Inteligencia Artificial

Tema 6: Introducción al Aprendizaje Automático

1. Introducción

Entendemos por aprendizaje a cualquier cambio en un sistema que le permite realizar la misma tarea de manera **más eficiente** la próxima vez. También podemos definirlo cómo la modificación del entorno que se está percibiendo.

Se dice que aprendemos de la experiencia de realizar una tarea si la realización de la misma mejora con dicha experiencia respecto a alguna medida de rendimiento.

Así, entendemos por **Aprendizaje Automático** a la idea de construir sistemas computacionales que a partir de datos y percepciones mejoran su rendimiento en la realización de una tarea determinada.

Tipos de aprendizaje:

- Supervisado
- No supervisado
- Con refuerzo

Paradigmas:

- Aprendizaje basado en instancias (memorización)
- Agrupamiento (Clustering)
- Aprendizaje inductivo
- Aprendizaje por analogía
- Descubrimiento
- Algoritmos genéticos (redes neuronales)

PD: subrayado aquellos que vamos a tratar.

Ejemplo:

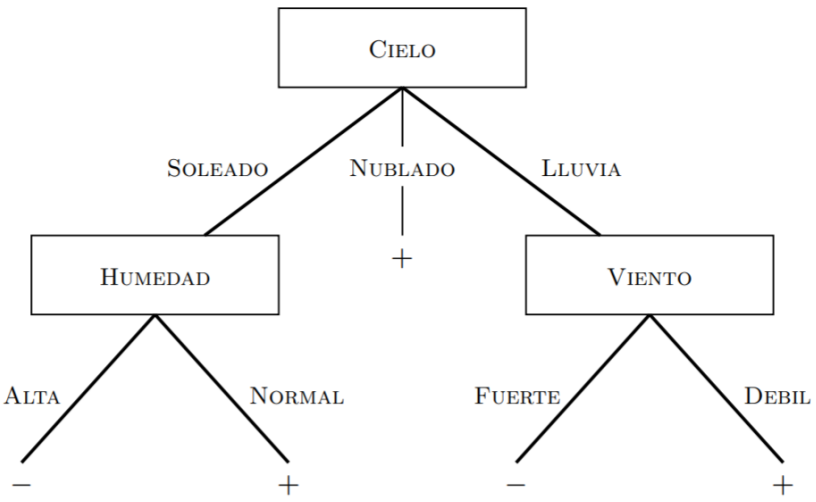
- Conjunto de entrenamiento: días en los que es recomendable (o no) jugar al tenis:

Ej.	Cielo	Temperatura	Humedad	Viento	JugarTenis
D_1	Soleado	Alta	Alta	Débil	-
D_2	Soleado	Alta	Alta	Fuerte	-
D_3	Nublado	Alta	Alta	Débil	+
D_4	Lluvia	Suave	Alta	Débil	+
...					

- Objetivo: dado el conjunto de entrenamiento, aprender el concepto "Días en los que se juega al tenis" → Aprendizaje supervisado.
- Problema: ¿cómo expresar lo aprendido? → Algoritmos que veremos a continuación.

2. Aprendizaje de árboles de decisión

Un árbol de decisión es un grafo etiquetado que representa un concepto; ejemplo:



Compuestos por:

- Nodos interiores (atributos)
- Aristas (posibles valores del nodo origen)
- Hojas (valor de clasificación: normalmente +/- pero no tiene porqué ser binario)
- Representación de una función objetivo

Clasificadores:

Algoritmo ID3

```
ID3(Ejemplos, Atributo-objetivo, Atributos)
1. Si todos los Ejemplos son positivos, devolver un nodo
   etiquetado con +
2. Si todos los Ejemplos son negativos, devolver un nodo
   etiquetado con -
3. Si Atributos está vacío, devolver un nodo etiquetado con el valor
   más frecuente de Atributo-objetivo en Ejemplos.
4. En otro caso:
4.1. Sea A el atributo de Atributos que MEJOR clasifica Ejemplos
4.2. Crear Árbol, con un nodo etiquetado con A.
4.3. Para cada posible valor v de A, hacer:
   * Añadir un arco a Árbol, etiquetado con v.
   * Sea Ejemplos(v) el subconjunto de Ejemplos con valor del
     atributo A igual a v.
   * Si Ejemplos(v) es vacío:
     - Entonces colocar debajo del arco anterior un nodo etiquetado
       con el valor más frecuente de Atributo-objetivo en Ejemplos.
     - Si no, colocar debajo del arco anterior el subárbol
       ID3(Ejemplos(v), Atributo-objetivo, Atributos-{A}).
4.4  Devolver Árbol
```

¿Qué atributo clasifica mejor?

➔ Dado la entropía de un conjunto de elementos D, dónde P y N son respectivamente los subconjuntos de ejemplos positivos y negativos de D, entonces, intuitivamente se mide la ausencia de homogeneidad de la clasificación.

$$Ent(D) = -\frac{|P|}{|D|} \cdot log_2 \frac{|P|}{|D|} - \frac{|N|}{|D|} \cdot log_2 \frac{|N|}{|D|}$$

➔ Ejemplos:

- $Ent([9^+, 5^-]) = -\frac{9}{14} \cdot log_2 \frac{9}{14} - \frac{5}{14} \cdot log_2 \frac{5}{14} = 0{,}94$
- $Ent([k^+, k^-]) = 1$ (ausencia total de homogeneidad)
- $Ent([p^+, 0^-]) = Ent([0^+, n^-]) = 0$ (homogeneidad total)

Ganancia de información → preferimos nodos con menos entropía (árboles pequeños).

➔ La entropía esperada después de usa un atributo A en el árbol será:

$$\sum_{v \in Valores(A)} \frac{|D_v|}{|D|} \cdot Ent(D_v)$$

dónde **Dv** es el subconjunto de ejemplos de **D** con valor del atributo **A** igual a **v**.

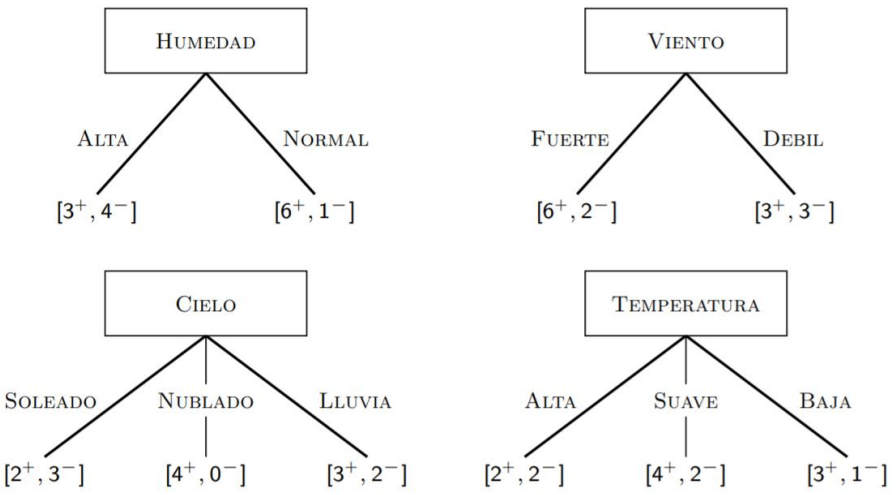
➔ La ganancia de información esperada después de usa un atributo A será:

$$Ganancia(D, A) = Ent(D) - \sum_{v \in Valores(A)} \frac{|D_v|}{|D|} \cdot Ent(D_v)$$

PD: en el algoritmo ID3, en cada nodo usamos el atributo con mayor ganancia de información.

Ejemplo del algoritmo ID3 dado el siguiente conjunto de entrenamiento:

Ej.	Cielo	Temperatura	Humedad	Viento	JugarTenis
D_1	Soleado	Alta	Alta	Débil	-
D_2	Soleado	Alta	Alta	Fuerte	-
D_3	Nublado	Alta	Alta	Débil	+
D_4	Lluvia	Suave	Alta	Débil	+
D_5	Lluvia	Baja	Normal	Débil	+
D_6	Lluvia	Baja	Normal	Fuerte	-
D_7	Nublado	Baja	Normal	Fuerte	+
D_8	Soleado	Suave	Alta	Débil	-
D_9	Soleado	Baja	Normal	Débil	+
D_{10}	Lluvia	Suave	Normal	Débil	+
D_{11}	Soleado	Suave	Normal	Fuerte	+
D_{12}	Nublado	Suave	Alta	Fuerte	+
D_{13}	Nublado	Alta	Normal	Débil	+
D_{14}	Lluvia	Suave	Alta	Fuerte	-



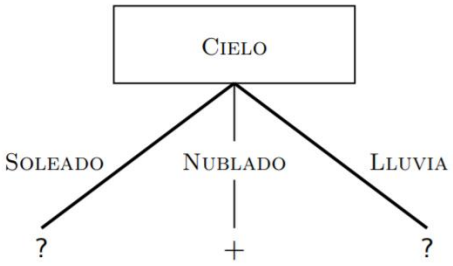
Entropía inicial: $Ent([9^+, 5^-]) = 0{,}94$

Selección del atributo para el nodo raíz:

- $Ganancia(D, \text{HUMEDAD}) = 0{,}94 - \frac{7}{14} \cdot Ent([3^+, 4^-]) - \frac{7}{14} \cdot Ent([6^+, 1^-]) = 0{,}151$
- $Ganancia(D, \text{VIENTO}) = 0{,}94 - \frac{8}{14} \cdot Ent([6^+, 2^-]) - \frac{6}{14} \cdot Ent([3^+, 3^-]) = 0{,}048$
- $Ganancia(D, \text{CIELO}) = 0{,}94 - \frac{5}{14} \cdot Ent([2^+, 3^-]) - \frac{4}{14} \cdot Ent([4^+, 0^-]) - \frac{5}{14} \cdot Ent([3^+, 2^-]) = 0{,}246$ (mejor atributo)
- $Ganancia(D, \text{TEMPERATURA}) = 0{,}94 - \frac{4}{14} \cdot Ent([2^+, 2^-]) - \frac{6}{14} \cdot Ent([4^+, 2^-]) - \frac{4}{14} \cdot Ent([3^+, 1^-]) = 0{,}02$

El atributo seleccionado es CIELO

Árbol parcialmente construido:



Selección del atributo para el nodo CIELO=SOLEADO

$D_{\text{SOLEADO}} = \{D_1, D_2, D_8, D_9, D_{11}\}$ con entropía $Ent([2^+, 3^-]) = 0{,}971$

- $Ganancia(D_{\text{SOLEADO}}, \text{HUMEDAD}) = 0{,}971 - \frac{3}{5} \cdot 0 - \frac{2}{5} \cdot 0 = 0{,}971$ (mejor atributo)
- $Ganancia(D_{\text{SOLEADO}}, \text{TEMPERATURA}) = 0{,}971 - \frac{2}{5} \cdot 0 - \frac{2}{5} \cdot 1 - \frac{1}{5} \cdot 0 = 0{,}570$
- $Ganancia(D_{\text{SOLEADO}}, \text{VIENTO}) = 0{,}971 - \frac{2}{5} \cdot 1 - \frac{3}{5} \cdot 0{,}918 = 0{,}019$

El atributo seleccionado es HUMEDAD

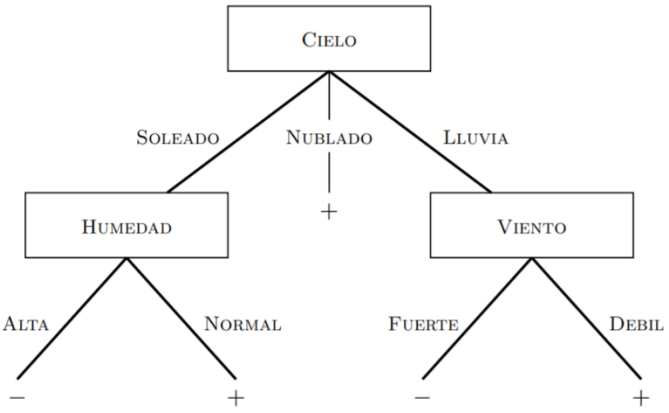
Selección del atributo para el nodo CIELO=LLUVIA:

$D_{\text{LLUVIA}} = \{D_4, D_5, D_6, D_{10}, D_{14}\}$ con entropía $Ent([3^+, 2^-]) = 0{,}971$

- $Ganancia(D_{\text{LLUVIA}}, \text{HUMEDAD}) = 0{,}971 - \frac{2}{5} \cdot 1 - \frac{3}{5} \cdot 0{,}918 = 0{,}020$
- $Ganancia(D_{\text{LLUVIA}}, \text{TEMPERATURA}) = 0{,}971 - \frac{3}{5} \cdot 0{,}918 - \frac{2}{5} \cdot 1 = 0{,}020$
- $Ganancia(D_{\text{LLUVIA}}, \text{VIENTO}) = 0{,}971 - \frac{3}{5} \cdot 0 - \frac{2}{5} \cdot 0 = 0{,}971$ (mejor atributo)

El atributo seleccionado es VIENTO

Árbol finalmente aprendido:



Valida la hipótesis aprendida: Medida del rendimiento del aprendizaje

- ➔ Conjuntos de entrenamiento y prueba (test); se aprende del conjunto de entrenamiento y se mide el rendimiento con el conjunto de prueba.
- ➔ A veces se emplea un tercer conjunto para validar el ajuste de parámetros del modelo.
- ➔ Si no hay suficientes ejemplos para construir un conjunto de prueba: validación cruzada:
    - Dividir el conjunto de aprendizaje en *k* partes, hacer *k* aprendizajes, cada uno de ellos tomando cómo prueba una de las de las partes y el resto cómo entrenamiento. Finalmente hacer la media de los rendimientos.

Sobreajuste: Lo aprendido se ajusta demasiado a los datos de aprendizaje

- ➔ Definimos un conjunto de hipótesis *H*: posibles modelos a aprender.
- ➔ Una hipótesis $h \in H$ sobreajusta los ejemplos de entrenamiento si existe un $h' \in H$ que se ajusta peor que *h* a los ejemplos, pero actúa mejor sobre la distribución completa de instancias.
- ➔ Causas del sobreajuste:
    - Ruido (errores en los datos).
    - Atributos que en los ejemplos presentan una aparente regularidad pero que no son relevantes en realidad.
    - Conjuntos de entrenamiento demasiado pequeños.
- ➔ Formas de evitarlo:
    - Evitar modelos excesivamente complejos (penalizar la complejidad).
    - Medir el rendimiento sobre conjuntos de validación independientes para comprobar la generalización de lo aprendido.
- ➔ Formas de evitarlo en árboles de decisión:
    - Parar el desarrollo del árbol durante el aprendizaje, antes de que se ajuste perfectamente a todos los datos (early stopping).
        - No generar árboles por encima de una profundidad dada.
        - Parar en nodos que corresponden con un porcentaje bajo de datos.
        - Parar en nodos en los que la mayoría de los ejemplos son de una clase.
    - Podar el árbol a posteriori (post pruning): dos formas:
        - Transformación a reglas → podado de las condiciones de las reglas.
        - Realizar podas directamente sobre el árbol.

Algoritmo de poda para reducir el error

```
1. Dividir el conjunto de ejemplos en Entrenamiento, Validación (y Prueba)
2. Árbol=árbol obtenido por ID3 usando Entrenamiento
3. Continuar=True
4. Mientras Continuar:
 * Medida = proporción de ejemplos en conjunto de Validación correctamente
   clasificados por Árbol
 * Por cada nodo interior N de Árbol:
   - Podar temporalmente Árbol en el nodo N y sustituirlo por una
     hoja etiquetada con la clasificación mayoritaria en ese nodo
   - Medir la proporción de ejemplos correctamente clasificados en
     el conjunto de Validación.
 * Sea K el nodo cuya poda produce mejor rendimiento
 * Si este rendimiento es mejor que Medida, entonces
   Árbol = resultado de podar permanentemente Árbol en K
 * Si no, Continuar=Falso
5. Devolver Árbol (y su rendimiento sobre Prueba)
```

3. Aprendizaje de reglas

Partimos de reglas de clasificación del tipo:

- R1: Si CIELO=SOLEADO ∧ HUMEDAD=ALTA, entonces JUGAR_TENIS=-

Esto aporta ciertas ventajas:

- Es más claro y modular.
- Se puede expresar cualquier conjunto de instancias.
- Son métodos generalizables a lógica de primer orden.
- Más formal.
- Es más fácil traducir de árbol a reglas (pero no a la inversa).

El objetivo será aprender un conjunto de reglas consistente con los ejemplos dados. Una regla cubre un ejemplo siempre que el ejemplo satisfaga una seria de condiciones; lo hace correctamente si además el valor del atributo en la conclusión de la regla coincide con el valor que el ejemplo toma en ese atributo.

Algunos algoritmos empleados:

- ID3 (con traducción a reglas)
- Cobertura
- Algoritmos genéticos

Dado un conjunto de entrenamiento:

Ej.	Edad	Dignostico	Astigmatismo	Lagrima	Lente
E_1	Joven	Miope	-	Reducida	Ninguna
E_2	Joven	Miope	-	Normal	Blanda
E_3	Joven	Miope	+	Reducida	Ninguna
E_4	Joven	Miope	+	Normal	Rígida
E_5	Joven	Hipermétrope	-	Reducida	Ninguna
E_6	Joven	Hipermétrope	-	Normal	Blanda
E_7	Joven	Hipermétrope	+	Reducida	Ninguna
E_8	Joven	Hipermétrope	+	Normal	Rígida
E_9	PrePresbicia	Miope	-	Reducida	Ninguna
E_{10}	PrePresbicia	Miope	-	Normal	Blanda
E_{11}	PrePresbicia	Miope	+	Reducida	Ninguna
E_{12}	PrePresbicia	Miope	+	Normal	Rígida
E_{13}	PrePresbicia	Hipermétrope	-	Reducida	Ninguna
E_{14}	PrePresbicia	Hipermétrope	-	Normal	Blanda
E_{15}	PrePresbicia	Hipermétrope	+	Reducida	Ninguna
E_{16}	PrePresbicia	Hipermétrope	+	Normal	Ninguna
E_{17}	Presbicia	Miope	-	Reducida	Ninguna
E_{18}	Presbicia	Miope	-	Normal	Ninguna
E_{19}	Presbicia	Miope	+	Reducida	Ninguna
E_{20}	Presbicia	Miope	+	Normal	Rígida
E_{21}	Presbicia	Hipermétrope	-	Reducida	Ninguna
E_{22}	Presbicia	Hipermétrope	-	Normal	Blanda
E_{23}	Presbicia	Hipermétrope	+	Reducida	Ninguna
E_{24}	Presbicia	Hipermétrope	+	Normal	Ninguna

, una regla R2 sería → R2: Si ASTIGMATISMO=+ ∧ LAGRIMA=NORMAL entonces LENTE=RIGIDA

Aprender una regla para clasificar LENTE = RIGIDA sería busca aquellas condiciones que si se cumplen, entonces hacen que LENTE=RIGIDA:

Algoritmo de aprendizaje por cobertura

```
Aprendizaje-por-Cobertura(D,Atributo,v)
1. Hacer Reglas-aprendidas igual a vacío
2. Hacer E igual a D
3. Mientras E contenga ejemplos cuyo valor de Atributo es v, hacer:
3.1 Crear una regla R sin condiciones y conclusión Atributo=v
3.2 Mientras que haya en E ejemplos cubiertos por R incorrectamente
    y queden atributos que usar, hacer:
3.2.1 Elegir la MEJOR condición A=w para añadir a R, donde A es
      un atributo que no aparece en R y w es un valor de los
      posibles que puede tomar A
3.2.2 Actualizar R añadiendo la condición A=w a R
3.3 Incluir R en  Reglas-aprendidas
3.4 Actualizar E quitando los ejemplos cubiertos por R
4. Devolver Reglas-Aprendidas
```

Consideraciones del algoritmo:

- Bucle externo:
  - Añade reglas, por cada una añadida cubre algunos ejemplos.
  - Elimina en cada iteración los ejemplos cubiertos por la regla añadida.
  - Se añaden reglas mientras queden ejemplos sin cubrir.
- Bucle interno:
  - Añade condiciones a la regla, por cada una excluye ejemplos cubiertos incorrectamente.
  - Se repite mientras haya ejemplos cubiertos incorrectamente.
- Algoritmo de Cobertura frente al ID3:
  - Aprende una regla cada vez mientras que ID3 trabaja simultáneamente.
  - ID3 hace elecciones de atributos mientras que Cobertura hace elecciones de parejas atributo/valor.
- Hay diferentes criterios para elegir la mejor condición en cada vuelta del bucle interno:
  - Se añade la condición que produzca la regla con mayor frecuencia relativa.
  - Se añade la que produzca mayor ganancia de información:
    - $$p \cdot \left(log_2 \frac{p'}{t'} - log_2 \frac{p}{t} \right)$$
      - Dónde p'/t' es la frecuencia relativa después de añadir la condición y p/t es la frecuencia relativa antes de añadir la condición.
- Peligro de sobreajuste: si las reglas aprendidas por el algoritmo se ajustan *perfectamente* al conjunto de entrenamiento:
  - Early stopping (no generar todas las condiciones).
  - Podado de las reglas a posteriori (eliminar progresivamente condiciones hasta que no se produzcan mejoras).

4. Aprendizaje basado en instancias: kNN

Clasificación mediante vecino más cercano:

- Técnica alternativa a construir el modelo probabilístico, consistente en calcular la clasificación directamente a partir de los ejemplos.
- La idea es obtener la clasificación de un nuevo ejemplo a partir de las categorías de los ejemplos más cercanos, para ello se introducen algunos conceptos:
    - Noción de "distancia" entre ejemplos.
    - Normalmente hablamos de elementos en R^n y la distancia será euclídea.

El algoritmo k-NN o kNN ("k nearest neighbors"):

- Dado un conjunto de entrenamiento (vectores numéricos con una categoría) y un ejemplo nuevo, devolver la categoría mayoritaria en los *k* ejemplos del conjunto de entrenamiento más cercanos al ejemplo que se quiere clasificar.

Distancias en k-NN:

- Euclídea: $d_e(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$
- Manhattan: $d_m(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{n} |x_i - y_i|$
- Hamming: el nº de componentes en las que se difiere.

Normalmente se emplea la Euclídea cuando cada dimensión mide propiedades similares y la Manhattan en caso contrario La distancia Hamming se puede usar aun cuando los vectores n son siquiera numéricos.

Normalización: cuando no todas las dimensiones son del mismo orden de magnitud, se normalizan las componentes restando la media y dividiendo por la desviación típica.

¿Cómo se elige *k*?

- Generalmente basándonos en algún conocimiento específico sobre el problema de clasificación o cómo resultado de pruebas en conjuntos más pequeños (conjuntos de validación).
- Si la clasificación es binaria, preferiblemente impar, para intentar evitar empates, k=5 por ejemplo.

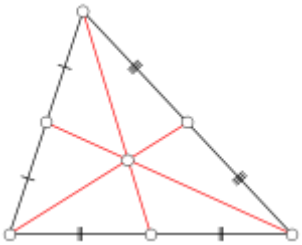
Variantes del algoritmo k-NN:

- Algoritmo k-NN con pesos:
  - Esta variante considera los *k* vecinos más cercanos {a1, … ,ak} al objeto *x* que queremos clasificar.
  - A cada uno de los *k* vecinos más cercanos, se les asigna un peso *Wi*:

$$w_i = \frac{1}{dist(a_i, x)}$$

  - Sumamos los pesos de cada una de las posibles clasificaciones; el valor asignado a *x* será el que obtenga mayor peso.

- Algoritmo NCC (Nearest Centroid Classifier):
  - Dado un conjunto de entrenamiento formado por puntos de R^n junto con su clasificación y un nuevo punto *x*, NCC asigna al nuevo punto la clasificación de la clase cuyo centroide este más cercano al punto.
  - Para cada valor de clasificación calculamos el centroide de los puntos asociados y, luego, aplicamos k-NN sobre el conjunto con k=1.
  - El centroide (o baricentro) es la intersección de todos los hiperplanos que dividen a un objeto *X* en dos partes iguales respecto al hiperplano.



- Algoritmo k-NN con rechazo:
  - Además del conjunto de entrenamiento **D**, el valor **k** y el objeto a clasificar **x**, necesitamos un umbral **μ**.
  - El algoritmo toma los ***k*** puntos del conjunto de entrenamiento más cercanos a **x** y devuelve el valor de clasificación mayoritario en esos **k** puntos sólo si el nº de puntos con esa clasificación supera el umbral **μ**.

5. Aprendizaje de modelos probabilísticos: Naive Bayes

Nos encontramos todavía en aprendizaje supervisado para clasificación, esto es:

- Tenemos un conjunto de datos de entrenamiento ya clasificados.
- Queremos generalizar y ser capaces de clasificar nuevos datos.

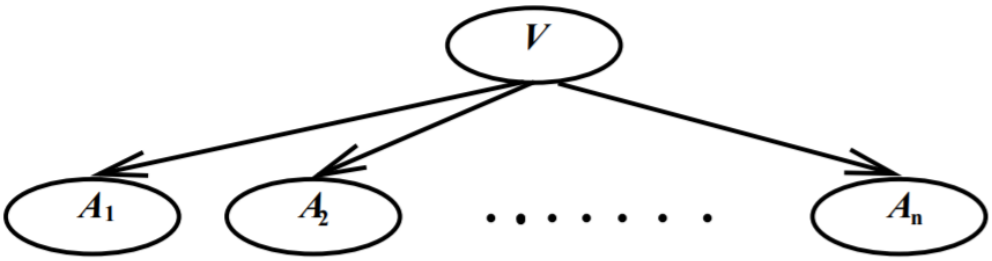
El resultado del aprendizaje será una distribución de probabilidad, en concreto, una red bayesiana muy simple. Con esta red podemos clasificar nuevos ejemplos calculando la probabilidad de pertenecer a cada clase.

Clasificadores *Naive Bayes*:

- Suponemos un conjunto de atributos A1, … , An cuyos valores determinan un valor en un conjunto finito V de posibles clases.
- Tenemos un conjunto de entrenamiento D con una serie de tuplas de valores concretos para los atributos, junto con su clasificación.
- Queremos aprender un clasificador tal que clasifique nuevas instancias {a1, … ,an}.

Suposición *Naive*:

- En el modelo probabilístico, tanto los atributos como el valor de clasificación se consideran variables aleatorias.
- Esto es un modelo naive: asumimos que los atributos son condicionalmente independientes entre sí, dado el valor de la clasificación. Esto implica que toda la distribución puede representarse cómo una red bayesiana de este tipo:



Para el proceso de aprendizaje, sólo hay que estimar las probabilidades **P(vj)** (probabilidades a priori) y **P(ai|vj)** (probabilidades condicionadas). Mediante el cálculo de sus frecuencias en el conjunto de entrenamiento, se obtienen las estimaciones de máxima verosimilitud:

$$P(v_j) = \frac{n(V = v_j)}{N} \qquad P(a_i|v_j) = \frac{n(A_i = a_i, V = v_j)}{n(V = v_j)}$$

dónde **N** es el nº total de ejemplos, **n(V=vj)** es el nº de ejemplos clasificados cómo **vj** y **n(Ai=ai, V=vj)** es el n º de ejemplos clasificados cómo **vj** cuyo valor en el atributo **Ai** es **ai**.

Ejemplo: dado el siguiente conjunto de entrenamiento,

Ej.	Cielo	Temperatura	Humedad	Viento	JugarTenis
D_1	Soleado	Alta	Alta	Débil	-
D_2	Soleado	Alta	Alta	Fuerte	-
D_3	Nublado	Alta	Alta	Débil	+
D_4	Lluvia	Suave	Alta	Débil	+
D_5	Lluvia	Baja	Normal	Débil	+
D_6	Lluvia	Baja	Normal	Fuerte	-
D_7	Nublado	Baja	Normal	Fuerte	+
D_8	Soleado	Suave	Alta	Débil	-
D_9	Soleado	Baja	Normal	Débil	+
D_{10}	Lluvia	Suave	Normal	Débil	+
D_{11}	Soleado	Suave	Normal	Fuerte	+
D_{12}	Nublado	Suave	Alta	Fuerte	+
D_{13}	Nublado	Alta	Normal	Débil	+
D_{14}	Lluvia	Suave	Alta	Fuerte	-

¿se quiere predecir si un día soleado, de temperatura suave, humedad alta y fuerte viento es bueno o no para jugar al tenis. Según el clasificador *naive*

$$v_{NB} = \underset{v_j \in \{+,-\}}{argmax}\ P(v_j)P(soleado|v_j)P(suave|v_j)P(alta|v_j)P(fuerte|v_j)$$

, luego necesitamos estimar todas esas probabilidades, lo hacemos calculando las frecuencias en la tabla anterior:

$$p(+) = 9/14,\ p(-) = 5/14,\ p(soleado|+) = 2/9,$$
$$p(soleado|-) = 3/5,\ p(suave|+) = 4/9,\ p(suave|-) = 2/5,$$
$$p(alta|+) = 3/9,\ p(alta|-) = 4/5,\ p(fuerte|+) = 3/9 \text{ y}$$
$$p(fuerte|-) = 3/5$$

, de lo que obtenemos que las dos probabilidades a posteriori son:

$$P(+)P(soleado|+)P(suave|+)P(alta|+)P(fuerte|+) = 0{,}0070$$
$$P(-)P(soleado|-)P(suave|-)P(alta|-)P(fuerte|-) = 0{,}0411$$

, por tanto, el clasificador devuelve la clasificación con mayor probabilidad a posteriori, es este caso la respuesta es **–** (ósea, que NO es un buen día para jugar al tenis).

Detalles técnicos sobre las estimaciones:

- Log-probabilidades:
  - Existe la probabilidad de que las estimaciones sean muy bajas.
  - Si tenemos pocos ejemplos, de devolver estimaciones bajas, estas serían 0.
  - Se plantean entonces dos problemas:
    - La inexactitud en si misma de los resultados.
    - La clasificación calculada; si una es 0, anula a las demás.
  - Una primera mejora es usar logaritmos en las probabilidades:

$$v_{NB} = \underset{v_j \in V}{argmax}[log(P(v_j)) + \sum_i log(P(a_i|v_j))]$$

- Suavizado:
  - Problema de las estimaciones:
    - Probabilidades nulas o muy bajas como resultado de la ausencia de valores de atributos en algunas categorías en el conjunto de entrenamiento.
    - Sobreajuste.
  - Idea: suponemos que tenemos *m* ejemplos adicionales, cuyos valores se distribuyen teóricamente a priori de alguna manera.
  - Estimación suavizada:

$$\frac{n' + m \cdot p}{n + m}$$

, dónde **n'** y **n** son, respectivamente, el nº de ejemplos favorables y totales observados. **p** es una estimación a priori de la probabilidad a estimar y **m** es una constante (tamaño de muestreo equivalente) que indica el nº de ejemplos adicionales (ficticios).

- Suavizado aditivo (o de Laplace): un caso particular del anterior que se suele emplear para estimar las probabilidades condicionales de *Naive Bayes*,

$$P(a_i|v_j) = \frac{n(A_i = a_i, V = v_j) + k}{n(V = v_j) + k|A_i|}$$

, dónde **k** es un nº fijado y **|Ai|** es el nº de posibles valores del atributo **ai**. Usualmente k=1.

6. Clustering

La idea consite en dividir un conjunto de datos de entrada en subconjuntos (clusters) de tal manera que los elementos de cada subconjunto compartan cierto patrón o características a priori desconocidas. Es un caso de aprendizaje supervisado dónde no tenemos información sobre que cluster corresponde a cada dato.

Aplicaciones:

- Minería de datos
- Procesamiento de imágenes digitales
- Bioinformática

Tipos:

- Clustering de partición estricta
- Clustering jerárquico
- Clustering basado en densidad

Clustering de partición estricta

Dado un conjunto de ejemplos **D = {X1,…,Xj,…,Xn}** con **xj = {Xj1,…,Xjd} ∈ R^d**, el clustering de partición estricta (Hard partitional clustering) busca una partición de **D** en **K** clusters, **P = {C1,…,Ck}** con **K** <= **N** tal que:

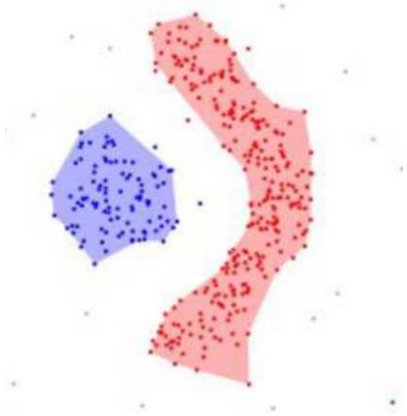
$$C_i \neq \emptyset \text{ para } i \in \{1, \ldots, K\}$$
$$\bigcup_{i=1}^{i=K} C_i = D$$
$$C_i \cap C_j = \emptyset \text{ para todo } i, j \in \{1, \ldots, K\} \text{ con } i \neq j.$$

Clustering jerárquico

Dado un conjunto de ejemplos **D = {X1,…,Xj,…,Xn}** con **xj = {Xj1,…,Xjd} ∈ R^d**, el clustering jerárquico (Hierarchical clustering) busca construir un conjunto de particiones anidadas de **D** con estructura de árbol, **H = {P1,…,Pq}** con **Q** <= **N** (dónde cada **Pj** es una partición) tal que si **Ci ∈ Pm** y **Cj ∈ Pi** con **m** > **l**, entonces **Ci ⊂ Cj** ó **Ci ∩ Cj = ∅**,, para todo **i**, **j**, **m**, **l ∈ {1,…,Q}**, **i != j**. Cada **Pi** es una partición de **D**.

Clustering basado en densidad

Basado en la densidad de puntos en distintas regiones. No necesitamos dar a priori el nº de clusters y estos pueden generarse de forma arbitraria.



Detalles de cada uno y algoritmos

El clustering de partición estricta divide el conjunto de datos en clusters que no tienen ninguna estructura interna.

Un algoritmo de clustering de partición estricta: k-medias:

Entrada: un número k de clusters, un conjunto de datos $\{x_i\}_{i=1}^{N}$ y una función de distancia

Salida: un conjunto de k centros $m_1, \ldots, m_k$

k-medias(k,datos,distancia)

```
1. Inicializar m_i (i=1,...,k) (aleatoriamente o con algún
   criterio heurístico)
2. REPETIR (hasta que los m_i no cambien):
   2.1 PARA j=1,...,N, HACER:
       Calcular el cluster correspondiente a x_j, escogiendo,
       de entre todos los m_i, el m_h tal que
       distancia(x_j,m_h) sea mínima
   2.2 PARA i=1,...,k HACER:
       Asignar a m_i la media aritmética de los datos
       asignados al cluster i-ésimo
3. Devolver m_1,...,m_n
```

Ejemplo de k-medias:

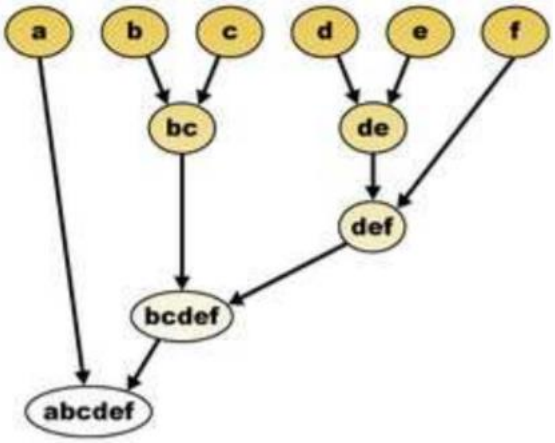
> Datos sobre pesos de la población: 51, 43, 62, 64, 45, 42, 46, 45, 45, 62, 47, 52, 64, 51, 65, 48, 49, 46, 64, 51, 52, 62, 49, 48, 62, 43, 40, 48, 64, 51, 63, 43, 65, 66, 65, 46, 39, 62, 64, 52, 63, 64, 48, 64, 48, 51, 48, 64, 42, 48, 41
>
> El algoritmo, aplicado con $k = 2$ y distancia euclídea, encuentra dos centros $m_1 = 63{,}63$ y $m_2 = 46{,}81$ en tres iteraciones
>
> 19 datos pertenecen al primer cluster y 32 al segundo cluster

El clustering jerárquico da estructura a los clusters. De forma recursiva, cada cluster está divido en clusters internos en una estructura anidada que va desde un cluster general hasta los individuos (clusters que contienen un único elemento).

Dependiendo del orden de la jerarquía, existen:

- Clustering jerárquico aglomerativo: partimos de clusters que contienen un único ejemplo y vamos agrupando los clusters generando grupos cada vez más grandes hasta obtener un cluster final que contiene a todos.
- Clustering jerárquico divisor: comenzamos con un cluster que contiene a todos y vamos realizando particiones hasta obtener clusters con un único ejemplo.

Con independencia de cuál se use, los resultados se representan en forma de árbol, llamado *dendrodrama*:



Un algoritmo genérico de clustering jerárquico aglomerativo sería:

1. Inicializamos el algoritmo con N clusters individuales. Calculamos la matriz de proximidad (basada en alguna definición de distancia) para los N clusters.
2. En la matriz de proximidad, buscamos la menor distancia entre clusters. Según definamos la *distancia entre clusters*, tendremos diferentes algoritmos. Combinamos en un único cluster aquellos que estén a distancia mínima.
3. Actualizamos la matriz de proximidad considerando los nuevos clusters.
4. Repetimos los pasos 2 y 3 hasta que quede un único cluster.

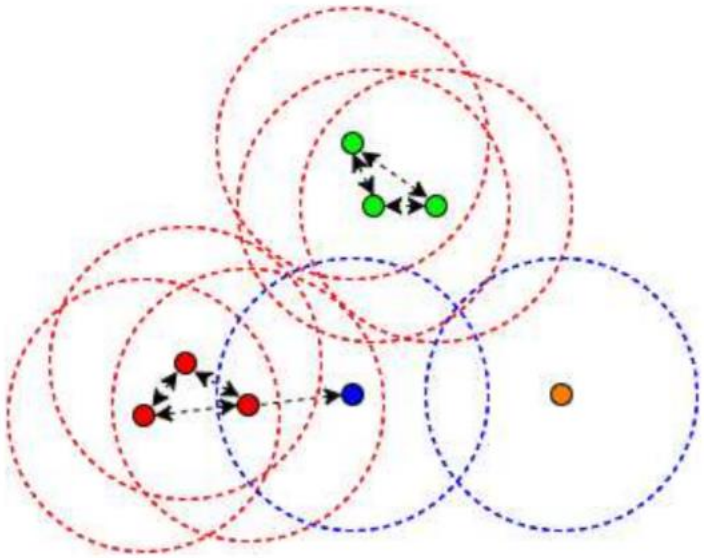
Un algoritmo de clustering basado en densidad: DBSCAN (Density Based Spatial Clustering of Applications with Noise):

- Necesitamos dos parámetros:
  - **minPts** que es un entero positivo que representa el nº mínimo de puntos.
  - **Ϙ** que es un real positivo (distancia).
- Clasificamos los puntos cómo:
  - Un punto **p** es *núcleo* si hay al menos **minPts** puntos a distancia menor que **ϙ** de **p**.
  - Un punto **q** es *frontera* si hay menos de **minPts** puntos a distancia menor que **ϙ** de **q**, pero hay al menos un punto **p** a distancia menor que **ϙ** de **q** cumpliendo que **p** es núcleo.
  - Un punto **r** es *ruido* si no es núcleo ni frontera.
- Se dice que **q** es *directamente alcanzable* desde **p** si **p** es núcleo y la distancia entre **p** y **q** es menor que **ϙ**.
- Se dice que un punto **p** es *alcanzable* desde **p** si existe una secuencia de puntos **p1,…,pn** donde **p1 = p** y **pn = q** tal que cada punto **pi + 1** es directamente alcanzable desde **pi**; es decir, todos los puntos de la secuencia deben ser puntos núcleos (con la posible excepción de **q**).

Algoritmo básico de DBSCAN
Entrada: Nube de puntos D, ϵ, *minPts*

1. Determinar los puntos núcleo, frontera y ruido en función de ϵ y *minPts*.
2. Crear un grafo conectando dos puntos núcleo si están a distancia menor que ϵ.
3. Conectar los puntos frontera a uno de los puntos núcleo que estén a distancia menor que ϵ (no determinista)
4. Devolver los puntos de cada componente conexa como un cluster.

Posible salida del algoritmo DBSCAN con un minPts = 2:



, dónde se han obtenido dos clusters; uno con tres puntos y otro con cuatro. Los puntos rojos y verdes son núcleos, el punto azul es frontera y el naranja es ruido.