UTEC
UNIVERSIDAD DE INGENIERÍA Y TECNOLOGÍA

- Tema 4 -

Introducción a Aprendizaje Supervisado

Prof. Oscar E. Ramos, Ph.D.

Lima, 24 de abril del 2020

Temas

1. **Introducción**
2. Aprendizaje Supervisado
3. Redes Neuronales
4. SVM
5. Otros Métodos
6. Algunas Métricas de Evaluación

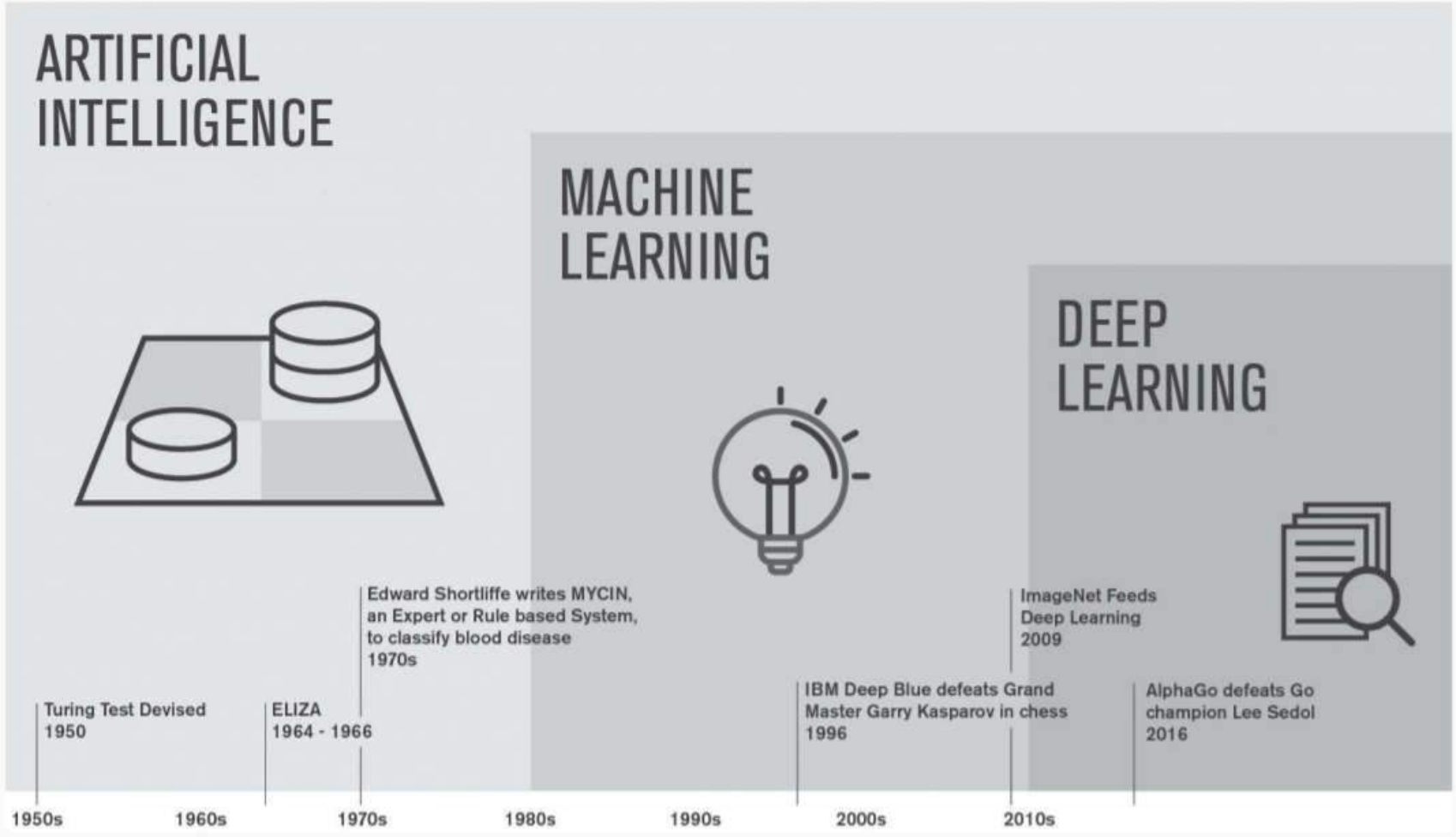
Machine Learning vs Inteligencia Artificial

- Machine learning (ML) es un “área” de inteligencia artificial (IA)
  - El área más popular actualmente

- De modo general:
  - La IA se enfoca en replicar la inteligencia humana (¿razonamiento, lógica?)
  - ML se enfoca principalmente en la computadora

- Áreas (temas) de inteligencia artificial:
  - Planeamiento
  - Sistemas multi-agente y basados en agentes
  - Algoritmos de búsqueda (depth first, breadth first, A*)
  - Representación del conocimiento
  - Razonamiento y lógica
  - Machine Learning
  - Procesamiento de lenguaje natural

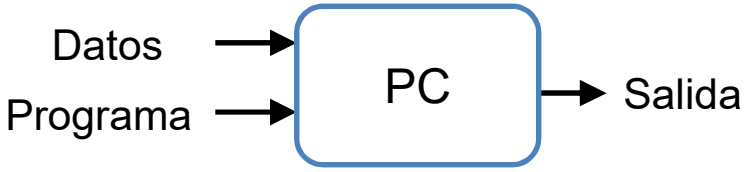
Machine Learning vs Inteligencia Artificial



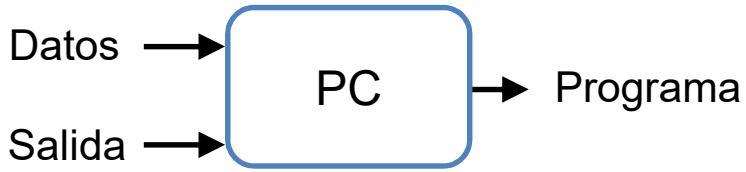
[Stefano Carrazza, CERN]

Machine Learning

- Programación tradicional



- Machine Learning



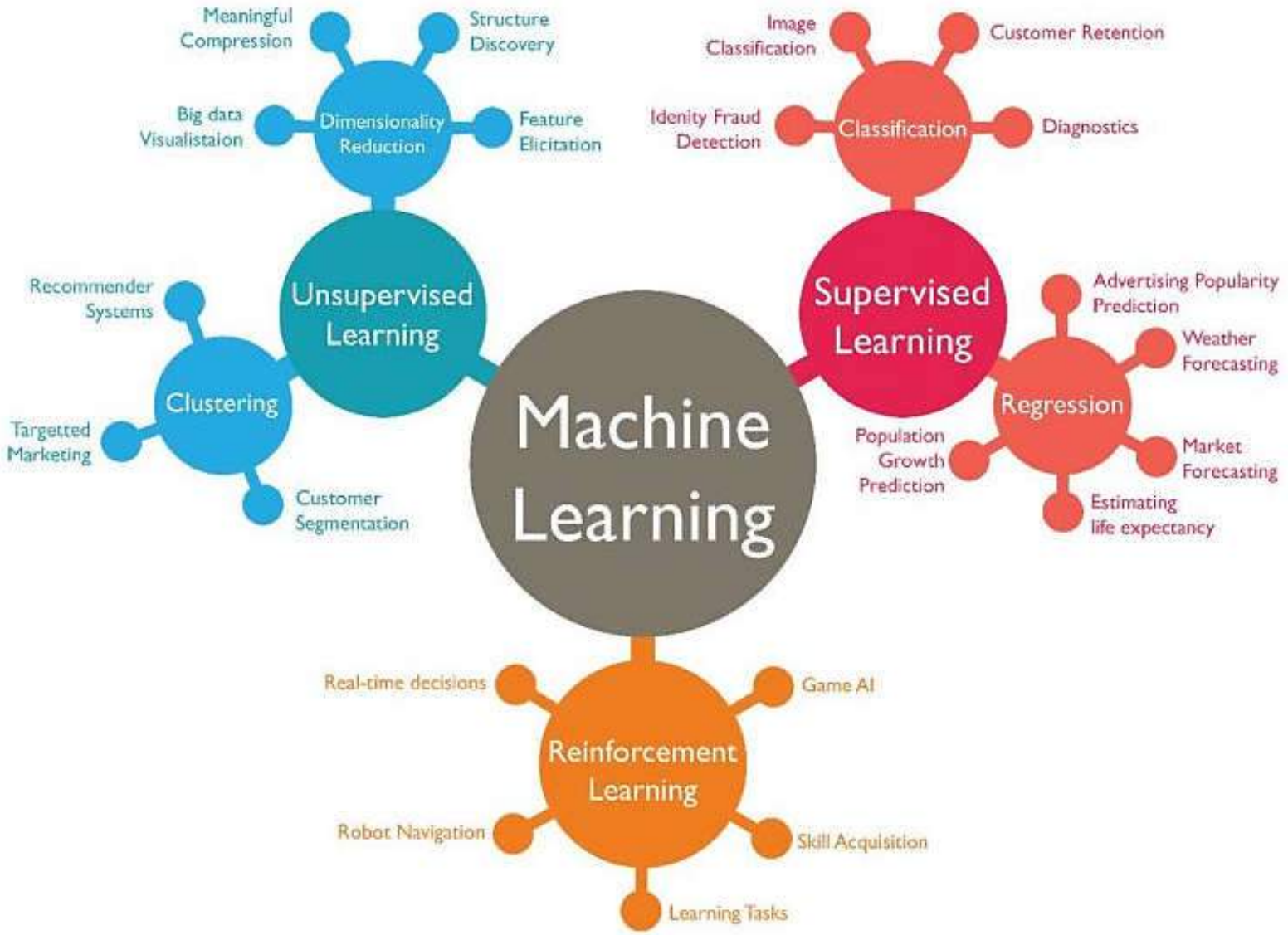
- Analogía de Machine Learning:

  Es como jardinería, donde

  - Semillas = algoritmos
  - Nutrientes = datos
  - Jardinero = humano
  - Plantas = programas

- Importante: capacidad de generalización
  - Realizar una tarea en una situación no encontrada anteriormente

Tipos de Aprendizaje



Tipos de Aprendizaje

- Aprendizaje supervisado (inductivo)
  - Los datos de entrenamiento incluyen las salidas ("respuestas") correctas
  - Objetivo: encontrar la predicción dada la entrada
  - Tipos: regresión, clasificación

- Aprendizaje no supervisado
  - Los datos de entrenamiento no incluyen salidas ("respuestas") correctas
  - Objetivo: descubrir patrones similares, estructuras, sub-espacios
  - Tipos: "clustering", reducción de dimensionalidad

- Reinforcement learning (Aprendizaje por refuerzo)
  - Recompensa dadas una serie de acciones
  - Objetivo: tratar de aprender usando feedback retrasado (recompensa)

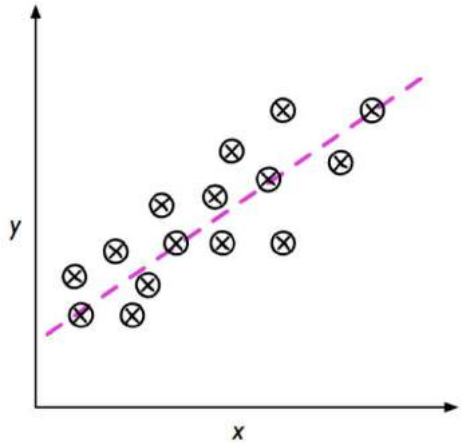
Temas

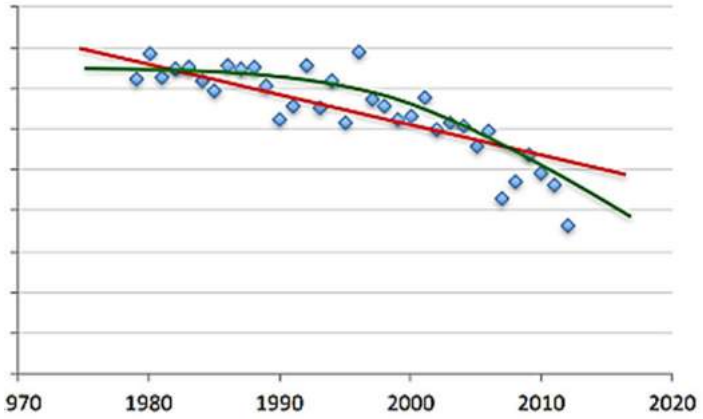
1. Introducción
2. **Aprendizaje Supervisado**
3. Redes Neuronales
4. SVM
5. Otros Métodos
6. Algunas Métricas de Evaluación

Aprendizaje Supervisado

- Regresión (predicción)
  - Dados $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
  - Objetivo: "Aprender" una función $h(x)$ que prediga y dado x
  - Salida: y es un número real

  - Ejemplo:
    - Precio de un carro usado



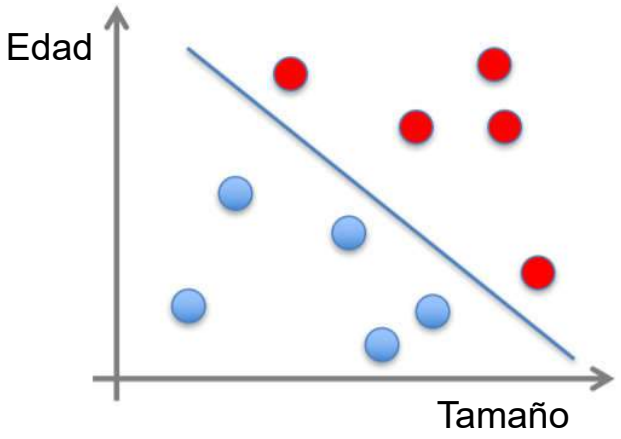


Aprendizaje Supervisado

- Clasificación
  - Dados $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
  - Objetivo: “Aprender” una función $h(x)$ que prediga y dado x
  - Salida: y es categórico (ejemplo: 0 o 1)

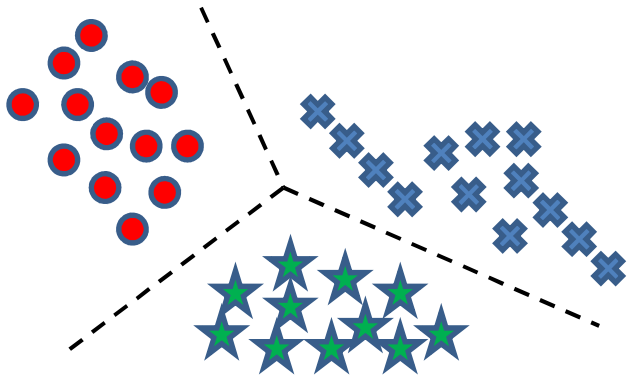
  - Ejemplo:

    Determinar si un tumor es benigno (0) o maligno (1) con base en su tamaño y en la edad del paciente



Aprendizaje Supervisado

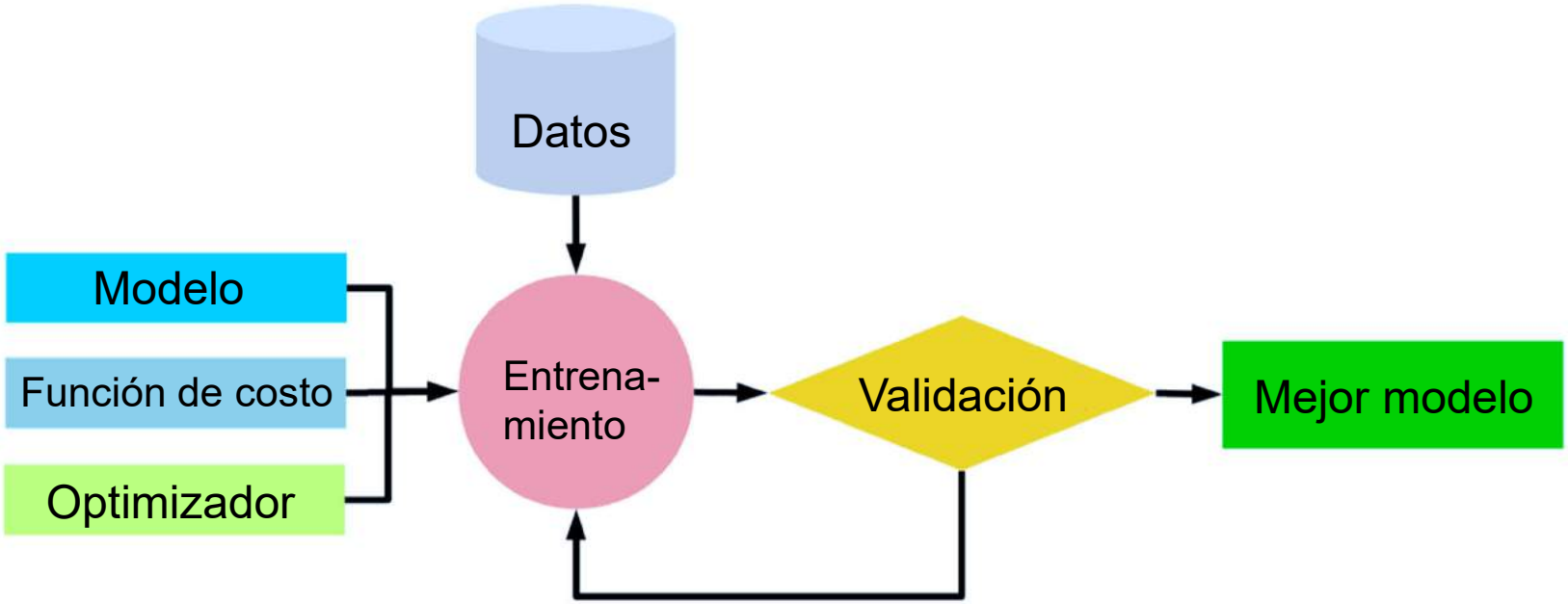
- Clasificación
  - Métodos usados
    - Regresión logística
    - Redes neuronales
    - Support Vector Machines (SVM)
    - Naive Bayes
    - Redes Bayesianas
    - Árboles de decisión
    - K-nearest neighbor
  - Aplicaciones
    - Reconocimiento de rostros
    - Reconocimiento de objetos
    - Reconocimiento de habla
    - Diagnóstico médico
    - Anuncios por internet



Clasificación de tres clases

Esquema General de Aplicación

- Entrenamiento (*training*):

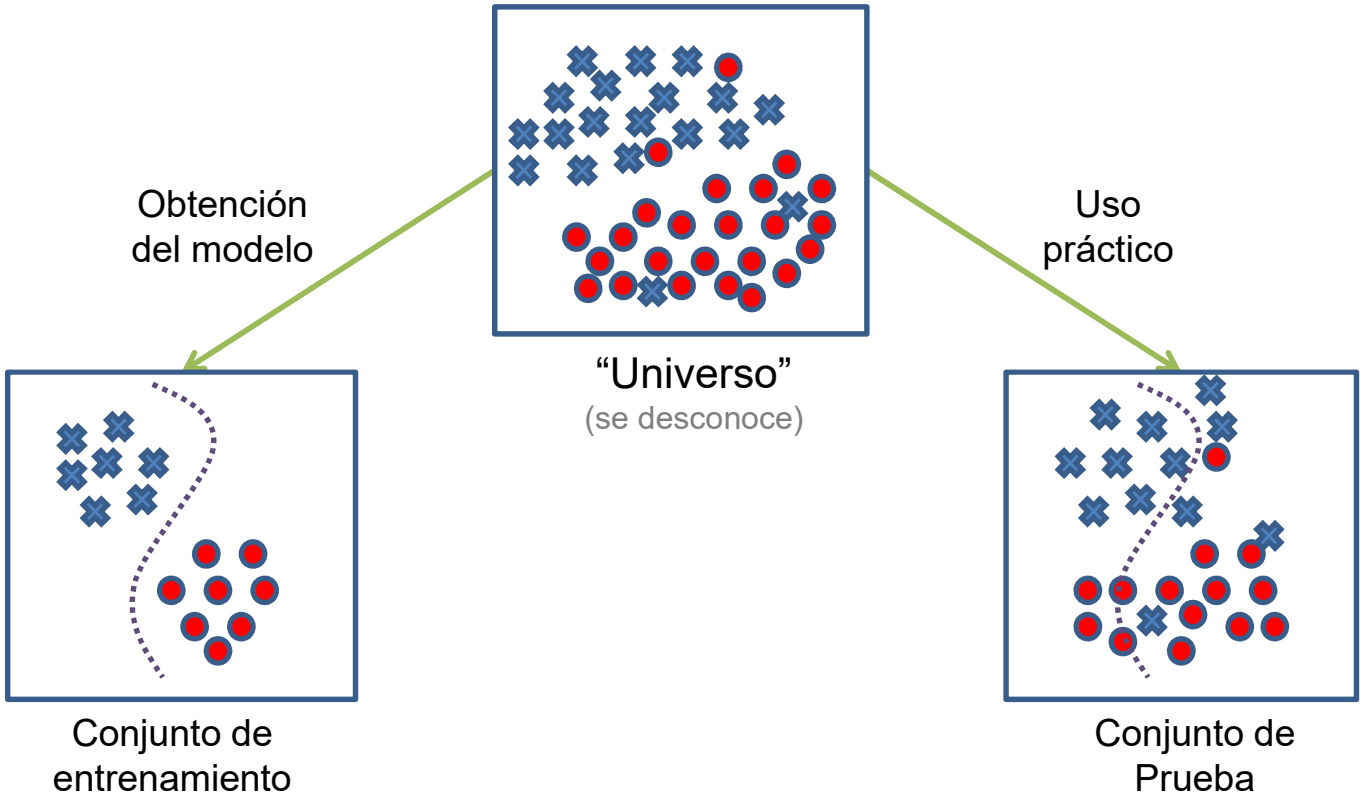


- Prueba (*testing*):
  - El mejor modelo se utiliza para los datos de prueba (testing)

Datos

- Los datos tienen diversos formatos
  - Texto, números, gráficos, tablas, imágenes, videos, etc.

- División de datos:



Datos

- División de los datos
  - Conjunto de entrenamiento (training set)
    - Usado para aprender los parámetros del modelo (y determinar el mejor modelo)
  - Conjunto de prueba (test set)
    - Usado para realizar la prueba de desempeño final
    - NO se debe usar este conjunto para “afinar” el modelo entrenado
    - Se usa solo al final del proceso
  - Conjunto de validación cruzada (cross-validation set)
    - Permite “afinar” el modelo (saber qué funciona mejor)
    - Es como un conjunto de prueba “falso”

- Datos se dividen en:

Modelo

- Un modelo $h(x)$ representa la relación existente entre los datos
  - Trata de predecir la salida y dada una entrada x

- Ejemplo:
  Problema de clasificación: a qué categoría pertenece cada imagen

$h($  $)$ = manzana

$h($  $)$ = tomate

$h($  $)$ = vaca

- Al modelo se le llama "*función de hipótesis*" (h)
  - Toda función de hipótesis (h) tiene parámetros (que se "aprenden")
  - El aprendizaje de los parámetros se realiza solamente con el conjunto de entrenamiento (training set)

Modelo

- Espacio de hipótesis
  - Es un “tipo de modelo”: agrupa a funciones de hipótesis (h) semejantes
  - Cada función de hipótesis (h) pertenece a un espacio de hipótesis
    - Cada h en un espacio de hipótesis tiene parámetros diferentes

- Regla “*No free lunch*” (no hay comida gratis):
  - Al usar una función de hipótesis (un modelo) siempre se hace suposiciones: no hay una función que sea útil para todos los casos

Función de Costo

- Una función de costo (J) mide cuán buena es una función de hipótesis

- Ejemplo:

  Suma de diferencias cuadradas entre lo predicho y lo real

$$J(x) = \sum (h(x) - y)^2$$

- ¿Cuál es la mejor función de hipótesis?
  - Aquella cuyos parámetros minimizan la función de costo
  - Cuanto menor sea J mejor es la función de hipótesis

- La minimización de la función de costo permite obtener la "mejor" función de hipótesis
  - Es más común minimizar usando métodos numéricos (ejemplo: descenso de gradiente)
  - Se busca que J sea continua y convexa (para que tenga un mínimo global)

Generalización

- Idea: qué tan bien se comporta un modelo (h) ante datos de prueba que no ha visto durante el entrenamiento

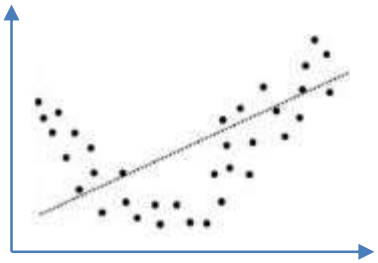
Conjunto de entrenamiento
(se conoce las respuestas)

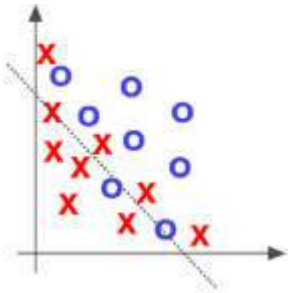
Conjunto de
prueba

[L. Lazebnik]

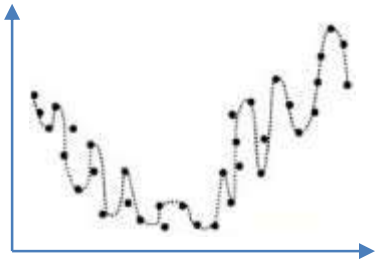
Overfitting vs Underfitting

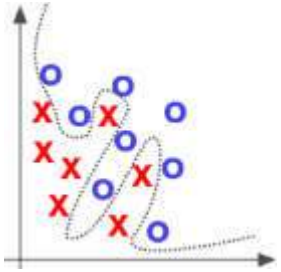
- Underfitting (subajuste)
  - El modelo es muy “simplista”: muy pocos parámetros
  - No generaliza adecuadamente





- Overfitting (sobreajuste)
  - El modelo se adapta “demasiado” a los datos de entrenamiento (y al ruido)
  - Demasiados parámetros
  - No generaliza adecuadamente





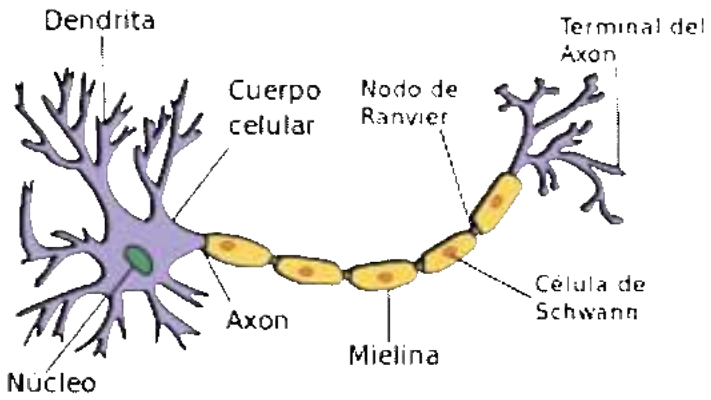
Temas

1. Introducción
2. Aprendizaje Supervisado
3. **Redes Neuronales**
4. SVM
5. Otros Métodos
6. Algunas Métricas de Evaluación

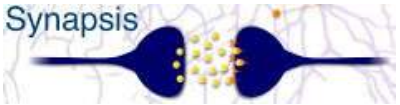
Redes Neuronales

- También llamadas “Redes Neuronales Artificiales” (ANN: *Artificial Neural Networks*)

- Se inspiran en:
  - Neuronas del cerebro humano

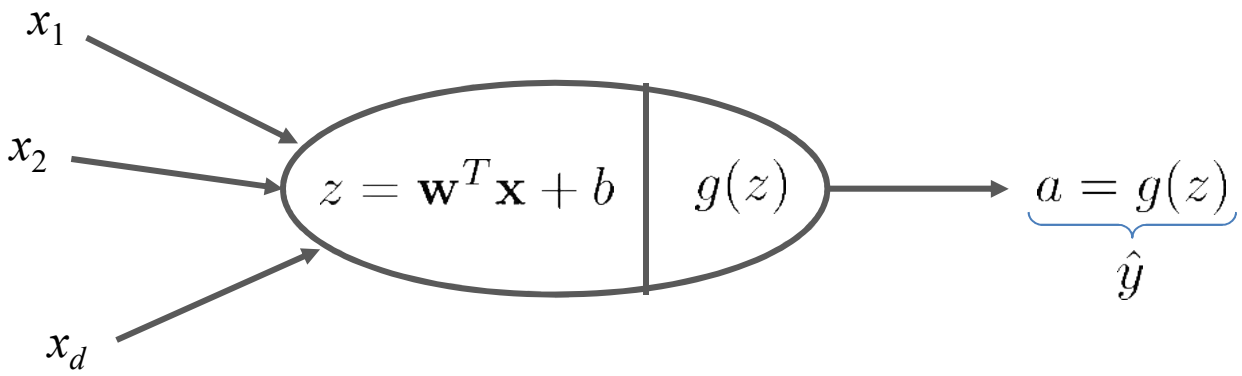


  - Interconexiones de las neuronas



Redes Neuronales

- Unidad neuronal
  - Representa una “neurona” dentro de la red (también: unidad logística)

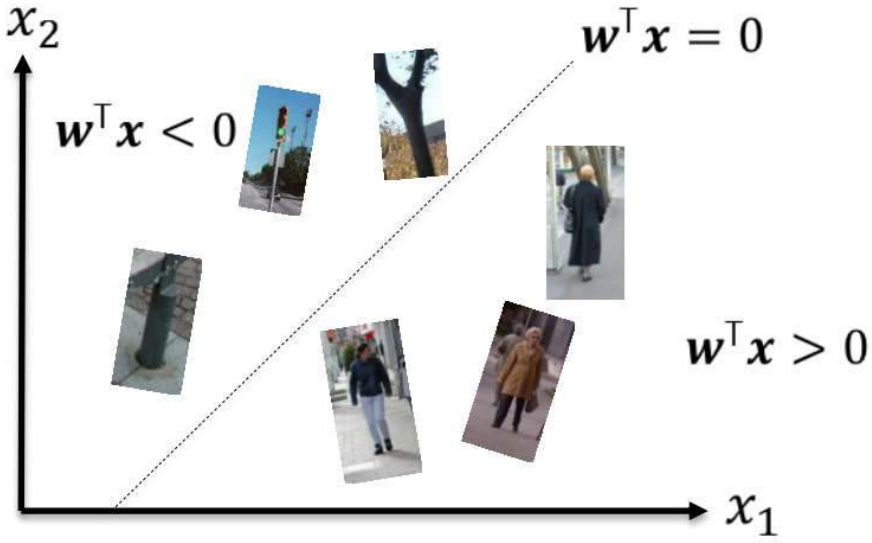


  - Partes importantes:
    - *Función de propagación* (combinación lineal de las entradas): $z \in \mathbb{R}$
      - Contiene pesos ($\boldsymbol{w} \in \mathbb{R}^d$) y bias ($b \in \mathbb{R}$)
    - *Función de activación*: $g \in \mathbb{R}$ y activación $a \in \mathbb{R}$

  - Entrada: $\boldsymbol{x} \in \mathbb{R}^d$ (d atributos)
  - Salida: $\hat{y} \in \mathbb{R}$

Redes Neuronales

- Unidad neuronal
  - La salida es una línea que divide el espacio de las características (según una frontera de decisión)



  - Para un nuevo objeto, se usa directamente los pesos entrenados para determinar su salida
  - Para curvas no lineales se usa varias neuronas (agrupadas en “capas”)

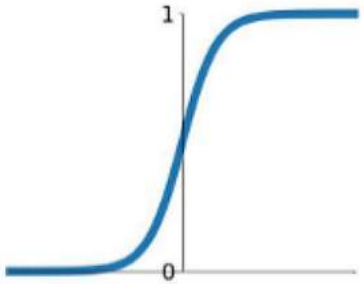
Redes Neuronales

Funciones de Activación

- Tienen como entrada la combinación lineal $z = w^T x + b$

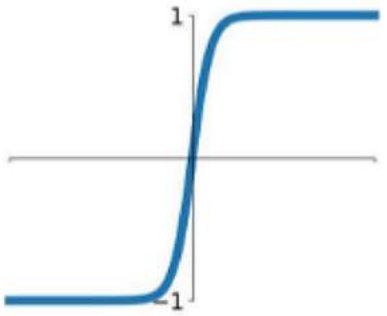
- Función sigmoidea
  - Llamada función logística
  - Mapea a (0,1)
  - Se usa para la salida de toda la red

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



- Función tangente hiperbólica
  - No tiene “bias” (centrado en 0)
  - Mapea a (-1, 1)
  - Se usa para capas ocultas

$$a(z) = \tanh(z)$$

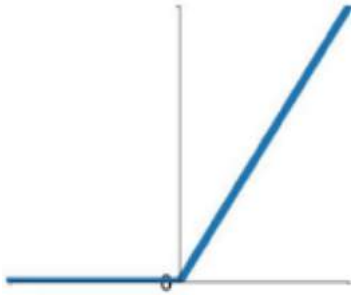


Redes Neuronales

Funciones de Activación

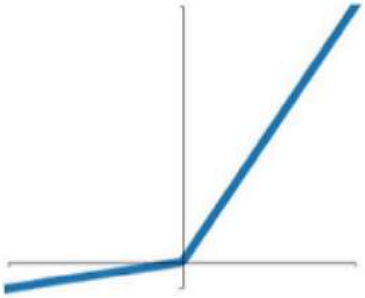
- Función ReLU
  - Significa: “Rectified Linear Unit”
  - Ventaja: facilidad de cálculo
  - Se suele usar en capas ocultas

$$a(z) = \text{máx}(0, z) = \begin{cases} 0, & \text{si } z < 0 \\ z, & \text{si } z \geq 0 \end{cases}$$



- Función Leaky ReLU
  - Variación de ReLU
  - El valor no se fija a cero: pequeña pendiente
  - Puede facilitar el “aprendizaje” (optimización)

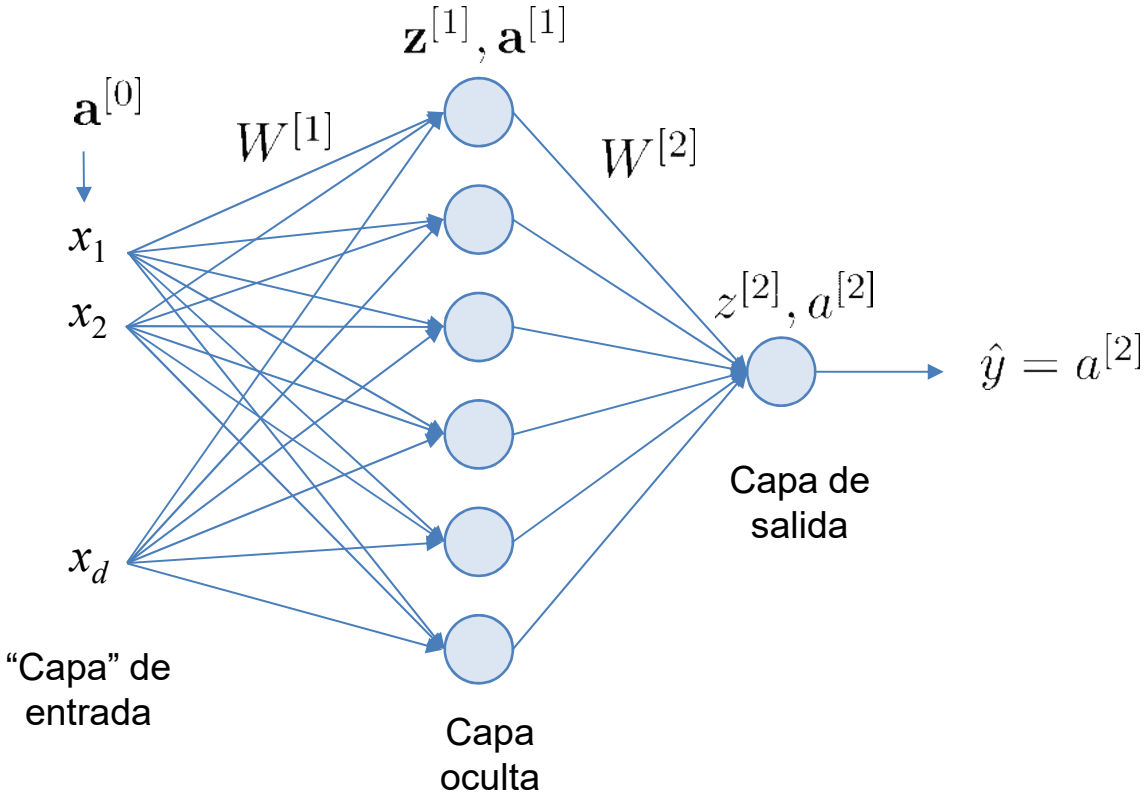
$$a(z) = \text{máx}(0.01z, z) = \begin{cases} 0.01z, & \text{si } z < 0 \\ z, & \text{si } z \geq 0 \end{cases}$$



Redes Neuronales

Esquema General

- Red neuronal con una capa oculta y una capa de salida



- En general puede haber más de 1 salida (para clasificar varias clases)

Redes Neuronales

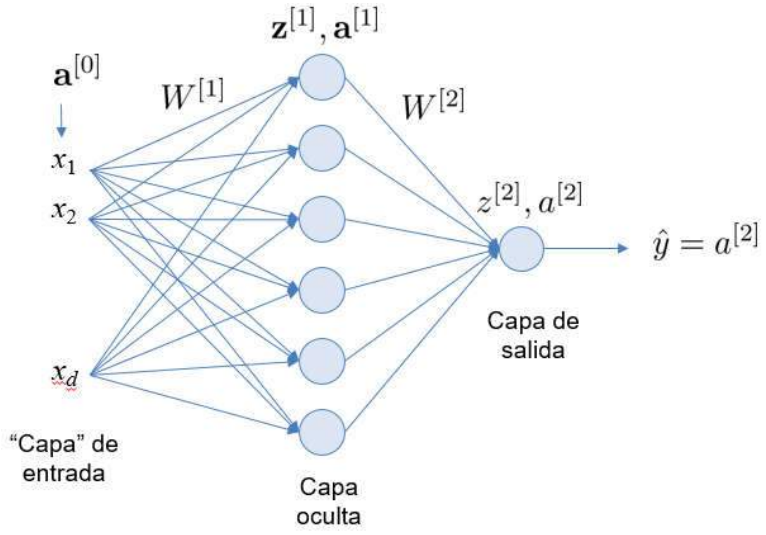
Esquema Genérico

- Elementos:

  - Entrada (capa de entrada): $\boldsymbol{x} = \boldsymbol{a}^{[0]} \in \mathbb{R}^d$

  - Capa oculta
    - Número de neuronas: $n^{[1]}$
    - Pesos $W^{[1]} \in \mathbb{R}^{n^{[1]} \times d}$, y bias $b^{[1]} \in \mathbb{R}$
    - Función de propagación: $\boldsymbol{z}^{[1]} \in \mathbb{R}^{n^{[1]}}$
    - Activación: $\boldsymbol{a}^{[1]} \in \mathbb{R}^{n^{[1]}}$

  - Capa de salida
    - Número de neuronas: $n^{[2]}$
    - Pesos $W^{[2]} \in \mathbb{R}^{1 \times n^{[1]}}$, y bias $b^{[2]} \in \mathbb{R}$
    - Función de propagación: $z^{[2]} \in \mathbb{R}$
    - Activación: $a^{[2]} \in \mathbb{R}$



Redes Neuronales

Función de Hipótesis

- En la capa oculta: [1]
  - Funciones de propagación:

$$z_1^{[1]} = \boldsymbol{w}_1^{[1]T}\boldsymbol{x} + b_1^{[1]} \qquad z_2^{[1]} = \boldsymbol{w}_2^{[1]T}\boldsymbol{x} + b_2^{[1]} \qquad \cdots \qquad z_{n_1}^{[1]} = \boldsymbol{w}_{n_1}^{[1]T}\boldsymbol{x} + b_{n_1}^{[1]}$$

  - Funciones de activación

$$a_1^{[1]} = g(z_1^{[1]}) \qquad a_2^{[1]} = g(z_2^{[1]}) \qquad \cdots \qquad a_{n_1}^{[1]} = g(z_{n_1}^{[1]})$$

  - De forma vectorial:

$$\boldsymbol{z}^{[1]} = \begin{bmatrix} \boldsymbol{w}_1^{[1]T}\boldsymbol{x} + b_1^{[1]} \\ \boldsymbol{w}_2^{[1]T}\boldsymbol{x} + b_2^{[1]} \\ \vdots \\ \boldsymbol{w}_{n_1}^{[1]T}\boldsymbol{x} + b_{n_1}^{[1]} \end{bmatrix} \qquad \text{Si: } W^{[1]} = \begin{bmatrix} \boldsymbol{w}_1^{[1]T} \\ \boldsymbol{w}_2^{[1]T} \\ \vdots \\ \boldsymbol{w}_{n_1}^{[1]T} \end{bmatrix} \qquad \boldsymbol{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{n_1}^{[1]} \end{bmatrix}$$

$$\boldsymbol{z}^{[1]} = W^{[1]}\boldsymbol{x} + \boldsymbol{b}^{[1]} \qquad \boldsymbol{a}^{[1]} = g(\boldsymbol{z}^{[1]})$$

Redes Neuronales

Función de Hipótesis

- En la capa de salida: [2]
  - Funciones de propagación: $z_i^{[2]} = \boldsymbol{w}_i^{[2]T} \boldsymbol{a}^{[1]} + b_i^{[2]}$
  - Funciones de activación: $a_i^{[2]} = g(z_i^{[2]})$
  - De forma vectorial:

$$W^{[2]} = \begin{bmatrix} \boldsymbol{w}_1^{[2]T} \\ \boldsymbol{w}_2^{[2]T} \\ \vdots \\ \boldsymbol{w}_{n_2}^{[2]T} \end{bmatrix} \qquad \boldsymbol{b}^{[2]} = \begin{bmatrix} b_1^{[2]} \\ b_2^{[2]} \\ \vdots \\ b_{n_2}^{[2]} \end{bmatrix}$$

$$\boldsymbol{z}^{[2]} = W^{[2]} \boldsymbol{a}^{[1]} + \boldsymbol{b}^{[2]}$$
$$\boldsymbol{a}^{[2]} = g(\boldsymbol{z}^{[2]})$$

- Función de hipótesis (en la capa de salida):

$$\boldsymbol{h}_w(\boldsymbol{x}) = \boldsymbol{a}^{[2]}$$

Asumiendo que hay 2 capas

Redes Neuronales

Función de Costo

- Si solo hubiese una unidad neuronal:

$$J(\boldsymbol{w}) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} \log(h_w(\boldsymbol{x}^{(i)})) + (1 - y^{(i)}) \log(1 - h_w(\boldsymbol{x}^{(i)}))$$

  - n: número de muestras (datos de entrenamiento)
  - $y^{(i)}$: valor (clase) real de la muestra i (0 o 1)
  - $h_w(\boldsymbol{x}^{(i)})$: predicción para la muestra i (función de hipótesis)

- Para toda la red neuronal
  - Suposición: hay K clases

$$J(\boldsymbol{w}, \boldsymbol{b}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} y_k^{(i)} \log(h_w(\boldsymbol{x}^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_w(\boldsymbol{x}^{(i)}))_k)$$

- Se minimiza la función de costo usando "*backpropagation*":
  - Obtiene los valores de las derivadas de J con respecto a $\boldsymbol{w}_i$ y b_i
  - Se usa "gradient descent" para encontrar $\boldsymbol{w}_i$, $\boldsymbol{b}_i$ óptimos: $w_i = w_i - \alpha \frac{\partial J}{\partial w_i}$

Redes Neuronales

Algoritmo asumiendo una instancia

- Propagación hacia adelante ("predicción"): obtiene la predicción

$$\mathbf{z}^{[1]} = W^{[1]}\mathbf{a}^{[0]} + b^{[1]}$$
$$\mathbf{a}^{[1]} = g^{[1]}(\mathbf{z}^{[1]})$$

} para la capa oculta (capa 1)

$$\mathbf{z}^{[2]} = W^{[2]}\mathbf{a}^{[1]} + b^{[2]}$$
$$\mathbf{a}^{[2]} = g^{[2]}(\mathbf{z}^{[2]}) = \hat{y}$$

} para la capa de salida (capa 2)

- Propagación hacia atrás ("backpropagation"): obtiene las derivadas parciales

$$d\mathbf{z}^{[2]} = \mathbf{a}^{[2]} - \mathbf{y}$$
$$dW^{[2]} = d\mathbf{z}^{[2]}\mathbf{a}^{[1]T}$$
$$db^{[2]} = d\mathbf{z}^{[2]}$$

} para la capa de salida (capa 2) ← asumiendo que $a^{[2]}$ es sigmoidea

$$dz^{[1]} = W^{[2]T}d\mathbf{z}^{[2]} * \mathbf{g}^{[1]'}(\mathbf{z}^{[1]})$$
$$dW^{[1]} = d\mathbf{z}^{[1]}\mathbf{x}^{T}$$
$$db^{[1]} = d\mathbf{z}^{[1]}$$

} para la capa oculta (capa 1)

multiplicación término a término

Notación

$$\frac{\partial J}{\partial \mathbf{z}} = d\mathbf{z}$$
$$\frac{\partial J}{\partial W} = dW$$
$$\frac{\partial J}{\partial b} = db$$
$$\mathbf{g}^{[1]'}(\mathbf{z}^{[1]}) = \frac{dg^{[1]}}{d\mathbf{z}^{[1]}}$$

Redes Neuronales

Algoritmo considerando n instancias

- El entrenamiento se realiza con n instancias
  - Todas las instancias $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ se agrupan en una matriz $X \in \mathbb{R}^{d \times n}$ para evitar bucles en la implementación

$$X = \begin{bmatrix} \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(n)} \end{bmatrix}$$

  - Las propagaciones usan $A^{[0]} = X$:

    Se genera matrices $Z^{[1]} = [z^{[1](1)}, z^{[1](2)}, \ldots, z^{[1](n)}]$ y $A^{[1]} = [a^{[1](1)}, a^{[1](2)}, \ldots, a^{[1](n)}]$

- Propagación hacia adelante ("predicción")

$$\left.\begin{aligned} Z^{[1]} &= W^{[1]} A^{[0]} + \mathbf{b}^{[1]} \\ A^{[1]} &= g^{[1]}(Z^{[1]}) \end{aligned}\right\}$$ para la capa oculta (capa 1)

$$\left.\begin{aligned} Z^{[2]} &= W^{[2]} A^{[1]} + \mathbf{b}^{[2]} \\ A^{[2]} &= g^{[2]}(Z^{[2]}) \end{aligned}\right\}$$ para la capa de salida (capa 2)

Redes Neuronales

Algoritmo considerando n instancias

- Propagación hacia atrás ("backpropagation")
  - Para la capa de salida (asumiendo que la función de activación es sigmoidea)

$$\frac{\partial J}{\partial Z^{[2]}} = dZ^{[2]} = A^{[2]} - Y$$

$$\frac{\partial J}{\partial W^{[2]}} = dW^{[2]} = \frac{1}{n} dZ^{[2]} A^{[1]T}$$

$$\frac{\partial J}{\partial b^{[2]}} = db^{[2]} = \frac{1}{n} dZ^{[2]T} dZ^{[2]}$$

  - Para la capa oculta

$$\frac{\partial J}{\partial Z^{[1]}} = dZ^{[1]} = W^{[2]T} dZ^{[2]} * \mathbf{g}^{[1]'}(Z^{[1]})$$

$$\frac{\partial J}{\partial W^{[1]}} = dW^{[1]} = \frac{1}{n} dZ^{[1]} X^{T}$$

$$\frac{\partial J}{\partial b^{[1]}} = db^{[1]} = \frac{1}{n} dZ^{[1]T} dZ^{[1]}$$

- Nota: la derivada de g es: $\mathbf{g}^{[1]'}(Z^{[1]}) = \frac{dg^{[1]}}{dZ^{[1]}}$

Redes Neuronales

Algoritmo considerando n instancias

- Inicialización
  - Los pesos $W^{[l]}$ deben inicializarse aleatoriamente con valores pequeños
  - En la propagación hacia adelante, $A^{[0]}$ es X (la entrada)
  - En la propagación hacia atrás $dA^{[l]}$ usa la derivada directa

- En cada iteración:
  - Propagación hacia adelante: de $l = 1$ (capa 1) hasta $l = L$ (última capa)
  - Cálculo de la función de costo J
  - Propagación hacia atrás: de $l = L$ hasta $l = 1$
    - Calcula $dW^{[l]}$, $db^{[l]}$ para cada capa
  - Actualización de los pesos en cada capa:

$$W[l] = W^{[l]} - \alpha dW^{[l]}$$

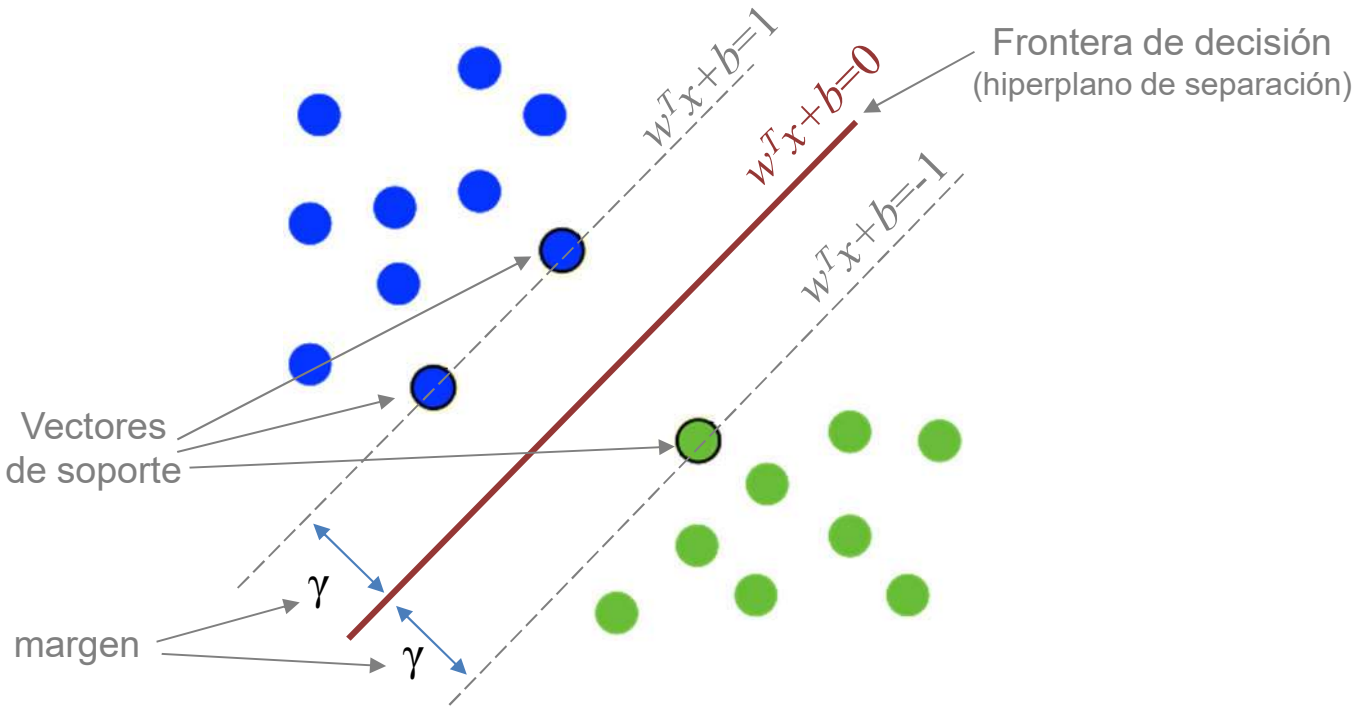
$$b[l] = b^{[l]} - \alpha db^{[l]}$$

Temas

1. Introducción
2. Aprendizaje Supervisado
3. Redes Neuronales
4. **SVM**
5. Otros Métodos
6. Algunas Métricas de Evaluación

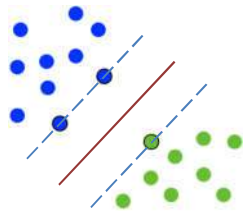
SVM

- Significa "*Support Vector Machine*" (Máquina de Soporte de Vectores)
- Es un clasificador lineal (para ser no lineal se "modifica" con un "kernel")
- Trata de maximizar la separación lineal (margen)
- Ejemplo



SVM

- Hiperplano de separación
  - Se utiliza para separar las 2 clases (frontera de decisión): $\boldsymbol{w}^T\boldsymbol{x} + b = 0$
  - Instancia $x^{(i)}$ en clase 1 si: $\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b > 0$
  - Instancia $x^{(i)}$ en clase -1 si: $\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b < 0$



- Vector de soporte
  - Son las instancias más cercanas al hiperplano de separación
  - Son las instancias con menor seguridad en la predicción (difíciles de clasificar)
  - SVM encuentra hiperplano que se aleja más de los vectores de soporte
- Margen (γ)
  - Es la distancia más pequeña del hiperplano a cualquier instancia del conjunto de entrenamiento

$$\gamma = \frac{1}{\|\boldsymbol{w}\|}$$

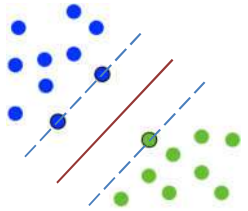
  - SVM busca maximizar el margen

SVM

SVM de Margen Duro (*Hard* SVM)

- *Suposición:* ambas clases son linealmente separables (no hay errores)
- Función de hipótesis
  - Dentro del margen “no es +1 ni -1”

$$h_w(\boldsymbol{x}) = \begin{cases} +1, & \boldsymbol{w}^T\boldsymbol{x} + b \geq 1 \\ -1, & \boldsymbol{w}^T\boldsymbol{x} + b \leq -1 \end{cases}$$



  - Predicción correcta si: $y(\boldsymbol{w}^T\boldsymbol{x} + b) \geq 1$

- Maximización del margen
  - Se considera como restricción la predicción correcta para cada instancia i

$$\max_{\boldsymbol{w},\gamma} \gamma$$

s.a.

$$y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq 1$$

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2$$

s.a.

$$y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq 1$$

SVM

SVM de Margen Duro (*Hard* SVM)

- Solución al problema de optimización
  - Se debe satisfacer las condiciones KKT (Karush-Kuhn-Tucker)
  - Se define el Lagrangiano

$$L(\boldsymbol{w}, b, \alpha) = \frac{1}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} \alpha_i (1 - y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b))$$

  - Se debe satisfacer las siguientes restricciones (KKT):

$$\frac{\partial L}{\partial \boldsymbol{w}} = w^* - \sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} = 0$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

$$\alpha_i \geq 0$$

$$\alpha_i (1 - y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b^*)) = 0$$

$$1 - y^{(i)}(\boldsymbol{w}^T \boldsymbol{x}^{(i)} + b^*) \leq 0$$

SVM

SVM de Margen Duro (*Hard* SVM)

- Solución al problema de optimización
  - Valores óptimos:

$$w^* = \sum_{i=1}^{n} \alpha_i y^{(i)} \boldsymbol{x}^{(i)} \qquad \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

  - Si x_{sv}, y_{sv} son vectores de soporte (hay N_{sv} vectores de soporte):

$$b^* = \frac{1}{N_{sv}} \sum_{i=1}^{N_{sv}} y_{sv}^{(i)} - \boldsymbol{w}^T \boldsymbol{x}_{sv}^{(i)}$$

- Clasificador lineal (óptimo)
  - El clasificador se puede expresar usando w*

$$f = \mathbf{w}^T \mathbf{x} + b \quad \Rightarrow \quad f = \sum \alpha_i y^{(i)} \mathbf{x}^T \mathbf{x} + b$$

  - Solo se requiere el producto punto de $\boldsymbol{x}$

SVM

SVM de Margen Duro (*Hard* SVM)

- Problema dual
  - La solución es igual a la solución del problema inicial

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \boldsymbol{x}^{(i)T} \boldsymbol{x}^{(j)}$$

s.a.

$$\sum_{i=1}^{n} \alpha_i y^{(i)} = 0, \quad i = 1, \cdots, n$$

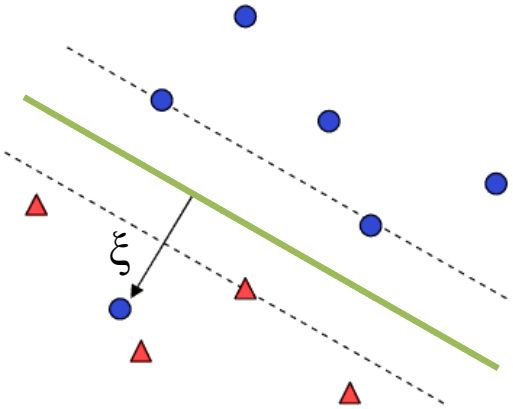
$$\alpha_i \geq 0, \quad i = 1, \cdots, n.$$

  - Se resuelve usando paquetes para QP (quadratic programms)
  - Importante:
    - Solamente está involucrado α
    - La función objetivo tiene el producto punto de $x^{(i)}$ con $x^{(j)}$

SVM

SVM de Margen Blando (*Soft* SVM)

- Asume que las clases no son linealmente separables
- Variable de holgura (ξ)
  - Es el grado de error en la clasificación
  - Distancia del margen de la clase correcta



  - Para toda instancia: $y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq 1 - \xi_i$

$$\xi_i = \text{máx}(0, 1 - y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b))$$

  - Si $\xi > 1$, la instancia está en el lado incorrecto

SVM

SVM de Margen Blando (*Soft* SVM)

- Definición del problema
  - Maximizar el margen y minimizar el error (variable de holgura)
  - Satisfacer las restricciones de la instancia (y de la variable de holgura)

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i$$

s.a.

$$y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b) \geq 1 - \xi_i, \quad i = 1, \cdots, n$$
$$\xi_i \geq 0, \quad i = 1, \cdots, n$$

- Función de costo
  - El problema se puede escribir como una función de costo sin restricciones

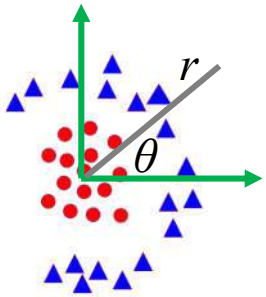
$$J(\boldsymbol{w}, b) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\text{máx}(0, 1 - y^{(i)}(\boldsymbol{w}^T\boldsymbol{x}^{(i)} + b))$$

  - Se puede resolver usando el descenso del gradiente

SVM

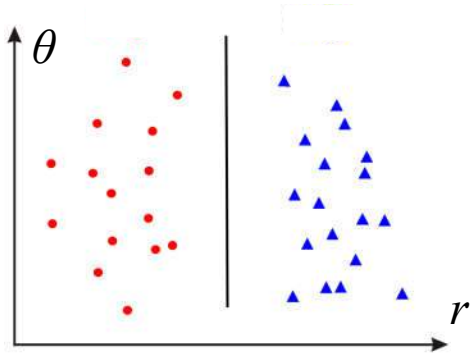
Kernels

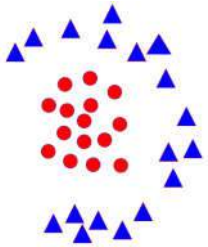
- Idea
  - Se desea separar “linealmente” datos que no son linealmente separables
  - Se realiza transformaciones a los datos (Φ)



$$\Phi : \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} r \\ \theta \end{bmatrix}$$

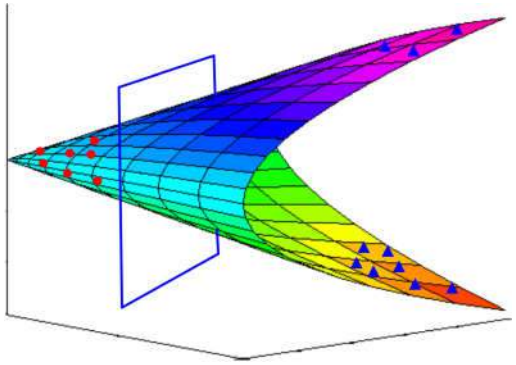
coordenadas polares





$$\Phi : \begin{bmatrix} x \\ y \end{bmatrix} \rightarrow \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2}x_1x_2 \end{bmatrix}$$

mapeo a más dimensiones



[Dibujo de Andrew Zisserman]

SVM

Kernels

- Transformación (Φ) aplicada al clasificador lineal

$$f(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x}) + b$$

$$f(\mathbf{x}) = \sum \alpha_i y^{(i)} \underbrace{\Phi(\mathbf{x})^T \Phi(\mathbf{x})} + b$$

Solo importa el producto punto

- Kernel (k):
  - Es el producto punto de la transformación

$$k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x})^T \Phi(\mathbf{x})$$

  - No es necesario calcular explícitamente la transformación ("truco" del kernel)
  - Ejemplos de kernels:
    - Kernels lineales: $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
    - Kernels polinomiales (grado d): $k(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^d$
    - Kernels Gaussianos:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\left\| \mathbf{x}_i - \mathbf{x}_j \right\|^2}{2\sigma^2} \right)$$

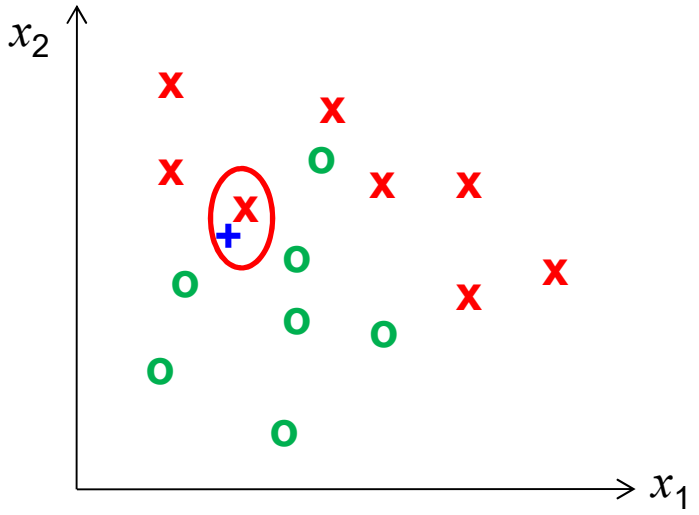
← RBF (Radial Basis Function) SVM

Temas

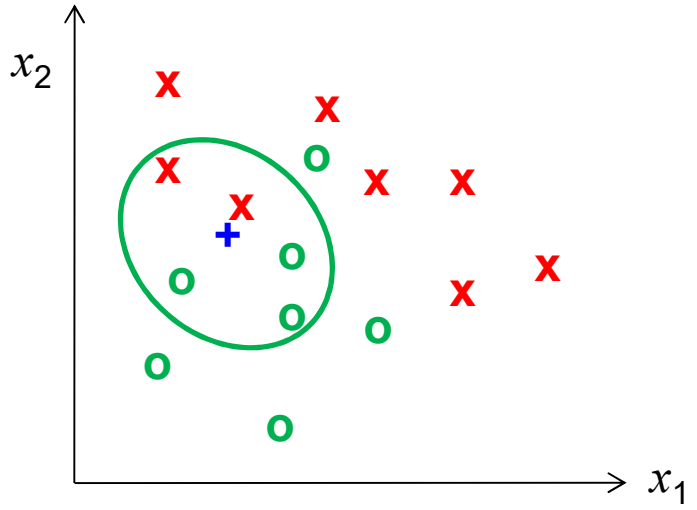
1. Introducción
2. Aprendizaje Supervisado
3. Redes Neuronales
4. SVM
5. **Otros Métodos**
6. Algunas Métricas de Evaluación

K-Nearest Neighbor

- Asigna la etiqueta de los k puntos más cercanos



1-nearest neighbor



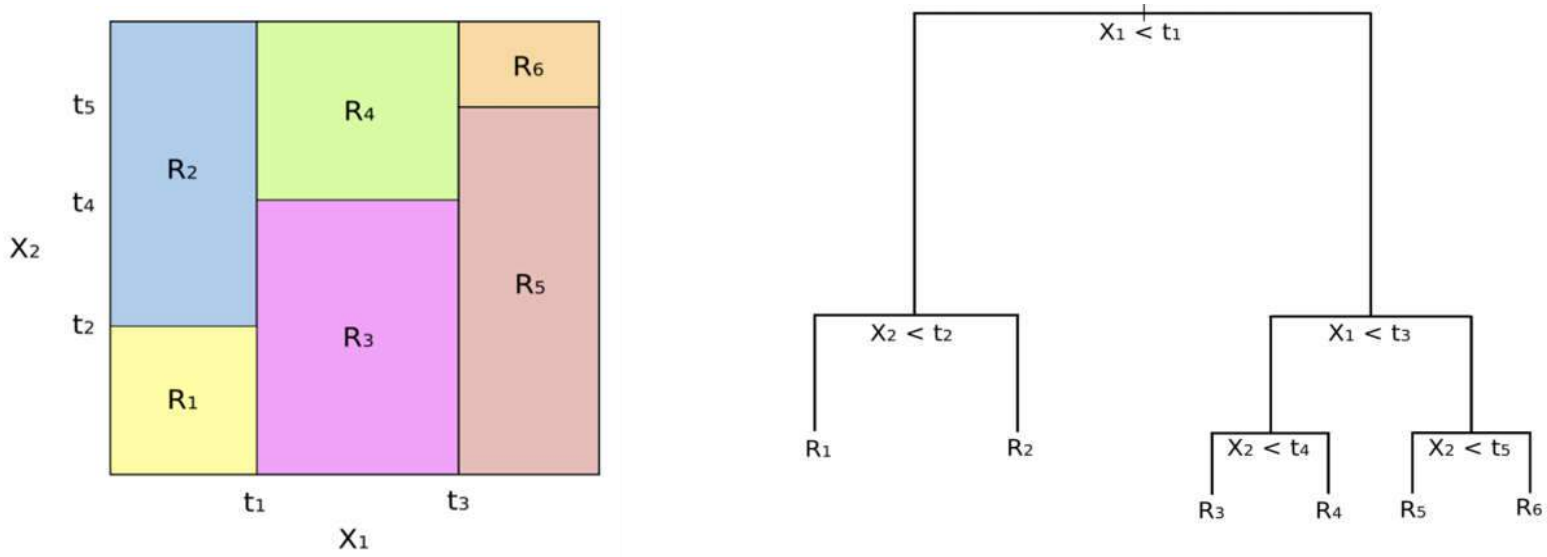
5-nearest neighbor

  - Si k es pequeño: sensible a puntos con ruido ("falsos")
  - Si k es grande: la vecindad puede incluir a puntos de otras clases

- Requiere "memorizar" los datos (no es rápido para la predicción)

Árboles de Decisión

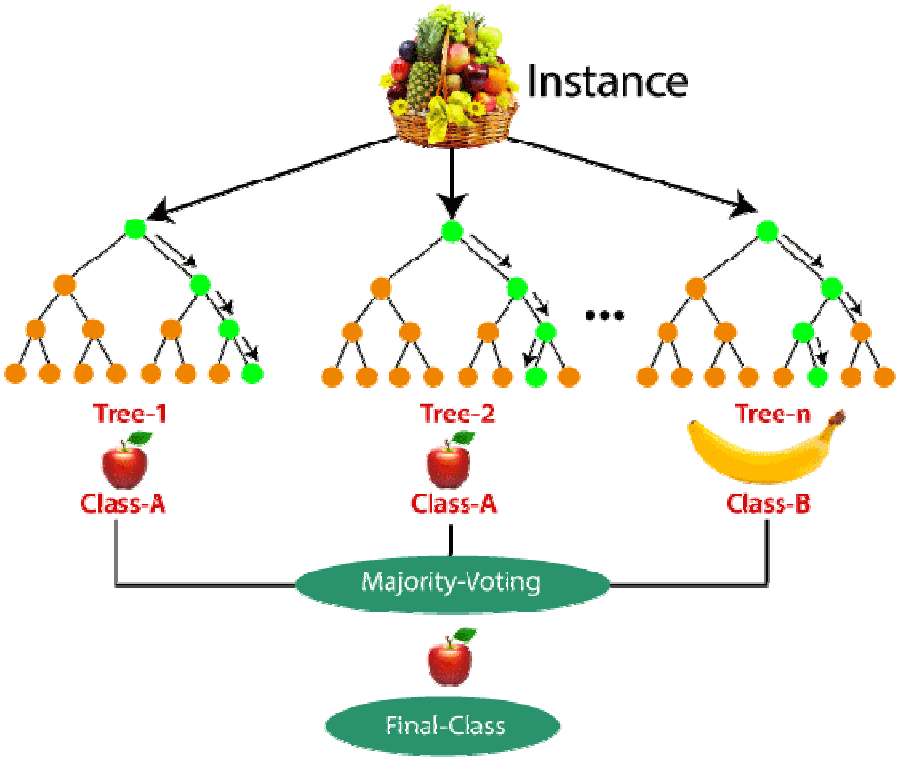
- Divide el espacio en regiones distintas
- Se basa en minimizar la entropía (desorden de las regiones)



- Se puede aplicar a datos continuos o categóricos
- Existen otros índices para medir el “desorden” de las regiones
  - Ejemplo: Índice Gini

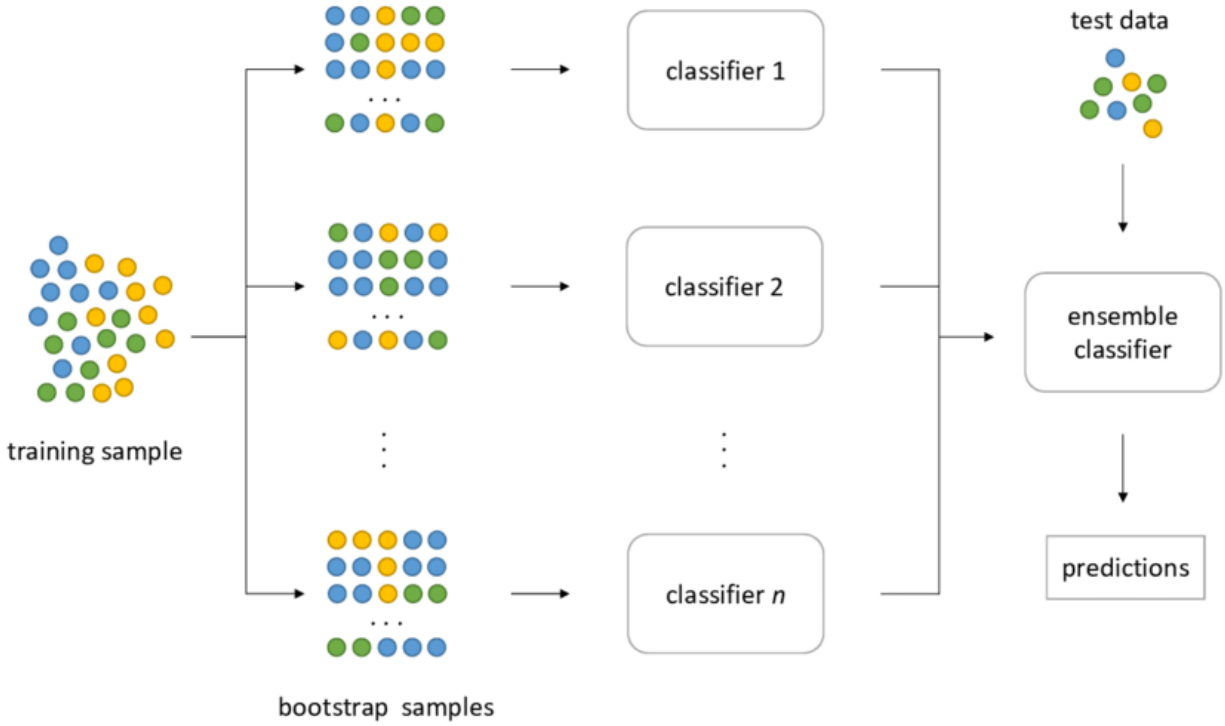
Random Forest

- Aplica varios árboles de decisión:
  - Submuestrea datos
  - Submuestrea atributos
- Se aplica un esquema de “votación” para clasificar



Bagging

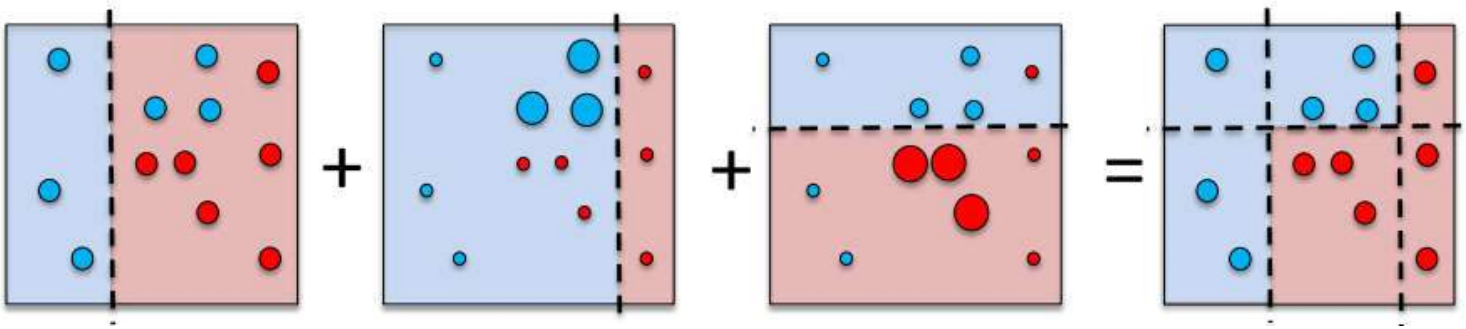
- Se genera muestras usando “bootstrap” (método de muestreo)
- Se aplica un clasificador en cada muestra
- Se “combina” los resultados (agregación)



Boosting

- Clasificador débil (*weak*): h_i
  - Solo un poco mejor que un clasificador aleatorio

- Procedimiento
  - Aplica (clasificador débil) h_1: se observa los resultados incorrectos
  - Se da más peso a los resultados incorrectos y se aplica h_2: se observa resultados incorrectos
  - Se repite y al final se combina los clasificadores h_i



- Ejemplo: AdaBoost

Temas

1. Introducción
2. Aprendizaje Supervisado
3. Redes Neuronales
4. SVM
5. Otros Métodos
6. **Algunas Métricas de Evaluación**

Matriz de Confusión

- Es un arreglo bidimensional que presenta:
  - Cantidad de instancias clasificadas como 0 y 1 (valores estimados)
  - Cantidad de instancias que realmente son 0 y 1 (valore reales)

		Valor estimado $\hat{y}=1$	Valor estimado $\hat{y}=0$	
Valor real	$y=1$	TP	FN	n_P
	$y=0$	FP	TN	n_N
		$\hat{n}_p$	$\hat{n}_N$	

$n_P = TP+FN$
$n_N = FP+TN$

- Terminología
  - TP (*True Positive*): instancias clasificadas como 1 cuando realmente son 1
  - TN (*True Negative*): instancias clasificadas como 0 siendo realmente 0
  - FP (False Positive): instancias clasificadas como 1 cuando son 0.
  - FN (False Negative): instancias clasificadas como 0 cuando son 1

Métricas

- Exactitud: Cantidad de instancias correctamente clasificadas con respecto al total de instancias

$$A = \frac{TP + TN}{n}$$

- Precisión: Indica de todas las instancias que han sido clasificadas como positivas, cuántas son realmente positivas

$$P = \frac{TP}{\hat{n}_P} = \frac{TP}{TP + FP}$$

- Error Tipo I: Cantidad de instancias negativas que han sido incorrectamente clasificadas como positivas (FPR: *False Positive Rate*)

$$E_1 = \frac{FP}{n_N} = \frac{FP}{FP + TN}$$

- Error Tipo II: Cantidad de instancias positivas que han sido incorrectamente clasificadas como negativas (FNR: *False Negative Rate*)

$$E_2 = \frac{FN}{n_P} = \frac{FN}{TP + FN}$$

Valores F

- Son combinaciones de precisión P y sensitividad R (*recall*)

$$P = \frac{TP}{\hat{n}_P} = \frac{TP}{TP + FP} \qquad R = \frac{TP}{n_P} = \frac{TP}{TP + FN}$$

- En general se define como:

$$F_\beta = (1 + \beta^2)\frac{PR}{\beta^2 P + R}$$

  - Si $\beta = 0$: $F_0 = P$
  - Si $\beta \to \infty$: $F_\infty = R$
  - Si $\beta = 1$: $F_1 = \frac{2}{\frac{1}{P} + \frac{1}{R}} = \frac{2PR}{P + R}$
    - Es una media armónica de P y R (siempre más cerca al menor valor de P o R)
    - Permite determinar clasificadores no son muy buenos independientemente de si P o R es más importante para el problema.
    - Un valor alto de P cuando R tiene un valor bajo no genera resultado muy alto (y viceversa)

Referencias

- Bishop, C. (2006) *Pattern recognition and machine learning*. Springer

- Hastie, T., Tibshirani, R., Friedman, J. (2009). *The elements of statistical learning*. Springer Science & Business Media.

- Murphy, K. (2012). *Machine Learning, A Probabilistic Perspective*. The MIT Press.