN tiene capas internas especializadas en tareas particulares para poder generar una salida

### Representación básica de una CNN

Entrada ————— Filtro local ————— Filtro volumen ————— Pooling ————— output



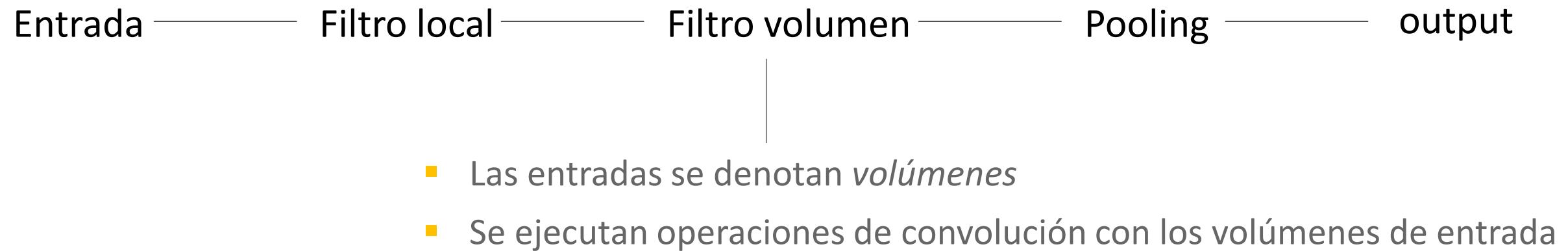
- La imagen es dividida en patches ( $p \times p$ )
- Por cada patch una FFN con una capa es aprendida
- Cada red aprende a identificar algún patrón e.j. color, forma, etc.
- Aprende una matriz F de tamaño ( $p \times p$ ) mediante backpropagation
- Se ejecuta una operación de **convolución**

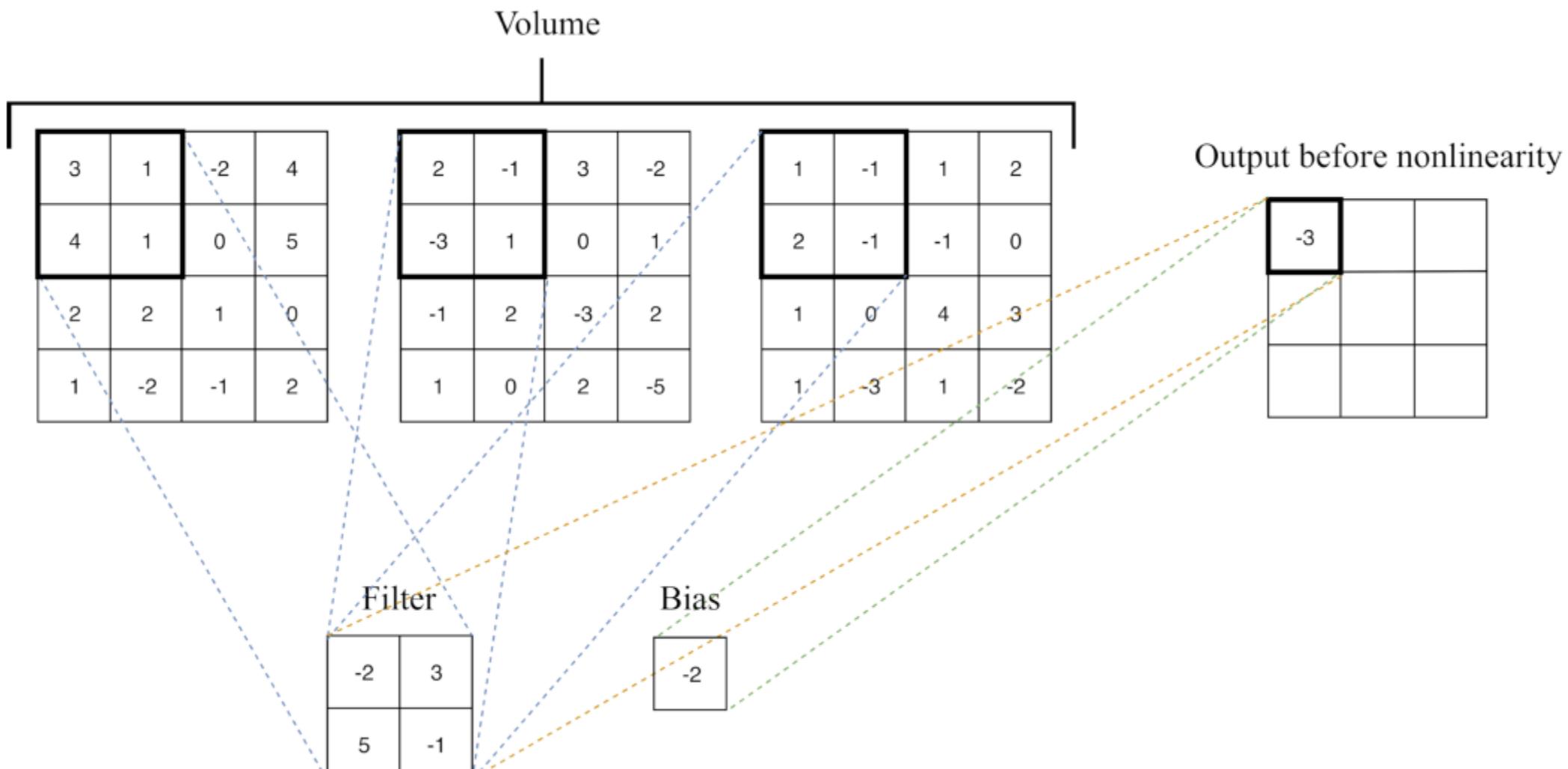


Convolución entre 2 matrices. Al resultado de la convolución se le aplica una función no lineal, e.j. ReLU

CNN tiene capas internas especializadas en tareas particulares para poder generar una salida

### Representación básica de una CNN





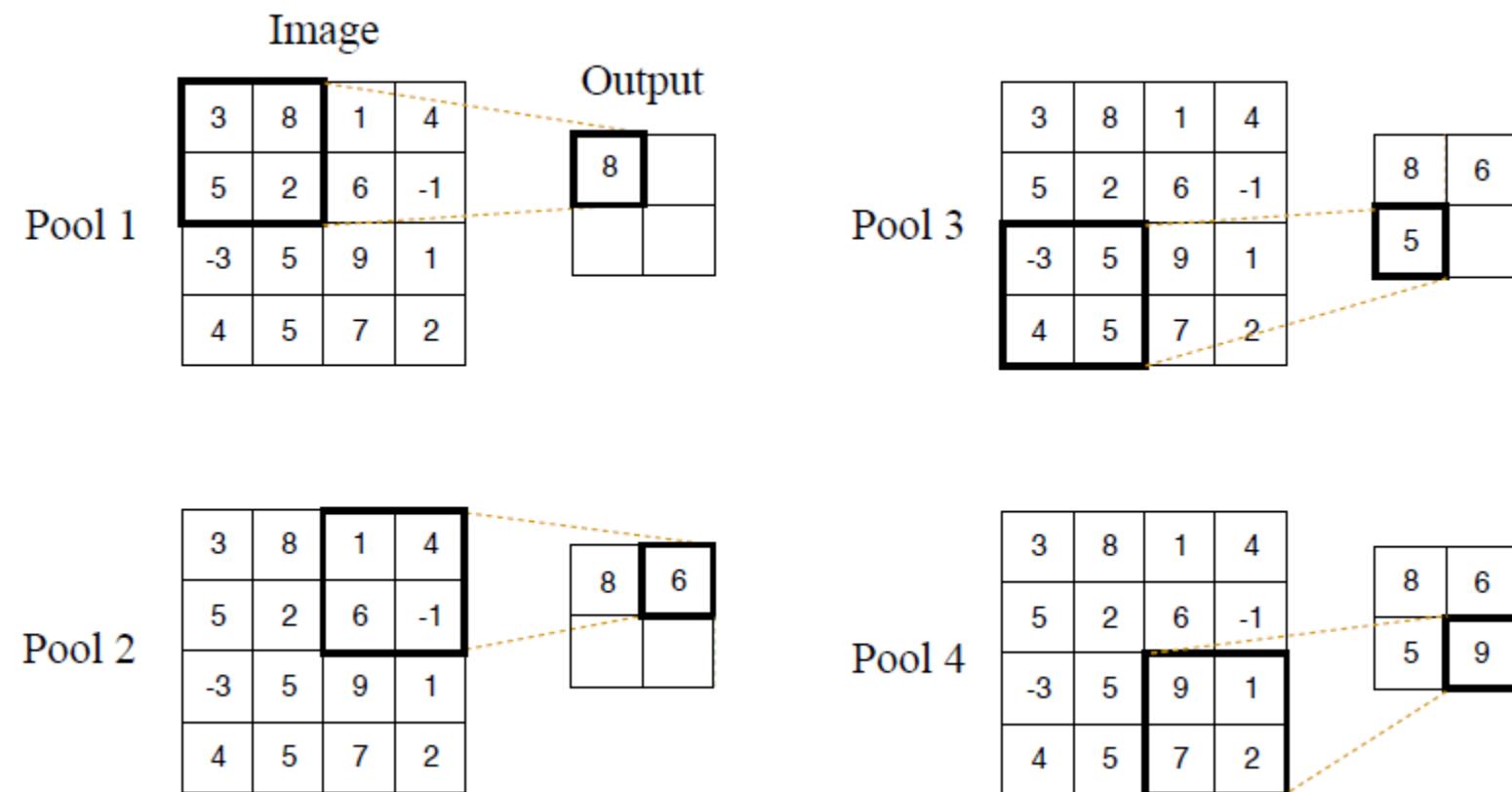
Convolución de un volumen consistente de tres matrices

CNN tiene capas internas especializadas en tareas particulares para poder generar una salida

### Representación básica de una CNN

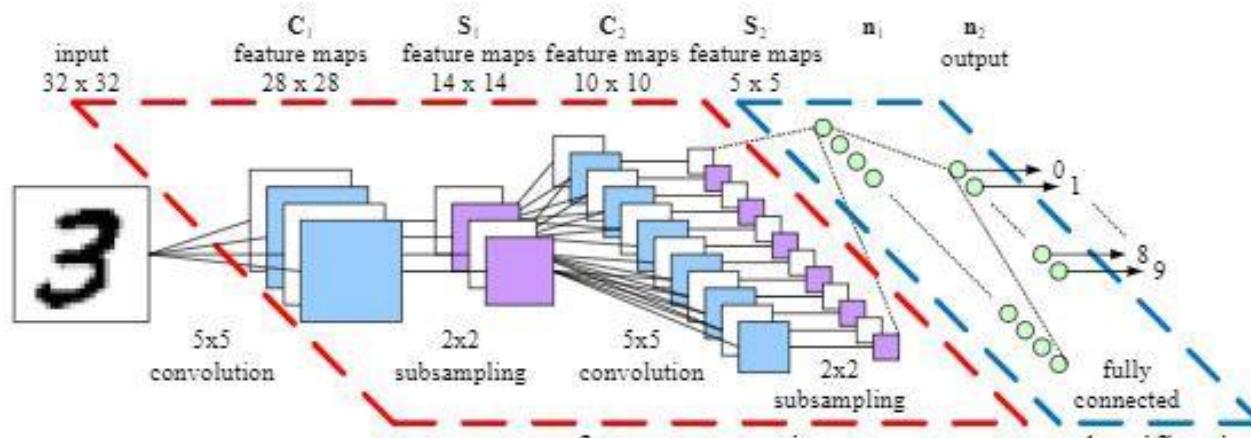
Entrada ————— Filtro local ————— Filtro volumen ————— Pooling ————— output

- 
- Sirve como un filtro aplicado a la salida de la capa anterior
  - Este filtro no se aprende, es un operador fijo: max o average
  - Se debe definir el tamaño del filtro y el stride (cantidad de elementos que se mueven en el moving window)
  - Mejora la velocidad del entrenamiento reduciendo el número de parámetros en la red



Pooling con tamaño de filtro 2 y stride 2

Se puede decir que una CNN tiene dos pasos bien diferenciados: feature extraction y classification



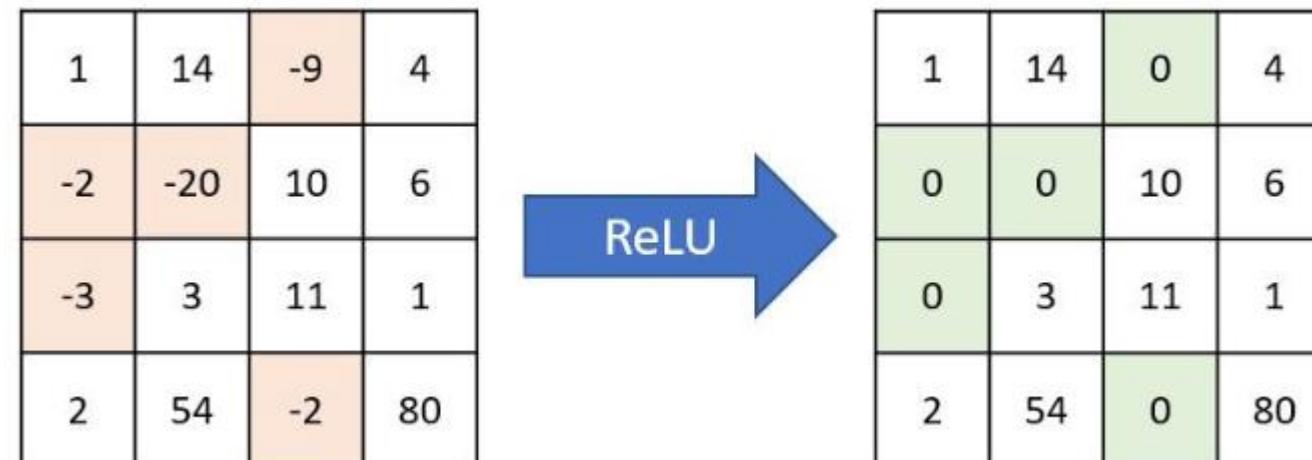
- Convolución es la operación más importante
- Como entrada se tiene e.j. una imagen. Salida: features
- E.j. si se analiza una foto de un gato, se podrían obtener el número de patas, ojos, cola, etc.
- Para el ejemplo de la figura, las características de salida serán curvas, rectas, etc.

# Rectified Linear Unit (ReLU) es la función de activación más utilizada en CNN

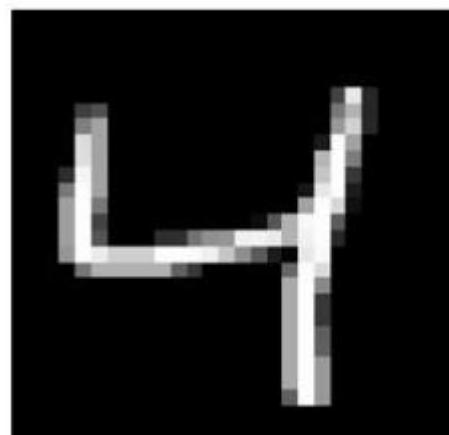
Las funciones de activación permiten clasificar datos que no son separables linealmente

ReLU convierte los datos negativos en 0, y los demás los deja igual

- Las operaciones de convolución generan múltiples imágenes de salida (feature maps)
- Se pueden aplicar más filtros para obtener más feature maps si es necesario

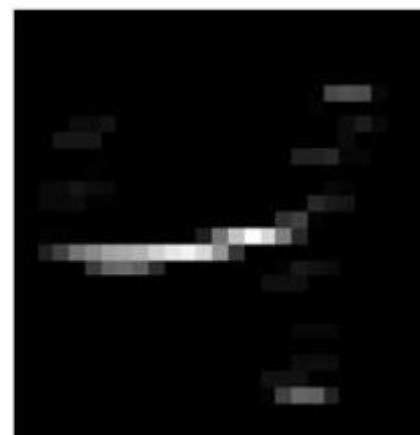


## Ejemplos de feature maps

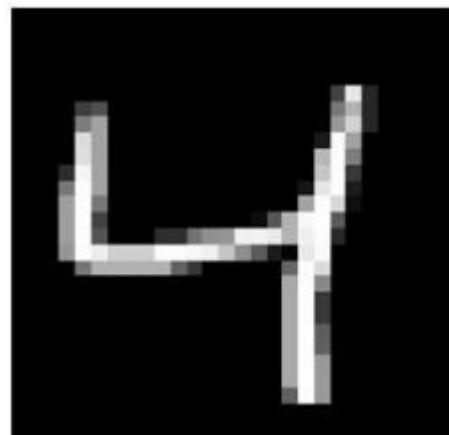


Image

\*   $\xrightarrow{\text{ReLU}}$  =  
Kernel

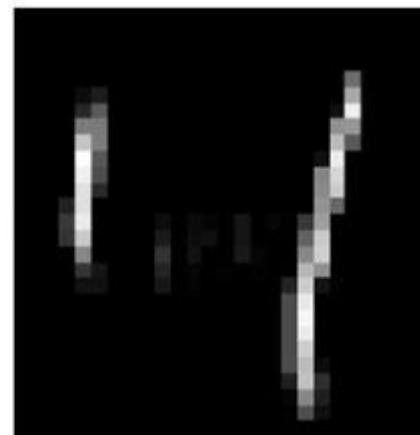


Output



Image

\*   $\xrightarrow{\text{ReLU}}$  =  
Kernel



Output

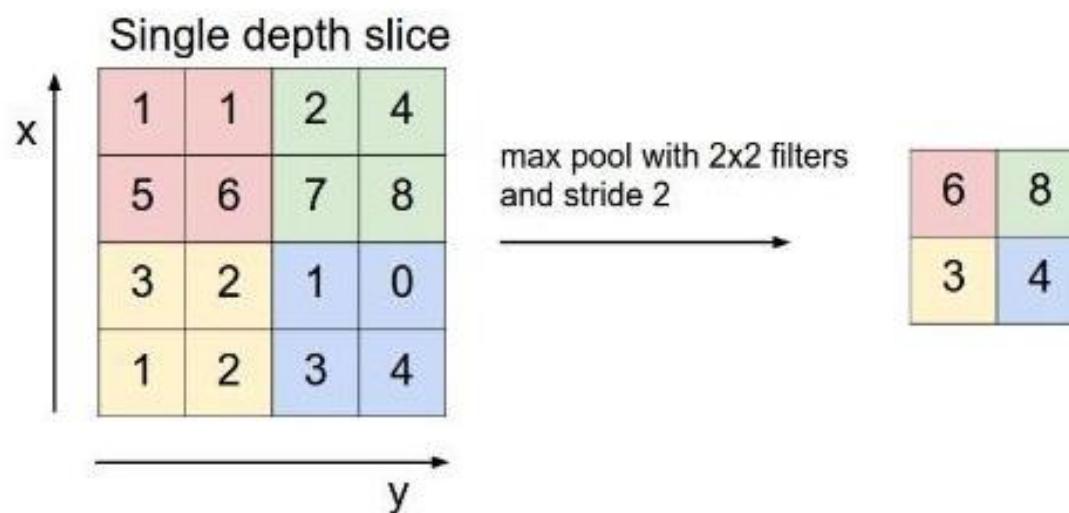
**Pooling** reduce el número de parámetros (mejora el tiempo de entrenamiento y evita el overfitting)

Después de una capa de convolución, es común agregar una de pooling

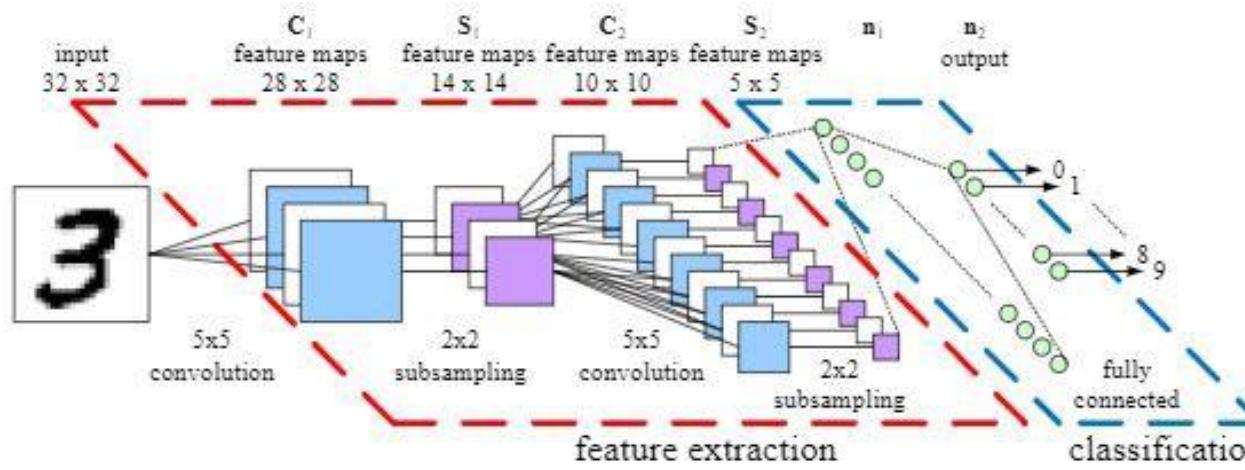
El método más utilizado es el de “max pooling”

- Toma el máximo valor de una ventana
- De esta manera se reducen las dimensiones del problema

Pooling reduce las dimensiones del problema, reteniendo información relevante



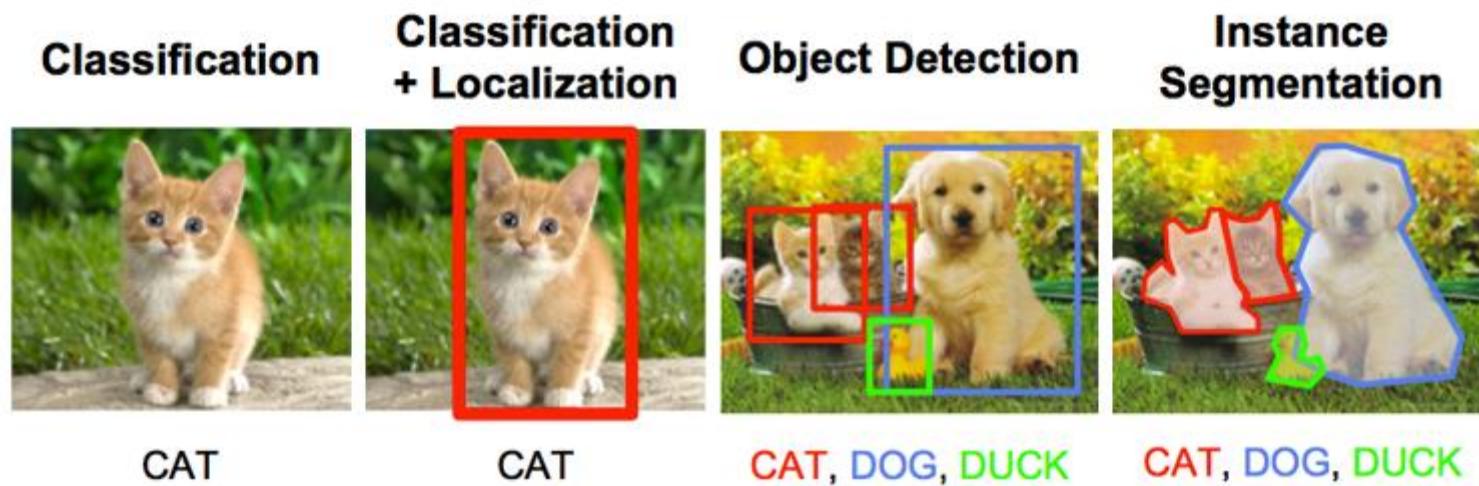
Se puede decir que una CNN tiene dos pasos bien diferenciados: feature extraction y classification



- Su entrada son las características
- Tipicamente utiliza un multilayer perceptrón para clasificación
- Produce una clasificación

# CNN son ampliamente utilizadas en análisis de imágenes

Tareas típicas en visión por computador

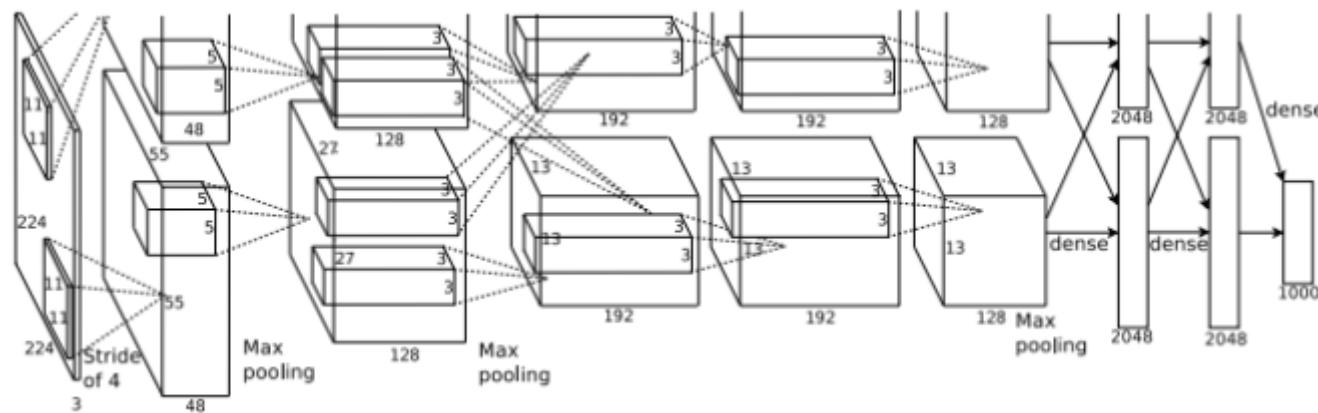


Existen muchas arquitecturas de CNN para atacar los problemas relacionados con imágenes

- AlexNet
- VGG Net
- GoogleNet
- Resnet
- RCNN
- YOLO
- ZFNET

# AlexNet

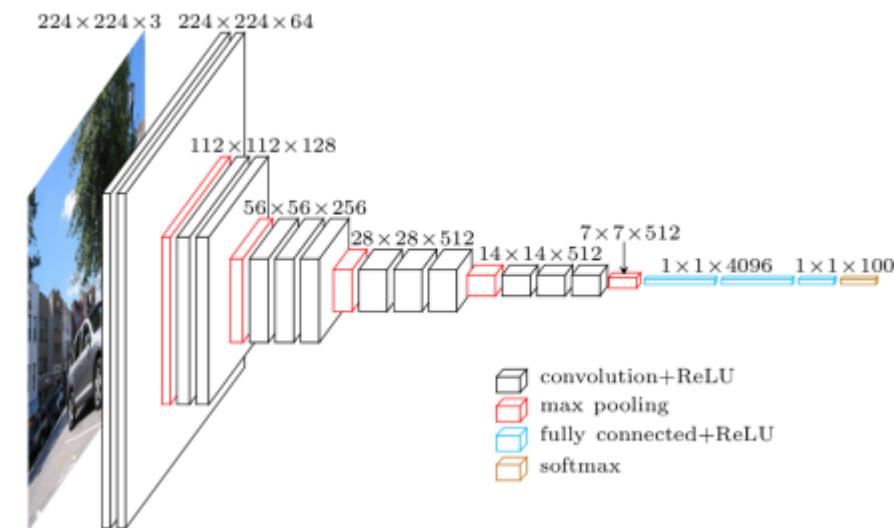
Modelo sencillo, pero altamente procesable por GPU's



# VGG Net Visual (Graphics Group at Oxford)

Ampliamente utilizada como benchmark

Existen muchos modelos pre-entrenados listos para usarse



# Investigar:

## Arquitecturas de CNN

- GoogleNet
- Resnet
- Yolo net
- RCNN
- ZFNET

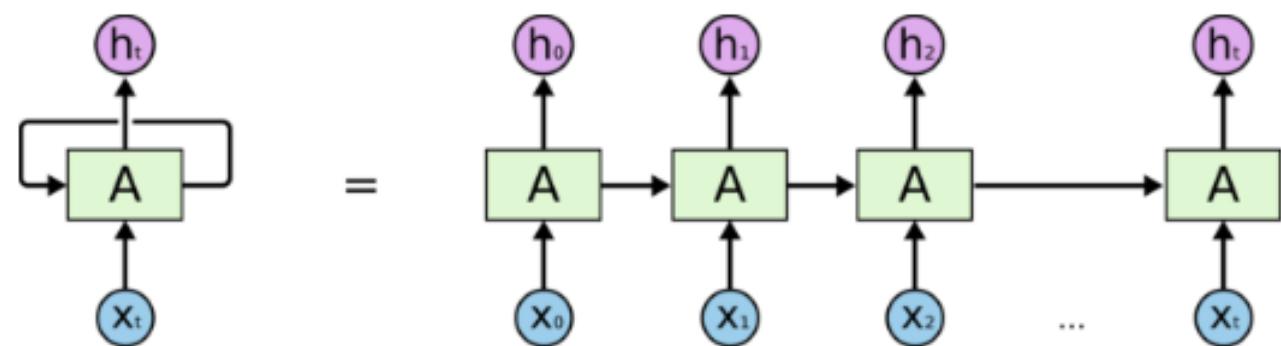
# Recurrent Neuronal Networks (RNN) es una generalización de una FFN que tiene memoria interna

Etiquetan, clasifican, y generan secuencias

- Entrada: matriz donde cada fila es una secuencia (el orden importa)
- Etiquetar: predecir una clase para cada feature vector
- Clasificar: predecir una clase para toda la secuencia de vectores
- Generar secuencias: generar otra secuencia (posiblemente de distinto tamaño que la entrada)

Muy utilizadas en procesamiento de texto y reconocimiento de voz (se componen de secuencias y su orden importa)

Una RNN contiene loops: cada unidad  $u$  de la capa  $l$  tiene un estado  $h_{l,u} \in R$  (memoria de la unidad)

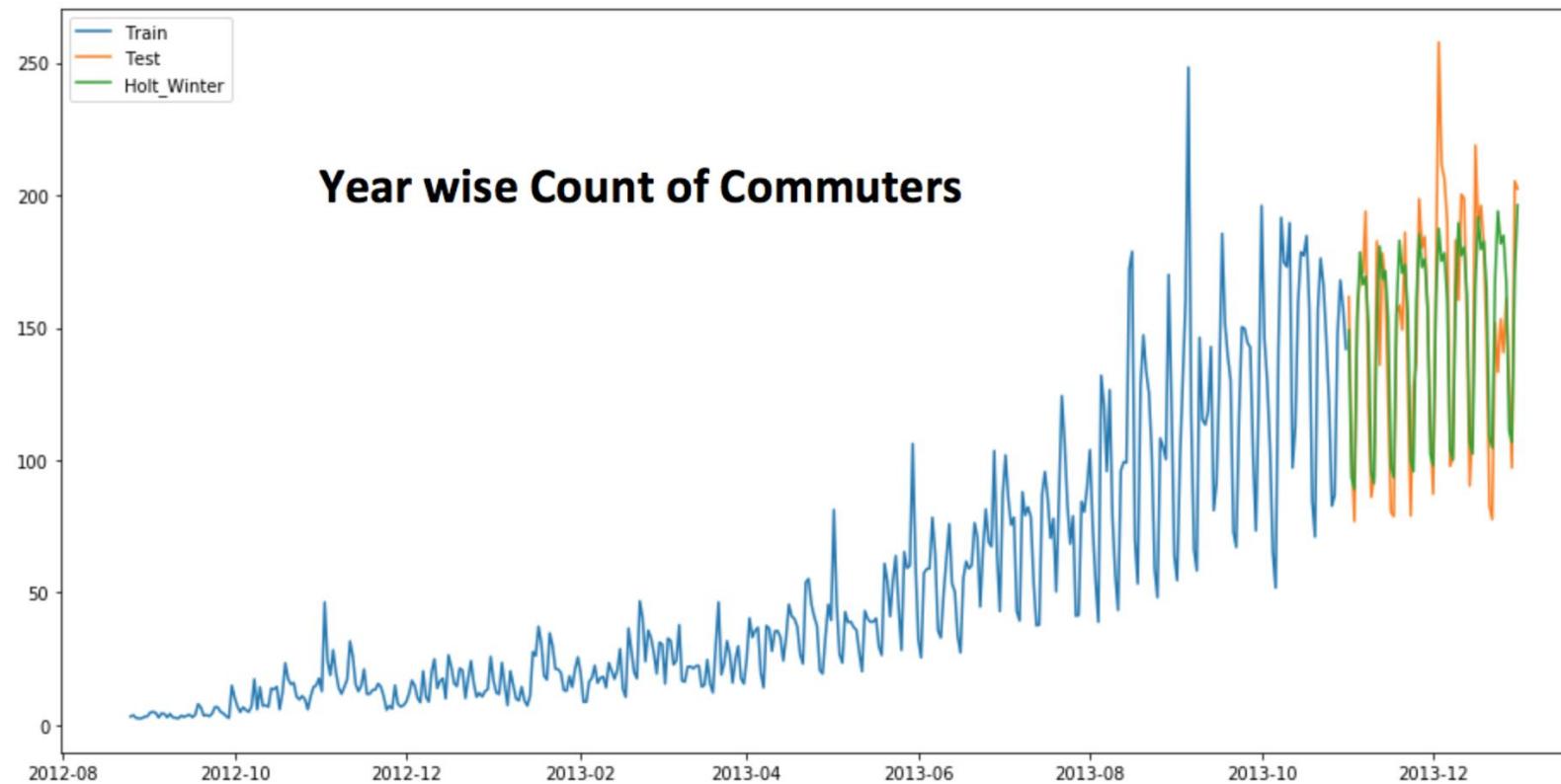


An unrolled recurrent neural network.

Una unidad recibe como entradas:

- Vector de estados de la capa previa  $l - 1$
- Vector de estados de la capa actual  $l$
- La secuencia actual  $x_i$

Intermezzo: una **serie de tiempo** es una secuencia de valores, típicamente, espaciadas de forma homogénea



Serie de tiempo  $y(t) \in R, t_i - t_{i-1} = \text{timestamp}$

Un timestamp puede ser anual, mensual, 15 minutos, 1 segundo, etc.

# La secuencialidad en la red viene determinada por timestamps (series de tiempo)

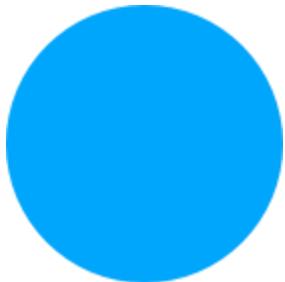
En RNN un timestamp  $t$  no representa tiempo, pero sí orden

El índice  $t$  denota el timestamp. Para calcular  $h_{l,u}^t$

- Se calcula la combinación lineal del input feature vector con el vector de estados  $h_{l,u}^{t-1}$
- La combinación se calcula mediante dos vectores  $w_{l,u}$ ,  $u_{l,u}$ , y un parámetro  $b_{l,u}$
- Esta combinación pasa por una función de activación no lineal  $g_1()$  – típicamente  $\tanh$
- La salida  $y_l^t$  se calcula usando una función de activación no lineal  $g_2()$  – típicamente  $softmax$
- Esta función se aplica a la combinación lineal de  $h_{l,u}^t$  usando una matriz  $V_l$  y un vector  $c_{l,u}$
- Los valores de  $w_{l,u}$ ,  $u_{l,u}$ ,  $b_{l,u}$ ,  $V_{l,u}$  y  $c_{l,u}$  son calculados usando gradiente descent mediante backpropagation

RNN intentan predecir una secuencia  
basado en otra secuencia

Se necesita predecir la dirección que tomará una circunferencia



Sin más información, podría ir hacia cualquier dirección

RNN intentan predecir una secuencia  
basado en otra secuencia

Pero si se tiene información previa



Muy probablemente, se moverá hacia la derecha

Ejemplo: Los humanos trabajan también  
prediciendo secuencias

Tarea: dada una letra del alfabeto, completar la secuencia

F

Secuencia generada:

F, G, H, I, ..., Z

Intuitivamente, para generar la secuencia, el cerebro piensa en las letras  
anteriores a F

Ejemplo: se requiere programar  
un chatbot que ejecute tareas

Tarea: dada una orden, clasificar la intención

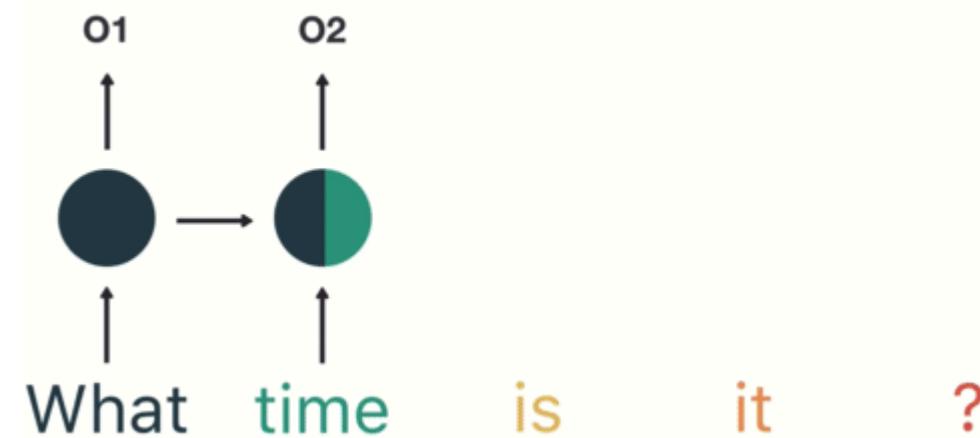
What time is it?

Esta entrada puede partirse en secuencias de palabras

What – time – is – it ?

Ejemplo: se requiere programar  
un chatbot que ejecute tareas

En cada timestamp, se presenta una palabra a la RNN. Se calculan las salidas en base a los estados previos de las unidades



El problema del **vanishing gradient** también es visible en este tipo de redes



En redes profundas, la actualización de los pesos de las unidades más lejanas se produce muy lentamente; o, en muchos casos, no se produce la actualización. A este efecto se lo denomina vanishing gradient

Cuando se dá este fenómeno, no existe aprendizaje

Este problema hace que las RNN “olviden” los estados de las primeras secuencias que se le presentaron

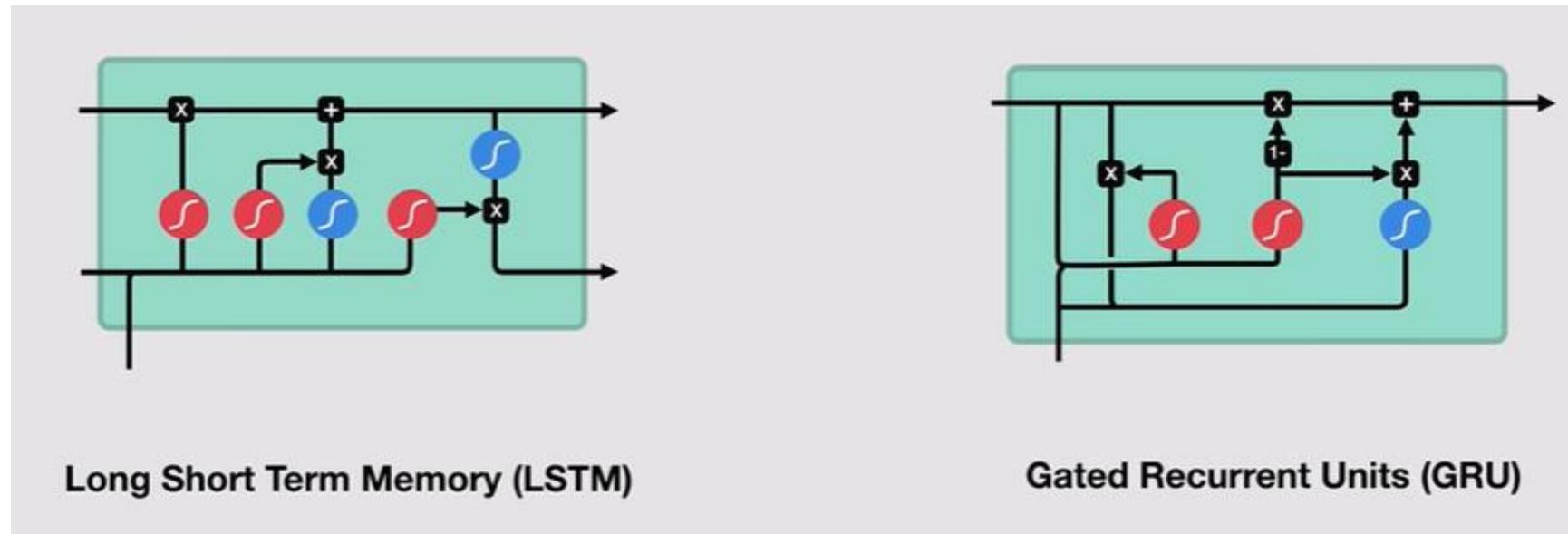


Volviendo al ejemplo de la secuencia de palabras, las palabras más actuales tienen mucho más peso en el modelo

Debido al “vanishing gradient”, las secuencias iniciales tiene muy poco peso en el modelo

Se dice que el modelo, olvidó las primeras secuencias

Para solventar los problemas del “vanishing gradient” se crearon los **gated RNNs**

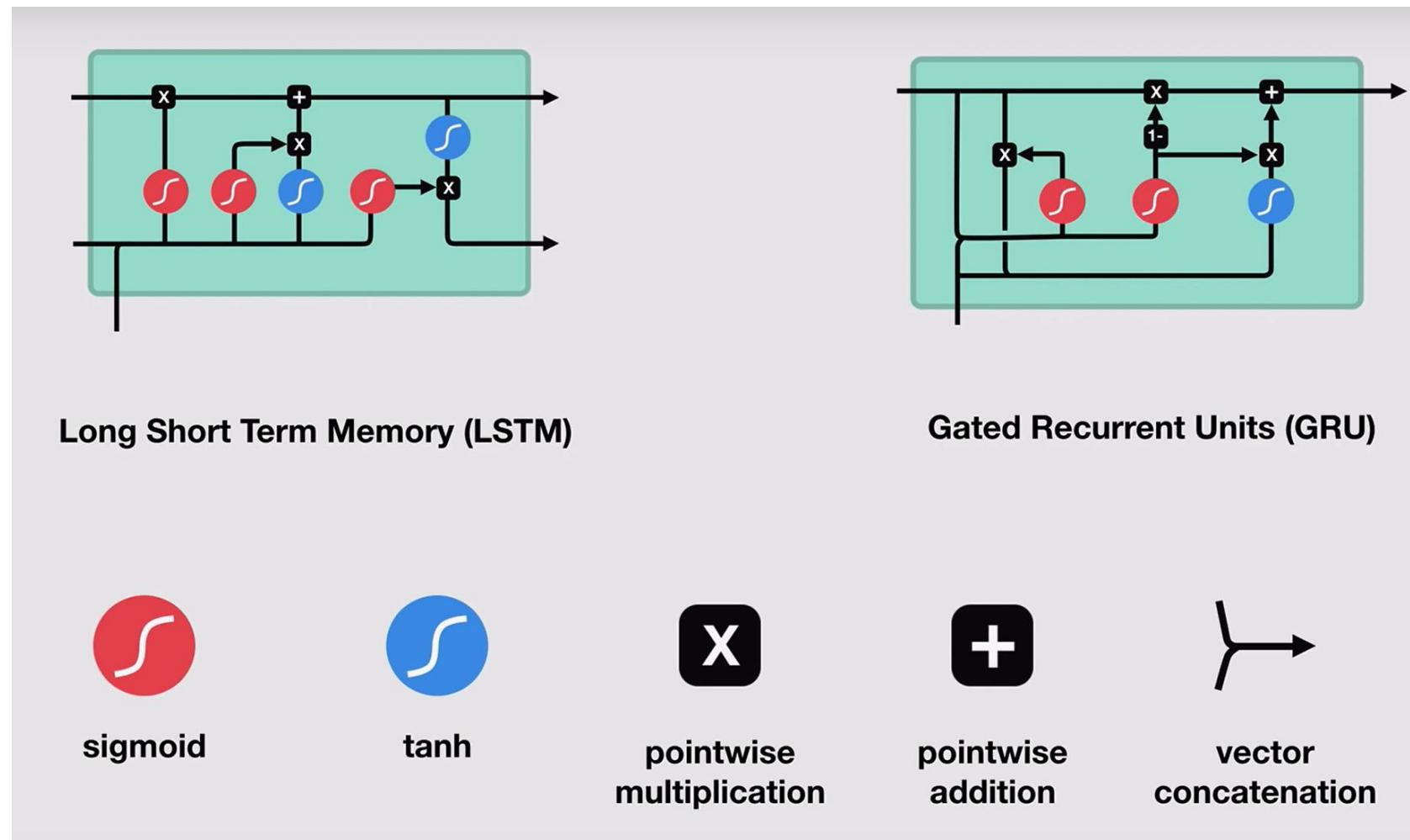


LSTM y GRU son RNN que implementan compuertas para seleccionar que recordar y que olvidar

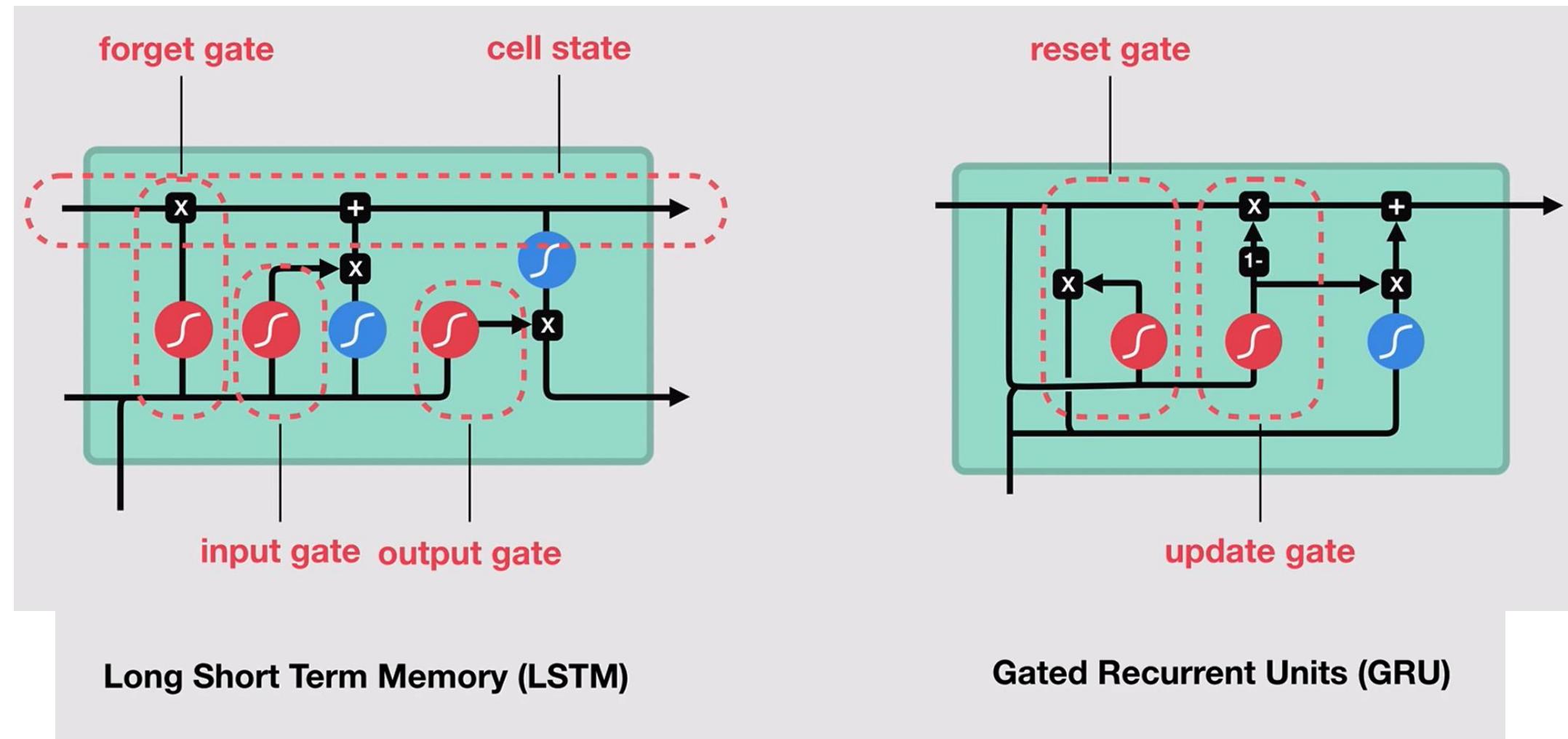
GRU es una arquitectura posterior a LSTM

Ambas son ampliamente utilizadas en la actualidad

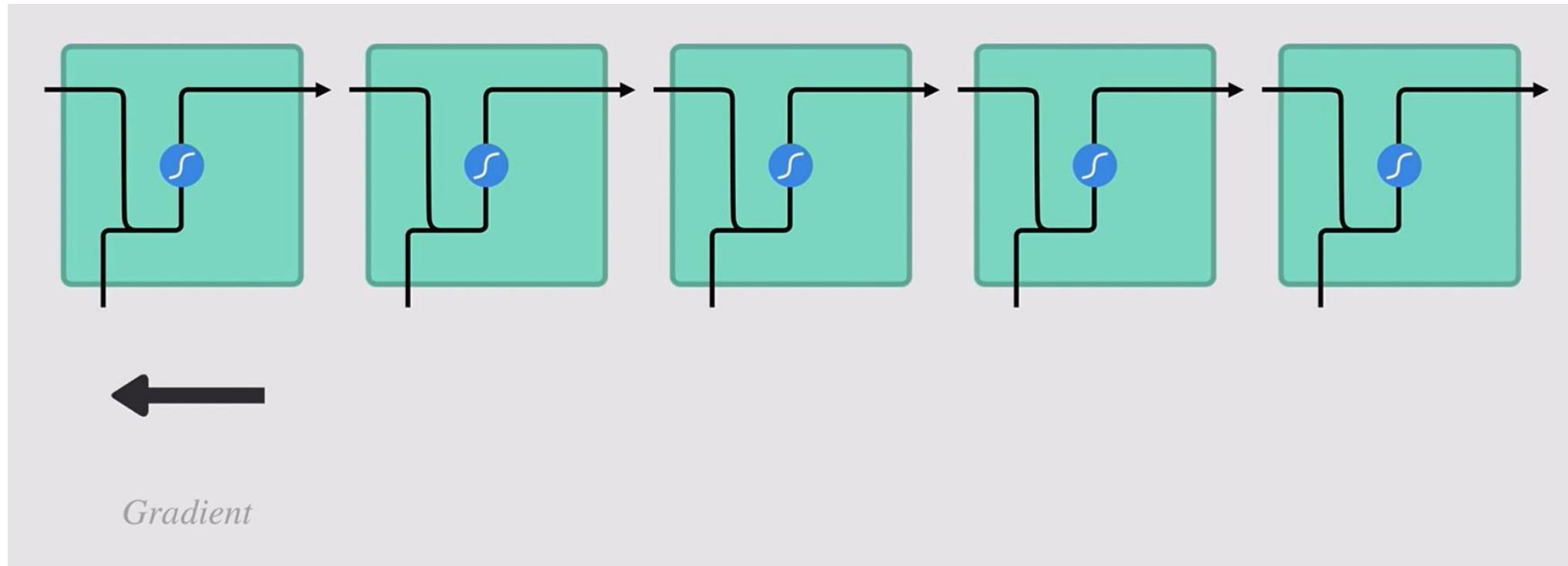
Gated RNNs tienen mecanismos de compuertas y activaciones no lineales para seleccionar lo que se debe recordar



Gated RNNs tienen mecanismos de compuertas y activaciones no lineales para seleccionar lo que se debe recordar



Las RNN olvidan secuencias antiguas  
debido al vanishing gradient problem



Cuando se tiene un conjunto de secuencias, no todas tienen la misma importancia para determinar el contexto

Por ejemplo, la siguiente revisión de un producto puede ser analizada en términos de importancia de palabras

September 2018

Verified Purchase

**Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!**

Cuando se tiene un conjunto de secuencias, no todas tienen la misma importancia para determinar el contexto

Por ejemplo, la siguiente revisión de un producto puede ser analizada en términos de importancia de palabras

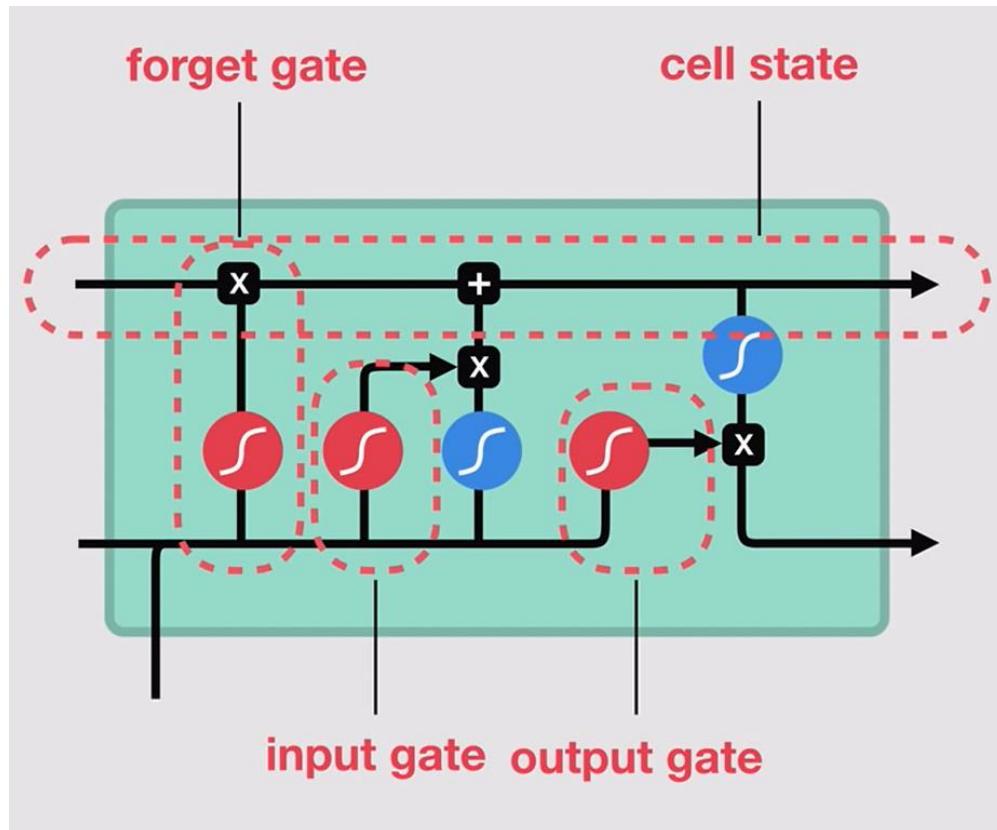
September 2018

Verified Purchase

**Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!**

No todas las palabras tienen la misma importancia

# LSTM procesa datos pasando información entre unidades y propagándolas



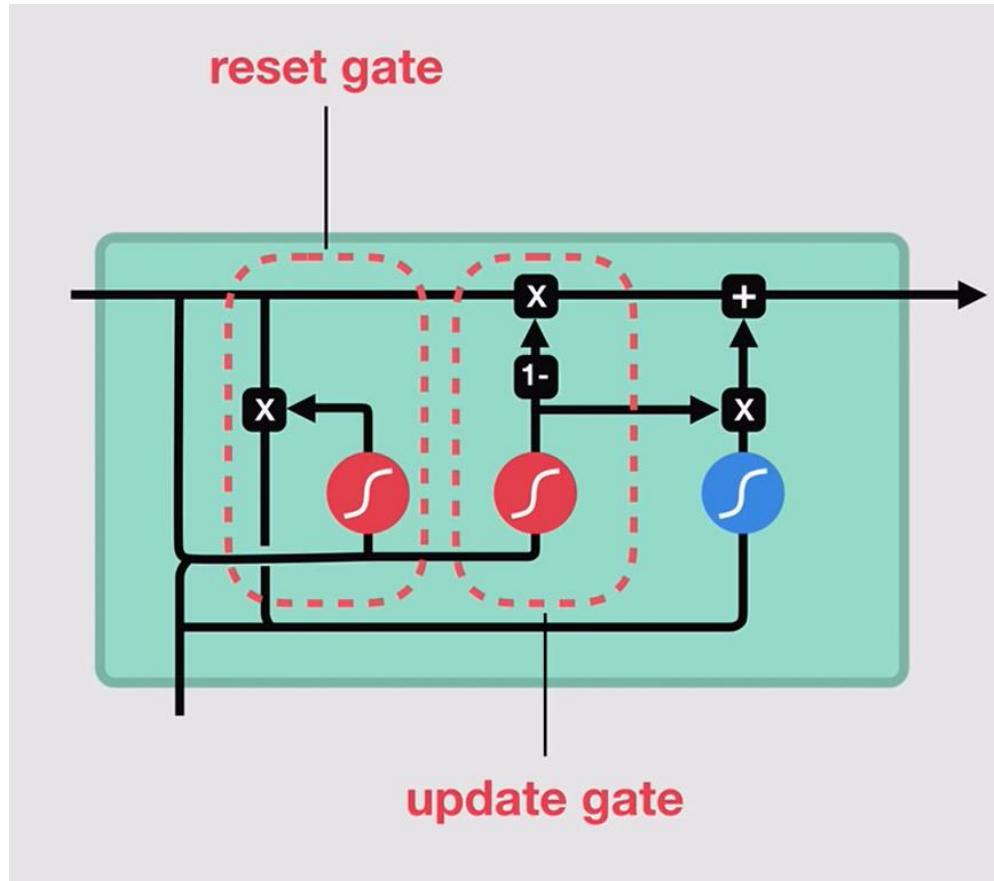
Cell state: transporta la información i.e. memoria (solo transporta lo relevante)

Forget gate: decide que información es relevante del pasado

Input gate: decide que información para agregar a estado actual

Output gate: determina cuál será el próximo hidden state

# GRU es una nueva generación de RNN



Update gate: similar al forget gate de LSTM

Reset gate: decide cuanta información pasada olvidar

GRU tiene menos tensores; por ello, se entra más rápido

No hay un ganador entre GRU y LSTM -> “no free lunch”

# Todos hablan de la **inteligencia artificial** (aunque muchos no la entiendan)

Los mayores avances están en el área de ML y Deep Learning

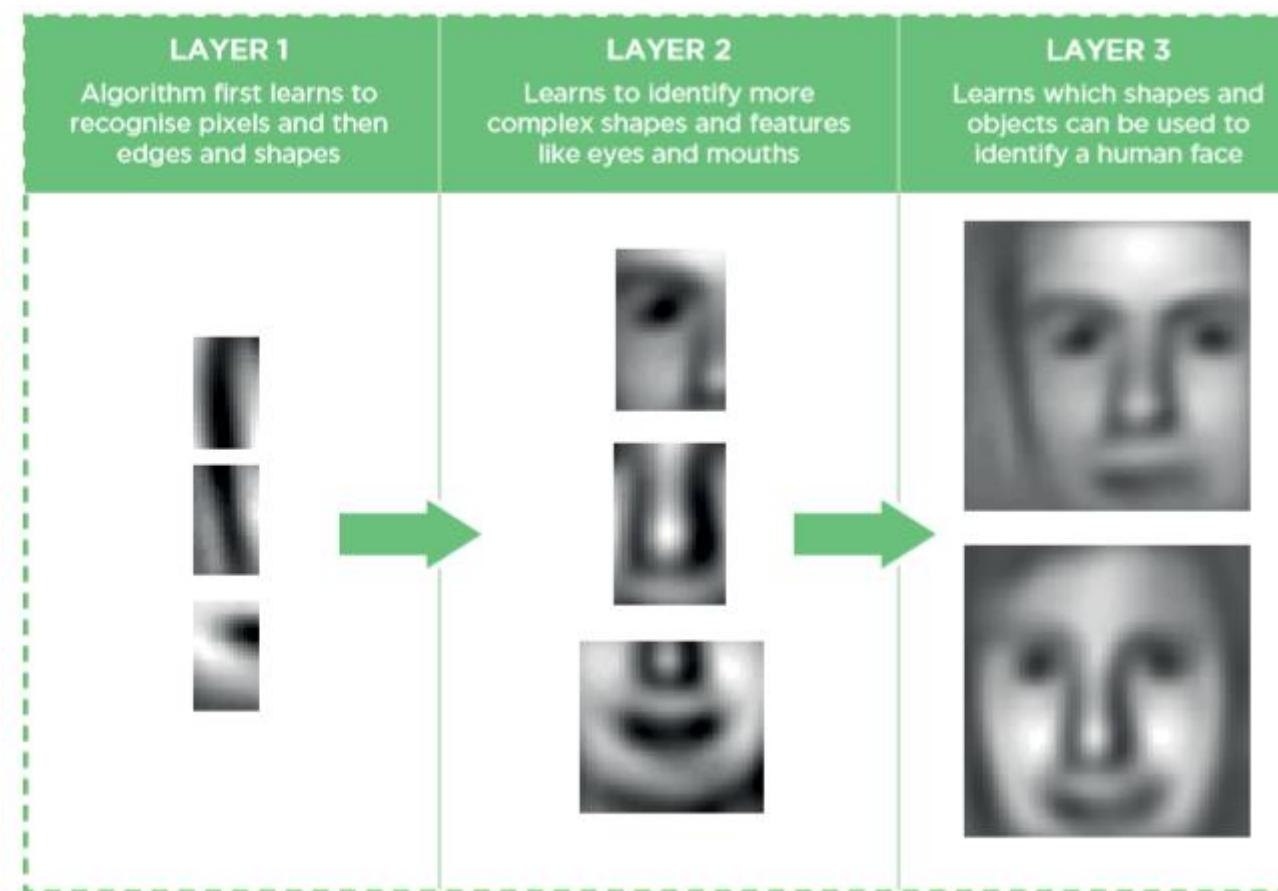


El interés en los últimos años para Deep Learning ha crecido de forma sostenida

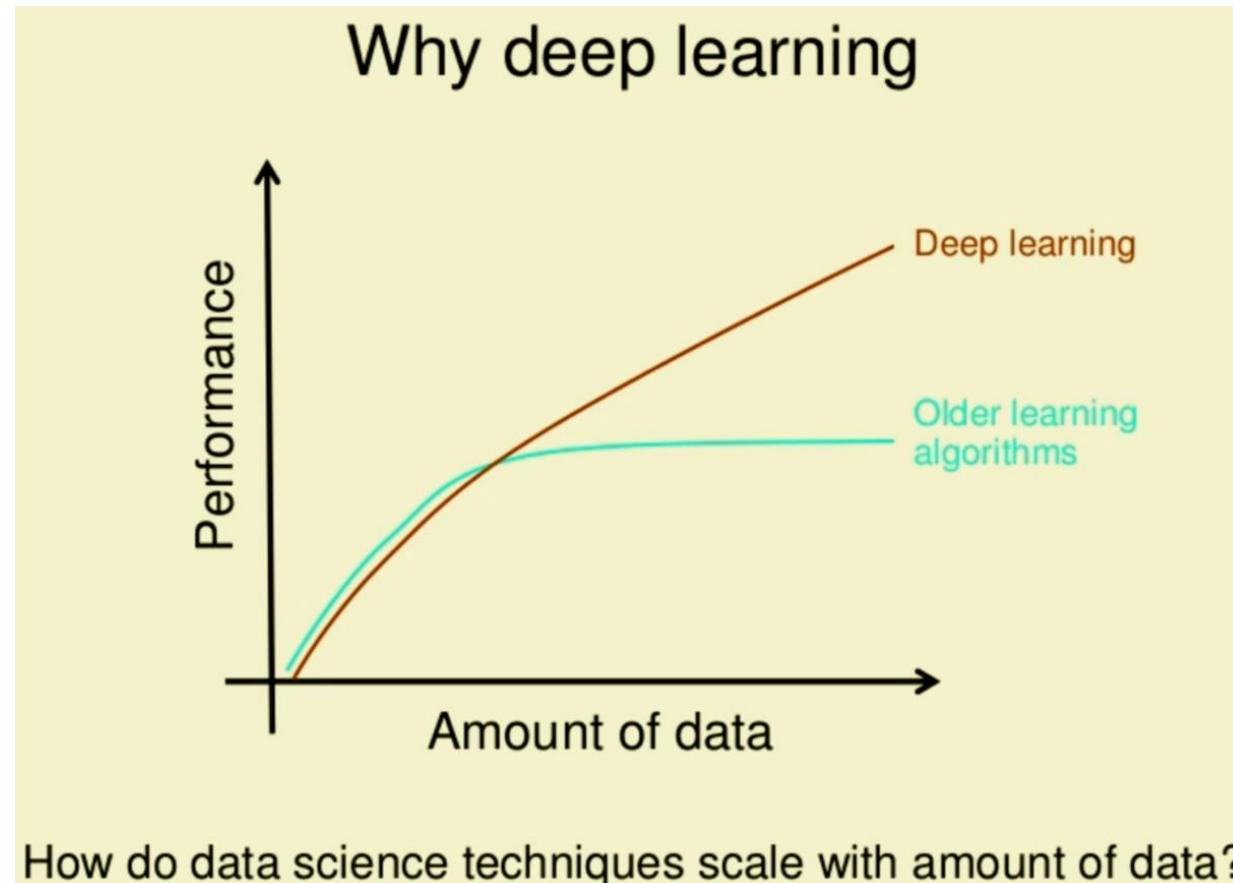
**Deep Learning** (revisited) es un subconjunto de ML que provee gran poder y flexibilidad al aprendizaje

DL representa el mundo como una jerarquía de conceptos

DL aprende características incrementalmente con cada capa



DL funciona si hay gran cantidad de datos... Hay abundancia de datos, pero no todos son de buena calidad



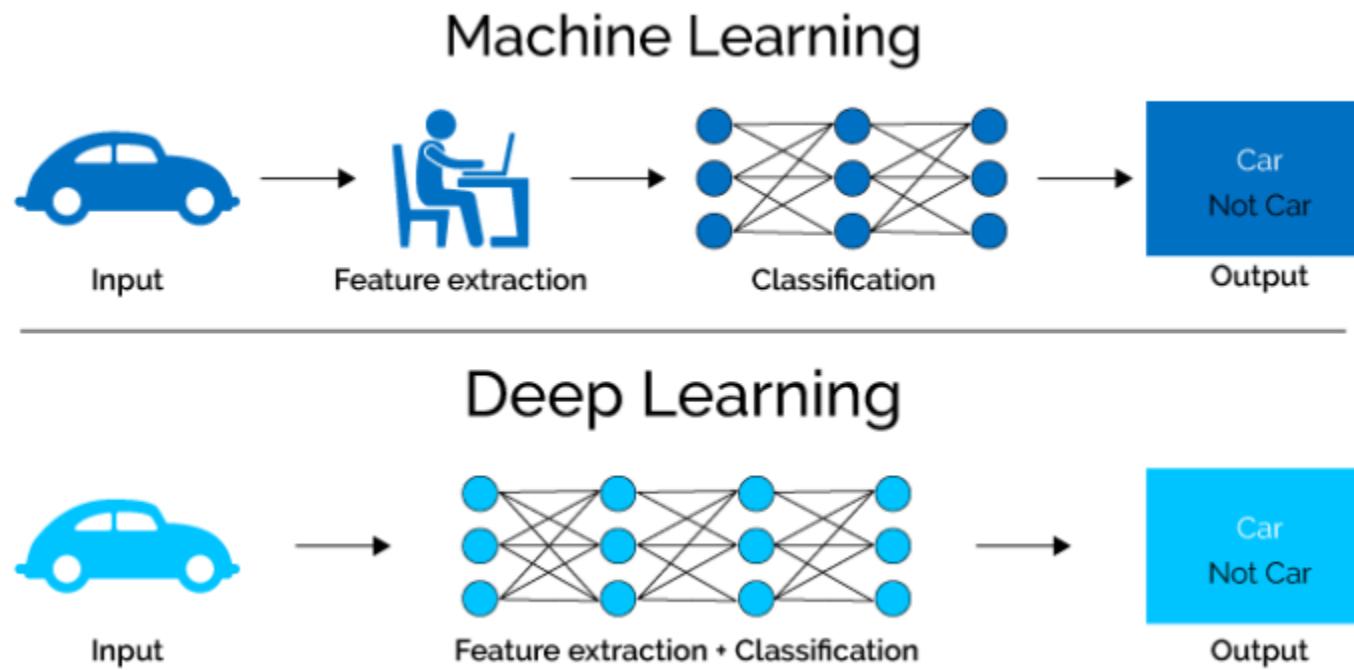
DL requiere alta capacidad de cómputo

Técnicas tradicionales pueden entrenarse en hardware de medianas prestaciones

DL y GPU's están muy ligados para hacer que el proceso de entrenamiento sea más rápido

DL puede requerir días para entrenar. ML normalmente requiere horas

DL puede aprender que características son importantes de forma automática



Algunas técnicas de ML automáticamente rankean features

DL lo hace de forma más eficiente

DL se debe preferir sobre técnicas tradicionales cuando existan datos suficientes para entrenar al modelo

Cuando no existen muchos datos  
ML tradicional tiene mejores resultados

Infraestructura  
DL requiere mucha capacidad de cómputo

Falta de conocimiento previo  
DL puede aprender características de forma automática

Clasificación de imágenes, NLP, reconocimiento de voz  
DL es el estándar actual de efectividad



**UNIVERSIDAD DE CUENCA**  
*desde 1867*

# Aprendizaje Supervisado

Andres Auquilla  
2020

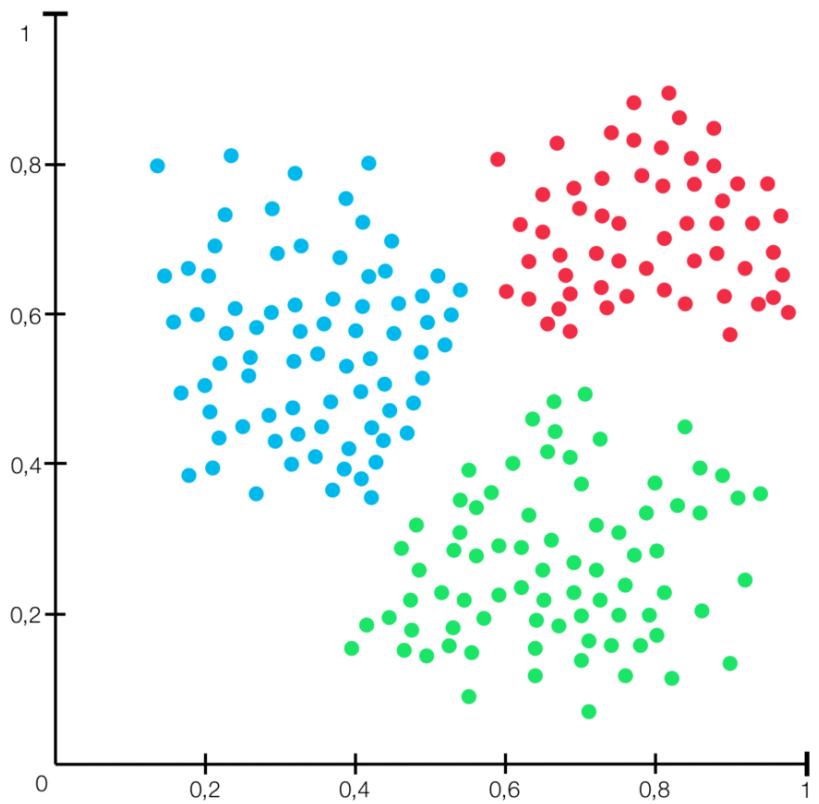


**UNIVERSIDAD DE CUENCA**  
*desde 1867*

# Aprendizaje No Supervisado

Andres Auquilla  
2020

# Content



**Variantes de clustering**

Tipos y características

**Evaluación de soluciones**

Determinar la calidad de una solución

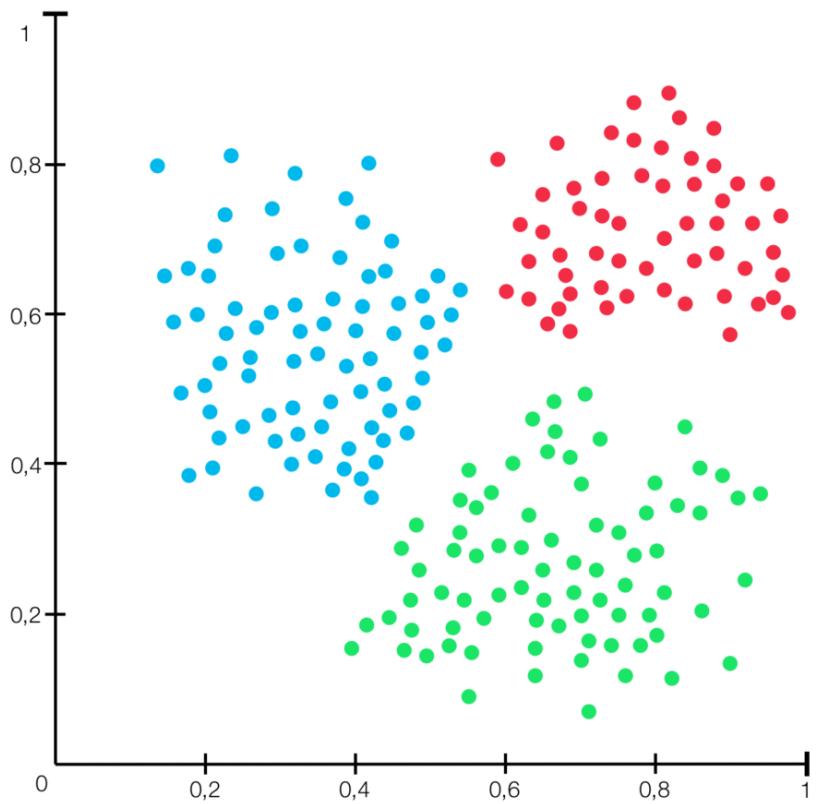
**Algoritmos de clustering**

K-means, jerárquicos, aglomerativos

**Reducción de dimensiones**

PCA, UMAP, codificadores

# Content



**Variantes de clustering**

**Tipos y características**

**Evaluación de soluciones**

Determinar la calidad de una solución

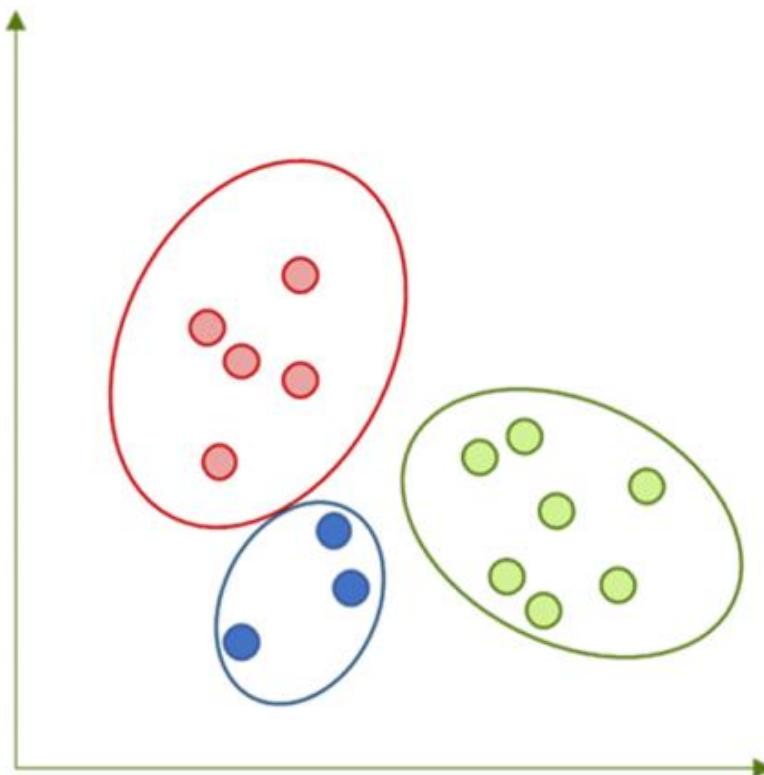
**Algoritmos de clustering**

K-means, jerárquicos, aglomerativos

**Reducción de dimensiones**

PCA, UMAP, codificadores

# Clustering es encontrar grupos (clusters) de instancias similares



Encontrar grupos de instancias que:

- Las instancias dentro del grupo sean similares
- Las instancias en grupos distintos sean diferentes

Comparado a una clasificación:

- Similaridad: asignar clases a los ejemplos
- Diferencia: las clases no son conocidas de antemano → aprendizaje no supervisado
- Estas clases son “inventadas”

Clustering se puede aplicar a problemas en donde no se conozcan las clases y exista una medida de similaridad

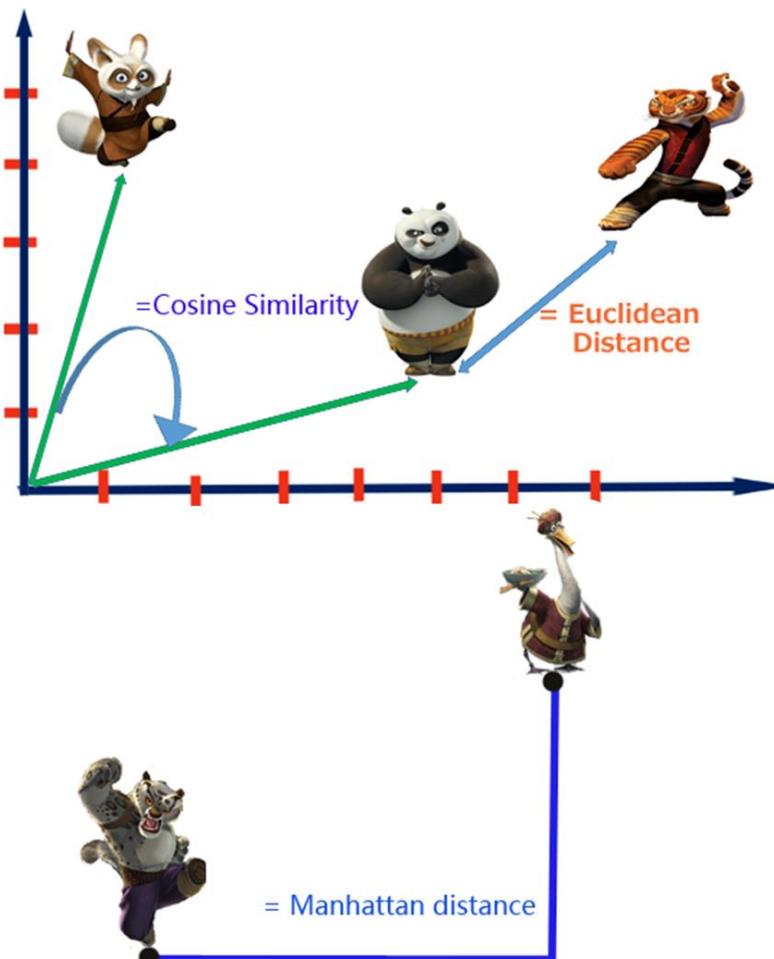
Ejemplo: Se requiere construir una taxonomía de animales

- Se puede utilizar un clustering jerárquico
- Cada nivel puede ser considerado una clase

Ejemplo: Problemas de marketing

- Identificar clientes típicos
- Producir productos orientados a un cierto grupo de clientes

Para poder comparar instancias y determinar su similaridad, se necesitan medidas



### Medidas basadas en distancia

- Distancia euclidiana
- Distancia Manhattan
- Distancia Hamming (para datos nominales)
- Coseno
- ...

### Formas más generales de similaridad

- No necesariamente satisfacen inequidades, simetría, etc.

Pueden existir distintas variantes de clusters

**Disjoint versus overlapping**

Una instancia pertenece a un cluster solamente

**Exhaustive versus partial**

Una instancia podría o no pertenecer a un cluster

**Flat versus hierarchical**

Diferentes niveles jerárquicos

**Extensional versus intensional**

Condiciones que una instancia debe cumplir

# Disjoint clusters versus overlapping clusters

Disjoint = una instancia puede pertenecer a un solo cluster

- Se da en la mayoría de definiciones de clustering
- Esta restricción podría ser relajada (overlapping)

Overlapping = una instancia puede pertenecer a más de un cluster

- También llamada subgroup Discovery
- También se puede interpretar como áreas de clusters que se intersectan

# Exhaustive versus partial clustering

Exhaustive = una instancia debe pertenecer a algún cluster

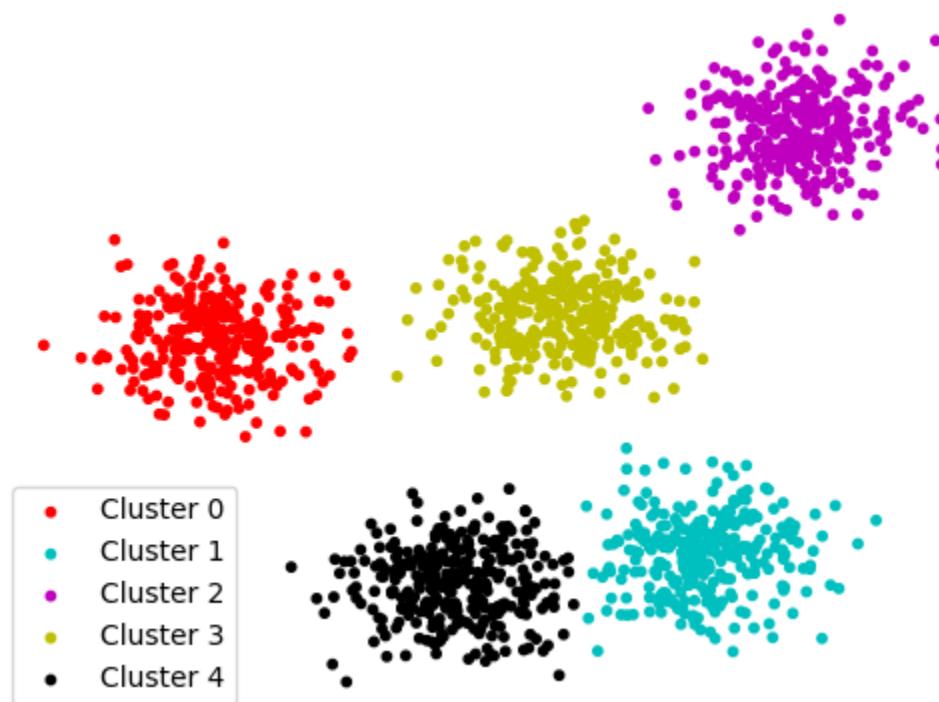
- Se da en la mayoría de definiciones de clustering
- Esta restricción podría ser relajada

Partial = una instancia podría no pertenecer a un cluster

- Útil para encontrar instancias anómalas

Otra variante de clustering es **flat** y  
**hierarchical** clustering

*Flat clustering:* dado un dataset, retornar sus particiones



## Task description: Flat clustering

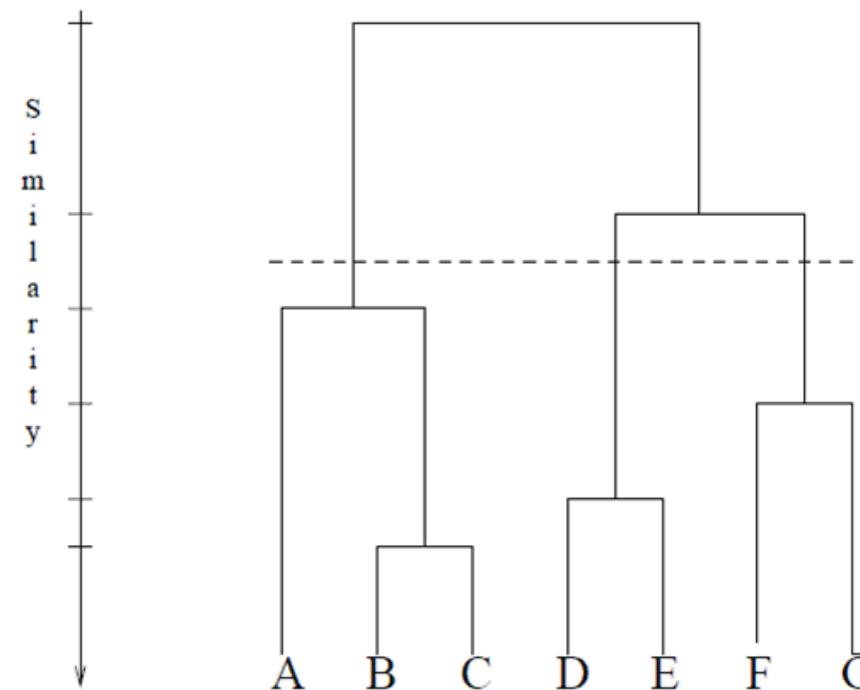
Given: a data set  $S$

Find: a set  $\mathcal{C}$  of  $k$  clusters  $C_i \subseteq S$ ,  $i = 1, \dots, k$  such that:

- $\forall C_i, C_j : C_i \cap C_j = \emptyset$  (disjoint)
- $C_1 \cup C_2 \cup \dots \cup C_k = S$  (exhaustive)
- $\mathcal{C}$  optimizes some quality criterion  $Q$ .

Otra variante de clustering es **flat** y  
**hierarchical** clustering

*Hierarchical clustering:* combina clusters para hacerlos más grandes hasta que exista un solo cluster (dataset)



Esta jerarquía genera una taxonomía

## Task description: Hierarchical clustering

**Given:** a data set  $S$

**Find:** a set  $\mathcal{C}$  of clusters such that:

- $S \in \mathcal{C}$
- $\forall x \in S : \{x\} \in \mathcal{C}$
- $\forall C_i, C_j \in \mathcal{C} : (C_i \cap C_j = \emptyset) \vee (C_i \subseteq C_j) \vee (C_j \subseteq C_i)$
- $\mathcal{C}$  optimizes some quality criterion  $Q$ .

Variaciones menos conocidas son  
las “**extensional**” y “**conceptual**”

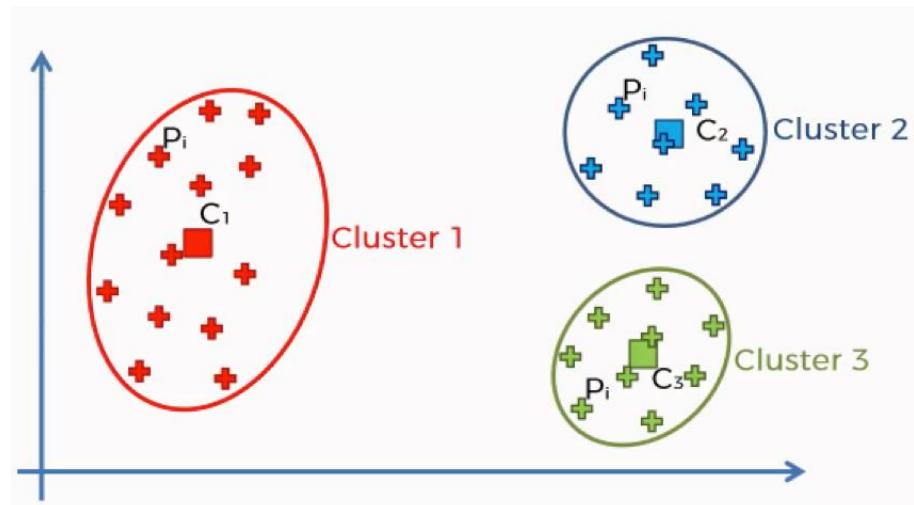
### Extensional clustering

- Los clusters se definen como conjuntos de ejemplos
- Utilizadas mayormente en el campo estadístico

### Conceptual clustering

- Los clusters son descritos en algún tipo de lenguaje (significado semántico)
- Intentan maximizar la similaridad intra-cluster

# Flat clustering junta elementos cercanos en un espacio multidimensional $R^n$



Dado un conjunto de datos no etiquetados  $X$

Encontrar clusters de instancias similares  $X_c, c \in C$

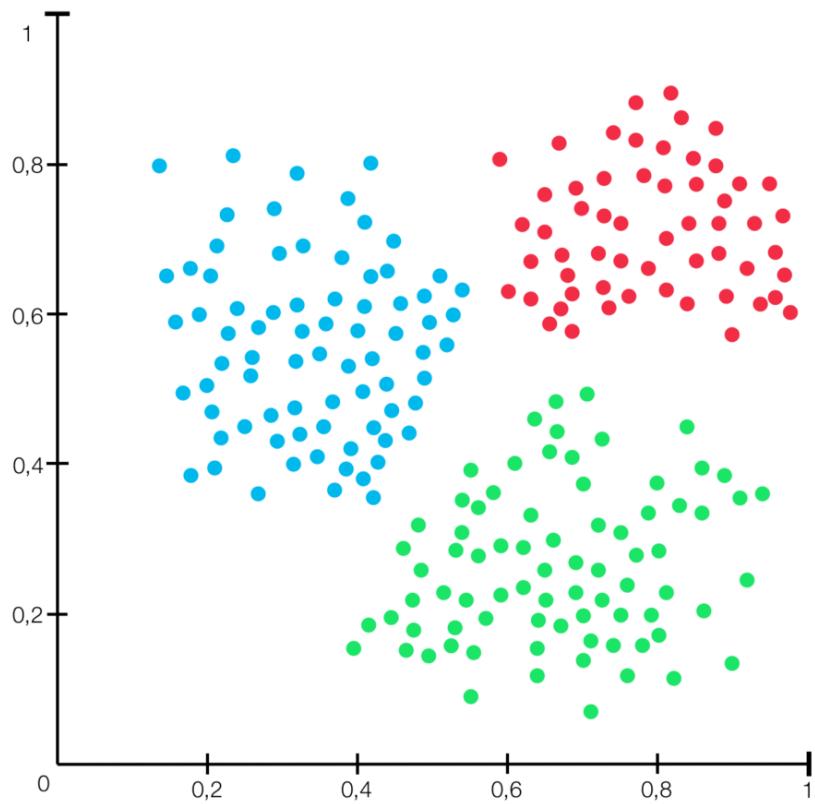
Similar -> instancias cercanas en algún espacio

El número de clusters puede ser dado como parámetro

Se puede asignar criterios de calidad

Ejemplos: K-means, Expected Maximization (EM)

# Content



**Variantes de clustering**

Tipos y características

Evaluación de soluciones

Determinar la calidad de una solución

**Algoritmos de clustering**

K-means, jerárquicos, aglomerativos

**Reducción de dimensiones**

PCA, UMAP, codificadores

Al no haber datos anotados, la **evaluación de los clusters** se vuelve una tarea subjetiva

Cuando hay datos anotados, se puede determinar clusters “correctos” o “incorrectos”

La mayoría de las veces, los datos no están anotados

- Hay veces que existe una división “natural” que puede ser comparada con los resultados obtenidos
- No se conoce ninguna información sobre los datos, por lo tanto, no se conoce una partición natural (la mayoría de los casos)

# En la mayoría de los casos, no se conoce una partición correcta de los datos (datos no anotados)

Instancias similares deberían estar en el mismo cluster

Instancias disimilares deberían estar en clusters distintos

Una medida de similaridad se usa para determinar distancias entre instancias

- Intra-cluster similarity: cuan similares son las instancias dentro del cluster en promedio (IntraCS)
- Inter-cluster similarity: cuan similares son las instancias que pertenecen a clusters distintos en promedio (InterCS)
- Buena partición cuando IntraCS es alto e InterCS es bajo
- Se debe optimizar el trade-off IntraCS/InterCS o intraCS/InterCS

Particionar datos (clustering) necesita la definición de una **medida de similaridad** entre instancias

Una medida de similaridad se define como  $s: X \times X \rightarrow R^+$  que expresa la similitud entre  $x_1$  y  $x_2$

Mientras más similitud exista, el valor  $s(x_1, x_2)$  será más alto

Así mismo, una medida de disimilaridad  $d: X \times X \rightarrow R^+$  se vuelve 0 cuando se comparan dos instancias iguales

Las **noción**es de similaridad son muy generales y  
no hay imposiciones en propiedades como simetría

Se definen las siguientes propiedades

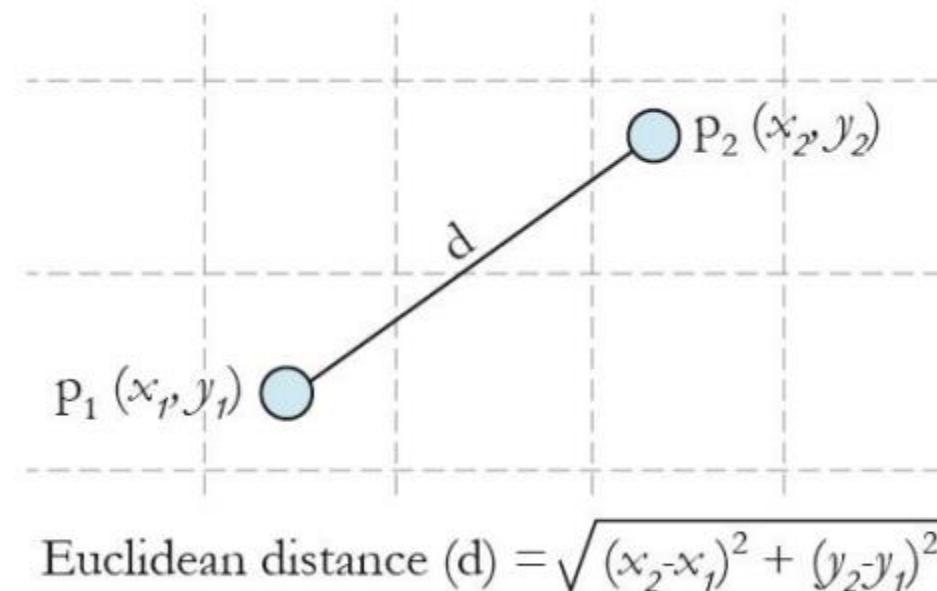
- $d(x, y) = 0 \leftrightarrow x = y$
- $d(x, y) = d(y, x)$ , simetría
- $d(x, y) \leq d(x, z) + d(z, y)$ , inequidad de triángulo

La distancia más comúnmente utilizada en ML es la **distancia euclídea**

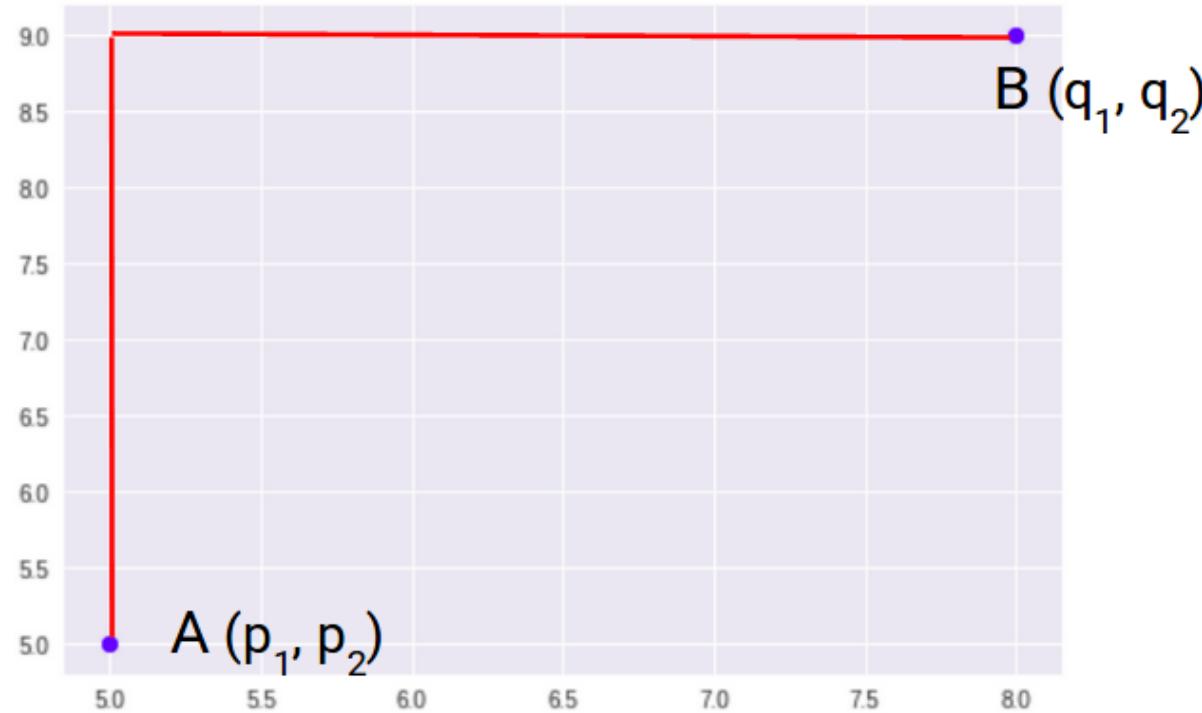
Asumiendo un espacio dimensional  $D$  y definiendo instancias del tipo  $x_j = (x_{j,1}, x_{j,2}, \dots, x_{j,D})$ , la distancia euclídea se define como:

$$d(x_1, x_2) = \sqrt{\sum_{i=1}^D (x_{1,i} - x_{2,i})^2}$$

La distancia más comúnmente utilizada  
en ML es la **distancia euclídea**



La distancia **manhattan** es otra medida comúnmente utilizada en ML



La distancia entre dos puntos es la suma de las diferencias absolutas de sus coordenadas

$$d(x_1, x_2) = \sum_{i=1}^D |x_{1,i} - x_{2,i}|$$

La distancia **Minkowski** es una forma generalizada para las distancias euclíadiana y manhattan

$$d(x_1, x_2) = \left( \sum_{i=1}^D |x_{1,i} - x_{2,i}|^p \right)^{1/p}$$

Donde  $p$  representa el orden de la distancia

Con  $p = 1$  -> Manhattan distance

Con  $p = 2$  -> Euclidian distance

La distancia de **Hamming** mide la similaridad de dos strings del mismo tamaño

$$d(x_1, x_2) = \sum_{i=1}^D \delta(x_{1,i}, x_{2,i})$$

$$\delta(x, y) = \begin{cases} 0, & x = y \\ 1, & x \neq y \end{cases}$$

Es una medida usada con datos categóricos o Strings

$$x_1 = 10\textcolor{red}{1}1\textcolor{red}{1}01 \text{ and } x_2 = 10\textcolor{red}{0}1001$$
$$d(x_1, x_2) = 2$$

Determinar el **número de clusters** es una tarea no trivial, pero existen técnicas para ayudar determinarlo

### The “elbow” method

Se calcula un score para diferentes configuraciones de  $K$

### The silhouette coefficient

Intra-cluster y distancia a otros clusters para diferentes  $K$

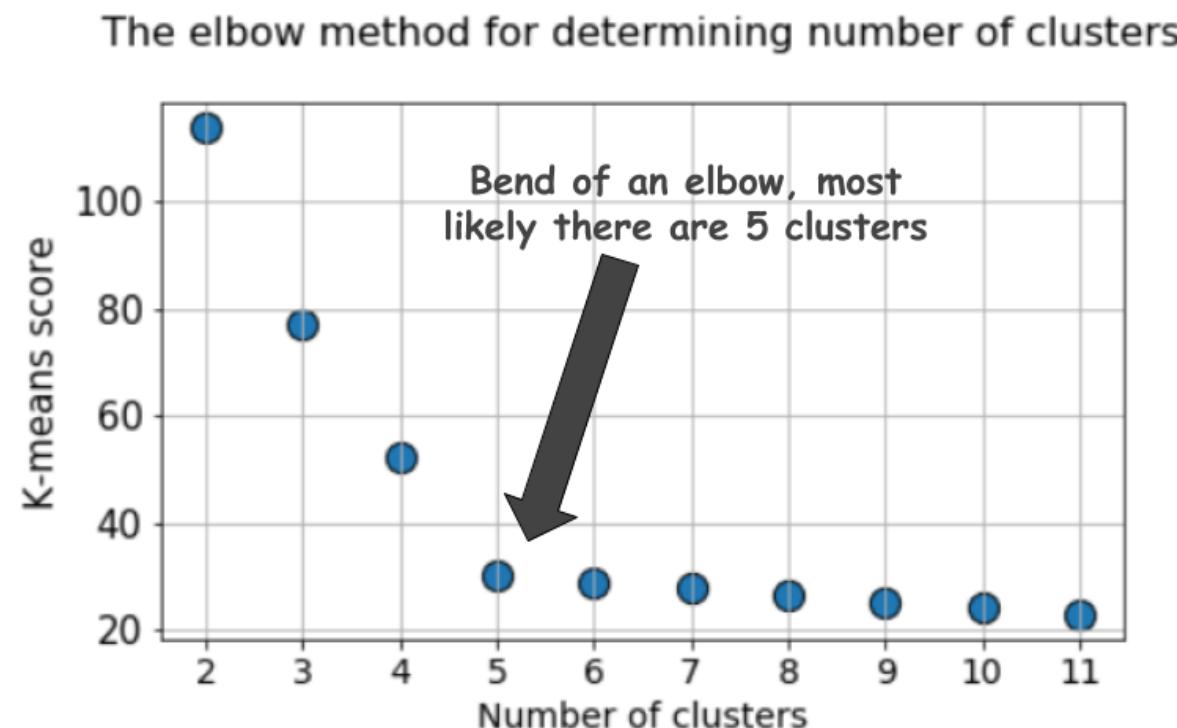
### The Gap statistic

Cuanto mejora o empeora un cluster cuando se aumenta o quita un cluster

El método más común es el de “elbow” por su simplicidad

Se corre el algoritmo de forma incremental aumentando el número clusters

El score calculado es la suma de las distancias al cuadrado de cada elemento al centro del cluster



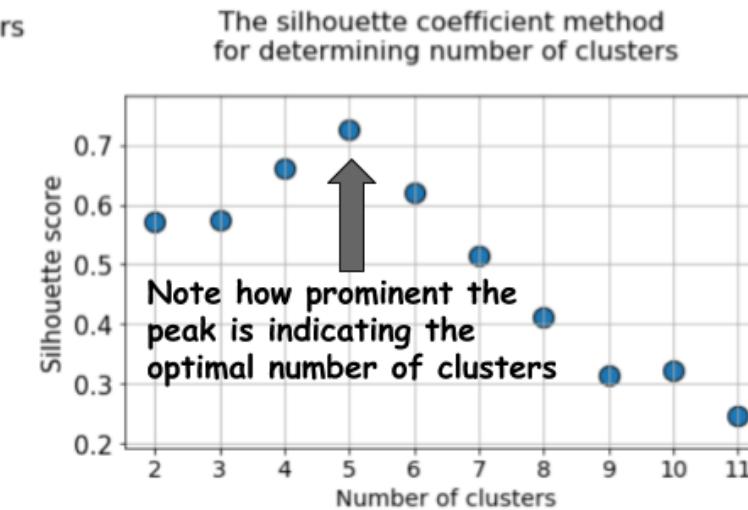
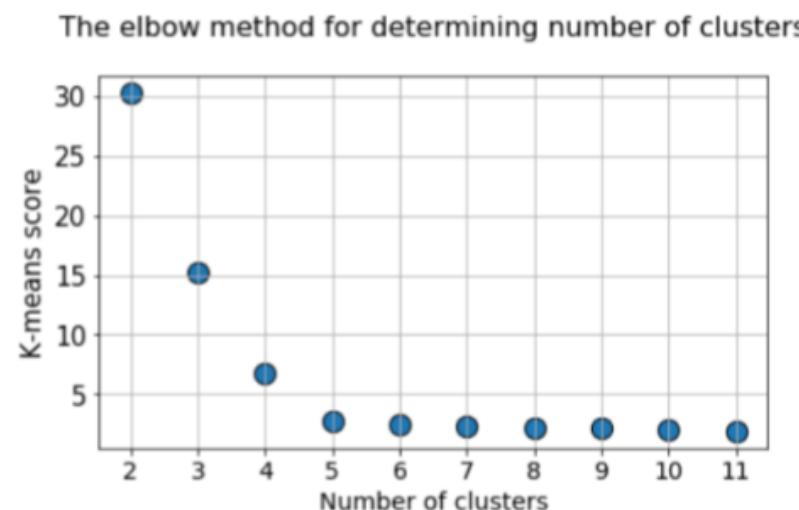
La determinación del  $K$  óptimo se lo hace visualmente (tarea altamente subjetiva)

# El silhouette score es una métrica que permite determinar $K$ de mejor manera

El score se calcula usando la distancia media intra-cluster (a) y la distancia media al cluster más cercano (b) para cada elemento

$$Score = \frac{b - a}{\max(a, b)}$$

Este score se calcula usando todos los elementos del dataset

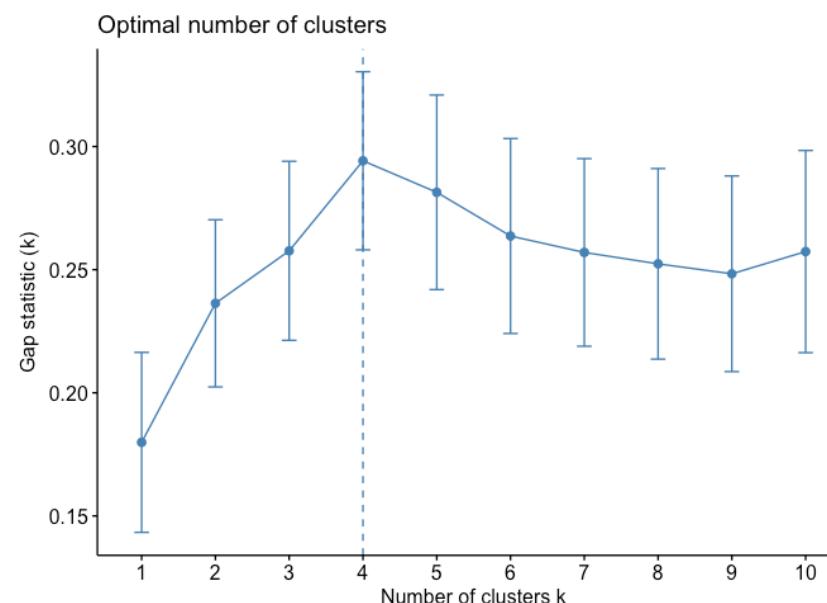


# Otra técnica es “**gap statictic**” propuesta por Tibshirani et. al (2001)

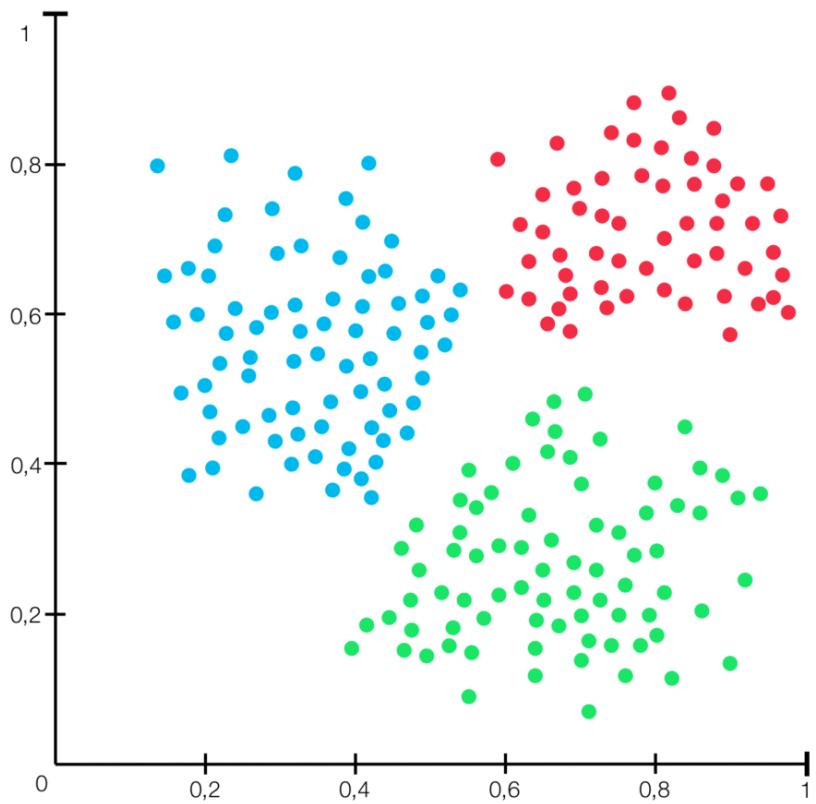
Compara la variación total intra-cluster para diferentes valores de K con los valores esperados bajo una distribución de referencia nula (distribución donde no hay clusters)

El dataset de referencia se crea usando simulación Montecarlo

Se calcula  $K$  como el gap más pequeño entre dos gap statistics consecutivas



# Content



**Variantes de clustering**

Tipos y características

**Evaluación de soluciones**

Determinar la calidad de una solución

**Algoritmos de clustering**

K-means, jerárquicos, aglomerativos

**Reducción de dimensiones**

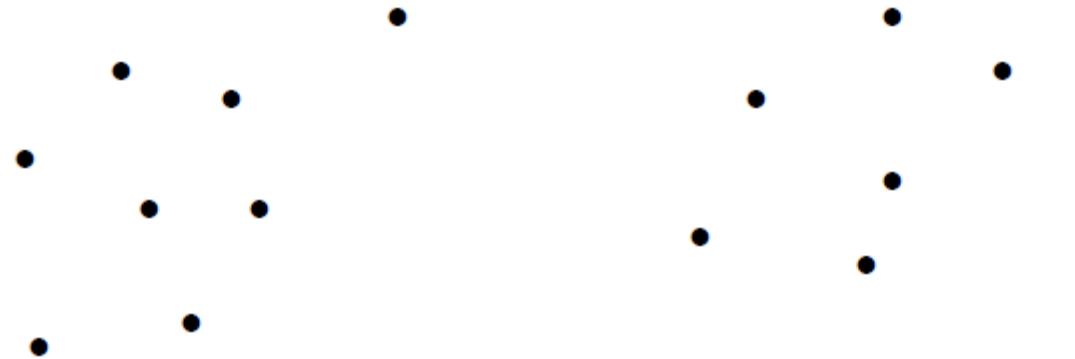
PCA, UMAP, codificadores

Uno de los algoritmos más simples y ampliamente utilizados es el **K-means**

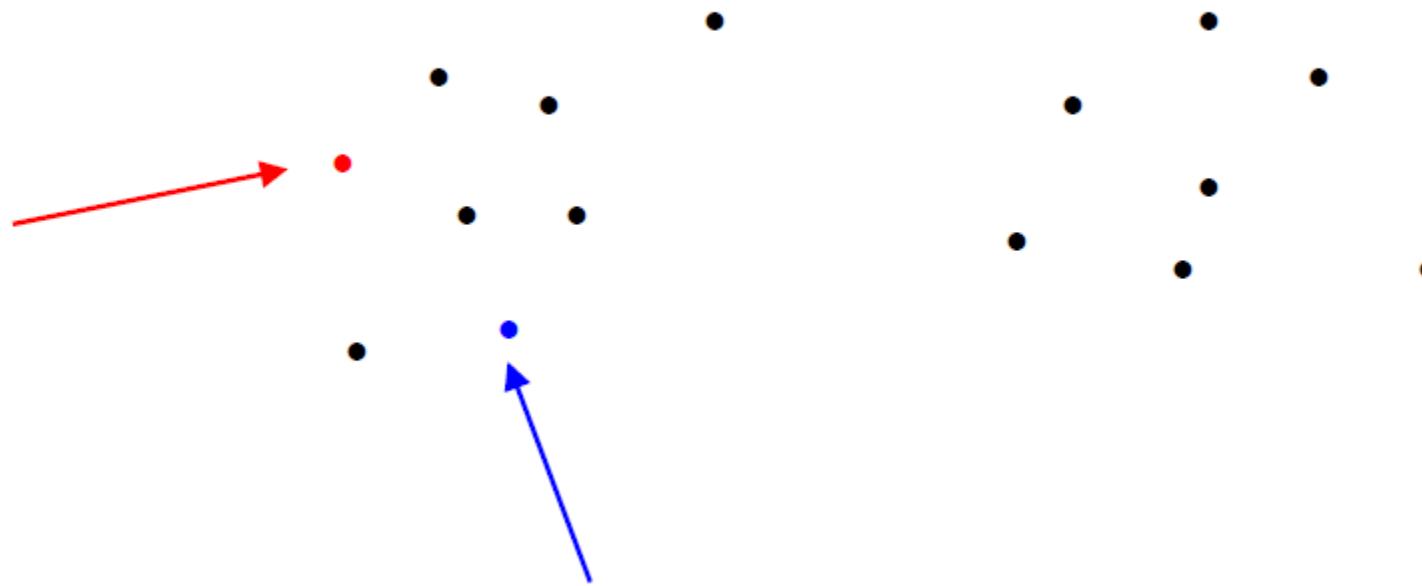
Input : data, number of clusters K

Algorithm:

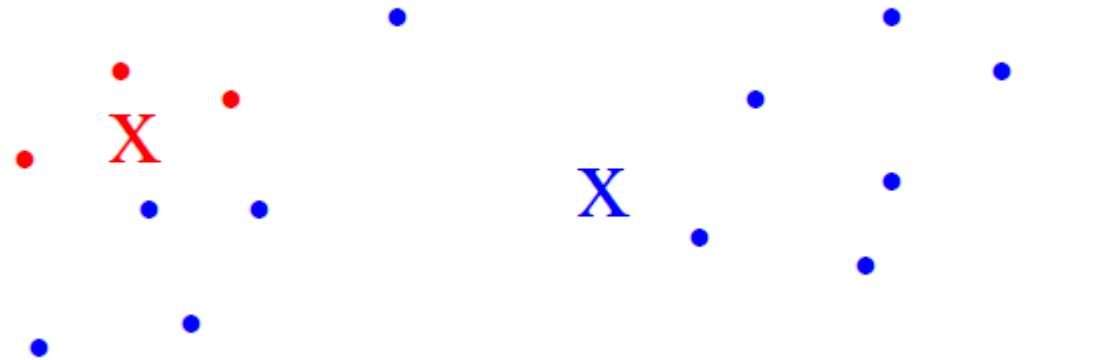
- Start with K random seeds for clusters
- Repeat until no changes:
  - For all instances e:
    - Add e to cluster of closest seed
  - Compute centres of all clusters
    - E.g., for numeric data: compute average
  - These centres are the seeds for the next iteration



Se parte de los datos originales que no están anotados

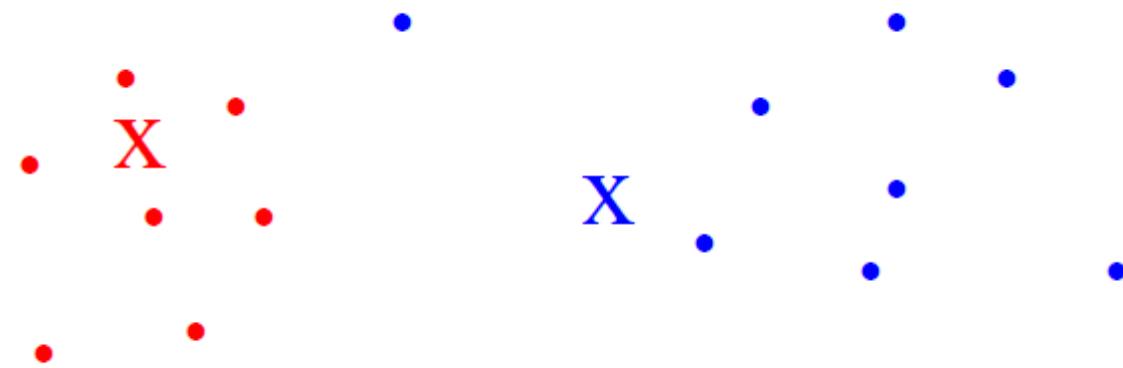


Con K=2, se eligen dos orígenes aleatoriamente



Los puntos son asignados a un cluster comparando las distancias del punto al origen (centroide) del cluster

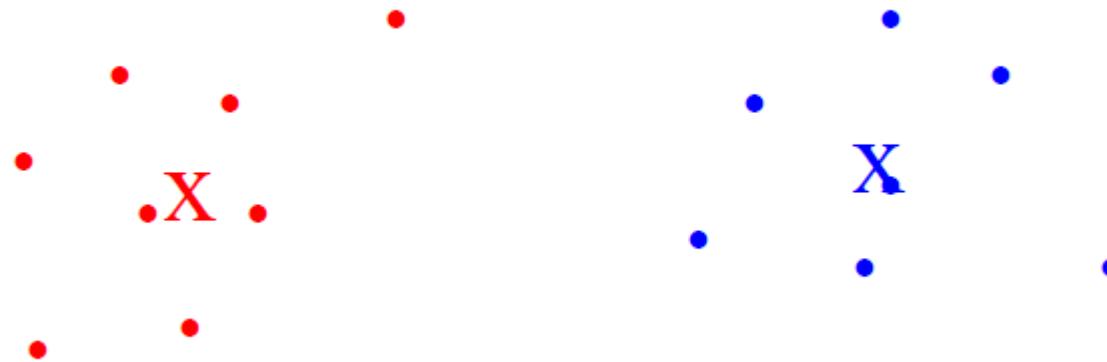
Se calculan los nuevos centroides



se reasignan las instancias al cluster más cercano



Se calculan nuevamente los centroides de los clusters



Se reasignan las instancias a los clusters más cercanos  
Cuando no hay nuevas re-asignaciones, el algoritmo para

El algoritmo **Expected Maximization (EM)** se parece al K-means, pero asume distribuciones normales de los datos

Similar a K-means, pero:

- Se asumen que los clusters son generados por distribuciones normales
- Una instancia puede pertenecer parcialmente a diferentes clusters, e.j. la instancia 1 pertenece 70% al cluster A y 30% al cluster B

EM es mucho más general que solamente una técnica de clustering

- Encuentra el número de distribuciones que generan los datos
- “mixture models”

**function** CLUSTEREM(data set  $S$ , integer  $k$ ) **returns** set of Gaussians:

Initialize  $\mu_i$  and  $\sigma_i$  randomly

**while** stop criterion not reached:

**for**  $j = 1$  **to**  $N$ :

**for**  $i = 1$  **to**  $k$ :

$$Pr(x_j \in C_i) := \frac{exp(-(x_j - \mu_i)^2 / 2\sigma_i^2)}{\sum_{i'=1}^k exp(-(x_j - \mu_{i'})^2 / 2\sigma_{i'}^2)}$$

**for**  $i = 1$  **to**  $k$ :

$$\mu_i := \frac{\sum_j x_j Pr(x_j \in C_i)}{\sum_j Pr(x_j \in C_i)}$$

$$\sigma_i^2 := \frac{\sum_j (x_j - \mu_i)^2 Pr(x_j \in C_i)}{\sum_j Pr(x_j \in C_i)}$$

**return**  $\{(\mu_1, \sigma_1), (\mu_2, \sigma_2), \dots, (\mu_k, \sigma_k)\}$

**Hierarchical Clustering** crea clusters de forma aglomerativa o divisiva generando una especie de jerarquía

### Top-down (“divisivo”)

- Se comienza con un cluster (todo el dataset)
- Este se divide en subsets
- Cada subset se divide en más subsets hasta que al final solo queden instancias individuales (un cluster por cada instancia)

**Hierarchical Clustering** crea clusters de forma aglomerativa o divisiva generando una especie de jerarquía

### Bottom-up (“aglomerativo”)

- Se comienza clusters individuales (cada instancia es un cluster)
- Se agrupan (aglomeran) clusters cercanos
- Se repite hasta que quede un solo cluster (el dataset completo)

# Existen muchas técnicas aglomerativas descritas en la literatura

**Variantes en función de la definición de “cercanía de clusters”**  
Criterios de distancia, distancia entre clusters

**Una opción es “single linkage”**  
Distancia entre clusters = distancia entre elementos más cercanos

**Otra opción es “complete linkage”**  
Distancia entre clusters = distancia entre los puntos más lejanos

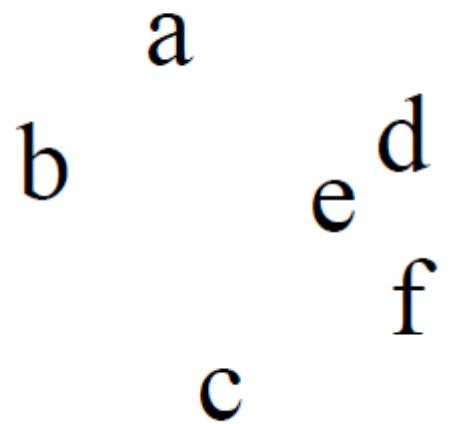
**Promedio**  
Distancia promedio entre single y complete linkage

Single linkage:  $d_c(C_1, C_2) = \min\{d(x_1, x_2) | x_1 \in C_1, x_2 \in C_2\}$

Complete linkage:  $d_c(C_1, C_2) = \max\{d(x_1, x_2) | x_1 \in C_1, x_2 \in C_2\}$

Average linkage:  $d_c(C_1, C_2) = \frac{\sum_{x_1 \in C_1, x_2 \in C_2} d(x_1, x_2)}{|C_1||C_2|}$

Single linkage: junta elementos  
más cercanos primero



d-e son los más cercanos

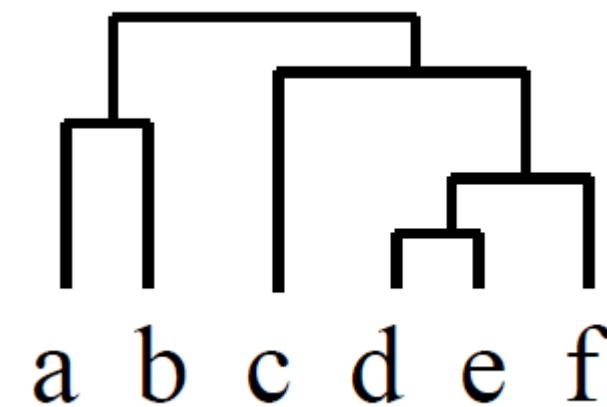
Luego e-f

Luego a-b

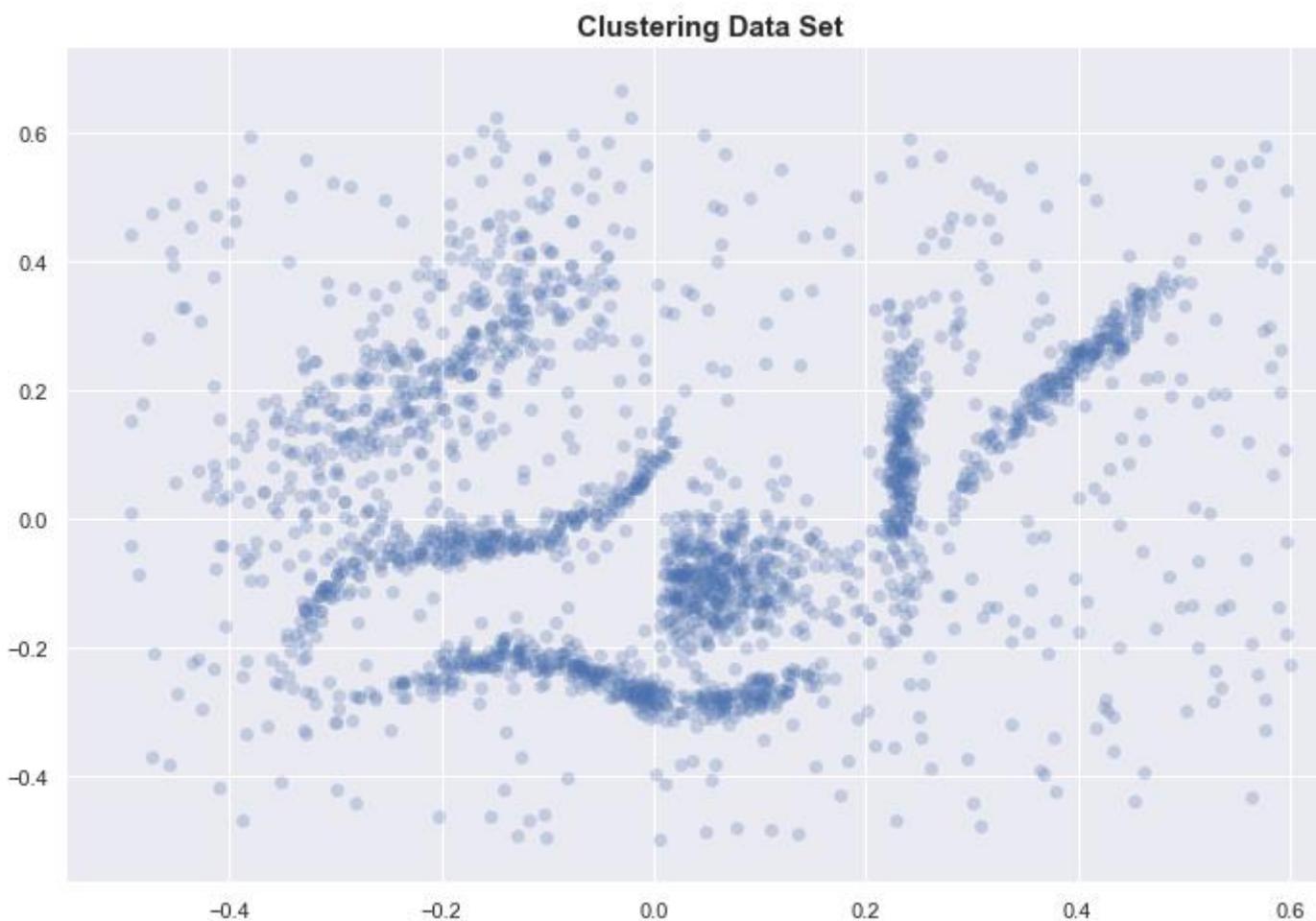
...

Single linkage: junta elementos  
más cercanos primero

a  
b  
c  
d  
e  
f

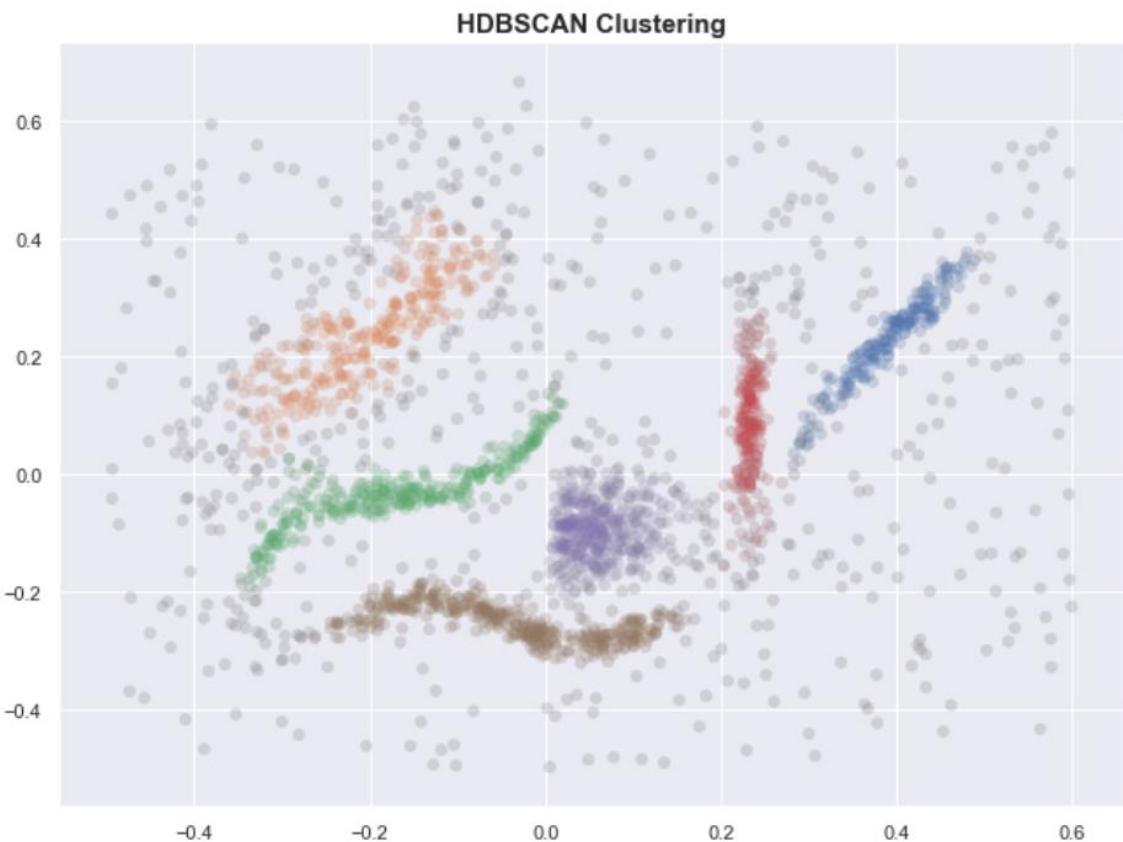
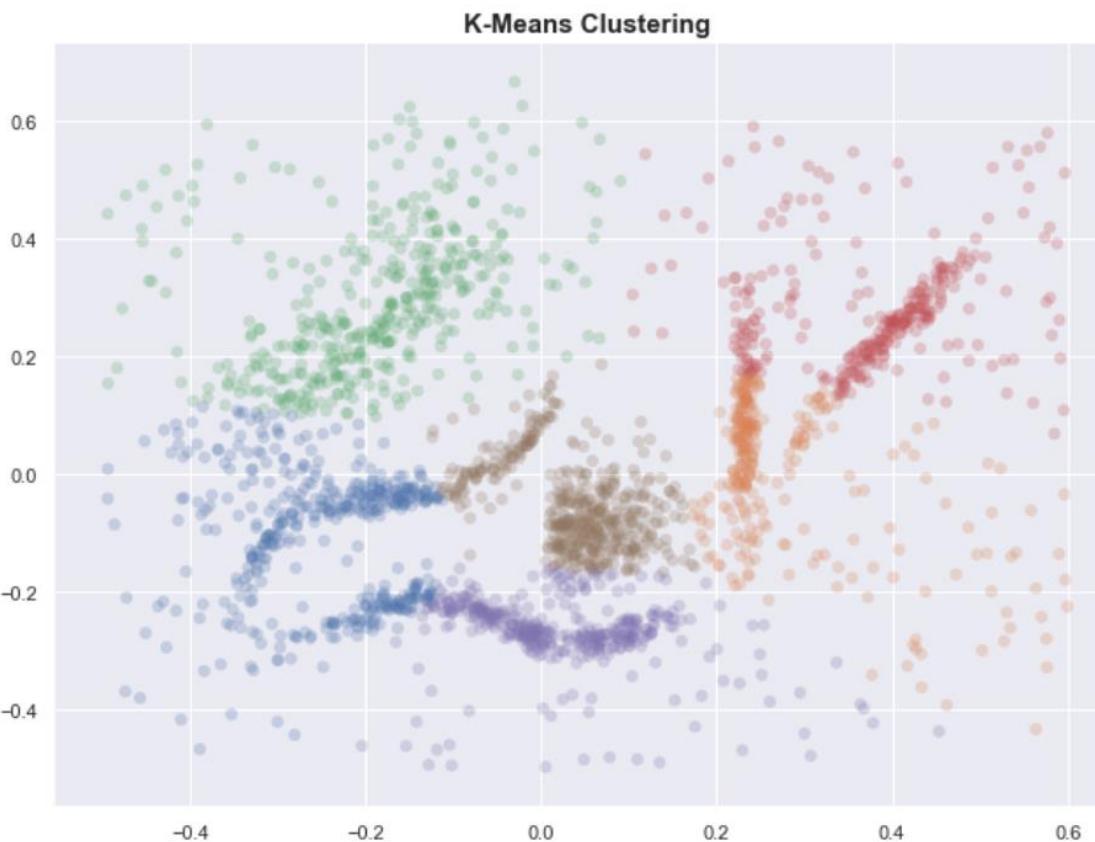


# Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN)



¿Cuántos clusters existen en este  
dataset? -> 6

Para este dataset, es obvio que HDBSCAN tiene un mejor resultado que K-Means

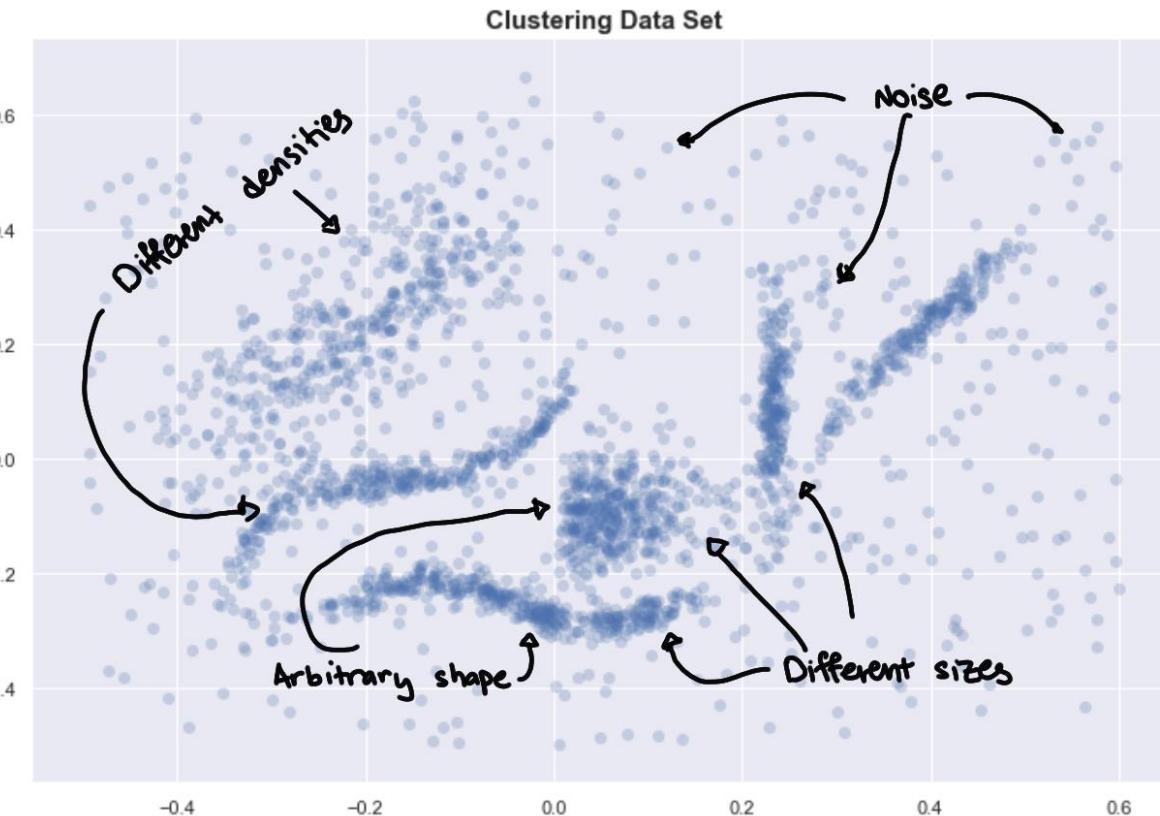


# K-Means falla por los supuestos que hace sobre los datos

K-Means funciona mejor cuando los clusters son:

- “redondos” o esféricos
- De tamaños similares
- Tienen la misma densidad
- Son más densos cerca al centroide
- No contiene ruido/outliers

# Ciertos datasets violan los supuestos que hace K-Means



Características del dataset:

- Clusters con formas diferentes
- Clusters con tamaños diferentes
- Clusters con diferentes densidades
- Existe ruido y outliers

HDBSCAN se basa en densidad, haciendo pocos supuestos sobre los datos

Método no paramétrico (no tiene parámetros)

Busca una jerarquía formada por modos multivariados

Busca por regiones que son más densas que sus alrededores

Extrae los clusters usando una técnica que se basa en la estabilidad de los clusters

El clustering conceptual retorna una descripción de los grupos en algún lenguaje  $L$

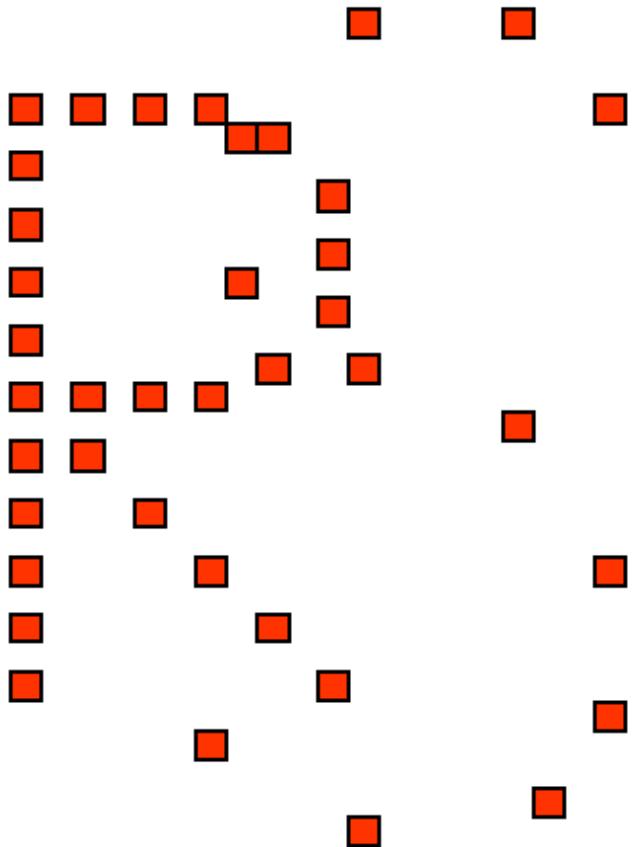
Los métodos anteriores forman grupos de elementos “similares”

Conceptual: descripción de grupos

- La calidad de los clusters depende de las propiedades del cluster
- Se prefiere una descripción *simple* de los clusters
- El lenguaje define el contexto en el que la calidad de los clusters se evalúa

Si dos elementos están en el mismo cluster, dependen de otros datos

¿Como se particionaría este dataset?



“clusters intuitivos” no pueden ser  
definidos solamente mediante  
distancias

TDIDT puede ser usado  
para crear clusters

Árbol de decisión = clustering jerárquico conceptual  
Cada nodo del árbol = 1 cluster

Pruebas desde la raíz al nodo  
Descripción conjuntiva del cluster

El lenguaje L de descripciones de clusters  
Atributos de prueba de conjunciones

El algoritmo **cobweb** (Fisher, 1987) encuentra clusters jerárquicos conceptuales

### Descripción probabilística

- Distribución probabilística de atributos por cada cluster

### Heurística

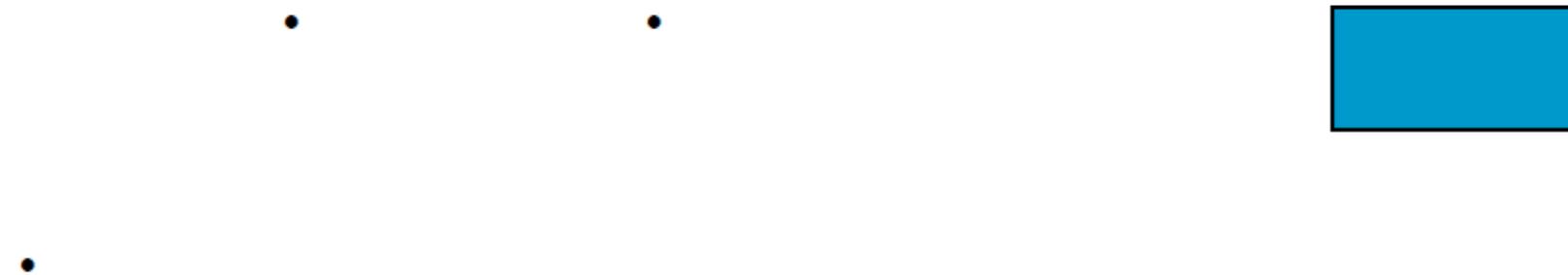
- Maximiza valores: predictiveness y predictability de los atributos
- Predictability: dado un cluster, que tan bien se pueden predecir valores de los atributos
- Predictiveness: dados valores de atributos, que tan bien se puede predecir un cluster
- Se maximiza la combinación de ambos

# Cobweb es un algoritmo incremental

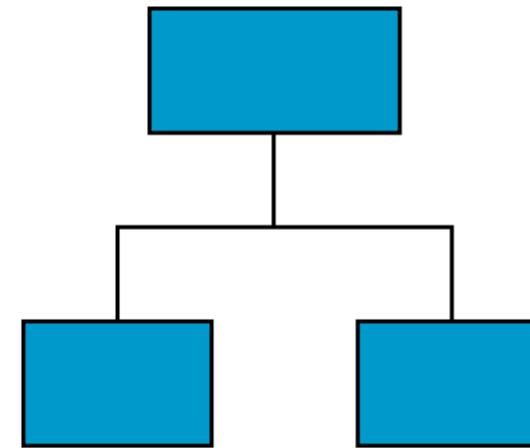
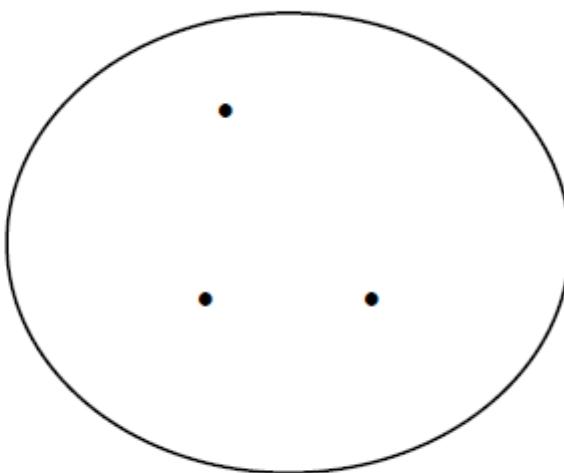
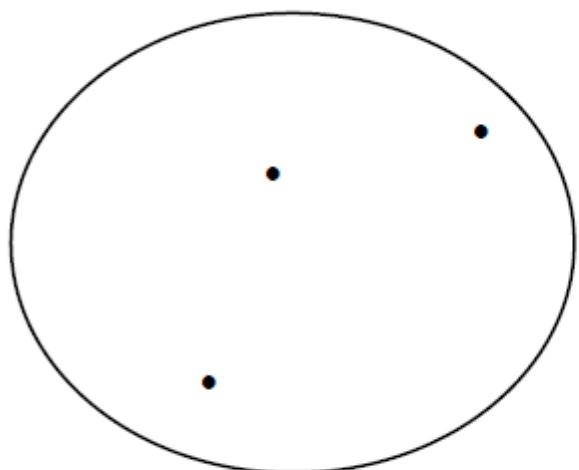
For each example:

- For each level (top to bottom) of current taxonomy:
  - Change current level using one of several *operators*
    - Add example to cluster
    - Create new cluster (with 1 example)
    - Merge clusters
    - Split clusters
  - Move down to relevant subcluster
- Evaluation of clustering: try to maximize a combination of predictiveness  $P(C_k|A_i=v_{ij})$  and predictability  $P(A_i=v_{ij}|C_k)$  ( $C_k$ =cluster,  $A_i$ =attribute,  $v_{ij}$  = value)

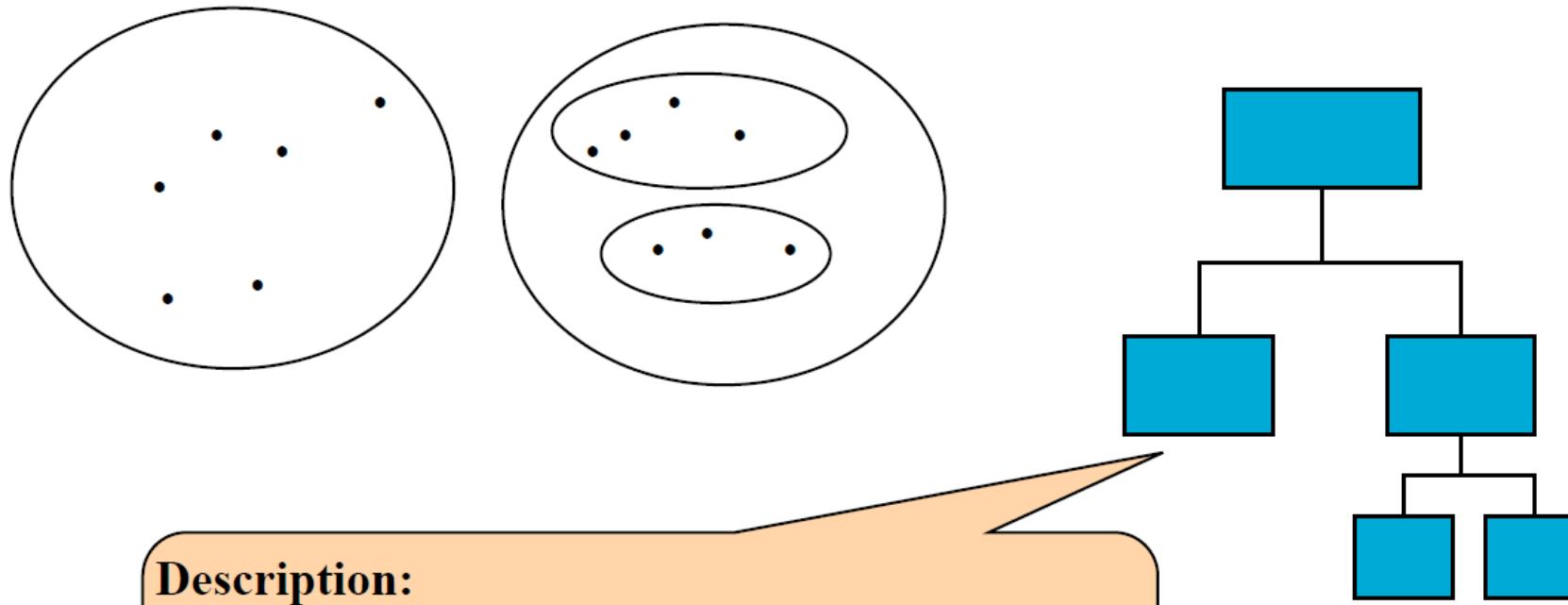
Ilustración:



## Ilustración:



# Ilustración:



**Description:**

Color=blue (90%), red (10%)

Size=small (80%), medium (15%), large (5%)

...

# Los clusters pueden ser usados para predicción

Clustering puede ser usado para predecir cualquier propiedad

Una vez que se tienen clusters, la predicción se realiza en dos pasos:

- Dados atributos de instancia, predecir cluster (función si existe alta predictiveness)
- Dado un cluster, predecir atributos desconocidos (funciona si existe alta predictability)

= “predicción flexible”: no se conoce de antemano lo que será entregado y lo que se necesita predecir

# Los clusters pueden ser usados para predicción

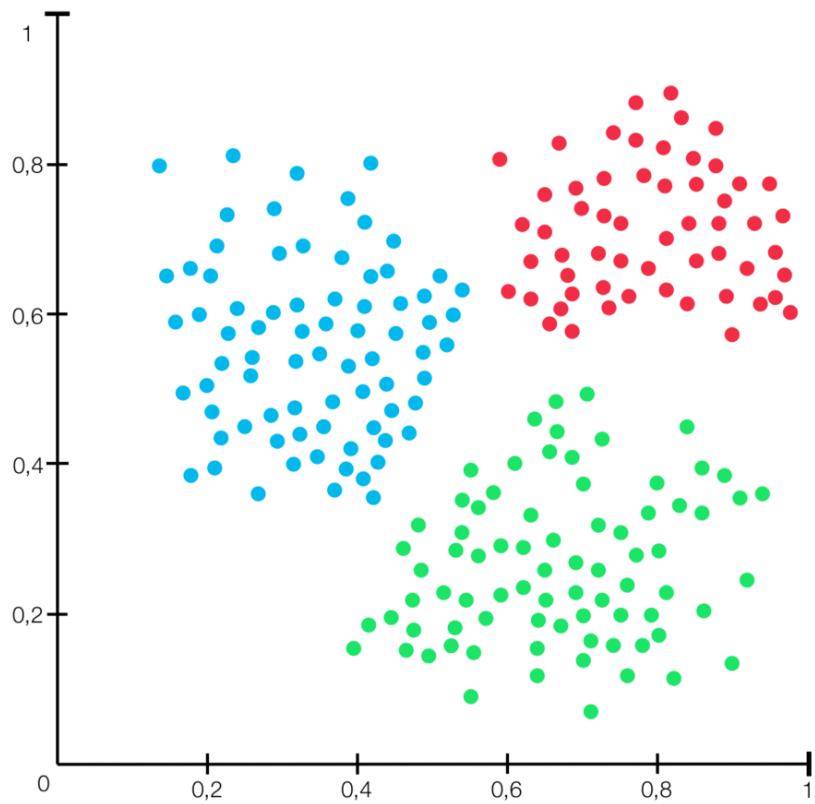
Si se conoce que será dado, el proceso de clustering puede ser tuneado

- Maximizar predictiveness de los atributos que serán dados
- Maximizar predictability de los atributos que necesitan ser predecidos

Muchas técnicas pueden ser consideradas en este framework

- E.j. árboles de decisión, instance-based learning, etc.

# Content



**Variantes de clustering**

Tipos y características

**Evaluación de soluciones**

Determinar la calidad de una solución

**Algoritmos de clustering**

K-means, jerárquicos, aglomerativos

**Reducción de dimensiones**

PCA, UMAP, codificadores

La **reducción de dimensiones** se usa para graficar datos, denoising, encontrar la dimensionalidad intrínseca

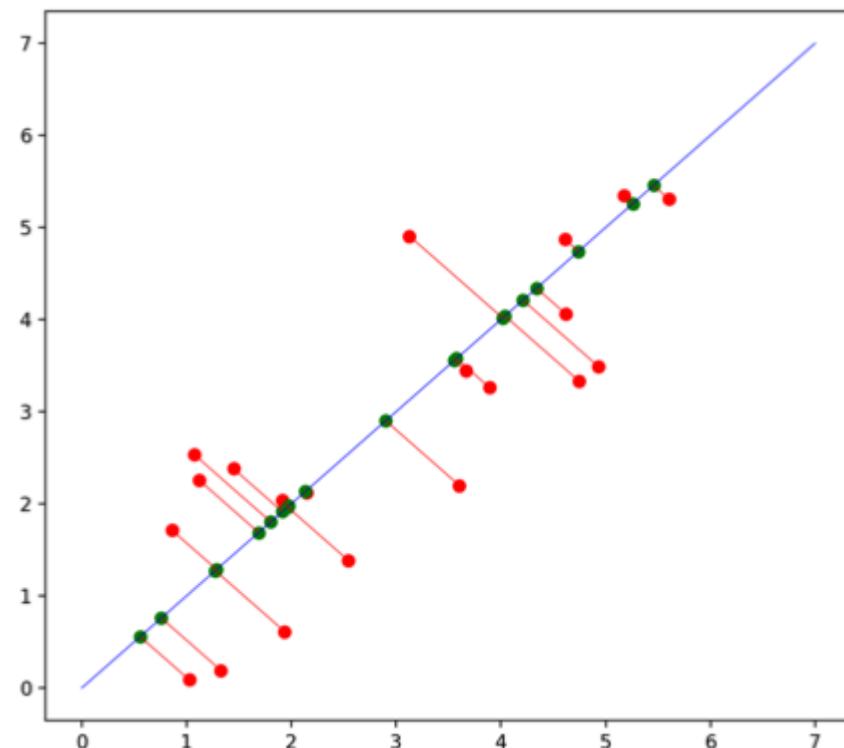
Las computadoras modernas tienen mayor potencia  
Pueden tratar más eficientemente múltiples dimensiones en los datos

Técnicas como DNN hacen uso eficientemente de GPUs  
Además, DNN encuentra la dimensionalidad de forma intrínseca

Técnicas más simples se benefician de reducción de dimensiones  
De esta forma sus modelos se vuelven más robustos

Remueven datos redundantes o altamente correlacionados  
La calidad de los datos no se pierde y en muchos casos mejora

# Principal Component Analysis (PCA) es uno de los métodos lineales más antiguos para reducción de dimensiones



Se basa en proyecciones de datos en componentes

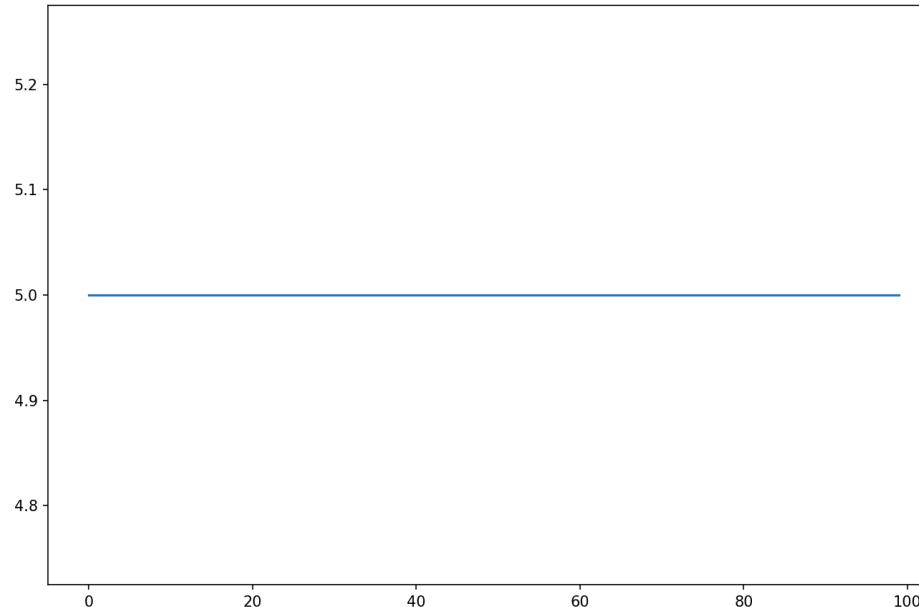
## Beneficios

- Reduce al dimensionalidad del dataset
- Resultados interpretables
- Tiempo de ejecución relativamente rápido

## Contras

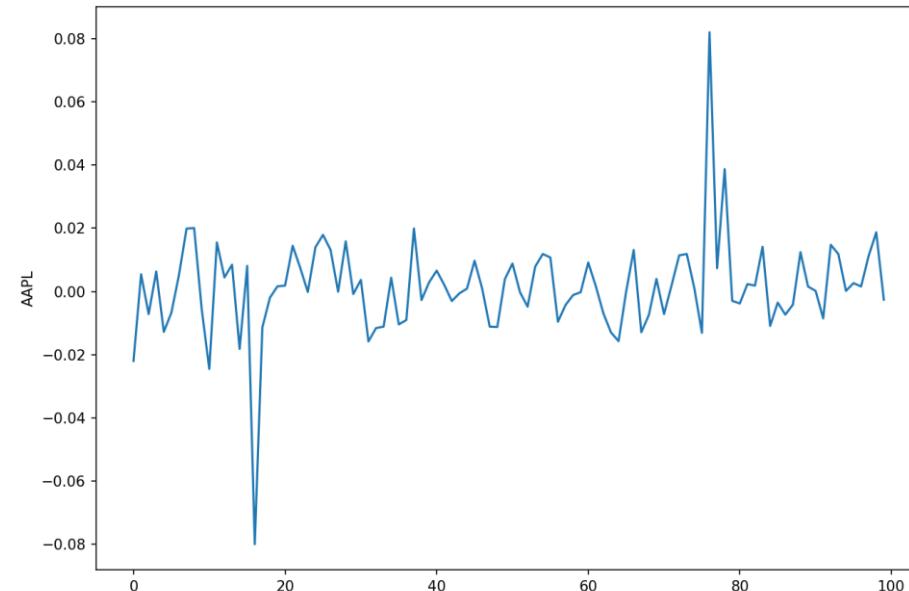
- No puede aprender representaciones no-lineales de las características del dataset

Si se tiene característica (columna) de un dataset con varianza zero, no ayuda a generar ningún modelo



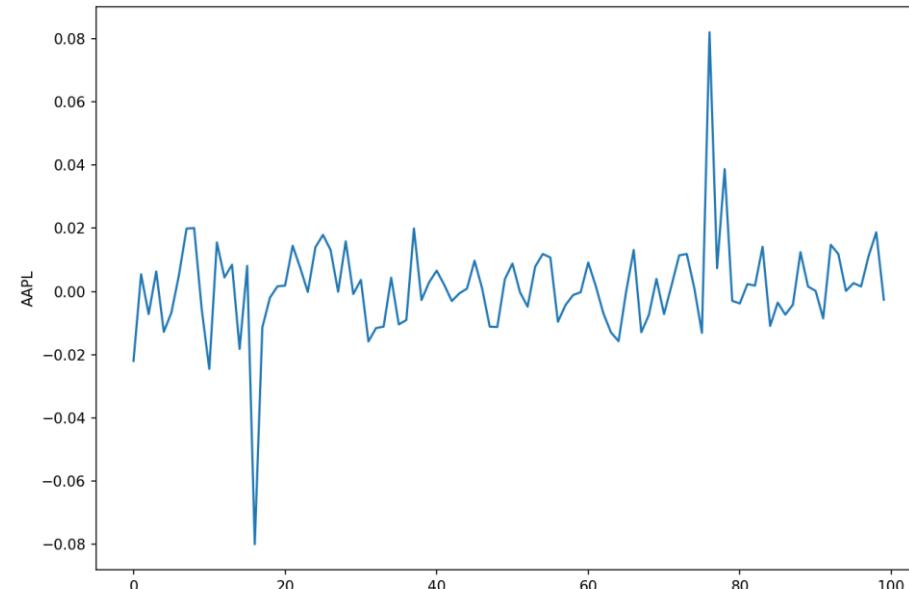
Este tipo de features no proveen información relevante para un modelo

Las características con varianza  $\neq 0$  proveen información para generar un modelo



La varianza es que tanto los datos cambian de valor

Las características con varianza  $\neq 0$  proveen información para generar un modelo



Se puede decir que los datos se componen de la señal y ruido

# PCA intenta capturar las señales con mayor varianza del dataset usando componentes principales

Si se tiene un dataset con más características que elementos, es muy probable que se genera overfitting

El conjunto ideal de características debe tener:

- Alta varianza: contienen mayor potencial de información
- No-correlacionados: características altamente correlacionadas entre sí son menos útiles
- No muchos elementos: menos características que elementos en el dataset

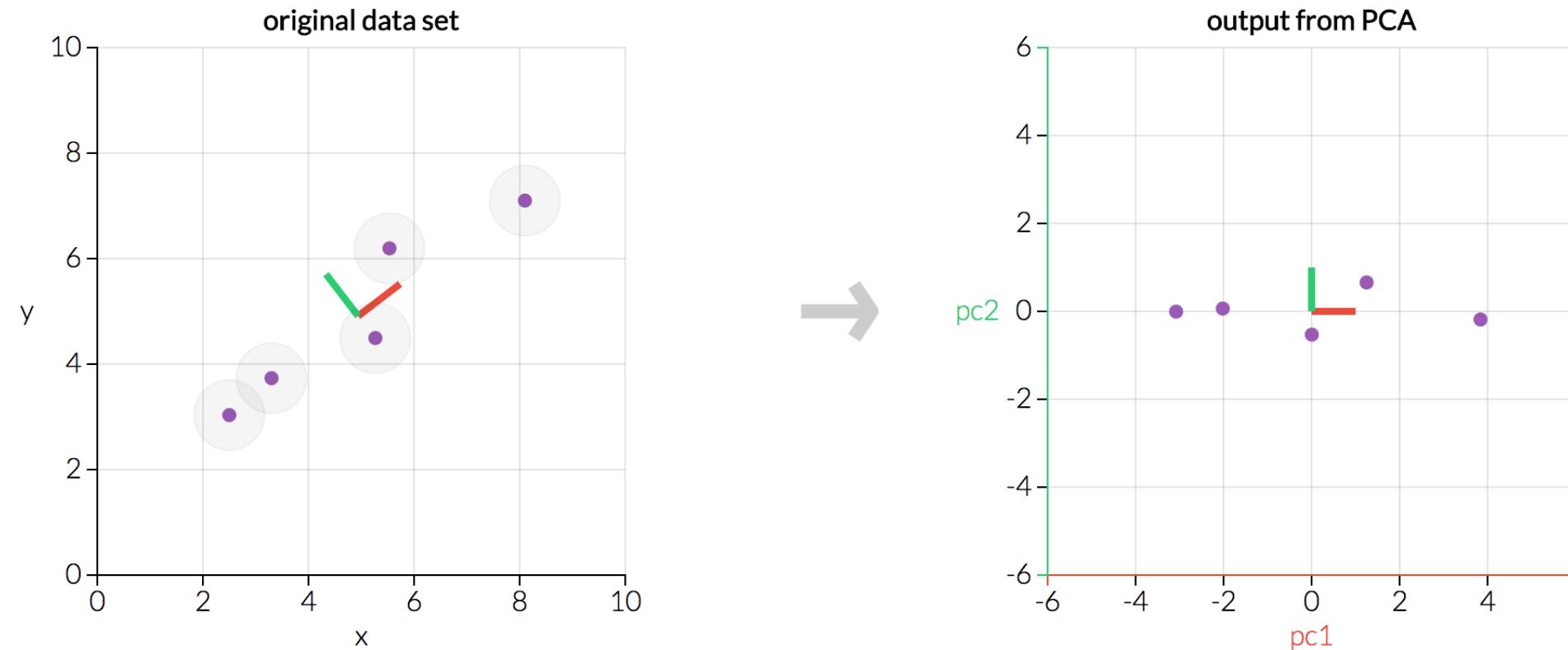
PCA crea un conjunto de componentes principales ranqueados en orden de varianza

PCA funciona buscando las características con mayor varianza, una a una

En un proceso iterativo:

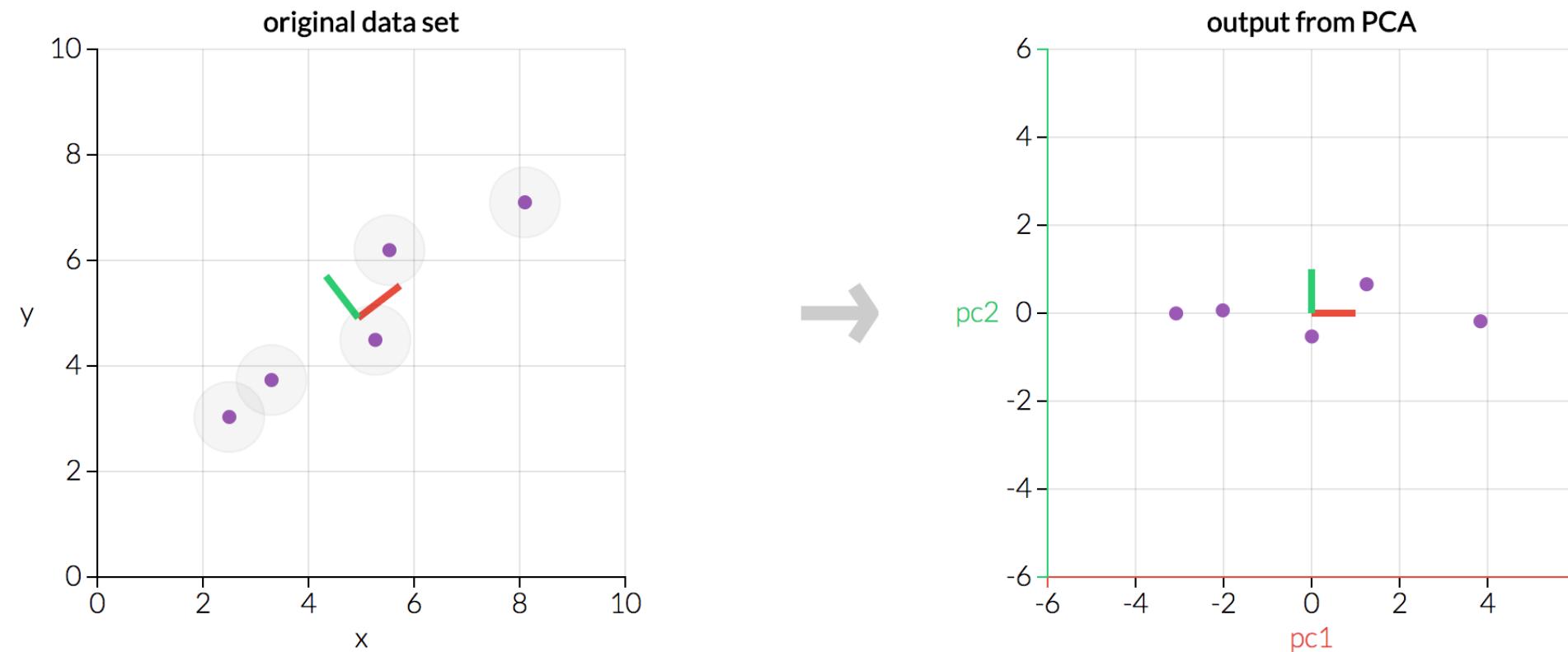
1. Busca el “trend” más grande en el conjunto de características (componente 1)
2. Busca el siguiente “trend” más grande (componente 2)
3. Continúa con el proceso hasta obtener todos los componentes

Los componentes encontrados son ortogonales entre sí  
(estadísticamente independientes unos con otros)



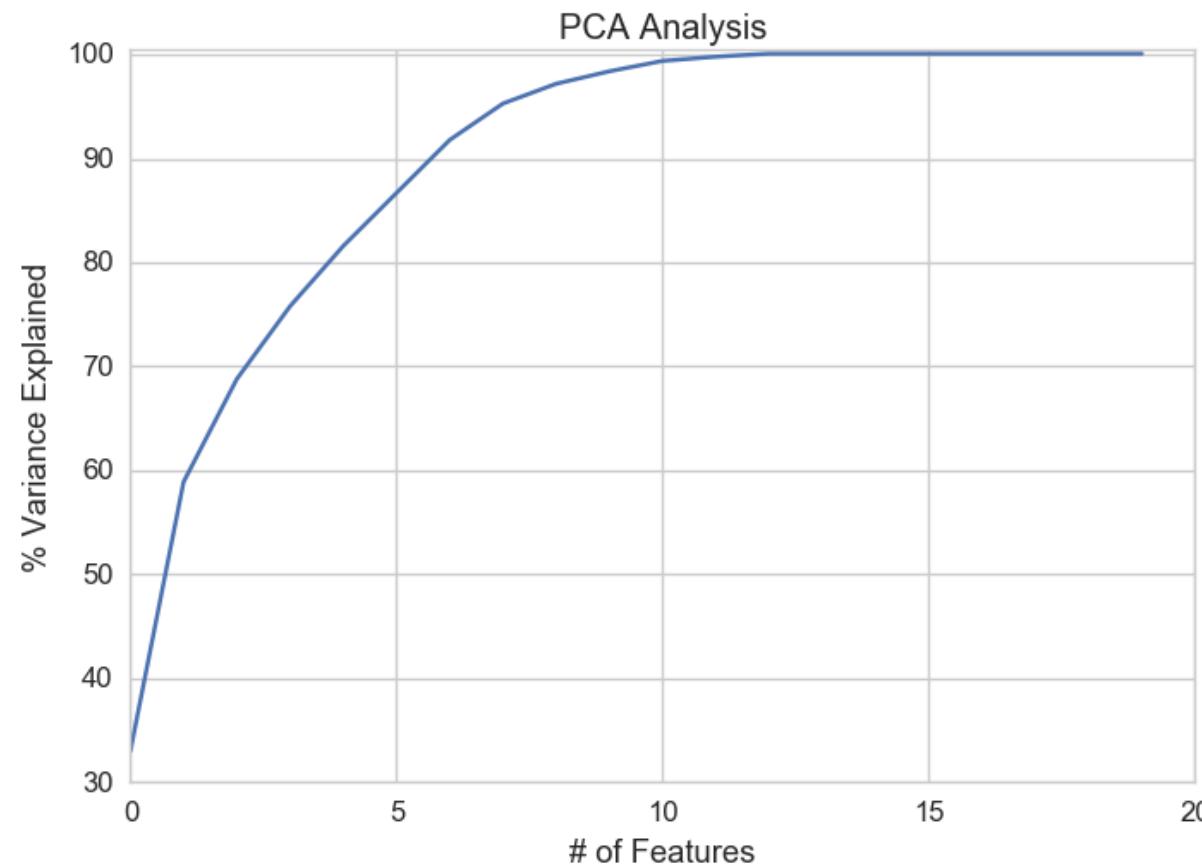
En este caso, el componente 1 (pc1) es el que más varianza explica del dataset original

Los componentes encontrados son ortogonales entre sí  
(estadísticamente independientes unos con otros)



Los *principal components* se calculan mediante los eigen-vectores de la matriz de datos original

El número de componentes finales se elige típicamente mediante un threshold para explicar una cantidad de varianza



E.j. si se quiere explicar el 90% de varianza, se necesitan 6 componentes

Componente != dimensión original

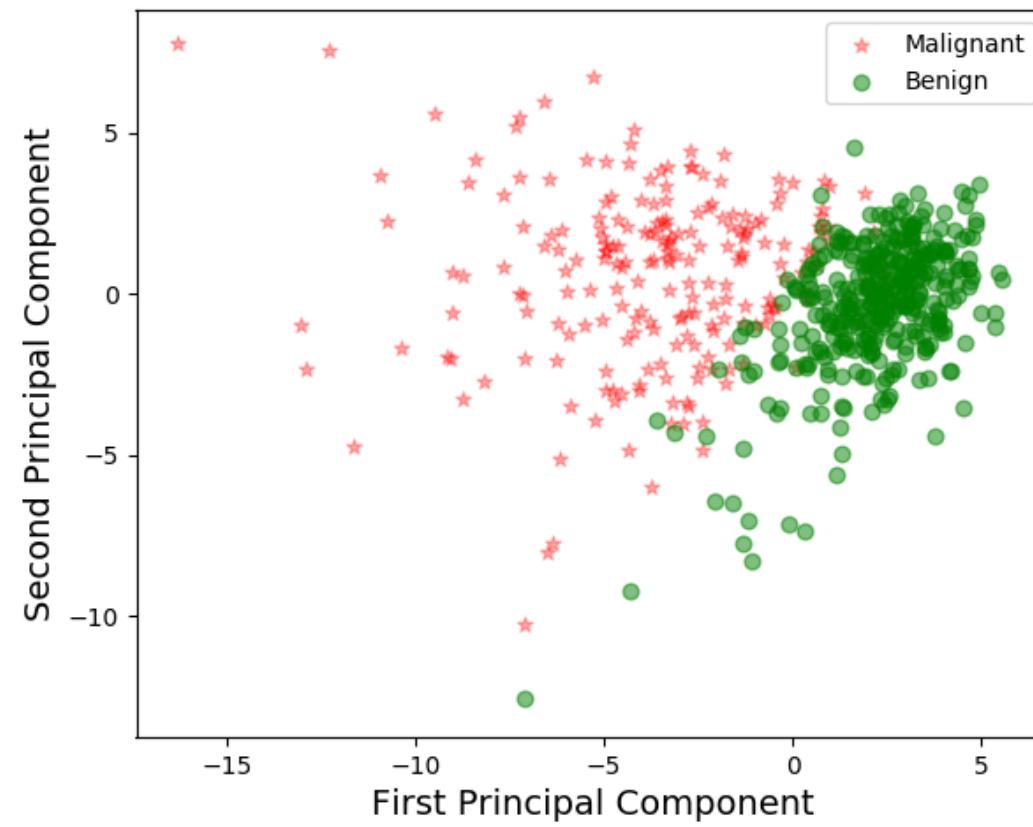
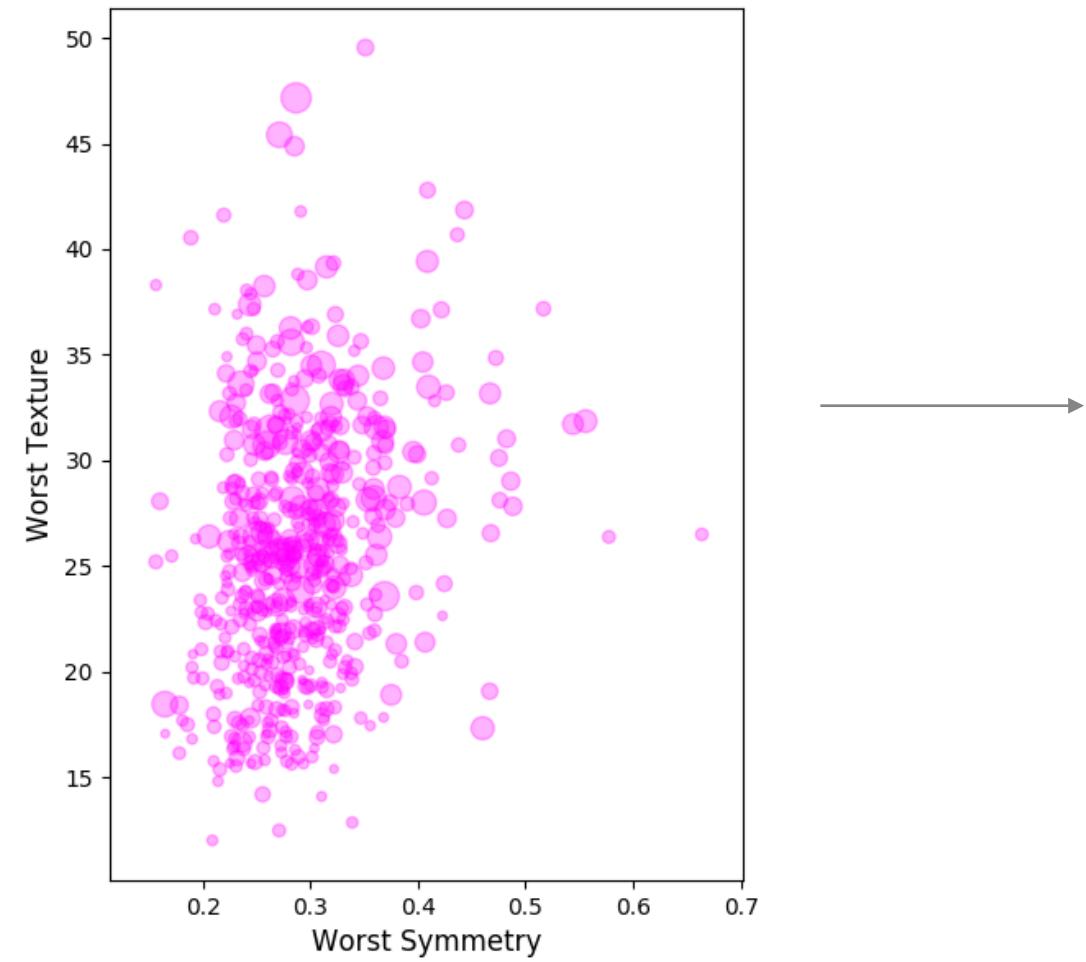
Los datos son proyectados a otro conjunto dimensional (su significado físico inicial desaparece)

# PCA es un método que justifica su efectividad mediante la aplicación de álgebra

Funciona por que:

- La matriz de covarianza explica las correlaciones entre las distintas características del dataset original
- Eigen-vectores: representan la direccionalidad de los datos
- Eigen-valores: representan la magnitud/importancia
- Se asume que a mayor varianza, mayor información se retiene para el modelo a generar

# E.j. Datos originales vs Datos transformados con PCA



Uniform manifold approximation and projection (UMAP) es una técnica nueva no-lineal para reducción de dimensiones

Se define una métrica de similaridad entre dos instancias

$$w(\mathbf{x}_i, \mathbf{x}_j) \stackrel{\text{def}}{=} w_i(\mathbf{x}_i, \mathbf{x}_j) + w_j(\mathbf{x}_j, \mathbf{x}_i) - w_i(\mathbf{x}_i, \mathbf{x}_j)w_j(\mathbf{x}_j, \mathbf{x}_i).$$

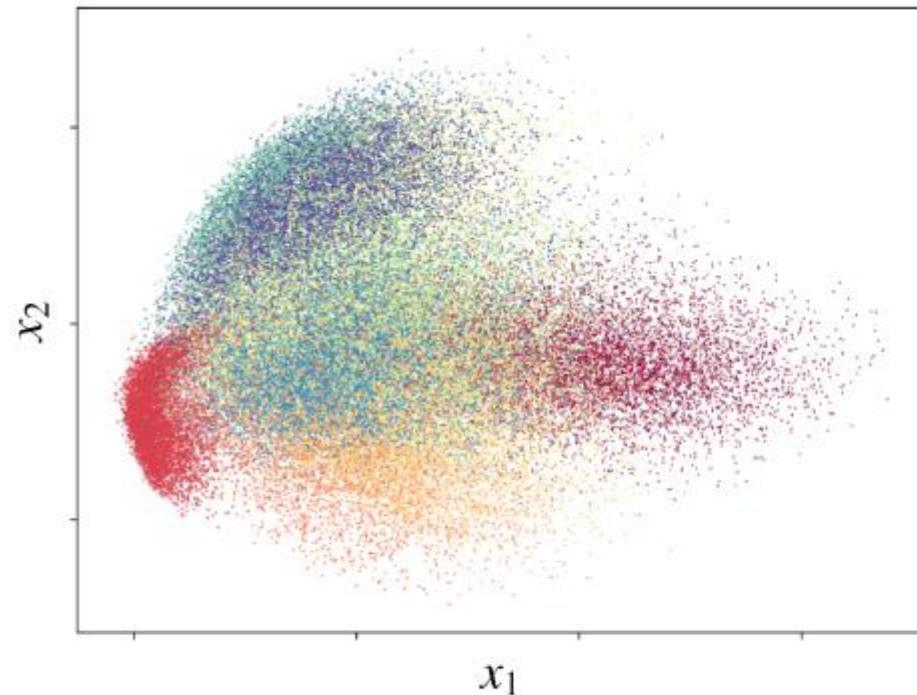
La función  $w_i(x_i, x_j)$  se define como:

$$w_i(\mathbf{x}_i, \mathbf{x}_j) \stackrel{\text{def}}{=} \exp\left(-\frac{d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i}{\sigma_i}\right)$$

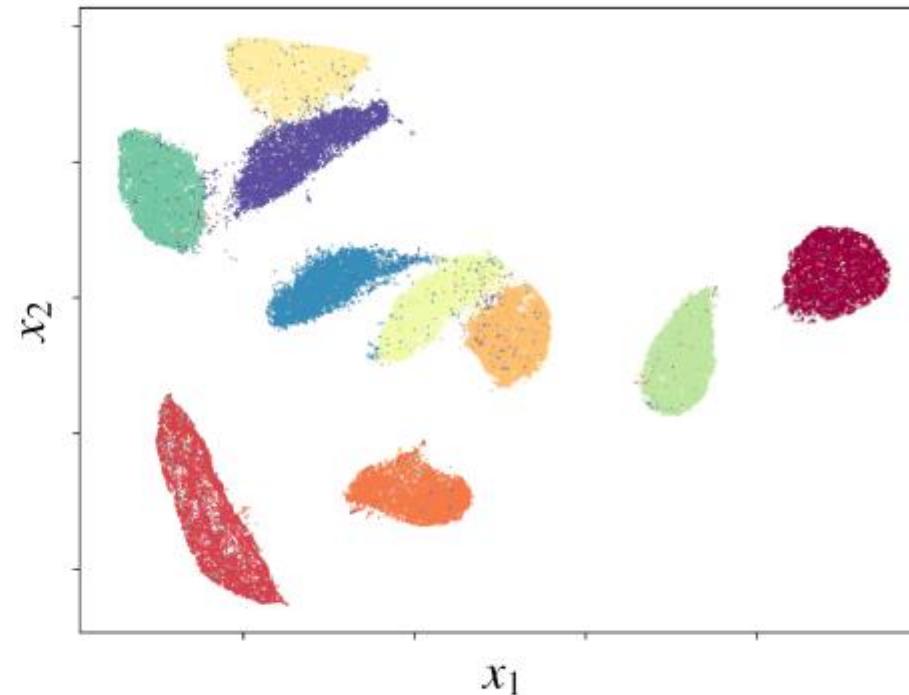
Donde  $d(x_i, x_j)$  es la distancia euclíadiana entre dos instancias,  $\rho_i$  la distancia desde  $x_i$  hasta su vecino más cercano y  $\sigma_i$  la distancia desde  $x_i$  a su  $k^{th}$  vecino más cercano

El algoritmo determina los valores de  $x'_i$   $i \in \{1..N\}$

# Comparando resultados de PCA y UMAP para el dataset de MINST

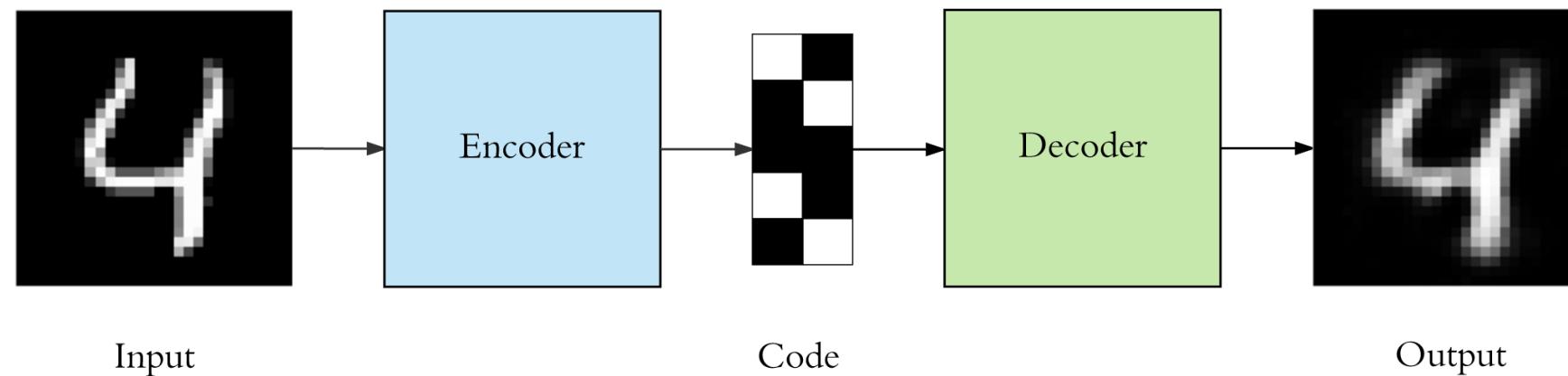


PCA



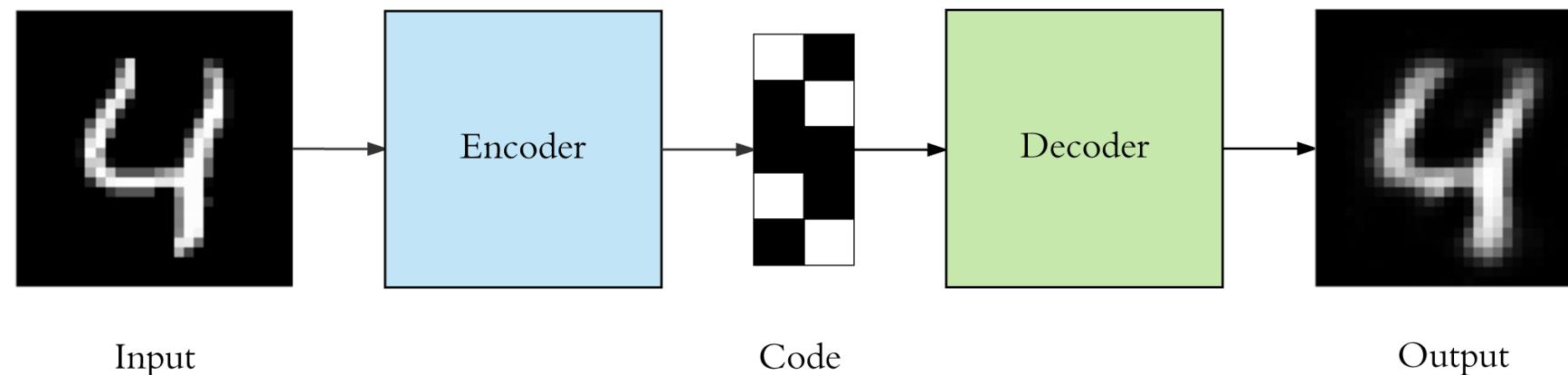
UMAP

**Auto-encoders** son redes feedforward en donde la entrada es la misma que la salida



Comprime la entrada en una dimensionalidad menor para luego reconstruir la entrada

**Auto-encoders** son redes feedforward en donde la entrada es la misma que la salida



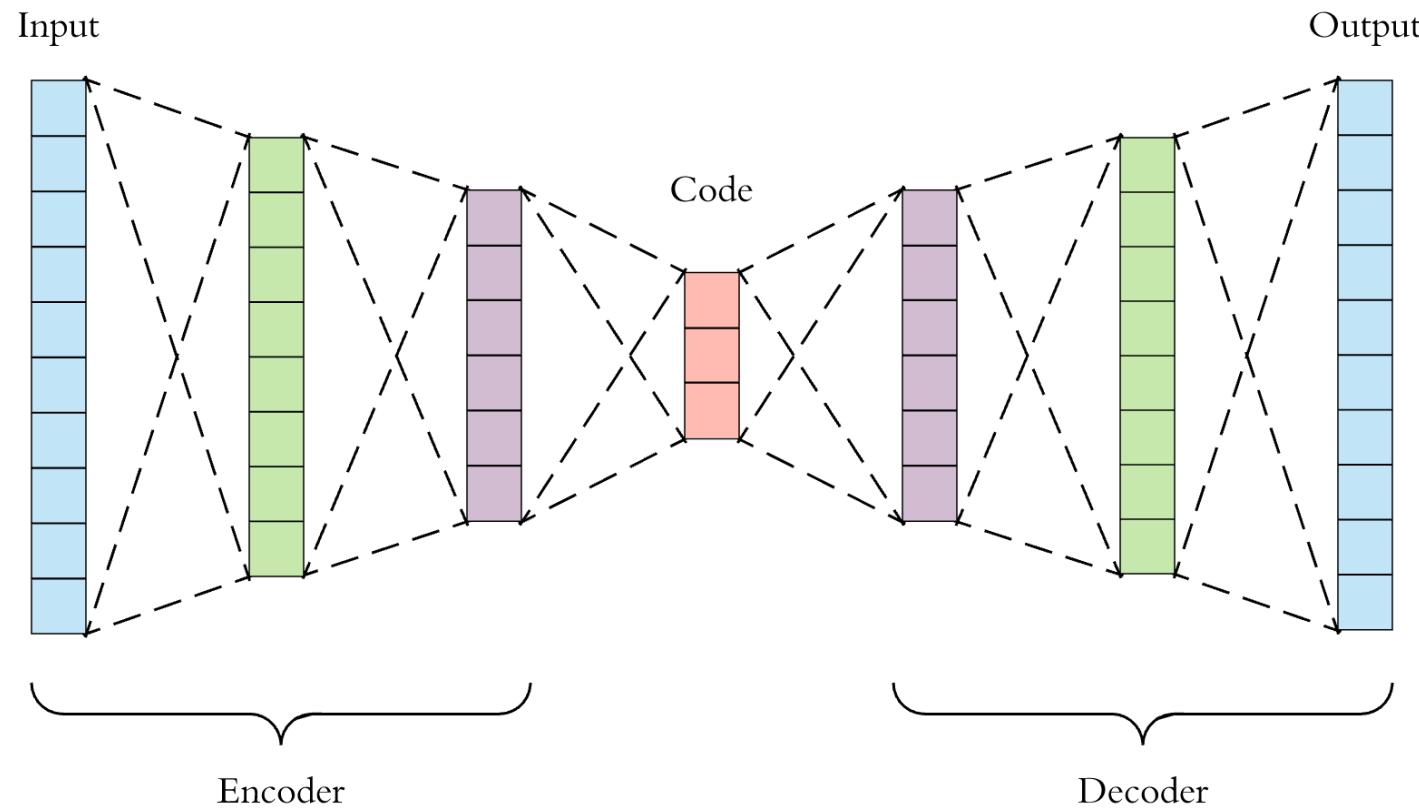
La dimensionalidad de la entrada comprimida se denomina  
“latent-space representation”

Un auto-encoder necesita: un método codificador, decodificador, y una función para comparar la salida con el valor real

Se utilizan mayormente para reducción de dimensiones (compresión)

- Funcionan con datos similares a los que fue entrenado
- La salida del auto-encoder no será exactamente igual a la entrada (denoising)
- Se consideran una técnica no supervisada por que solo hace uso de los datos de entrada (genera sus propias etiquetas)

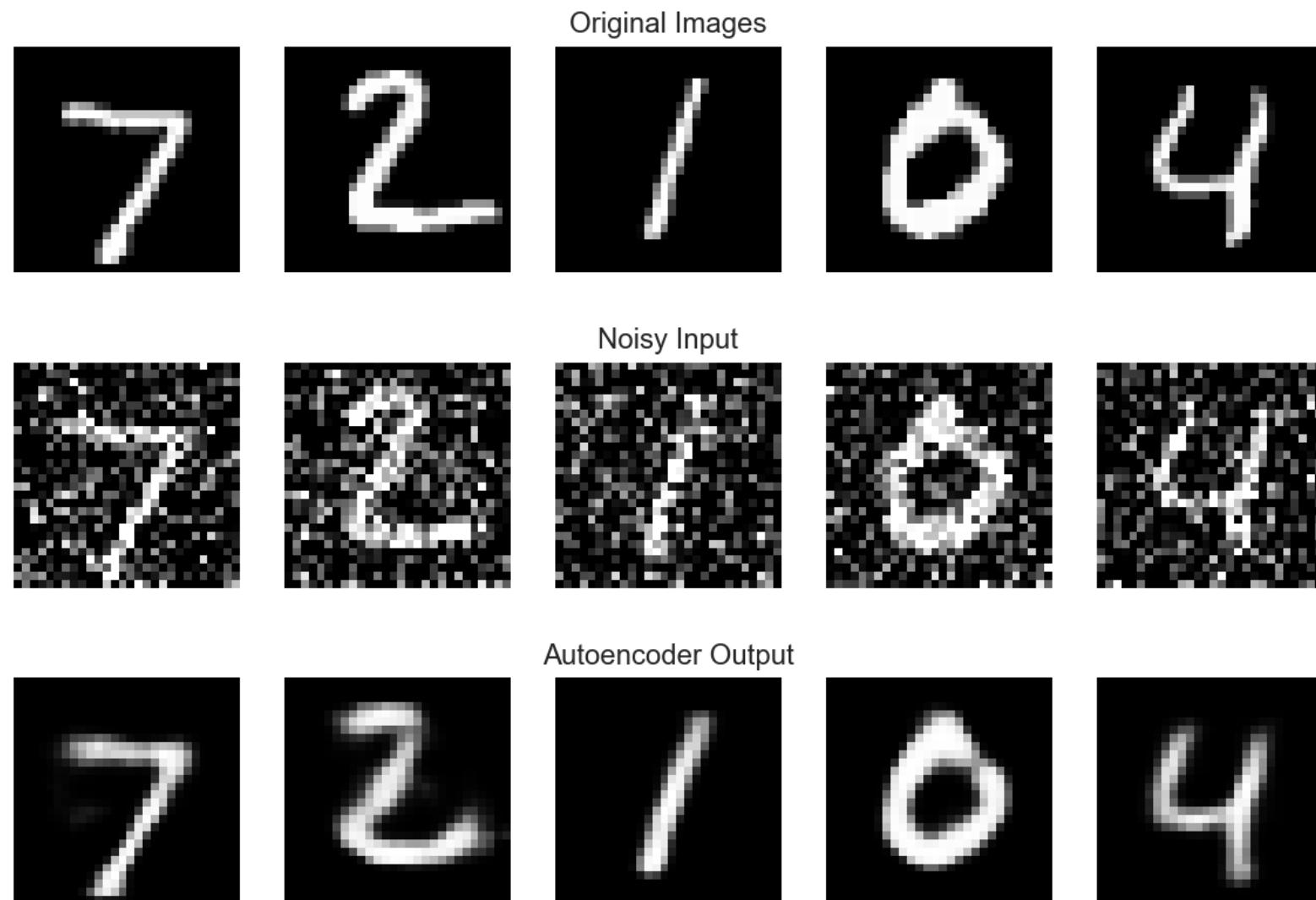
# La arquitectura de un auto-encoder es una red feedforward completamente conectada



## Hyper-parámetros:

- Tamaño del código
- Número de capas
- Número de nodos por capa
- Loss function (mse, binary cross entropy, etc.)

Los auto-encoders son muy utilizados para  
realizar *denoising* de datos





**UNIVERSIDAD DE CUENCA**  
*desde 1867*

# Aprendizaje No Supervisado

Andres Auquilla  
2020