

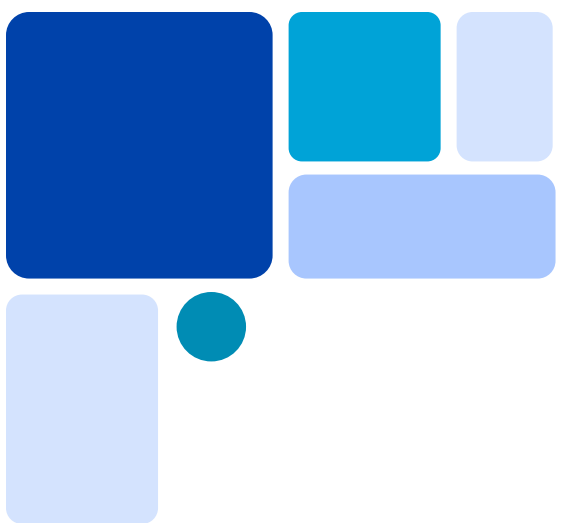
# Aprendizaje Automático

Dr. Hiram Eredín Ponce Espinosa






(diapositiva en blanco)



# Aprendizaje Supervisado

Aprendizaje Automático

- 
1. Aprendizaje Supervisado
  2. Regresión
  3. Estandarización de Datos
  4. Reducción de Dimensiones
  5. Árboles de Decisión
  6. Clasificación por Naïve Bayes
  7. Redes Neuronales Artificiales
  8. Maximización de la Esperanza

*Presentación basada en:*

- Rusell, S., Norvig, P., (2003), *Artificial Intelligence: A Modern Approach*, Prentice Hall/Pearson Education, New Jersey.
- Mitchell, T. (2003), *Machine Learning*, McGraw Hill, New York.
- Alpaydin, E. (2010), *Introduction to Machine Learning*, The MIT Press, England.

El aprendizaje tiene múltiples facetas. Entre ellas, cuando se presentan problemas en los cuales se conocen tanto ejemplos de cómo se resuelve, como las soluciones a las que se llegan, obteniendo así información importante de la cual aprender.

El **aprendizaje supervisado** consiste en llevar a cabo procesos de entrenamiento basados en pares ordenados de ejemplos, del tipo  $(x,y)$ , donde  $x$  representa los atributos mientras que  $y$  representa las etiquetas.



## Conjuntos

Para aprendizaje supervisado, se requieren de dos conjuntos de datos:

*datos de entrenamiento* (entrada-salida)

*datos de prueba* (entrada); se analiza con la *salida*

## Datos

Cada dato es un par ordenado (entrada-salida); conocido como *ejemplo* o *muestra*.

## Atributos, categorías o dimensiones

La entrada de un ejemplo es un vector de elementos llamados *atributos* o *dimensiones*:

$$\mathbf{x} = (x_1, \dots, x_n)$$

En aprendizaje, se le conoce como la representación *punto-de-datos* (*datapoint*)

## Etiquetas o valores

La salida de un ejemplo es un valor escalar o una etiqueta que define al dato:

$$y$$

## *Ejemplo:*

Una fábrica de computadoras lleva el registro de la velocidad del procesador, el tamaño de su memoria RAM y el número de ventiladores. Con un conjunto de datos similar, la empresa busca determinar cuáles de las computadoras tendrán un tiempo de vida menor a tres años y cuáles tendrán un tiempo de vida mayor o igual a tres años.

*Ejemplo:*

Una fábrica de computadoras lleva el registro de la velocidad del procesador, el tamaño de su memoria RAM y el número de ventiladores. Con un conjunto de datos similar, la empresa busca determinar cuáles de las computadoras tendrán un tiempo de vida menor a tres años y cuáles tendrán un tiempo de vida mayor o igual a tres años.

<i>ejemplo</i>	<i>velocidad de procesador (GHz)</i>	<i>tamaño de memoria RAM (GB)</i>	<i>número de ventiladores (#)</i>	<i>tiempo de vida (clase)</i>
1	2.3	4	4	mayor
2	2.1	8	3	menor
3	1.8	4	1	menor

*problema de clasificación*



*Ejemplo:*

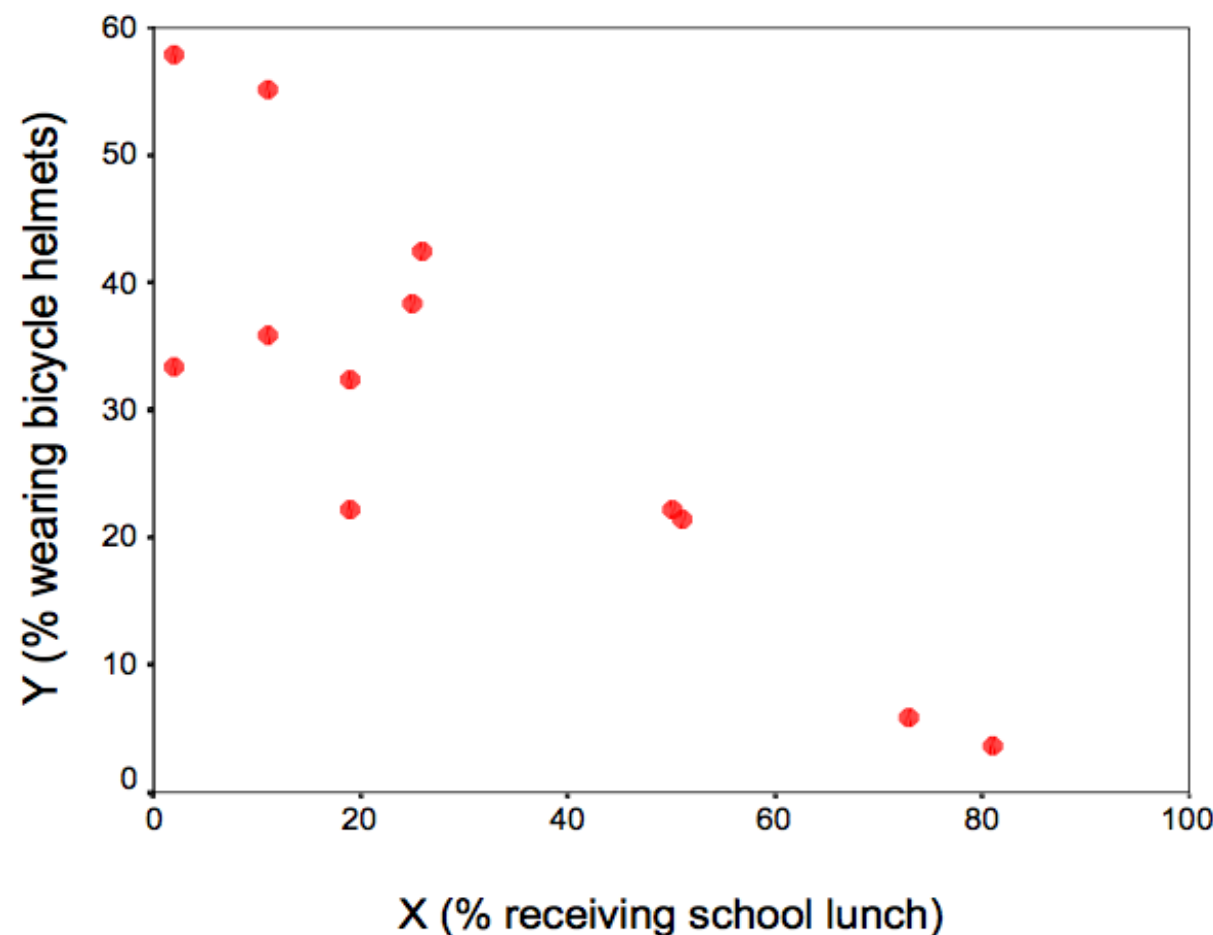
Una fábrica de computadoras lleva el registro de la velocidad del procesador, el tamaño de su memoria RAM y el número de ventiladores. Con un conjunto de datos similar, la empresa busca determinar cuáles de las computadoras tendrán un tiempo de vida menor a tres años y cuáles tendrán un tiempo de vida mayor o igual a tres años.

<i>ejemplo</i>	<i>velocidad de procesador (GHz)</i>	<i>tamaño de memoria RAM (GB)</i>	<i>número de ventiladores (#)</i>	<i>tiempo de vida (años)</i>
1	2.3	4	4	3.7
2	2.1	8	3	2.9
3	1.8	4	1	2.1

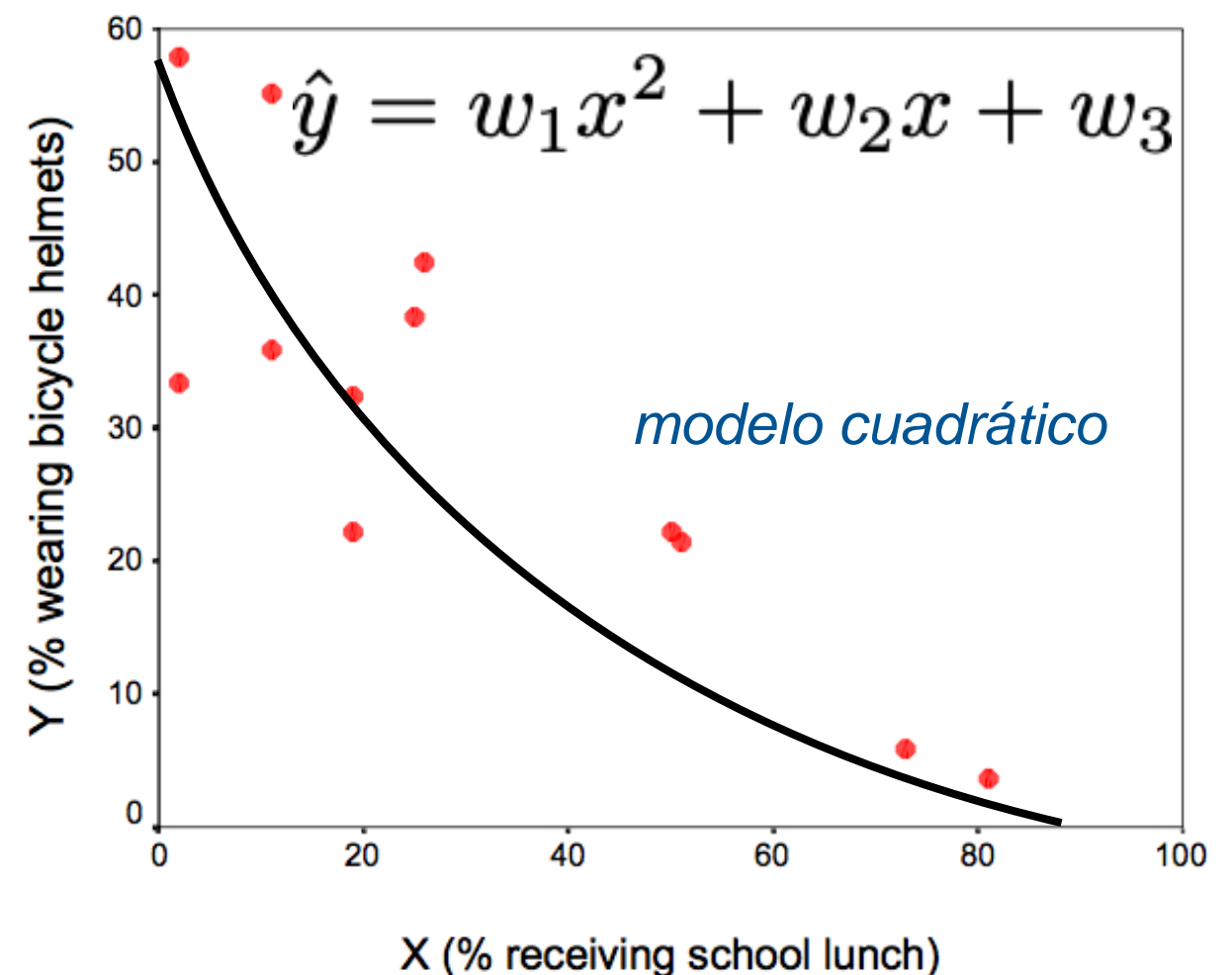
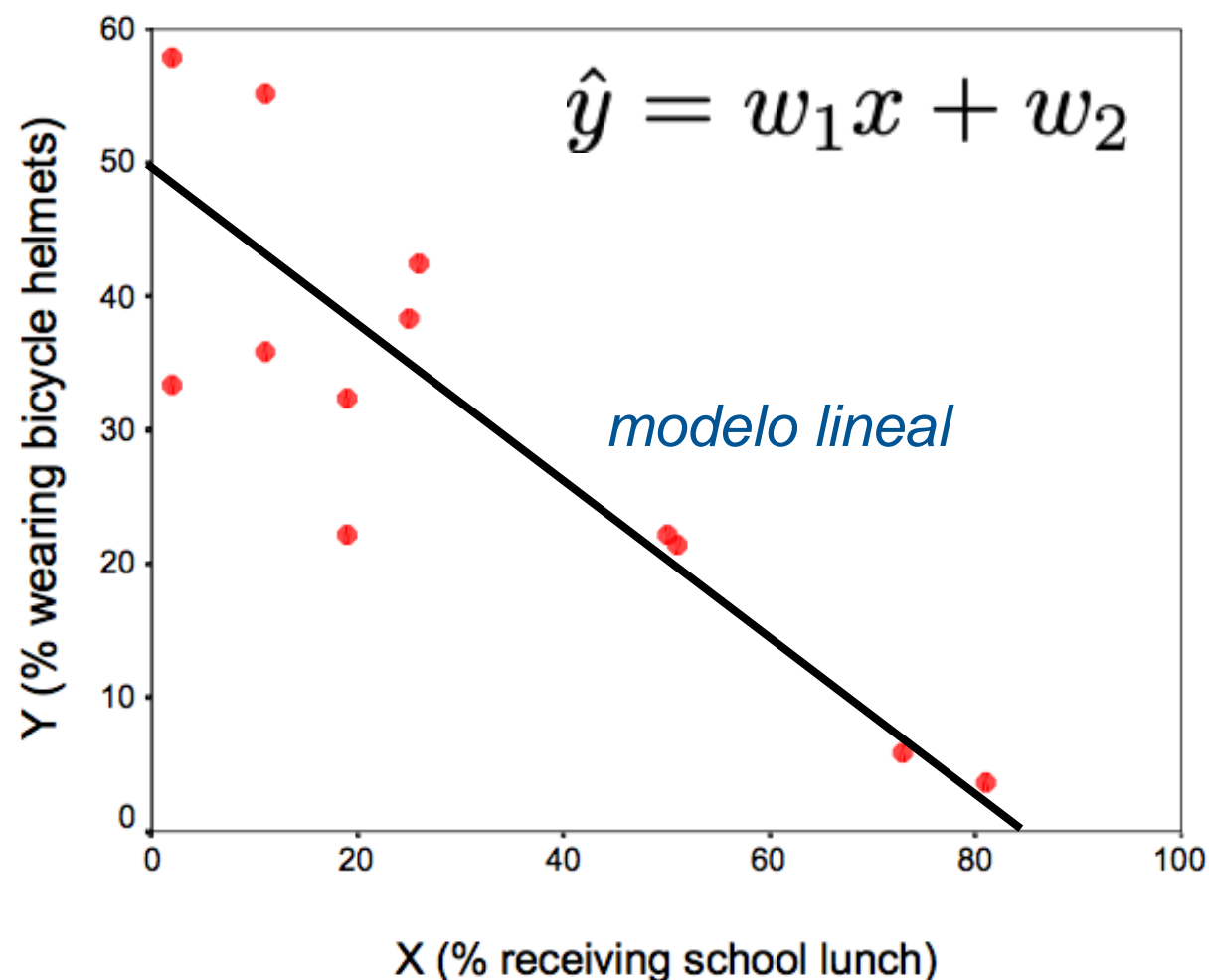
*problema de regresión*

La **regresión** es la tarea de aprender la transformación entre las entradas y las salidas de un conjunto de datos de entrenamiento, siempre que las salidas sean *valores numéricos*.

En otras palabras, la regresión permite encontrar un modelo del problema que se está planteando. No confundir con un problema de *interpolación* o *extrapolación*.



La regresión implica dos procesos:  
*definir el modelo de aproximación*  
*encontrar los parámetros del modelo*



Encontrar los parámetros del modelo implican un proceso de *optimización* (o *búsqueda*).

Se utiliza una función objetivo que evalúe el desempeño del modelo respecto de los valores reales (salidas del conjunto de entrenamiento). Por ejemplo:

$$E = \frac{1}{2} \sum_{i=1}^q (y_i - \hat{y}_i)^2$$

*error cuadrático*

$$\hat{y} = w_1 x^2 + w_2 x + w_3$$

Algunos algoritmos de entrenamiento:

- Estimación por mínimos cuadrados
- Algoritmos genéticos
- Hill-climbing
- Recocido simulado

$$E = \frac{1}{q} \sum_{i=1}^q (y_i - \hat{y}_i)^2$$

*error cuadrático medio*

La **compensación ajuste-varianza** (*bias-variance tradeoff*) es una negociación de minimización que se desea realizar en términos del error por ajuste y el error por varianza.

## Error por ajuste

Diferencia entre el valor real y el valor estimado por el modelo.

## Error por varianza

Es la variabilidad de la predicción del modelo dada una muestra de los datos, normalmente calculada a través de la desviación estándar del modelo.

## Error irreducible

Valores incorrectos en los datos debido al error intrínseco ocurrido en los modelos, notablemente definido como ruido.

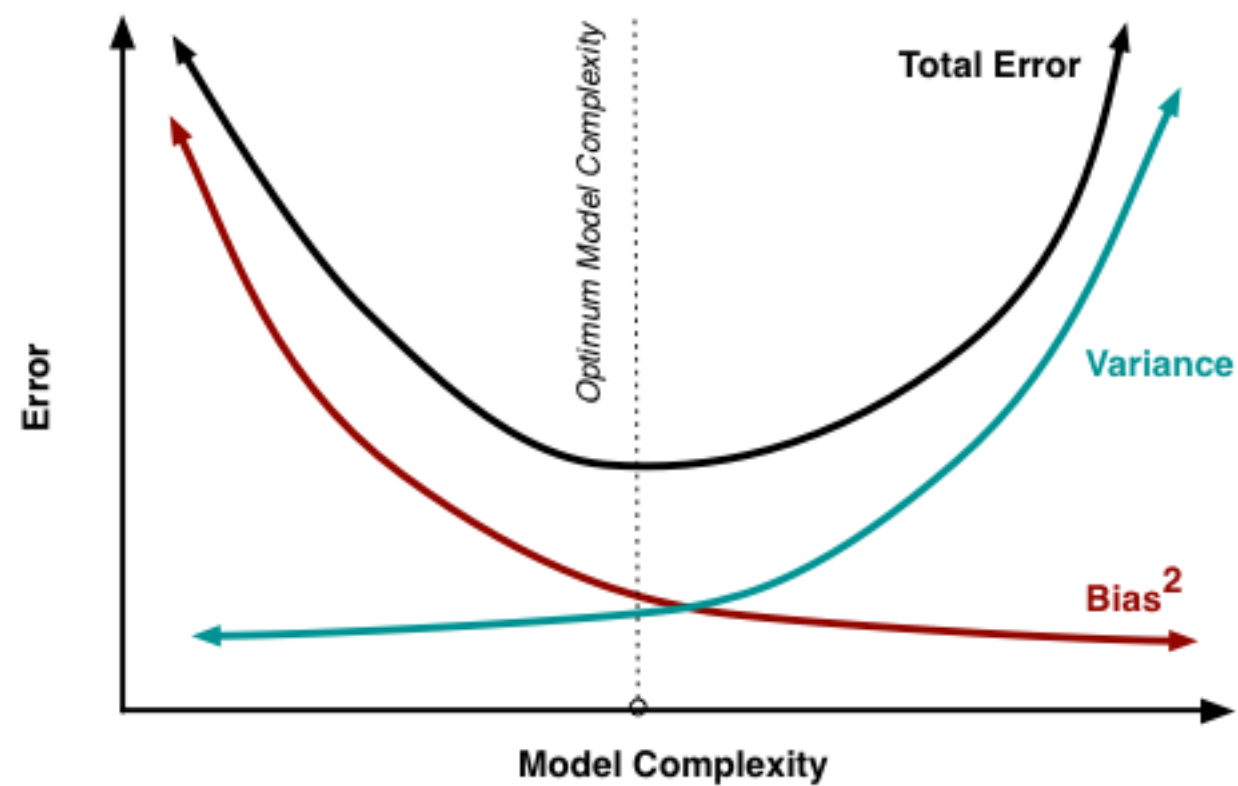
La **compensación ajuste-varianza** (*bias-variance tradeoff*) es una negociación de minimización que se desea realizar en términos del error por ajuste y el error por varianza.

Matemáticamente, se expresa como:

$$E = \underbrace{\frac{1}{k} \sum_i^k (y_i - \hat{y}_i)^2}_{\text{error por ajuste}} + \underbrace{\frac{1}{k} \sum_i^k (\hat{y}_i - \bar{\hat{y}})^2}_{\text{error por varianza}} + \eta$$

error irreducible

Gráficamente, se puede observar la siguiente tendencia:



## En MATLAB:

```
p = polyfit(x,y,k);    // devuelve los coeficientes del polinomio de grado  $k$   
ym = polyval(p,x);    // genera la regresión de los valores  $x$  con uso del polinomio  $p$ 
```

### *Ejemplo:*

```
x = 0 : 0.1 : 10;  
y = x + 2*rand(size(x));  
p = polyfit(x,y,1);  
ym = polyval(p,x);  
plot(x,y,'*',x,ym,'r*');
```



En problemas reales, los datos en los conjuntos de entrenamiento y de prueba pueden encontrarse dispersos, con magnitudes ampliamente diferentes, ruido, entre otros.

Sin embargo, la mayoría de los procesos de aprendizaje automático tienen suposiciones o condiciones bajo las que los datos deben encontrarse. Por lo tanto, se realizan *procesamientos previos* (*pre-procesamiento*).

Procesamientos previos comunes:

- *Limpieza de datos* - *valores atípicos*
- *Escalamiento de datos* - *normalización lineal*
- *Estandarización de datos* - *normalización uniforme*
- *Reducción de dimensiones*

## **Valores atípicos** (*outliers*)

Valores que se asumen fuera de la tendencia de distribución de los datos; normalmente, se obtiene por inspección o pruebas estadísticas.

## **Escalamiento** - *normalización lineal*

Ajuste de la magnitud de los datos para que estén definidos dentro de un intervalo específico. Ejemplo: regla de tres.

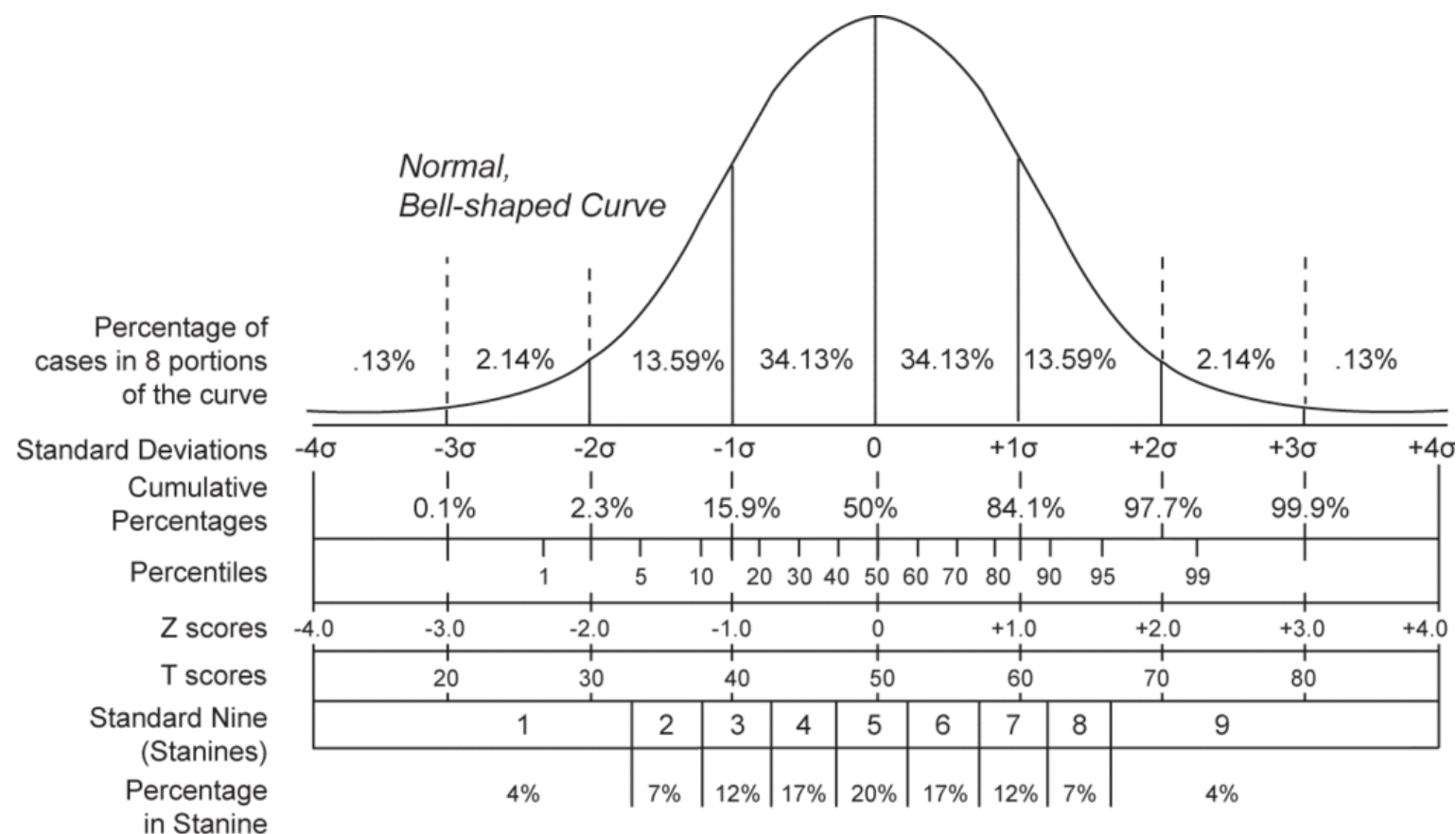
Si el proceso de escalamiento define al conjunto de datos dentro del intervalo  $[0, 1]$ , entonces se conoce como *normalización unitaria*.

$$z_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

## **Estandarización** - *normalización uniforme*

Ajuste de la magnitud de los datos para que estén definidos mediante una distribución uniforme.

$$z_i = \frac{x_i - \mu}{\sigma}$$



Un problema común en aprendizaje automático es el excesivo número de atributos o dimensiones que tiene un conjunto de datos; dicho problema se conoce como la *maldición de la dimensionalidad* (*curse of dimensionality*).

Para resolver este problema, se han generado algoritmos de reducción de dimensionalidad.

Se asume que cada punto en un vector de  $n$  dimensiones; por lo tanto, una reducción de dimensionalidad es una transformación de  $n$  dimensiones a  $k$  dimensiones (para  $k < n$ ).

$$RD : \mathbb{R}^n \rightarrow \mathbb{R}^k$$

Normalmente, se lleva a cabo mediante proyecciones espaciales.

El **análisis de componente principal** (*principal components analysis; PCA*) es una transformación ortogonal sobre un conjunto de datos posiblemente correlacionados que deriva en un conjunto de datos de atributos linealmente no correlacionados.

Surge de la suposición de que los datos obtenidos tienen influencia de ciertos atributos y por lo tanto pueden estar correlacionados (hay patrones implícitos).

## Componentes principales

Atributos no correlacionados en la transformación.

Los componentes son ortogonales entre sí.

Cada componente es una combinación lineal de los atributos originales.

Cada componente es generado maximizando la variabilidad de los datos proyectados sobre un eje ortogonal.

El conjunto de componentes principales es una base ortogonal en la transformación que no posee información redundante.

Un componente principal tiene:

- w** **coeficientes, pesos o cargas** (coeficientes de la combinación lineal)
- t** **puntuación** (datos transformados)
- Q** **varianza** (estadístico que mide cuánto un componente explica la variabilidad)

Un componente principal se obtiene a través de la optimización siguiente:

*Primer componente principal*

$$\mathbf{w}_{(1)} = \arg \max_{||\mathbf{w}||=1} \left\{ \sum_i (t_1)_{(i)}^2 \right\} = \arg \max_{||\mathbf{w}||=1} \{ ||\mathbf{X}\mathbf{w}||^2 \}$$

$$t_{k(i)} = \mathbf{x}_{(i)} \cdot \mathbf{w}_{(i)}$$

Un componente principal se obtiene a través de la optimización siguiente:

*Los demás componentes principales*

$$\mathbf{w}_{(k)} = \arg \max_{\|\mathbf{w}\|=1} \{ \|\mathbf{X}_{k-1} \mathbf{w}\|^2 \}$$

$$\mathbf{X}_{k-1} = \mathbf{X} - \sum_{i=1}^{k-1} \mathbf{X} \mathbf{w}_{(i)} \mathbf{w}_{(i)}^T$$

Al final, el análisis de componente principal se obtiene mediante:

$$\mathbf{T} = \mathbf{X} \mathbf{W}$$

Usando MATLAB, el análisis de componentes principales se puede llevar a cabo:

$$[\text{coefs}, \text{scores}, \text{variances}] = \text{princomp}(\mathbf{X})$$

Los datos están en  $\mathbf{X}$ :

Los renglones representan una muestra u observación

Las columnas representan los atributos

Los coeficientes **coefs**:

Los renglones representan los coeficientes por cada atributo original

Las columnas representan los componentes principales (ordenados del mejor al peor)

Las puntuaciones **scores**:

Los renglones representan una muestra u observación en el nuevo sistema

Las columnas representan componentes (nuevos atributos)

Las varianzas **variances**:

El vector representa la varianza de cada componentes principal



Usando MATLAB, el análisis de componentes principales se puede llevar a cabo:

$$[\text{coefs}, \text{scores}, \text{variances}] = \text{princomp}(X)$$

Para determinar el número de componentes principales a elegir, se utiliza la varianza acumulada:

$$v = \text{cumsum}(\text{variances}) ./ \text{sum}(\text{variances})$$

Elegir las primeras componentes principales que cubran aproximadamente entre el 80% y 90% de la variabilidad de los datos.

Normalmente, las primeras 2 o 3 componentes pueden representar correctamente los datos.

En aprendizaje supervisado existen muchas técnicas automáticas que realizan el proceso de encontrar un modelo que satisfaga el problema de aprendizaje.

Las técnicas que se verán en este curso son:

- *Árboles de decisión*
- *Clasificador de Naïve Bayes*
- *Redes neuronales artificiales*
- *Maximización de la esperanza*

Otras técnicas:

- *Máquinas de soporte vectorial*
- *Redes Bayesianas*
- *Redes orgánicas artificiales*

Los **árboles de decisión** (*decision trees*) es un método de aprendizaje supervisado para aproximar valores discretos dados, en donde la función aprendida está representada en un árbol con nodos condicionales (de decisión).

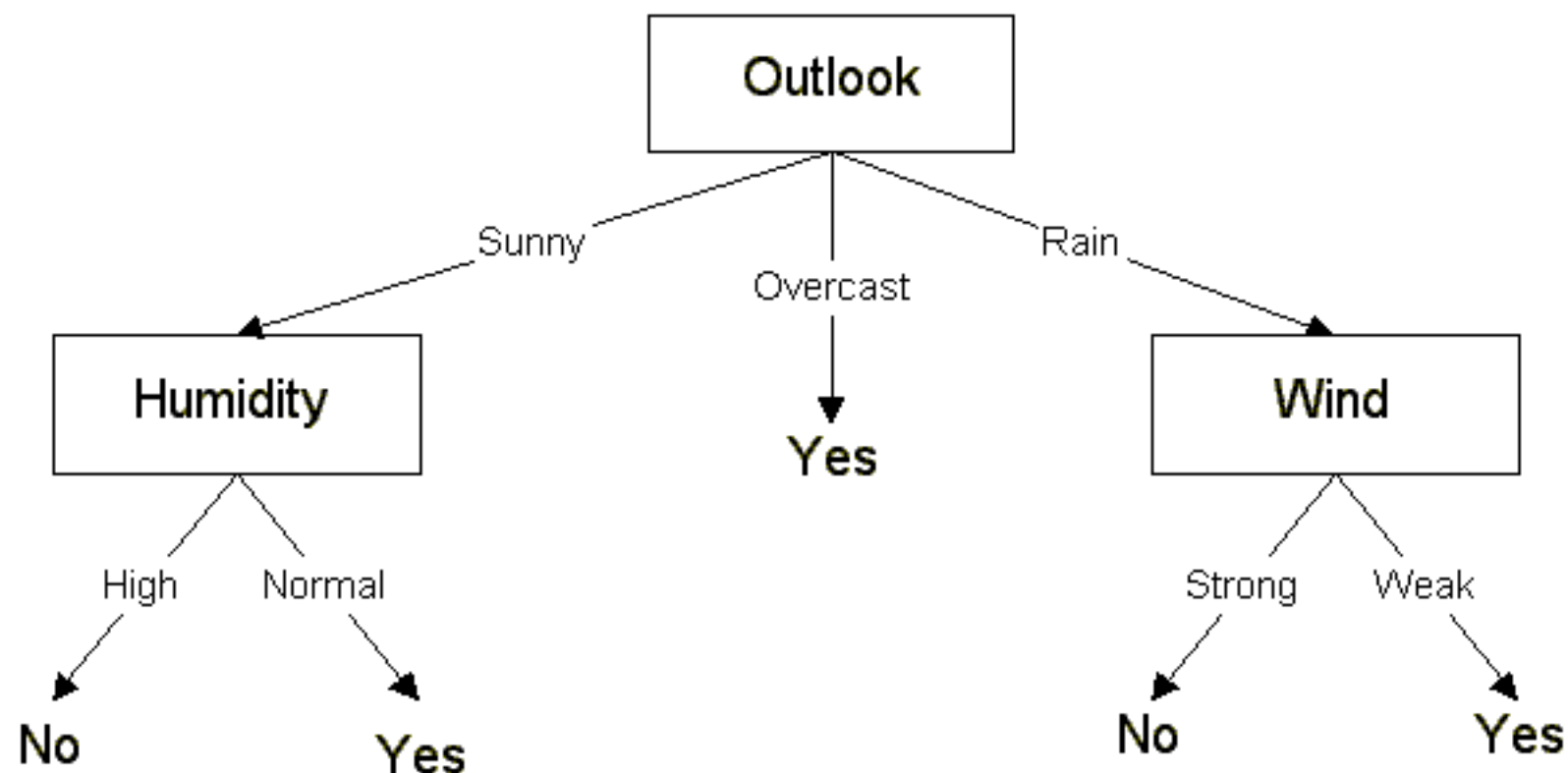
Existen dos algoritmos básicos:

- ID3 Para valores discretos en los valores meta
- C4.5 Para valores continuos en los valores meta

Los árboles están generados bajo los siguientes supuestos:

- Cada nodo es un atributo.
- Ningún atributo puede repetirse.
- Cada rama de los atributos es una categoría de los mismos.
- Los nodos hoja son las categorías de la variable meta.
- Cada rama es una conjunción de instancias de atributos.
- El árbol en sí es una disyunción de las conjunciones.
- Se asumen valores discretos (para ID3).
- Los datos de entrenamiento pueden contener valores de atributos incompletos.

Ejemplo de un árbol de decisión:



$$(Outlook = Sunny \wedge Humidity = Normal) \vee (Outlook = Overcast) \vee (Outlook = Rain \wedge Wind = Weak)$$

La clave de los árboles de decisión es determinar qué atributo será el nodo raíz, por lo que se ideó el concepto de **ganancia de información** basado en la **entropía** del sistema.

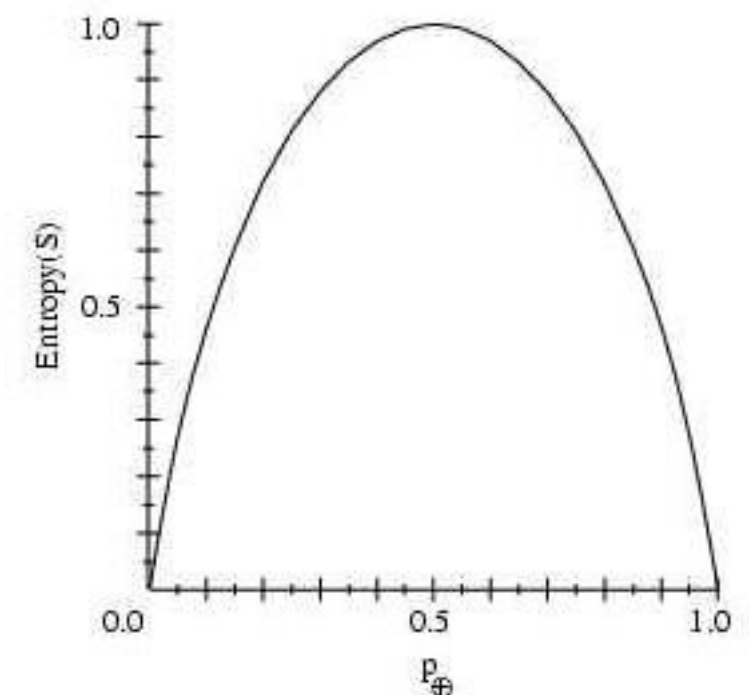
La **entropía** se puede definir como la medida de impureza en una colección arbitraria de muestras.

Para una colección  $S$  (*Clase1*, ... , *ClaseC*), la entropía relativa se define como:

$$Entropy(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

$$0 \log_2 0 \equiv 0$$

Donde,  $p_i$  es la proporción de datos de la categoría  $i$ .



La **ganancia de información** mide la efectividad de un atributo para clasificar un conjunto de datos de entrenamiento.

Otra forma de ver a la ganancia de información es como la reducción esperada en la entropía del sistema  $S$  causada por la partición de datos basada en un atributo  $A$ :

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Se calcula la ganancia de información para cada atributo y se selecciona como nodo raíz al atributo con mayor valor de dicha ganancia.

Ejemplo:

Determinar el nodo raíz a partir de los siguientes datos de entrenamiento.

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No



## ID3(*Ejemplos*, *Metas*, *Atributos*)

Crear un *NodoRaíz*.

Si todos los *Ejemplos* son positivos, regresar el *NodoRaíz* con valor *positivo*.

Si todos los *Ejemplos* son negativos, regresar el *NodoRaíz* con valor *negativo*.

Si *Atributos* está vacío, regresar el *NodoRaíz* con valor igual al valor más común de *Metas* en *Ejemplos*.

En otro caso:

Asignar a *A* el atributo de *Atributos* que mejor clasifique a *Ejemplos* (usar ganancia de información).

Asignar a *NodoRaíz* el atributo *A*.

Por cada categoría *vi* de *A*, hacer:

Añadir al árbol una rama bajo el *NodoRaíz* que corresponda con el valor *vi* de *A*.

Hacer *Ejemplos\_vi* un subconjunto de *Ejemplos* que contengan el valor *vi* de *A*.

Si *Ejemplos\_vi* está vacío:

Bajo la nueva rama añadir un nodo hoja con valor igual al valor más común de *Metas* en *Ejemplos*.

En otro caso:

Bajo la nueva rama añadir un subárbol del tipo ID3(*Ejemplos\_vi*, *Metas*, *Atributos* - {*A*}).

Terminar

El **clasificador de Naïve Bayes** (*Naïve Bayes Classifier*) es un método de aprendizaje supervisado para modelar un clasificador probabilístico que sea capaz de predecir a qué categoría pertenece un dato expresado en sus atributos (variables predictoras); asumiendo que todos los atributos son independientes entre sí.

Matemáticamente, el modelo predictor se expresa como:

$$p(y|x_1, \dots, x_n) = \frac{1}{Z} p(y) \prod_{i=1}^n p(x_i|y)$$

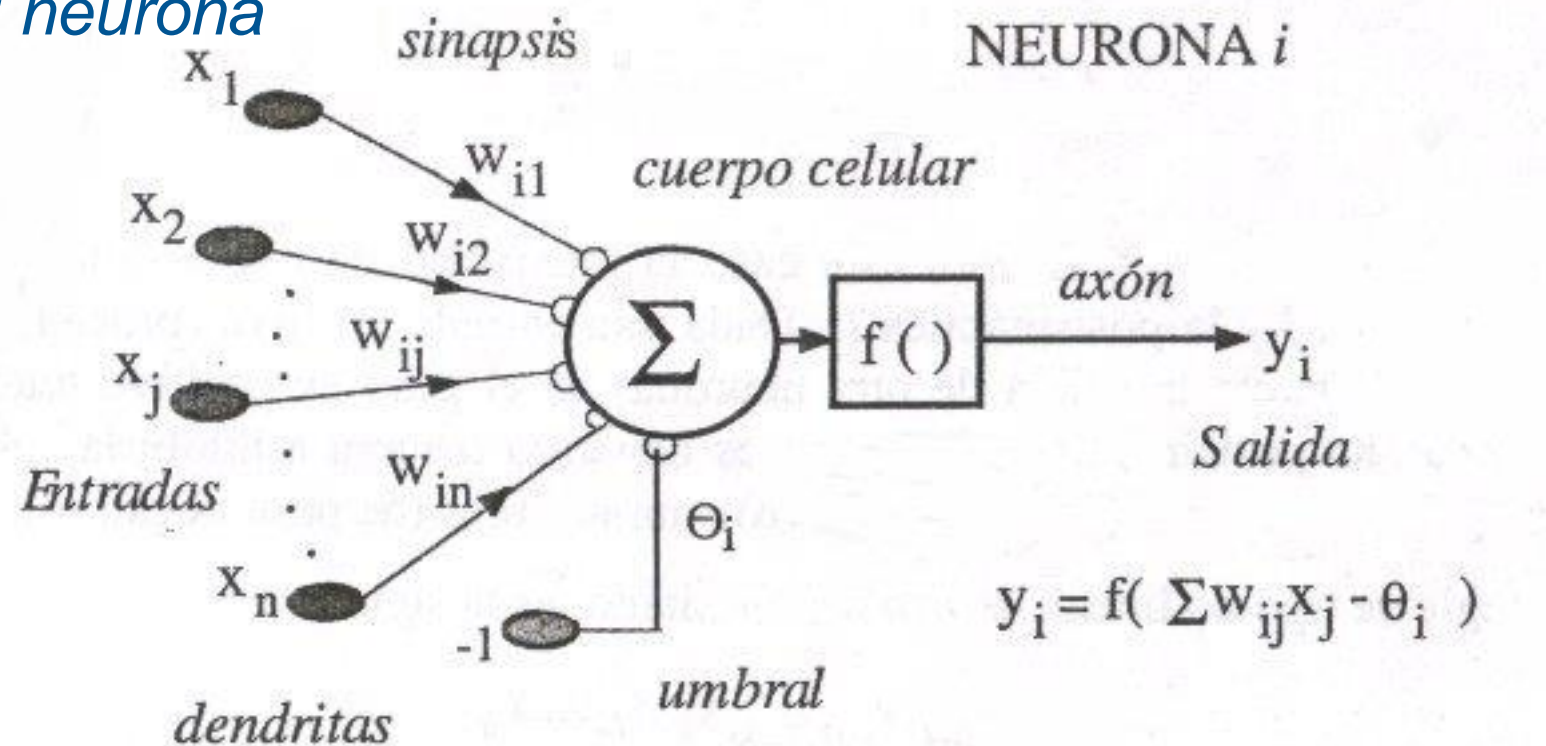
las probabilidades se obtienen de los datos de  
entrenamiento

Para predecir si un dato pertenece a una clase en particular, no es necesario el conocimiento de todos los atributos para obtener una aproximación.

## Redes neuronales artificiales

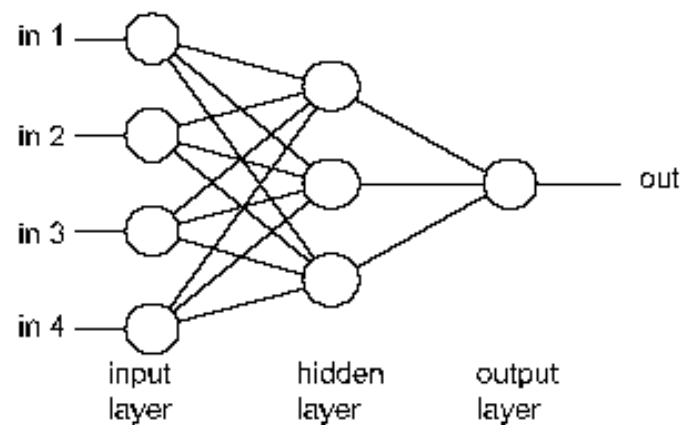
Técnica de aprendizaje automático e inteligencia computacional (*soft computing*) que simula la adquisición de conocimiento de las redes neuronales biológicas del ser humano.

### Unidad básica: neurona

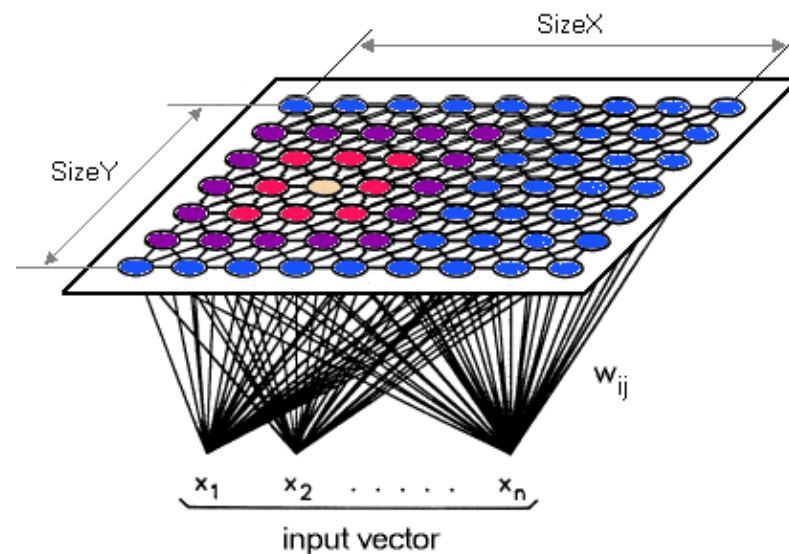


## Redes neuronales artificiales

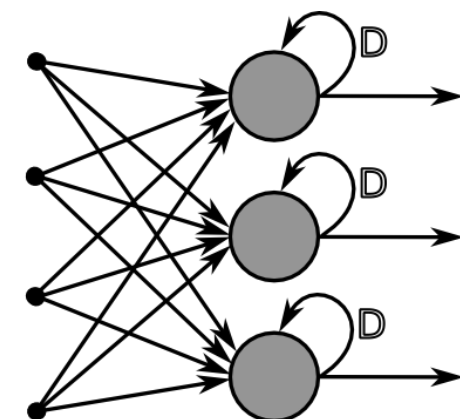
### *Distintos tipos de redes*



*redes multicapa*



*redes auto-organizadas*



*redes recurrentes*

## Redes neuronales artificiales

Dependiendo de las características de las redes neuronales artificiales, se utilizan distintas técnicas para obtener los parámetros de la red neuronal para que se obtenga una hipótesis adecuada. A dichas técnicas se le conocen como *algoritmos de entrenamiento*.

La primera neurona creada fue el perceptron; el cual incluye una función de activación de discriminación o umbral:

$$f(x) = \begin{cases} 1 & \text{si } w \cdot x - u > 0 \\ 0 & \text{en otro caso} \end{cases}$$

Por lo tanto, el algoritmo de entrenamiento busca encontrar el conjunto de pesos que permitirán que la salida de la neurona clasifique de manera adecuada un conjunto de datos lineales.

## Redes neuronales artificiales

El algoritmo de entrenamiento de un perceptron se conoce como la *regla delta*:

1. Inicializar los pesos con valores pequeños
2. Por cada ejemplo:
  - 2.1 Si el valor aproximado es distinto al valor real:
    - 2.1.1 Obtener el valor delta (cambio) de cada peso
    - 2.1.2 Actualizar los pesos
3. Volver al paso (2) mientras alguno de los valores aproximados sea distinto del real

## Redes neuronales artificiales

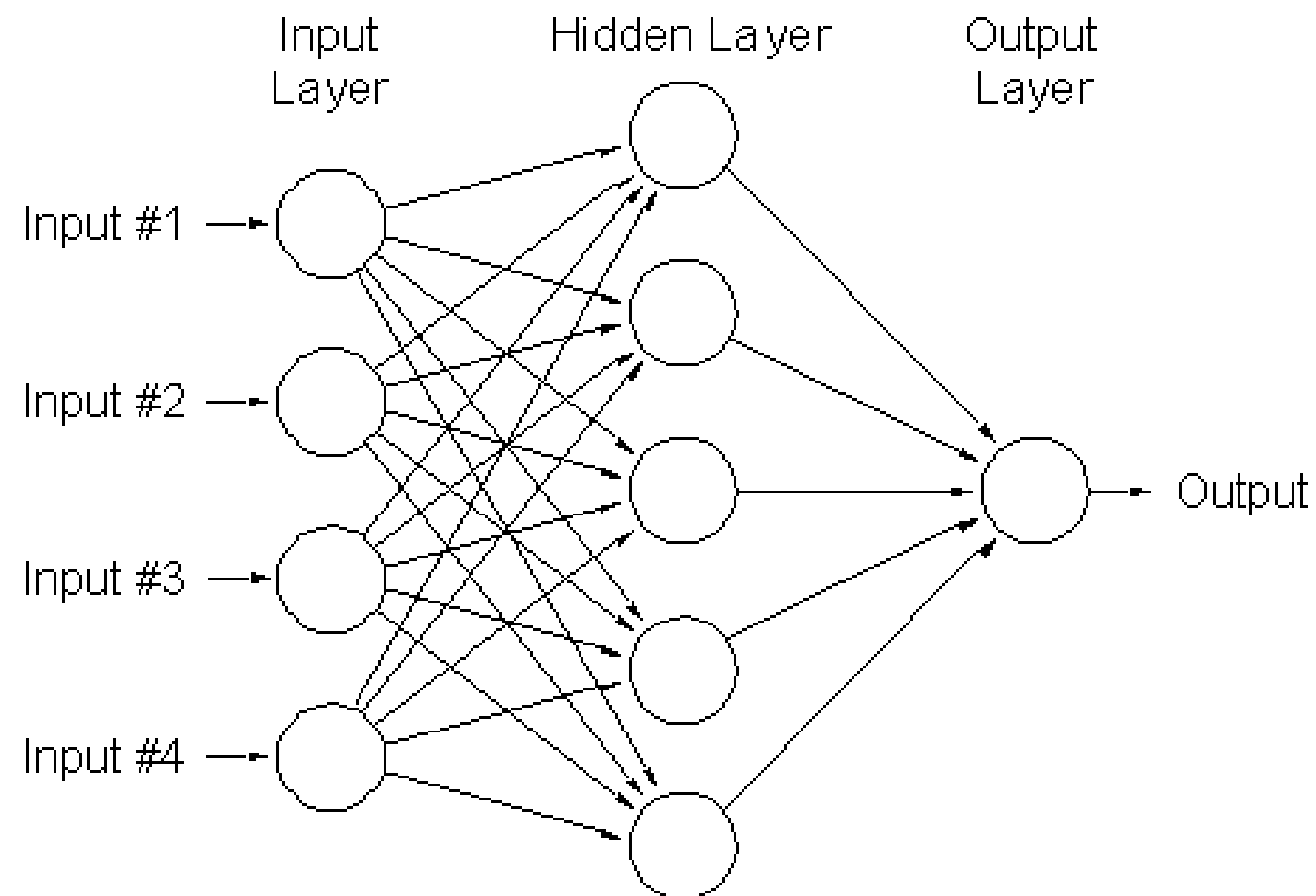
Las redes con perceptrones resultan complicadas de entrenar, por lo que se propusieron las redes neuronales multicapa con funciones de activación lineales (ADALINE) y posteriormente con funciones de *activación continuas*.

El algoritmo de entrenamiento más simple para redes neuronales artificiales multicapa con alimentación hacia delante es el conocido como *retro-propagación* (*backpropagation*).

Generaliza la regla delta para funciones continuas (comúnmente lineales y sigmoidales).

## Redes neuronales artificiales

*Ejemplo de una red neuronal artificial multicapa con alimentación hacia delante:*





## Redes neuronales artificiales (MATLAB)

```
net = newff(X, Y, [Si], {Ti})
```

Inicializa la estructura de la red neuronal

X: matriz  $A \times M$  de valores de entrada; A: número de atributos; M: número de ejemplos

Y: matriz  $E \times M$  de valores de salida; E: número de salidas; M: número de ejemplos

[Si]: vector del número de neuronas por capa (exceptuando la capa de salida)

{Ti}: celda de los tipos de función de activación por capa incluyendo la de salida.

```
net.trainparam.max_fail
```

Define el número máximo de fallas antes de detener el algoritmo

```
net.trainparam.min_grad
```

Define el valor mínimo de tolerancia en el gradiente

```
net = train(net,X,Y)
```

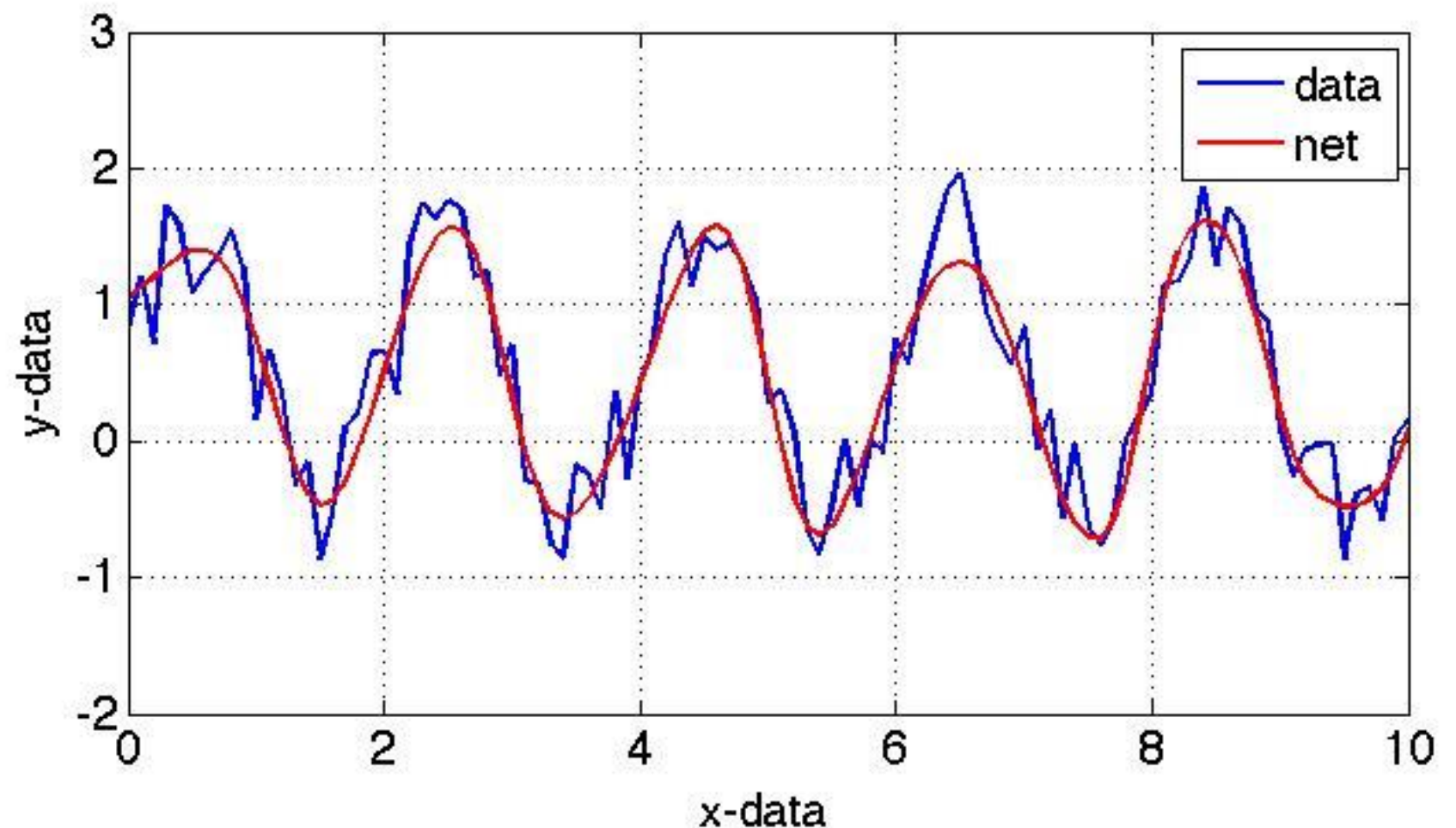
Entrena la red neuronal

```
yModel = sim(net,X)
```

Método de inferencia de valores provisto por la red neuronal entrenada

## Redes neuronales artificiales (MATLAB)

```
x = 0:0.1:10;  
y = sin(pi*x) + rand(size(x))  
net = newff(x, y, 10);  
net = train(net,x,y);  
ym = sim(net,x);  
plot(x,y,'b',x,ym,'r');
```

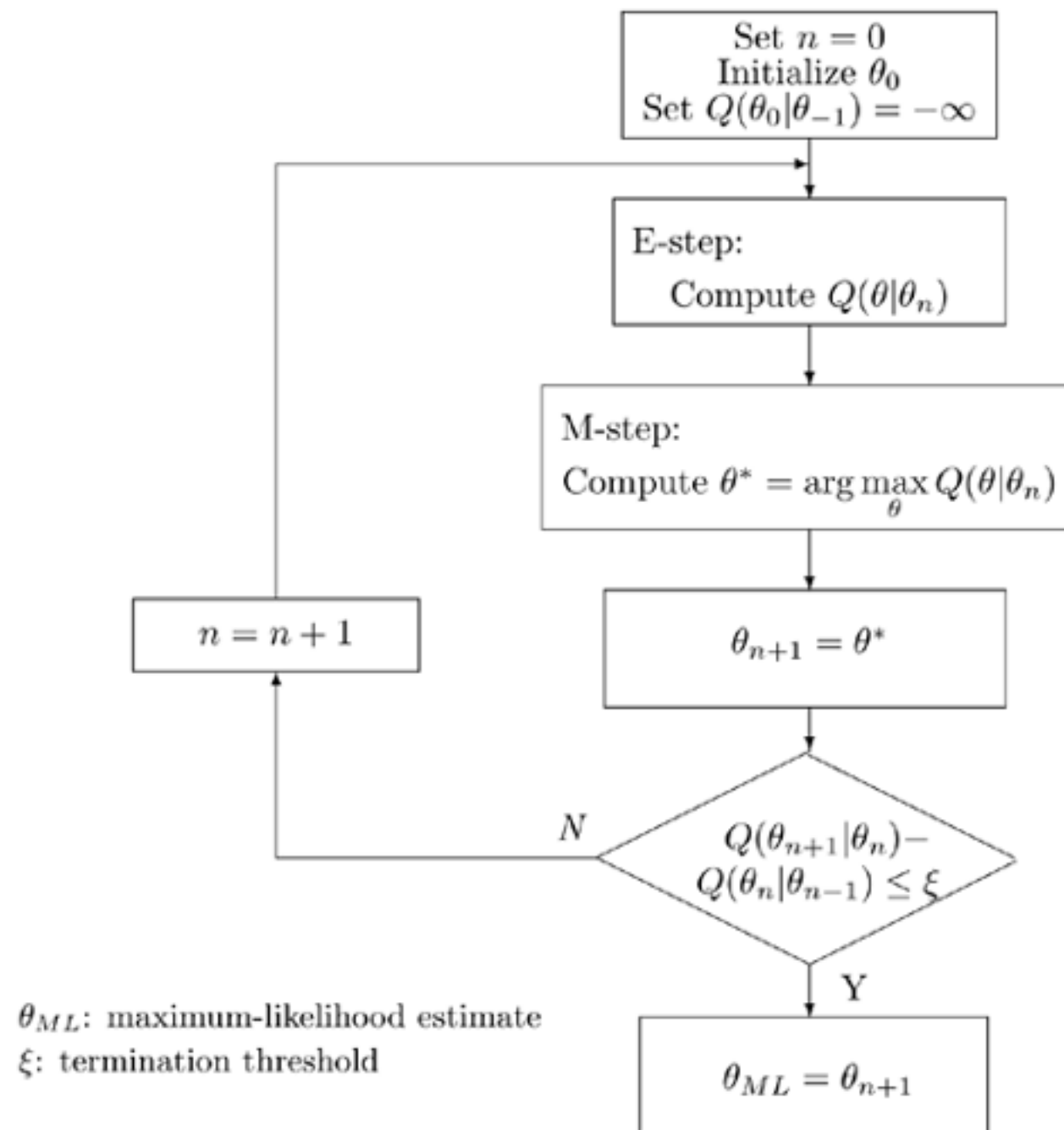


El algoritmo de **maximización de la esperanza** (*expectation-maximization*) se emplea para el entrenamiento, normalmente no supervisado, de otros algoritmos más robustos.

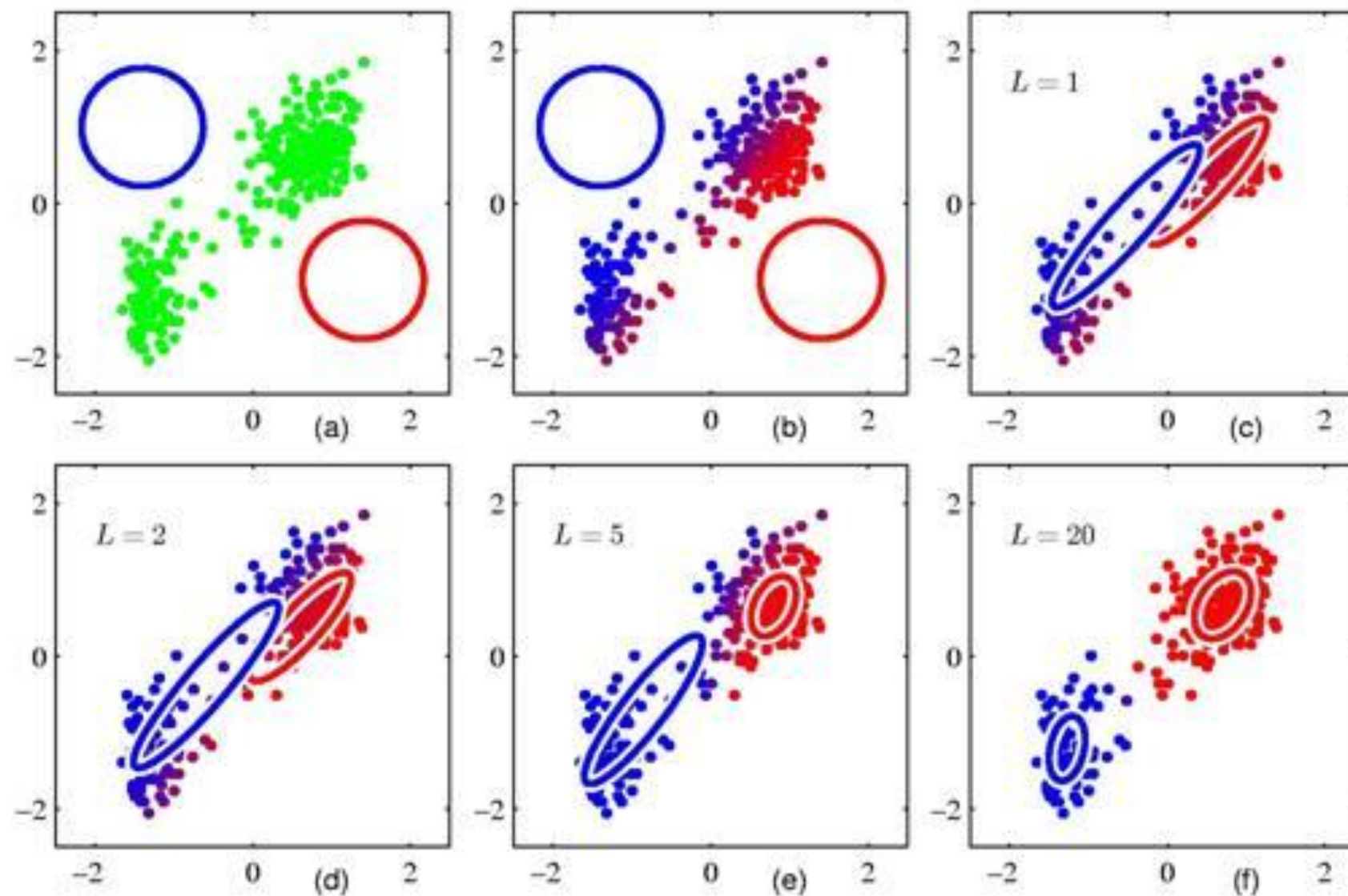
Algoritmo de aprendizaje que propone la solución al tema de la ausencia de datos de entrenamiento con variables de salida.

Es un algoritmo iterativo con convergencia en el infinito y se basa en dos pasos:

- **Esperanza:** *obtiene las medias de los parámetros ocultos en el sistema*
- **Maximización:** *utiliza los valores nuevos para optimizar la función objetivo*



*diagrama de flujo del algoritmo EM*



*ejemplo del algoritmo EM en k-means*

El algoritmo de **modelos de mezclas Gaussianas** (*Gaussian mixture models*) es un algoritmo de aprendizaje no supervisado que determina el mejor conjunto de distribuciones Gaussianas que interpretan a los datos, con la finalidad de entender cómo han sido generado dichos datos, ahora mezclados.

Las variables ocultas que buscan los modelos de mezclas Gaussianas son las medias y desviaciones estándar de cada clase-conjunto que representa a los datos.

Matemáticamente, las mezclas Gaussianas se expresan como:

$$p(x_1, \dots, x_n | w, \mu, \sigma) = \sum_{i=1}^M w_i g(x_1, \dots, x_n | \mu_i, \sigma_i)$$

$$g(x | \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp - \frac{(x - \mu_i)^2}{2\sigma_i^2}$$



# Modelos de mezclas Gaussianas

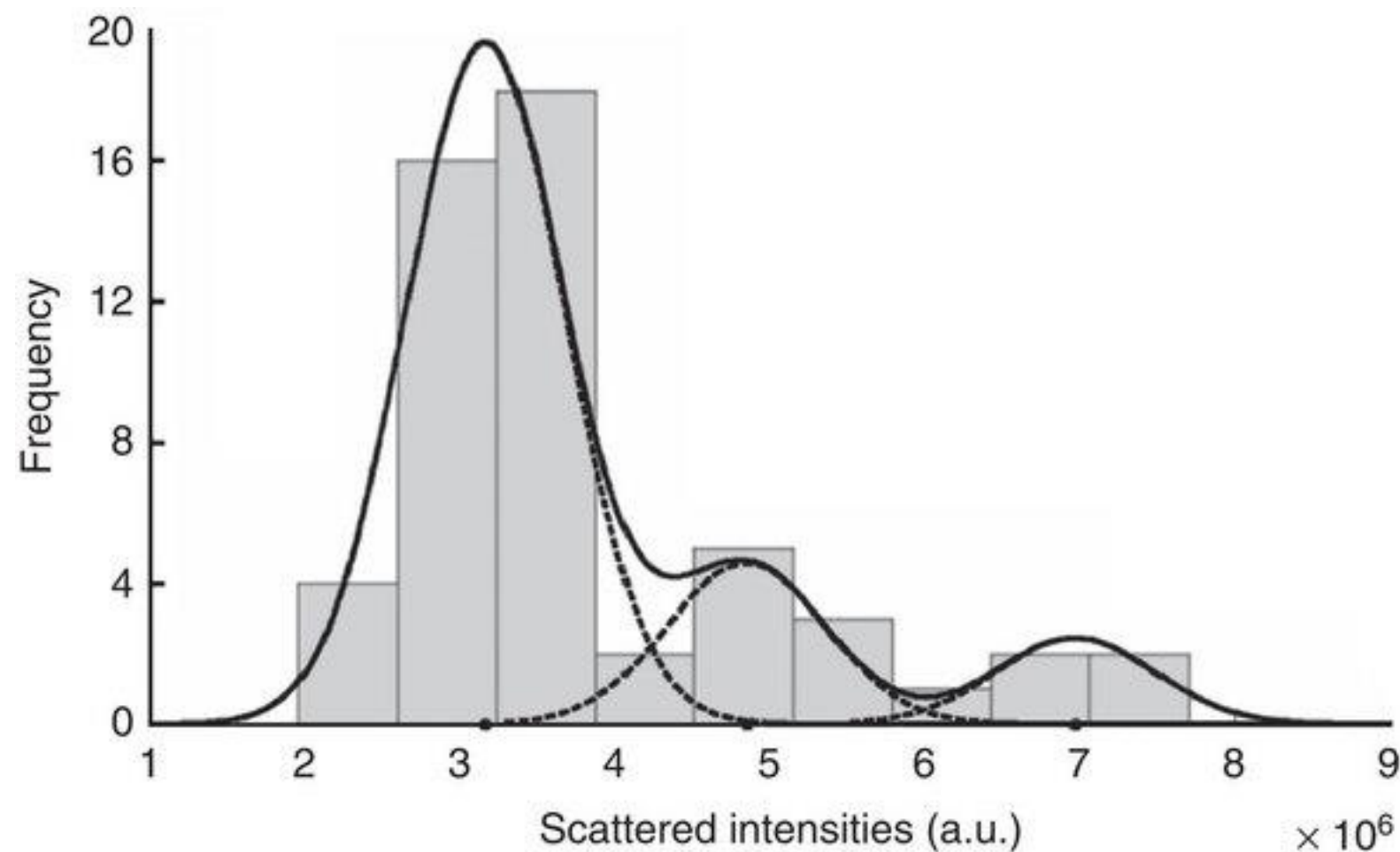
Introducción

Regresión

Estandarización de Datos

Reducción de Dimensiones

Técnicas Representativas



*ejemplo del algoritmo GMM*



(diapositiva en blanco)