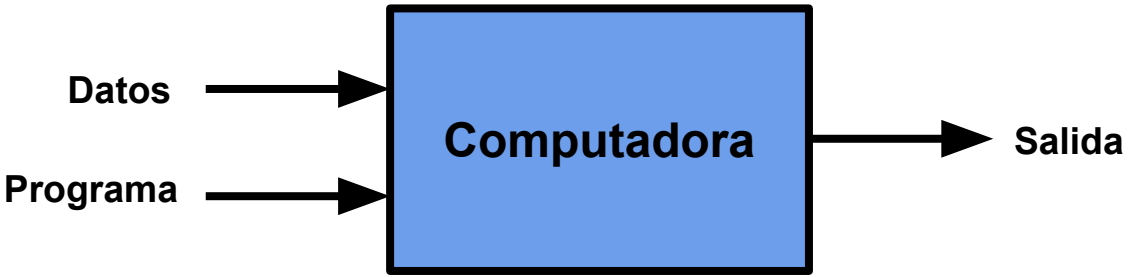
Introducción al aprendizaje automático

•••

#1. Introducción al aprendizaje automático.
Regresión. Clasificación.

Programación tradicional



Aprendizaje automático

Tipos de aprendizaje

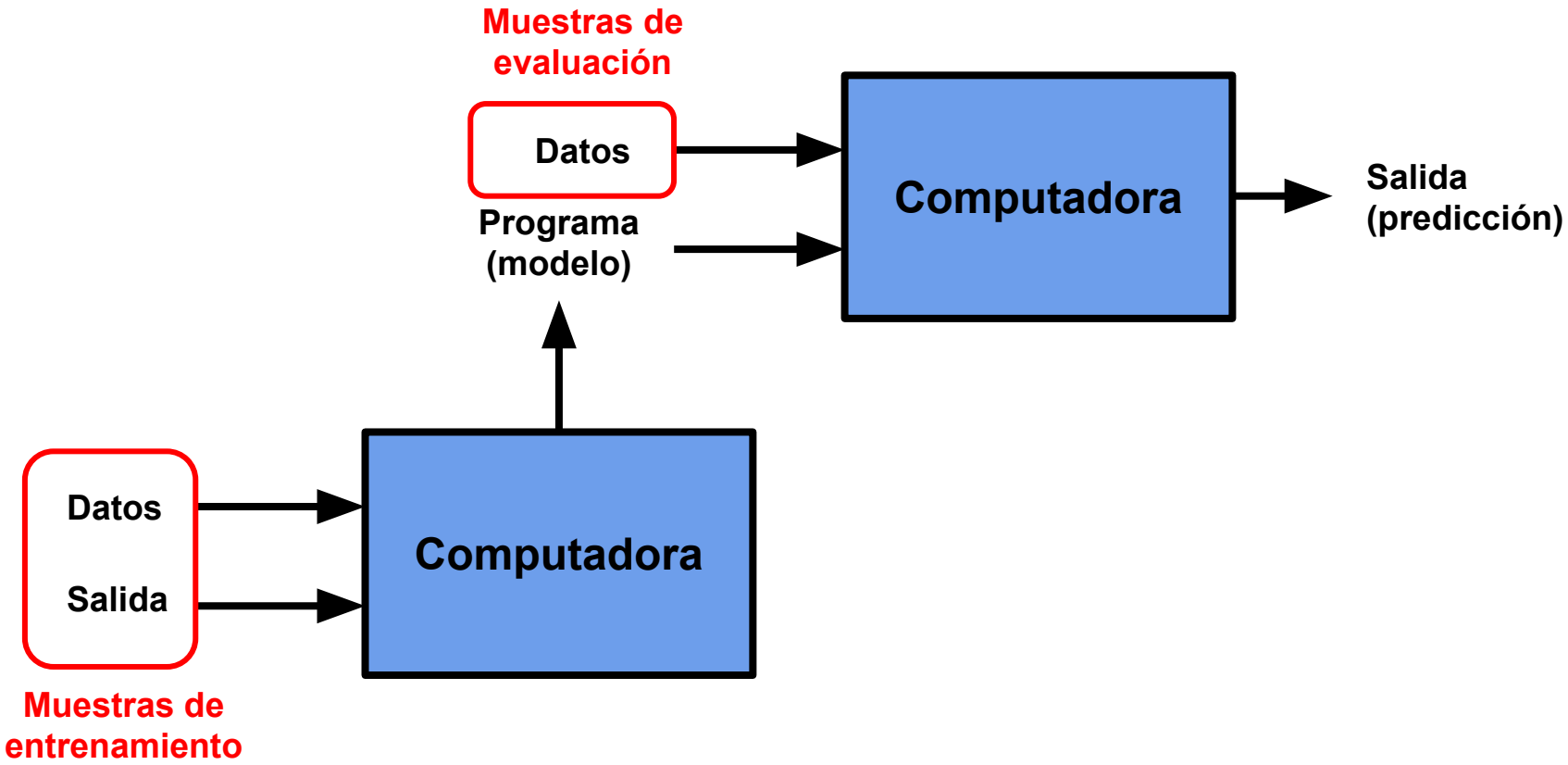
- **Aprendizaje supervisado (inductivo)**
  Datos de entrenamiento + salida esperada

- **Aprendizaje no supervisado**
  Datos de entrenamiento (sin salida esperada)

- **Aprendizaje semi-supervisado**
  Datos de entrenamiento + **pocas** salida esperadas

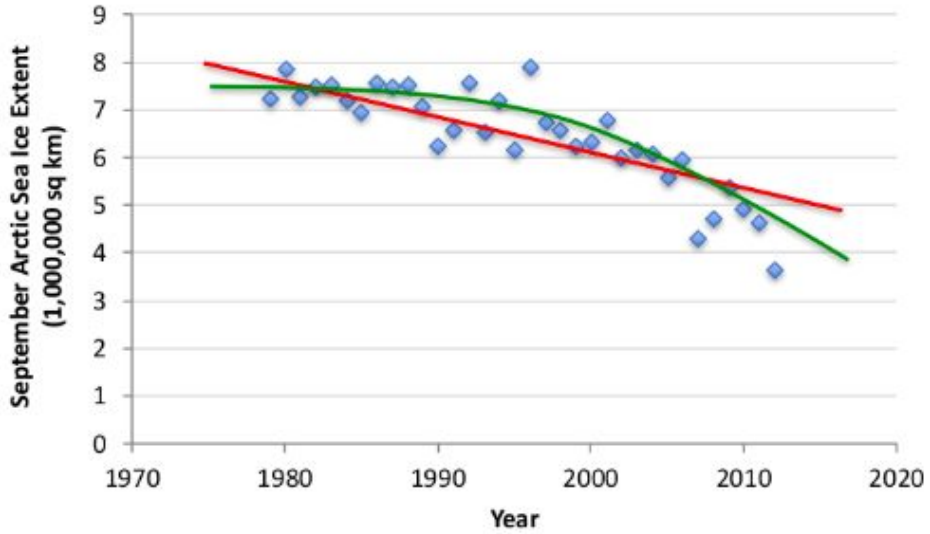
- **Aprendizaje por refuerzo**
  "Recompensas" por secuencias de acciones

Aprendizaje supervisado: entrenamiento vs. evaluación



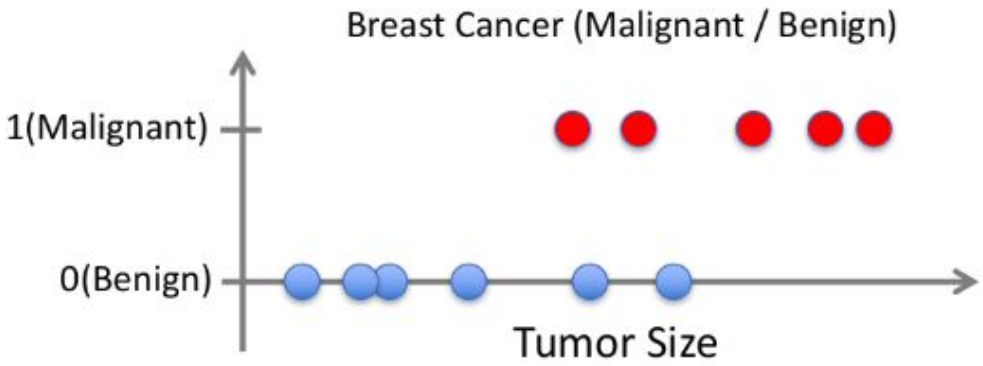
Aprendizaje supervisado: regresión

- Dados $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
- Aprender una $f(x)$ que permita predecir y a partir de x
    - Si y está en $\mathbb{R}^n$ → **regresión**



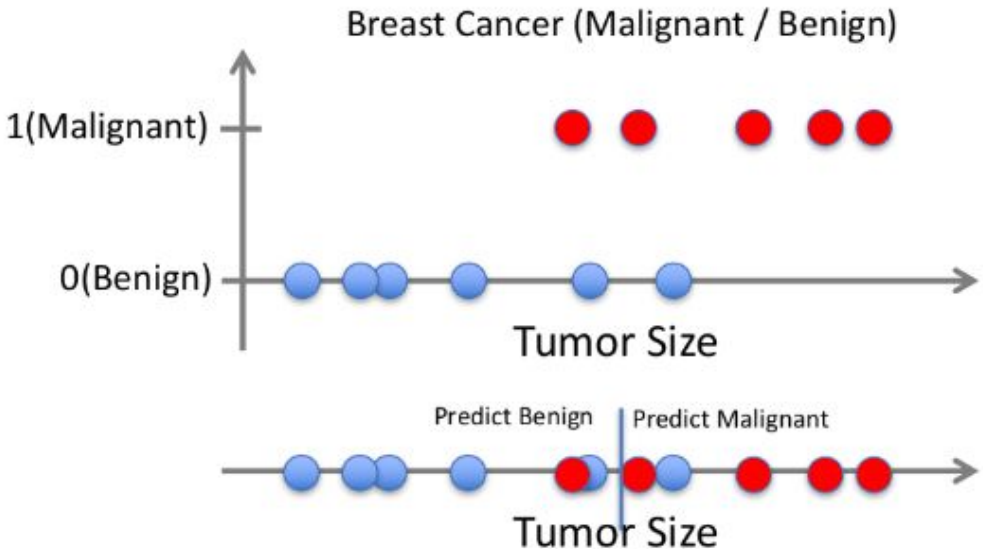
Aprendizaje supervisado: clasificación

- Dados $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
- Aprender una $f(x)$ que permita predecir y a partir de x
    - Si y es categórica → **clasificación**



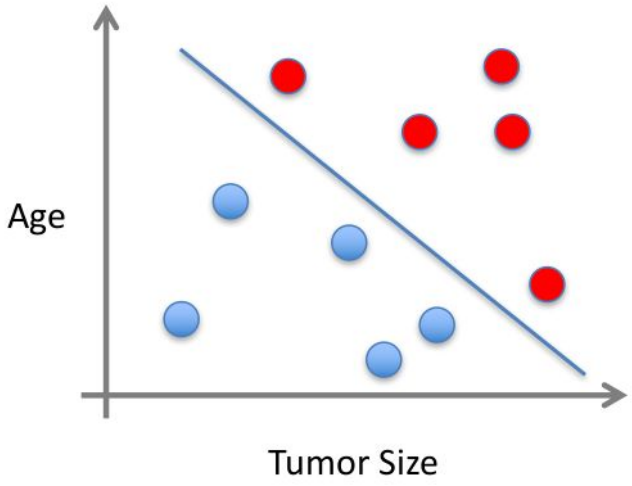
Aprendizaje supervisado: clasificación

- Dados $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$
- Aprender una $f(x)$ que permita predecir y a partir de x
  - Si y es categórica → **clasificación**



Supervised Learning

- x can be multi-dimensional
  - Each dimension corresponds to an attribute



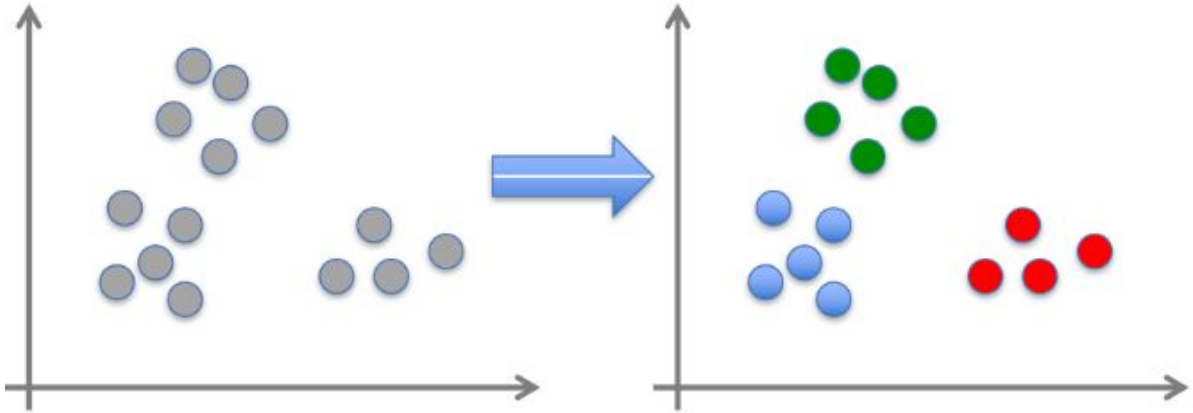
- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape

...

Based on example by Andrew Ng

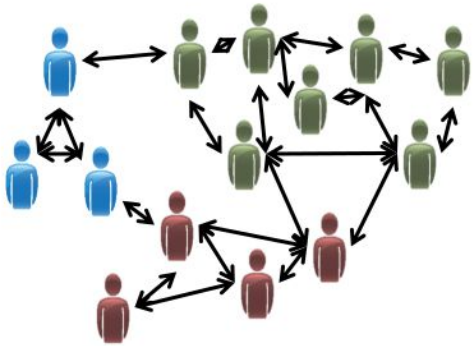
Aprendizaje no supervisado

- Dados $x_1, x_2, ..., x_n$
- Aprender la estructura interna de los datos
    - p.ej. *clustering*



Aprendizaje no supervisado

Organize computing clusters



Social network analysis

Market segmentation

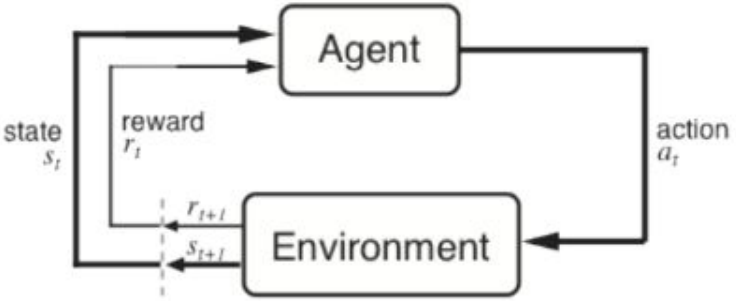
Astronomical data analysis

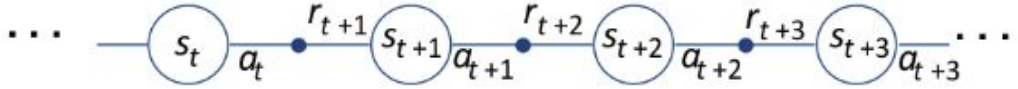
Aprendizaje por refuerzo

- Dada una secuencia de estados y acciones con recompensa (*reward*), generar una política (*policy*)
    - política = mapeo estados → acciones que nos dicen que hacer en un determinado estado
- Ejemplos:
    - Juegos
    - Navegación en robótica
    - Control
    - ...

La interfaz agente-entorno

- El agente y el entorno interactúan a instantes discretos de tiempo
  - $t=0,1,...,K$
  - el agente observa el estado S_t en el paso t
  - produce una acción a_t en el paso t
  - obtiene una recompensa r_{t+1} en el paso $t+1$
  - genera un nuevo estado s_{t+1} en el paso $t+1$

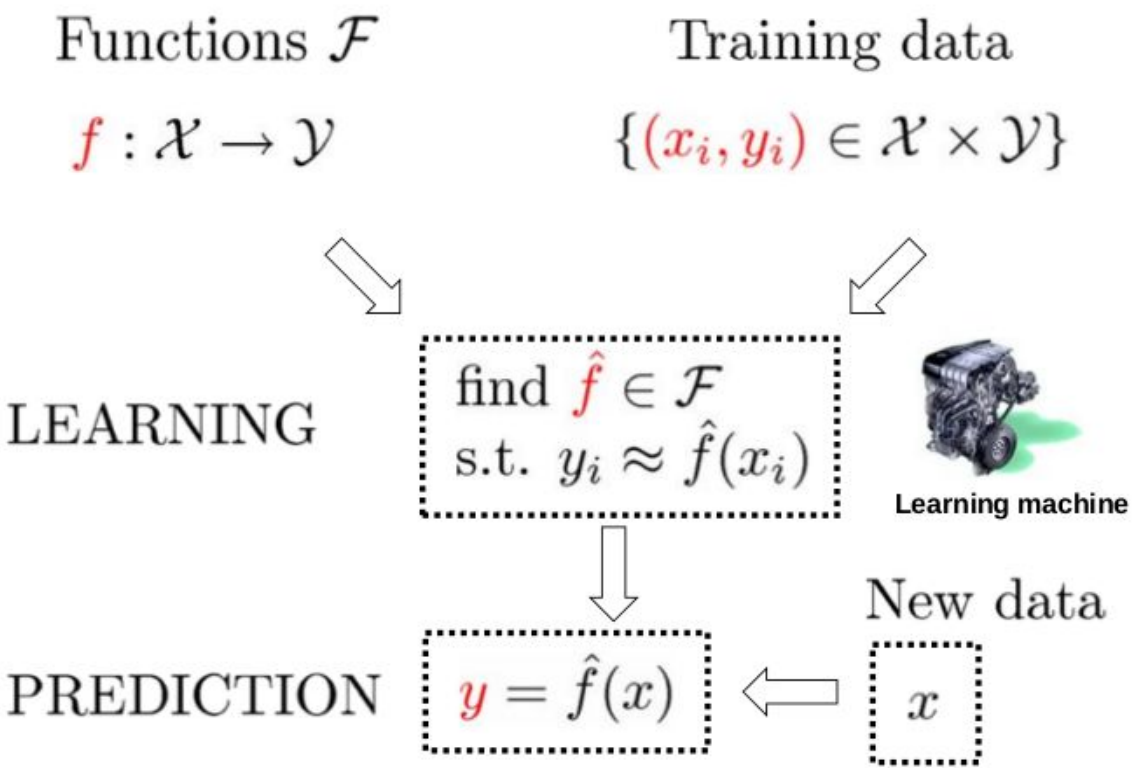




Sobre "aprendizaje"

- Se puede ver como la utilización directa o indirecta de la experiencia para aproximar una determinada función.
- La aproximación de dicha función corresponde a una búsqueda en un espacio de hipótesis (espacio de funciones) por aquella que mejor ajusta el conjunto de datos de entrenamiento.
- Distintos métodos de aprendizaje automático asumen distintos espacios de hipótesis o utilizan distintas estrategias de búsqueda.

Aprendizaje supervisado



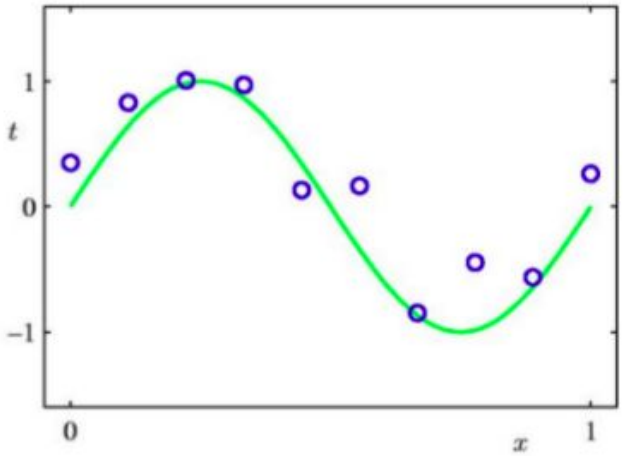
Regresión

Regresión

- Disponemos de N pares de entrenamiento (observaciones)

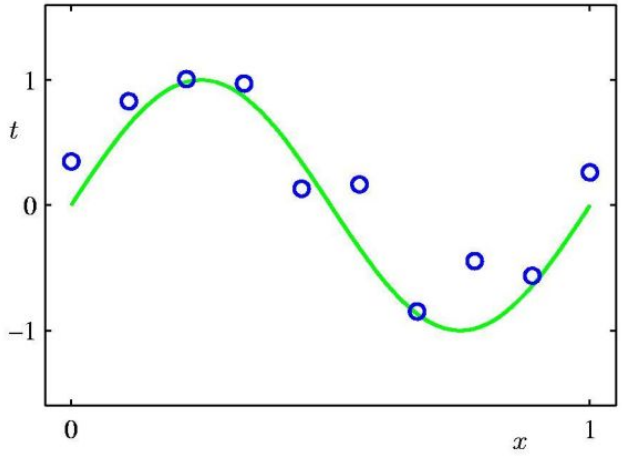
$$\{(x_i, y_i)\}_{i=1}^{N} = \{(x_1, y_1), \cdots, (x_N, y_N)\}$$

- El problema de regresión consiste en estimar *f(x)* a partir de estos datos



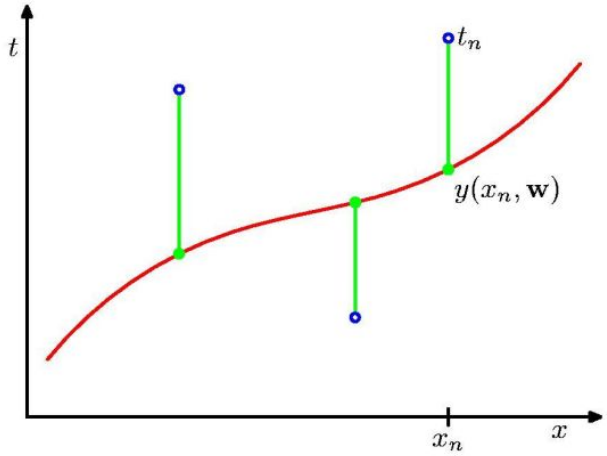
Regresión polinomial

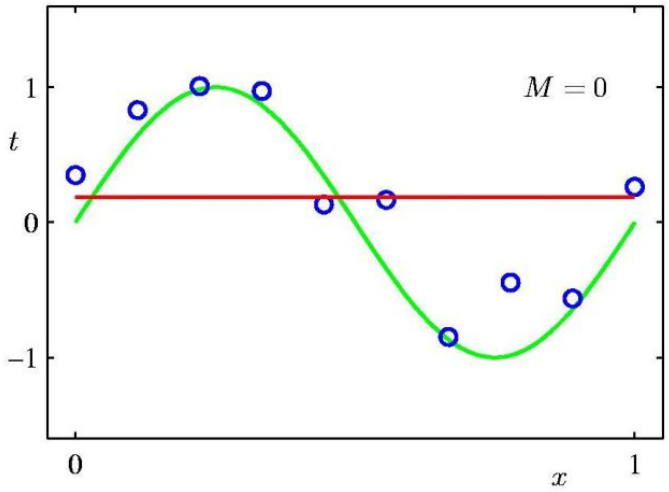
- En verde se ilustra la función "verdadera" (inaccesible)
- Las muestras son uniformes en x y poseen ruido en y
- Utilizaremos una **<u>función de costo</u>** (error cuadrático) que mida el error en la predicción de y mediante $f(x)$

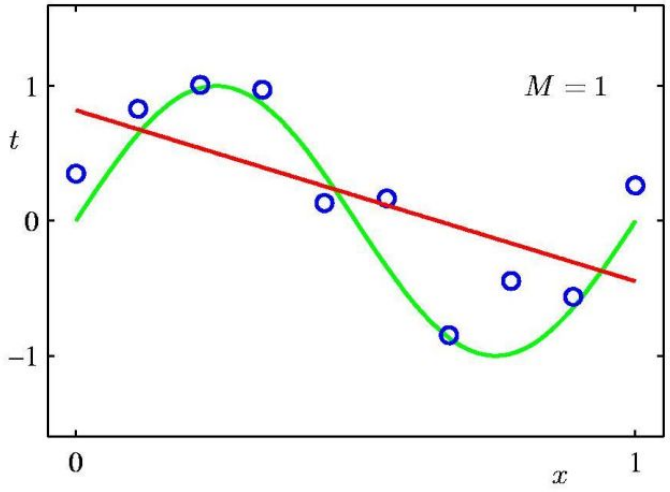


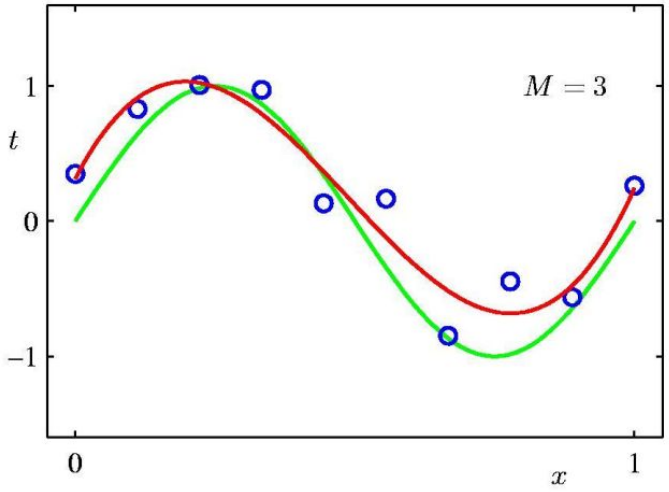
$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2$$

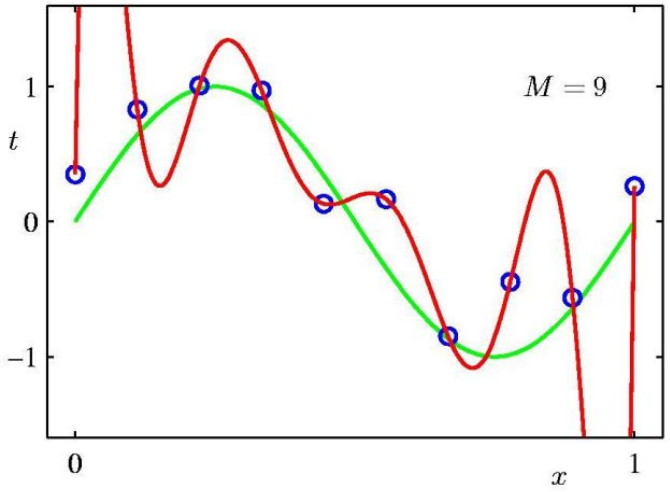
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$




M = 0
t
x
1
0
−1
0
1

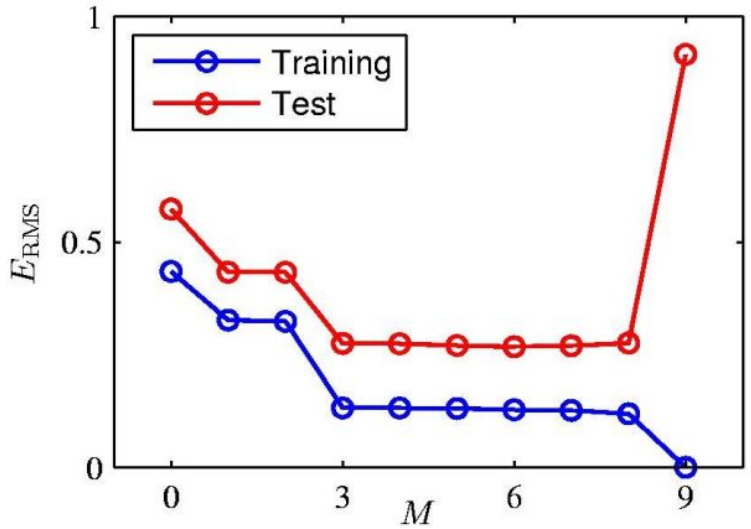

M = 1
t
x
1
0
−1
0
1


M = 3
t
x
1
0
−1
0
1


M = 9
t
x
1
0
−1
0
1

Sobreajuste (*overfitting*)

- Datos de test: otra muestra de los misma función subyacente
- El error de entrenamiento se hace cero, pero el de test crece con M



Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$

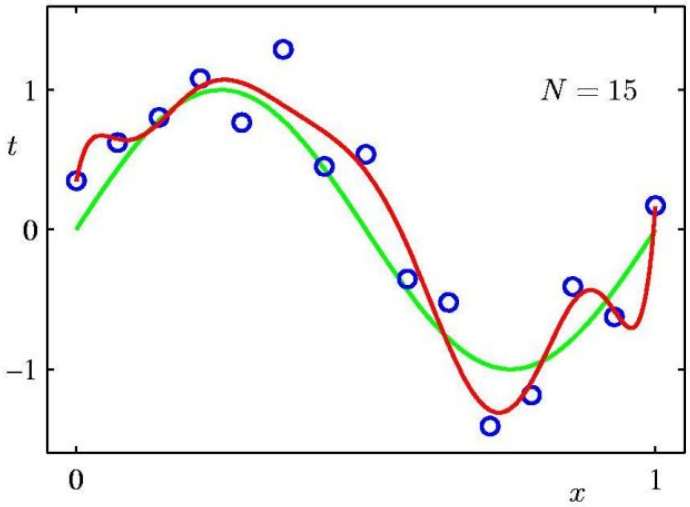
Bondad de ajuste vs. complejidad de modelo

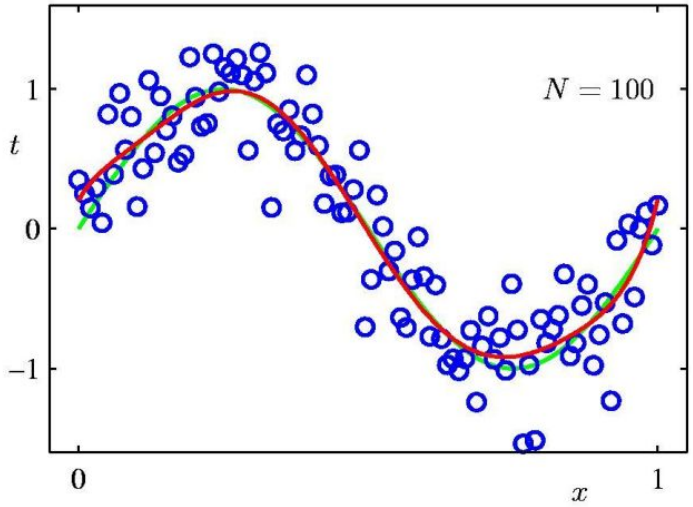
- Si el modelo tiene tantos grados de libertad como los presentes en los datos de entrenamiento, puede ajustarlos perfectamente
- El objetivo en aprendizaje automático no es el ajuste perfecto, sino la generalización a conjuntos no vistos
- Podemos decir que un modelo generaliza, si puede explicar los datos empleando una complejidad acotada

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Prevenir el sobreajuste (I)

- Agregar más datos (más que la "complejidad" del modelo)

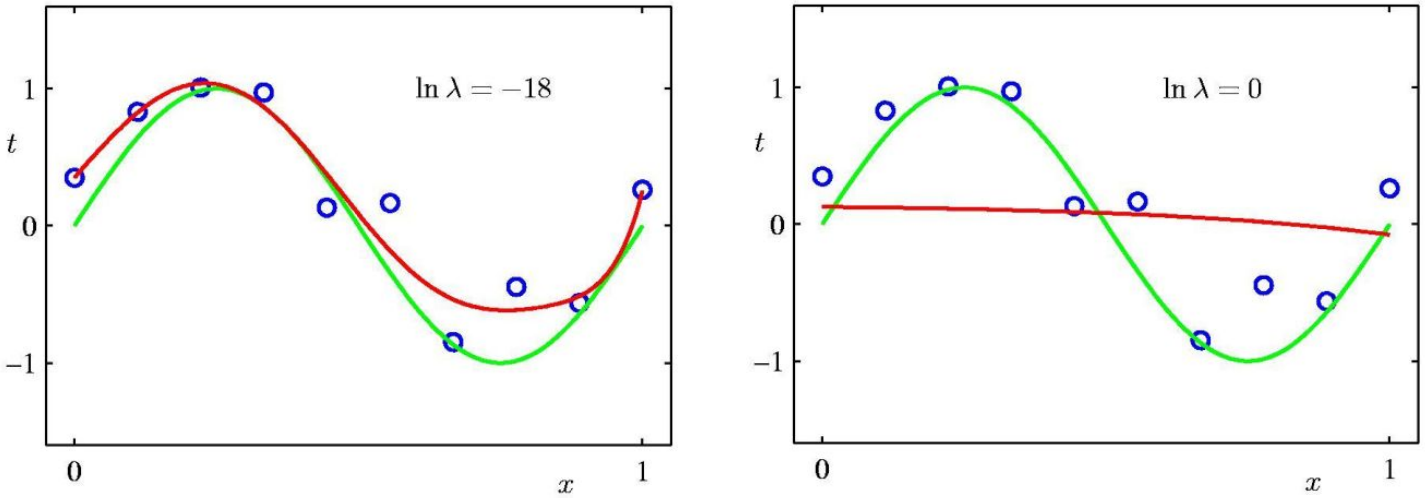




Prevenir el sobreajuste (II)

- Regularización: penalizar valores grandes de los coeficientes

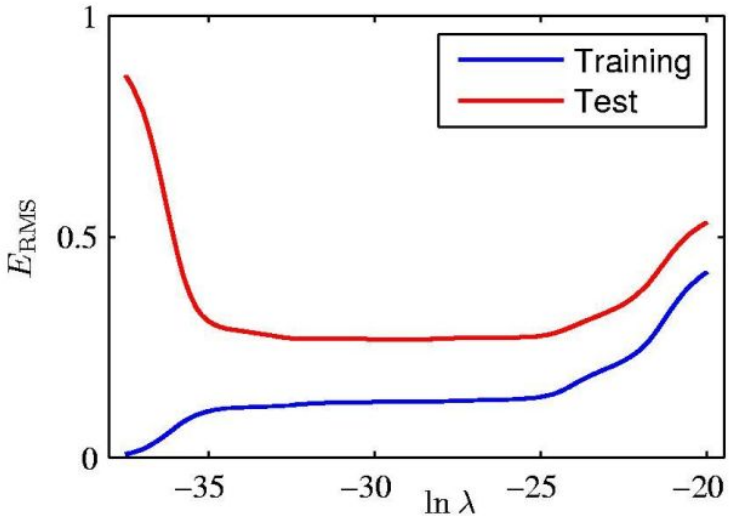
$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



Prevenir el sobreajuste (II)

- Regularización: penalizar valores grandes de los coeficientes

$$\widetilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$



Término de regularización (*ridge*)

λ = hiperparámetro

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Regresión lineal: solución de cuadrados mínimos

- Función de predicción lineal $y = f_w(x) =< x, w >= \sum_{k=1}^{K} x_k w_k$
- Función de costo: $L(w) = \sum_{i=1}^{N} (y^i - < x^i, w >)^2$
- Ecuaciones normales

$$\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1^1 & \dots & x_k^1 & \dots & x_K^1 \\ & & \vdots & & \\ x_1^N & \dots & x_k^N & \dots & x_K^N \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_k \\ \vdots \\ w_K \end{bmatrix}$$

$$\mathbf{e} = \mathbf{y} - \mathbf{X}\mathbf{w}$$

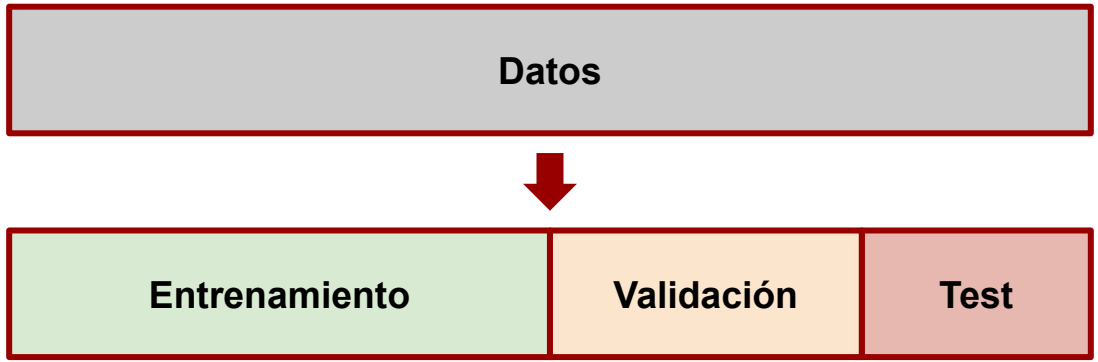
$$L(\mathbf{w}) = \mathbf{e}^T\mathbf{e} \rightarrow \mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

$$L(\mathbf{w}) = \mathbf{e}^T\mathbf{e} + \lambda\mathbf{w}^T\mathbf{w} \rightarrow \mathbf{w}^* = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

Elección de *hiperparámetros*

Dividir el conjunto total de ejemplos en tres subconjuntos

- **Entrenamiento**: aprendizaje de variables del modelo
- **Validación**: ajuste/elección de hiperparámetros
- **Test**: estimación final de la performance del modelo entrenado (y con hiperparámetros elegidos adecuadamente



Clasificación

Clasificación binaria

- Disponemos de N pares de entrenamiento (observaciones)

$$\{(x_i, y_i)\}_{i=1}^{N} = \{(x_1, y_1), \cdots, (x_N, y_N)\}$$

con $x_i \epsilon \mathbb{R}^n$, $y_i \epsilon \{-1, +1\}$.

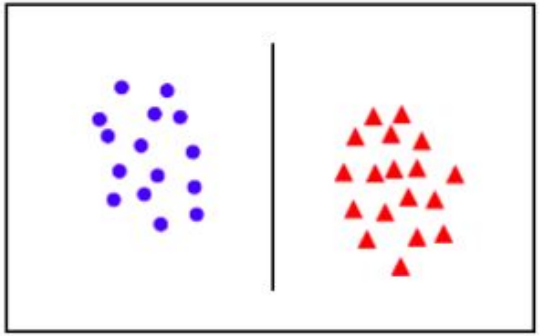
- Aprender una $f(x)$ tal que

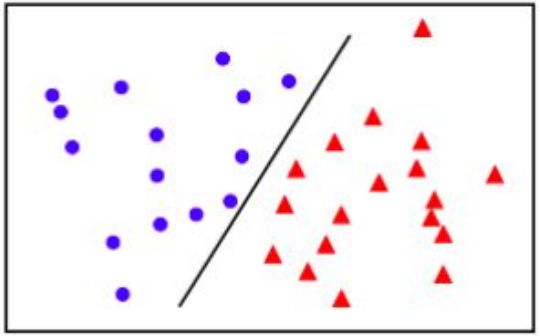
$$f(\mathbf{x}_i) \begin{cases} \geq 0 & y_i = +1 \\ < 0 & y_i = -1 \end{cases}$$

es decir: $y_i f(x_i) > 0$ para una clasificación correcta.

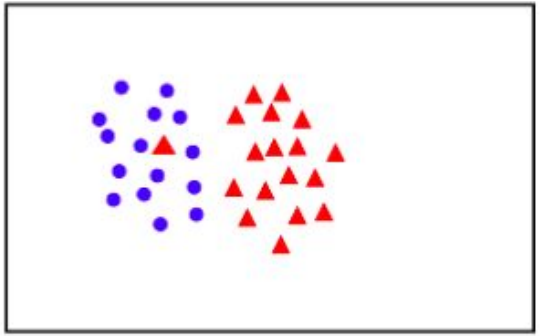
Separabilidad lineal

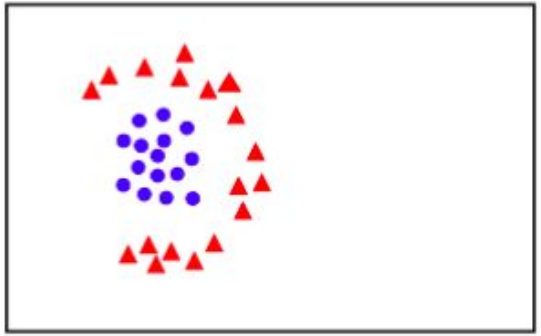
linealmente separable





no linealmente separable





Clasificadores lineales

- La entrada es un vector x_i de dimensionalidad n
- La salida es una etiqueta $y_i \epsilon \{-1, +1\}$
- Clasificador = función de predicción + función de decisión

$$g(f(x)) \rightarrow \{-1, +1\}$$

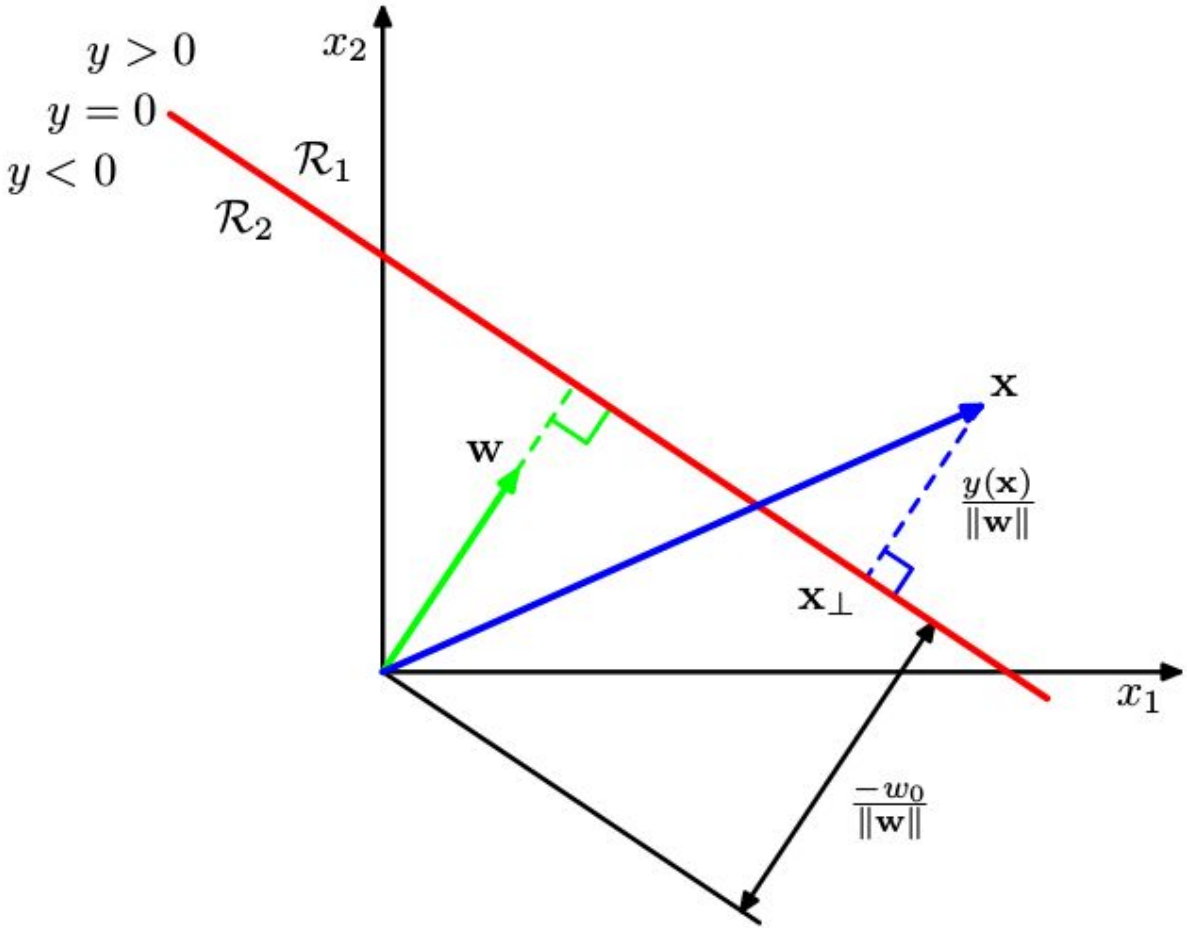
- Función de predicción **lineal**

$$f(x) = w^{\mathrm{T}}x + w_0$$

- Función de decisión

$$g(z) = sign(z)$$

$$g(f(x)) = sign(w^{\mathrm{T}}x + w_0)$$


$y > 0$
$y = 0$
$y < 0$
x_2
$\mathcal{R}_1$
$\mathcal{R}_2$
$\mathbf{x}$
$\mathbf{w}$
$\frac{y(\mathbf{x})}{\|\mathbf{w}\|}$
$\mathbf{x}_\perp$
x_1
$\frac{-w_0}{\|\mathbf{w}\|}$

El algoritmo del "perceptrón"

- Propuesto por Rosemblatt en 1958

- El objetivo es encontrar un hiperplano de separación
  - Si los datos son linealmente separables, lo encuentra

- Es un algoritmo *online* (procesa un ejemplo a la vez)

- Muchas variantes ...

El algoritmo del "perceptrón"

Entrada:

- una secuencia de pares de entrenamiento (x_1,y_1), (x_2, y_2) ...
- Una tasa de aprendizaje r (número pequeño y menor a 1)

Algoritmo:

- Inicializar $w^{(0)} \epsilon \mathbb{R}^n$
- Para cada ejemplo (x_i,y_i)
  - Predecir $y_i' = sign(w^T x_i + w_0)$
  - Si $y_i' \neq y_i$:

    $w^{(t+1)} \leftarrow w^{(t)} + r\,(y_i x_i)$

El algoritmo del "perceptrón"

Entrada:

- una secuencia de pares de entrenamiento (x_1,y_1), (x_2, y_2) ...
- Una tasa de aprendizaje r (número pequeño y menor a 1)

Algoritmo:

- Inicializar $w^{(0)} \epsilon \mathbb{R}^n$
- Para cada ejemplo (x_i,y_i)
  - Predecir $y_i' = sign(w^T x_i + w_0)$
  - Si $y_i' \neq y_i$:

    $w^{(t+1)} \leftarrow w^{(t)} + r\,(y_i x_i)$

Nota: el término de bias se puede contemplar definiendo las entrada como $(x_i^T\ 1)^T \epsilon \mathbb{R}^{n+1}$.

Pregunta: ¿qué implica que $w_0=0$?

El algoritmo del "perceptrón"

Entrada:

- una secuencia de pares de entrenamiento (x_1,y_1), (x_2, y_2) ...
- Una tasa de aprendizaje r (número pequeño y menor a 1)

Algoritmo:

- Inicializar $w^{(0)} \epsilon \mathbb{R}^n$
- Para cada ejemplo (x_i,y_i)
  - Predecir $y_i' = sign(w^T x_i)$
  - Si $y_i' \neq y_i$:

    $$w^{(t+1)} \leftarrow w^{(t)} + r\,(y_i x_i)$$

Actualiza solo cuando comete un error

Error en positivos:

$$w^{(t+1)} \leftarrow w^{(t)} + r\,x_i$$

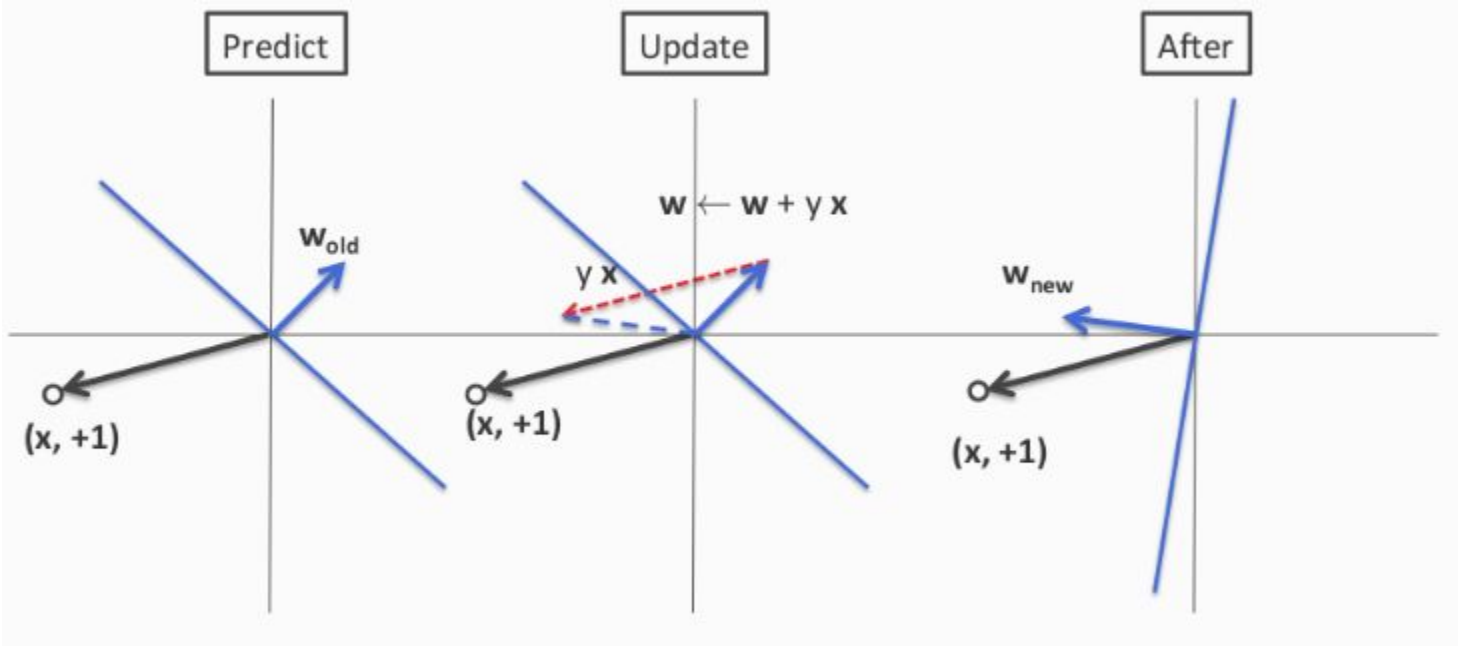
Error en negativos:

$$w^{(t+1)} \leftarrow w^{(t)} - r\,x_i$$

Si $y_i w^T x_i \leq 0 \rightarrow$ error

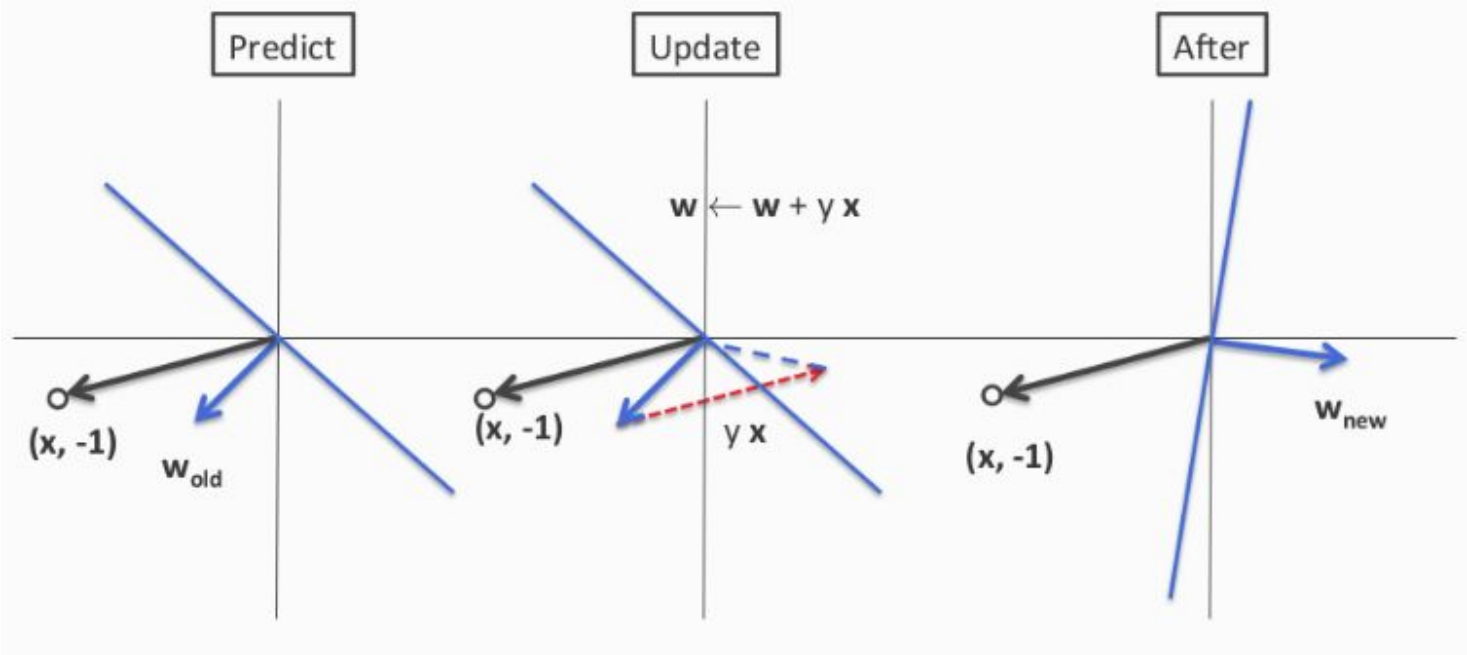
Dinámica de actualización

Error en ejemplo **positivo**:



Dinámica de actualización

Error en ejemplo **negativo**:



El algoritmo "estándar"

Given a training set D = $\{(\mathbf{x}_i, y_i)\}$, $\mathbf{x}_i \in \Re^n$, $y_i \in \{-1,1\}$

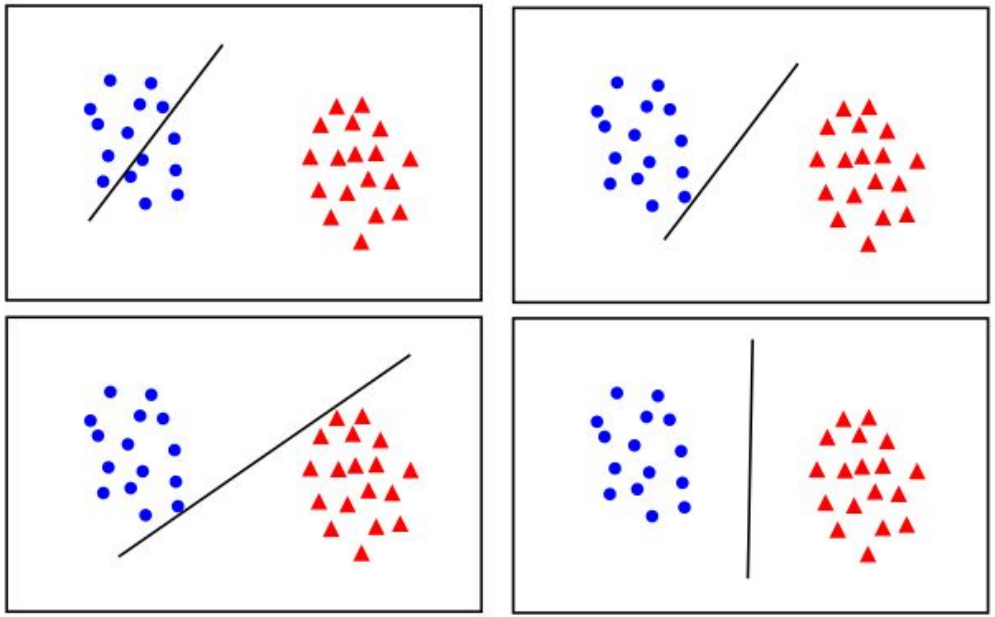
1. Initialize $\mathbf{w} = 0 \in \Re^n$
2. For epoch = 1 … T:
   1. Shuffle the data
   2. For each training example $(\mathbf{x}_i, y_i) \in D$:
      - If $y_i \mathbf{w}^T \mathbf{x}_i \leq 0$, update $\mathbf{w} \leftarrow \mathbf{w} + r\, y_i\, \mathbf{x}_i$
3. Return $\mathbf{w}$

T is a hyper-parameter to the algorithm

Another way of writing that there is an error

Prediction: sgn($\mathbf{w}^T\mathbf{x}$)

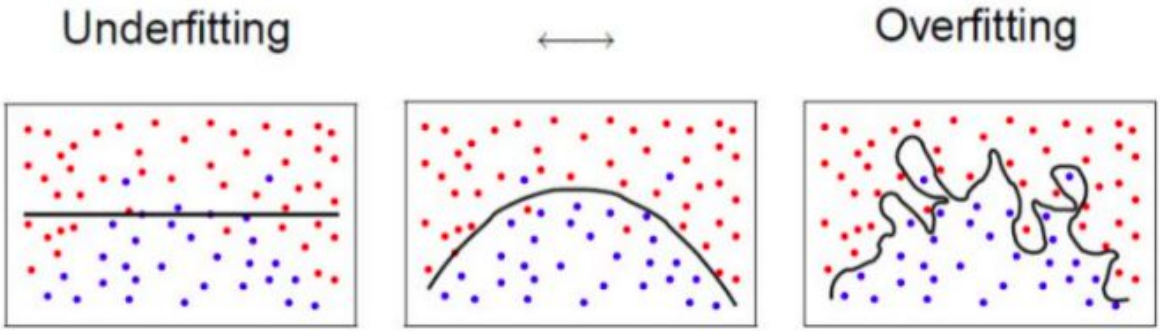
¿Cuál es el mejor w?



Solución de **margen máximo**: el hiperplano más estable ante perturbaciones de la entrada

Generalización en clasificación

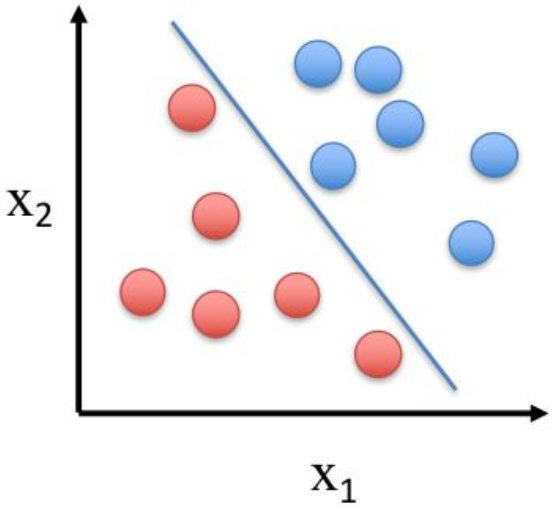
- Complejidad del modelo ⇔ complejidad de la frontera de decisión



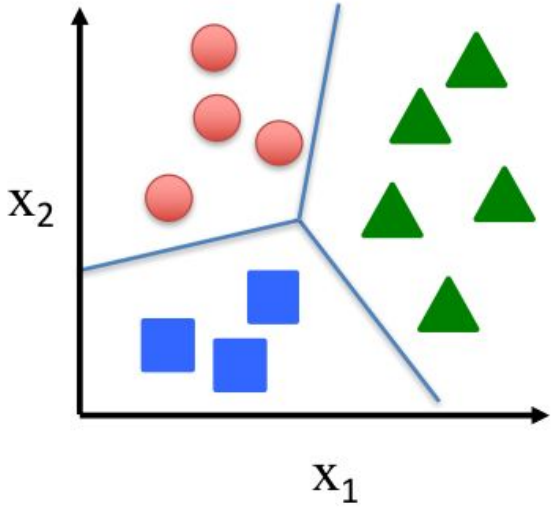
Problemas multiclase

Multi-Class Classification

Binary classification:



Multi-class classification:



Disease diagnosis: healthy / cold / flu / pneumonia

Object classification: desk / chair / monitor / bookcase

What is multiclass classification?

- An input can belong to one of K classes

- Training data: Input associated with class label (a number from 1 to K)
- Prediction: Given a new input, predict the class label

Each input belongs to exactly one class. Not more, not less.

- Otherwise, the problem is not multiclass classification
- If an input can be assigned multiple labels (think tags for emails rather than folders), it is called *multi-label classification*

Binary to multiclass

- Can we use a binary classifier to construct a multiclass classifier?
  - Decompose the prediction into multiple binary decisions
- How to decompose?
  - One-vs-all
  - All-vs-all
  - Error correcting codes

1. One-vs-all classification

- Assumption: Each class individually separable from ***all*** the others

- Learning: Given a dataset $D = \{(\boldsymbol{x_i}, \boldsymbol{y_i})\}$ $\quad \boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Decompose into K binary classification tasks
  - For class k, construct a binary classification task as:
    - **Positive examples**: Elements of D with label k
    - **Negative examples**: All other elements of D
  - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \cdots \mathbf{w}_K$ using any learning algorithm we have seen

1. One-vs-all classification

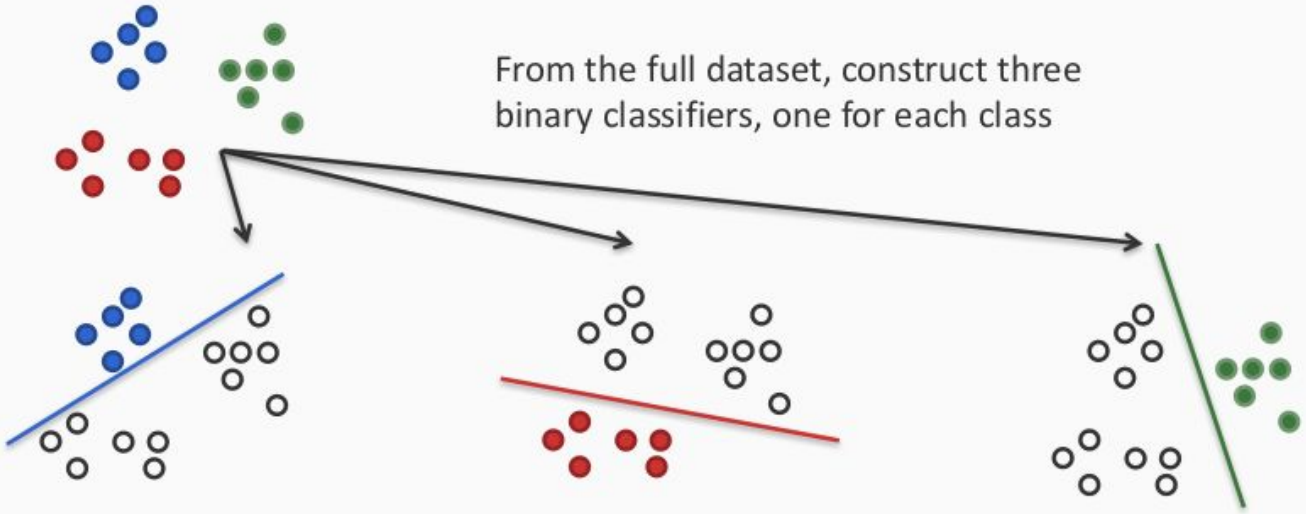
- Assumption: Each class individually separable from ***all*** the others

- Learning: Given a dataset $D = \{(\boldsymbol{x_i}, \boldsymbol{y_i})\}$ $\quad \boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Train K binary classifiers $\mathbf{w}_1, \mathbf{w}_2, \cdots \mathbf{w}_K$ using any learning algorithm we have seen

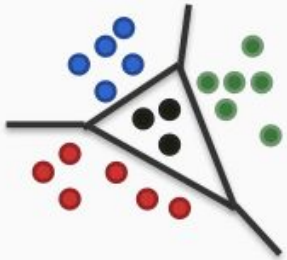
- Prediction: "*Winner Takes All*"

$$\text{argmax}_i \ \mathbf{w}_i^T \mathbf{x}$$

Visualizing One-vs-all

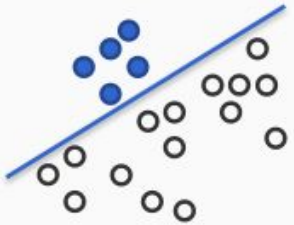


One-vs-all may not always work

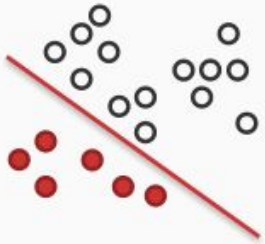


Black points are not separable with a single binary classifier

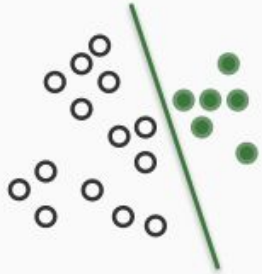
The decomposition will not work for these cases!



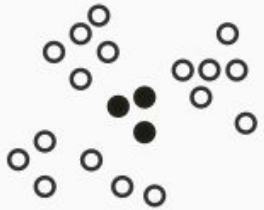
$\mathbf{w}_{blue}^T\mathbf{x} > 0$ for **blue inputs**



$\mathbf{w}_{red}^T\mathbf{x} > 0$ for **red inputs**



$\mathbf{w}_{green}^T\mathbf{x} > 0$ for **green inputs**



???

2. All-vs-all classification

Sometimes called one-vs-one

- Assumption: *Every* pair of classes is separable

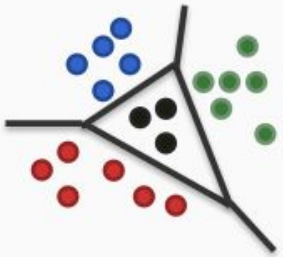
- Learning: Given a dataset $D = \{(\boldsymbol{x_i}, \boldsymbol{y}_i)\}$, $\boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - For every pair of labels (j, k), create a binary classifier with:
    - Positive examples: All examples with label j
    - Negative examples: All examples with label k
  - Train $\binom{K}{2} = \frac{K(K-1)}{2}$ classifiers to separate every pair of labels from each other

2. All-vs-all classification

Sometimes called one-vs-one

- Assumption: *Every* pair of classes is separable
- Learning: Given a dataset $D = \{(\boldsymbol{x_i}, \boldsymbol{y}_i)\}$, $\boldsymbol{x} \in \Re^n$, $\boldsymbol{y} \in \{1,2,\cdots,K\}$
  - Train $\binom{K}{2} = \frac{K(K-1)}{2}$ classifiers to separate every pair of labels from each other
- Prediction: More complex, each label get K-1 votes
  - How to combine the votes? Many methods
    - Majority: Pick the label with maximum votes
    - Organize a tournament between the labels

All-vs-all classification



- Every pair of labels is linearly separable here
  - When a pair of labels is considered, all others are ignored
- Problems
  1. $O(K^2)$ weight vectors to train and store
  2. Size of training set for a pair of labels could be very small, leading to overfitting of the binary classifiers
  3. Prediction is often ad-hoc and might be unstable

     Eg: What if two classes get the same number of votes? For a tournament, what is the sequence in which the labels compete?

3. Error correcting output codes (ECOC)

- Each binary classifier provides one bit of information
- With K labels, we only need $\log_2 K$ bits
  - One-vs-all uses K bits (one per classifier)
  - All-vs-all uses $O(K^2)$ bits

- Can we get by with O(log K) classifiers?
  - Yes! Encode each label as a binary string
  - Or alternatively, if we do train more than O(log K) classifiers, can we use the redundancy to improve classification accuracy?

Using $\log_2 K$ classifiers

label#	Code		
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

8 classes, code-length = 3

- Learning:
  - Represent each label by a bit string
  - Train one binary classifier for each bit
- Prediction:
  - Use the predictions from all the classifiers to create a $\log_2 N$ bit string that uniquely decides the output
- What could go wrong here?
  - Even if one of the classifiers makes a mistake, final prediction is wrong!

Error correcting output coding

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

Answer: Use redundancy

- Assign a binary string with each label
  - Could be random
  - Length of the code word **L** >= $\log_2 K$ is a parameter
- Train one binary classifier for each bit
  - Effectively, split the data into random dichotomies
  - We need only $\log_2 K$ bits
    - Additional bits act as an error correcting code
- One-vs-all is a special case.
  - How?

How to predict?

- Prediction
  - Run all **L** binary classifiers on the example
  - Gives us a predicted bit string of length **L**
  - Output = label whose code word is "closest" to the prediction
  - Closest defined using Hamming distance
    - Longer code length is better, better error-correction

- Example
  - Suppose the binary classifiers here predict 11010
  - The closest label to this is 6, with code word 11000

#	Code				
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	1
3	0	1	1	0	1
4	1	0	0	1	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

8 classes, code-length = 5

Error correcting codes: Discussion

- Assumes that columns are independent
  - Otherwise, ineffective encoding
- Strong theoretical results that depend on code length
  - If minimal Hamming distance between two rows is d, then the prediction can correct up to (d-1)/2 errors in the binary predictions
- Code assignment could be random, or designed for the dataset/task
- One-vs-all and all-vs-all are special cases
  - All-vs-all needs a ternary code (not binary)