Metaheurísticas

Seminario 2. Problemas de optimización con técnicas basadas en búsqueda local

1. Problema de Asignación Cuadrática (QAP)
   - Definición del Problema
   - Representación y Ejemplo: Diseño de un Hospital. Solución Greedy
   - Búsquedas por Trayectorias Simples
   - La Biblioteca QAPLIB
2. Problema del Aprendizaje de Pesos en Características (APC)

Definición del Problema

- El Problema de Asignación Cuadrática (QAP) está considerado como uno de los problemas de optimización combinatoria más complejos

- Es NP-completo. Incluso la resolución de problemas pequeños de tamaño $n>25$ se considera una tarea computacionalmente muy costosa

- El problema general consiste en encontrar la asignación óptima de n unidades a n localizaciones, conociendo la distancia entre las primeras y el flujo existente entre las segundas

Definición del Problema

- Sean n unidades (u_i, i=1,...,n) y n localizaciones (l_j, j=1,...,n). Se dispone de dos matrices $F=(f_{ij})$ y $D=(d_{ij})$, de dimensión (n×n) con la siguiente interpretación:
  - F es la matriz de flujo, es decir, f_{ij} es el flujo que circula entre la unidad i y la j
  - D es la matriz de distancias, es decir, d_{kl} es la distancia existente entre la localización k y la l

- El costo de asignar simultáneamente u_i a l_k y u_j a l_l es:

$$f_{ij} \cdot d_{kl}$$

Definición del Problema

- La definición matemática del problema consiste en minimizar el costo de las asignaciones:

$$\min \quad \sum_{i,j=1}^{n} \sum_{k,p=1}^{n} f_{ij} d_{kp} x_{ij} x_{kp}$$

$$\text{s.a.} \quad \sum_{i=1}^{n} x_{ij} = 1 \qquad 1 \le j \le n \,,$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad 1 \le i \le n \,,$$

$$x_{ij} \in \{0,1\} \qquad 1 \le i, j \le n \,.$$

- Como se puede ver, la función de coste es cuadrática, lo que da nombre al problema y lo complica sustancialmente

Definición del Problema

- **Problema de la asignación cuadrática, *QAP***:

  Dadas n unidades y n localizaciones posibles, el problema consiste en determinar la asignación óptima de las unidades en las localizaciones conociendo el flujo existente entre las primeras y la distancia entre las segundas

- Si se considera una permutación para representar las asignaciones, se verifican directamente las restricciones del problema:

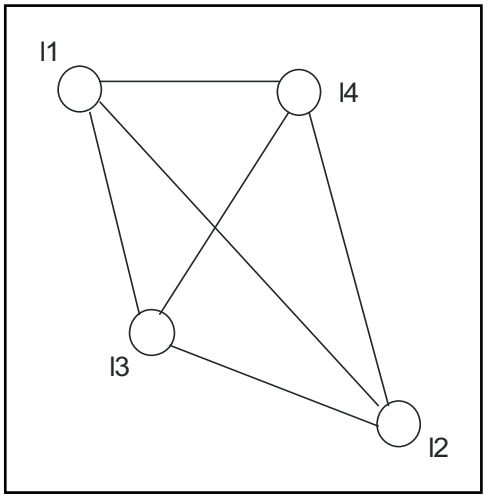
$$QAP = \min_{S \in \Pi_N} \left(\sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij} \cdot d_{S(i)S(j)} \right)$$

Representación y Ejemplo: Diseño de un Hospital

Elshafei, A.N. (1977). Hospital Layout as a Quadratic Assignment Problem. Operations Research Quarterly 28, 167-179.

- Supongamos que se ha de diseñar un hospital que comprende cuatro unidades distintas:
  - u1: Maternidad
  - u2: Urgencias
  - u3: Unidad de Cuidados Intensivos
  - u4: Cirugía

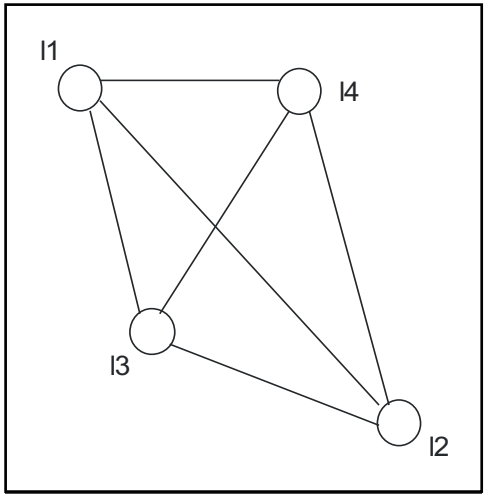
Que han de ser situadas en un edificio con la siguiente distribución: →



Representación y Ejemplo: Diseño de un Hospital

- La matriz *D* contiene las distancias existentes entre las diferentes salas
- La matriz *F* recoge el número medio de pacientes que pasan de una unidad a otra cada hora (por ejemplo, podrían ser las medias mensuales, medias anuales, totales anuales, …)

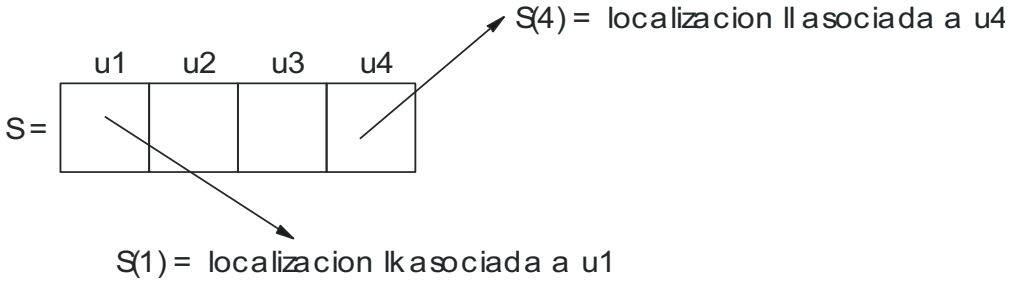
$$D = \begin{pmatrix} 0 & 12 & 6 & 4 \\ 12 & 0 & 6 & 8 \\ 6 & 6 & 0 & 7 \\ 4 & 8 & 7 & 0 \end{pmatrix} \qquad F = \begin{pmatrix} 0 & 3 & 8 & 3 \\ 3 & 0 & 2 & 4 \\ 8 & 2 & 0 & 5 \\ 3 & 4 & 5 & 0 \end{pmatrix}$$



u_1: Maternidad. u_2: Urgencias. u_3: Unidad de Cuidados Intensivos. u_4: Cirugía.

Representación y Ejemplo: Diseño de un Hospital

- Las soluciones son permutaciones del conjunto $N=\{1, 2, 3, 4\}$
- Para entenderlo mejor, podemos pensar en su representación en forma de vector permutación: *las posiciones se corresponden con las unidades y el contenido de las mismas con las localizaciones (salas del hospital) en la que se sitúan las unidades correspondientes*:

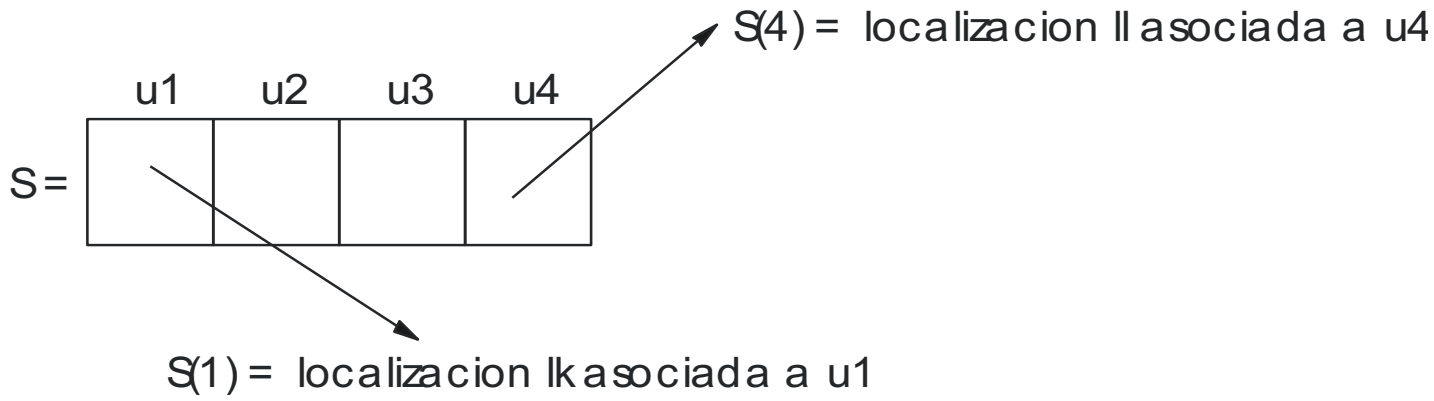


- En este caso, usamos una permutación para representar una asignación, al contrario que en el TSP en el que representa un orden
- Esto nos permite verificar de forma sencilla las restricciones del problema, lo que sería más complicado en otras representaciones como una matriz binaria

Representación y Ejemplo: Diseño de un Hospital

- Así, la solución S={3,4,1,2} representa la siguiente distribución de asignaciones:

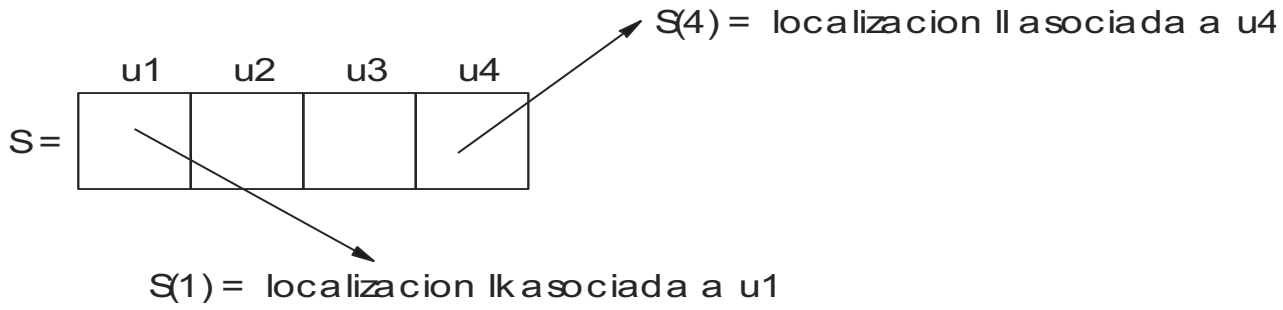
u1 ↔ l3 u2 ↔ l4 u3 ↔ l1 u4 ↔ l2



Representación y Ejemplo: Diseño de un Hospital

- cuyo costo $C(S)$ es:

$$
\begin{aligned}
& f_{12} \cdot d_{34} + f_{13} \cdot d_{31} + f_{14} \cdot d_{32} && 3 \cdot 7 + 8 \cdot 6 + 3 \cdot 6 + \\
+ & f_{21} \cdot d_{43} + f_{23} \cdot d_{41} + f_{24} \cdot d_{42} && 3 \cdot 7 + 2 \cdot 4 + 4 \cdot 8 + \\
+ & f_{31} \cdot d_{13} + f_{32} \cdot d_{14} + f_{34} \cdot d_{12} && 8 \cdot 6 + 2 \cdot 4 + 5 \cdot 12 + \\
+ & f_{41} \cdot d_{23} + f_{42} \cdot d_{24} + f_{43} \cdot d_{21} && 3 \cdot 6 + 4 \cdot 8 + 5 \cdot 12 = 374
\end{aligned}
$$



- Cada asignación unidad-localización influye globalmente en la red, es decir, en todas las transferencias que se efectúan desde la unidad en cuestión

Otras Aplicaciones

- Diseño óptimo de teclados para distintos idiomas en función de la frecuencia de pares de letras. Unid: letras; loc: teclas; flujo: frecuencia de pares de letras; dist: distancia entre las teclas en el teclado

  Burkard, R.E. and J. Offerman (1977). Entwurf von Schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. Z. Operations Research 21, B121-B123

- Cableado óptimo de placas madre: localización de componentes para reducir el cableado. Unid: componentes; loc: posición en la placa; flujo: nº de cables conectando componentes; dist: distancia entre locs.

  Brixius, N.W. and K.M. Anstreicher (2001). The Steinberg Wiring Problem. In: The Sharpest Cut, The Impact of M. Padberg and His Work, M. Grötschel, ed., SIAM, 2004, 293-307

- Asignación óptima de directores a oficinas: Unid: directores; loc: oficinas; flujo: frecuencia de interacciones entre directores; dist: distancia entre oficinas

  Hanan, M. and J.M. Kurtzberg (1972). A Review of the Placement and Quadratic Assignment Problems, SIAM Review 14, 324-342

Otras Aplicaciones

- Diseño de turbinas: Localización de las aspas de la turbina (con masa ligeramente distinta por la fabricación) de modo que el centro de gravedad coincida con el eje del motor

  J. Mosevich (1986). Balancing hydraulic turbine runners--A discrete combinatorial optimization problem, European Journal of Operational Research 26(2), 202-204

- Diseño óptimo de campus

  J.W. Dickey, J.W. Hopkins (1972). Campus building arrangement using TOPAZ, Transportation Research 6, 59–68

- Diseño de parques forestales

  Bos, J. (1993). A quadratic assignment problem solved by simulated annealing. Journal of Environmental Management, 37(2), 127-145

- Diseño de líneas de producción

  Geoffrion, A.M., and G.W. Graves (1976). Scheduling Parallel Production Lines with Changeover Costs: Practical Applications of a Quadratic Assignment/LP Approach. Operations Research 24, 595-610

Solución Greedy

- La complejidad del problema ha provocado que se hayan aplicado muchos algoritmos aproximados para su resolución

- Analizando la función objetivo podemos determinar que una buena fórmula heurística para resolver el problema es:

Asociar unidades de gran flujo con localizaciones céntricas en la red y viceversa

Solución Greedy

- Podemos construir un algoritmo greedy usando esta heurística mediante dos vectores, el potencial de flujo y el de distancia:

$$\hat{f}_i = \sum_{j=1}^{n} f_{ij} \; ; \qquad i = 1, \ldots, n$$

$$\hat{d}_k = \sum_{l=1}^{n} d_{kl} \; ; \qquad k = 1, \ldots, n$$

- Cuanto mayor sea $\hat{f}_i$, más importante es la unidad en el intercambio de flujos y cuanto menor sea $\hat{d}_k$, más céntrica es la localización. Por tanto:

el algoritmo irá seleccionando la unidad i libre con mayor $\hat{f}_i$ y le asignará la localización k libre con menor $\hat{d}_k$

Solución Greedy

Algoritmo Greedy-QAP

1. Calcular los potenciales $\hat{f}_i$ y $\hat{d}_k$.
2. S $\leftarrow \varnothing$.
3. Repetir para x =1 hasta n (asignaciones *1* a n):

   3.1. Escoger la unidad u_i no asignada aún ($u_i \notin S$) con mayor valor de $\hat{f}_i$.

   3.2. Escoger la localización l_k no asignada aún ($l_k \notin S$) con menor valor de $\hat{d}_k$.

   3.3. $a_x = (u_i, l_k)$. $S \leftarrow S \cup a_x$.

4. Calcular el costo de S, $C(S)$. Devolver S y $C(S)$.

Solución Greedy

- Si aplicamos el algoritmo greedy propuesto al ejemplo del hospital obtenemos los siguientes resultados:

$$\hat{f} = \begin{pmatrix} 14 \\ 9 \\ 15 \\ 12 \end{pmatrix} \qquad \hat{d} = \begin{pmatrix} 22 \\ 26 \\ 17 \\ 19 \end{pmatrix}$$

$$S = \{(u_3,l_3), (u_1,l_4), (u_4,l_1), (u_2,l_2)\}$$

$$\begin{aligned} C(S) = {} & f_{12} \cdot d_{42} + f_{13} \cdot d_{43} + f_{14} \cdot d_{41} && 3 \cdot 8 + 8 \cdot 7 + 3 \cdot 4 + \\ & + f_{21} \cdot d_{24} + f_{23} \cdot d_{23} + f_{24} \cdot d_{21} && 3 \cdot 8 + 2 \cdot 6 + 4 \cdot 12 + \\ & + f_{31} \cdot d_{34} + f_{32} \cdot d_{32} + f_{34} \cdot d_{31} && 8 \cdot 7 + 2 \cdot 6 + 5 \cdot 6 + \\ & + f_{41} \cdot d_{14} + f_{42} \cdot d_{12} + f_{43} \cdot d_{13} && = \quad 3 \cdot 4 + 4 \cdot 12 + 5 \cdot 6 = 364 \end{aligned}$$

Búsquedas por Trayectorias Simples

- **Representación**: Problema de asignación: una permutación $\pi=[\pi(1), \ldots, \pi(n)]$ en el que las posiciones del vector $i=1,\ldots,n$ representan las unidades y los valores $\pi(1), \ldots, \pi(n)$ contenidos en ellas las localizaciones. Permite verificar las restricciones

- **Operador de vecino de intercambio y su entorno**: El entorno de una solución π está formado por las soluciones accesibles desde ella a través de un movimiento de intercambio

  Dada una solución (asignación de unidades a localizaciones) se escogen dos unidades distintas y se intercambia la localización asignada a cada una de ellas ($Int(\pi,i,j)$):

$$\pi=[\pi(1), \ldots, \pi(i), \ldots, \pi(j), \ldots, \pi(n)]$$

$$\pi'=[\pi(1), \ldots, \pi(j), \ldots, \pi(i), \ldots, \pi(n)]$$

Búsquedas por Trayectorias Simples

- *Int*(π,*i*,*j*) verifica las restricciones, si la solución original π es factible siempre genera una solución vecina π' factible

- Su aplicación provoca que el tamaño del entorno sea:

$$|E(\pi)| = \frac{n \cdot (n-1)}{2}$$

- Las instancias del QAP no suelen ser demasiado grandes y el cálculo factorizado del coste de una solución se realiza de forma eficiente ($O(n)$), permitiendo explorar el entorno completo

  Aún así, dicha exploración requería $O(n^3)$ por lo que es recomendable utilizar una estrategia avanzada, considerando una modalidad de lista de candidatos y seleccionado primero los movimientos más prometedores

Búsqueda Local para el QAP

Stützle, Iterated local search for the quadratic assignment problem, European Journal of Operational Research 174 (2006) 1519–1539

- Algoritmo de búsqueda local del primer mejor: en cuanto se genera una solución vecina que mejora a la actual, se aplica el movimiento y se pasa a la siguiente iteración
  - Se detiene la búsqueda cuando se ha explorado el vecindario completo sin obtener mejora
- Se considera una factorización para calcular el coste de π' a partir del de π considerando sólo los cambios realizados por el movimiento de intercambio
- Se usa la técnica *don't look bits* para la lista de candidatos
  - Se emplea un vector binario de tamaño *n* que asocia un bit a cada unidad
  - Si dicho bit está activado en la iteración actual, no se considera ningún movimiento "que arranque" de la unidad en cuestión

BL-QAP: Factorización del Movimiento de Intercambio

- Sea $C(\pi)$ el coste de la solución original π. Para generar π', el operador de vecino $Int(\pi,r,s)$ escoge dos unidades r y s e intercambia sus localizaciones $\pi(r)$ y $\pi(s)$: $\pi=[\pi(1), \ldots, \pi(r), \ldots, \pi(s), \ldots, \pi(n)]$

  $t \leftarrow \pi(r)$; $\pi(r) \leftarrow \pi(s)$; $\pi(s) \leftarrow t$ $\Rightarrow$ $\pi'=[\pi(1), \ldots, \pi(s), \ldots, \pi(r), \ldots, \pi(n)]$

- Por lo tanto, quedan afectados 2·n sumandos, los relacionados con las dos ***viejas*** y las dos ***nuevas*** localizaciones de las dos unidades alteradas

- El coste del movimiento (la diferencia de costes entre las dos soluciones) $\Delta C(\pi,r,s)=C(\pi')-C(\pi)$ se puede factorizar como:

$$\sum_{k=1,k\neq r,s}^{n} \left[\begin{array}{l} f_{rk}\cdot(d_{\pi(s)\pi(k)}-d_{\pi(r)\pi(k)})+f_{sk}\cdot(d_{\pi(r)\pi(k)}-d_{\pi(s)\pi(k)})+ \\ f_{kr}\cdot(d_{\pi(k)\pi(s)}-d_{\pi(k)\pi(r)})+f_{ks}\cdot(d_{\pi(k)\pi(r)}-d_{\pi(k)\pi(s)}) \end{array} \right]$$

nuevas ***viejas***

BL-QAP: Factorización del Movimiento de Intercambio

- Si $\Delta C(\pi,r,s)$ es negativo ($\Delta C(\pi,r,s)<0$), la solución vecina π' es mejor que la actual π (el QAP es un problema de minimización) y se acepta. Si no, se descarta y se genera otro vecino

- El pseudocódigo de la BL del Primer Mejor del Tema 2 de Teoría quedaría:

Repetir

$\pi' \leftarrow$ GENERA_VECINO(π_{act});

Hasta ($\Delta C(\pi,r,s)<0$) **O**
(se ha generado E(π_{act}) al completo)

- El coste $C(\pi')$ de la nueva solución vecina es: $C(\pi')=C(\pi)+\Delta C(\pi)$. Sólo es necesario calcularlo para la solución vecina aceptada

BL-QAP: Definición de la Lista de Candidatos: *Don't Look Bits*

- Técnica que permite focalizar la BL en una zona del espacio de búsqueda en la que potencialmente puede ocurrir algo
- Reduce significativamente el tiempo de ejecución con una reducción muy pequeña de la eficacia de la BL
- Sólo es aplicable con la BL del primer mejor
- En la primera iteración, todos los bits están a 0, es decir, todos las unidades están activadas en un bucle externo y todos sus movimientos pueden ser considerados para explorar el entorno:
  - Si tras probar todos los movimientos asociados a esa unidad, ninguno provoca una mejora, se pone su bit a 1 para desactivarla en el futuro
  - Si una unidad está implicada en un movimiento que genera una solución vecina con mejor coste, se pone su bit a 0 para reactivarla

BL-QAP: Definición de la Lista de Candidatos: *Don't Look Bits*

```
procedure iterative improvement
    for i = 1 to n do
      if dlb[i] = 0 then
        improve_flag ← false
        for j = 1 to n do
            CheckMove(i, j)
            if move improves then
                ApplyMove(i, j); dlb[i] ← 0, dlb[j] ← 0
                improve_flag ← true
        endfor
        if improve_flag = false then dlb[i] ← 1
    end
end iterative improvement
```

IMPORTANTE: Este es el bucle interno de la BL, el de exploración del vecindario de la solución actual. Sea cual sea la BL usada, siempre habrá un bucle externo que repetirá el proceso mientras se produzca mejora

La Biblioteca QAPLIB

- La QAPLIB es una biblioteca que contiene distintas instancias del QAP llevando un registro de las mejores soluciones obtenidas hasta el momento para las mismas y de las cotas teóricas de la calidad de la mejor solución que se puede obtener

- Es accesible en la Web en las direcciones siguientes:

  http://www.opt.math.tu-graz.ac.at/qaplib (hasta Feb. 2002)

  http://www.seas.upenn.edu/qaplib/ (hasta la actualidad)

- En dicha dirección pueden encontrarse tanto los datos como las soluciones de distintas instancias del problema, relacionadas con diferentes aplicaciones

La Biblioteca QAPLIB

- El formato de los ficheros de datos es:

donde *n* es el tamaño del problema y *A* y *B* son, respectivamente las matrices de flujo y distancia, de acuerdo a nuestra codificación de las soluciones del problema

- El formato de los ficheros de soluciones (óptimas o mejores conocidas) es:

n *sol*
p

donde *n* es el tamaño del problema, *sol* es el coste de la solución y *p* es la permutación correspondiente

La Biblioteca QAPLIB

- La siguiente tabla es un ejemplo de la información que proporciona la QAPLIB:

	name	n	feas.sol.	permutation/bound	gap
	Tai12a	12	224416 (OPT)	(8,1,6,2,11,10,3,5,9,7,12,4)	
	Tai12b	12	39464925 (OPT)	(9,4,6,3,11,7,12,2,8,10,1,5)	
	Tai15a	15	388214 (OPT)	(5,10,4,13,2,9,1,11,12,14,7,15,3,8,6)	
	Tai15b	15	51765268 (OPT)	(1,9,4,6,8,15,7,11,3,5,2,14,13,12,10)	
	Tai17a	17	491812 (OPT)	(12,2,6,7,4,8,14,5,11,3,16,13,17,9,1,10,15)	
	Tai20a	20	703482 (OPT)	(10,9,12,20,19,3,14,6,17,11,5,7,15,16,18,2,4,8,13,1)	
*	Tai20b	20	122455319 (OPT)	(8,16,14,17,4,11,3,19,7,9,1,15,6,13,10,2,5,20,18,12)	
*	Tai25a	25	1167256 (OPT)	(9,4,6,11,5,1,15,10,14,3,17,12,19,18,23,8,21,2,22,7,16,20,24,25,13)	
*	Tai25b	25	344355646 (OPT)	(4,15,10,9,13,5,25,19,7,3,17,6,18,20,16,2,22,23,8,11,21,24,14,12,1)	
	Tai30a	30	1818146 (Ro-TS)	1706855 (L&P)	6.12 %
*	Tai30b	30	637117113 (OPT)	(4 8 11 15 17 20 21 5 14 30 2 13 6 29 10 26 27 24 28 22 12 9 7 23 19	
	Tai35a	35	2422002 (Ro-TS)	2216627 (L&P)	8,48 %

Metaheurísticas

Seminario 2. Problemas de optimización con técnicas basadas en búsqueda local

1. Problema de Asignación Cuadrática (QAP)

2. Problema del Aprendizaje de Pesos en Características (APC)
   - Definición del Problema de Clasificación. Clasificador k-NN.
   - Definición y Representación del Problema del Aprendizaje de Pesos en Características.
   - Solución Greedy.
   - Búsquedas por Trayectorias Simples.
   - Bases de Datos a Utilizar.

Definición del Problema de Clasificación

Disponemos de una muestra de objetos ya clasificados $w_i,\ldots,w_n$, representados en función de sus valores en una serie de atributos:

w_i tiene asociado el vector $(x_1(w_i),\ldots,x_n(w_i))$

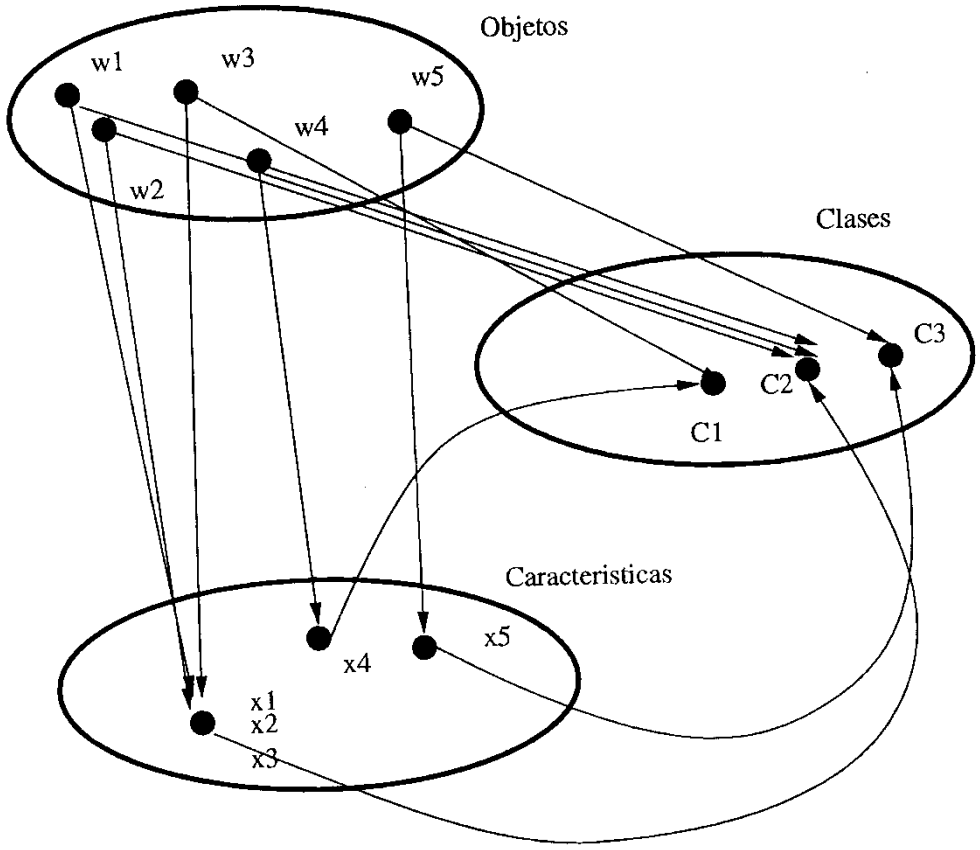
Cada objeto pertenece a una de las clases existentes $\{C_1,\ldots,C_M\}$

OBJETIVO: Obtener un sistema que permita clasificar dichos objetos de modo automático

$$\begin{array}{lcl} w_1 = (x_1(w_1),\ldots,x_n(w_1)) & \rightarrow & C_{i_1} \\ \vdots & & \vdots \\ w_k = (x_1(w_k),\ldots,x_n(w_k)) & \rightarrow & C_{i_k} \end{array} \qquad i_j \in \{1,\ldots,M\},\ j \in \{1,\ldots,k\}$$

Definición del Problema de Clasificación

El problema fundamental de la clasificación está directamente relacionado con la separabilidad de las clases



Definición del Problema de Clasificación

Concepto de aprendizaje supervisado en clasificación

- Se conocen las clases existentes en el problema
- Se conoce la clase concreta a la que pertenece cada objeto del conjunto de datos

Existen una gran cantidad de técnicas para el aprendizaje supervisado de Sistemas de Clasificación:

- Técnicas estadísticas: k vecinos más cercanos, discriminadores bayesianos, etc…
- Árboles de clasificación, Sistemas basados en reglas, Redes Neuronales, Máquinas de Soporte Vectorial, …

Definición del Problema de Clasificación

Ejemplo: *Diseño de un Clasificador para la flor del Iris*

- *Problema simple muy conocido: clasificación de lirios*
- *Tres clases de lirios: setosa, versicolor y virgínica*
- *Cuatro atributos: longitud y anchura de pétalo y sépalo, respectivamente*
- *150 ejemplos, 50 de cada clase*
- *Disponible en* http://www.ics.uci.edu/~mlearn/MLRepository.html

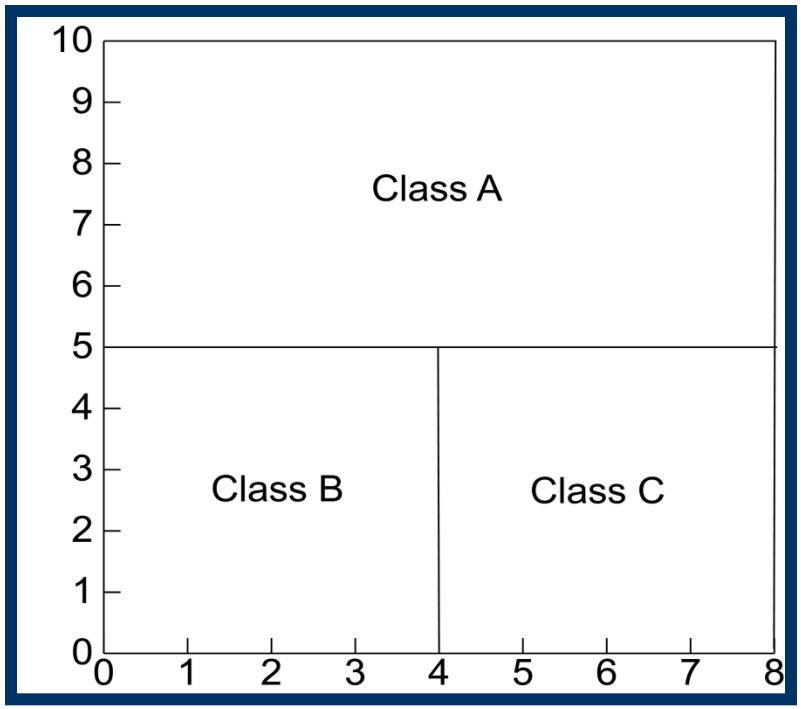
setosa

versicolor

virgínica

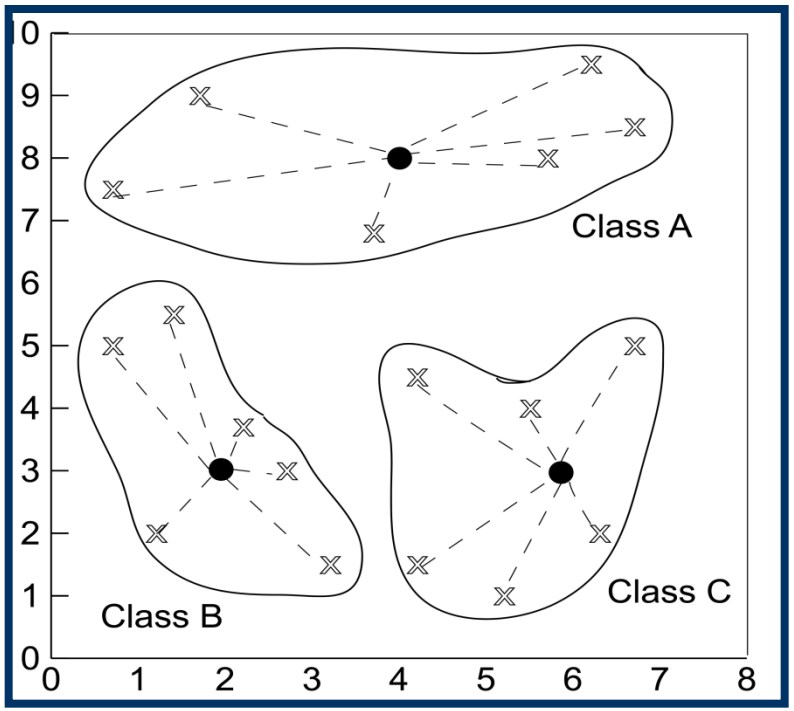
Definición del Problema de Clasificación

Ejemplos de clasificación sobre clases definidas: Basada en particiones y en distancias



Basado en Particiones

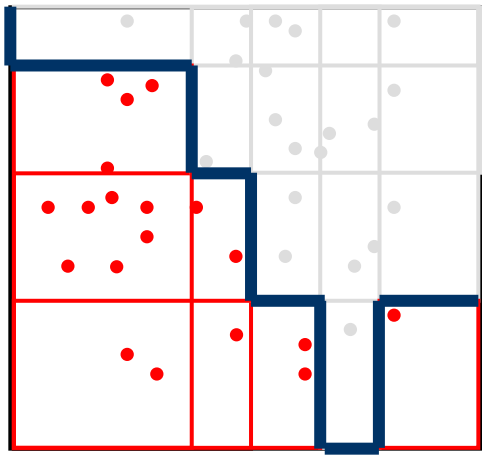
Basado en Distancias



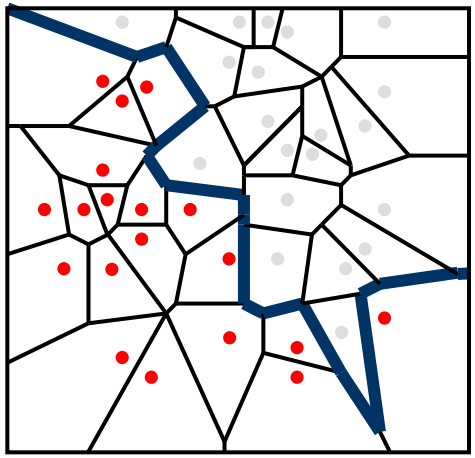
Definición del Problema de Clasificación

Ejemplos de clasificación sobre clases definidas

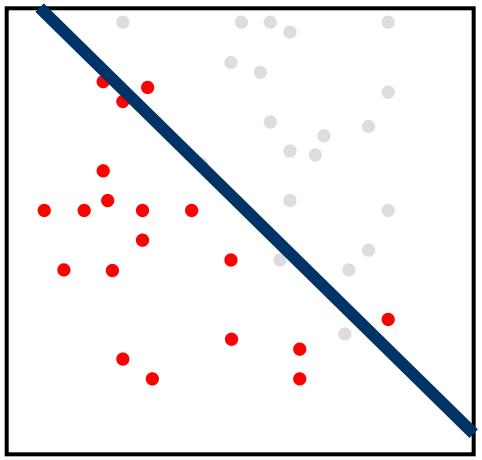
- ***Reglas intervalares***



- ***Basado en distancias***



- ***Clasificador lineal***



Definición del Problema de Clasificación

Para diseñar un clasificador, son necesarias dos tareas: **Aprendizaje y Validación**

El conjunto de ejemplos se divide en dos subconjuntos:

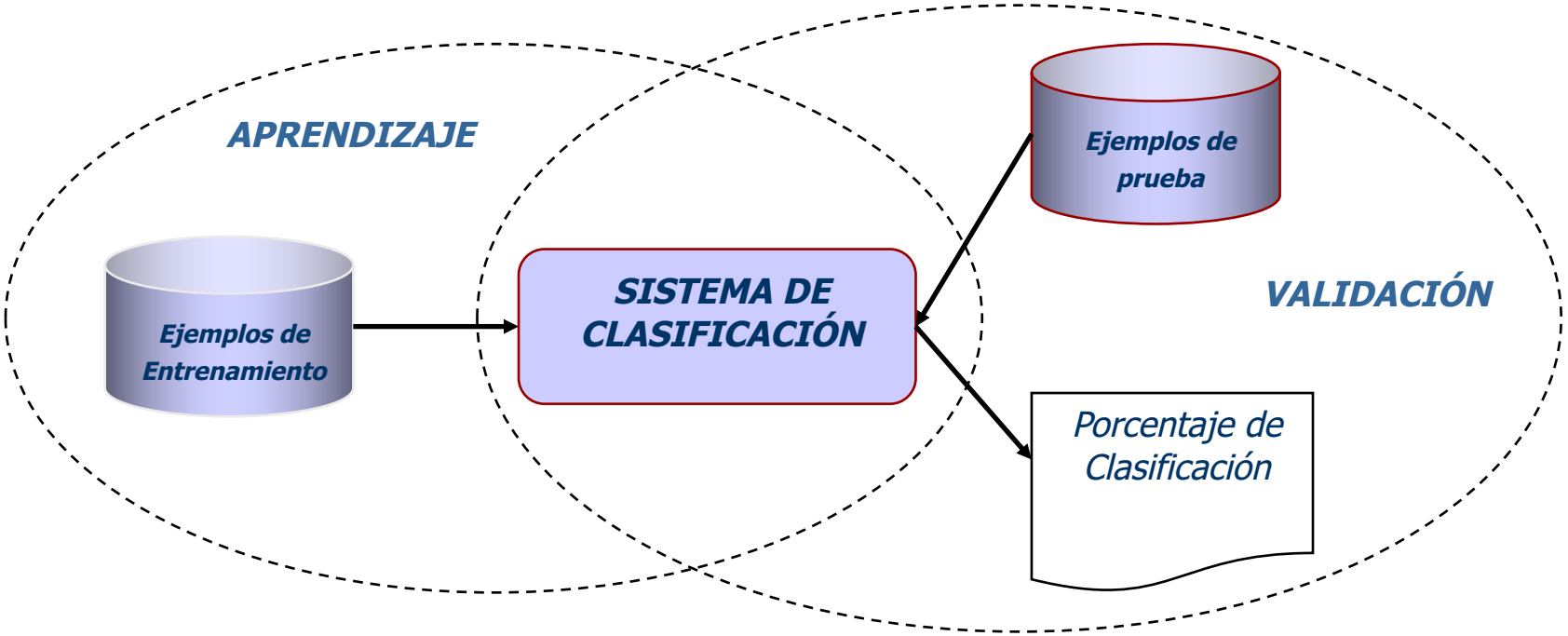
- **Entrenamiento**: Utilizado para aprender el clasificador
- **Prueba**: Se usa para validarlo. Se calcula el porcentaje de clasificación sobre los ejemplos de este conjunto (desconocidos en la tarea de aprendizaje) para conocer su poder de generalización

Para mayor seguridad, se suele hacer varias particiones entrenamiento-prueba

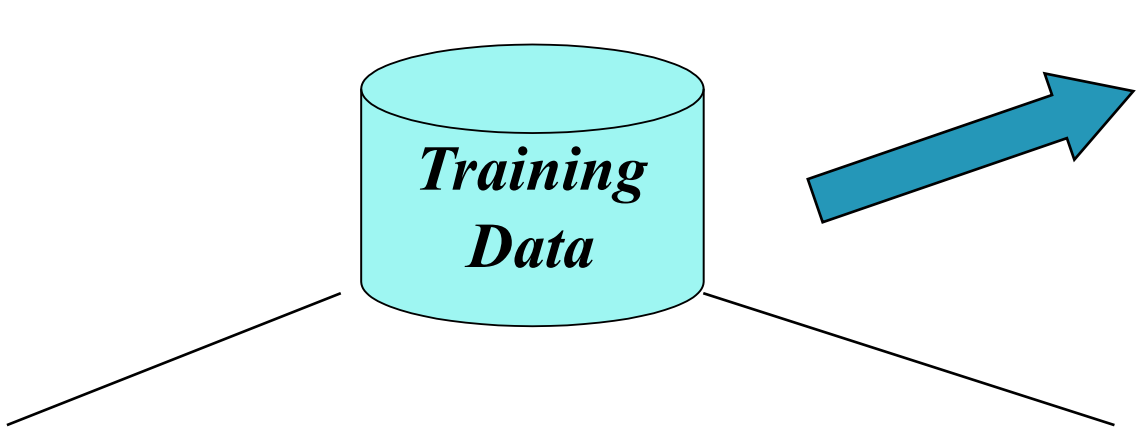
Para cada una, se diseña un clasificador distinto usando los ejemplos de entrenamiento y se valida con los de prueba

Definición del Problema de Clasificación

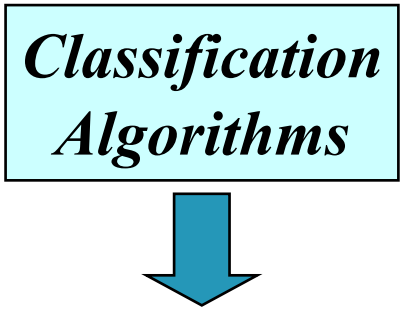
Esquema de aprendizaje en clasificación

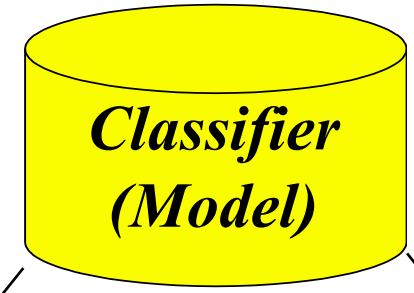


Definición del Problema de Clasificación



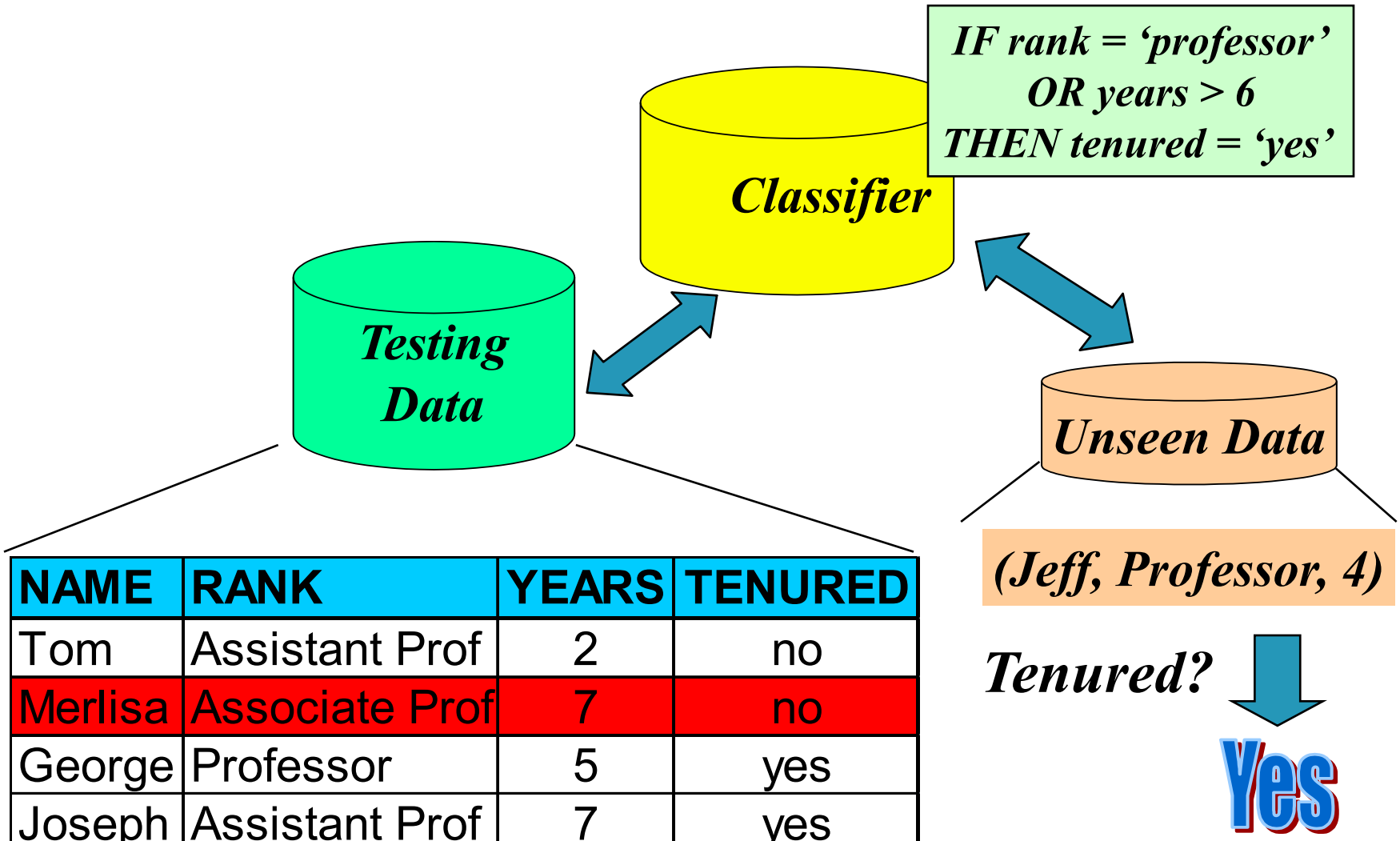
NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no





IF rank = 'professor'
OR years > 6
THEN tenured = 'yes'

Definición del Problema de Clasificación



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Definición del Problema de Clasificación

Nosotros aplicaremos la técnica de validación llamada ***5×2-cross validation*** (validación cruzada)

- Usaremos 5 particiones distintas de los datos al 50%
- Aprenderemos un clasificador con la mitad de los datos y lo validaremos con la otra mitad. Luego repetiremos el proceso a la inversa
- Por tanto, de cada partición obtendremos dos valores de porcentaje de clasificación en el conjunto de prueba
- La calidad del método de clasificación se medirá con un único valor, correspondiente a la media de los 10 porcentajes de clasificación (2 de cada partición)

Clasificador k-NN

El k-NN (k vecinos más cercanos) es uno de los clasificadores más utilizados por su simplicidad

i. El proceso de aprendizaje de este clasificador consiste en almacenar una tabla con los ejemplos disponibles, junto a la clase asociada a cada uno de ellos

ii. Dado un nuevo ejemplo a clasificar, se calcula su distancia (usaremos la Euclídea) a los *n* ejemplos existentes en la tabla y se escogen los *k* más cercanos

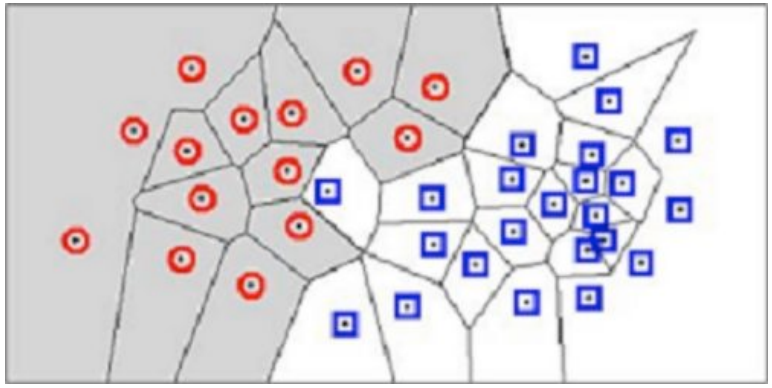
iii. El nuevo ejemplo se clasifica según la clase mayoritaria de esos *k* ejemplos más cercanos

iv. El caso más simple es cuando $k = 1$ (1-NN)

Clasificador k-NN

Regla del vecino más próximo o *Nearest neighbour* (1-NN)

- Si tenemos *m* ejemplos $\{e_1, \ldots, e_m\}$ en nuestro conjunto de datos, para clasificar un nuevo ejemplo e' se hará lo siguiente:



1. c_{min} = clase (e_1)
2. d_{min} = d(e_1, e')
3. Para $i=2$ hasta m hacer
   $d=d(e_i,e')$
   Si $(d<d_{min})$
   Entonces c_{min} = clase (e_i), $d_{min} = d$
4. Devolver c_{min} como clasificación de e'

- d(·,·) es una función de distancia
- En el caso de variables nominales o categóricas se utiliza la distancia de *Hamming:*

$$d_h(a,b) = \begin{cases} 0, & si\ a = b \\ 1, & si\ a \neq b \end{cases}$$

Clasificador k-NN

Distancias para las variables numéricas

- Las variables numéricas se suelen normalizar al intervalo [0,1]
- Si e_i^j es el valor de la variable j en e_i, es decir $e_i=(e_i^1,\ldots,e_i^n)$ entonces algunas de las distancias más utilizadas son
  - Euclídea $$d_e(e_1,e_2)=\sqrt{\sum_{i=1}^{n}(e_1^i-e_2^i)}$$
  - Manhattan: $$d_m(e_1,e_2)=\sum_{i=1}^{n}\left|e_1^i-e_2^i\right|$$
  - Minkowski $$d_m^k(e_1,e_2)=\left(\sum_{i=1}^{n}\left|e_1^i-e_2^i\right|^k\right)^{1/k}$$

Como se puede observar, $d_m^1=d_m$ y $d_m^2=d_e$

Clasificador k-NN

- Por tanto, la distancia entre dos ejemplos e_1 y e_2, utilizando p.e. d_e para las variables numéricas sería

$$d_e(e_1, e_2) = \sqrt{\sum_i (e_1^i - e_2^i)^2 + \sum_j d_h(e_1^j, e_2^j)}$$

siendo *i* el índice que se utiliza para recorrer las variables numéricas y *j* el índice que se utiliza para recorrer las variables nominales o categóricas.

- Nosotros usaremos la distancia Euclídea para variables numéricas y la distancia de Hamming para variables nominales o categóricas.

Clasificador k-NN

Dado el siguiente conjunto con 4 instancias, 3 atributos y 2 clases:

x_1: 0.4 0.8 0.2 positiva
x_2: 0.2 0.7 0.9 positiva
x_3: 0.9 0.8 0.9 negativa
x_4: 0.8 0.1 0.0 negativa

Queremos clasificar con 1-NN el ejemplo:
x_q: 0.7 0.2 0.1

Calculamos la distancia del ejemplo con todos los de la tabla:

$$d(x_1, x_q) = \sqrt{(0.4-0.7)^2 + (0.8-0.2)^2 + (0.2-0.1)^2} = 0.678$$

$$d(x_2, x_q) = \sqrt{(0.2-0.7)^2 + (0.7-0.2)^2 + (0.9-0.1)^2} = 1.068$$

$$d(x_3, x_q) = \sqrt{(0.9-0.7)^2 + (0.8-0.2)^2 + (0.9-0.1)^2} = 1.020$$

$$d(x_4, x_q) = \sqrt{(0.8-0.7)^2 + (0.1-0.2)^2 + (0.0-0.1)^2} = 0.173$$

Por tanto, el ejemplo se clasificará con respecto a la clase negativa

IMPORTANTE: Los atributos deben estar normalizados en [0,1] para no priorizar unos sobre otros

Clasificador k-NN

Para normalizar los datos, hay que saber el intervalo de dominio de cada uno de los atributos

Dado un valor x_j perteneciente al atributo j del ejemplo x y sabiendo que el dominio del atributo j es $[Min_j , Max_j]$, el valor normalizado de x_j es:

$$x_j^N = \frac{x_j - Min_j}{Max_j - Min_j}$$

Definición del Problema del Aprendizaje de Pesos en Características

- Un problema que optimiza el rendimiento del clasificador k-NN es el **Aprendizaje de Pesos en Características (APC)**.

- APC asigna valores reales a las características, de tal forma que se describe o pondera la relevancia de cada una de ellas al problema del aprendizaje.

- APC funciona mejor cuando se asocia a clasificadores sencillos, locales y muy sensibles a la calidad de los datos.

- Nosotros usaremos el clasificador 1-NN, por lo que vamos a considerar el vecino más cercano para predecir la clase de cada objeto.

Definición del Problema del Aprendizaje de Pesos en Características

Objetivo:

- Ajustar un conjunto de ponderaciones o pesos asociados al conjunto total de características, de tal forma que los clasificadores que se construyan a partir de él sean *mejores.*

- Existen distintos criterios para determinar cuándo el clasificador generado es mejor.

- Es un problema de **búsqueda con codificación real** en el espacio *n*-dimensional, para *n* características.

Definición del Problema del Aprendizaje de Pesos en Características

- La expresión anterior considera que todas las variables tienen igual importancia.

- El problema del Aprendizaje de Pesos en Características (APC) asigna pesos a los atributos de forma que se pondere su importancia dentro del contexto.

$$d_e(e_1, e_2) = \sqrt{\sum_i w_i \cdot (e_1^i - e_2^i)^2 + \sum_j w_j \cdot d_h(e_1^j, e_2^j)}$$

- Los pesos vienen dados por un vector *W*, tal que cada w_i ó w_j, representa un valor real en *[0, 1]*.

Definición del Problema del Aprendizaje de Pesos en Características

- Como *W* es independiente al tipo de característica asociada (numérica o nominal), utilizaremos a partir de ahora el índice *i* (w_i) indistintamente al tipo de característica.
- El tamaño de *W* será *n*, debido a que condiciona a todas las características del problema *n*-dimensional.
- *W* también puede verse con un punto *n*-dimensional en $\mathbb{R}$ acotado en *[0, 1]*.
- El objetivo consiste en encontrar el mejor *W* para el problema concreto y el clasificador 1-NN.

Nuestra función de evaluación:

- Rendimiento promedio de un clasificador 1-NN (considerando *k*=1 vecino y *leave one out*) aplicando validación sobre el conjunto *T* de datos: *tasa_clas.*

- *tasa_clas* mide el porcentaje de instancias correctamente clasificadas pertenecientes a *T*:

$$tasa\_clas = 100 \cdot \frac{\text{nº instancias bien clasificadas de } T}{\text{nº instancias en } T}$$

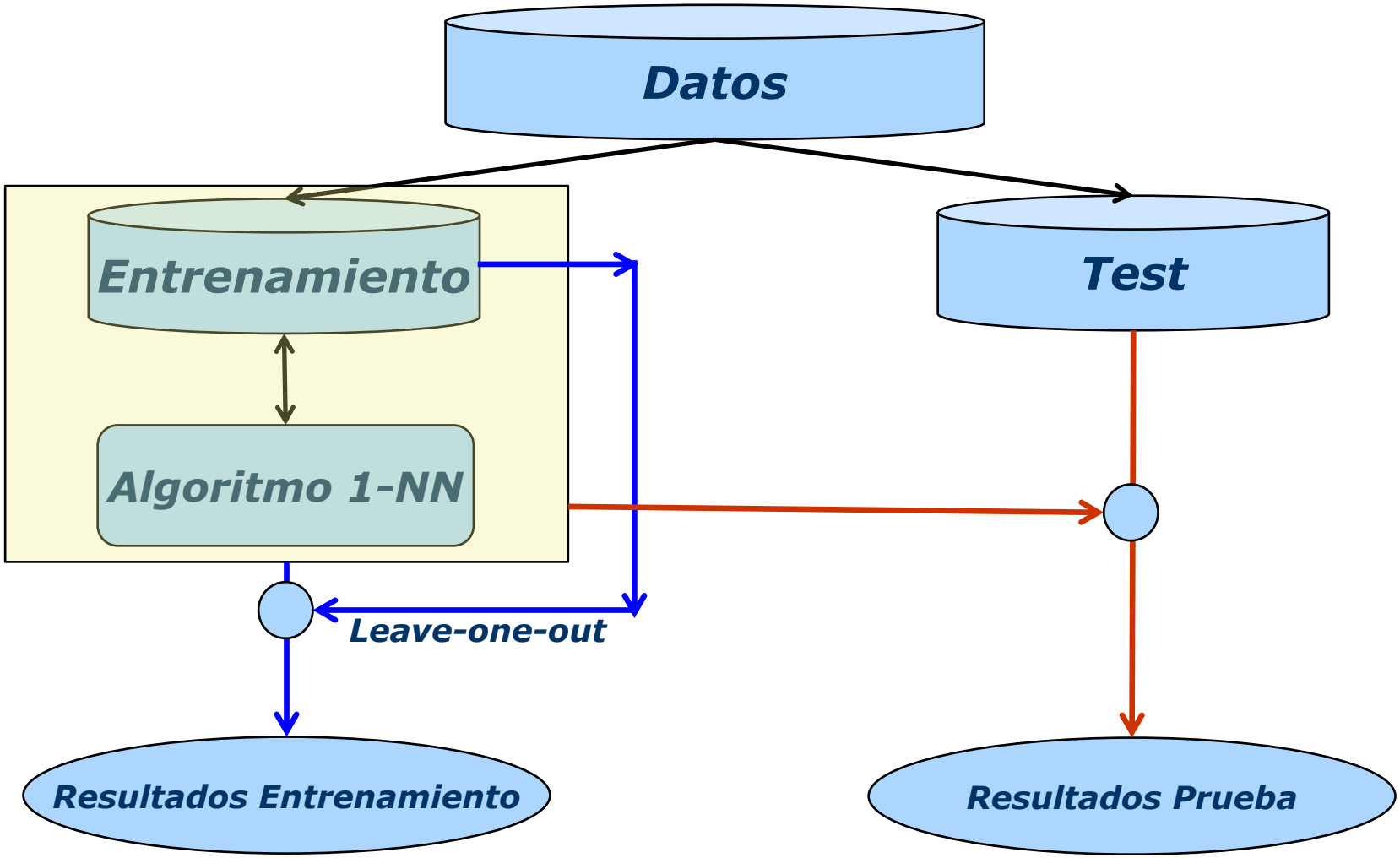
- El objetivo es obtener el mejor *W* que maximiza esta función.

Definición del Problema del Aprendizaje de Pesos en Características

Cálculo del Porcentaje de Entrenamiento en 1-NN:

- *En el algoritmo 1-NN, no es posible calcular el porcentaje de acierto sobre el conjunto de entrenamiento de un modo directo.*
- *Si intentásemos clasificar un ejemplo del conjunto de entrenamiento directamente con el clasificador 1-NN, el ejemplo más cercano sería siempre él mismo, con lo que se obtendría un 100% de acierto.*
- *Para remediar esto, se debe seguir el procedimiento denominado dejar uno fuera ("leave one out").*
- *Para clasificar cada ejemplo del conjunto de entrenamiento, se busca el ejemplo más cercano* ***sin considerar a él mismo****.*
- *Por lo demás, se opera igual: cada vez que la clase devuelta coincida con la clase real del ejemplo, se contabiliza un acierto.*
- *El porcentaje final de acierto es el número de aciertos entre el número total de ejemplos.*

Definición del Problema del Aprendizaje de Pesos en Características



Representación del Problema del Aprendizaje de Pesos en Características

Representación real: Vector real de tamaño *n*:

$W = (w_1, w_2, \ldots, w_n)$, donde $w_i \in [0, 1]$

w_1	w_2	.....	w_{n-1}	w_n

Un 1 en la posición w_i indica que la característica en cuestión se considera completamente en el cálculo de la distancia.

Un 0 en la posición w_i indica que la característica no se considera en el cálculo de la distancia.

Cualquier otro valor intermedio gradúa el peso asociado a cada característica y pondera su importancia en la clasificación final.

Métodos de búsqueda:

- Búsqueda secuencial
  - Método voraz (*greedy*) que parte de un vector de pesos inicializado a *0* que incrementa cada componente en función de la distancia al enemigo más cercano de cada ejemplo, y disminuye cada componente en función de la distancia al amigo más cercano de cada ejemplo.
- Búsqueda probabilística
  - Métodos MonteCarlo y Las Vegas
- Búsqueda con metaheurísticas

Solución *greedy*

Descripción método *RELIEF*:

- Parte de un vector de pesos *W* inicializado a *0*: $w_i = 0$.
- En cada paso se modifica *W* utilizando cada uno de los ejemplos del conjunto de entrenamiento.
- Para cada ejemplo *e* del conjunto de entrenamiento, se busca a su enemigo más cercano e_e (ejemplo más cercano con clase diferente) y a su amigo más cercano e_a (ejemplo más cercano de la misma clase sin considerar él mismo).
- *W* se actualiza con la distancia dimensión a dimensión entre *e* y e_e y entre *e* y e_a.
- Si $w_i < 0 \rightarrow w_i = 0$. Después, *W* se normaliza al intervalo *[0, 1]*.

Solución *greedy*

Algoritmo método RELIEF:

$W \leftarrow \{0, 0, \ldots, 0\}$

Para cada e_i *en* T

Buscar el enemigo más cercano de e_i*:* e_e

Buscar el amigo más cercano de e_i*:* e_a

$W = W + |e_i - e_e| - |e_i - e_a|$

w_m = *máximo (W)*

Para cada w_i *en* W

Si $w_i < 0$ *entonces*

$w_i = 0$

Si no

$w_i = w_i / w_m$

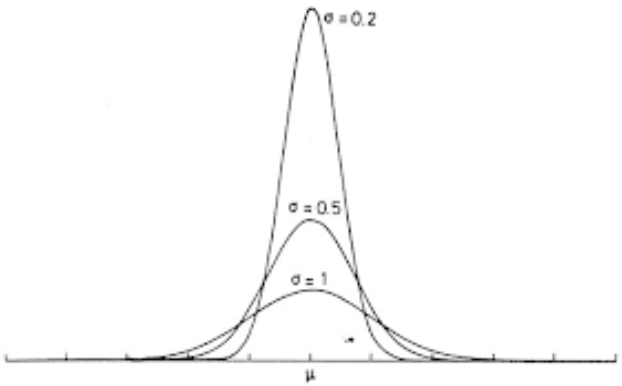
Devolver W

Búsquedas por Trayectorias Simples

- **Representación real**: Problema de codificación real: un vector real W=(w_1, …, w_n) en el que cada posición *i* representa una característica y su valor en *[0, 1]* indica el peso aprendido para cada característica.No tiene restricciones exceptuando el dominio [0, 1] para cada dimensión.
- **Operador de vecino por Mutación Normal**: El entorno de una solución *W* está formado por las soluciones accesibles desde ella a través de un movimiento proveniente de la mutación de una componente z_i, con un radio que depende de σ:

$$Mov(W, \sigma) = W' = (w_1, \ldots, w_i + z_i, \ldots, w_n)$$

$$z_i \sim N_i(0, \sigma^2)$$



Búsquedas por Trayectorias Simples

- *Mov(W,σ)* verifica las restricciones si después de aplicarlo, truncamos el w_i modificado a *[0,1]*. De esta manera, si la solución original *W* es factible siempre genera una solución vecina *W'* factible.

- El tamaño del entorno es infinito, debido a la naturaleza de problema de codificación real. Sin embargo, en cada iteración se mutará una componente $i \in \{1,\ldots,n\}$ distinta hasta completar todas las posibilidades. Una vez dada una vuelta, se vuelve a empezar en un orden aleatorio.

- El problema del APC **no permite realizar un cálculo factorizado del coste** de forma sencilla.

  Tampoco es fácil definir una preferencia entre las características para explorar el entorno. Podría considerarse alguna medida de cantidad de información para esta tarea.

Búsqueda Local para el APC

- Algoritmo de búsqueda local del primer mejor: en cuanto se genera una solución vecina que mejora a la actual, se aplica el movimiento y se pasa a la siguiente iteración.

- Se detiene la búsqueda cuando se ha explorado una parte del vecindario; es decir, cuando se genera un número máximo de vecinos.

- No se considera ningún tipo de factorización ni ningún mecanismo específico de exploración del entorno.

Bases de Datos a Utilizar

- En la actualidad, hay muchas bases de datos que se utilizan como bancos de prueba (*benchmarks*) para comprobar el rendimiento de los algoritmos de clasificación

- El UCI es un repositorio de bases de datos para aprendizaje automático muy conocido

- Está accesible en la Web en:

  http://www.ics.uci.edu/~mlearn/MLRepository.html

Bases de Datos a Utilizar

- A partir de este repositorio, se ha desarrollado un formato para definir todas las cualidades de una base de datos en un único fichero

- Se trata del formato ARFF, utilizado en WEKA (http://www.cs.waikato.ac.nz/ml/weka/)

- Un fichero ARFF contiene dos partes:

  - Datos de cabecera: Líneas que comienzan por @. Contienen información acerca de los atributos: significado y tipo

  - Datos del problema: Líneas de datos. Son los ejemplos en sí. Cada línea se corresponde con un ejemplo y los valores están separados por comas

Bases de Datos a Utilizar

Ejemplo fichero ARFF:

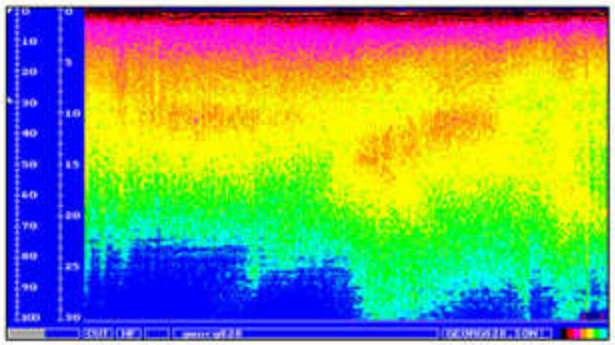
```
@relation iris
@attribute sepalLength real
@attribute sepalWidth real
@attribute petalLength real
@attribute petalWidth real
@attribute class {Iris-setosa, Iris-versicolor, Iris-virginica}
@data
5.1, 3.5, 1.4, 0.2, Iris-setosa
4.9, 3.0, 1.4, 0.2, Iris-setosa
7.0, 3.2, 4.7, 1.4, Iris-versicolor
6.0, 3.0, 4.8, 1.8, Iris-virginica
```

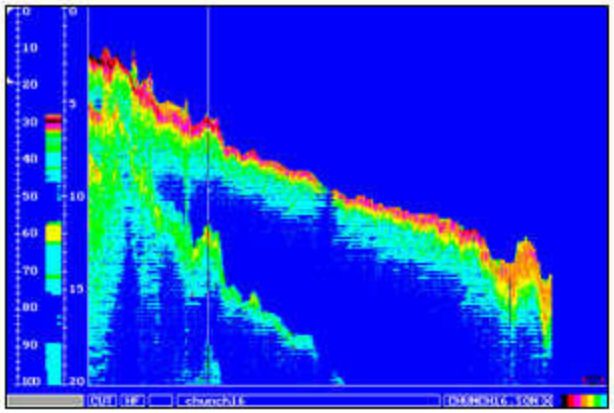
- 5 Atributos, los 4 primeros de tipo real y el último de tipo nominal
- La clase es el último atributo (atributo de salida) con los posibles valores definidos
- Los datos van a continuación de la directiva @data

Bases de Datos a Utilizar

Trabajaremos con tres bases de datos: Sonar, Wdbc y SpamBase.

Sonar es una base de datos de detección de materiales mediante señales de sónar, discriminando entre objetos metálicos y rocas



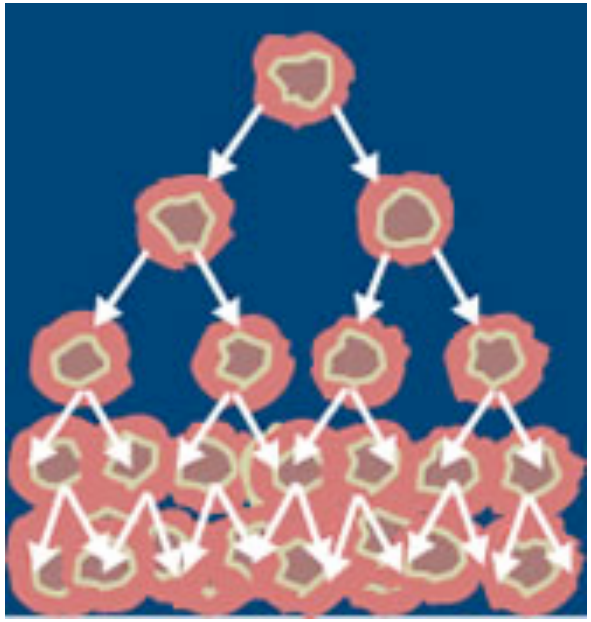


- Consta de 208 ejemplos
- Consta de 61 atributos (clase incluida)
- Consta de 2 clases (R: roca; M: metal)
- Atributos: Cada atributo representa la energía dentro de una banda de frecuencia en concreto. Están definidos en el rango [0.0,1.0]
- Hay 111 ejemplos referentes a señales obtenidas a partir de cilindros metálicos a partir de diversos ángulos y condiciones, y 97 referentes a rocas bajo condiciones similares

Bases de Datos a Utilizar

Trabajaremos con tres bases de datos: Sonar, Wdbc y SpamBase.

Wdbc es una base de datos contiene atributos calculados a partir de una imagen digitalizada de una aspiración con aguja fina de una masa en la mama. Se describen las características de los núcleos de las células presentes en la imagen. La tarea consiste en determinar si un tumor encontrado es benigno o maligno..



- Consta de 569 ejemplos
- Consta de 31 atributos (clase incluida)
- Consta de 2 clases (M = maligno, B = benigno)
- Atributos: Valores reales calculados de cada núcleo de la célula (10 valores * 3 células): radio, textura, perímetro, área, concavidad, simetría, …

Bases de Datos a Utilizar

Trabajaremos con tres bases de datos: Sonar, Wdbc y SpamBase.

SpamBase es una base de datos de detección de SPAM frente a correo electrónico seguro

- Consta de 460 ejemplos
- Consta de 58 atributos (clase incluida)
- Consta de 2 clases (1: SPAM, 0: correo seguro)
- Atributos:
  - 48 atributos en [0,100] sobre la frecuencia de una palabra concreta en el correo
  - 6 atributos en [0,100] sobre la frecuencia de un carácter
  - 1 atributo sobre el número de caracteres seguidos en mayúsculas, etc.