

Metaheurísticas

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

1. Problema de la Mínima dispersión diferencial (MDD)

2. Definición del Problema
3. Ejemplo de Aplicación
4. Análisis del Problema
5. Solución Greedy
6. Búsquedas por Trayectorias Simples
7. Casos del problema.
8. Agradecimientos

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

Definición del Problema

- El Problema de Dispersión Diferencial, Minimum Differential Dispersion Problem (Min-Diff) es un problema de optimización combinatoria con una formulación sencilla pero una resolución compleja (**es NP-completo**), que solo con tamaño 50 implica más de 1 hora.
- 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 **minimize** la dispersión 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

Definición del Problema MDDD

- Para el problema de la mínima dispersión diferencial, Minimum Differential *Dispersion*, MDD, **con el que trabajaremos en**

$$\text{Minimize } \text{Max}_{i \in M} \left\{ \sum_{j \in M} d_{ij} \right\} - \text{Min}_{i \in M} \left\{ \sum_{j \in M} d_{ij} \right\}$$

$$\text{Subject to } M \subset N, |M| = m$$

- 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 MDD

- Para el problema MDD, **con el que trabajaremos en prácticas**, se calcula la dispersión como:

1) Para cada punto elegido v se calcula $\Delta(v)$ como la suma de las distancias de este punto al resto.

$$\Delta(v) = \sum_{u \in S} d_{uv}.$$

2) La dispersión de una solución, denotada como $\text{diff}(S)$ se define como la diferencia entre los valores extremos:

$$\text{diff}(S) = \max_{u \in S} \Delta(u) - \min_{v \in S} \Delta(v).$$

3) El objetivo es minimizar dicha medida de dispersión:

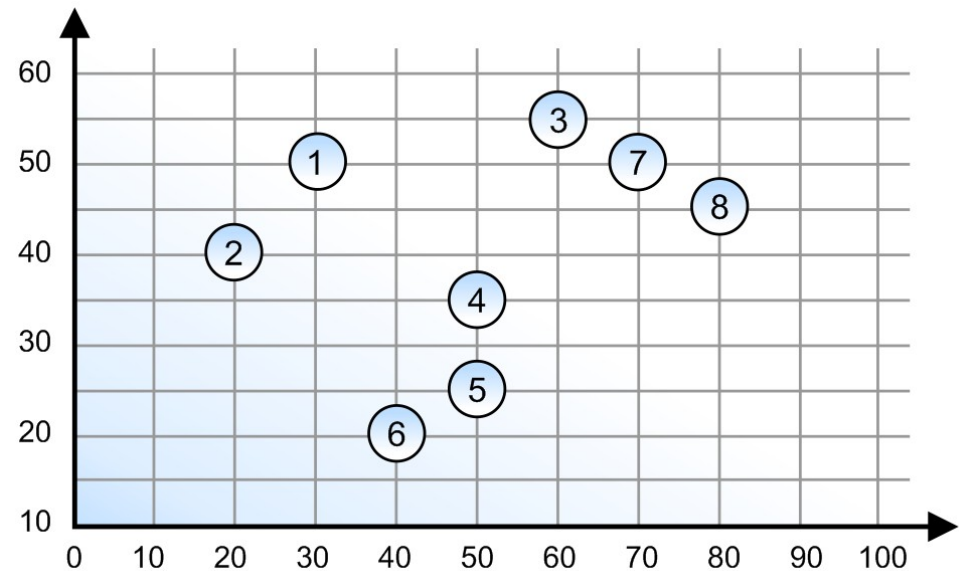
$$S^* = \arg \min_{S \subseteq V_m} \text{diff}(S),$$

donde S es el conjunto solución al problema

Ejemplo de Aplicación: Colocando Farmacias

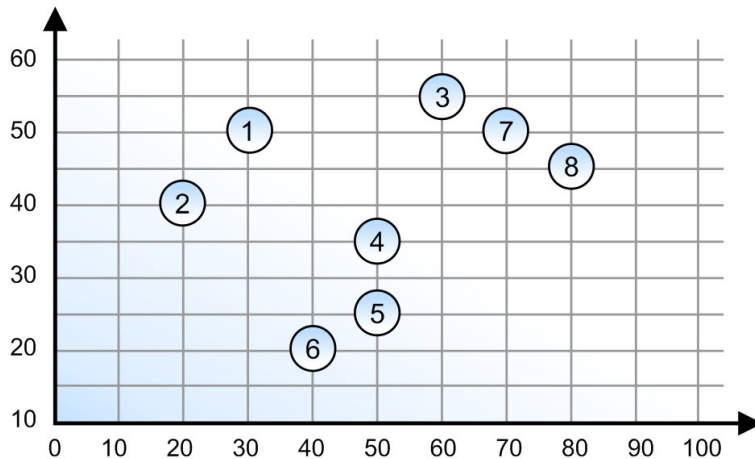
- Tenemos $n=8$ posibles localizaciones para colocar $m=4$ farmacias. Queremos situarlas de tal manera que estén separadas entre sí a una distancia parecida (mínima dispersión entre sí):

	Latitud	Longitud
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: Colocando Farmacias

- La distancia entre los puntos del gráfico refleja la distancia entre las distintas localizaciones
- 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	30	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Matriz de distancias D

Ejemplo de Aplicación: Colocando Farmacias

EJEMPLO DEL MODELO MIN-DIFF (Min-Diff)

- La dispersión entre los elementos escogidos es la **máxima diferencia** de las sumas 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	30	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Localizaciones seleccionadas:

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

$$\left. \begin{aligned} V(3) &= 22 + 40 + 22 = 84 \\ V(4) &= 22 + 18 + 32 = 72 \\ V(6) &= 40 + 18 + 47 = 105 \\ V(8) &= 22 + 32 + 47 = 101 \end{aligned} \right\} \text{Diferencia}$$

$$\text{diff}(x) = 105 - 72 = 33$$

La solución del modelo **MinDiff** consiste en encontrar las localizaciones con **la menos dispersión** entre ellos

Ejemplo de Aplicación: Colocando Farmacias

EJEMPLO DEL MODELO MINDIFF (MinDiff) (2/3)

- La dispersión entre los elementos escogidos es la **máxima diferencia** de las sumas 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	30	22
4				10	18	25	32
5					11	32	36
6						42	47
7							11

Sol 2: Localizaciones seleccionadas:

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

$$V(1) = 30 + 32 + 40 = 102$$

$$V(3) = 30 + 40 + 30 = 100$$

$$V(6) = 32 + 40 + 42 = 114$$

$$V(7) = 40 + 30 + 42 = 112$$

Diferencia



$$\text{diff}(x) = 114 - 100 = 14$$

La solución del modelo **MinDiff** consiste en encontrar las localizaciones con **la menos dispersión** entre ellos

Ejemplo de Aplicación: Colocando Farmacias

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

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

Sol 1: seleccionadas:

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

84 72 105 101

Diferencia

$$\text{diff}(x_1) = 33$$

Sol 2: seleccionadas:

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

102 100 114 112

Diferencia

$$\text{diff}(x_2) = 14$$

La solución del modelo **MinDiff** consiste en encontrar las localizaciones con **la menor dispersión** entre ellos

Aplicaciones del Problema

- Elegir localización de elementos públicos (farmacias, ...)
- Selección de grupos homogéneos
- Identificación de redes densas
- Reparto equitativo
- Problemas de flujo

Duarte, A, Sánchez-Oro, J., Resende, M.G.C, Glover, F, Martí, R (2015). Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. Information Sciences 296, 46-60

Solución Greedy

Duarte, A, Sánchez-Oro, J., Resende, M.G.C, Glover, F, Martí, R (2015). Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. Information Sciences 296, 46-60

- 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 reduzca la dispersión con respecto a los ya seleccionados

Solución Greedy

- El algoritmo valora en cada caso cómo varía la dispersión al seleccionar cada nuevo elemento:
- El primer elemento seleccionado no está definido, puede ser aleatorio.
- Cada vez que se añade un nuevo elemento al conjunto de seleccionados Sel , se valora cuál incrementa menos (o reduce) la dispersión respecto a los ya elegidos
- El proceso itera hasta seleccionar los m elementos deseados

Solución Greedy

ALGORITMO GREEDY:

```
1:  $S \leftarrow \emptyset$ 
2:  $CL \leftarrow V$ 
3:  $v_0 \leftarrow \text{SelectRandom}(CL)$ 
4:  $S \leftarrow S \cup \{v_0\}$ 
5:  $CL \leftarrow CL \setminus \{v_0\}$ 
6: while  $|S| < m$  do
7:    $RCL \leftarrow CL$ 
8:    $u \leftarrow \arg \min_{v \in RCL} g(v)$ 
9:    $S \leftarrow S \cup \{u\}$ 
10:   $CL \leftarrow CL \setminus \{u\}$ 
11: end while
12: return  $S$ 
```

Solución Inicial

Aplicar heurística

Solución Greedy

El cálculo de $g(u)$ se aplica de la siguiente manera:

1) Para cada elemento u no escogido:

$$\forall u \in V - Sel, \partial(u) = \sum_{v \in Sel} d_{uv}$$

2) Luego para cada elemento v existente:

$$\forall v \in Sel, \partial(v) = SumaAnterior(v) + d_{uv}$$

3) Una vez actualizado las sumas para cada elemento, se calcula:

$$\partial_{max}(u) = \max(\partial(u), \max_{v \in Sel} \partial(v)) \quad \partial_{min}(u) = \min(\partial(u), \min_{v \in Sel} \partial(v))$$

4) El cálculo final de $g(u)$ es:

$$g(u) = \partial_{max}(u) - \partial_{min}(u)$$

Búsquedas por Trayectorias Simples: Búsqueda Local del Mejor

- **Representación:** Problema de selección: un conjunto $S \subseteq \{s_1, \dots, s_m\}$ 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 ($Int(Sel, i, j)$):

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

- $Int(Sel, i, j)$ 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 demasiado grande, $m \cdot (m-1)$.
- La BL del Mejor 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 $z_{MS}(Sol)$** para acelerar la ejecución ($O(n)$)
- Es recomendable utilizar una estrategia avanzada más eficiente

Búsqueda Local del Primer Mejor

Duarte, A, Sánchez-Oro, J., Resende, M.G.C, Glover, F, Martí, R (2015). Greedy randomized adaptive search procedure with exterior path relinking for differential dispersion minimization. Information Sciences 296, 46-60

- 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 considera una **factorización para calcular el coste de Se'** a partir del de Se considerando sólo el cambio realizado en la función objetivo por el movimiento aplicado. Además, se **“factoriza” también el cálculo de la contribución**

Factorización del Movimiento de Intercambio

- Para generar Sel' , el operador de vecino $Int(Sel, i, j)$ escoge un elemento seleccionado i y lo cambia por uno no seleccionado j :
- $Sel = \{s_1, \dots, i, \dots, s_m\} \Rightarrow Sel' \leftarrow Sel - \{i\} + \{j\} \Rightarrow Sel' = \{s_1, \dots, j, \dots, s_m\}$
- 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 calcular la dispersión por el 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
- Se calcula el nuevo coste de la solución original Sel como:

$$Z_{MM}(Sel, u, v) = (\partial_{max} - \partial_{min}) - z_{MM}(Sel)$$

$$\partial_{max} = \max(\partial(v), \max_{w \in Sel} \partial(w)) \quad \partial_{min} = \min(\partial(v), \min_{w \in Sel} \partial(w))$$

$$\partial(v) = \sum_{w \in Sel} d_{vw} \quad \forall w \in Sel, \partial(w) = \text{anterior}(w) - d_{wu} + d_{wv}$$

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 calcula factorizado.
- El cálculo original, implicaba calcular para cada uno de los m elementos la distancia al resto, por tanto era $O(n^2)$. De forma factorizada es sólo $O(n)$ considerando las $m-1$ distancias del elemento que **se elimina** y las $m-1$ que **se añaden**.
- Si $\Delta z_{MM}(Sel, i, j)$ es negativo ($\Delta z_{MM}(Sel, i, j) < 0$), la solución vecina Sel' es mejor que la actual Sel (**es un problema de minimización**) 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,
 - 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:

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

Repetir

$Sel' \leftarrow \text{GENERA_VECINO}(Sel);$

Hasta $(\Delta z_{MM}(Sel, i, j) < 0)$ **○**

(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 Euclideas 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 $\{0,9\}$ 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** (Duarte y Martí, 2007):
 - **Tipo a**: 40 matrices $n \times n$ con distancias enteras aleatorias en $\{0,10\}$: 20 con $n=500$ y $m=50$; y 20 con $n=2000$ y $m=200$
 - **Tipo b**: 40 matrices $n \times n$ con distancias reales aleatorias en $[0,1000]$: 20 con $n=500$ y $m=50$; y 20 con $n=2000$ y $m=200$
 - **Tipo c**: 20 matrices $n \times n$ con distancias enteras aleatorias en $\{0,1000\}$. $n=3000$ 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:

<https://grafo.etsii.urjc.es/optsicom/mindiff/>

- 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.
Referencias:
 - Duarte A., Sánchez-Oro J., Resende M.G.C., Glover F., Martí R. Greedy randomized search procedure with exterior path relinking for differential dispersion minimization. *Information Sciences*, (2016), 46-60.
 - Resende M.G.C., Werneck R.F.A hybrid heuristic for the p-median problem *Journal of Heuristics*, 10(1) (2016), 59-88
 - Lai X., Hao J-K, Glover, Fred, Yue D. Intensification-driven tabu search for the minimum differential dispersion problem. *Knowledge-Based System*, 5, January 2019.
 - Aringhieri, R., Cordone R., Grosso A. Construction and improvement algorithms for dispersion problems. *European Journal of Operational Research* 242 (2015). 21-33.

Metaheurísticas

Seminario 2. Problema para las prácticas: Aprendizaje de Pesos en Características

Problema del Aprendizaje de Pesos en Características (APC)

1. Definición del Problema de Clasificación. Clasificador k-NN.
2. Definición y Representación del Problema del Aprendizaje de Pesos en Características.
3. Solución Greedy.
4. Búsquedas por Trayectorias Simples.
5. Bases de Datos a Utilizar.

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

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

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

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



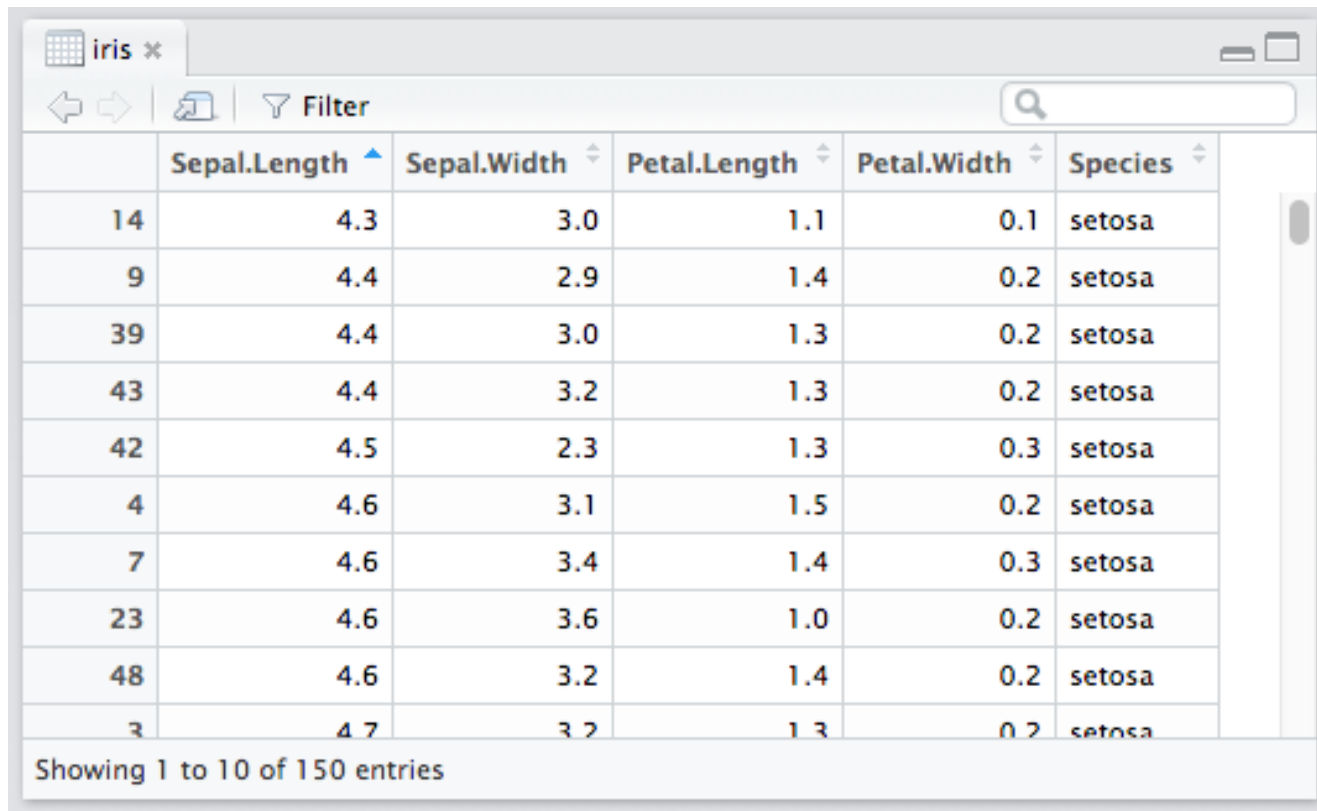
versicolor



virgínica

Echemos un vistazo a los datos de Iris

- Es un datasets muy sencillo, sólo 4 atributos.
- Para cada nueva orquídea busca las más parecidas y le asigna su clase.



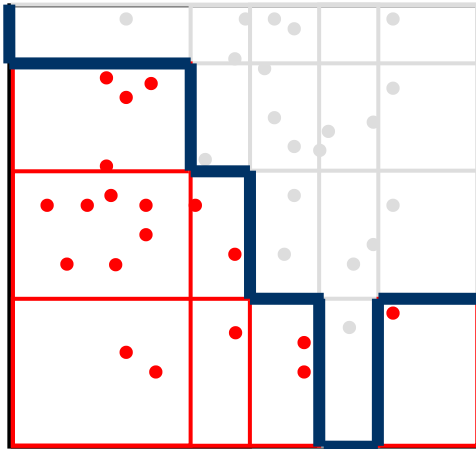
	Sepal.Length ▲	Sepal.Width ▼	Petal.Length ▼	Petal.Width ▼	Species ▼
14	4.3	3.0	1.1	0.1	setosa
9	4.4	2.9	1.4	0.2	setosa
39	4.4	3.0	1.3	0.2	setosa
43	4.4	3.2	1.3	0.2	setosa
42	4.5	2.3	1.3	0.3	setosa
4	4.6	3.1	1.5	0.2	setosa
7	4.6	3.4	1.4	0.3	setosa
23	4.6	3.6	1.0	0.2	setosa
48	4.6	3.2	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa

Showing 1 to 10 of 150 entries

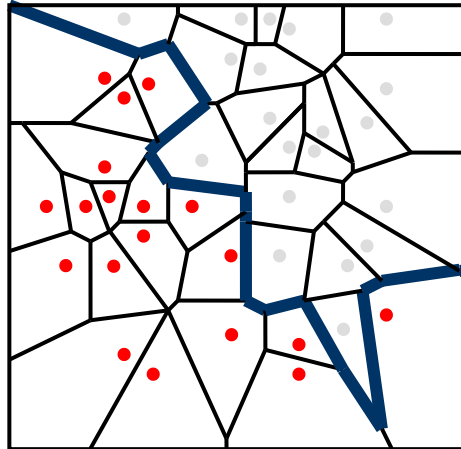
Definición del Problema de Clasificación

Ejemplos de clasificación sobre clases definidas

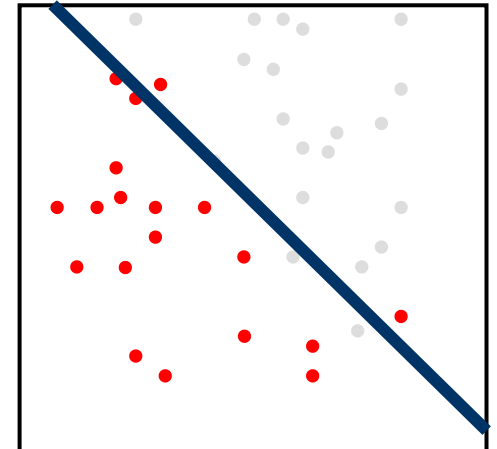
- *Reglas intervalares*



- *Basado en distancias*

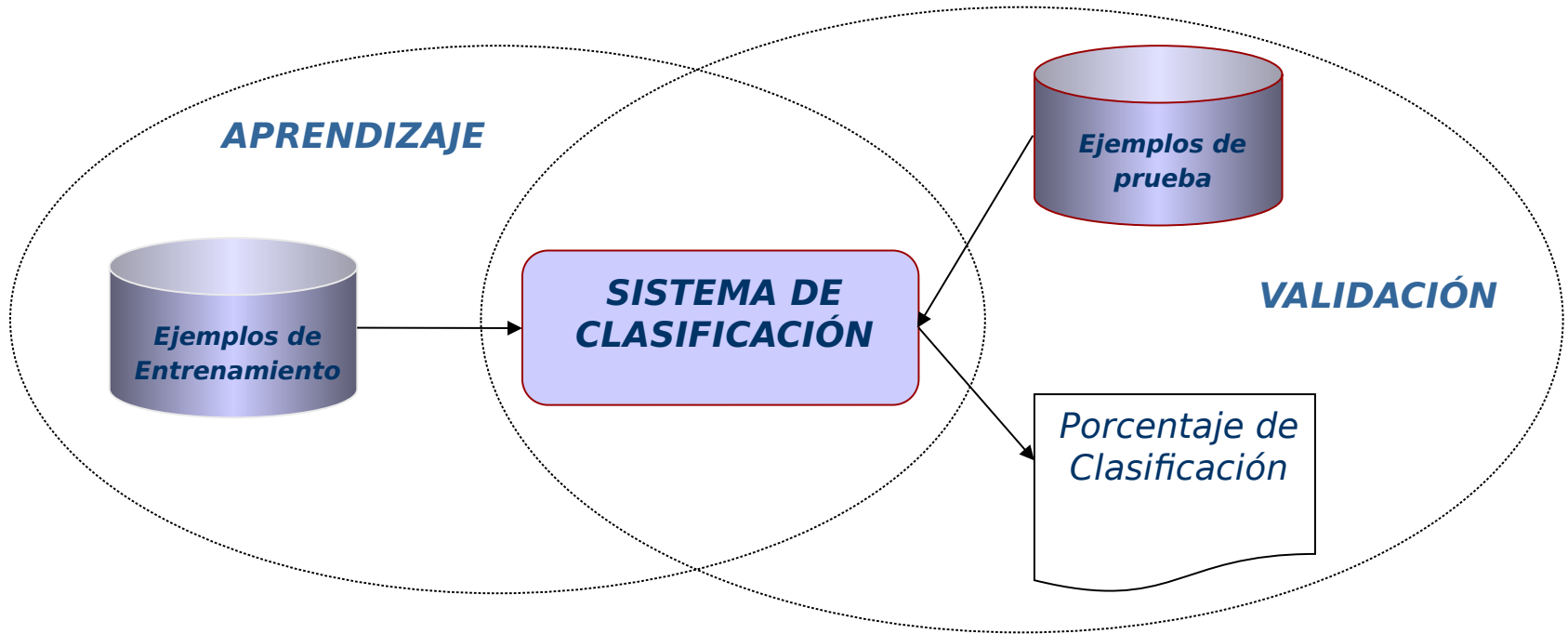


- *Clasificador lineal*



Definición del Problema de Clasificación

Esquema de aprendizaje en clasificación



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

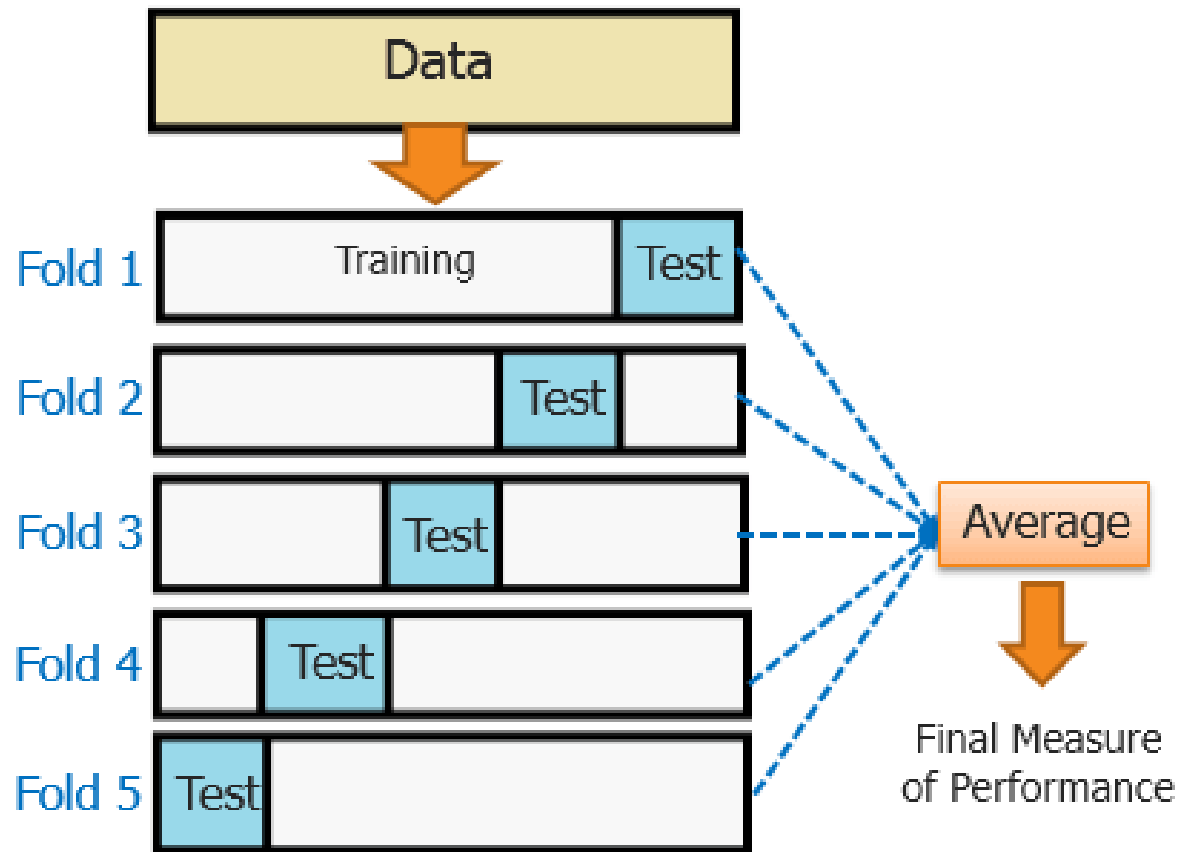
Para cada una, se diseña un clasificador distinto usando los ejemplos de entrenamiento y se valida con los de prueba

Definición del Problema de Clasificación

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 final del método de clasificación se medirá con la media de los 5 porcentajes de clasificación del conjunto de prueba

Definición del Problema de Clasificación



Clasificador k-NN

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

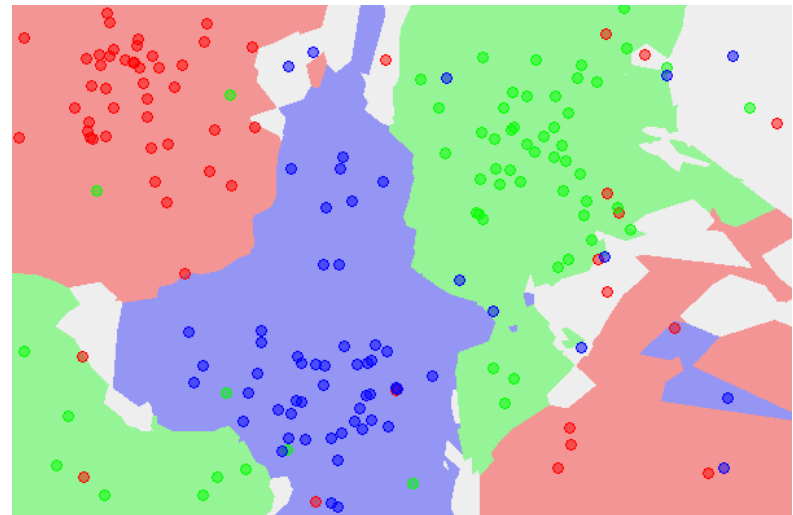
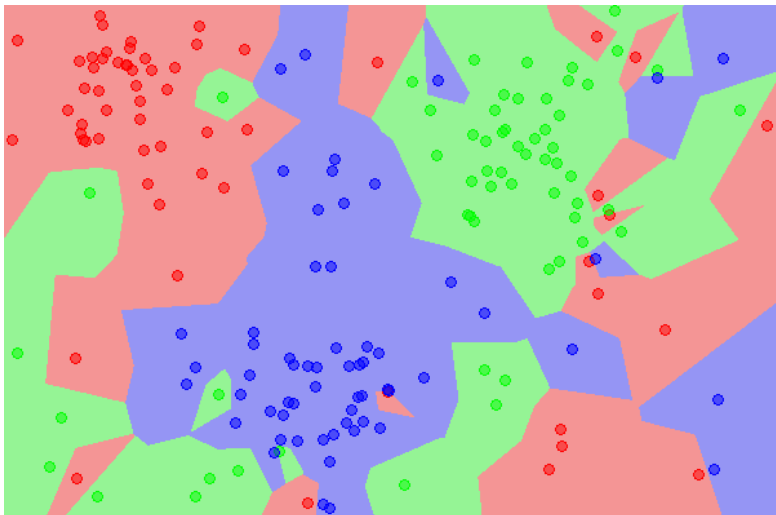
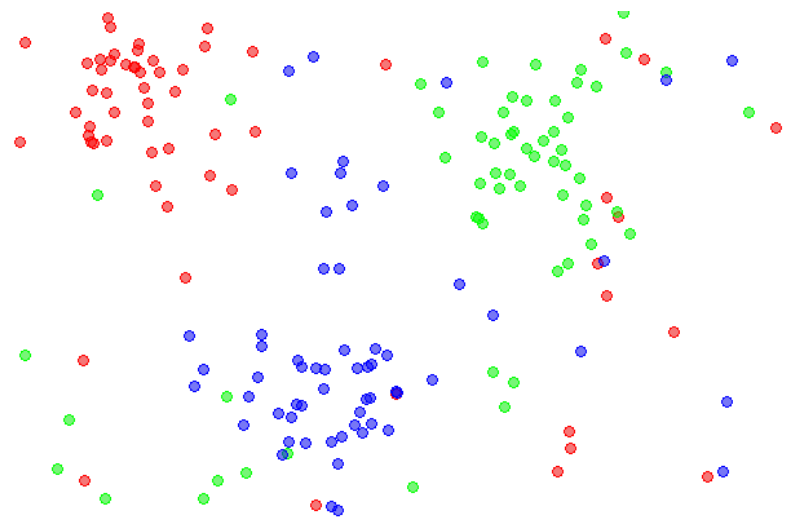
- i. El proceso de aprendizaje de este clasificador consiste en almacenar una tabla con los ejemplos disponibles, junto a la clase asociada a cada uno de ellos
- ii. Dado un nuevo ejemplo a clasificar, se calcula su distancia (usaremos la euclídea) a los n ejemplos existentes en la tabla y se escogen 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

Imagen derecha: Puntos, su posición (x,y) indican sus características, y el color indica su categoría.

Imagen inferior Izquierda: Clasificación mediante el 1-NN.

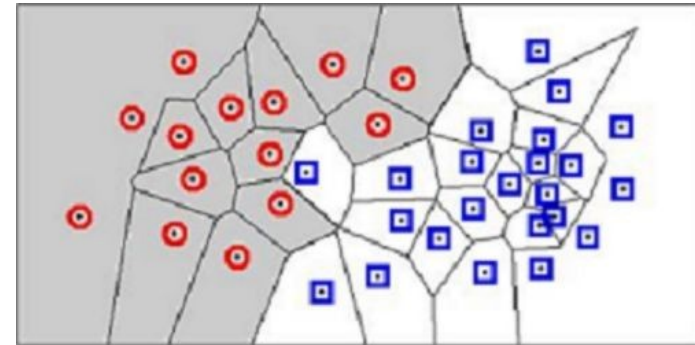
Imagen inferior derecha: Clasificación mediante el 5-NN.



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
 - 4) $d = d(e_i, e')$
 - 5) Si $(d < d_{min})$
 - 6) Entonces $c_{min} = \text{clase}(e_i)$, $d_{min} = d$
- 7) 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

$$d_e(e_1, e_2) = \sqrt{\left(\sum_i (e_1^i - e_2^i)^2\right)}$$

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 ejemplo 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:
(todos los pesos iguales) x_q : 0.7 0.2 0.1

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

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

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

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

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

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:

$$d_e(e_1, e_2) = \sqrt{\left(\sum_i w_i * (e_1^i - e_2^i)^2\right)}$$

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***):

$$tasa_clas = 100 \cdot \frac{\text{n}^\circ \text{ instancias bien clasificadas de } T}{\text{n}^\circ \text{ instancias en } T}$$

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 0.1 en w_i para considerar que se descarta una característica:

$$tasa_red = 100 \cdot \frac{\text{n}^\circ \text{ valores } w_i < 0.1}{\text{n}^\circ \text{ características}}$$

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:

$$F(W) = \alpha \cdot \text{tasaclas}(W) + (1 - \alpha) \cdot \text{tasared}(W)$$

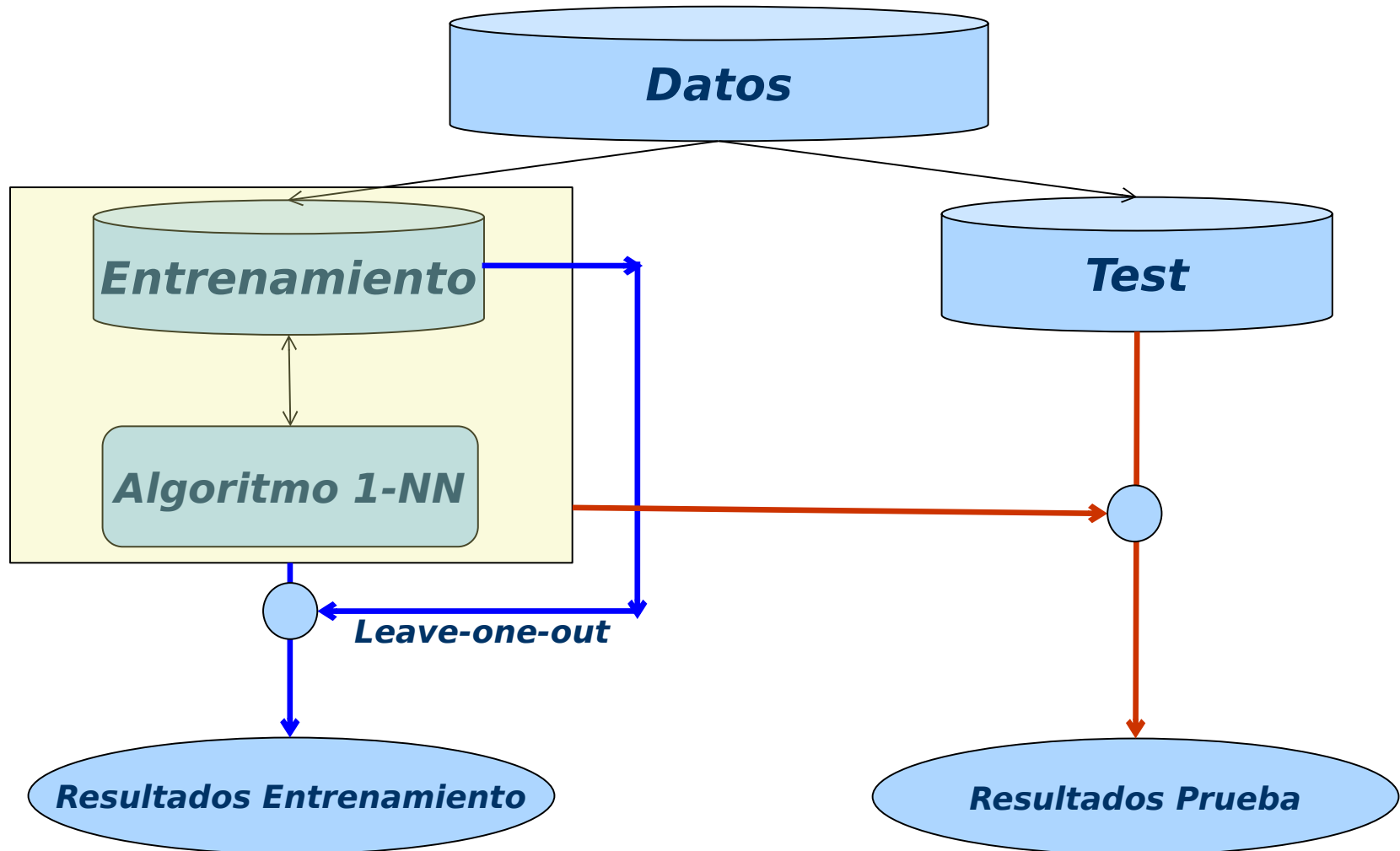
- El valor de α pondera la importancia entre el acierto y la reducción de características de la solución encontrada (el clasificador generado). Usaremos $\alpha = 0.5$, dando la misma importancia a ambos
- El objetivo es obtener el conjunto de pesos W que maximiza esta función, es decir, que maximice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

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

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 0.1 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:

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

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

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

$w_m = \text{máximo}(W)$

Para cada w_i en W

 Si $w_i < 0$ entonces

$w_i = 0$

 Si no

$w_i = w_i / w_m$

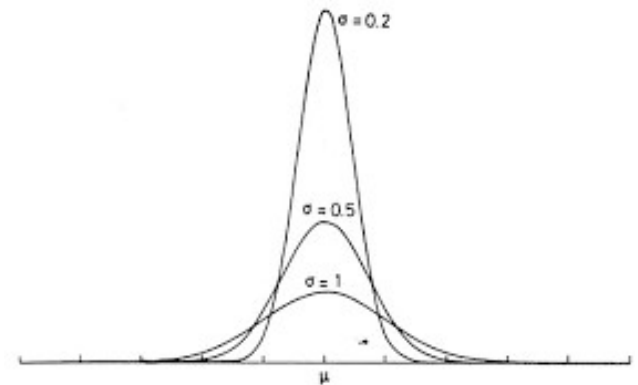
Devolver W

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 σ :

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

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



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 allá 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: Ionosphere, Parkinsons y Spectf-heart

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

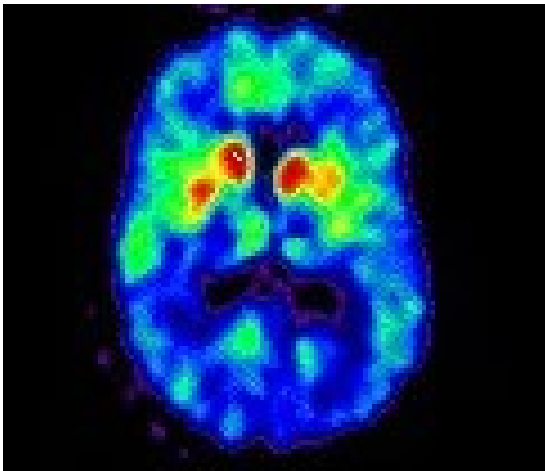


- § Consta de 352 ejemplos
- § Consta de 34 atributos
- § Consta de 2 clases (retornos buenos y malos).
- § Atributos: Señales procesadas.

Bases de Datos a Utilizar

Trabajaremos con tres bases de datos: Ionosphere, Parkinsons y Spectf-heart

Parkinsons contiene datos que se utilizan para distinguir entre la presencia y la ausencia de la enfermedad de Parkinson en una serie de pacientes a partir de medidas biomédicas de la voz

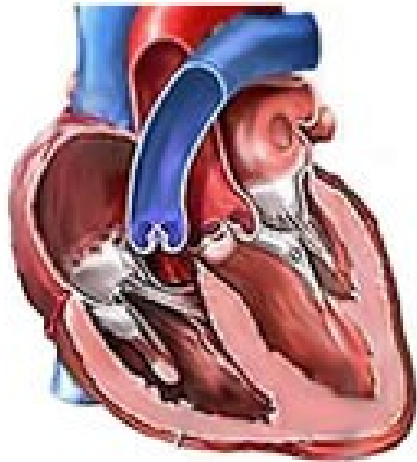


- Consta de 195 ejemplos
- Consta de 23 atributos (clase incluida)
- Consta de 2 clases. La distribución de ejemplos está desbalanceada (147 enfermos, 48 sanos)
- Atributos: Frecuencia mínima, máxima y media de la voz, medidas absolutas y porcentuales de variación de la voz, medidas de ratio de ruido en las componentes tonales, ...

Bases de Datos a Utilizar

Trabajaremos con tres bases de datos: Ionosphere, Parkinsons y Spectf-heart

Spectf-heart contiene atributos calculados a partir de imágenes médicas de tomografía computerizada (SPECT) del corazón de pacientes humanos. La tarea consiste en determinar si la fisiología del corazón analizado es correcta o no



- Consta de 267 ejemplos
- Consta de 45 atributos enteros (clase incluida)
- Consta de 2 clases (paciente sano o patología cardiaca)
- Atributos: 2 características (en reposo y en esfuerzo) de 22 regiones de interés de las imágenes