Aprendizaje automático

Departamento de Ingeniería del Software
e Inteligencia Artificial

Aprendizaje automático

- Introducción
- Aprendizaje no supervisado
  - Algoritmos de agrupamiento jerárquico
  - Algoritmos de agrupamiento basado en particiones
- Aprendizaje supervisado
  - k vecinos más cercanos
  - Árboles de decisión
  - Perceptrón multicapa

INTRODUCCIÓN

¿Qué entendemos por aprendizaje?

- No se puede hablar de inteligencia sin aprendizaje

- La **capacidad de aprendizaje** permite realizar nuevas tareas que previamente no podían realizarse, o bien realizar mejor (más rápidamente, con mayor exactitud, etc.) las que ya se realizaban, como resultado de los cambios producidos en el sistema al resolver problemas anteriores.

La capacidad de aprendizaje no puede añadirse a posteriori.

- Aprendizaje
  - "El acto, proceso o experiencia de adquirir conocimiento o aptitudes"

Aprendizaje Automático *(machine learning)*

- La **IA simbólica** trata con representaciones simbólicas de alto nivel (cercanas al entendimiento humano). Adecuada para:
  - Representar conocimiento humano y razonar con él
  - Resolver problemas bien-definidos o de índole lógica
- La **IA subsimbólica** es capaz de tratar con representaciones cercanas al problema y extraer conocimiento de ellas
- Existen aproximaciones de aprendizaje de conceptos simbólicas
  - Ej. Algoritmos de Winston y de Mitchell
- Sin embargo, el aprendizaje automático ha experimentado una explosión en su desarrollo desde la década de los 90 gracias a aproximaciones subsimbólicas que se benefician de
  - La abundancia de datos almacenados
  - El aumento en la capacidad de cálculo de los ordenadores

Aprendizaje Automático *(machine learning)* e IA subsimbólica

- En el aprendizaje automático, mediante algoritmos de aprendizaje se extraen “reglas” o “patrones” de los datos.
- Las aproximaciones de aprendizaje subsimbólico proporcionan
  - Mayor robustez frente al ruido
  - Mayor capacidad de escalamiento (nuevos ejemplos o nuevas variables)
  - Mayor capacidad para trabajar con cantidades ingentes de datos
  - Menor dependencia del experto
    - Éste puede ayudar a la selección de variables o formas de representación relevantes y a la interpretación de los resultados

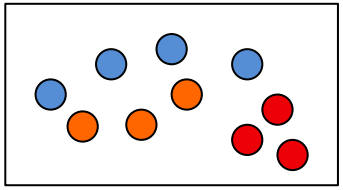
Tipos de Aprendizaje

- La IA subsimbólica está relacionada con la estadística
  - Sin embargo, la estadística requiere que se cumplan hipótesis y sigue una aproximación más “formal”
  - Mientras que la IA subsimbólica sigue una aproximación más “ingenieril”

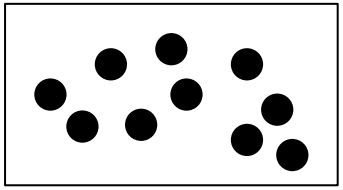
- Aunque existen diferentes formas de clasificar el aprendizaje nos centraremos en el aprendizaje según el grado de realimentación:
  - Supervisado,
  - No supervisado
  - Por refuerzo

Tipos de aprendizaje: según el grado de realimentación

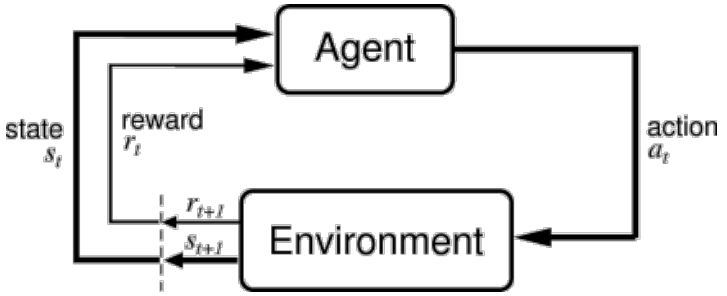
- Aprendizaje supervisado
  - Hay que suministrar al sistema ejemplos clasificados
  - El objetivo es descubrir “reglas” de clasificación



- Aprendizaje no supervisado o por descubrimiento
  - No hay información sobre la clase a la que pertenecen los ejemplos
  - Objetivo: descubrir patrones en el conjunto de entrenamiento que permitan agrupar y diferenciar unos ejemplos de otros



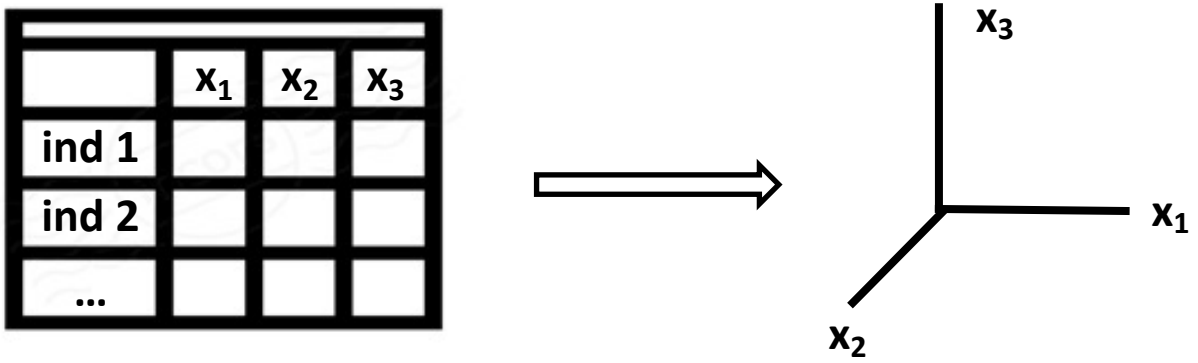
- Aprendizaje por refuerzo
  - El sistema recibe algún tipo de recompensa (positiva o negativa) cada vez que produce una respuesta, ajustando su comportamiento en función de dicha recompensa
  - Típico sistema de aprendizaje robótico
  - No lo veremos en esta asignatura



REPRESENTACIÓN DE LOS INDIVIDUOS

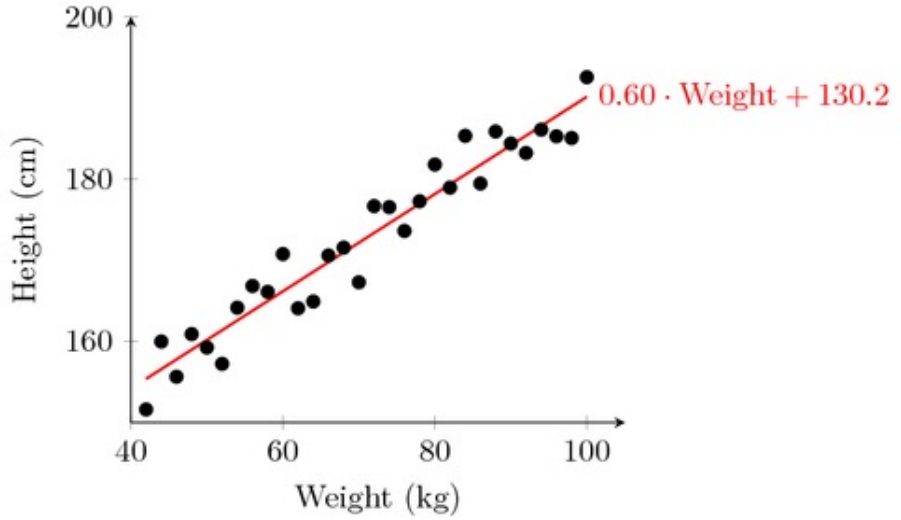
Representación de los individuos

- En el aprendizaje supervisado y no supervisado partimos de datos que son un conjunto de descripciones de objetos
- Tendremos un número de individuos o elementos (n) descrito por un conjunto de variables (m)
  - Nuestros datos se representan en forma de **matriz $n \times m$** donde las filas son los individuos y las columnas las variables
  - Las variables pueden ser de diferente naturaleza
    - Cuantitativa: variables reales o enteras
    - Cualitativa: variables categóricas u ordinales (p.ej. grupos de edad)
  - Las variables son las dimensiones en las que representamos a los individuos
    - P.ej. Si tenemos 3 variables cuantitativas nuestros individuos pueden considerarse como puntos en un espacio tridimensional



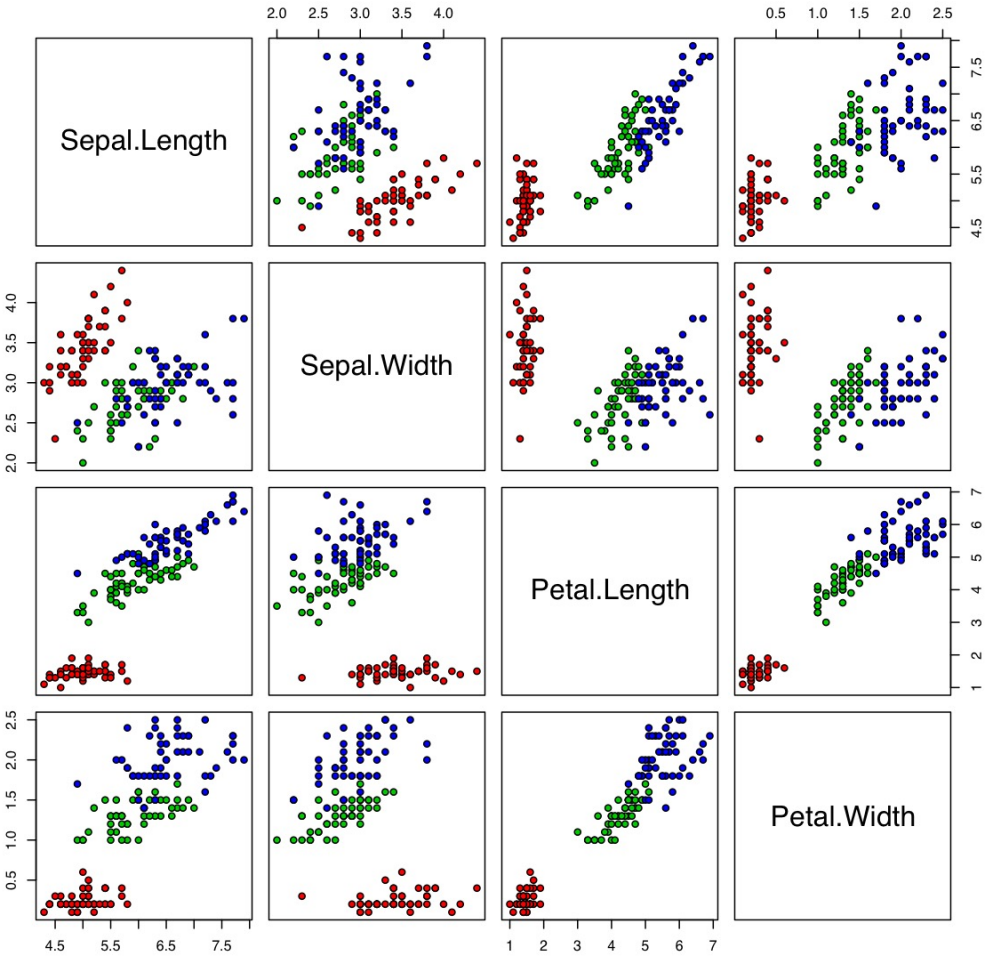
Representación de los individuos en el espacio

- Nuestros individuos son puntos representados en un espacio m-dimensional
  - De hecho, así los consideran muchos algoritmos de aprendizaje automático
- Suele ser de gran ayuda poder inspeccionar los datos visualmente, para ello se suele utilizar el diagrama de dispersión (*scatter plot* en inglés)
  - Se usa habitualmente para ajustar una regresión lineal



- Nosotros no podemos ver los datos en más de dos (o tres) dimensiones
  - Sin embargo, podemos mostrar las variables 2 a 2, si no son demasiadas

Ejemplo de diagramas de dispersión



Conjunto de datos de la flor del Iris

Es un conjunto clásico de aprendizaje estadístico

Variables de representación:

- Longitud del sépalo (cuantitativa)
- Ancho del sépalo (cuantitativa)
- Longitud del pétalo (cuantitativa)
- Ancho del pétalo (cuantitativa)
- Especie de Iris (categórica: setosa, virginica, versicolor)

Cada gráfico muestra las variables cuantitativas dos a dos y la especie usando el color.

El color es la “tercera” dimension en cada gráfico (**rojo** setosa, **azul** virginica y **verde** versicolor)

Importancia de la adecuada representación de los individuos

- Los algoritmos de Aprendizaje Automático son sensibles a la forma en que los individuos de nuestro problema están representados. Por ejemplo:
  - La representación puede contar con variables redundantes o no relevantes para el problema
    - Sin embargo, no siempre se puede determinar a priori cuáles son
  - Los datos se verán afectados por ruido, errores de medida, valores perdidos
- Los algoritmos de aprendizaje automático trabajan bien cuando el conjunto de datos tiene numerosos individuos que cubren bien la casuística que se puede dar en la vida real
  - Un algoritmo puede aprender sobre individuos “parecidos” a los que ya conoce, pero si hay “regiones” del espacio de representación sin individuos, de ellas es complicado aprender nada
    - Si cuando el algoritmo está en producción se le presenta un individuo que no se parece a los que empleó en su aprendizaje, su comportamiento será imprevisible!!!
  - Esto es más relevante cuanto mayor es el número de variables (como veremos más adelante).

Preparando los datos para el análisis

- Es una tarea crucial que afecta a los resultados, aunque no ahondaremos en ello. Incluye operaciones como:
  - Selección de variables relevantes para el problema, puede incluir
    - Transformación de variables iniciales
      - Cambiar la escala de representación de las variables (normalizar)
        - » Típicamente, las variables se escalan (se ponen en escala entre 0 y 1) o se estandarizan (se hace que tengan media 0 y varianza 1)

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}} \qquad x' = \frac{x - \bar{x}}{\sigma}$$

escalar ***estandarizar***

¿Cuál uso? Depende del problema pero en general:
- valores pequeños y rango pequeño -> no hace falta
- distribución original normal -> estandarizar
- en otro caso -> escalar

        - » También se puede usar la transformada logarítmica de una variable para "concentrar" su rango,
        - » etc.
      - Discretizar una variable numérica, es decir, dividir su rango de valores posibles en categorías ordenadas
        - » Edad: [0 – 100] ➔ Bebé, Niño, Adolescente, Joven, Adulto, Anciano
      - Transformación de variables categóricas en variables binarias (una variable por categoría)
        - » Cuando las categorías no tienen orden

	Nacionalidad
id1	Español
id2	Frances

	Español	Francés
id1	1	0
id2	0	1

Preparando los datos para el análisis

- Combinación de variables iniciales
  - P. ej. Agrupando aquellas variables que están muy correlacionadas
- Eliminación de variables no relevantes o redundantes

– Tratamiento de valores perdidos. Es posible que algún valor de alguna variable para algún individuo no esté disponible en nuestra matriz
  - Si faltan pocos datos quizás podemos descartar esas filas
  - Otras opciones: asignar el valor medio en variables cuantitativas,0 o el valor modal (el más repetido) en variables cualitativas

	Edad	**Altura**	**Peso**	**Educación**
id1	20	175	80	Grado
id2	?	158	?	Secundaria
id3	28	166	65	Grado
id4	24	190	83	?
…	…	…	…	…

Entendiendo los datos

- Antes de acometer una tarea de aprendizaje automático conviene entender los datos lo mejor posible, para ello suele ser recomendable "trabajarlos" previamente
  - Visualización de datos:
    - Permite determinar qué variables están más relacionadas entre sí y cuál es la naturaleza de la relación (lineal, exponencial, etc)
    - En los problemas de clasificación permite ver cómo de bien se separan las clases y qué variables separan mejor
  - Calcular estadísticos descriptivos:
    - Tendencia central: media, mediana, moda
    - Dispersión: desviación típica, valores mínimos y máximos, percentiles
    - Bivariantes: coeficiente de correlación o tablas de contingencia
  - Representar la distribución de las variables
    - Observar valores más frecuentes y valores extremos (¿tienen sentido o son aberrantes?)
  - Usar técnicas de **aprendizaje no supervisado** para encontrar *estructura* en los datos

APRENDIZAJE NO SUPERVISADO

Aprendizaje no supervisado

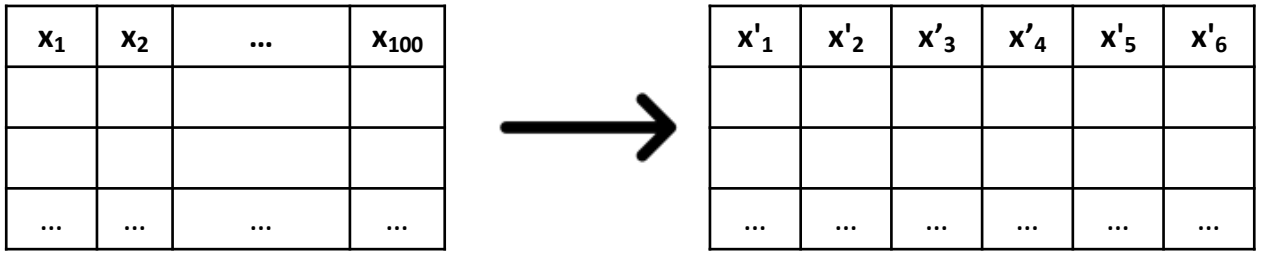
- Su objetivo es encontrar “estructura” en los datos proporcionados sin atender a ninguna categoría prefijada
  - Se les conoce también como técnicas exploratorias
- Típicamente se busca estructura en los individuos o en las variables
  - Estructura en los individuos: **técnicas de agrupamiento o *clustering***
  - Estructura en las variables: **técnicas de reducción de la dimensionalidad**
    - Nosotros nos centraremos en las técnicas de agrupamiento
- La estructura nos permite
  - “Reducir” nuestro conjunto de datos (reducir la dimensión de las variables o la de los individuos a grupos)
  - Ganar comprensión sobre los datos (a nivel de variables y/o de los individuos)
    - Ver qué variables están más (o menos) relacionadas entre sí y ver cómo se agrupan los individuos según su similitud

El problema de la dimensionalidad

- A menudo contar con muchas variables es un problema
  - "Maldición de la dimensionalidad": Podemos tener un número de dimensiones/variables muy elevado y no contar con suficientes ejemplos en muchas regiones de ese espacio multidimensional
    - Lo que "aprendamos" de esas regiones seguramente sea espúreo
  - Existen variables irrelevantes o redundantes
    - Algunas técnicas de aprendizaje no son capaces de elegir las variables relevantes para el problema y pueden confundirse si incluimos variables irrelevantes
  - Complica la visualización de los datos y su comprensión por el humano
    - ¿Cómo visualizar puntos con 100 o 1000 dimensiones?

Reducción de la dimensionalidad

- En las técnicas de reducción de la dimensionalidad
  - Se parte de un conjunto de variables m (es decir, m dimensiones)
  - Se busca reducirlo en un conjunto p mucho menor de factores que conserven el máximo posible de la información inicial (es decir, la variabilidad de las m variables)
  - En muchas de estas técnicas los factores se obtienen como una combinación de las variables originales
    - El Análisis de Componentes Principales (PCA) es la técnica clásica y usa combinación lineal
    - Existen variantes más sofisticadas para hacer combinaciones no lineales usando kernels



$$x'_1 = w_{1,1}x_1 + w_{1,2}x_2 + \cdots + w_{1,100}x_{100}$$
$$\cdots$$
$$x'_6 = w_{6,1}x_1 + w_{6,2}x_2 + \cdots + w_{6,100}x_{100}$$

Reducción de la dimensionalidad

- La interpretación de los p factores resultantes nos habla de dimensiones "ocultas" y de la estructura de las variables originales
  - De las variables originales habrá algunas que "contribuyen" más a un factor que a otros, lo que quiere decir que están más relacionadas con dicho factor
  - Las variables que contribuyen más en un factor están relacionadas entre sí
    - P.ej. En un conjunto de datos de pacientes variables como el peso y la altura estarán correlacionadas y podrían quedar agrupadas en un factor que nos hable del tamaño del individuo

APRENDIZAJE NO SUPERVISADO
TÉCNICAS DE CLUSTERING

Técnicas de agrupamiento o clustering

- El objetivo es agrupar los n individuos de nuestro conjunto de datos en una serie de grupos de forma que
  - Los individuos del mismo grupo sean lo más parecidos posible entre sí
  - Los individuos de grupos diferentes sean lo más diferentes entre sí
- De esta forma los grupos nos revelarán cierta "estructura" de los individuos de nuestro conjunto de datos, por ejemplo:
  - Cuáles son los grupos más numerosos y menos numerosos
  - Cuáles son los grupos más homogéneos y más dispersos
  - Qué individuos están más "alejados" de su grupo (outliers) o forman un grupo propio
- El concepto de parecido o de similitud suele requerir el uso de una medida de disimilitud o de distancia
  - Matemáticamente no toda disimilitud es una distancia
- Existen muchas familias de algoritmos de agrupamiento, aunque se suelen dividir en dos grandes grupos de los que veremos un ejemplo:
  - **Algoritmos de clustering jerárquico**
  - **Algoritmos de clustering basados en particiones**

Algoritmos de clustering jerárquico aglomerativo

- En un algoritmo de clustering jerárquico aglomerativo la estrategia general es la siguiente:
  1. Cada individuo empieza siendo un *cluster*, es decir, hay n *clusters*, tantos como individuos
  2. Repetir hasta que todos los individuos formen un único *cluster*
     - Agrupar los *clusters* más próximos en un único *cluster*

- Necesitamos definir:
  - La distancia (o disimilitud) que se usa para medir la proximidad (o similitud)
  - ¿Cómo se calcula la distancia entre *clusters* con más de un individuo?

- Existe una gran cantidad de distancias que pueden usarse
  - Su elección no debe ser casual ya que usar una distancia u otra hace que "priorices" unos aspectos frente a otros

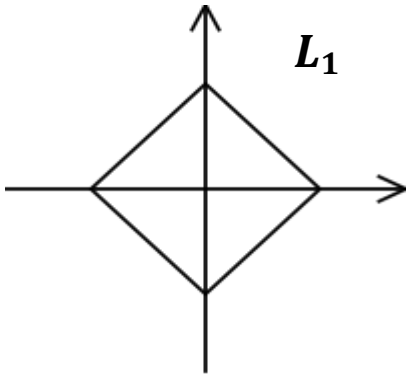
Distancias entre individuos

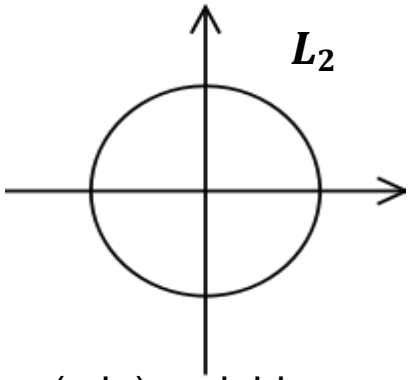
- La familia de métricas de Minkowski (L_p) suelen usarse habitualmente, especialmente sus variantes más famosas (Manhattan y Euclídea)
  - Sean dos individuos A y B descritos por m variables X_i con $i = 1, \dots, m$ definimos las distancias

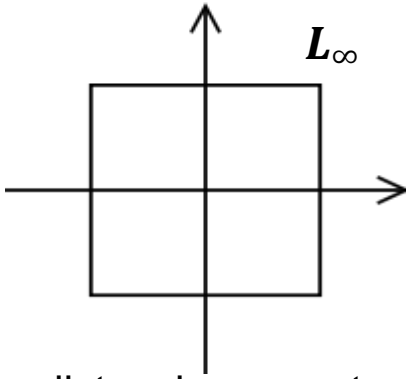
Manhattan: $L_1(A,B) = \sum_{i=1}^{m} |x_{Ai} - x_{Bi}|$

Euclídea: $L_2(A,B) = (\sum_{i=1}^{m} (x_{Ai} - x_{Bi})^2)^{1/2}$

Chebychev: $L_\infty(A,B) = max_i |x_{Ai} - x_{Bi}|$







En estas imágenes se muestran los puntos que se encuentran a la misma distancia en un espacio bidimensional con cada una de estas distancias

  - La importancia que tienen las (o la) variables con mayor distancia aumenta en $L_1 < L_2 < L_\infty$
    - En L_1 damos igual valor a todas las diferencias, en L_2 el cuadrado hace que pesen más las diferencias grandes y en L_∞ solamente se tiene en cuenta la variable donde la diferencia es mayor
- Es importante tener en cuenta que cuando tenemos varias variables, la magnitud en la que se mueven sus valores puede afectar a la distancia
  - No es lo mismo medir distancia entre dos variables una en centímetros y otra en kilos, que si lo hacemos en metros y gramos
  - Para evitar estos efectos se suelen **normalizar** o **estandarizar** los datos

Distancias entre *clusters*

- Existen varias estrategias para medir distancias entre *clusters*
  - Se utiliza una de ellas durante todo el algoritmo
  - La estrategia usada afectará a la forma final de los *clusters*
- Las más típicas son:
  - **Centroide**: Se toma la distancia entre los puntos medios (el vector medio) de los dos *clusters*
  - **Enlace simple (*single linkage*)**: Se toma la distancia entre los puntos más próximos de los dos *clusters*
  - **Enlace completo (*complete linkage*)**: Se toma la distancia entre los puntos más alejados de los dos *clusters*

Ejemplo con dos clusters en un espacio bidimensional

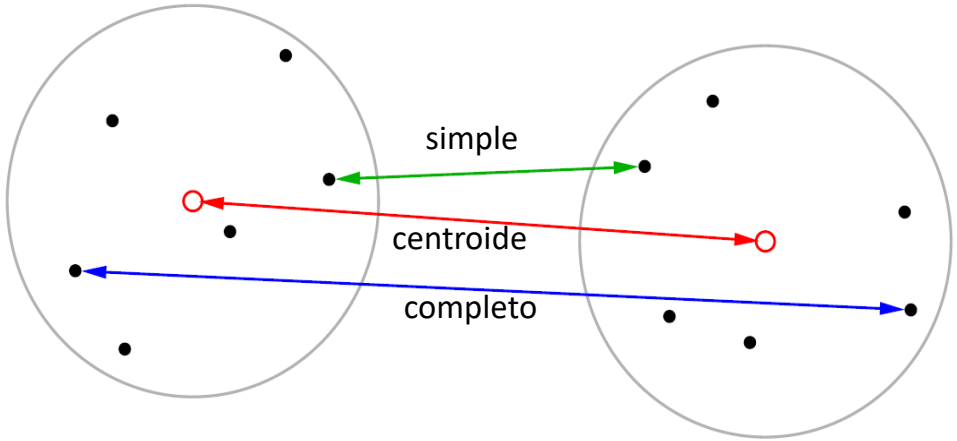


Imagen de C. Borgelt (2012). Intelligent Data Analysis

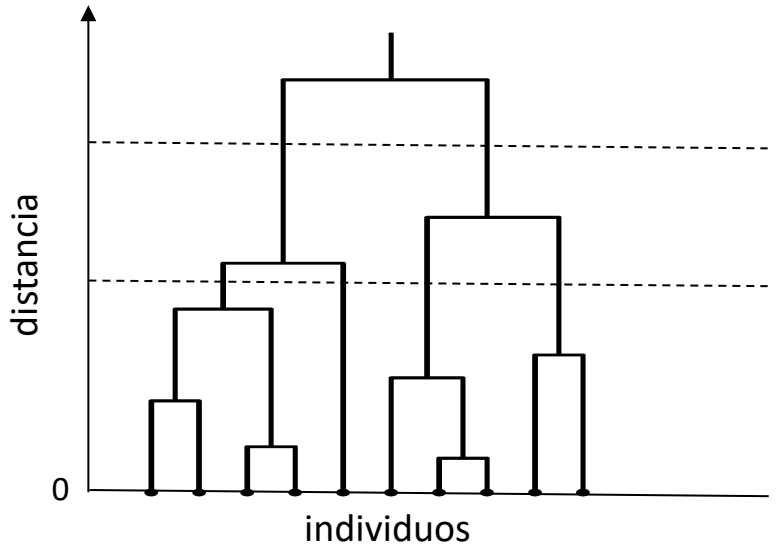
Algoritmo extendido

- FASE 1: Crear la matriz de distancias inicial D
  - Es una matriz simétrica (basta con usar una de las matrices triangulares)
- FASE 2: Agrupación de Individuos
  1. Partición inicial P_o: Cada objeto es un *cluster*
  2. Calcular la partición siguiente usando la matriz de distancias D
     - Elegir los dos *clusters* más cercanos
       - Serán la fila y la columna del mínimo de la matriz D
     - Agrupar los dos en un *cluster*
       - Eliminar de la matriz la fila y columna de los *clusters* agrupados
       - Generar la nueva matriz de distancias D
         - » Añadir una fila y una columna con el *cluster* nuevo
         - » Calcular la distancia del resto de *clusters* al cluster nuevo
  3. Repetir paso 2 hasta tener sólo un *cluster* con todos los individuos
     - Representar el dendograma (árbol de clasificación)

- La complejidad con los enlaces simple y completo son computacionalmente costosas, $O(n^3)$, aunque existen implementaciones eficientes más livianas en $O(n^2)$

Dendrograma

- El dendrograma (*dendro* es árbol en griego) es una representación bidimensional de la jerarquía inferida por el algoritmo de *clustering* jerárquico
  - En un eje ponemos los individuos y los *clusters* (abcisas en nuestro caso)
    - El orden de los mismos favorecerá la representación del dendrograma para que no haya cruces
  - El otro eje representa la distancia (ordenadas en nuestro caso)
  - El gráfico representa que la unión entre dos *clusters* se produce a una distancia determinada
- El resultado es una jerarquía en la que podemos ver la estructura de agrupación que ha ido siguiendo el algoritmo



Según el punto de corte que usemos obtendremos un número diferente de clusters

La elección del punto de corte la suele marcar el analista o usar alguna heurística

Ejemplo del Algoritmo

- Dadas estas seis observaciones unidimensionales $\{2, 12, 16, 25, 29, 45\}$
- Podemos calcular su matriz de distancias (independientemente de la distancia elegida)

$$D = \begin{pmatrix} 0 & 10 & 14 & 23 & 27 & 43 \\ 10 & 0 & 4 & 13 & 17 & 33 \\ 14 & 4 & 0 & 9 & 13 & 29 \\ 23 & 13 & 9 & 0 & 4 & 20 \\ 27 & 17 & 13 & 4 & 0 & 16 \\ 43 & 33 & 29 & 20 & 16 & 0 \end{pmatrix}$$

- Según la estrategia de enlace usada obtendremos diferentes dendrogramas

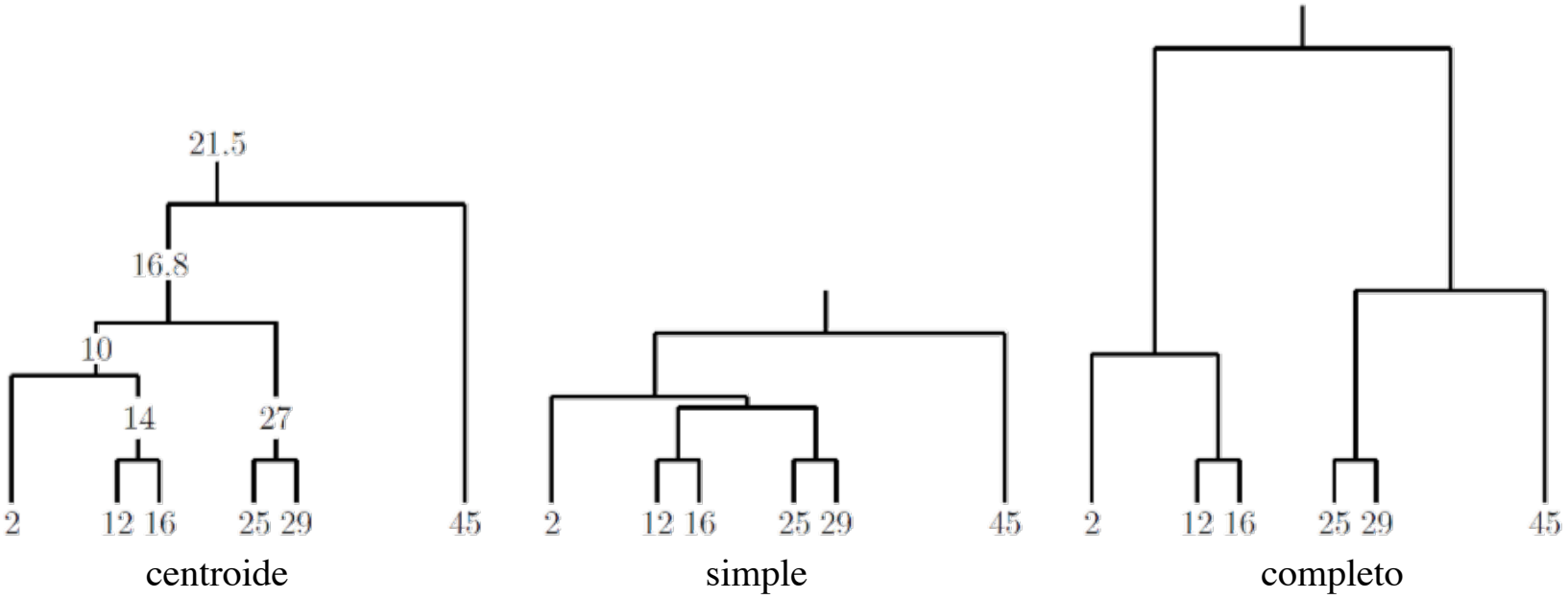
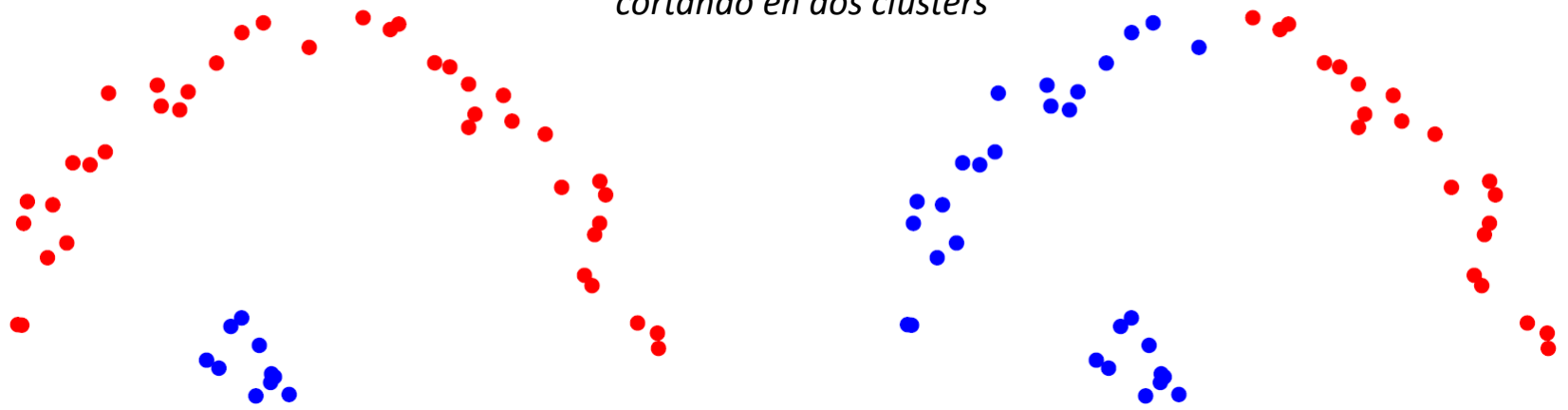


Imagen de C. Borgelt (2012). Intelligent Data Analysis

Impacto de las estrategias de clustering

- El enlace simple encadena observaciones próximas y tiende a generar cadenas
  - Funciona bien en clusters de formas diversas no necesariamente “homogéneas en todas sus direcciones” siempre y cuando los datos estén bien separados
- El enlace completo y el del centroide tienden a generar clusters más compactos
  - Funciona bien en clusters homogéneos y es más resistente al ruido en los datos

Ejemplo bidimensional cortando en dos clusters



simple

completo

Imagen de C. Borgelt (2012). Intelligent Data Analysis

Eligiendo el número adecuado de *clusters*

- Los problemas de clustering son aprendizaje no supervisado, por lo que **no hay una solución correcta** (una agrupación o un número de clusters correcto)
  - El objetivo es descubrir estructura en los datos y a priori todas son igual de correctas
  - El analista debe **interpretar** esa estructura inferida y ver qué le dice sobre el conjunto de datos que está analizando
    - Una solución es buena si los clusters son interpretables
- Hay varias aproximaciones para determinar el número de *clusters* haciendo un corte en el dendrograma
  - Usar una aproximación visual
  - Usar conocimiento experto y buscar *clusters* que sean interpretables
  - Especificar una distancia máxima a partir de la cual no formar *clusters*
  - Analizar la secuencia de distancias de los diferentes enlaces que se van haciendo al formarse la jerarquía y elegir un punto en el que la distancia se incremente mucho con respecto al enlace anterior
    - Existen heurísticas basadas en este concepto
- En las dos primeras aproximaciones podríamos utilizar diferentes alturas de cortes para diferentes *subclusters*

Algoritmos *de clustering* basados en particiones

- El objetivo es dividir las n observaciones o individuos en un número de clusters k
  - Siendo el número $\boldsymbol{k}$ **un valor de entrada del algoritmo**
- Lo que hacen estos algoritmos es dividir nuestro espacio de representación m-dimensional en k regiones, siendo m las variables consideradas
  - Normalmente es una aproximación computacionalmente menos costosa que la jerárquica
- Las particiones se realizan optimizando un criterio ya sea globalmente o localmente (es decir, en un subconjunto de individuos)
- Existen muchos algoritmos de este tipo de los que el *k*-medias (*k-means* en inglés) es el más famoso

Algoritmo de k-medias

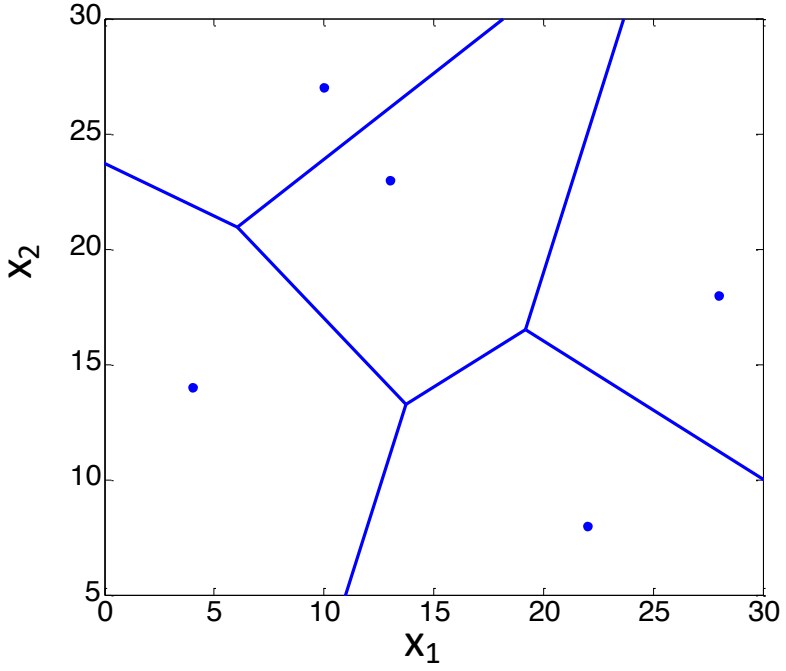
- Una vez fijado el valor de k, se realizan los siguientes pasos:
  1. Inicializa los k centros (o centroídes) de los *clusters* de forma aleatoria, típicamente
     - Generando k puntos aleatorios en el espacio m dimensional, o
     - Seleccionando aleatoriamente k individuos
  2. Repite el siguiente proceso hasta que los centros no cambien
     1. Fase de asignación: Asigna a cada punto la pertenencia al *cluster* que esté más cercano
        - Requiere el uso de una distancia (normalmente se usa la **distancia euclídea**)
     2. Actualiza el centro de los *clusters*, también llamado prototipo
        - Se calcula como el **individuo medio** de todos los individuos que pertenecen a ese *cluster*, es decir, se calcula la media de todas las variables consideradas de los individuos de cada *cluster*
        - El individuo medio es, además, el punto que minimiza la distancia euclídea entre sí mismo y los demás individuos del *cluster*

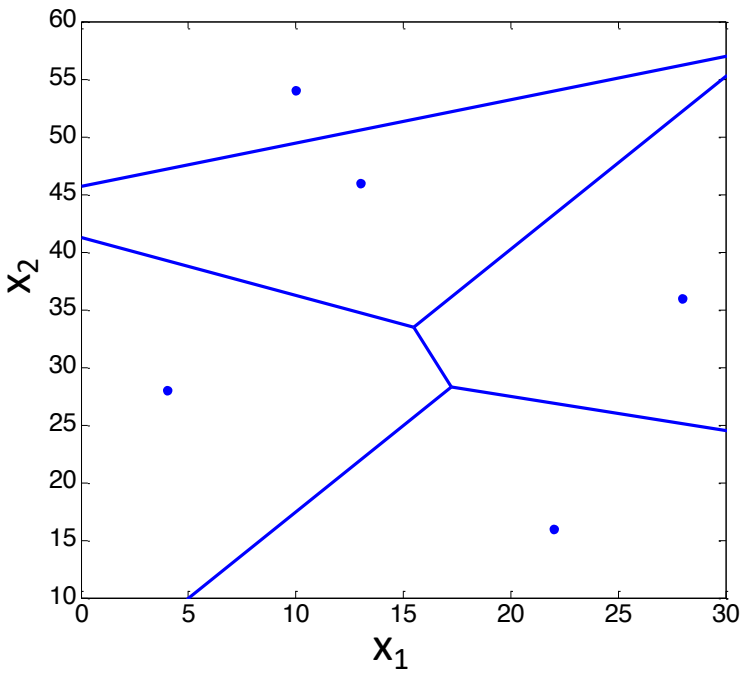
Algoritmo de k-medias

- El algoritmo de k-medias converge porque llega a un punto en el que los centros no cambian más
  - Y es bastante rápido (salvo en casos excepcionales)
- Sin embargo, la solución para un mismo k puede ser dependiente de la inicialización de los centros
  - Se puede ejecutar varias veces y ver si hay configuraciones más frecuentes o que nos convenzan más

Las regiones de Voronoi

- El algoritmo de k medias genera k particiones del espacio m-dimensional en el que se representan los individuos
- Estas regiones vienen determinadas por los k centros (o prototipos)
  - Cualquier punto dentro de una región determinada pertenece al *cluster* generado por dicho centro
- Por ejemplo, en los siguientes espacios bidimensionales con 5 prototipos se obtienen las siguientes regiones
  - En derecha, hemos hecho un cambio de escala en X_2 (multiplicando por 2)





¡La forma de las regiones cambia! Las unidades de medida de las variables (es decir, su escala) afecta a la forma de la región

El algoritmo k-medias en acción

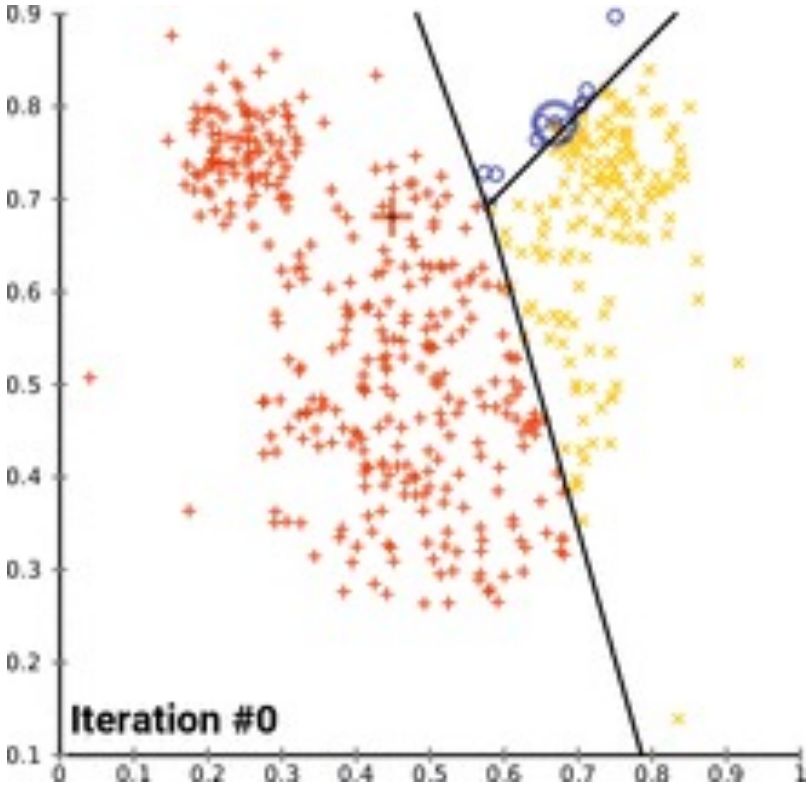
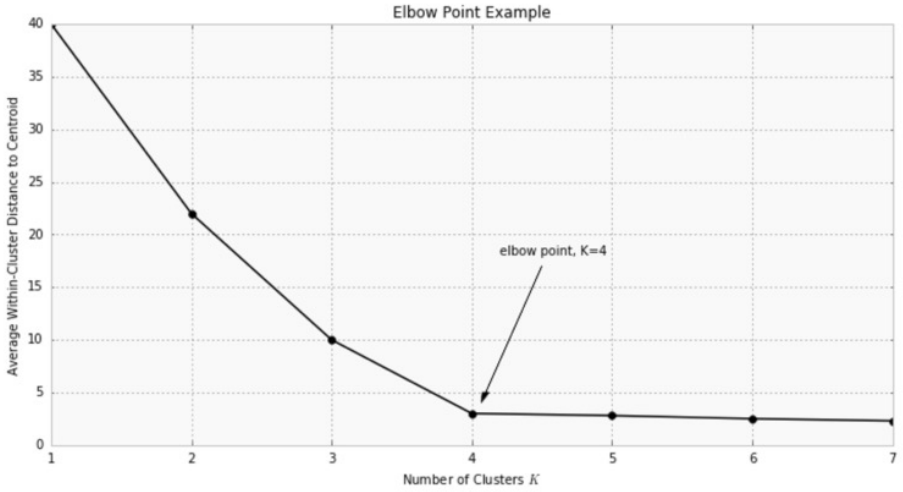


Imagen de Wikimedia Commons

Eligiendo el número adecuado de *clusters*

- De nuevo, como estamos en aprendizaje no supervisado, no hay una solución correcta
  - El objetivo es descubrir estructura en los datos
- Hay varias aproximaciones para ayudarnos en nuestra tarea
  - Usar conocimiento experto y buscar *clusters* que sean interpretables
    - Analizando los prototipos de los *clusters* y los individuos que pertenecen a ellos, así como lo dispersos o compactos que son (usando, p.ej. la desviación típica)
  - Existen índices que miden lo "compacto" de una representación
    - **Diagrama del codo**: distancia media de los puntos de cada cluster a su centroide en función del número de clusters



Eligiendo el número adecuado de *clusters*

- **Dunn**: cociente entre la distancia mínima inter-*cluster* (entre los centros) y la distancia máxima intra-*cluster* (entre dos individuos de un *cluster*)
  - Cuanto mayor es el valor para una partición, mejor
- **Davies-Bouldin** es un cociente entre la dispersión de los *clusters* y la separación de los mismos.
  - Cuanto más pequeño el valor obtenido para una partición, mejor

– Hay otras aproximaciones más sofisticadas

- Por ejemplo: usar validación cruzada (la veremos más adelante) para comprobar si la partición obtenida en los diferentes conjuntos de entrenamiento es homogénea

APRENDIZAJE SUPERVISADO

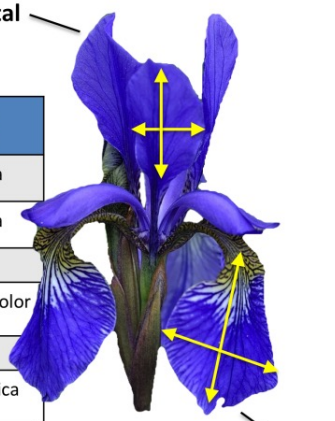
Aprendizaje supervisado

- En el aprendizaje supervisado seguimos contando con nuestra matriz de datos de n individuos, pero distinguiremos entre dos tipos de variables
  - Un conjunto de m predictores o variables de entrada, x_i con $i = 1, \dots, m$
  - Variable de respuesta o salida, normalmente se denota como y
- La variable de salida o respuesta, determina el tipo de problema que estamos tratando
  - Si se trata de una variable numérica, es un **problema de regresión**
    - Predecir el precio de una casa en función de sus características
  - Si se trata de una variable categórica, es un **problema de clasificación**
    - Predecir el tipo de flor iris dadas sus características (versicolor, setosa, virginica)
- El objetivo es ajustar un modelo que relacione las variables de entrada con la de salida, de forma que
  - Seamos capaces de predecir la respuesta lo mejor posible
  - Entendamos mejor la relación entre las variables de entrada y la de salida
    - Este último punto, no lo proporcionan algunas técnicas o algoritmos que funcionan como una “caja negra”

Aprendizaje supervisado

Samples
(instances, observations)

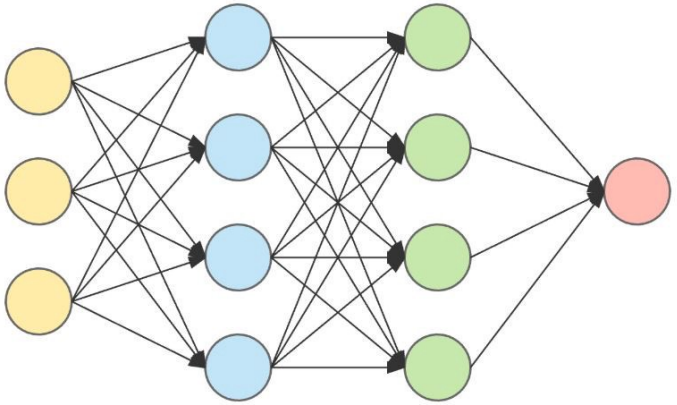
	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica



Features
(attributes, measurements, dimensions)

Class labels
(targets)

Entrenar modelo



Predecir

Sepal length	Sepal width	Petal length	Petal width	Class label
4.9	3.6	1.4	0.3	?
6.1	2.8	1.3	0.9	?
...	...	...	...	...

Midiendo el error en problemas de regresión

- Debemos comparar los resultados pronosticados $\hat{y}_i$ por nuestro modelo con los resultados observados en el conjunto de datos y_i con $i = 1, \dots, n$
- En problemas de predicción se suele utilizar el Error Cuadrático Medio o su raíz cuadrada (*Mean Square Error* o *Root Mean Square Error* en inglés)

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

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

Como medida de error a optimizar/reducir, el RMSE es equivalente al MSE, pero el RMSE es más inteligible ya que está en las mismas unidades de medida que la variable de salida

  - Los errores están al cuadrado, luego los errores grandes pesarán mucho!
  - Al buscar un modelo que minimice el MSE o el RMSE los parámetros de dicho modelo se ajustarán para evitar cometer errores "grandes"
- Otra alternativa sería el Error Absoluto Medio, que al tomar valores absolutos da igual peso a errores grandes y pequeños y está también en la misma unidad de medida que la variable de salida

Midiendo el error en problemas de clasificación

- Debemos comparar los resultados pronosticados $\hat{y}_i$ por nuestro modelo con los resultados observados en el conjunto de datos y_i con $i = 1, \dots, n$
  - Según la variable de salida sea numérica o categórica, usaremos diferentes tipos de medida de error
- En problemas de clasificación podemos visualizar el error mediante la **matriz de confusión**, que para el caso de una clase binaria sería

		Clase observada	
		1	0
Clase pronosticada	1	*Verdaderos Positivos*	*Falsos Positivos*
	0	*Falsos Negativos*	*Verdaderos Negativos*

- La **tasa de aciertos o exactitud** (*accuracy* en inglés) es, en principio, una buena medida

$$Exactitud = \frac{VP + VN}{VP + VN + FP + FN}$$

Otras medidas de rendimiento para problemas de clasificación

- La tasa de aciertos puede presentar problemas en casos en los que las clases estén desequilibradas
  - Si quiero detectar una enfermedad congénita que afecta al 1% de la población, un clasificador naïve que siempre dé negativo como resultado acertará el 99% de las veces!
- Necesitamos medidas que reflejen otros aspectos de la clasificación (**por clase**)
  - $Tasa\ de\ Verdaderos\ Positivos(TVP) = \frac{VP}{VP+FN} = \frac{VP}{P}$
    - También llamada sensitividad (*sensitivity*) y exhaustividad (*recall*)
  - $Tasa\ de\ Verdaderos\ Negativos(TVN) = \frac{VN}{VN+FP} = \frac{VN}{N}$
    - También llamada especificidad (*specificity*)
  - $Valor\ Predictivo\ Positivo(VPP) = \frac{VP}{VP+FP}$
    - Más conocida como precisión (*precision*)

		Clase observada	
		1	0
Clase pronosticada	1	*VP*	*FP*
	0	*FN*	*VN*

- Para elegir un modelo entre varios posibles necesitamos guiarnos por un único valor, por ello se suele usar alguna medida que combina otras medidas simples

$$medida\ F1 = \underbrace{2 \cdot \frac{VPP \cdot TVP}{VPP + TVP}} = \frac{2 \cdot VP}{2 \cdot VP + FP + FN}$$

Media armónica de precision y exhaustividad (F1 Score)

Entendiendo el rendimiento de un clasificador

- Podemos obtener modelos (i.e. clasificadores) de muy distinto comportamiento, las medidas de rendimiento nos dicen cómo funcionan
  - La utilidad del clasificador depende además del problema donde se esté aplicando

- Por ejemplo, consideremos un problema donde la clase es binaria (enfermo, no-enfermo) y tres clasificadores con estas medidas:

1. Precisión(enfermo)=0.99 Exhaustividad(enfermo)=0.50
2. Precisión(enfermo)=0.50 Exhaustividad(enfermo)=1
3. Precisión(enfermo)=0.80 Exhaustividad(enfermo)=0.95

  - ¿Cuál es más útil para detectar una enfermedad leve que se cura con un medicamento? ¿y si es una enfermedad letal y contagiosa? ¿y si en lugar de “enfermo/no-enfermo” la clasificación es “buen-pagador/moroso” para concederle un préstamo?

- En la fase de aprendizaje, podemos indicar qué medida queremos que “optimice” el clasificador en su aprendizaje

El proceso generador de datos

- Cuando analizamos problemas reales, estos tienen mucha riqueza y variabilidad
  - Además, los datos no estarán exentos de errores de medida, de problemas de incompletitud (podemos no contar con variables relevantes o con valores de las variables para algunos individuos), de situaciones anómalas, de excepciones…
- La variable de salida que observamos no habrá sido generada mediante un modelo matemático (ni determinista, ni estocástico) en base a unas variables de entrada
  - Es decir, en situaciones reales **no existirá un proceso generador de los datos** que usar para validar nuestro modelo
  - Pero no debemos desanimarnos, como dijo el estadístico George Box "*todos los modelos son incorrectos, pero algunos son útiles*" ☺
- Para medir la bondad de nuestra solución solamente contamos con nuestro conjunto de datos y debemos usarlo bien
  - Es posible que nuestro modelo aprenda nuestros datos perfectamente (minimizando el error a cero o casi)
  - En ese caso, seguramente nuestro modelo no sea muy útil porque haya aprendido "de más", haya aprendido "ruido"
  - Existen mecanismos para evitar este problema llamado **sobreaprendizaje** (*overfitting* en inglés)

Validación de nuestro modelo

- La validación es una forma de estimar cómo de bueno es nuestro modelo ante datos nuevos

- Como solamente tenemos un conjunto de datos, la validación simple consiste en partirlo en dos conjuntos complementarios (**entrenamiento** y **test**):
  - Un conjunto para **entrenar** nuestro modelo
  - Un conjunto para **evaluar** nuestro modelo

Normalmente, 2/3 para entrenamiento y 1/3 para test, ó 3/4 y 1/4.
Si temenos muuuchos datos (Big Data) puede ser 95% entrenamiento y 5% test.

- Si nuestro modelo está entrenado correctamente su error en el conjunto de entrenamiento no debe ser muy diferente que en el conjunto de test
  - Esto querrá decir que en el entrenamiento no hemos sobre-aprendido y que el modelo generaliza bien cuando se usa con datos con los que no se entrenó

Validación de nuestro modelo

- Si queremos elegir el mejor modelo entre varios, se suele optar por partir los datos en tres conjuntos complementarios (entrenamiento, validación y test):
  - **Entrenamiento**: Para entrenar nuestro modelo
  - **Validación**: Para comparar los modelos parametrizados y elegir el mejor
  - **Test**: Para estimar cómo se comporta el modelo elegido sobre datos limpios

- Esta partición en tres se utiliza especialmente cuando estamos, por ejemplo, probando distintas parametrizaciones de un modelo o comparando distintos modelos
  - En ese caso, el tercer conjunto se hace para no "sobreaprender" ya que las dos primeras particiones las usamos para entrenar y elegir el mejor modelo

- Existen además estrategias más sofisticadas para realizar el entrenamiento y validación de un modelo

Validación cruzada en *k* partes

- La validación cruzada en *k* partes (*k-fold cross validation*) es una forma de estimar cómo de bueno es nuestro modelo ante datos nuevos
- Consiste en los siguientes pasos
  1. Los datos se dividen aleatoriamente en *k* subconjuntos iguales
  2. Se entrena el conjunto con (*k*-1) y se valida con el restante
  3. Se repite *k* veces el paso 2, cambiando el conjunto que se usa para validar
  4. Como medida de error final se suele presentar la media de las *k* medidas de error de validación (aunque puede ser interesante comprobar que las *k* medidas sean similares)

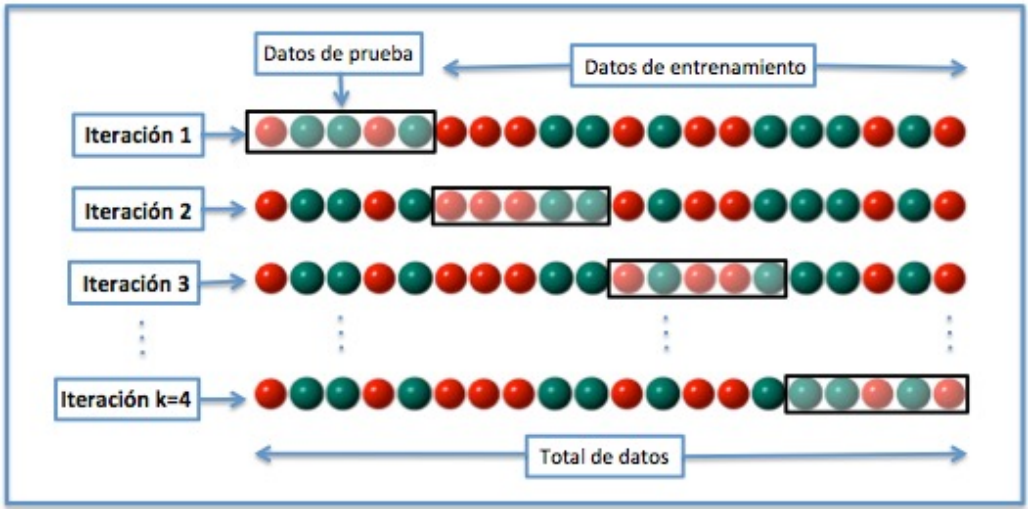


Imagen de Wikimedia Commons

Validación cruzada en *k* partes

- En la validación cruzada en *k* partes todas las observaciones son usadas una vez para validación y *k*-1 veces para entrenamiento
- A veces puede ser interesante **estratificar** las *k* partes:
  - En un problema de regresión significa que la media de la variable respuesta sea similar en todas las partes (o incluso la distribución)
  - En un problema de clasificación significa que la proporción de clases en cada parte sea similar
- Si llevamos la validación cruzada al extremo y tomamos $k = n$, siendo n el número de invididuos en nuestro conjunto de datos, estaríamos utilizando la validación dejando-uno-fuera (*Leave One Out validation* o *LOO validation*)
  - Esta validación es adecuada cuando tenemos pocos datos y no tiene sentido dividirlo en partes mayores

APRENDIZAJE SUPERVISADO
K VECINOS MÁS CERCANOS (K-NN)

Los *k* Vecinos Más Cercanos

- Los *k* vecinos más cercanos (*k nearest neighbors)* o k-NN *es* una técnica de aprendizaje supervisado muy sencilla

- Pertenece al paradigma de aprendizaje perezoso o basado en instancias
  - Perezoso porque no calcula ningún modelo y demora todos los cálculos hasta el momento en que se le presenta un ejemplo nuevo
  - Basado en instancias porque usa todos los individuos disponibles y ante un ejemplo nuevo recupera los más relevantes para componer la solución

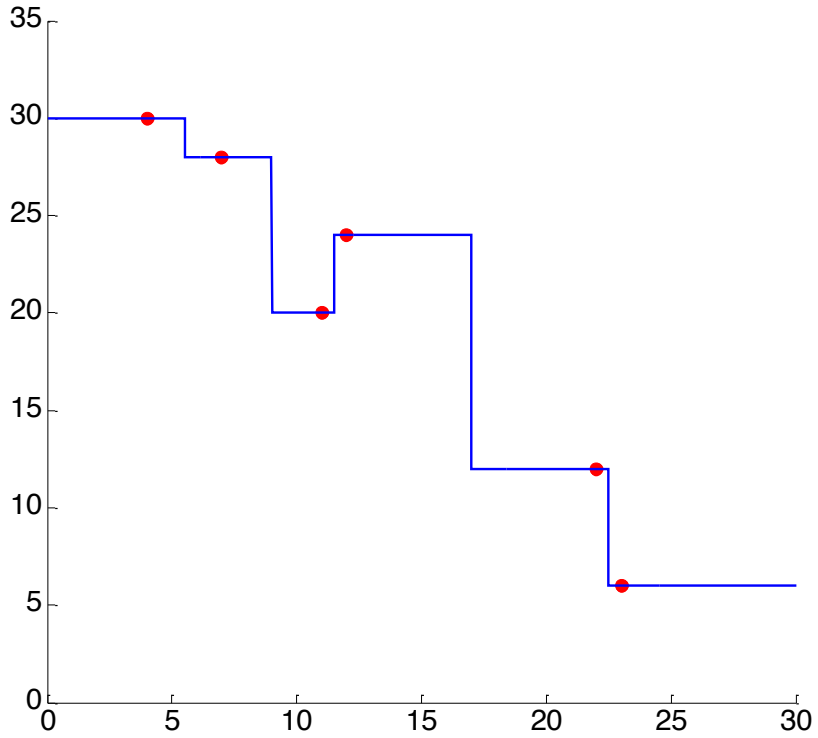
- Se trata de una técnica aplicable en problemas de **regresión** y de **clasificación**

- Dada su sencillez puede ser un buen primer método de aprendizaje automático contra el que comparar otros métodos más sofisticados

El Vecino Más Cercano

- Dado un ejemplo nuevo con variable de salida desconocida,
  1. Seleccionar de los n ejemplos disponibles, el ejemplo que más se le parezca
     - Para ello podemos usar una distancia que consideremos adecuada, p.ej. la Euclídea
  2. Ofrecer como solución del ejemplo nuevo la solución del ejemplo seleccionado

Un ejemplo en un problema sencillo de regresión



datos observados

X	11	22	4	12	23	7
Y	20	12	30	24	6	28

Los datos siguen aprox. una relación lineal y el 1-NN parece que produce un sobreajuste, porque la predicción sigue los datos excesivamente

Los *k* vecinos más cercanos (*k* –NN)

- Distancia: Euclídea
  - Se pueden usar otras distancias (L_1, Mahalanobis,…)
- Nº de ejemplos que recupera: *k*
  - Se puede determinar el valor de *k* utilizando alguna estrategia de validación
- Composición de la solución:
  - Si la variable de salida es cuantitativa:
    - **Devuelve como valor solución la media** de los valores solución de los *k* individuos recuperados, es decir, de los *k* individuos más cercanos
    - **Se puede hacer una media ponderada**, donde se dé más peso a los vecinos según su cercanía al ejemplo nuevo . Siendo u el ejemplo nuevo, el peso del vecino v_p es

      $$\psi_p = \frac{1}{d(u,v_p)+\xi}$$

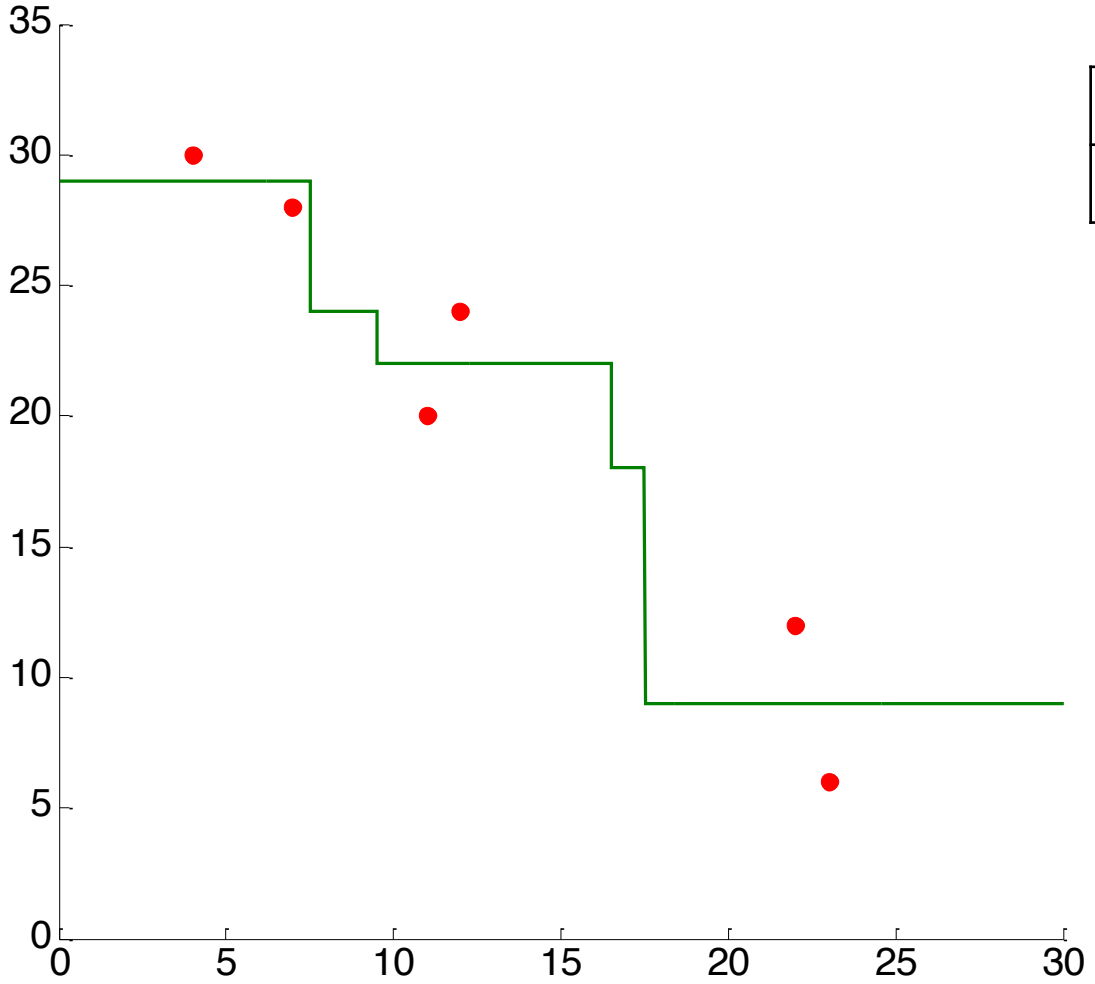
      Con ξ un valor constante muy pequeño que evite que el peso ψ_p tome un valor infinito

      Los pesos de cada vecino se normalizan para que su suma sea 1

      $$\omega_p = \frac{\psi_p}{\sum_{i=1}^{k} \psi_i}$$
  - Si la variable de salida es categórica:
    - Devuelve como valor solución el **valor más frecuente (voto mayoritario)** entre los valores solución de los k individuos recuperados
    - Se puede utilizar la estrategia de voto ponderado, como en el caso cuantitativo

K-NN en un problema de regresión

- Ejemplo con una variable de entrada y $k = 2$



datos observados

X	11	22	4	12	23	7
Y	20	12	30	24	6	28

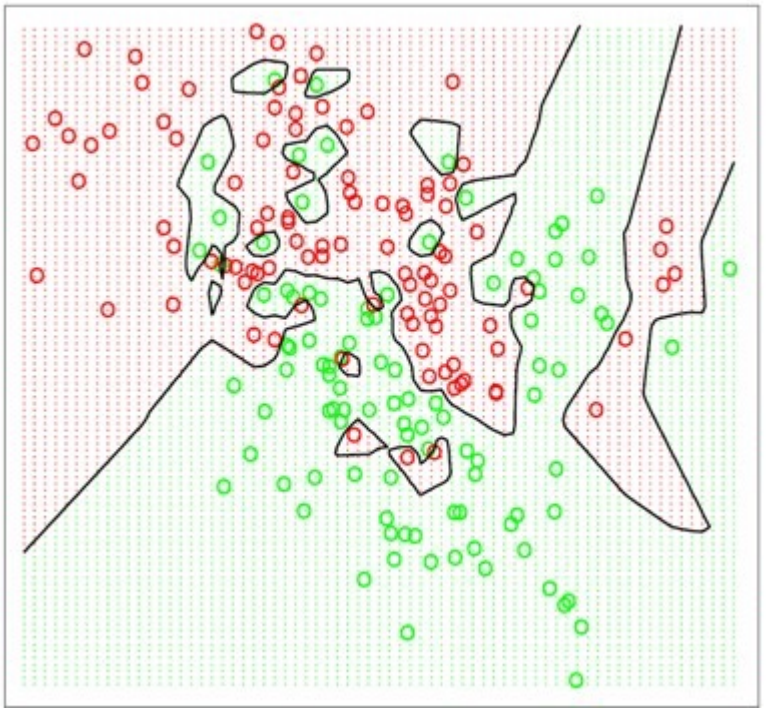
La predicción con $k = 2$ es ligeramente más suave que para $k = 1$
Disminuye el sobreajuste

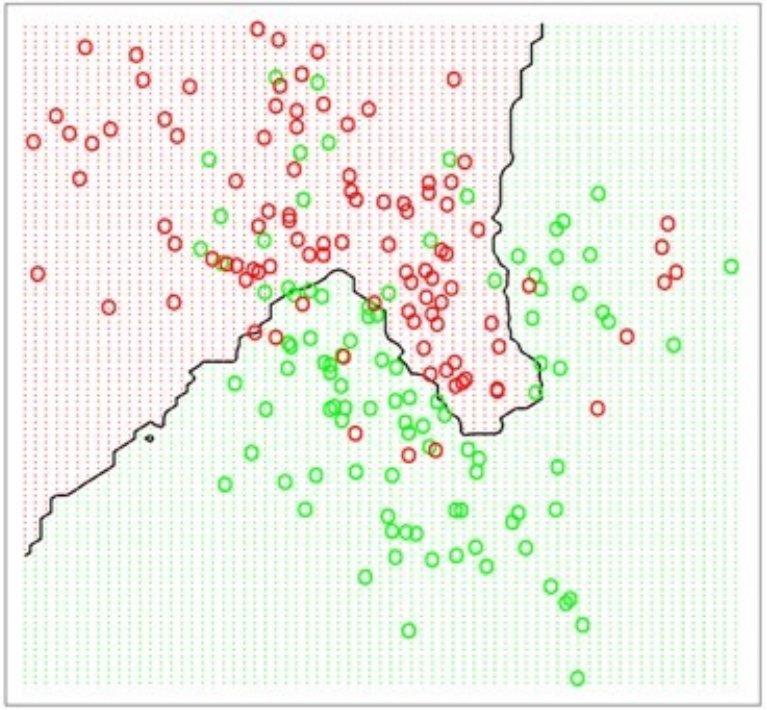
En general, a medida que aumenta k, el suavizado es más acusado (si $k = n$ se obtiene la media)

El k-NN presenta carencias inevitables:
No extrapola bien en los extremos
La predicción es escalonada

K-NN en un problema de clasificación

- Ejemplo con dos variables de entrada y $k = 1$ (izq.) Vs $k = 15$ (dcha.)





Si aumenta k disminuye el sobreajuste y aumenta el suavizado

Si $k = n$ se obtiene la clase mayoritaria

No funciona bien en regiones despobladas, siempre es difícil predecir en una zona donde no hay datos.

Imágenes de Hastie et al. (2001). Elements of Statistical Learning

Determinando la *k* en el *k*-NN

- Aunque el k-NN no estime ningún modelo, debemos determinar el valor de k
  - Si es muy pequeño, corre el riesgo de sobreaprender (aprender conocimiento espúreo)
  - Si es muy grande, corre el riesgo de generalizar demasiado
- Podemos usar una estrategia de validación cruzada para encontrar el valor de k
  - Determinamos un rango de valores de k, por ejemplo $k \in [1,10]$
  - Determinamos una estrategia de validación, p.ej. validación cruzada en k' partes (OJO, ponemos k' porque no es la k del k-NN)
  - Elegimos la k que minimiza el error de validación

El *k*-NN y las variables de entrada

- El k-NN funciona bien en problemas de pocas dimensiones (variables) con datos abundantes.
  - En estos problemas, puede encontrar puntos cercanos relevantes
- El k-NN sufre la maldición de la dimensionalidad,
  - Al aumentar los dimensiones los vecinos no suelen estar tan cerca
- Además, su rendimiento empeora si hay variables irrelevantes
  - P.ej. Un punto puede ser muy parecido a otro en cuanto a las variables más relevantes, pero si consideramos variables irrelevantes puede ser que no se le considere como uno de los vecinos más cercanos
- Por tanto, se deben evitar incluir variables irrelevantes o redundantes
  - Si esto no se sabe a priori, **se puede usar validación cruzada** para seleccionar las variables de entrada que se considerarán de entre todas las posibles
  - Se elegirá aquel subconjunto de variables que minimice el error de validación

Tiempo de cálculo de los *k* Vecinos Más Cercanos

- La búsqueda de los vecinos más cercanos, como mide la distancia a todos los ejemplos, es lógicamente de complejidad $O(\mathrm{n})$, donde n es el número de ejemplos
  - Esto hace que sea un mal método para trabajar con conjuntos de datos muy grandes en tiempo real
- Pero existen técnicas para aliviar este tiempo de búsqueda
  - En primer lugar, se puede **paralelizar** el proceso de forma muy sencilla
    - Dividir los n ejemplos en varias partes y recuperar los k vecinos de cada una
    - Agregar todos los vecinos "parciales" y recuperar los k vecinos "globales"
  - Estrategias basadas en **árboles k-d**
    - Subdividen el espacio de entrada en regiones y reducen la complejidad a $O(\log(n))$ en un espacio con puntos aleatoriamente distribuidos
  - Estrategias aproximadas basadas en **tablas hash**
    - No hacen la búsqueda exacta sino una búsqueda aproximada sobre un conjunto de tablas hash que representan proyecciones de los datos en espacios unidimensionales
      - La consulta de una tabla hash es de complejidad $O(1)$

APRENDIZAJE SUPERVISADO
ÁRBOLES DE DECISIÓN

Árboles de decisión

- Los árboles de decisión son una técnica de aprendizaje supervisado
- Pueden usarse en problemas de clasificación o de regresión
  - Aquí los veremos únicamente en clasificación
- Permite el aprendizaje de conceptos de forma inductiva
  - A partir de un conjunto de ejemplos de entrenamiento,
  - Construye un **árbol de decisión** donde cada nodo pregunta por el valor de una variable
  - El árbol construido sirve para clasificar ejemplos nuevos (sin etiquetar o del conjunto de validación)
    - Colocando el ejemplo nuevo en la raíz y respondiendo las preguntas
    - Con ello desciende por las ramas hasta llegar a un nodo hoja
- El árbol se construye recursivamente
  - En cada nodo se hace una pregunta que busca disminuir la entropía de los nodos hijos
- El árbol aprendido es muy **interpretable** y puede ser analizado por el experto para entender la clasificación o definir conceptos

Un problema de clasificación

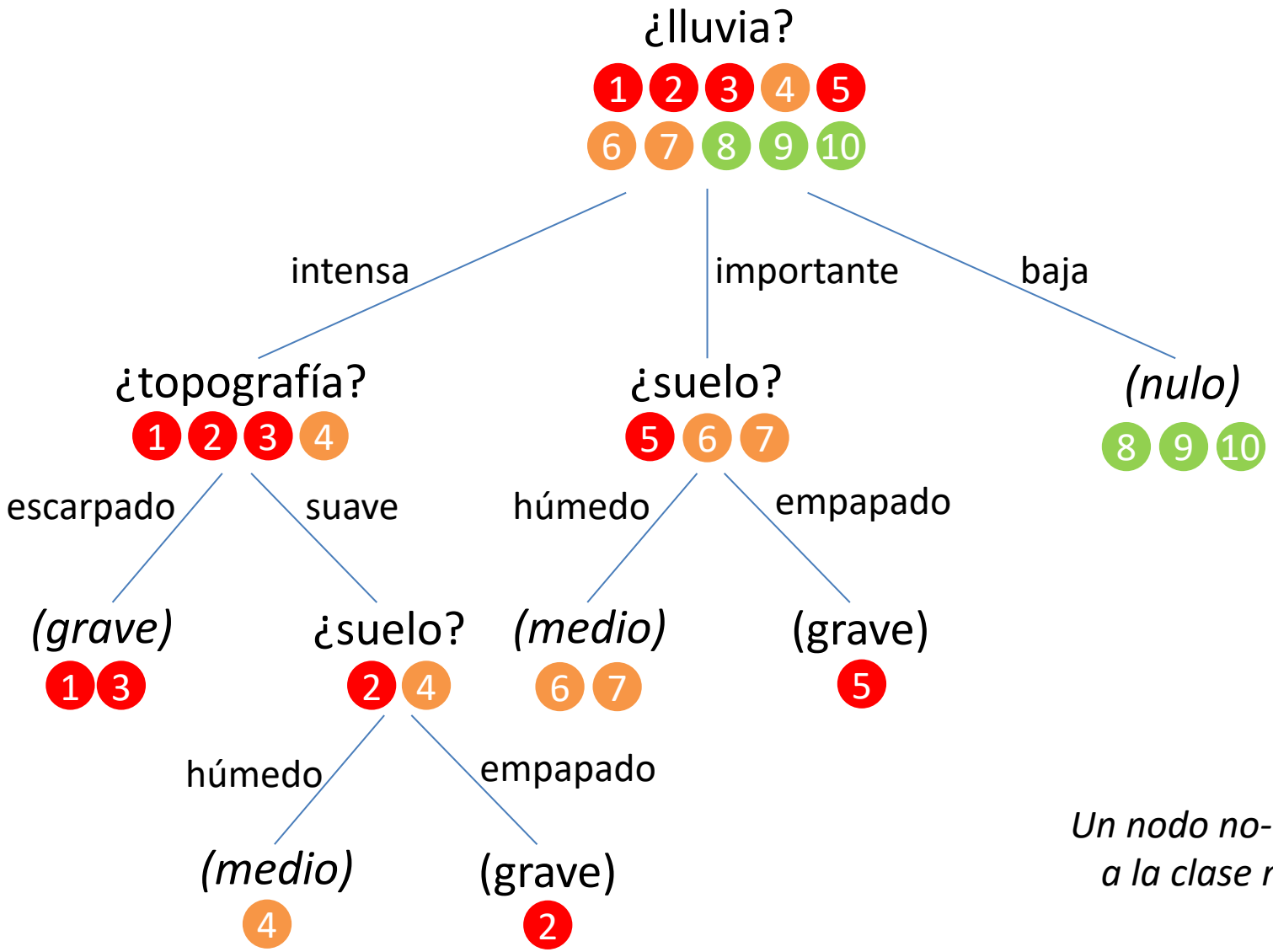
- Aprender a clasificar situaciones de riesgo de inundación
  - En función de las variables *lluvia, suelo y topografía*
- Ejemplos de entrenamiento:

Caso	Lluvia	Suelo	Topografía	Riesgo
1	intensa	empapado	escarpada	grave
2	intensa	empapado	suave	grave
3	intensa	húmedo	escarpada	grave
4	intensa	húmedo	suave	medio
5	importante	empapado	escarpada	grave
6	importante	húmedo	escarpada	medio
7	importante	húmedo	suave	medio
8	baja	empapado	escarpada	nulo
9	baja	húmedo	escarpada	nulo
10	baja	húmedo	suave	nulo

Representación arborescente

- Representación natural de criterios de decisión de las personas
  - Nodos: preguntas (atributos o características)
  - Arcos: posibles respuestas (valores posibles en los ejemplos)
  - Hojas están etiquetadas con la predicción (clase de ese ejemplo)

Árbol de decisión de nuestro problema



Un nodo no-hoja clasifica a la clase mayoritaria

Construcción de árboles de decisión

- Se trata de un **proceso inductivo de la raíz a las hojas** (*top-down*)

- En cada nodo se da una **selección voraz del mejor atributo**
  - Calcular un criterio de optimalidad para todos los atributos
  - Seleccionar el atributo que mejor valor de optimalidad tiene
  - Generar los nodos hijos en función del atributo seleccionado

- **Divide y vencerás** aplicado recursivamente
  - Se dividen los ejemplos de cada nodo en función del atributo elegido
  - El proceso se aplica recursivamente en los nodos hijos
  - El proceso termina si:
    - Todos los elementos pertenecen a la misma clase
    - No hay más atributos que elegir

- El algoritmo más famoso que construye árboles de decisión es el ID3 (Quinlan, 86)
  - Usa teoría de la información para estimar el mejor candidato
  - Se consigue complejidad lineal

Algoritmo ID3

- El árbol se construye de arriba a abajo, trabajando por niveles y dentro de cada nivel trabaja por nodos
- En cada nodo se pretende:
  - Obtener el atributo en base al cual obtener los nodos hijos
  - Se selecciona el que mejor discrimine entre el conjunto de ejemplos
    - El atributo más discriminante será aquél que conduzca a un estado con menor entropía o menor desorden (mayor información)
  - Heurística para obtener árboles pequeños (en profundidad)

- La entropía (Shannon, 1948) mide la ausencia de homogeneidad de un conjunto de ejemplos con respecto a su clase
  - Es una medida estándar del desorden (*0 es homogeneidad total)*
    - *Utilizada en física y en teoría de la información*
- **Ganancia de información** es la diferencia entre la entropía del conjunto original y la de los subconjuntos obtenidos
  - También se emplea el término **disminución de la entropía**

Entropía

- En un problema de clasificación donde la variable de salida y tiene s_i posibles clases con $i = 1, \dots, p$, la entropía de un nodo N se define como:

$$E(N) = -\sum_{i=1}^{p} P(s_i) \log_2 P(s_i),$$

siendo $P(s_i)$ la probabilidad de que un ejemplo tenga la clase s_i en el nodo N, y calculándose únicamente para las clases observadas en el nodo N (para evitar el cero en el logaritmo)

- A esta entropía se le llama entropía inicial de un nodo N (antes de clasificar los ejemplos que contiene en base a alguno de los atributos)

- La entropía final de un nodo N tras usar el atributo A que tiene q valores (a_j)

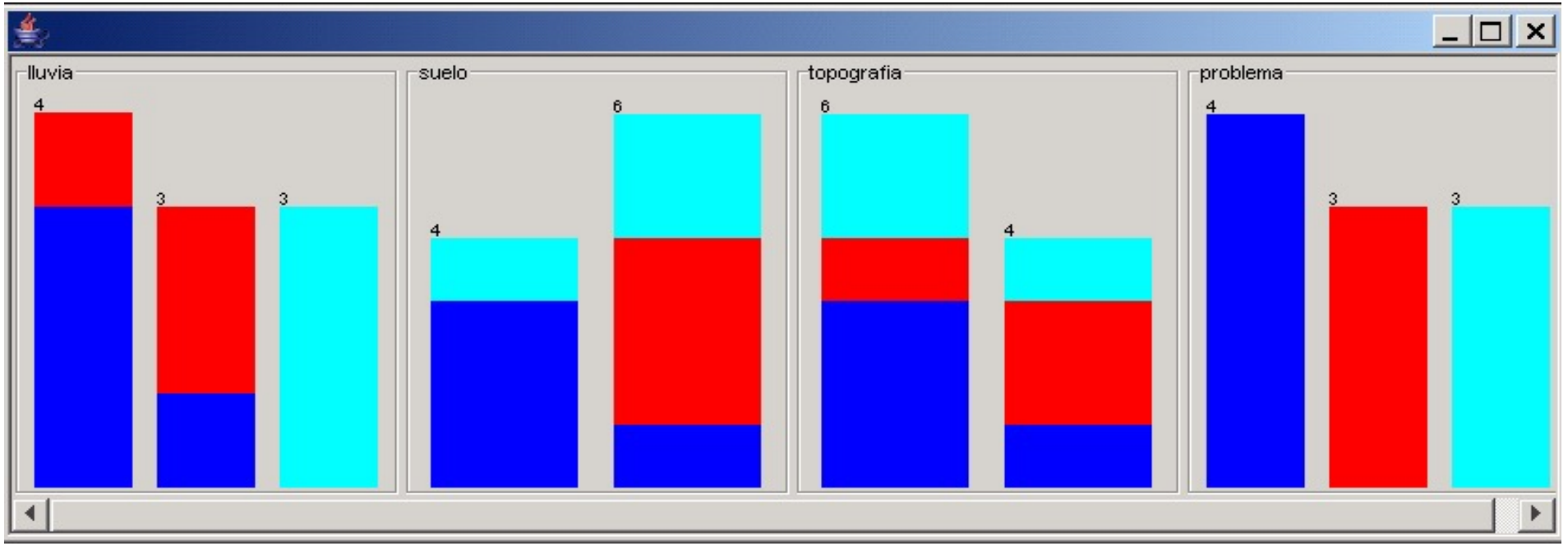
$$E(N|A) = \sum_{j=1}^{q} P(a_j) \left(-\sum_{i=1}^{p} P(s_i|a_j) \log_2 P(s_i|a_j) \right),$$

siendo $P(s_i|a_j)$ la probabilidad de que un ejemplo del nodo hijo con $A = a_j$ tenga la clase s_i

Disminución de la Entropía o Ganancia de Información

- Para cada atributo A, B, C ... se calcula la disminución de entropía (DE) causada por su utilización
  - $DE(N, A) = E(N) - E(N|A)$
  - $DE(N, B) = E(N) - E(N|B)$
  - $DE(N, C) = E(N) - E(N|C)$...
- En cada nodo, se selecciona aquel atributo que mayor **disminución de entropía** genera o **ganancia de información**
- Aplicado al ejemplo
  - Hay 3 clases y 3 atributos

ID3: ejemplo



intensa

importante

baja

empapado

húmedo

escarpada

suave

Grave

Medio

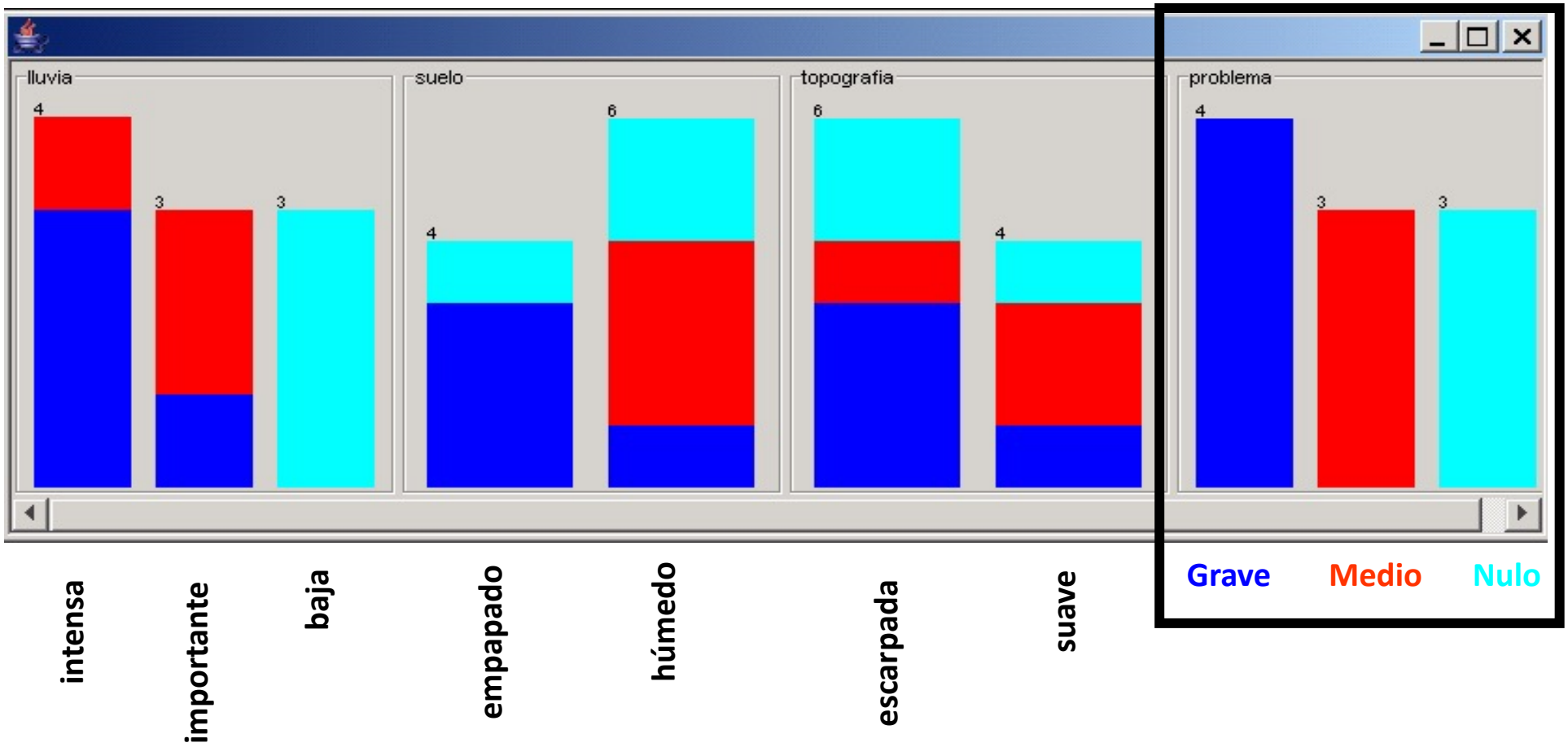
Nulo

ID3: ejemplo

- Entropía inicial en la raíz del árbol: (del problema global)

P(grave) = 0,4 P(medio) = 0,3 P(nulo) = 0,3

$E(raíz) = -0,4 \log_2 0,4 - 0,3 \log_2 0,3 - 0,3 \log_2 0,3 = 1,571$



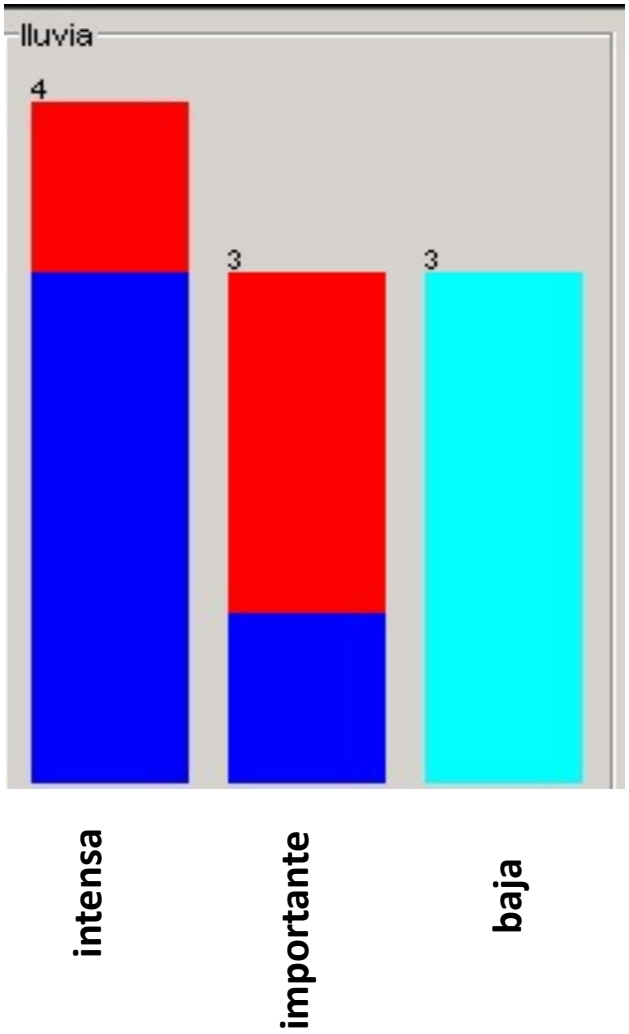
ID3: ejemplo

- Entropía final clasificando según lluvia (A):

  a_1: lluvia intensa,

  a_2: lluvia importante,

  a_3: lluvia baja



$E(raíz|a_1)= -0,75 \log_2 0,75 – 0,25 \log_2 0,25 = 0,811$

$E(raíz|a_2)= 0,918$

$E(raíz|a_3)= 0$

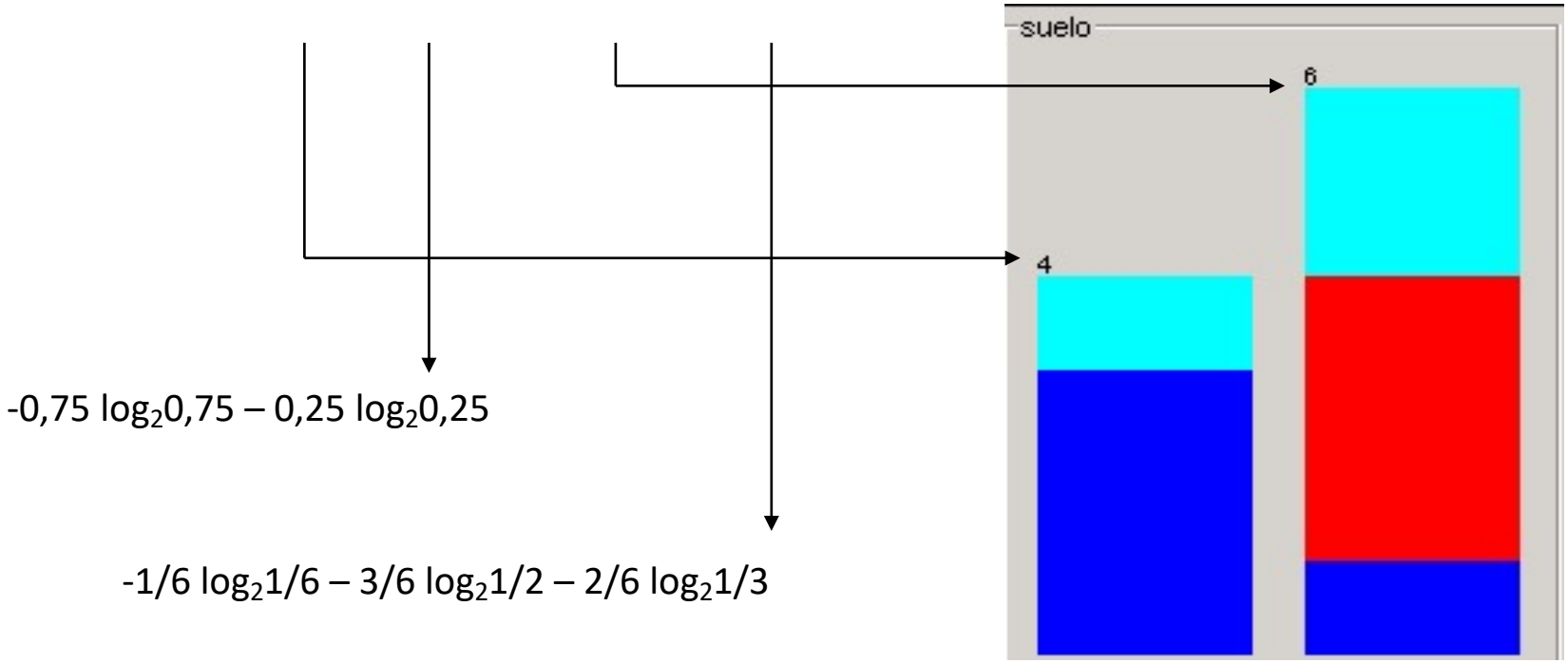
$E(raíz|A)= 0,4*0,811 + 0,3*0,918= 0,6$

$DE(raíz,A) = 1,571 – 0,60 = 0,971$

ID3: ejemplo

- Entropía final clasificando según suelo (B):

$$E(raíz|B) = 0{,}4*0{,}811 + 0{,}6*1{,}459 = 1{,}20$$



$$DE(raíz,B) = 1{,}571 – 1{,}20 = 0{,}371$$

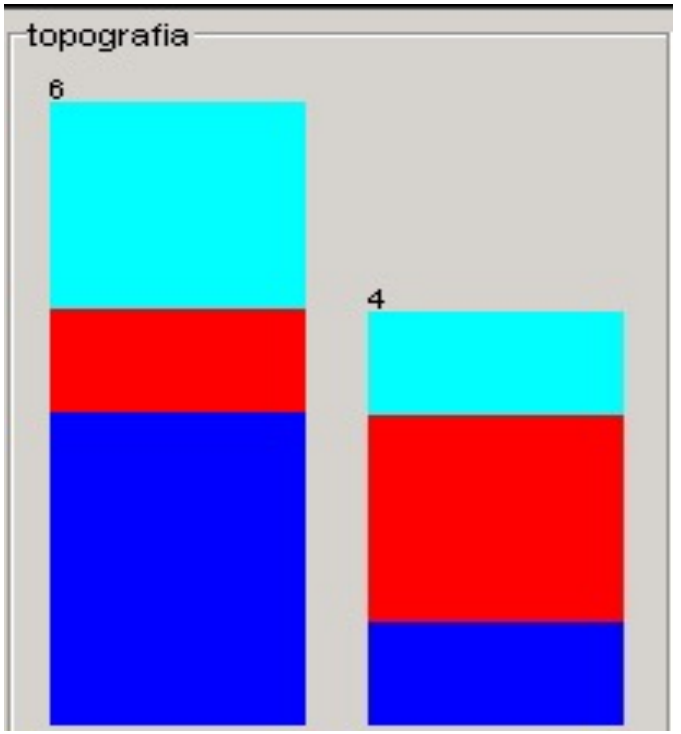
ID3: ejemplo

- Entropía final clasificando según topografía (C):

$-1/4 \log_2 1/4 - 2/4 \log_2 2/4 - 1/4 \log_2 1/4$

$E(raíz|C) = 0{,}6*1{,}459 + 0{,}4*1{,}50 = 1{,}475$

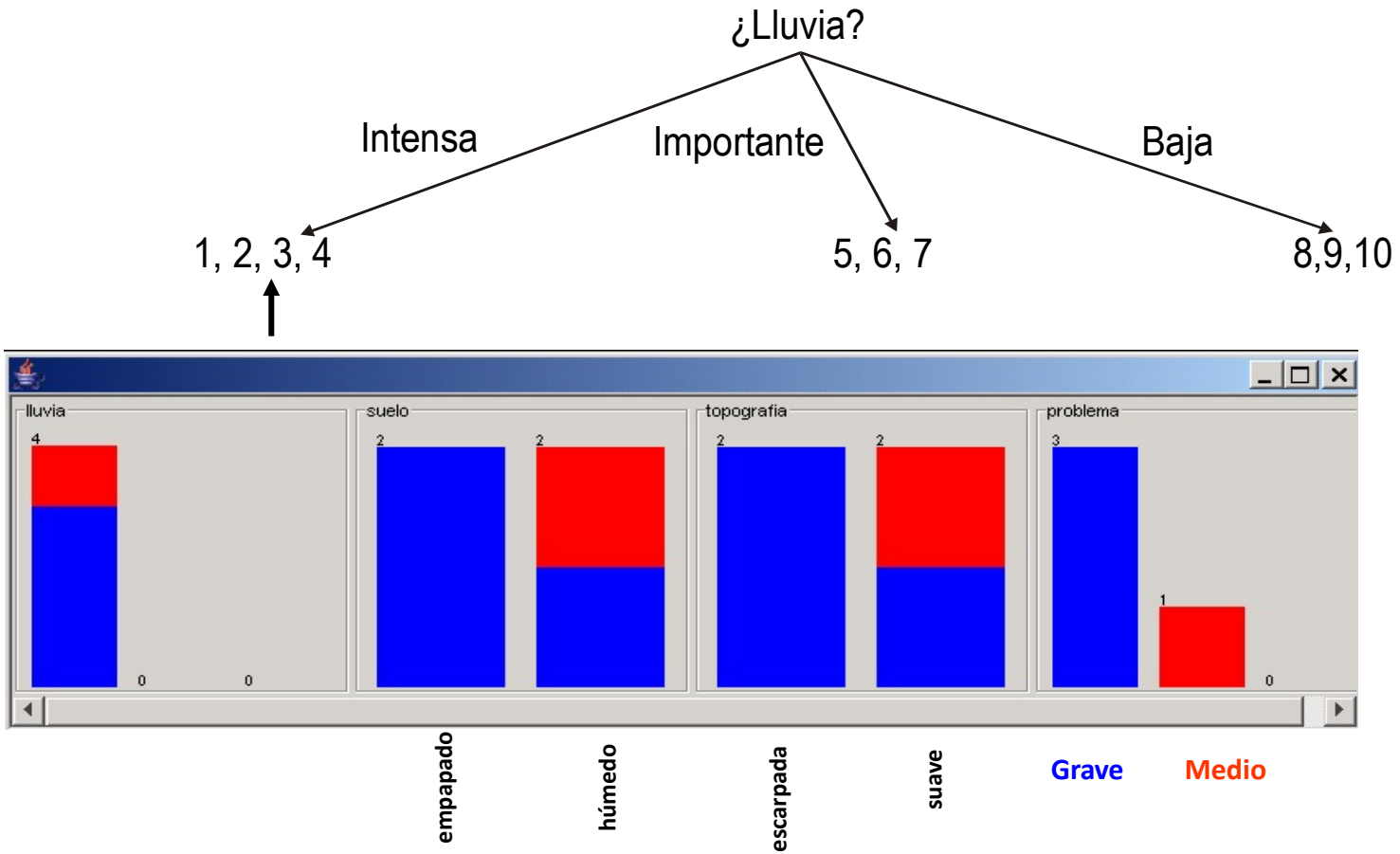
$-1/6 \log_2 1/6 - 3/6 \log_2 1/2 - 2/6 \log_2 1/3$



$DE(raíz,C) = 1{,}571 - 1{,}475 = 0{,}096$

La mayor disminución de entropía se consigue con el atributo A y por ello éste es el seleccionado para el primer nivel del árbol

ID3: ejemplo



- En la siguiente iteración se vuelve a aplicar el algoritmo sobre cada uno de los tres nuevos nodos, considerando en cada uno el subconjunto de ejemplos obtenido y habiendo eliminado el atributo lluvia del conjunto de atributos

Interpretando el árbol del ID3

- Selección “automática” de variables relevantes
  - Como en cada nodo se elige el atributo que más “información” aporta a la clasificación, el ID3 por definición está eligiendo las variables más relevantes para solucionar el problema
  - En ese sentido, no es grave que haya variables irrelevantes entre las variables de entrada → el algoritmo no las elegirá (en principio)
  - Para un nodo determinado, hay variables que pueden “medir” el mismo aspecto y/o tener similar poder discriminante, la selección de una u otra dependerá de su valor de disminución de la entropía
    - Sin embargo, la elección puede condicionar la estructura del árbol a partir de ese nodo
- Legibilidad de la solución
  - La estructura arborescente es **inteligible** por un experto
  - El experto puede aprender sobre el problema que tiene entre manos leyendo el árbol, por ejemplo:
    - Qué variables tienen mayor poder discriminante
    - Qué valores de variables están asociados con una clase u otra
    - Etc

Finalización del ID3

- Terminación:
  - La expansión de un nodo se detiene cuando todos sus ejemplos pertenecen a la misma clase (≡ entropía nula)
  - El proceso se detiene cuando no se puede seguir expandiendo ningún nodo porque no quedan atributos (variables de entrada)
  - A las hojas se les asigna la clase mayoritaria a la que pertenecen sus ejemplos

- Riesgo de sobreaprendizaje
  - Si extendemos un árbol hasta el final, estamos obteniendo una representación que aprende lo mejor posible los datos de partida
    - Es posible que algunas ramificaciones aprendan errores o aspectos de los datos de entrenamiento no generalizables a otras muestras
  - Eso constituye un riesgo a evitar y para ello hay alternativas que no generan árboles completos. Por ejemplo:
    - Utilizar un umbral de disminución de la entropía por debajo del cual no ramificar
    - Usar mecanismos de poda
  - Además, siempre conviene evaluar el sobreaprendizaje de un árbol (ya sea completo o no) mediante una estrategia de validación

Poda del árbol de decisión

- La poda del árbol de decisión se realiza para
  - Simplificar el árbol y ganar en capacidad de interpretación y legibilidad
  - Evitar el sobreajuste y mejorar la capacidad de generalización
- La poda consiste principalmente en cortar las ramas malas (subárboles) haciendo que el árbol no se expanda en dicho punto
- Vamos a ver dos ejemplos:
  - Poda mediante reducción del error
  - Poda pesimista

Poda mediante reducción del error

- Requiere un conjunto de ejemplos no usados para generar el árbol

- Consta de los siguientes pasos:
  1. Clasifica dichos ejemplos y calcula el error que cometen todas las hojas
  2. El número de errores de un subárbol es la suma de los errores de todas sus hojas
  3. Calcula el número de errores que cometería el nodo que origina el subárbol si no se expandiera
  4. Si los errores sin expandir son iguales o menores que expandido, se poda el subárbol
     - Se recalcula el número de errores de los subárboles que contenían dicho subárbol

  – Ventaja: Evita el sobreajuste de forma efectiva
  – Inconveniente: Requiere de nuevos ejemplos

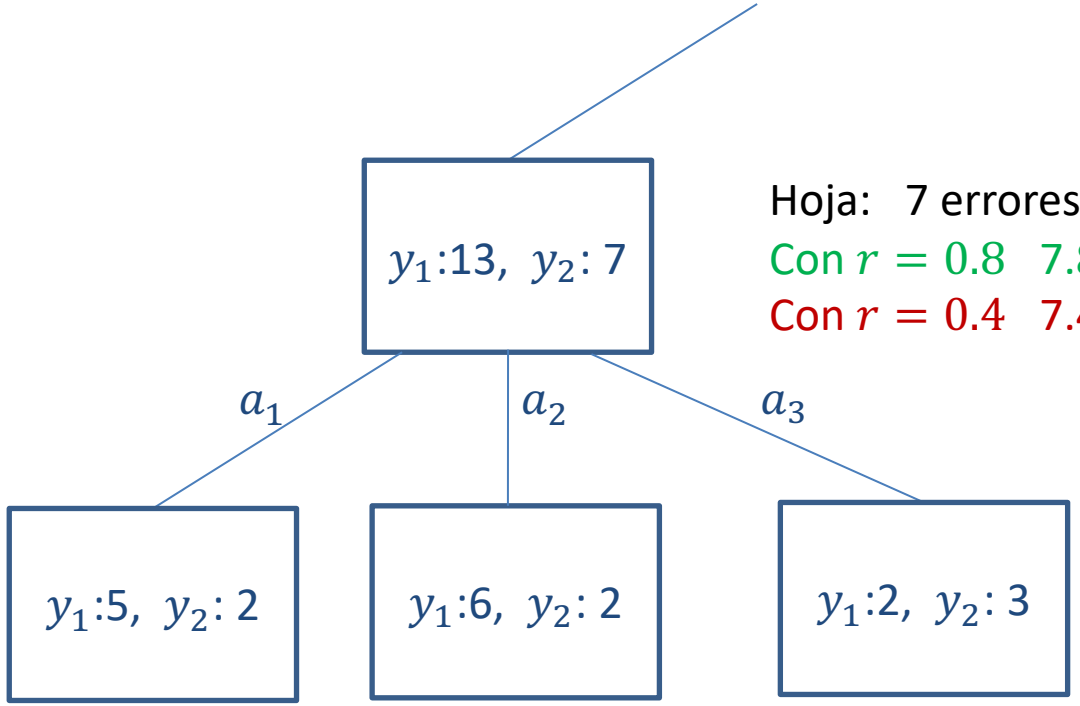
Poda pesimista

- Consta de los siguientes pasos:
  1. Clasifica un conjunto de ejemplos (usado o no para generar el árbol)
  2. Para cada nodo hoja, calcula los errores que comete y añade un valor prefijado r
  3. El número de errores de un subárbol es la suma de los errores de todas sus hojas
  4. Calcula el número de errores que cometería el nodo que origina el subárbol si no se expandiera e increméntalo también con r
  5. Si los errores sin expandir son iguales o menores que expandido, se poda el subárbol
     - Se recalcula el número de errores de los subárboles que contenían dicho subárbol

  – Ventaja: No necesita ejemplos extra
  – Inconveniente: El número de elementos en la hoja no afecta

Ejemplo de poda pesimista

Poda pesimista con $r = 0.8$ y $r = 0.4$



Hoja: 7 errores
Con $r = 0.8$ 7.8 errores
Con $r = 0.4$ 7.4 errores

TOTAL= 6 errores

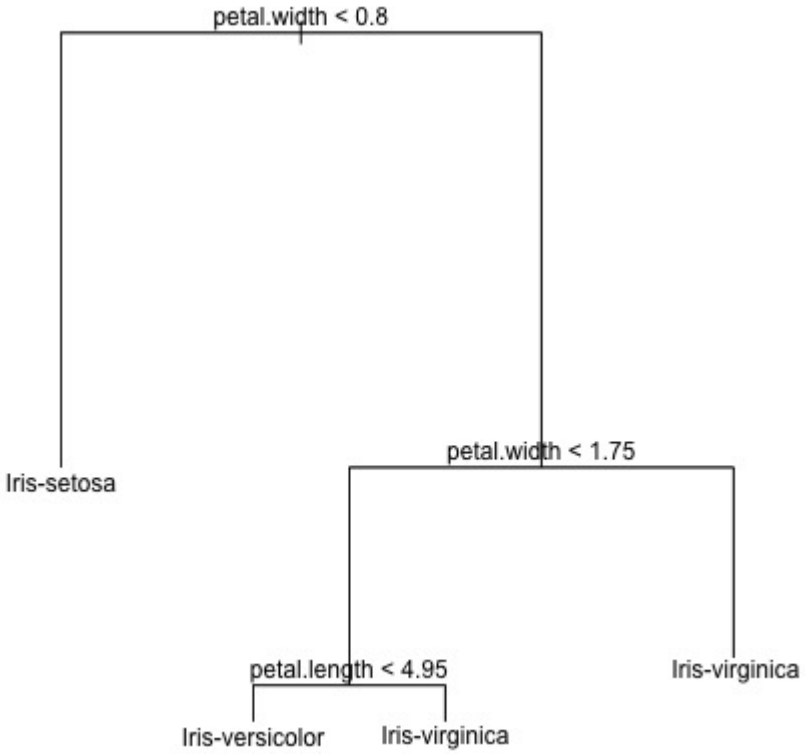
Con $r = 0.8$	2.8 errores	2.8 errores	2.8 errores	TOTAL=8.4 errores	**Podar**
Con $r = 0.4$	2.4 errores	2.4 errores	2.4 errores	TOTAL=7.2 errores	**No podar**

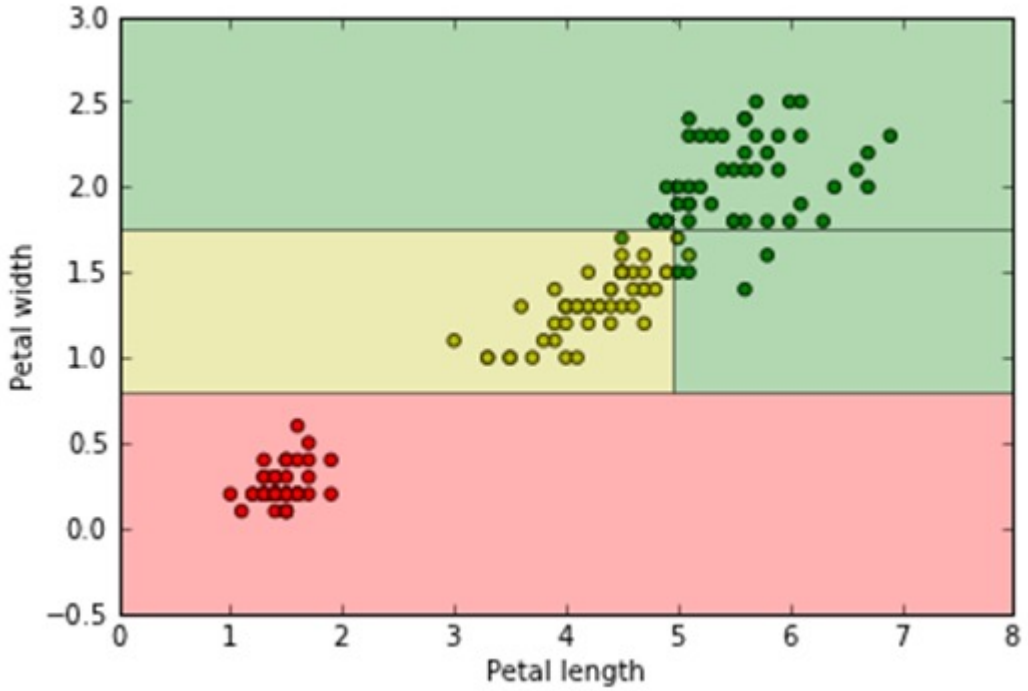
Manejo de variables de entrada continuas en ID3

- Para manejar variables de entrada numéricas se discretizan los valores de las variables en intervalos generalmente booleanos.
  - En primer lugar, ordenamos los valores de la variable cuantitativa con su respectivo valor de la clase
  - Se determinan los puntos umbrales que son los valores para los cuales la clase cambia
  - De entre los umbrales elegimos el que produce mayor disminución de la entropía.
- Ejemplo con una variable continua:
    - X_i: 40,48,60,72,80,90
    - Y: no, no, si, si, si, no
  - Puntos umbrales candidatos
    - Entre 48 y 60 (el punto medio sería 54)
      - La partición binaria candidata según el atributo X podría ser $X_i < 54$
    - Entre 80 y 90 (el punto medio sería 85)
      - La partición binaria candidata según el atributo X podría ser $X_i < 85$
- En nuestro ejemplo, elegiríamos la partición que genera la pregunta $X_i < 54$ porque genera una mayor disminución de la entropía
  - Si hubiera otras variables (numéricas o categóricas) a considerar, tendríamos que comparar la disminución de la entropía $X_i < 54$ con la disminución de la entropía que generan las particiones usando las otras variables

Ejemplo con el problema de la flor del Iris

- Si usamos los datos del Iris y dos variables ancho y largo del pétalo, podemos obtener un árbol como éste





Observa que en el nodo raíz se podría haber elegido también petal.length < 2.5
El resto del árbol no cambiaría, pero el color de las regiones donde no hay observaciones sí → En regiones sin observaciones es arriesgado "opinar"

Refinamientos del ID3

- El propio Quinlan creó una versión posterior del ID3 que es el algoritmo C4.5 (Quinlan 93)
  - Soluciona un pequeño problema de ID3: tiene una cierta tendencia a favorecer la elección de atributos con muchos valores posibles, lo que redunda en una peor generalización de las observaciones
  - Para evitarlo el algoritmo C4.5 utiliza como criterio de selección de atributos la ratio de ganancia de información
  - El algoritmo también incluye manejo de variables continuas, valores perdidos, mecanismo de poda
- Quinlan refinó el algoritmo C4.5 en la versión C5.0, pero es una versión comercial licenciada a través de la empresa *RuleQuest Research*
  - *Existen versiones libres, pero que son también interpretaciones "libres" del algoritmo*
- En sklearn se usa CART (Classification and Regression Trees) que es muy parecido a C4.5.

APRENDIZAJE SUPERVISADO
REDES NEURONALES

Redes Neuronales

- Las redes neuronales son una de las técnicas de aprendizaje automático más utilizadas hoy en día por su capacidad de aprender a partir de conocimiento no estructurado (imágenes, sonido, texto, …).
  - Es un caso paradigmático de enfoque de aprendizaje subsimbólico

- Inicialmente inspiradas en el modelo biológico de neuronas interconectadas. Hoy en día nos interesa más su estudio desde el punto de vista matemático.

- Una red neuronal es una función capaz de representar complejas relaciones no lineales entre los datos de entrada y la variable a predecir.

- Cada neurona es una unidad de proceso distribuido paralelo.
  - Desarrollo de hardware especializado: GPUs, tensor cores, …

Evolución histórica

- **Fase inicial (1943-1969)**
  - 1943: McCulloch y Pitts
  - 1949: Hebb, modelo de memoria dinámica
  - 1951 Minsky y Edmons construyen el primer ordenador que implementa una red neuronal
  - 1957-1962: Rosenblatt, propone el perceptrón con un algoritmo de aprendizaje
  - 1969: Minsky y Papert, limitaciones teóricas del perceptrón

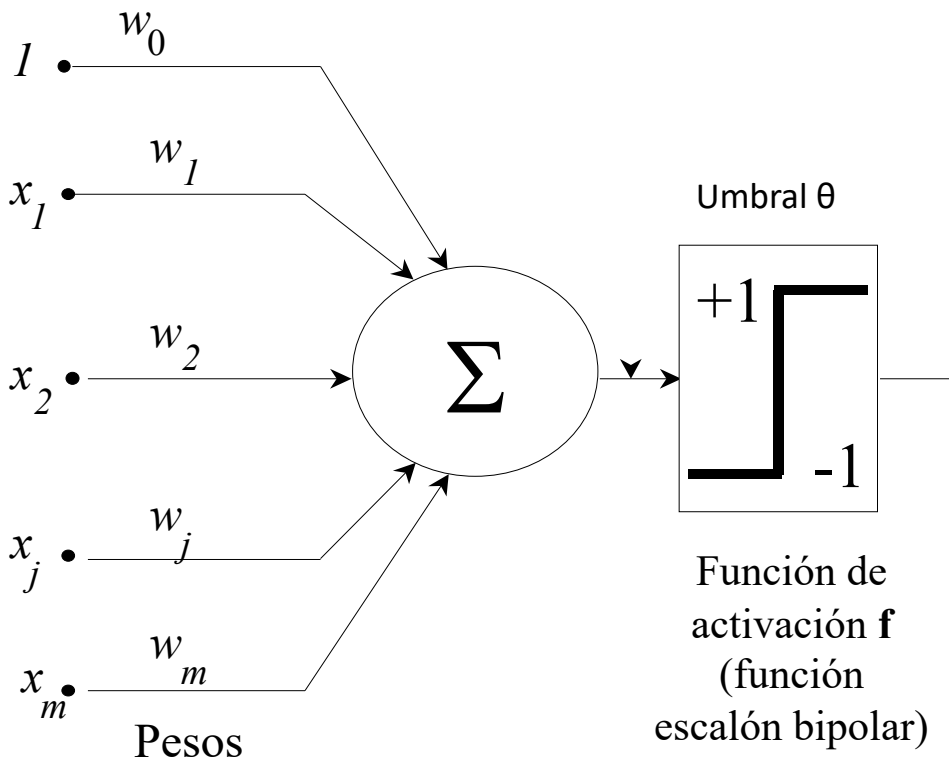
- **Resurgimiento (1975-1992)**
  - 1975: Werbos propone el algoritmo de bacpropagation.
  - 1982: Hopfield, redes autoasociativas (reverberaciones estables para construir memorias)
  - 1984: Hinton, máquina de Boltzmann
  - 1986: Rumelhart, Hinton y Williams muestran experimentalmente que el aprendizaje por retropropagación genera representaciones internas en las capas ocultas del perceptrón
  - 1987: 1ª conferencia DARPA
  - 1992: max-pooling para ayudar a reconocer objetos 3D

Evolución histórica

- **Aprendizaje profundo (>2010)**
  - 2010: Ciresan, *backpropagation* usando GPUs en redes multicapa
  - 2012: Ng y Dean, aprendizaje de conceptos abstractos (ej: gatos) a partir de imágenes no etiquetadas
  - 2009-2012: acercamiento a nivel humano en varias tareas (reconocimiento de patrones en imágenes, reconocimiento de escritura, …)
  - 2013: DeepMind, aprendizaje por refuerzo profundo para jugar a juegos de Atari.
  - 2014: DeepFace, reconocimiento de caras de Facebook con rendimiento “humano”.
  - 2016: AlphaGo, capaz de ganar a jugadores de Go profesionales.

Perceptrón simple

x
Vector de entrada con *m* variables y una constante



$$\mathbf{w} \cdot \mathbf{x} = \sum_{j=1}^{m} w_j x_j + w_0$$

Perceptrón simple

- Para cada ejemplo de entrenamiento (x_i,y_i) el perceptrón calcula

$$\hat{y}_i = f\left(w_0 + \sum_{j=1}^{m} w_j x_{i,j}\right)$$

- En función del valor real (y_i) y de la salida de la red ($\hat{y}_i$) se calcula el error para cada ejemplo de entrenamiento

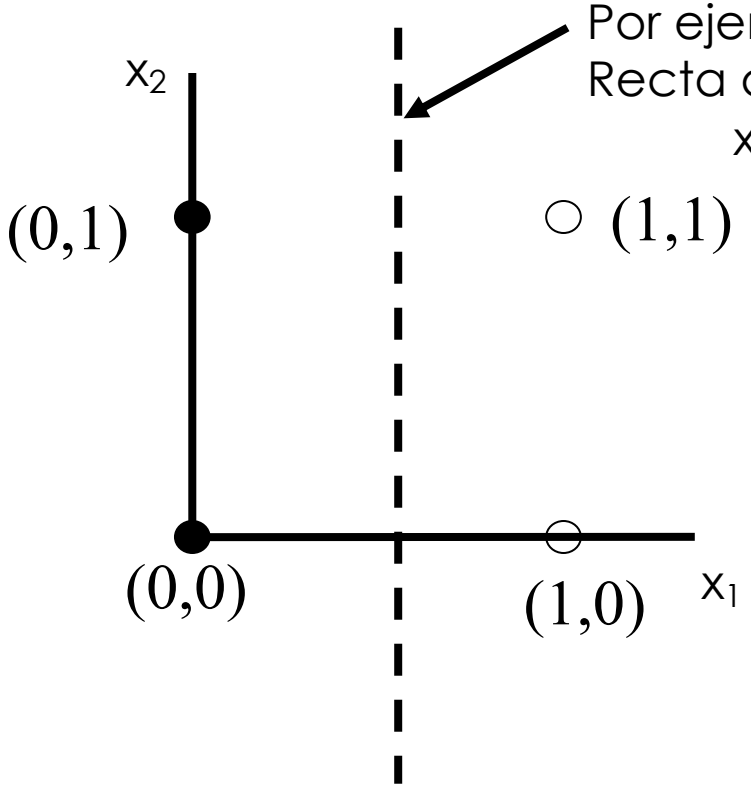
- **Entrenar la red significa encontrar los valores de los pesos w_j que minimicen el error total**
  - Búsqueda en el espacio de pesos

- Compromiso entre tiempo, error y nivel de generalización

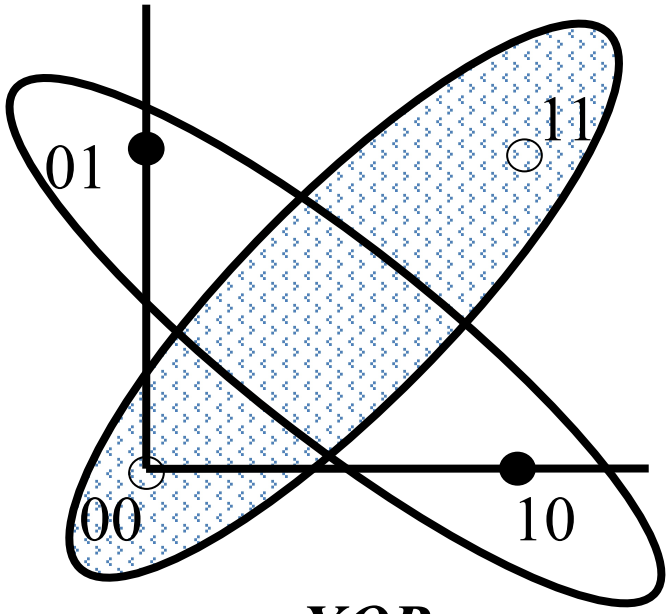
Limitaciones del Perceptrón

Función de decisión del ejemplo anterior:

Cualquier recta que separe las clases c1 y c2



…pero No existe recta que separe estas clases:



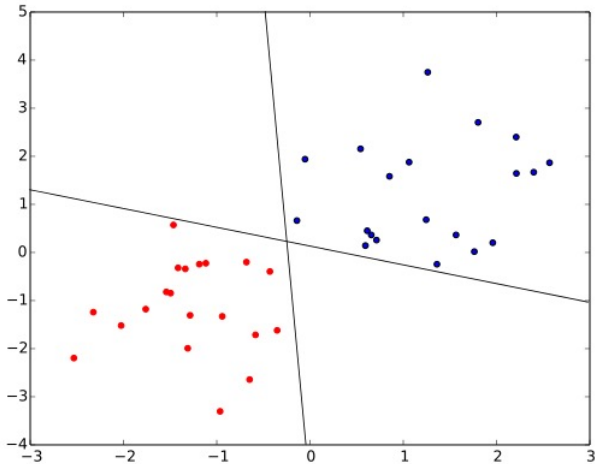
XOR

…necesitamos dos rectas para separar estas clases

… y cómo se consiguen?

Idea base del perceptrón multicapa

- El perceptrón simple es una combinación lineal de las variables de entrada, y por tanto permite "configurar" un hiperplano que separe dos clases
  - El hiperplano será eficaz solamente si las clases son linealmente separables, es decir, si una clase puede quedar a un lado del hiperplano y la otra al otro

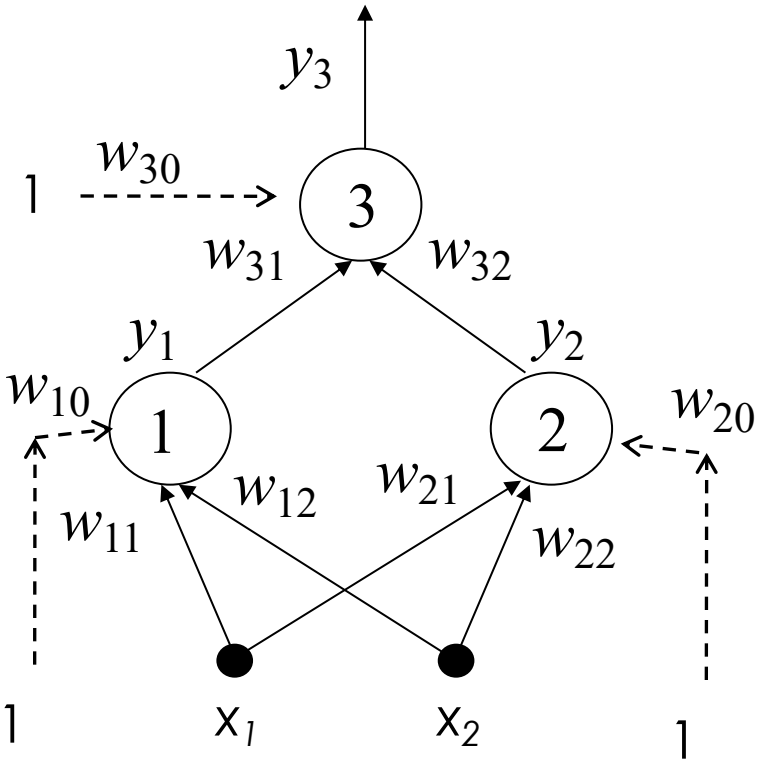


Dos de las posibles rectas que separan ambas clases (Imagen de Wikimedia.org)

- En el perceptrón multicapa la combinación de neuronas en diferentes capas, permite crear diferentes hiperplanos que al combinarse linealmente en la siguiente capa crean hipersuperficies no lineales que permiten separar las clases

Perceptrón Multicapa

Ejemplo: solución para XOR



Pesos

w_{10}=-0.5 w_{11}= 1 w_{12}= 1
w_{20}=-1.5 w_{21}= 1 w_{22}= 1
w_{30}=-0.5 w_{31}= 1 w_{32}= -1.5

Salidas de las neuronas

$y_1 = f(w_{11}x_1 + w_{12}x_2 + w_{10}) = f(x_1 + x_2 - 0.5)$
$y_2 = f(w_{21}x_1 + w_{22}x_2 + w_{20}) = f(x_1 + x_2 - 1.5)$
$y_3 = f(w_{31}y_1 + w_{32}y_2 + w_{30}) = f(y_1 - 1.5\, y_2 - 0.5)$

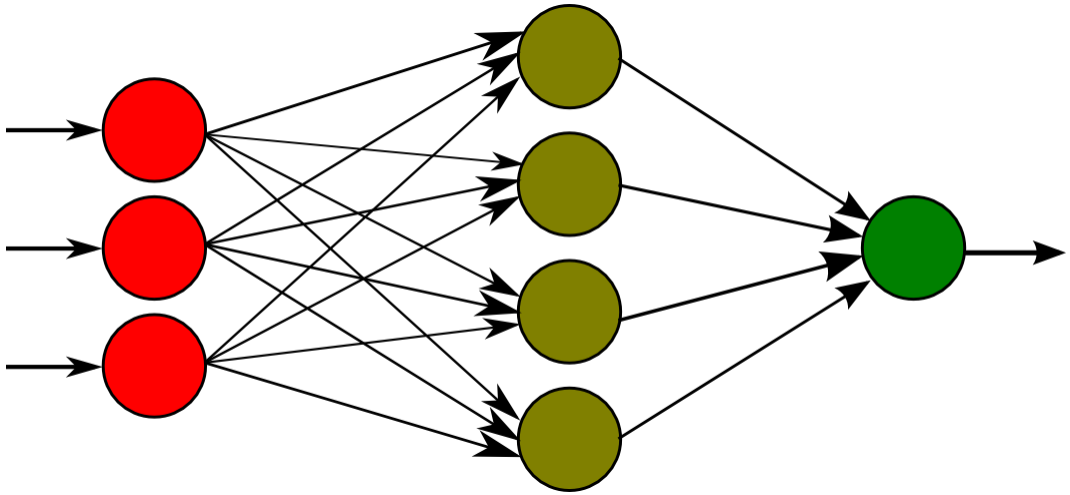
Resultados

x_1	x_2	y_1	y_2	y_3 = XOR
0	0	$f(-0.5) = 0$	$f(-1.5) = 0$	$f(-0.5) = 0$
0	1	$f(0.5) = 1$	$f(-0.5) = 0$	$f(0.5) = 1$
1	0	$f(0.5) = 1$	$f(-0.5) = 0$	$f(0.5) = 1$
1	1	$f(1.5) = 1$	$f(0.5) = 1$	$f(-1.0) = 0$

Función de activación: escalón unidad (f (x) = 1 si x > 0; f (x) = 0 si x <= 0)

Perceptron Multicapa (MLP)

- El perceptrón multicapa (*multi-layer perceptron* o *MLP*) es una red neuronal de tipo perceptrón con **al menos** tres capas
  - 1 **capa de entrada**, tantas neuronas como variables de entrada
  - 1 o más **capas ocultas** con un número variable de neuronas cada una de ellas con una función de activación
  - 1 **capa de salida** con una o varias neuronas con su función de activación
    - 1 neurona si es un problema de clasificación binario o de regresión simple, o tantas neuronas como clases en problemas de clasificación múltiple
- La capa intermedia hace que el MLP sea un "aproximador universal de funciones" capaz de modelar **relaciones no lineales** entre las variables de entrada y de salida.



Funciones de activación

- Las funciones de activación son funciones no lineales que permiten modelar funciones complejas “apilando” capas en la red neuronal.
  - Sin función de activación cada capa calcularía una combinación lineal de las salidas de la capa anterior. Y una combinación lineal de combinaciones lineales sigue siendo una combinación lineal ➔ no ganaríamos expresividad
  - Con las funciones de activación se calculan combinaciones lineales de funciones no lineales
- Las funciones de activación pueden ser de distintos tipos: logística, tangente hiperbólica, rectificada lineal (ReLU)
  - Todas ellas monótonas crecientes y derivables
    - Es necesario que sean derivables para poder entrenar la red (como veremos más adelante).
  - Las funciones de activación de la capa de salida dependen de los rangos de las variables que estamos modelando.
    - ¿La salida es una probabilidad? Logística
    - ¿La salida es un número positivo? ReLU
    - ¿La salida es un número entre -1 y 1? Tangente hiperbólica
    - ¿La salida es un número real? Identidad (sin función de activación)

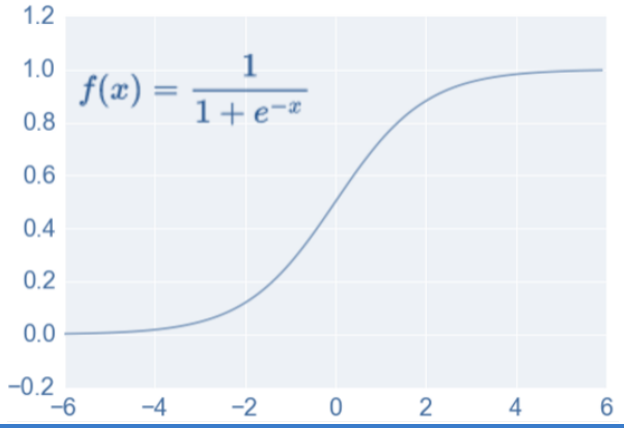
Tipos de función de activación

- **Logística**: Función sigmoidal que varía entre 0 y 1
  - Permite representar la probabilidad de pertenecer o no a la clase
  - La función sigmoidal es acotada, monótona creciente y diferenciable

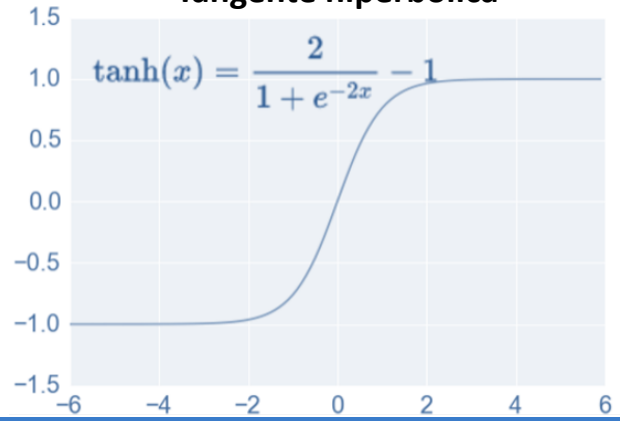
- **Tangente hiperbólica**: Función sigmoidal que varía entre -1 y 1
  - Permite asociar valores negativos a entradas negativas lo que la hace adecuada para separar entre dos clases

- **Identidad:** Útil en la capa de salida para problemas de regresión no acotados

Logística

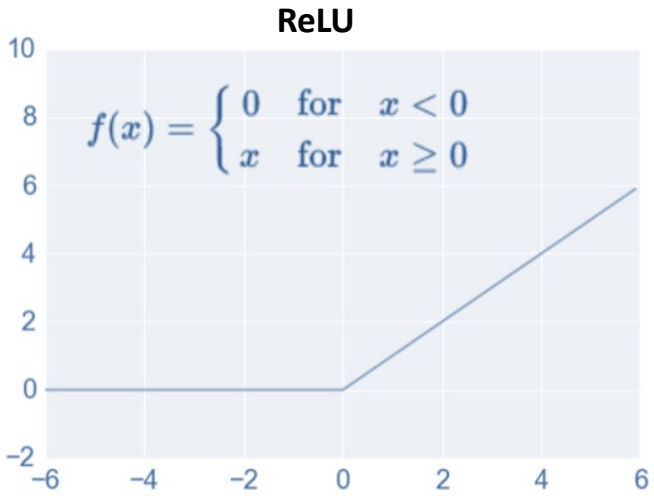


Tangente hiperbólica



Tipos de función de activación

- **Rectificada linear (ReLU):** Función con dos tramos, uno lineal que varía entre 0 e Infinito (muy usada en Deep learning)
  - Al ser esencialmente una función lineal reduce mucho el tiempo de cálculo
  - Al no estar acotada en un extremo no satura, lo cual es beneficioso en la fase de aprendizaje y para modelar problemas de regresión
  - A diferencia de las anteriores, puede tomar el valor cero (las otras se aproximan infinitamente), lo que inhibe la activación de la neurona y reduce los tiempos de cálculo en redes muy grandes



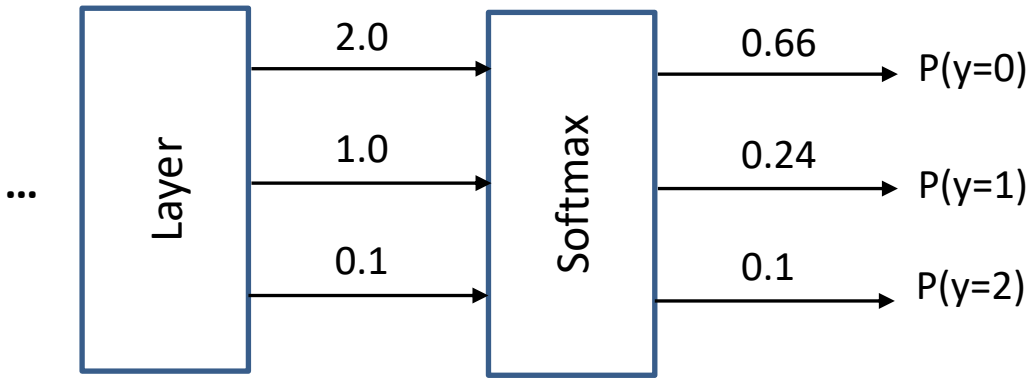
La función de activación de la capa de salida queda determinada por el rango de valores de la función que queremos modelar.

Las funciones de activación de las capas internas son un parámetro más de la Red Neuronal aunque actualmente ReLU es la opción más habitual.

Tipos de función de activación

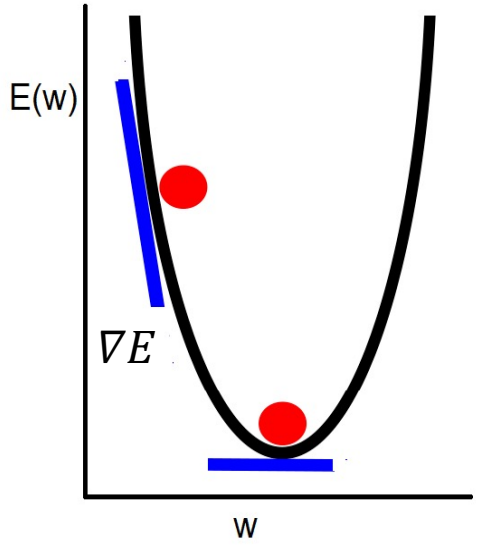
- **Softmax:** Transforma un conjunto de valores numéricos al rango [0,1] con suma total 1
  - Habitualmente se emplea en la capa de salida en problemas de clasificación multiclase (n clases ➔ n salidas)
  - Permite re-interpretar las salidas de la red como probabilidades (valores entre 0 y 1 que suman 1)
    - La probabilidad de que el ejemplo pertenezca a cada una de las n clases

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_k \exp(x_k)} \text{ para i} = 1, \ldots, n$$



Descenso por gradiente

- Dada una función de error E(w_1, .., w_s) diferenciable, podemos encontrar uno de sus mínimos (locales) mediante el algoritmo de descenso de gradiente
  - Partimos de un punto aleatorio *p* con un cierto error *E(p)*
  - Calculamos el gradiente de la función $\nabla E = (\frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_s})$ en el punto p
    - El vector gradiente indica la dirección en la que la función de error crece más rápido
  - Calculamos un nuevo punto p = p $-\alpha \nabla E$ y volvemos al paso anterior.
    - α indica cuanto nos desplazamos y se conoce como tasa de aprendizaje (*learning rate*)



En cada paso del algoritmo disminuimos el error hasta llegar a un mínimo local donde el gradiente será 0.

Si la función no es convexa no se garantiza encontrar un mínimo global.

Algoritmo de retropropagación

- Idea: disminuir el error de la red mediante descenso de gradiente
- Se inicializan los pesos de forma aleatoria y con valores pequeños
- Para cada ejemplo *k* del conjunto de entrenamiento se hacen dos fases:
  - Hacia delante
    - Calculamos la salida de la red para el ejemplo $\hat{y}_k$
    - Calculamos el error (*loss value*) entre la salida real de la red y la salida deseada para ese ejemplo $E_k(y_k, \hat{y}_k)$
  - Hacia atrás
    - Calculamos el gradiente de la función de error con respecto a los pesos de la red (mide cuánto afecta un pequeño cambio en cada peso al error) para el ejemplo *k*

$$\nabla E_k = \left(\frac{\partial E_k}{\partial w_{10}}, \ldots, \frac{\partial E_k}{\partial w_{rs}}\right)$$

      - Se calcula usando la regla de la cadena desde la capa final de la red hacia atrás
    - Ajustamos los pesos para que reduzcan el error respecto al ejemplo k

$$w' = w - \alpha \nabla E_k$$

Algoritmo de retropropagación

- De esa forma reducimos el error con respecto al ejemplo *k,* pero en realidad nos interesa reducir el error total con respecto a todos los ejemplos

$$E_{total} = \sum E_k$$

- Así que en realidad calculamos el gradiente medio y modificamos los pesos de la red en dirección contraria

$$\nabla E_{total} = \frac{1}{n} \sum \nabla E_k$$

$$w' = w - \alpha \nabla E_{total}$$

- La actualización de la red para todo el conjunto de entrenamiento recibe el nombre de **época** (*epoch*)
  - El entrenamiento de una red puede requerir entre decenas y miles de épocas

Cálculo del error

- Es importante utilizar una función de error que sea derivable para entrenar la red. A esta función se le llama **función de pérdida** (loss).

- Error en problemas de regresión: error cuadrático

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

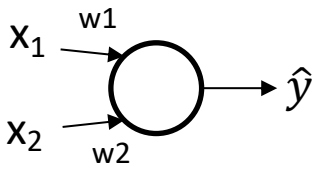
- Error en problemas de clasificación: entropía cruzada. Por ejemplo, para dos clases:

$$CE = \frac{1}{n}\sum_{i}^{n} y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i)$$

- El error de la red se calcula como la suma de los errores en cada uno de los ejemplos de entrada.

Ejemplo

Red neuronal



Función de activación: sigmoide

Función de error: $E = \frac{1}{2}(y - \hat{y})^2$

Cálculo del gradiente

$$z = w_0 + x_1 w_1 + x_2 w_2 \qquad \hat{y} = \frac{1}{1 + e^{-z}} \qquad \nabla E = (\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2})$$

Sólo voy a calcular una de las componentes del gradiente a modo de ejemplo usando la regla de la cadena:

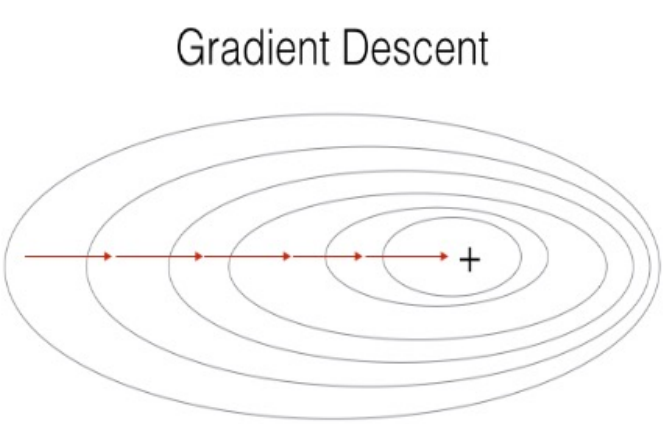
$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial w_1} \qquad \frac{\partial E}{\partial \hat{y}} = -(y - \hat{y}) \qquad \frac{\partial \hat{y}}{\partial z} = \hat{y}(1 - \hat{y}) \qquad \frac{\partial z}{\partial w_1} = x_1$$

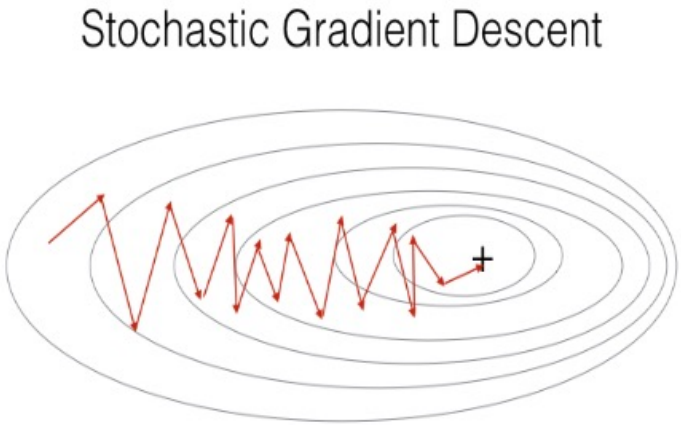
Una vez tengo el gradiente puedo evaluarlo en cualquier punto y me dará un vector de 3 componentes. Calculo el gradiente medio para todos los puntos del conjunto de entrenamiento y actualizo los pesos:

$$w'_0 = w_0 - \alpha \frac{\partial E_{total}}{\partial w_0} \qquad w'_1 = w_1 - \alpha \frac{\partial E_{total}}{\partial w_1} \qquad w'_2 = w_2 - \alpha \frac{\partial E_{total}}{\partial w_2}$$

Minibatches

- El gradiente nos indica la dirección que mejor minimiza el error pero para calcularlo es necesario recorrer todos los ejemplos de entrenamiento en cada paso
  - Esto es muy costoso computacionalmente
- Una alternativa sería calcular el gradiente a cada ejemplo (GD estocástico)
  - La convergencia fluctúa mucho
- Lo que se hace es dividir los ejemplos de entrenamiento en conjuntos pequeños (32, 64, …) y dar muchos más pasos en cada *epoch*.
  - Búsqueda más eficiente pero con mucho más ruido
  - Estándar de facto: converge mucho antes





Regularización

- Para evitar el sobreaprendizaje podemos utilizar varias estrategias
  - Conseguir más ejemplos de entrenamiento
  - Interrumpir antes el proceso de entrenamiento
  - Utilizar técnicas de regularización

- Regularización L2

$$Error = \frac{1}{n}\sum_{i}^{n}(y_i - \hat{y}_i)^2 + \frac{\lambda}{2n}\sum_{i}^{n}\|w\|^2$$

  - La idea es penalizar los pesos demasiado grandes para que la red generalice mejor.
  - λ es la tasa de regularización (permite controlar cuánto regularizar)
  - Otras técnicas usadas: regularización L1 y *dropout* (eliminación probabilística de nodos).

Consideraciones de uso del percetrón multicapa

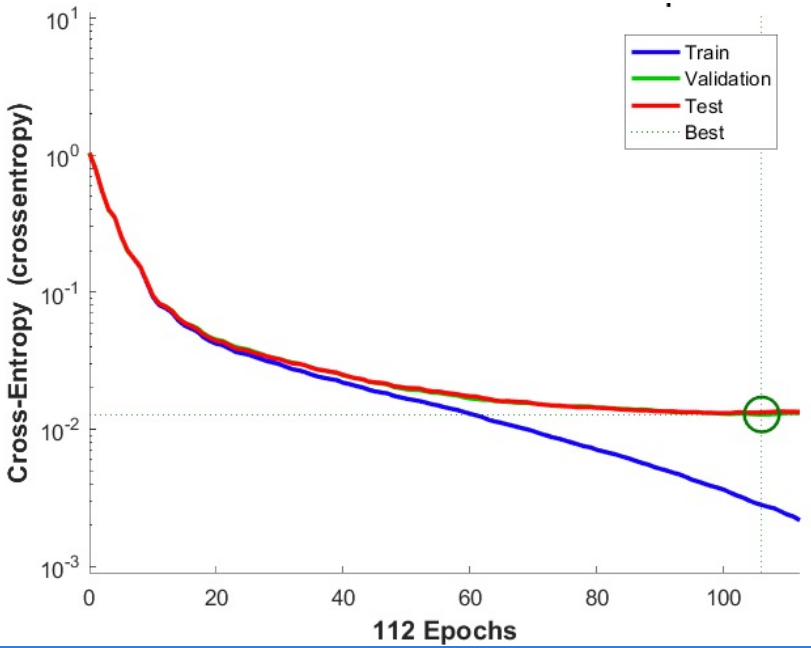
- Los datos de entrada deben ser numéricos
  - Las variables categóricas binarias se pueden adaptar con 0 y 1
  - Las variables categóricas múltiples como un vector binario de tantos elementos como clases
- El MLP puede tratar con variables sin normalizar o sin estandarizar
  - Sin embargo, la normalización o estandarización de las variables disminuye el tiempo de entrenamiento y evita los “mínimos locales” en la optimización del error
- En principio el MLP es capaz de tratar con variables irrelevantes, ya que les acabará asignando peso cero
  - La arquitectura del perceptrón no será necesariamente más compleja (no requerirá de más neuronas en la capa oculta)
  - Sin embargo, añadir variables irrelevantes hace que sean necesarios más datos de entrenamiento

Configurando el perceptrón multicapa

- La arquitectura del MLP la determina el número de capas (a la hora de contarlas se suele ignorar la de entrada) y el número de neuronas de cada capa
  - En principio, un MLP con una única capa oculta y "suficientes" neuronas, puede resolver igual un problema que un MLP con más capas ocultas
  - Las capas ocultas añaden "representaciones" de los datos con un nivel de abstracción mayor y en principio es más fácil llegar a la solución a partir de ellas
    - En esto se basa grosso modo el deep learning que usa gran cantidad de capas intermedias
  - Determinar el número de capas y de neuronas puede hacerse probando varias configuraciones y determinando la mejor mediante validación cruzada

Configurando el perceptrón multicapa

- El MLP cuenta también con parámetros (tasa de aprendizaje, regularización) que controlan el sobreaprendizaje del modelo
  - La validación cruzada también nos ayuda a ajustar el modelo del MLP y no sobreaprender los datos de entrenamiento
  - De hecho, también hay estrategias de regularización que controlan el error en validación en paralelo al entrenamiento y detienen este último si el error de validación aumenta



Parámetros del perceptrón multicapa

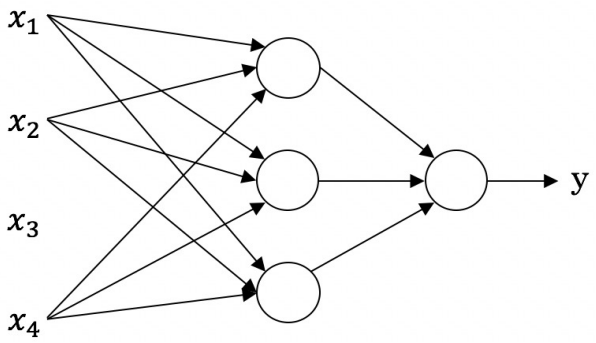
- Número de capas y número de neuronas por capa
  - Intuición: necesitaremos una red más compleja cuando más compleja sea la función que tratamos de aproximar
- Funciones de activación de las capas intermedias
  - Las de la capa de salida suelen quedar determinada por el tipo de problema
- Tasa de aprendizaje
  - Cuanto más pequeña mejores resultados puede encontrar pero más tiempo de entrenamiento
  - Hay estrategias para modificarla dinámicamente
- Tipo de regularización y tasa de regularización
  - Regularización L1 o L2, dropout, etc.
- Criterio para dejar de entrenar
  - Evitar sobreaprendizaje vs encontrar buenos mínimos del error
- ¡Ojo! a veces el problema no está en la red sino en los datos

Elegir los parámetros de la red

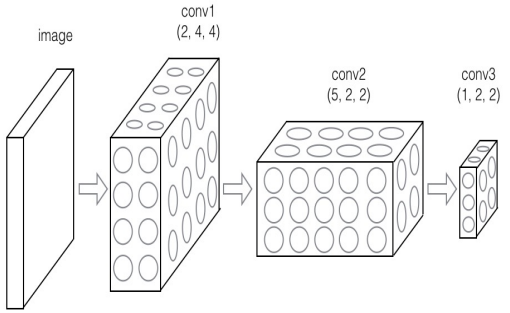
- Cualquier estrategia de validación cruzada sirve, por ejemplo:
  - Dividir el conjunto de datos en 3 trozos: entrenamiento, validación y test.
  - Elegir una configuración de parámetros, entrenar la red con el conjunto de entrenamiento y evaluar su funcionamiento con el conjunto de validación.
  - Una vez he encontrado una buena configuración evaluamos su funcionamiento con el conjunto de test.
  - **Lo importante es no usar el mismo subconjunto de datos para entrenar, configurar y evaluar.**
- Podemos simplificar la búsqueda de configuraciones de parámetros buscando el valor de cada uno de forma secuencial
  - Optimizar uno de ellos, dejarlo fijo y optimizar el siguiente
  - Estrategia voraz: mucho más rápido pero no es óptimo
- Cada entrenamiento puede tardar minutos, horas, días…

Otros tipos de redes neuronales

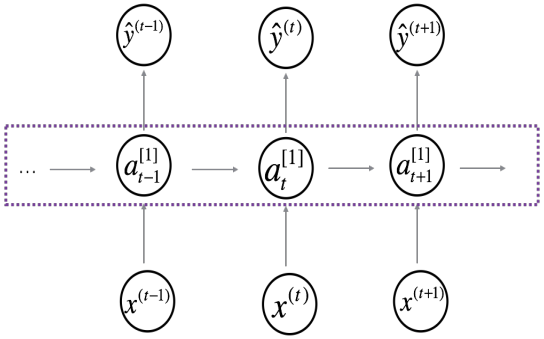
- **Perceptrón multicapa**: rede neuronal básica
- **Redes convolucionales**: tratamiento de imágenes
- **Redes recurrentes**: tratamiento de secuencias y series temporales (Hopfield, Boltzmann, LSTM)
- **Arquitecturas híbridas** y ad-hoc: topologías mixtas creadas a medida para un dominio concreto.
- **Mapas auto-organizados**: red neuronal para aprendizaje no supervisado



Perceptrón multicapa



Convolutional NN



Recurrent NN

MÁS SOBRE APRENDIZAJE AUTOMÁTICO

Aplicaciones de aprendizaje automático

- Procesado de voz
- Procesado de imágenes
  - Clasificación de imágenes
  - Análisis de pruebas médicas
  - Monitorización y control
- Sensorización, control y robótica
- Diagnóstico
  - Diagnóstico médico
  - Averías
  - Morosidad
- Segmentación de clientes y estudios de mercado
- Control y robótica
- Bioinformática
- Finanzas
- Etc

Otras técnicas de aprendizaje automático

- Las diferentes técnicas de aprendizaje automático pueden ayudar a solucionar un problema y la elección de una o de otra depende de las características del problema que tengamos
  - Problema de clasificación, de regresión, de detección de relaciones entre variables o de ocurrencia de sucesos, detección de anomalías o de fraudes,
  - Volumen de datos (número de ejemplos)
  - Calidad de los datos, p.ej. valores perdidos y escasos (sparsity)
  - Cantidad de las variables y calidad o relevancia de las mismas
  - etc

- Además, la complejidad de los datos en situaciones reales hace que la pericia de la persona al frente del problema sea determinante
  - Entendiendo el problema y cómo los datos pueden resolverlo
  - Limpiando y transformando los datos,
  - Seleccionando las variables relevantes,
  - Interpretando los resultados
  - Ayudando a traducir los resultados en decisiones útiles

Otras técnicas de aprendizaje automático

- Hay una gran cantidad de técnicas de aprendizaje automático y cada técnica puede tener su punto fuerte sobre otras. Hay muchas más de las que hemos visto:
  - Máquinas de Vectores de Soporte (*Support Vector Machines*)
  - Reglas asociativas
  - Naïve Bayes
  - …
- Hoy día en problemas de aprendizaje supervisado suelen obtener muy buenos resultados las técnicas que agregan muchos "predictores" sencillos (*ensemble learning*)
  - En lugar de usar un único predictor que aprenda "todo" el conjunto de datos, se combinan múltiples predictores sencillos que se "especializan" en partes del conjunto de datos.
  - Las dos estrategias más usadas para combiner predictors son:
    - Votación (*bagging*): por ejemplo, los random forests son multiples árboles de decision sencillos donde cada uno clasifica (vota) la clase a la que pertenece el ejemplo nuevo y la clase que se le asigna es la que gane la votación por mayoría
    - *Boosting*: Iterativamente se concatenan predictores de forma que cada uno de ellos intenta "corregir" los errores que cometió el predictor anterior, es decir, aprender a predecir lo que el anterior erró

Retos y nuevas tendencias del aprendizaje automático

- **Data Science**
  - La ciencia de datos es un área que combina la estadística, la minería de datos, el aprendizaje automático, la visualización de datos y la recuperación y el procesado de datos de fuentes diversas (bases de datos, sensores, ficheros, etc)
  - Surge para hacer frente a los desafíos del almacenamiento masivo de datos y la necesidad de convertir dichos datos en información y en soporte a la toma de decisiones

- **Big Data**
  - Aunque el término se usa a menudo inadecuadamente, se refiere a tres “V” :
    - Variedad: los tipos de datos son diversos (imágenes, sensores, texto, etc.)
    - Velocidad: supone un reto tratar los datos en tiempo real y en algún caso se descarta su almacenamiento
    - Volumen: los datos que se manejan tienen un volumen tal que no cabe en la memoria de un ordenador
  - A menudo se añaden Variabilidad (cambiantes) y Veracidad (confiables)
  - Las tecnologías big data son aquellas que procesan los datos en la nube en paralelo y que reformulan algoritmos de aprendizaje estadístico en paralelo o en tiempo real

Retos y nuevas tendencias del aprendizaje automático

- **Deep learning**
  - El deep learning es un paradigma de aprendizaje fundamentalmente basado en redes neuronales
  - Se caracteriza por la existencia de numerosas capas intermedias de forma que en la red los objetos se expresan como una composición de primitivas de sofisticación creciente
  - La capacidad de aprendizaje es muy elevada y se obtienen resultados muy buenos en vision artificial, reconocimiento del habla, procesamiento de lenguaje natural, etc.

Retos y nuevas tendencias del aprendizaje automático

- **Explainable Artificial Intelligence**
  - Depender de las decisiones de un sistema de IA que no sabemos cómo funciona puede generar frustración o desconfianza
    - Especialmente en dominios críticos: sistemas médicos, coches autónomos
  - En el aprendizaje automático abundan las técnicas *opacas* como las redes neuronales (deep learning) que cada vez se implantan más
  - La IA Explicable busca que los modelos expliquen por qué toman una decisión y no otra, así como sus fortalezas y limitaciones

Referencias

- **Witten, I.H., Frank, E., Hall, M.A.**
  Data Mining
  Elsevier, 2017 (versión electrónica en la Biblioteca UCM)

- **Chollet, F.**
  Deep Learning with Python
  Manning, 2017 (https://www.manning.com/books/deep-learning-with-python)

- **James, G. et al.**
  An Introduction to Statistical Learning with Applications in R
  Springer, 2013 (http://www-bcf.usc.edu/~gareth/ISL/)

- **Borrajo, D., González, J., Isasi, P.**
  Aprendizaje automático
  Sanz y Torres, 2006

- **Sierra, B.**
  Aprendizaje automático
  Pearson Prentice-Hall, 2006

- **Hilera, J.R., Martínez, V. J.**
  Redes Neuronales Artificiales
  RA-MA, 1995