

# Metaheurísticas

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

---

### 1. Problema de la Máxima Diversidad (MDP)

- Definición y Variantes del Problema
- Ejemplo de Aplicación
- Análisis del Problema
- Solución Greedy
- Búsquedas por Trayectorias Simples
- Casos del problema
- Agradecimientos

### 2. Problema del Aprendizaje de Pesos en Características (APC)

# Definición del Problema

---

- El Problema de la Máxima Diversidad (*Maximum Diversity Problem*, MDP) es un problema de optimización combinatoria con una formulación sencilla pero una resolución compleja (**es NP-completo**)
- El problema general consiste en seleccionar un subconjunto  $Sel$  de  $m$  elementos ( $|M|=m$ ) de un conjunto inicial  $S$  de  $n$  elementos (obviamente,  $n > m$ ) de forma que se **maximice** la diversidad entre los elementos escogidos
- Además de los  $n$  elementos ( $e_i$ ,  $i=1,\dots,n$ ) y el número de elementos a seleccionar  $m$ , se dispone de una matriz  $D=(d_{ij})$  de dimensión  $n \times n$  que contiene las distancias entre ellos

# Variantes del Problema

---

Existen distintos modelos (variantes) del problema que dependen de la forma en la que se calcula la **diversidad**:

- *MaxSum (MDP)*: La diversidad se calcula como la suma de las distancias entre cada par de elementos seleccionados
- *MaxMin (MMDP)*: La diversidad se calcula como la distancia mínima entre los pares de elementos seleccionados
- *Max Mean Model*: La diversidad se calcula como el promedio de las distancias entre los pares de elementos seleccionados
- *Generalized Max Mean*: Existen pesos asociados a los elementos empleados en el denominador al calcular el promedio de distancias (*hay un orden de importancia de los elementos*)

# Definición del Problema MaxSum (MDP)

---

- Así, la definición matemática del problema *MaxSum Diversity Problem* (MDP), **con el que trabajaremos en prácticas**, es:

$$\text{Maximizar } z_{MS}(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij} x_i x_j$$

$$\begin{aligned} \text{Sujeto a } & \sum_{i=1}^n x_i = m \\ & x_i = \{0, 1\}, \quad i = 1, \dots, n. \end{aligned}$$

donde  $x$  es el vector binario solución al problema

- Las distancias entre pares de elementos se usan para formular el modelo como un problema de optimización binario cuadrático
- Esa formulación es poco eficiente. Se suele resolver como un problema equivalente de programación lineal entera

# Definición del Problema MaxMin (MMDP)

---

- La definición matemática del problema *MaxMin Diversity Problem* es:

$$\begin{aligned} \text{Maximizar} \quad & z_{MM} = \min_{i < j} d_{ij} x_i x_j \\ \text{s.a.} \quad & \sum_{i=1}^n x_i = m \\ & x_i \in \{0, 1\} \end{aligned}$$

# Definición del Problema Generalized Max Mean

---

- La definición matemática del problema *Generalized Max Mean Diversity Problem* es:

$$\text{Maximizar} \quad \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^n d_{ij} x_i x_j}{\sum_{i=1}^n w_i x_i}$$

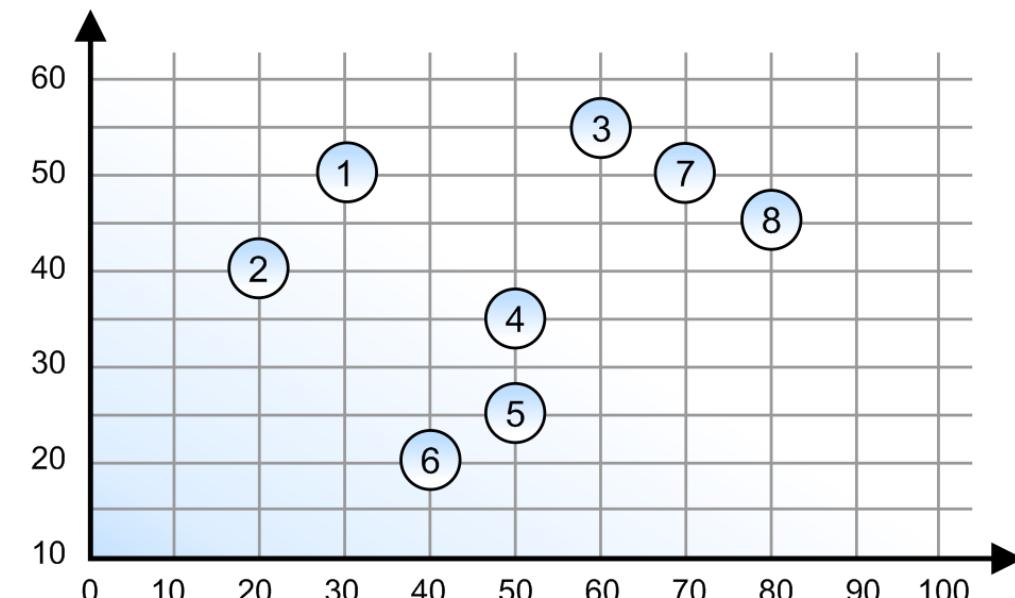
$$\begin{aligned} \text{s.a.} \quad & \sum_{i=1}^n x_i = m \\ & x_i \in \{0, 1\} \end{aligned}$$

donde  $x$  es el vector binario solución al problema y  $w$  es un vector de pesos que indica la importancia de los elementos ( $w_i \in [0,1]$  y normalmente  $\sum_{i=1}^n w_i = 1$ )

# Ejemplo de Aplicación: Selección de miembros de un Comité

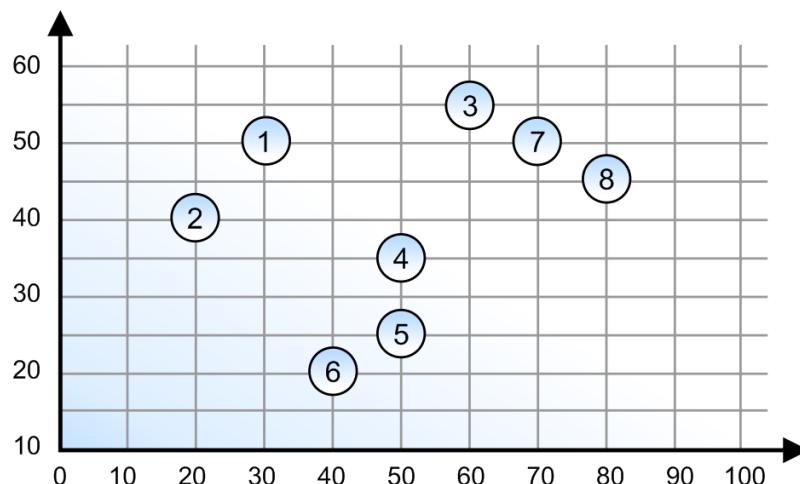
- En un centro médico, tenemos que crear un comité formado por  $m=4$  empleados. Queremos diseñar el comité más diverso de entre los  $n=8$  miembros de plantilla con respecto a su edad e ingresos:

	Edad años	Ingresos miles €/año
1	50	30
2	40	20
3	55	60
4	35	50
5	25	50
6	20	40
7	50	70
8	45	80



# Ejemplo de Aplicación: Selección de miembros de un Comité

- La distancia entre los puntos del gráfico refleja la diferencia entre los empleados representados
- La matriz  $D$  contiene los valores de dichas distancias. En este ejemplo se ha empleado la distancia Euclídea aunque se pueden usar otras métricas



	2	3	4	5	6	7	8
1	14	30	25	32	32	40	50
2		43	30	34	28	51	60
3			22	32	40	11	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Matriz de distancias  $D$

# Ejemplo de Aplicación: Selección de miembros de un Comité

## EJEMPLO DEL MODELO MAXSUM (MDP)

- La diversidad entre los elementos escogidos es la **suma** de las distancias existentes entre ellos:

	2	3	4	5	6	7	8
1	14	30	25	32	32	40	50
2		43	30	34	28	51	60
3			22	32	40	11	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Empleados seleccionados:

$$x = \{ 3, 4, 6, 8 \}$$

22 40 22 18 32 47

suma

$$z_{MS}(x) = 181$$

La solución del modelo **MaxSum** consiste en encontrar el conjunto de 4 empleados con **la mayor suma de distancias** entre ellos

# Ejemplo de Aplicación: Selección de miembros de un Comité

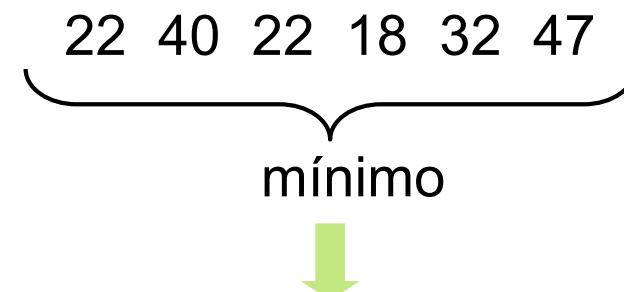
## EJEMPLO DEL MODELO MAXMIN (MMDP) (1/3)

- La diversidad entre los elementos escogidos es el **mínimo** de las distancias existentes entre ellos:

	2	3	4	5	6	7	8
1	14	30	25	32	32	40	50
2		43	30	34	28	51	60
3			22	32	40	11	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Sol 1: Empleados seleccionados:

$$x = \{ 3, 4, 6, 8 \}$$



$$z_{MS}(x_1) = 18$$

La solución del modelo **MaxMin** consiste en encontrar el conjunto de 4 empleados con **la mayor distancia mínima** entre ellos

# Ejemplo de Aplicación: Selección de miembros de un Comité

## EJEMPLO DEL MODELO MAXMIN (MMDP) (2/3)

- La diversidad entre los elementos escogidos es el **mínimo** de las distancias existentes entre ellos:

	2	3	4	5	6	7	8
1	14	30	25	32	32	40	50
2	43	30	34	28	51	60	
3		22	32	40	11	22	
4			10	18	25	32	
5				11	32	36	
6					42	47	
7						11	

Sol 2: Empleados seleccionados:

$$x = \{ 1, 3, 6, 7 \}$$

30 32 40 40 11 42

mínimo

$$z_{MS}(x_2) = 11$$

La solución del modelo **MaxMin** consiste en encontrar el conjunto de 4 empleados con **la mayor distancia mínima** entre ellos

# Ejemplo de Aplicación: Selección de miembros de un Comité

## EJEMPLO DEL MODELO MAXMIN (MMDP) (3/3)

- La diversidad entre los elementos escogidos es el **mínimo** de las distancias existentes entre ellos:

Sol 1: Empleados seleccionados:

$$x = \{ 3, 4, 6, 8 \}$$

22 40 22 18 32 47

mínimo



$$z_{MS}(x_1) = 18$$

Sol 2: Empleados seleccionados:

$$x = \{ 1, 3, 6, 7 \}$$

30 32 40 40 11 42

mínimo



$$z_{MS}(x_2) = 11$$

La solución del modelo **MaxMin** consiste en encontrar el conjunto de 4 empleados con **la mayor distancia mínima** entre ellos

# Otras Aplicaciones

---

- **Planificación de Mercado:** Maximización del número y la diversidad en las fortalezas del perfil de una marca comercial.  
Keely, A. (1989). From Experience— Maxi-niching the Way to a Strong Brand: Positioning According to System Dynamics, Journal of Product Innovation Management 6:3, 202-206
- Cultivo de plantas, preservación ecológica y gestión de recursos genéticos
- Diseño de productos
- Gestión de equipos de trabajo en empresas (*workforces*)
- Diseño de currícula
- **Otras áreas:** contabilidad y auditoría, experimentación química, diseño experimental, investigación médica, exploración geológica, selección de portafolios, ingeniería estructural, ...

Kuo, C.C., Glover, F., Dhir, K.S. (1993). Analyzing and modeling the maximum diversity problem by zero-one programming, Decision Sciences 24:6, 1171-1185

# Análisis del Problema

---

## CORRELACIÓN ENTRE LOS DISTINTOS MODELOS

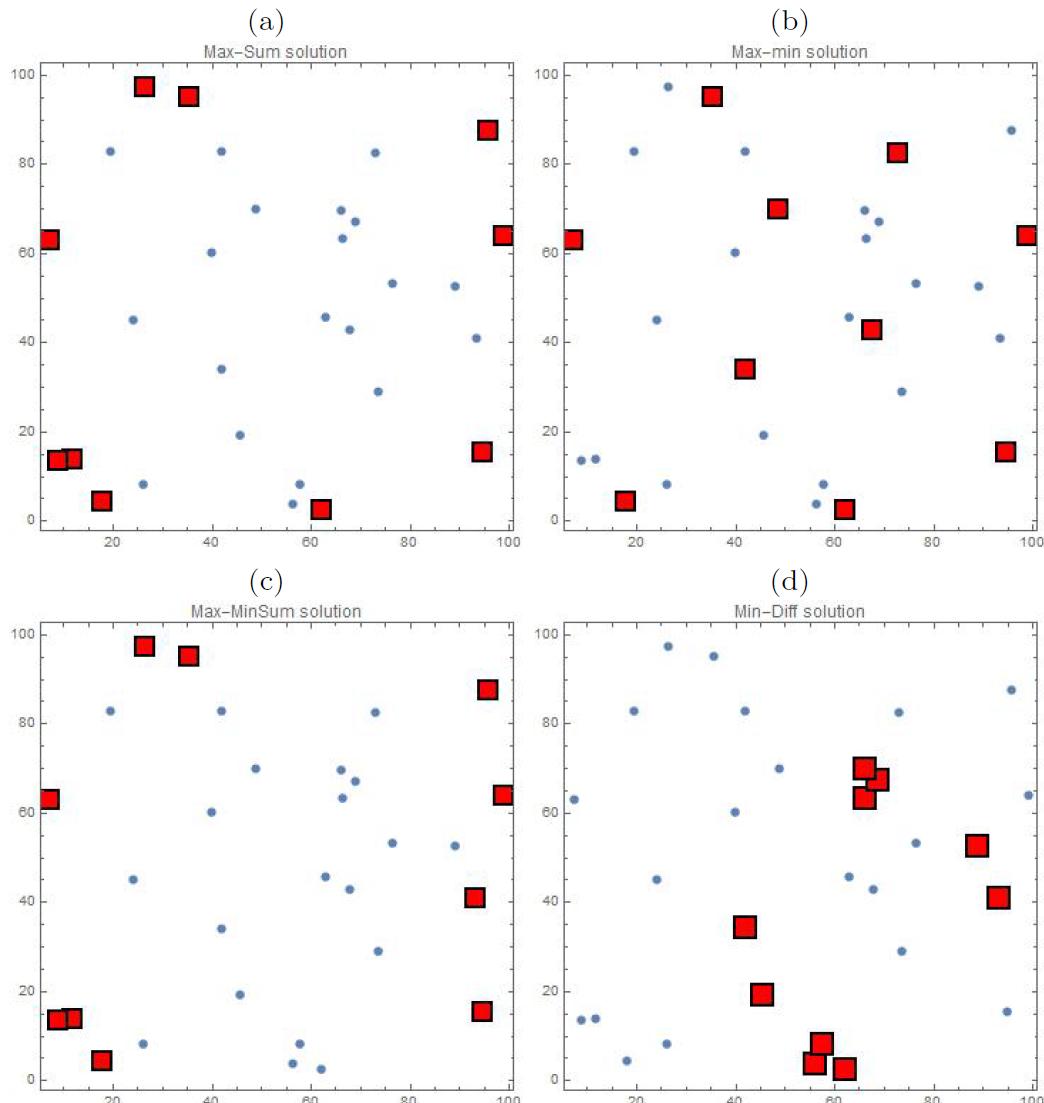
- Aunque los modelos están relacionados, no se puede esperar que un método de resolución diseñado para uno concreto funcione bien en el resto
- El grupo de investigación del Prof. Rafael Martí realizó un experimento con 30 casos aleatorios con tamaños  $n=20$  y  $m=5$ . Calcularon la correlación entre las soluciones óptimas para cuatro modelos:

Opticom Research Group	Max-Sum	Max-Min	Max-MinSum	Min-Diff
Max-Sum	1	0.60*	0.96*	-0.17
Max-Min	0.60*	1	0.73*	-0.63*
Max-MinSum	0.96*	0.73	1	-0.44
Min-Diff	-0.17	-0.63*	-0.44	1

\* diferencias significativas de acuerdo a un test estadístico

# Análisis del Problema

## DIFERENCIAS EN LAS SOLUCIONES ÓPTIMAS



- No se trata únicamente de diferencias en los valores de las funciones objetivo sino en la propia composición de las soluciones

# Solución Greedy

---

Glover, Kuo, Dhir. Heuristic algorithms for the maximum diversity problem.  
Journal of Information and Optimization Sciences 19:1 (1998) 109–132

- La complejidad del problema ha provocado que se hayan aplicado muchos algoritmos aproximados para su resolución
- Podemos determinar que una buena fórmula heurística para resolver el problema es:

*Añadir secuencialmente el elemento no seleccionado que más diversidad aporte con respecto a los ya seleccionados*

# Solución Greedy

---

- El algoritmo Greedy de Glover hace uso del concepto de elemento más central de un conjunto  $X$ :

$$s_{central}(X) = \frac{\sum_{s_i \in X} s_i}{|X|}$$

- El primer elemento seleccionado es el de mayor distancia del elemento más central
- Cada vez que se añade un nuevo elemento al conjunto de seleccionados  $Sel$ , se actualiza el elemento más central
- El proceso itera hasta seleccionar los  $m$  elementos deseados

# Solución Greedy

---

## ALGORITMO GREEDY MDP DE GLOVER 1998:

---

1.  $Sel = \emptyset$

2. Compute  $s_c = s\_center(S)$

while ( $|Sel| < m$ )

3. Let  $i^* / d(s_{i^*}, s_c) = \max_{s_i \in S} \{d(s_i, s_c)\}$

4.  $Sel = Sel \cup \{s_{i^*}\}$

5.  $S = S - \{s_{i^*}\}$

6.  $s_c = s\_center(Sel)$

end while

Centroide del conjunto  
INICIAL de elementos  $S$

Centroide del conjunto  
ACTUAL de elementos  
seleccionados  $Sel$

# Solución Greedy

---

- Como en nuestros casos del problema no dispondremos de los valores concretos de los elementos sino sólo de las distancias entre ellos, no podemos calcular los centroides de los conjuntos
- Por ello, modificaremos levemente el algoritmo:
  1.  $Sel = \emptyset$
  2. Calcular la distancia acumulada de cada elemento al resto:  $DistAc(s_i) = \sum_{s_j \in S} d(s_i, s_j)$ , incluir el elemento  $s_{i^*}$  que la maximice en  $Sel$ :  $Sel = Sel \cup \{s_{i^*}\}$  y eliminarlo de  $S$ :  $S = S - \{s_{i^*}\}$
- while** ( $|Sel| < m$ )
  3. Calcular las distancias de los elementos no seleccionados,  $s_i \in S$ , al conjunto de elementos seleccionados  $Sel$ :  $Dist(s_i, Sel) = \min_{s_j \in Sel} d(s_i, s_j)$
  4. Incluir el elemento  $s_{i^*}$  que la maximice en  $Sel$ :  $Sel = Sel \cup \{s_{i^*}\}$
  5. Eliminar el elemento  $s_{i^*}$  seleccionado de  $S$ :  $S = S - \{s_{i^*}\}$

**end while**

# Búsquedas por Trayectorias Simples: Búsqueda Local del Mejor

---

- **Representación:** Problema de selección: un conjunto  $Sel = \{s_1, \dots, s_m\}$  que almacena los  $m$  elementos seleccionados de entre los  $n$  elementos del conjunto  $S$

Para ser una solución candidata válida, tiene que satisfacer las restricciones (ser un conjunto de tamaño  $m$ ):

- No puede tener elementos repetidos
- Ha de contener exactamente  $m$  elementos
- El orden de los elementos no es relevante

# Búsquedas por Trayectorias Simples: Búsqueda Local del Mejor

---

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

Dada una solución (conjunto de elementos seleccionados) se escoge un elemento y se intercambia por otro que no estuviera seleccionado ( $Int(Sel,i,j)$ ):

$$Sel = \{s_1, \dots, i, \dots, s_m\} \quad \Rightarrow \quad Sel' = \{s_1, \dots, j, \dots, s_m\}$$

- $Int(Sel,i,j)$  verifica las restricciones: si la solución original  $Sel$  es factible y el elemento  $j$  se escoge de los no seleccionados en  $Sel$ , es decir, del conjunto  $S-Sel$ , siempre genera una solución vecina  $Sel'$  factible

# Búsquedas por Trayectorias Simples: Búsqueda Local del Mejor

---

- Su aplicación provoca que el tamaño del entorno sea:

$$|E(Sel)| = m \cdot (n - m)$$

- La BL del Mejor del MDP explora todo el vecindario, las soluciones resultantes de los  $m \cdot (n-m)$  intercambios posibles, escoge el mejor vecino y se mueve a él siempre que se produzca mejora
- Si no la hay, detiene la ejecución y devuelve la solución actual
- El método funciona bien pero es muy lento incluso para casos no demasiado grandes ( $n=500$ ) y usando un **cálculo factorizado del coste  $z_{MS}(Sel)$**  para acelerar la ejecución ( $O(n)$ )
- Es recomendable utilizar una estrategia avanzada más eficiente

# Búsqueda Local del Primer Mejor para el MDP

---

Duarte, Martí, Tabu search and GRASP for the maximum diversity problem,  
European Journal of Operational Research 178 (2007) 71–84

- 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 (o tras un número fijo de evaluaciones)
- Se **explora el vecindario de forma inteligente**:
  - Se calcula la **contribución** de cada elemento seleccionado al coste de la solución actual (valor de la función objetivo  $z_{MS}(Sel)$ )
  - Se aplican primero los intercambios de elementos que menos contribuyen
- Se considera una **factorización para calcular el coste** de  $Sel'$  a partir del de  $Sel$  considerando sólo el cambio realizado en la función objetivo por el movimiento aplicado. Además, se “**factoriza**” también el cálculo de la contribución

# BL-MDP: Exploración Inteligente del Vecindario

---

- 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 del Mejor (incluso puede mejorarla en algunos problemas)
- Se basa en definir un orden de aplicación de los intercambios (exploración de los vecinos) en una BL del primer mejor
- En cada iteración, se define un valor  $d_i$  para cada elemento seleccionado  $s_i$  de la solución actual que mide la **contribución del elemento a su coste**  $z_{MS}(Sel)$ :

$$d_i = \sum_{s_j \in Sel} d_{ij} = d(s_i, Sel)$$

# BL-MDP: Exploración Inteligente del Vecindario

---

- En lugar de calcular el valor del movimiento  $Int(Sel, i, j)$  para todos los intercambios posibles, se escoge el elemento  $s_{i^*}$  de  $Sel$  que presenta el menor aporte (es decir, el valor mínimo  $d_{i^*}$ )
- Tras escoger el elemento a extraer, se prueban sucesivamente los intercambios por los elementos no seleccionados:
  - Si se encuentra un movimiento de mejora, se aplica. Si no, se pasa al siguiente elemento con menor aporte y se repite el proceso
  - Si ningún movimiento del vecindario provoca mejora, se finaliza la ejecución y se devuelve la solución actual

# BL-MDP: Factorización del Movimiento de Intercambio

---

- Sea  $z_{MM}(Sel)$  el coste de la solución original  $Sel$ :

$$z_{MM}(Sel) = \sum_{s_i, s_j \in Sel} d_{ij}$$

- Para generar  $Sel'$ , el operador de vecino  $Int(Sel, i, j)$  escoge un elemento seleccionado  $i$  y lo cambia por uno no seleccionado  $j$ :

$$Sel = \{s_1, \dots, \textcolor{blue}{i}, \dots, s_m\} \Rightarrow Sel' \leftarrow Sel - \{\textcolor{blue}{i}\} + \{\textcolor{red}{j}\} \Rightarrow Sel' = \{s_1, \dots, \textcolor{red}{j}, \dots, s_m\}$$

- No es necesario recalcular todas las distancias de la función objetivo:
  - Al añadir un elemento, las distancias entre los que ya estaban en la solución se mantienen y basta con sumar la distancia del nuevo elemento al resto de elementos seleccionados
  - Al eliminar un elemento, las distancias entre elementos que se quedan en la solución se mantienen y basta con restar la distancia del elemento eliminado al resto de elementos en la solución
- Por tanto, quedan afectados  $2 \cdot (m-1)$  sumandos de la función objetivo, las  $m-1$  distancias del elemento que **se elimina** y las  $m-1$  del que **se añade**

# BL-MDP: Factorización del Movimiento de Intercambio

---

- El coste del movimiento (la **diferencia de costes entre las dos soluciones**)  $\Delta z_{MM}(Sel, i, j) = z_{MM}(Sel') - z_{MM}(Sel)$  se puede factorizar:

$$\Delta z_{MM}(Sel, i, j) = \sum_{s_k \in Sel - \{s_i\}} d_{kj} - d_{ki} \quad \text{eliminada añadida}$$

- Si  $\Delta z_{MM}(Sel, i, j)$  es positivo ( $\Delta z_{MM}(Sel, i, j) > 0$ ), la solución vecina  $Sel'$  es mejor que la actual  $Sel$  (**el MDP es un problema de maximización**) y se acepta. Si no, se descarta y se genera otro vecino
- Podemos combinar fácilmente la factorización del coste con el cálculo de la contribución de los elementos para mejorar aún más la eficiencia:
  - Las distancias del elemento eliminado equivalen directamente a la contribución de dicho elemento,  $d_i = \sum_{s_k \in Sel - \{s_i\}} d_{ki}$
  - El cálculo de las aportaciones de los elementos actualmente seleccionados también se puede factorizar. No es necesario recalcularlo completamente, basta con restar la distancia del elemento eliminado y sumar la del añadido:  $d_k = d_k + d_{kj} - d_{ki}$

# BL-MDP: Factorización del Movimiento de Intercambio

---

- El coste  $z_{MM}(Sel')$  de la nueva solución vecina es:

$$z_{MM}(Sel') = z_{MM}(Sel) + \Delta z_{MM}(Sel, i, j)$$

- Sólo es necesario calcularlo al final de la ejecución. Durante todo el proceso, basta con trabajar con el coste del movimiento
- El pseudocódigo de la BL del Primer Mejor del Tema 2 de Teoría quedaría:

## Repetir

$Sel' \leftarrow \text{GENERA\_VECINO}(Sel);$

**Hasta** ( $\Delta z_{MM}(Sel, i, j) > 0$ ) **O**  
(se ha generado  $E(Sel)$  al completo)

# Casos del Problema

---

- Existen distintos grupos de casos del problema para los que se conoce la solución óptima que permiten validar el funcionamiento de los algoritmos de resolución
- Para el MDP, disponemos de cuatro grandes grupos de casos:
  - **Casos GKD** (Glover, Kuo and Dhir, 1998): Entre otras, 20 matrices  $n \times n$  con distancias Euclídeas calculadas a partir de puntos con  $r$  coordenadas ( $r \in \{2, \dots, 21\}$ ) aleatorias en  $[0,10]$ .  $n=500$  elementos y  $m=50$
  - **Casos SOM** (Silva, Ochi y Martins, 2004): Entre otras, 20 matrices  $n \times n$  con distancias enteras aleatorias en  $\{0,999\}$  con  $n \in \{100, \dots, 500\}$  elementos y  $m \in \{0.1 \cdot n, \dots, 0.4 \cdot n\}$ . P.ej. para  $n=100$  hay 4 casos con  $m=10, 20, 30, 40$
  - **Casos MDG Tipo I** (Duarte y Martí, 2007): Entre otras, 20 matrices  $n \times n$  con distancias reales aleatorias en  $[0,10]$ ,  $n=500$  y  $m=50$ . 20 con distancias enteras aleatorias en  $\{0,10\}$ ,  $n=2000$  y  $m=200$
  - **Casos MDG Tipo II** (Duarte y Martí, 2007): 60 matrices  $n \times n$  con distancias reales aleatorias en  $[0,1000]$ . 20 con  $n=500$  y  $m=50$ ; 20 con  $n=2000$  y  $m=200$ ; y 20 con  $n=3000$  y  $m=\{300,400,500,600\}$

# Casos del Problema

---

- Los casos están recopilados en la **biblioteca MDPLib**, accesible en la Web en la dirección siguiente:  
<http://www.optsicom.es/mdp/>
- En dicha dirección pueden encontrarse tanto los datos como los valores de las mejores soluciones encontradas para 315 casos del problema
- Además, están disponibles los resultados de un ejemplo de una experimentación comparativa de distintos algoritmos con 10 minutos de tiempo de ejecución por caso

# La Biblioteca MDPLIB

---

- El formato de los ficheros de datos es un fichero de texto con la siguiente estructura:

$n \ m$

$D$

donde  $n$  es el número de elementos,  $m$  es el número de elementos seleccionados y  $D$  es la matriz de distancias entre elementos que está precalculada

- Al ser  $D$  una matriz simétrica, sólo se almacena la diagonal superior. El fichero contendrá  $n \cdot (n-1)/2$  entradas, una por línea, con el siguiente formato:

$i \ j \ d_{ij}$

donde  $i, j \in \{0, \dots, n-1\}$  son respectivamente la fila y la columna de la matriz  $D$ , mientras que  $d_{ij}$  es el valor de la distancia existente entre los elementos  $i+1$  y  $j+1$

# La Biblioteca MDPLIB

---

## EJEMPLO: FICHERO DEL CASO GKD-c\_1\_n500\_m50:

```
500 50
0 1 11.17945
0 2 12.18565
0 3 15.82056
0 4 7.17287
0 5 12.63171
0 6 10.45706
0 7 12.37497
0 8 12.13219
0 9 13.07364
0 10 10.54751
0 11 9.96995
0 12 12.55428
0 13 12.86351
0 14 7.08237
...
496 497 14.48240
496 498 11.37189
496 499 13.94453
497 498 15.47191
497 499 17.05433
498 499 10.37931
```

# Agradecimientos

---

- Para la preparación de las transparencias de presentación del problema MDPLIB se han usado materiales de los profesores:
  - Rafael Martí. Universidad de Valencia
  - Abraham Duarte. Universidad Rey Juan Carlos
  - Jesús Sánchez-Oro. Universidad Rey Juan Carlos
- Su grupo de investigación ha realizado muchas publicaciones sobre el problema y mantiene la biblioteca MDPLIB:
  - Tabu Search for the **Maximum Diversity** Problem, Abraham Duarte and Rafael Martí, **European Journal of Operational Research** 178, 71-84 (2007)
  - Hybrid heuristics for the **Maximum Diversity** Problem, M. Gallego, A. Duarte, M. Laguna and R. Martí, **Computational Optimization and App.** 44(3), 411-426 (2009)
  - A Branch and Bound algorithm for the **Maximum Diversity** Problem, Martí, Gallego and Duarte, **European Journal of Operational Research** 200(1), 36-44 (2010)
  - Heuristics and Metaheuristics for the **Maximum Diversity** Problem, Rafael Martí, Micael Gallego and Abraham Duarte, **Journal of Heuristics**, 19, 591-615 (2013)

# Metaheurísticas

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

---

1. Problema de la Máxima Diversidad (MDP)

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  $w_1, \dots, w_n$ , representados en función de sus valores en una serie de atributos:

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

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

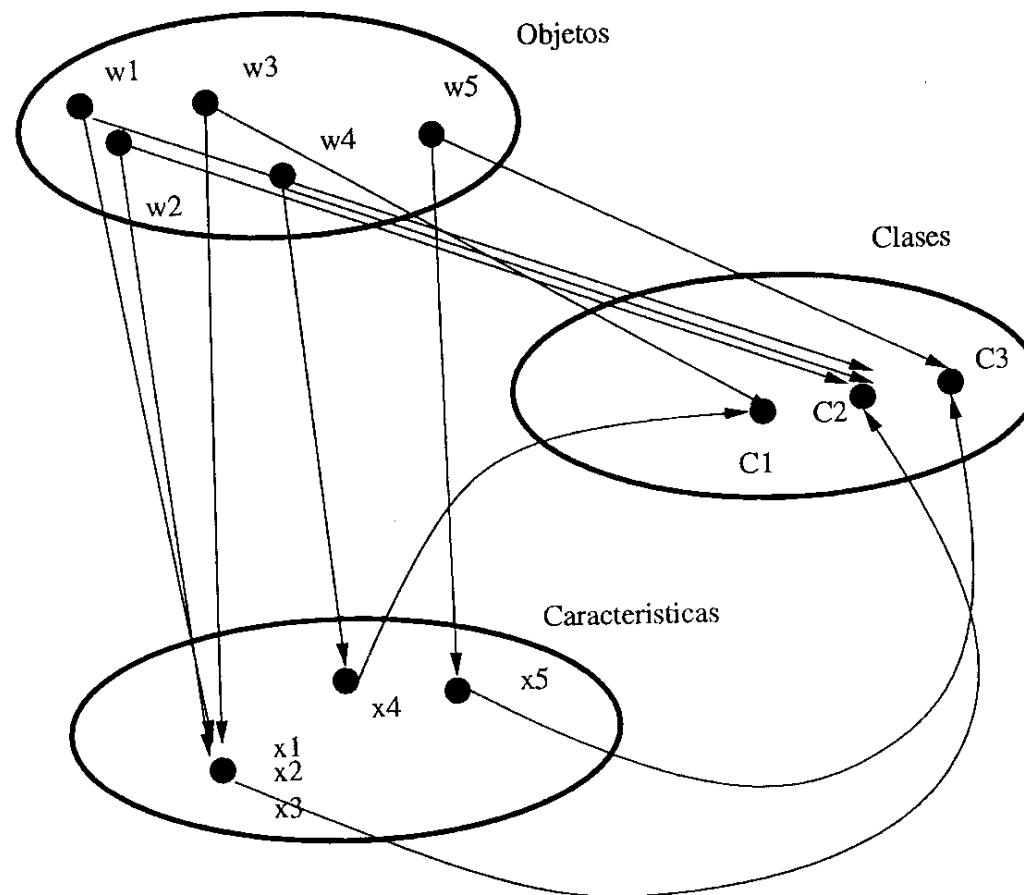
**OBJETIVO: Obtener un sistema que permita clasificar dichos objetos de modo automático**

$$\begin{aligned} w_1 = (x_1(w_1), \dots, x_n(w_1)) &\rightarrow C_{i_1} \\ \dots &\dots & i_j \in \{1, \dots, M\}, \quad i \in \{1, \dots, k\} \\ w_k = (x_1(w_k), \dots, x_n(w_k)) &\rightarrow C_{i_k} \end{aligned}$$

# 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*

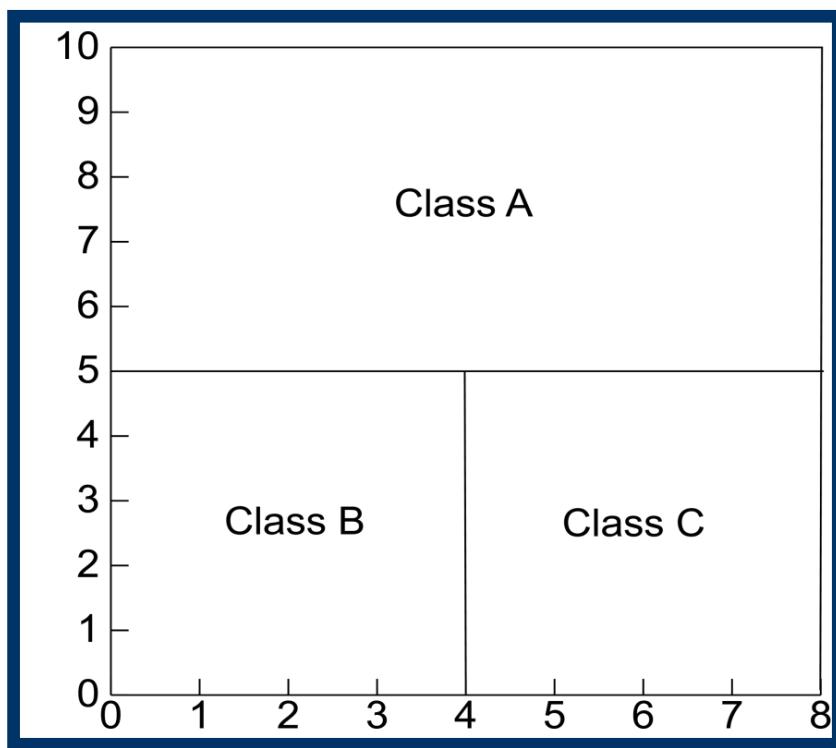


*virgínica*

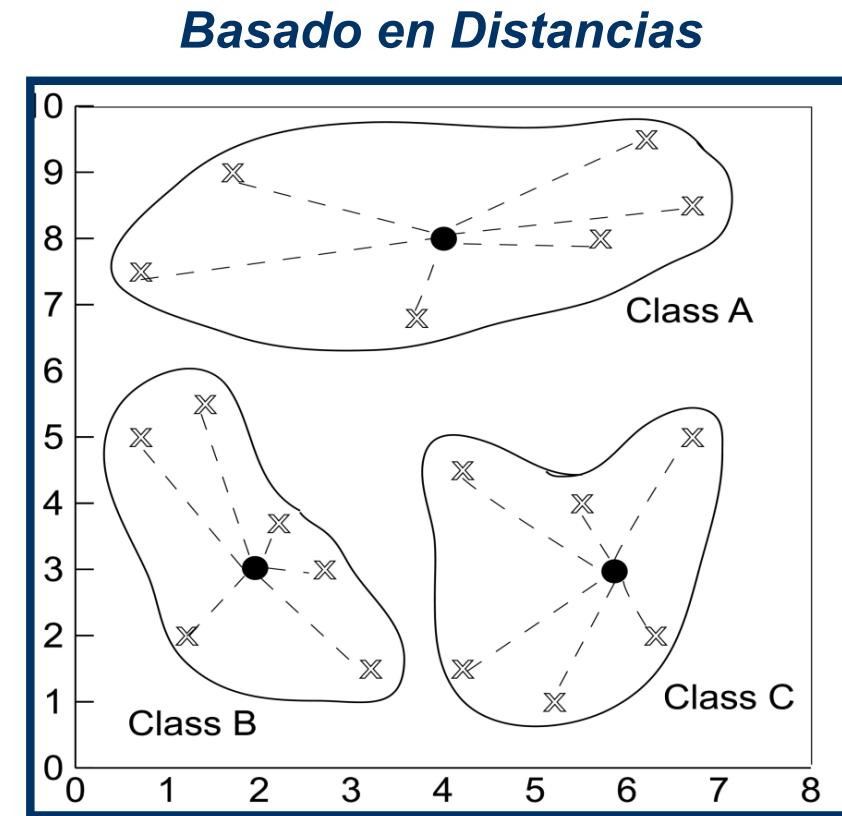
# Definición del Problema de Clasificación

---

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



*Basado en Particiones*

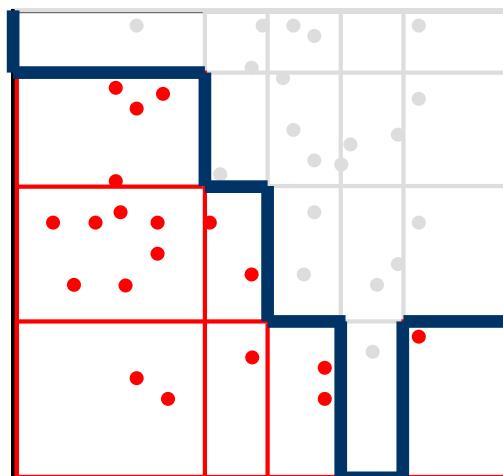


# Definición del Problema de Clasificación

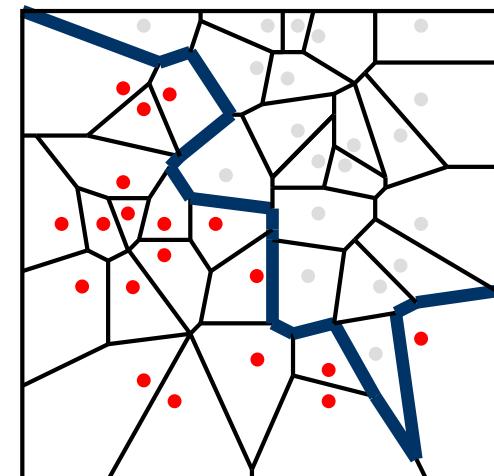
---

## Ejemplos de clasificación sobre clases definidas

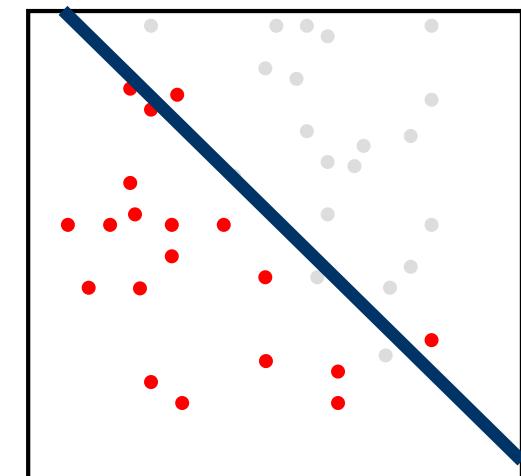
- *Reglas intervalares*



- *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 validar. Se calcula el porcentaje de clasificación sobre los ejemplos de este conjunto (desconocidos en la tarea de aprendizaje) para conocer su poder de generalización

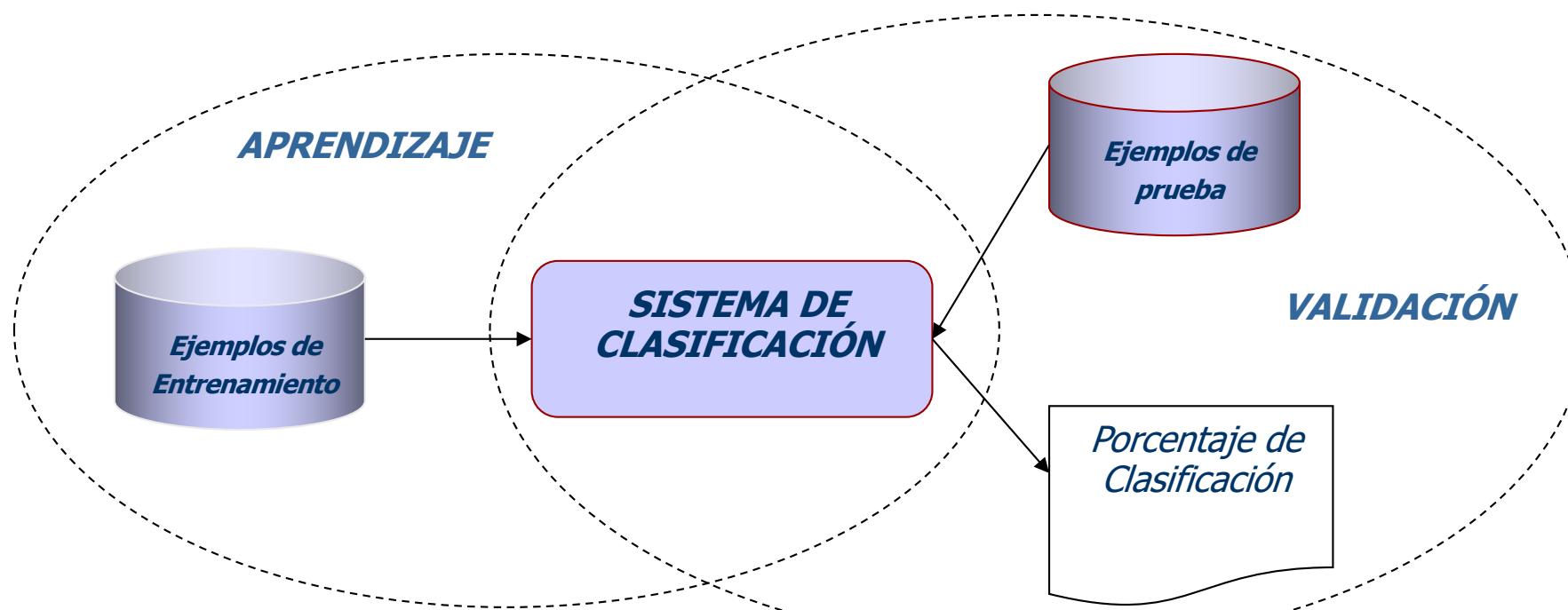
Para mayor seguridad, se suele hacer varias particiones entrenamiento-prueba

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

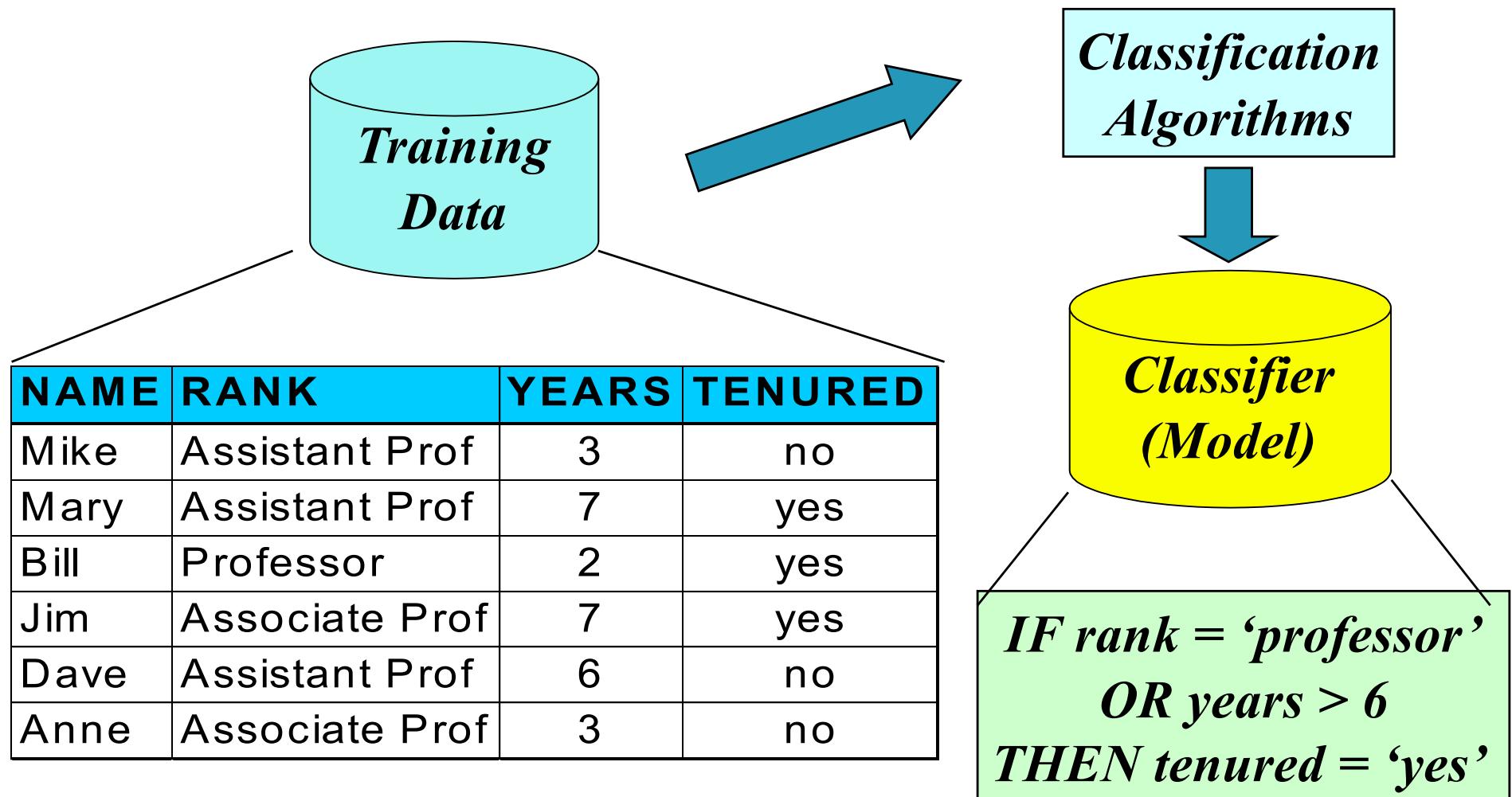
# Definición del Problema de Clasificación

---

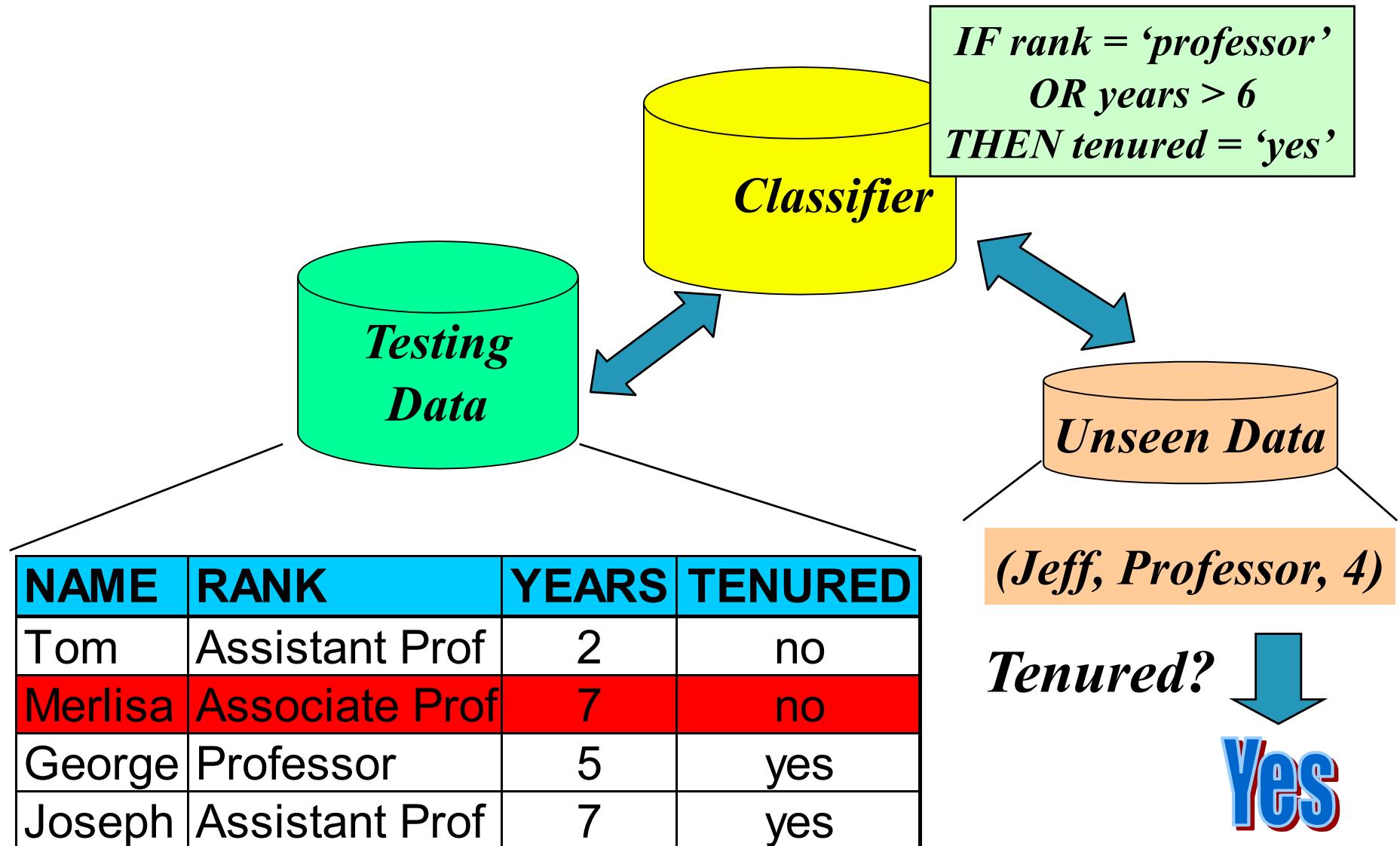
## Esquema de aprendizaje en clasificación



# Definición del Problema de Clasificación



# Definición del Problema de Clasificación



# Definición del Problema de Clasificación

---

Usaremos la técnica de validación cruzada **5-fold cross validation**:

- El conjunto de datos se divide en 5 particiones disjuntas al 20%,  
**con la distribución de clases equilibrada**
- Aprenderemos un clasificador utilizando el 80% de los datos disponibles (4 particiones de las 5) y validaremos con el 20% restante (la partición restante) → 5 particiones posibles al 80-20%
- Así obtendremos un total de 5 valores de porcentaje de clasificación en el conjunto de prueba, uno para cada partición empleada como conjunto de validación
- La calidad del método de clasificación se medirá con un único valor, correspondiente a la media de los 5 porcentajes de clasificación del conjunto de prueba

# Clasificador k-NN

---

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

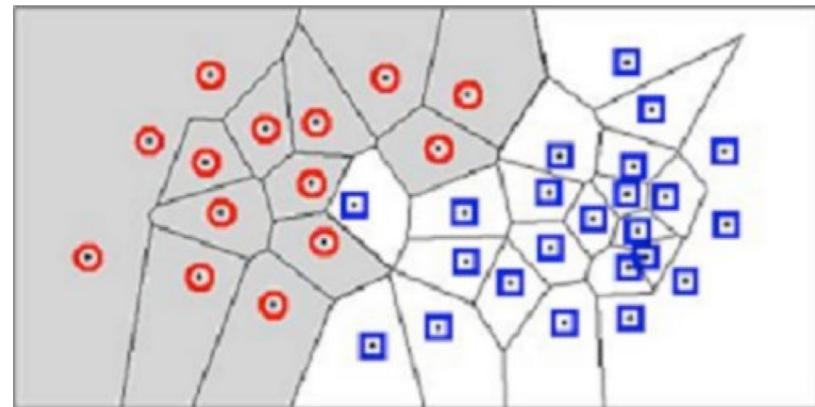
- i. El proceso de aprendizaje de este clasificador consiste en almacenar una tabla con los ejemplos disponibles, junto a la clase asociada a cada uno de ellos
- ii. Dado un nuevo ejemplo a clasificar, se calcula su distancia (usaremos la euclídea) a los  $n$  ejemplos existentes en la tabla y se escogen 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  $\{e_1, \dots, e_m\}$  en nuestro conjunto de datos, para clasificar un nuevo ejemplo  $e'$  se hará lo siguiente:

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



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

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

# Clasificador k-NN

---

## Distancias para las variables numéricas

- Las variables numéricas se suelen normalizar al intervalo  $[0,1]$
- Si  $e_j$  es el valor de la variable  $j$  en  $e_i$ , es decir  $e_i=(e_i^1,\dots,e_i^n)$  entonces algunas de las distancias más utilizadas son:

- Euclídea: 
$$d_e(e_1, e_2) = \sqrt{\sum_{i=1}^n (e_1^i - e_2^i)^2}$$

- Manhattan: 
$$d_m(e_1, e_2) = \sum_{i=1}^n |e_1^i - e_2^i|$$

- Minkowski: 
$$d_m^k(e_1, e_2) = \left( \sum_{i=1}^n |e_1^i - e_2^i|^k \right)^{1/k}$$

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

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

*Queremos clasificar con 1-NN el ejemplo:*

$x_q: 0.7 \ 0.2 \ 0.1$

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

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

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

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

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  $[0, 1]$
- Los valores bajos de  $w_i$  pueden emplearse para seleccionar características y diseñar un clasificador más simple y preciso

## 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 de clasificación concreto y el clasificador 1-NN

# Definición del Problema del Aprendizaje de Pesos en Características

---

## Función de evaluación: combinación de dos criterios, precisión y simplicidad

- 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  $T$  (**precisión**):

$$tasa - clas = 100 \cdot \frac{n^o \text{ instancias bien clasificadas en } T}{n^o \text{ instancias en } T}$$

# Definición del Problema del Aprendizaje de Pesos en Características

---

## Función de evaluación: combinación de dos criterios, precisión y simplicidad

- Tasa de reducción asociada al número de características utilizadas por el clasificador con respecto al número total de características  $n$ : ***tasa\_red***
- *tasa\_red* mide el porcentaje de características **descartadas**, aquellas cuyo peso esté cercano a cero en  $W$ , con respecto a  $n$  (***simplicidad***)
- Consideraremos un umbral de 0.2 en  $w_i$  para considerar que se descarta una característica:

$$tasa - red = 100 \cdot \frac{n^o \text{ valores } w_i < 0.2}{n^o \text{ características}}$$

# Definición del Problema del Aprendizaje de Pesos en Características

---

## Función de evaluación: combinación de dos criterios, precisión y simplicidad

- Utilizaremos una agregación sencilla que combine ambos objetivos en un único valor. La función objetivo será:

$$F(W) = \alpha \cdot tasa_{\text{clas}(W)} + (1 - \alpha) \cdot tasa_{\text{red}(W)}$$

- El valor de  $\alpha$  pondera la importancia entre el acierto y la reducción de características de la solución encontrada (el clasificador generado). Usaremos  $\alpha = 0.5$ , dando la misma importancia a ambos
- El objetivo es obtener el conjunto de pesos  $W$  que maximiza esta función, es decir, que maximice el acierto del clasificador 1-NN y, a la vez, que considere el menor número de características posible

# Definición del Problema del Aprendizaje de Pesos en Características

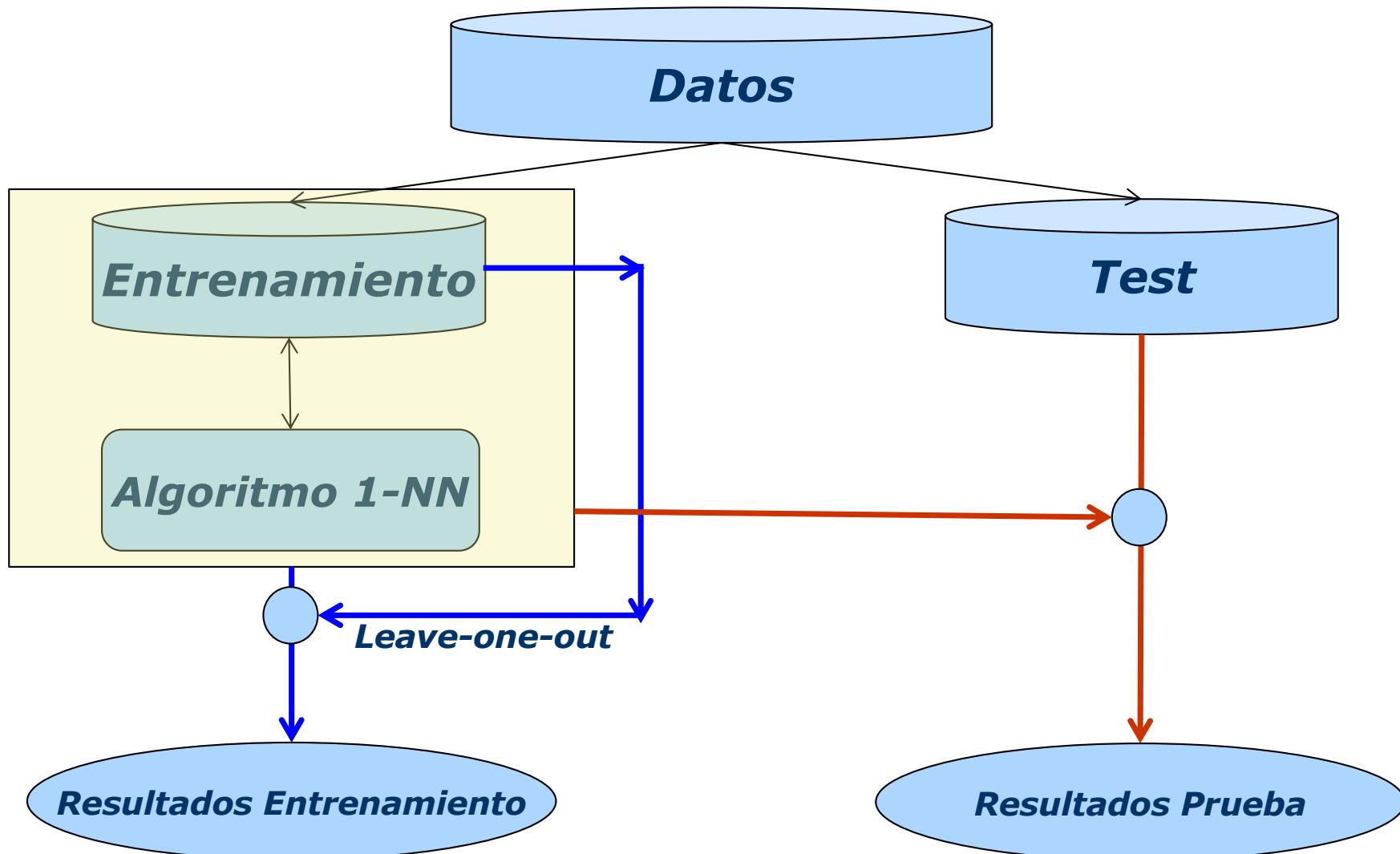
---

## Cálculo del Porcentaje de Entrenamiento (tasa-clas) en 1-NN: Leave one out

- 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 usar el procedimiento “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 coincide 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  $n$ :

$$W = (w_1, w_2, \dots, w_n), \quad \text{donde } w_i \in [0, 1]$$

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



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

Un **valor menor que 0.2** 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

# Representación del Problema del Aprendizaje de Pesos en Características

---

## 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:

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

Para cada  $e_i$  en  $T$

    Buscar el enemigo más cercano de  $e_i$ :  $e_e$

    Buscar el amigo más cercano de  $e_i$ :  $e_a$

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

$$w_m = \text{máximo } (W) \quad \text{---}$$

Para cada  $w_i$  en  $W$

    Si  $w_i < 0$  entonces

$$w_i = 0$$

    Si no

$$w_i = w_i / w_m$$

Devolver  $W$

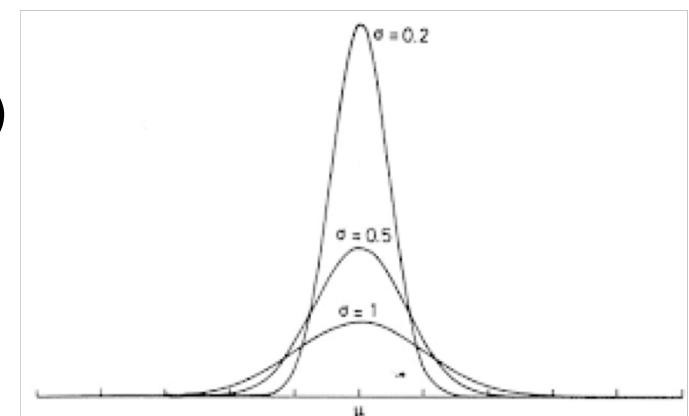
# Búsquedas por Trayectorias Simples

---

- **Representación real:** Problema de codificación real: un vector real  $W=(w_1, \dots, w_n)$  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 basado en la mutación de una componente  $z_i$ , con un radio que depende de  $\sigma$ :

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

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



# Búsquedas por Trayectorias Simples

---

- $Mov(W, \sigma)$  verifica las restricciones si después de aplicarlo, truncamos el  $w_i$  modificado a  $[0,1]$ . Así, si la solución original  $W$  es factible siempre genera una solución vecina  $W'$  factible
- El problema del APC **no permite realizar un cálculo factorizado del coste** de forma sencilla
- El tamaño del entorno es infinito, al ser un problema de codificación real. No es fácil definir una preferencia entre las características para explorar el entorno. Podría considerarse alguna medida de cantidad de información para ello
- En nuestro caso, en cada paso de la exploración se mutará una componente  $i \in \{1, \dots, n\}$  **distinta** sin repetición hasta que haya mejora o se hayan modificado todas. Si se produce mejora, se acepta la solución vecina y se comienza de nuevo el proceso
- Si ninguna de las  $n$  mutaciones produce mejora, se empieza de nuevo el proceso de exploración de vecindario sobre la solución actual

# 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 haya generado un número máximo de vecinos
- No se considera ningún tipo de **factorización** ni ningún mecanismo específico de exploración del entorno más haya de la generación aleatoria de la componente a mutar sin repetición

# 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
```

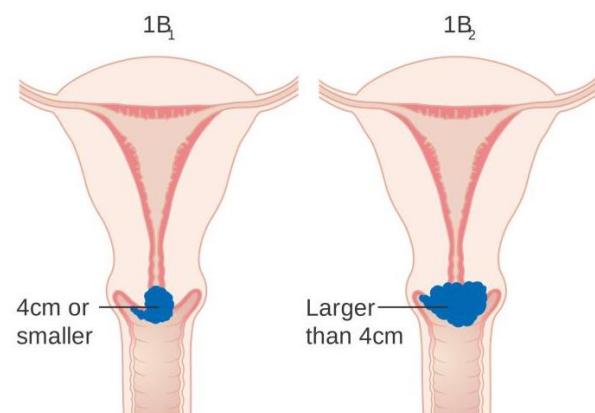
- 5 Atributos, los 4 primeros de tipo real y el último de tipo nominal
- La clase es el último atributo (atributo de salida) con los posibles valores definidos
- Los datos van a continuación de la directiva @data

# Bases de Datos a Utilizar

---

Trabajaremos con tres bases de datos: Colposcopy, Ionosphere y Texture

**Colposcopy** La colposcopia es un procedimiento ginecológico que consiste en la exploración del cuello uterino. El conjunto de datos fue adquirido y anotado por médicos profesionales del Hospital Universitario de Caracas. Las imágenes fueron tomadas al azar de las secuencias colposcópicas



- Consta de 287 ejemplos
- Consta de 62 atributos reales
- Consta de 2 clases (positivo o negativo)
- Atributos: extracciones de características desde la imagen

# Bases de Datos a Utilizar

---

Trabajaremos con tres bases de datos: Colposcopy, Ionosphere y Texture

**Ionosphere** datos de radar recogidos por un sistema en Goose Bay, Labrador. Este sistema consiste en un conjunto de fases de 16 antenas de alta frecuencia con una potencia total transmitida del orden de 6,4 kilovatios. Los objetivos eran electrones libres en la ionosfera. Los "buenos" retornos de radar son aquellos que muestran evidencia de algún tipo de estructura en la ionosfera. Los retornos "malos" son aquellos que no lo hacen, sus señales pasan a través de la ionosfera



- Consta de 352 ejemplos
- Consta de 34 atributos
- Consta de 2 clases (retornos buenos y malos).
- Atributos: Señales procesadas.

# Bases de Datos a Utilizar

---

Trabajaremos con tres bases de datos: Colposcopy, Ionosphere y Texture

**Texture** El objetivo de este conjunto de datos es distinguir entre 11 texturas diferentes (césped, piel de becerro prensada, papel hecho a mano, rafia en bucle a una pila alta, lienzo de algodón,...), caracterizándose cada patrón (píxel) por 40 atributos construidos mediante la estimación de momentos modificados de cuarto orden en cuatro orientaciones: 0, 45, 90 y 135 grados



- Consta de 550 ejemplos (**seleccionados de los 5500 originales, manteniendo la distribución de clases intacta**)
- Consta de 40 atributos
- Consta de 11 clases (tipos de texturas)
- Atributos: Extracción de características desde imágenes