Técnicas Supervisadas
Aproximación no paramétrica

2016

- *Notas basadas en el curso Reconocimiento de Formas de F.Cortijo, Univ. de Granada*
- *Pattern Classification de Duda, Hart y Storck*
- *The Elements of Statistical Learning de Hastie, Tibshirani y Friedman*
- *Parte del material se extrajo de las notas:Técnicas Supervisadas II: Aproximación no paramétrica de F.Cortijo, Univ. de Granada*

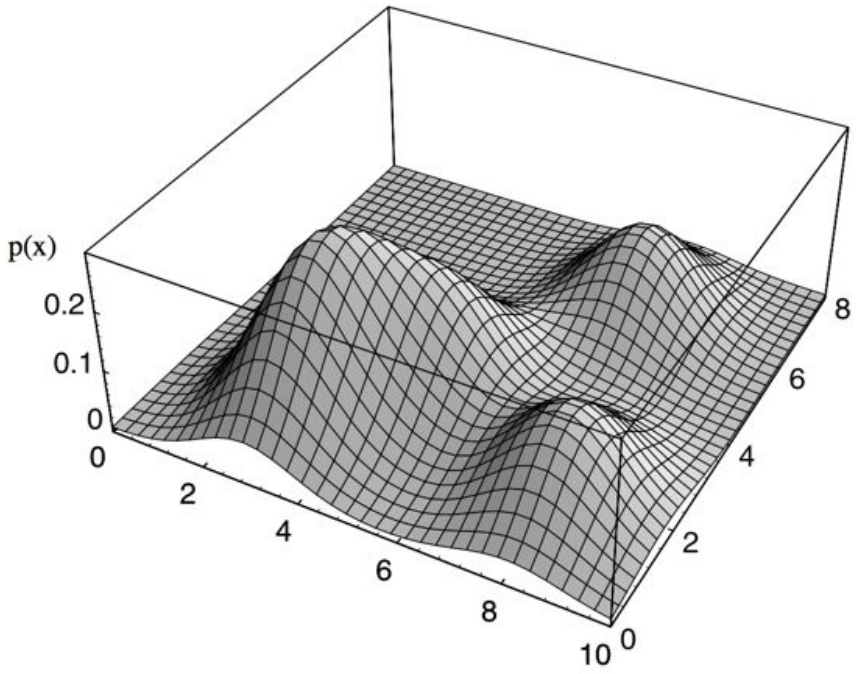
Contenido

- Estimación no paramétrica de la función de densidad
  - Estimadores de Parzen
  - Estimación mediante los k-vecinos más próximos

- Estrategias para disminuir el conjunto de referencia (M<<N):
  - EDICIÓN,
  - CONDENSADO,
  - APRENDIZAJE ADAPTIVO (LVQ-DSM)

Estimación de densidades no paramétricas

- Formas paramétricas raramente ajustan densidades encontradas en la práctica (multimodales)



Estimación de densidades

$$P \text{ probabilidad de } \mathbf{x} \in R : \quad P = \int_R p(x')dx'$$

Estimación de densidades

Si $p(\mathbf{x})$ continua en R y R tan pequeña que $p(\mathbf{x}) = cte$

$P = \int_R p(\mathbf{x}')dx' \approx p(\mathbf{x})V$ con V volumen envuelve a R

$$p(\mathbf{x}) \approx \frac{k/n}{V}$$

si fijo V y $n \to \infty$ $\frac{P}{V} = \frac{\int_R p(\mathbf{x}')d\mathbf{x}'}{\int_R d\mathbf{x}'}$ versión promediada de $p(\mathbf{x})$

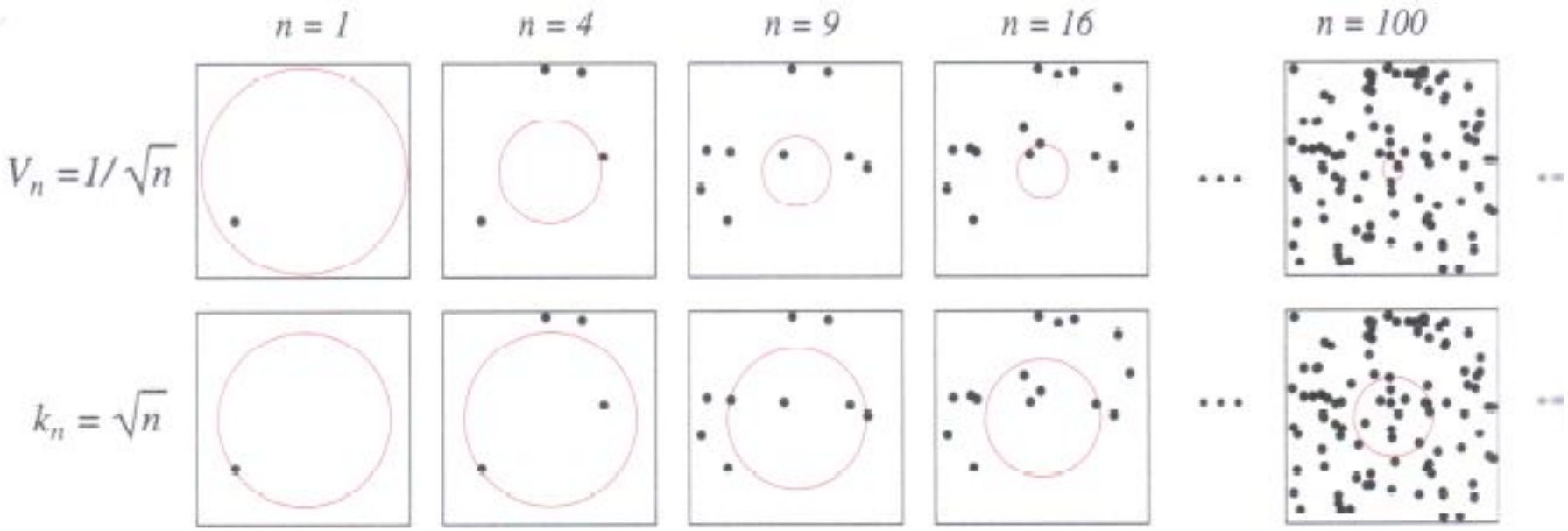
Estimación de densidades

$$p_n(\mathbf{x}) = \frac{k_n/n}{V_n}$$

- Dado un conjunto de n muestras existen 2 formas de analizar la estimación de p(x):
  - Fijar V_n , por ejemplo V_n = 1/sqrt(n) y determinar k_n empíricamente (Ventanas de Parzen)
  - Fijar k_n, por ejemplo k_n = sqrt(n) y determinar V_n empíricamente para que k_n muestras esten en V_n (Estimación de los k-vecinos más cercanos.)
- Ambos métodos convergen al valor verdadero de p(**x**) (respetando hipótesis claro) aunque es difícil hacer aseveraciones del comportamiento para *n* finito.

Ventanas de Parzen vs k- vecinos

Ejemplo: Dos métodos para estimar la densidad en el centro del cuadrado



Duda Hart - Stork

Estimación por ventanas de Parzen

$$p_n(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{V_n}\varphi\left(\frac{\mathbf{x}-\mathbf{x}_i}{h_n}\right)$$

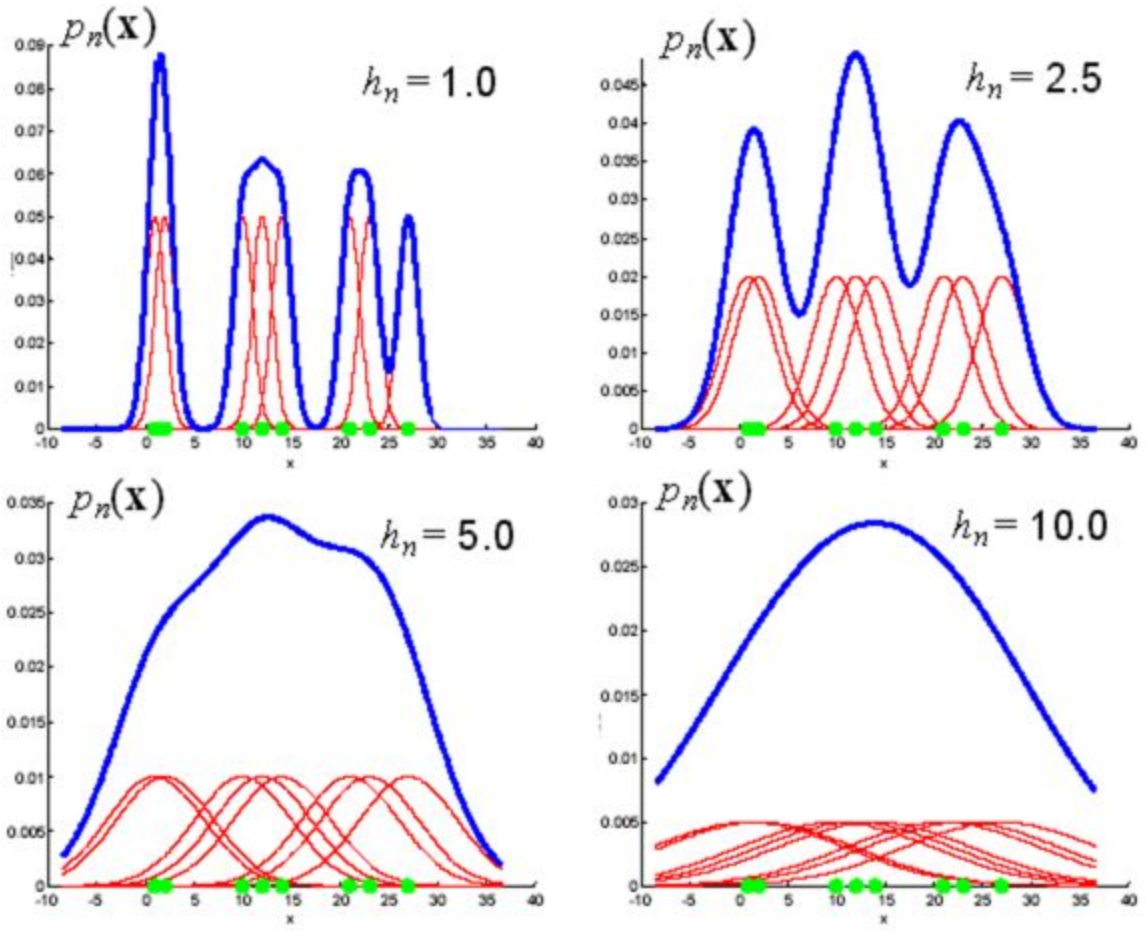
- Esta ecuación sugiere una manera más general para estimar funciones de densidad definiendo una función de ventana
- La ventana $\varphi(\mathbf{u})$ cumple un rol de *interpolador*

- Qué necesitamos para que $p_n(x)$ sea una densidad de probabilidad? Si

$$\varphi(\mathbf{x}) \geq 0 \qquad \int \varphi(\mathbf{u})\, d\mathbf{u} = 1,$$

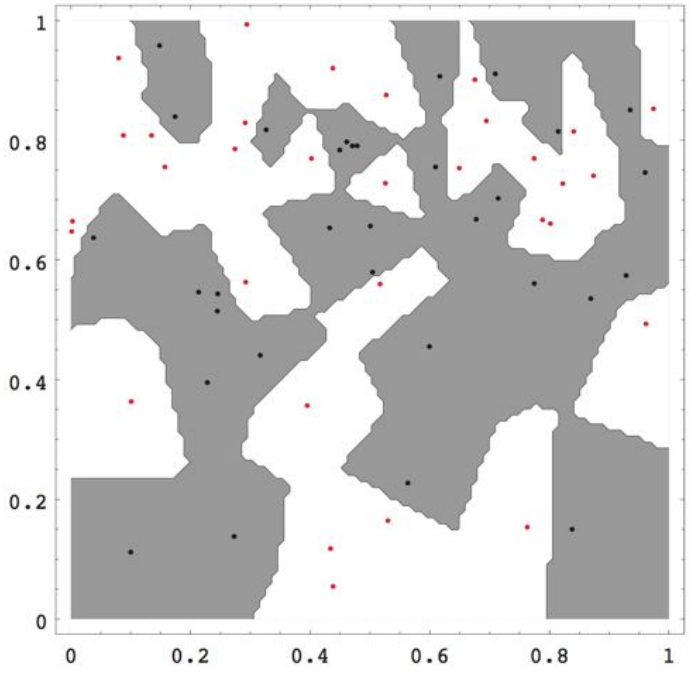
y además se mantiene que $V_n = h_n^d$ entonces $p_n(x)$ hereda estas condiciones.

Estimación por ventanas de Parzen

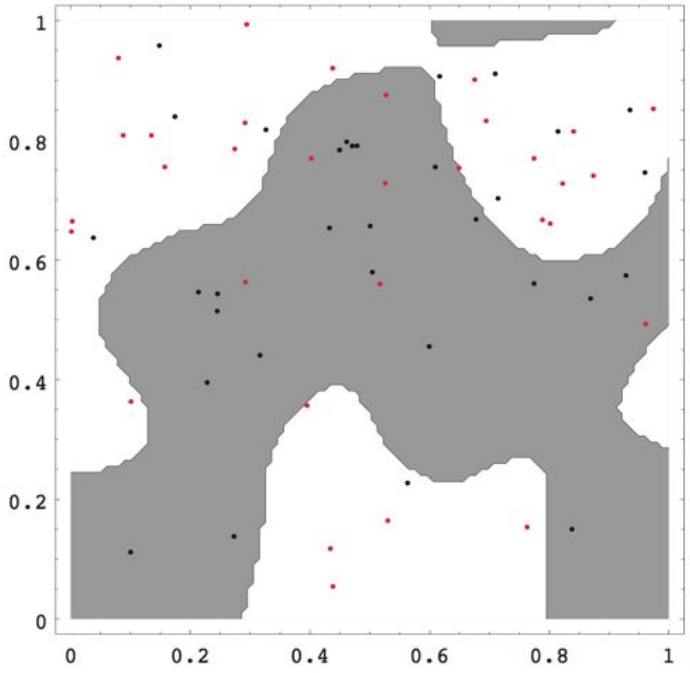


Ejemplo Clasificación

h pequeño



h grande

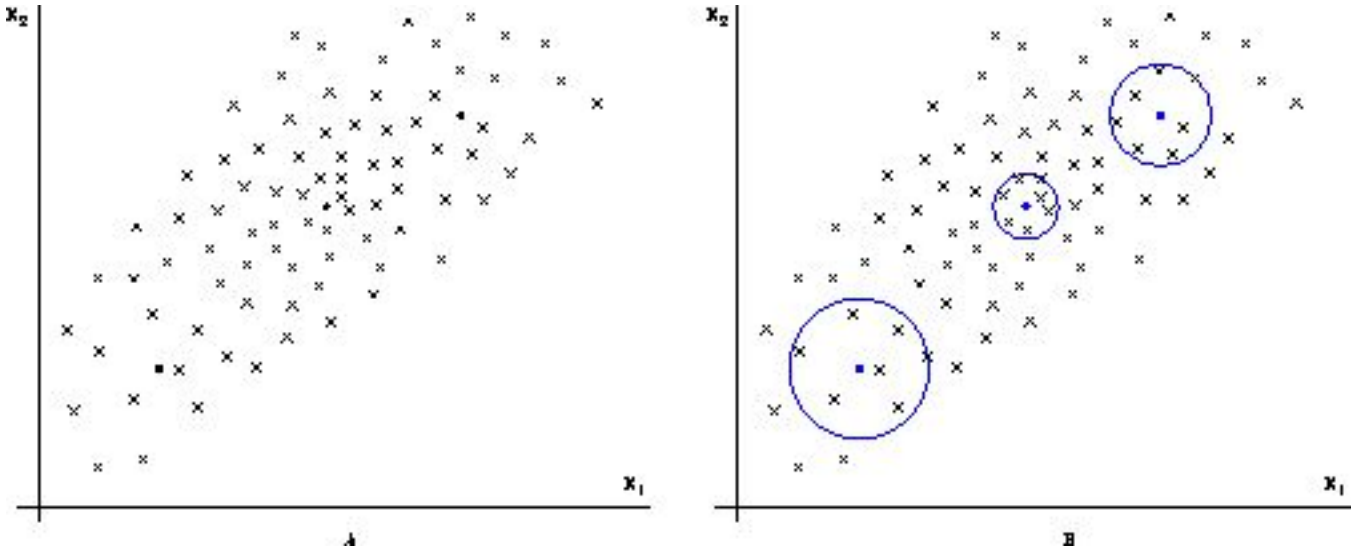


- Se puede ajustar el ancho de la ventana mediante validación cruzada.
- Se parte el conjunto de muestras disponibles en dos conjuntos y se entrena con uno y se calcula el error con el otro.
- Se busca el ancho que minimiza el error de clasificación en validación.

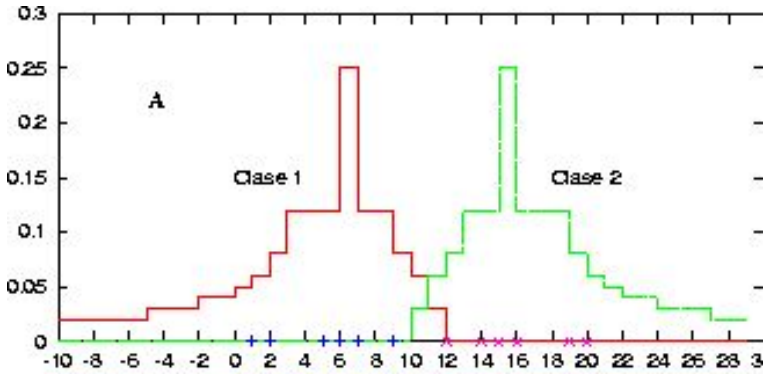
Estimación mediante los k vecinos más próximos

- Si la densidad p(x) es muy alta en x, entonces la celda será pequeña (buena resolución)
- Si la densidad p(x) es muy baja en x, entonces la celda será grande.

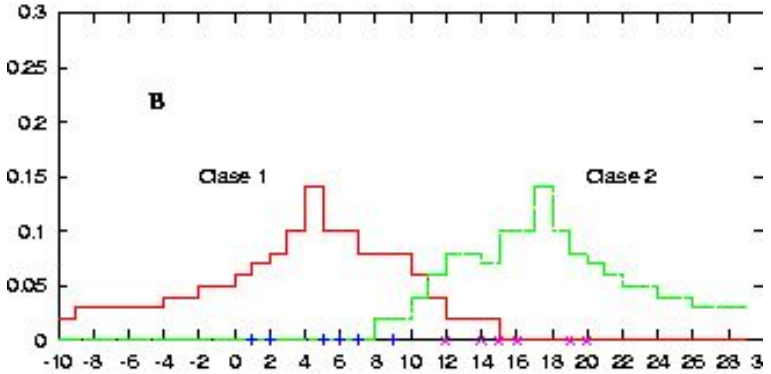
$$p_n(\mathbf{x}) = \frac{k_n/n}{V_n}$$



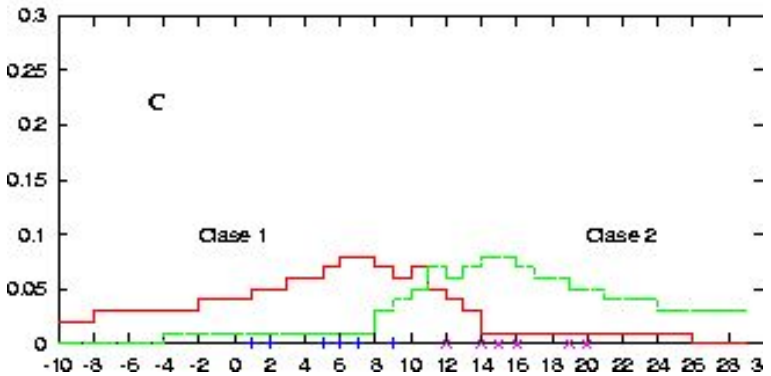
k-vecinos *k*=3



k-vecinos *k*=5



k-vecinos *k*=7



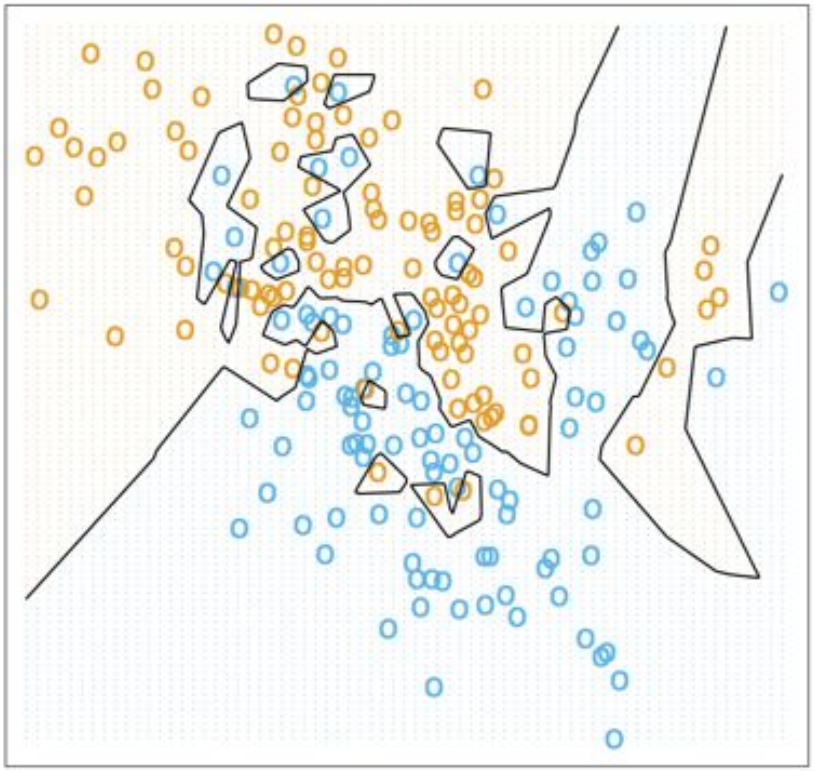
Regla del vecino más cercano 1-NN

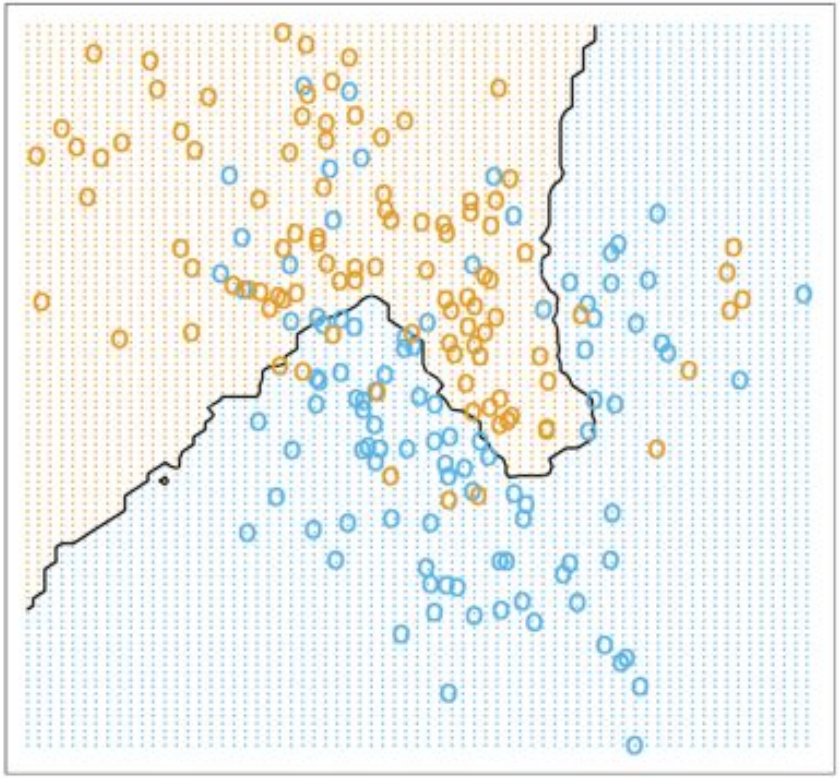
- Se selecciona w_j si:

$$d(x, x_{NN}) = \min_{i=1..n} \{\delta(x, x_i)\}$$

$$x_{NN} \in w_j$$

- Interpretación:
  - El espacio es dividido en n celdas (regiones de Voronoi)


1−Nearest Neighbor Classifier


15-Nearest Neighbor Classifier

Curse of Dimensionality
(La maldición de la dimensionalidad)

- Supongamos que tiro al azar (uniformemente) N puntos en la Esfera p dimensional de radio 1.
- Sea d la distancia del origen al punto más cercano.
- Obs. d es un variable aleatoria.
- Cuál es su mediana?

Nota la mediana de d es d_m si $P(d<=d_m) = P(d>=d_m) = 1/2$

Curse of Dimensionality
(La maldición de la dimensionalidad)

Sol.

- Si tengo un solo punto, la probabilidad $P_1(d>x)$ es igual a la probabilidad de que el punto esté fuera de la bola de radio x, es decir $P_1(d>x) = 1 - x^p$
- Si tengo N puntos necesito que todos estén fuera de la bola por lo que $P_N(d>x) = (1-x^p)^N$
- La mediana es tal que $P_N(d>d_m) = (1-d_m^{\ p})^N = ½$,

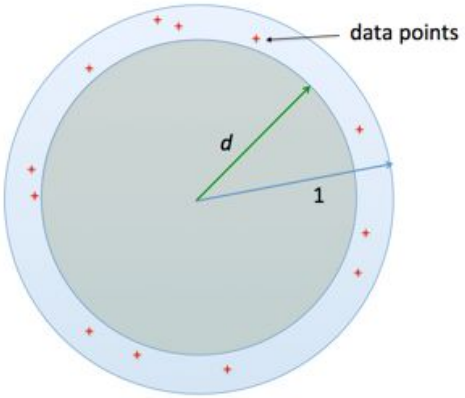
  por lo que $d_m = (1- ½^{1/N})^{1/P}$

Curse of Dimensionality
(La maldición de la dimensionalidad)

- Mediana de distancia a vecino más cercano $d_m = (1 - ½^{1/N})^{1/P}$
- Si N=500, p=10, $d_m = 0.52$!

Curse of Dimensionality
(La maldición de la dimensionalidad)

- mediana de distancia a vecino más cercano $d_m = (1 - ½^{1/N})^{1/P}$
- Si N=500, p=10, $d_m = 0.52$!
- Distancia del origen al punto más cercano es más de la mitad de la distancia a la frontera.
- Todos los puntos están apretados en la frontera.
- Todos los puntos están lejos



Costo Computacional k-NN

- Costo computacional asociado a las reglas 1-NN y *k*-NN es equiparable.
- Búsqueda de *k* vecinos, se debe mantener estructura auxiliar que mantenga ordenados los *k* vecinos encontrados
- Requiere explorar todo el conjunto de referencia O(n).
- Cálculo de la distancia euclídea lineal con d O(nd) o Mahalanobis distance $O(nd^2)$
- Espacio de almacenamiento de todos los prototipos O(nd)
- **Inaplicable** si tengo un conjunto de referencia grande y alta dimensionalidad.

Estrategia:

- Selección de características: para bajar **d**
- Disminuir el conjunto de referencia (**M<<N**):
  - EDICIÓN,
  - CONDENSADO,
  - APRENDIZAJE ADAPTIVO
- Mejorar la eficiencia del cálculo del vecino más cercano: ordenar, métodos jerárquicos.

Edición del conjunto de entrenamiento

Objetivo:

- Reducir el conjunto de referencia
- Mejorar la “calidad” del mismo
- Eliminar outliers → Aumentar desempeño de 1-NN

Edición de Wilson: EW (S, k)

Entradas

$S \in A$ Conjunto de prototipos original.

$k \in \mathbb{N}$ Numero de vecinos.

Salidas

$S_{EW} \subseteq S$ Conjunto de prototipos editado de S.

Algoritmo

$S_{EW} \leftarrow S$

Para todo $Z \in S$ hacer

Si (**Dif** $(Z, S_{EW} - \{Z\}, k)$) entonces

Marcar Z para su eliminacion. { *Edicion* }

Fin-si

Fin-para

Eliminar de S_{EW} los prototipos marcados.

Edición de Wilson

- Reduce cantidad de prototipos problemáticos (outliers)
- Edición con la regla k-nn
- Genera dependencia estadística entre los prototipos retenidos (validación cruzada - leave one out)

Edición por particiones

- Modificación del algoritmo de Wilson para garantizar independencia estadística entre prototipos retenidos
- Se realiza una partición del conjunto de entrenamiento (mínimo 3 subconjuntos)
- Aplico la regla *k*-NN a cada prototipo pero considerando los vecinos de una partición particular distinta a la del prototipo.
- Desempeño depende:
  - Paso de difusión (número de particiones)
  - Partición inicial
  - Regla *k*-NN (cantidad de vecinos)

Edición por particiones: EP (S, m, k)

Entradas

$S \in A$	Conjunto de entrenamiento original.
$m \in \mathbb{N}$	Numero de bloques ($m > 2$)
$k \in \mathbb{N}$	Numero de vecinos.

Salidas

$S_{EP} \subseteq S$	Conjunto de entrenamiento editado de S.

Auxiliares

T: vector de A	Partición de un conjunto.

Algoritmo

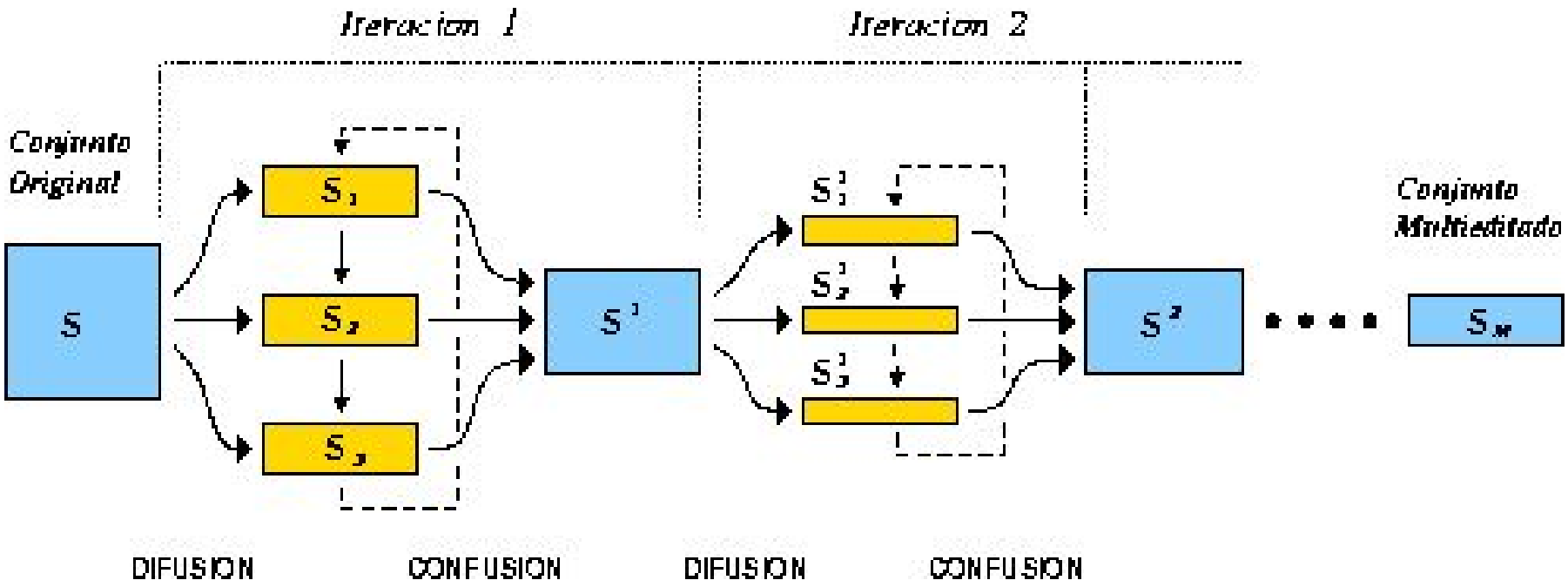
$T \leftarrow$ **Particion** (S, m) { *Difusion* }
Para $i \leftarrow 1, 2, \ldots, m$ { *Clasificacion* }
  Para todo $Z \in T_i$ hacer
    Si (**Dif** $(Z, T_{(i \bmod m)+1}, k)$) entonces
      Marcar Z para su eliminacion. { *Edicion* }
    Fin-si
  Fin-para
Fin-para
Eliminar los prototipos marcados en $T_1, T_2, \ldots, T_m$.
$S_{EP} = T_1 \cup T_2 \cup \ldots \cup T_m$ { *Confusion* }

Edición por particiones

- **Difusión:** Cada conjunto $\mathbf{T_i}$ generado por la partición de S (muestra aleatoria representativa del conjunto original). m>2
- **Clasificación:** Se clasifica cada prototipo usando como referencia otro conjunto diferente. Rotación de índices evita interacción mutua entre cada pareja de conjuntos de una partición (elimina la dependencia estadística)
- Ejemplo: m = 3, prototipos de T1 se clasifican con T2, los de T2 usando T3 y los de T3 usando T1.
- **Edición**: Prototipo mal clasificado se eliminan al final (no en cada iteración) - Igual que en la edición de Wilson.
- **Confusión**: Conjunto de prototipos resultante contiene todos los que han sido correctamente clasificados.

Multiedición

- Para evitar la dependencia respecto a la partición inicial se aplica en forma iterativa el método de edición por particiones.
- En cada iteración se genera una nueva partición del conjunto de prototipos que se mantienen sin editar.
- Edición con la regla 1-NN. Al hacerlo iterativo y en distintas particiones no es necesario usar k-vecinos.


Iteracion 1
Iteracion 2
Conjunto Original
S
S1
S2
S3
S'
S'1
S'2
S'3
S''
Conjunto Multieditado
Sm
DIFUSION
CONFUSION
DIFUSION
CONFUSION

Multiedición: ME (S, m, I)

Entradas

$S \in A$	Conjunto de entrenamiento original.
$m \in \mathbb{N}$	Numero de bloques ($m > 2$)
$I \in \mathbb{N}$	Criterio de finalizacion (Num. de it. sin descartes).

Salidas

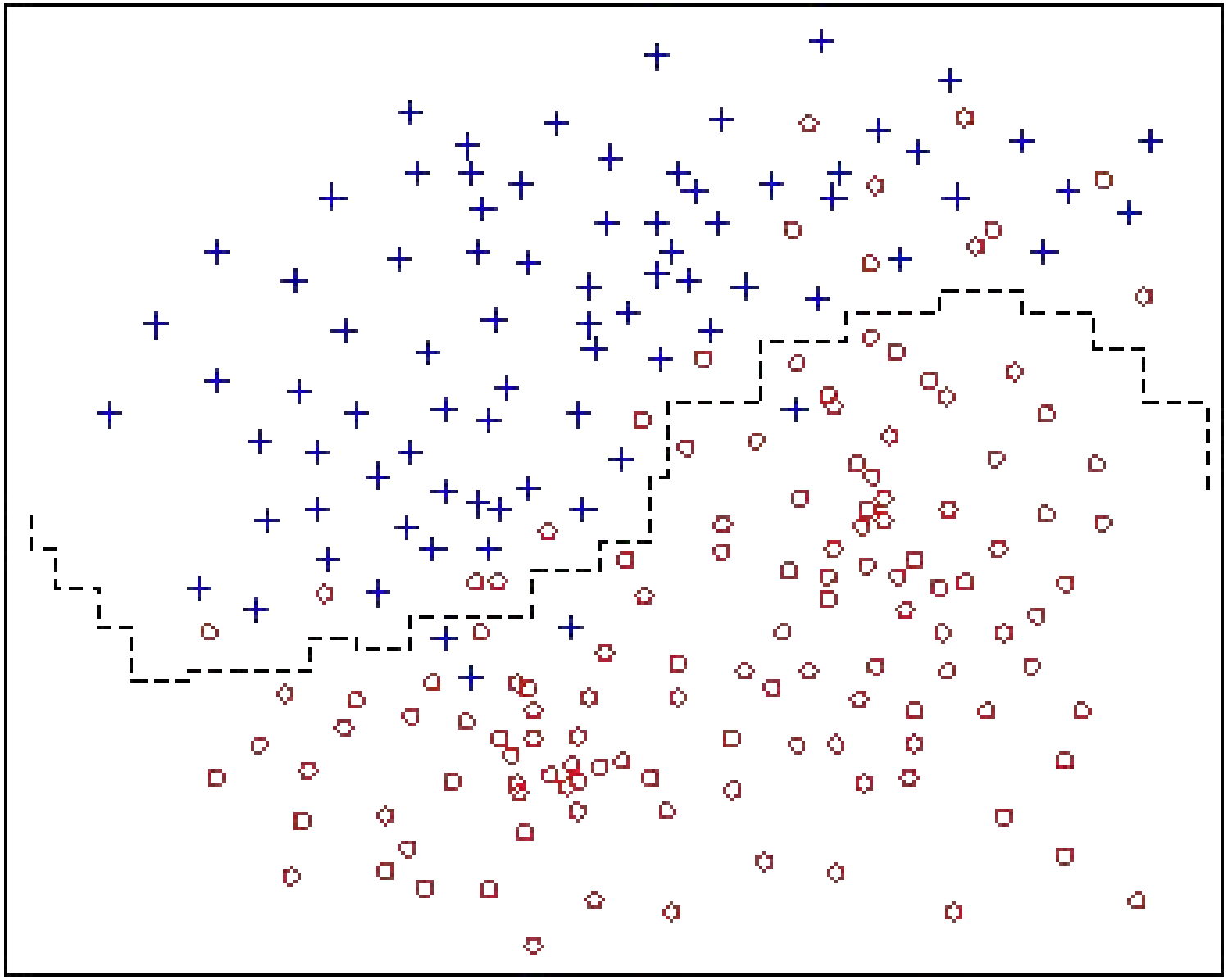
$S_M \subseteq S$	Conjunto de entrenamiento editado de S.

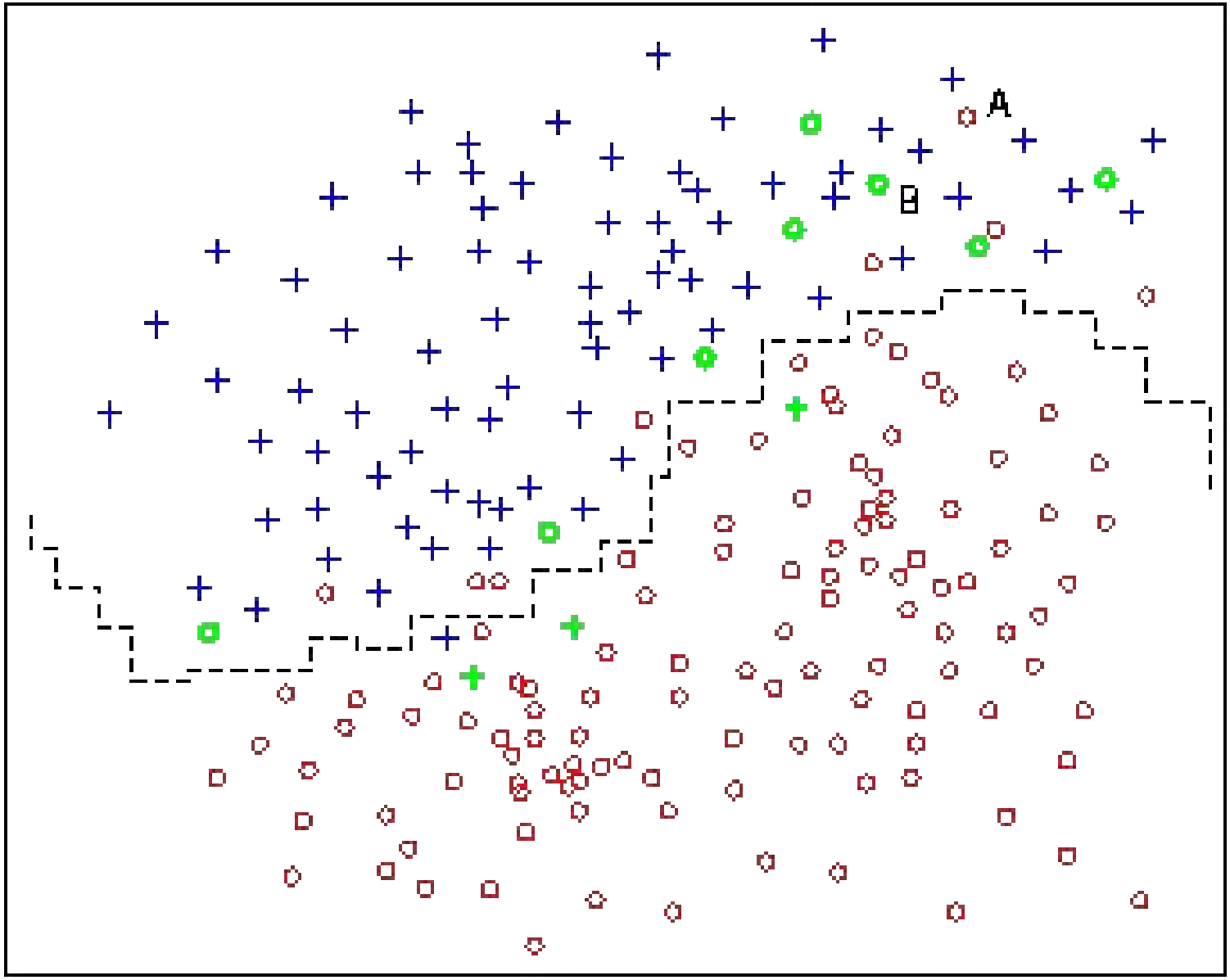
Auxiliares

$S_{EP} \subseteq S$	Conjunto de muestras auxiliar.
$i \in \mathbb{N}$	Contador de numero de iteraciones sin descartes.

Algoritmo

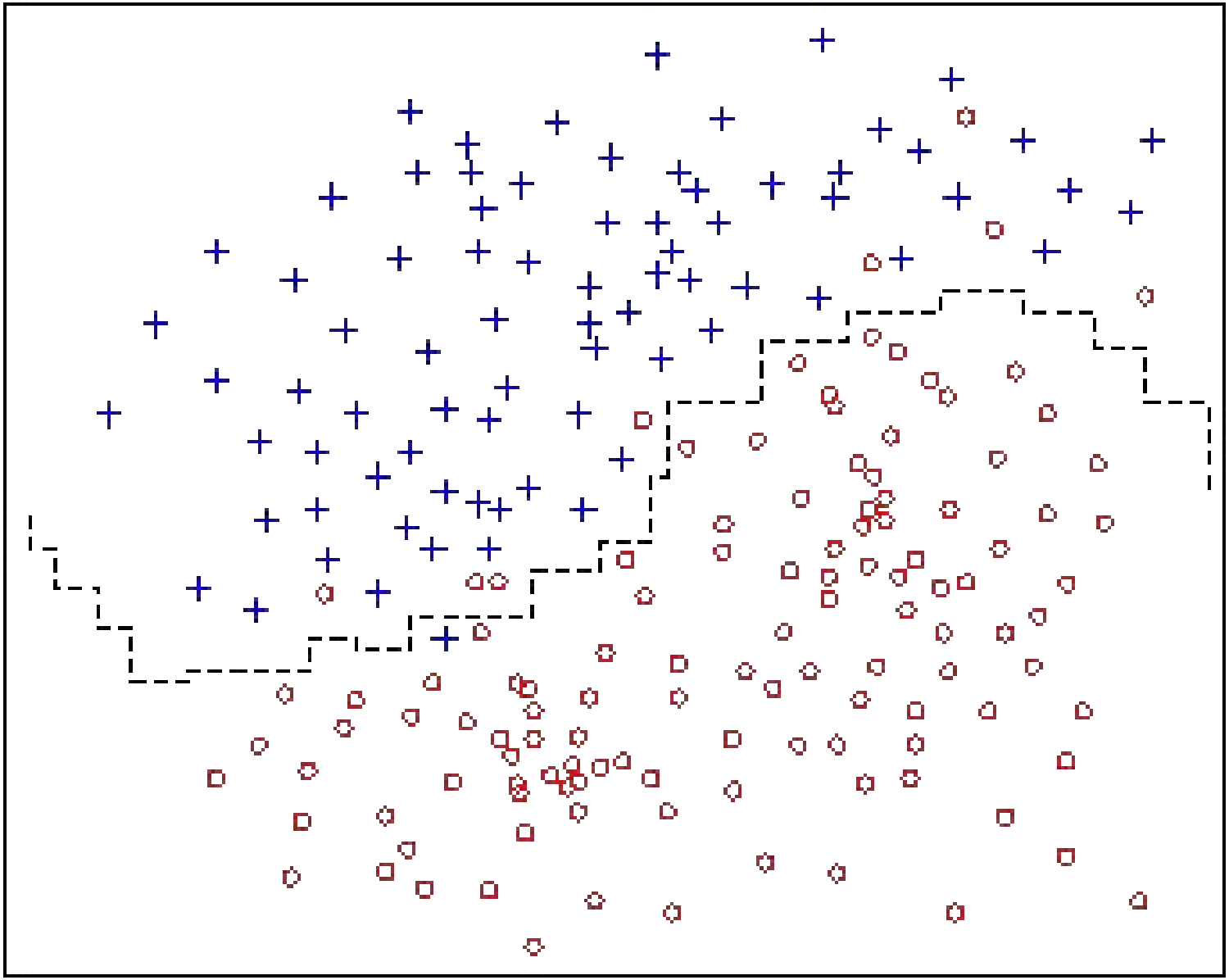
$S_M \leftarrow S$
$i \leftarrow 0$
Repetir
  $S_{EP} \leftarrow \mathbf{EP}\ (S_M, m, 1)$ {*Edicion por particiones*}
  Si $(|S_M| = |S_{EP}|)$ **entonces** {*no hay descartes*}
    $i \leftarrow i + 1$
  Si-no {*hay descartes*}
    $S_M \leftarrow S_{EP}$
    $i \leftarrow 0$
  Fin-si
Hasta $(i = I)$



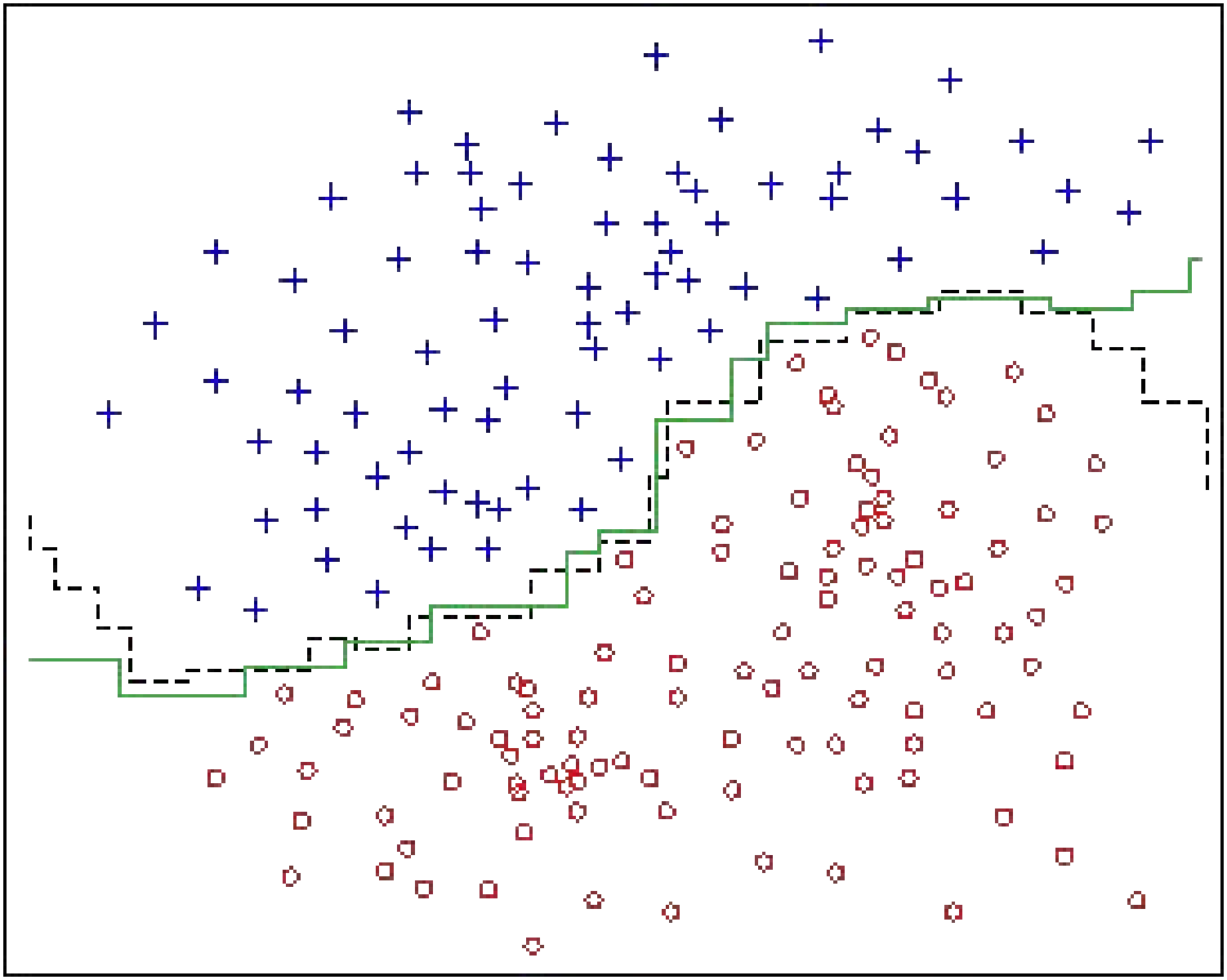


Prototipos marcados para eliminación mediante multiedición y 1nn

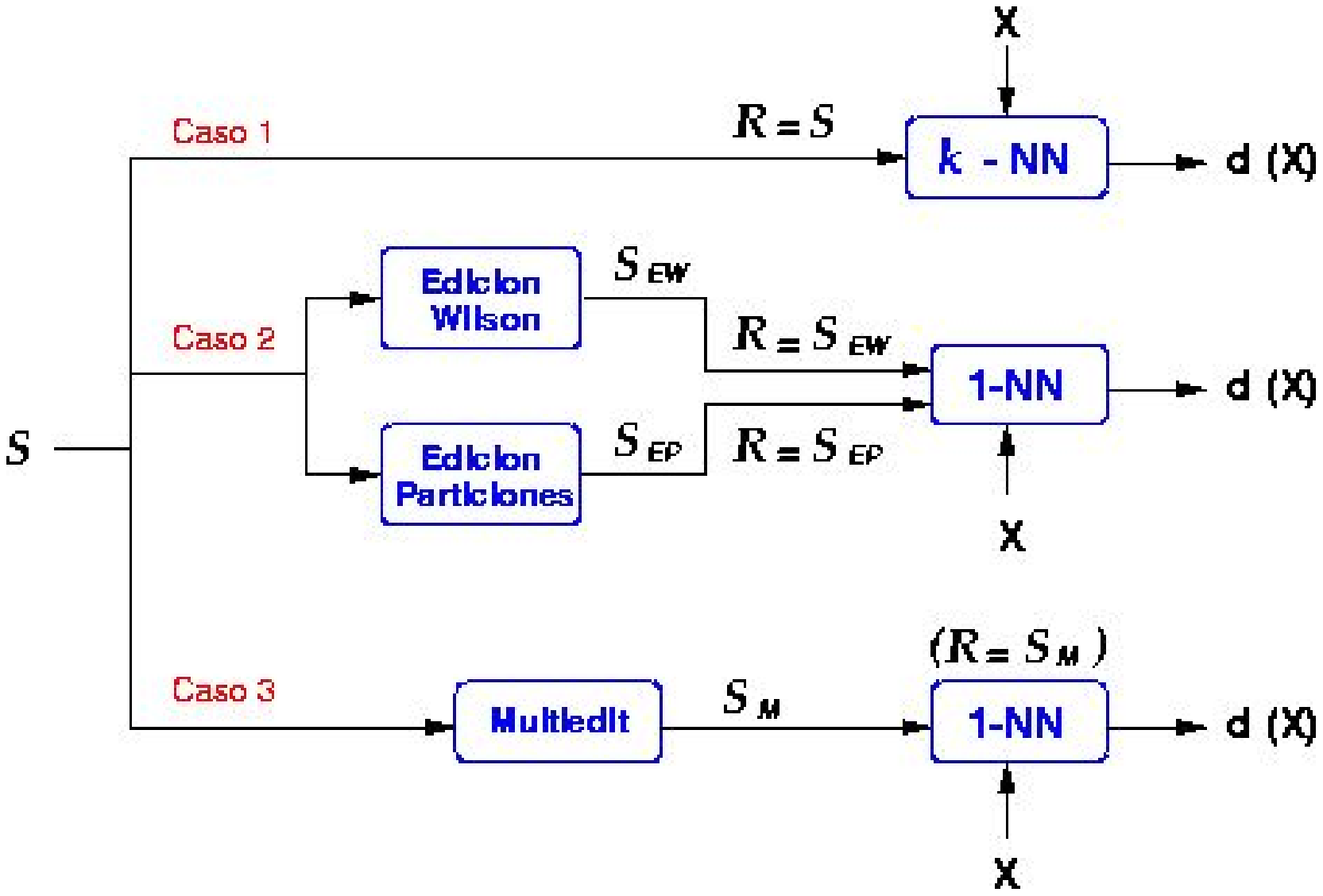
"A" es un prototipo malo pero no fue marcado para eliminar (particiones aleatorias)



Resultado luego de la primera iteración.



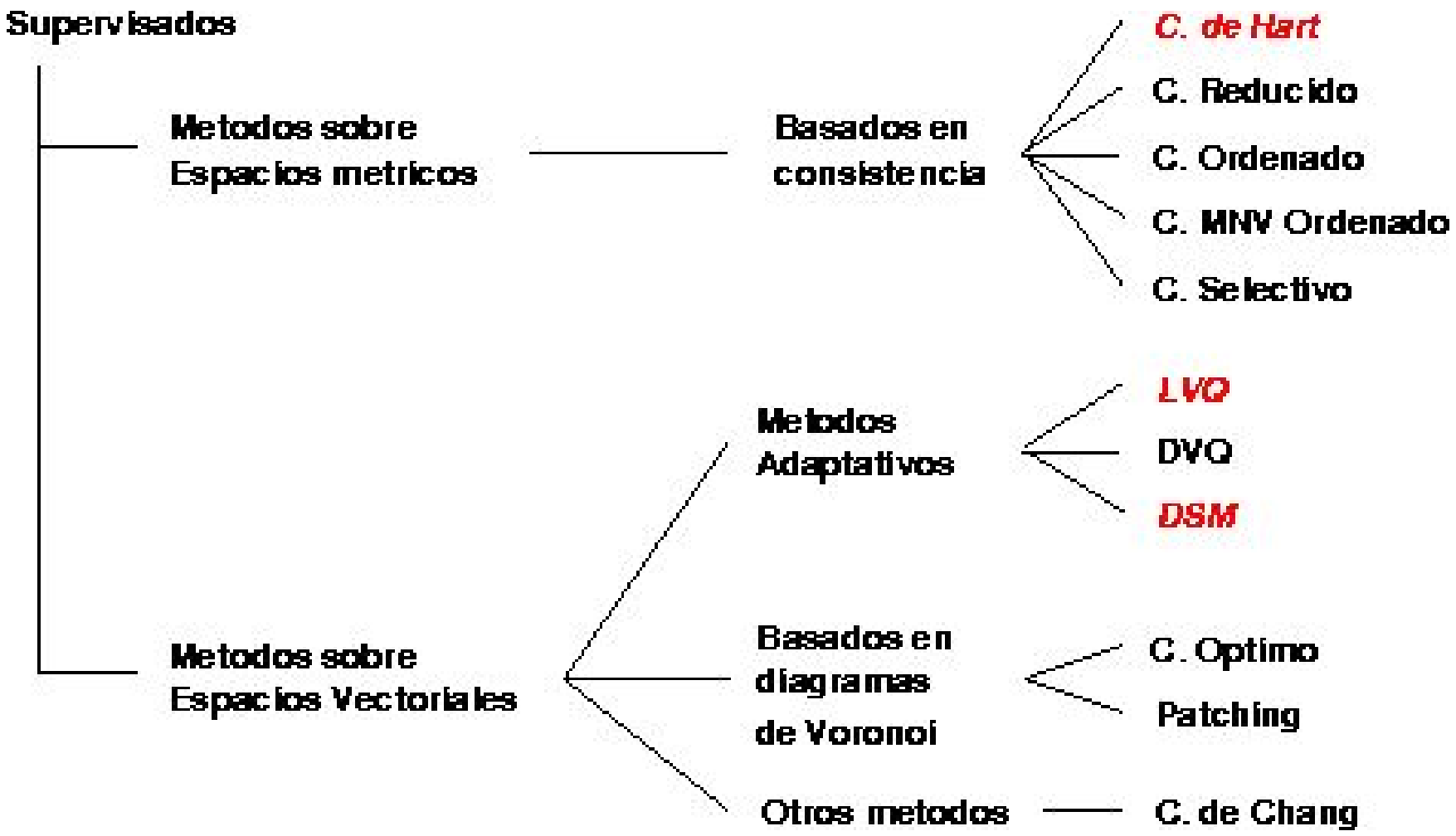
Resultado al finalizar las iteraciones (se eliminaron los outliers)


X
Caso 1
R = S
k - NN
d (X)
Edición Wilson
S EW
Caso 2
R = S EW
1-NN
d (X)
S
Edición Particiones
S EP
R = S EP
X
(R = S M)
Caso 3
Multiedit
S M
1-NN
d (X)
X

Reducción del coste computacional para los métodos del vecino más cercano

- **Objetivo**:
  - Incrementar eficacia computacional mediante la selección de un conjunto reducido y representativo del conjunto de prototipos
- **Contrapartida**:
  - Genera ligera pérdida de bondad.

Métodos


Supervisados
Metodos sobre Espacios metricos
Basados en consistencia
C. de Hart
C. Reducido
C. Ordenado
C. MNV Ordenado
C. Selectivo
Metodos sobre Espacios Vectoriales
Metodos Adaptativos
LVQ
DVQ
DSM
Basados en diagramas de Voronoi
C. Optimo
Patching
Otros metodos
C. de Chang

Condensado de Hart

- **Idea:**
- Un conjunto $\boldsymbol{S}_C$ se dice ***consistente*** respecto a otro conjunto $\boldsymbol{S}$, donde $\boldsymbol{S}_C \subset \boldsymbol{S}$, si $\boldsymbol{S}$ puede clasificarse correctamente usando los elementos de $\boldsymbol{S}_C$ como referencia.
- Selección de prototipos que determinen las fronteras de decisión
- Incremental sin vuelta atrás

Condensado de Hart: HART (S)

Entradas

$S \in A$ Cto. de prototipos a condensar (editado).

Salida

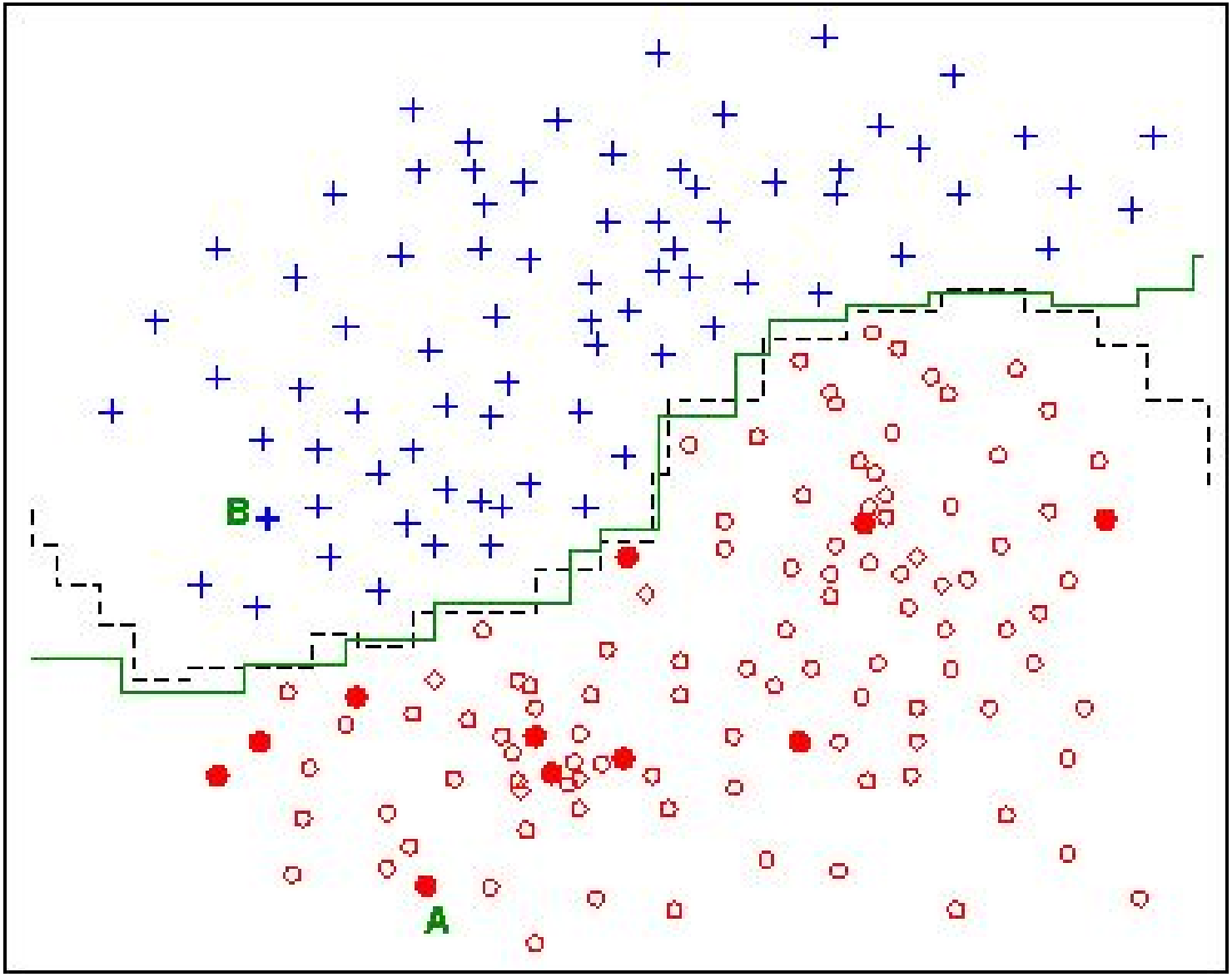
$S_C \subseteq S$ Conjunto de prototipos condensado a partir de S y consistente con S.

Auxiliares

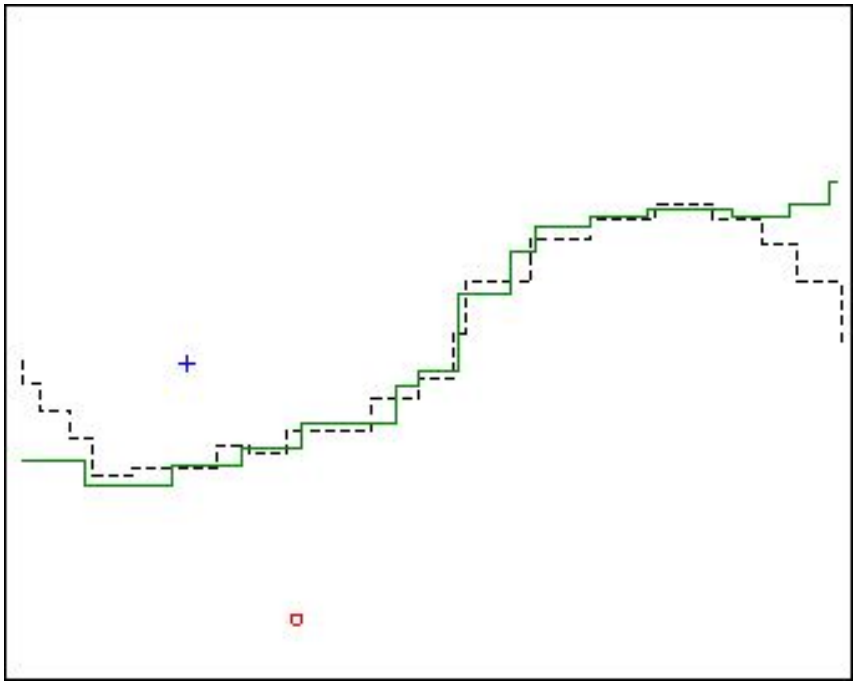
$S_{aux} \in A$ Cto. auxiliar de prototipos. Inicialmente, $S_{aux} = S$

Algoritmo

$S_C \leftarrow \emptyset$
$S_{aux} \leftarrow S$
Repetir
  salir $\leftarrow$ VERDAD
  Para todo $Z \in S_{aux}$ hacer
    Si (Dif (Z, S_C, 1)) entonces
      $S_{aux} \leftarrow S_{aux} - \{Z\}$
      $S_C \leftarrow S_C \cup \{Z\}$
      salir $\leftarrow$ FALSO
    Fin-si
  Fin-para
Hasta ((*salir*) O ($S_{aux} = \emptyset$))

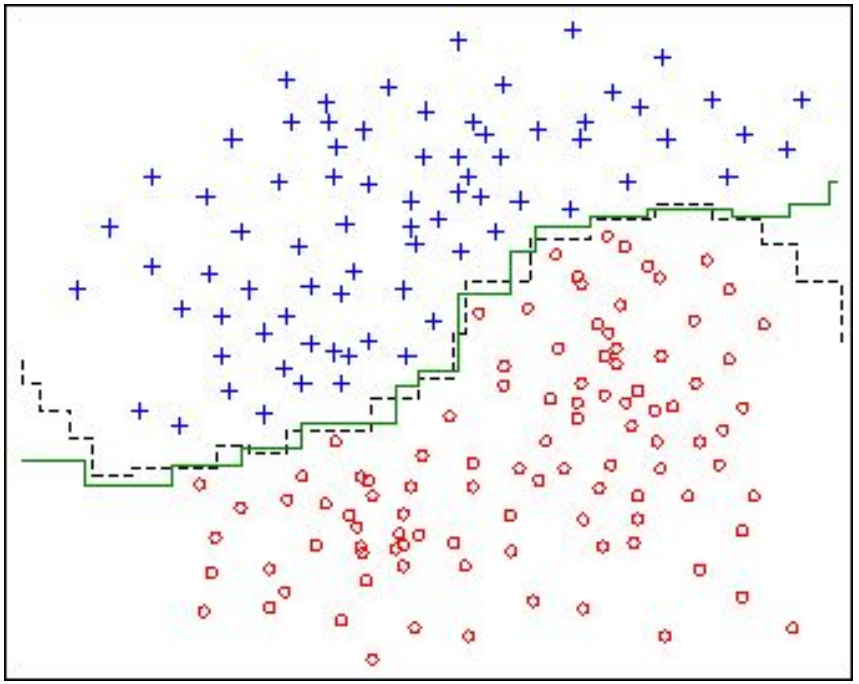

B
A

S_c

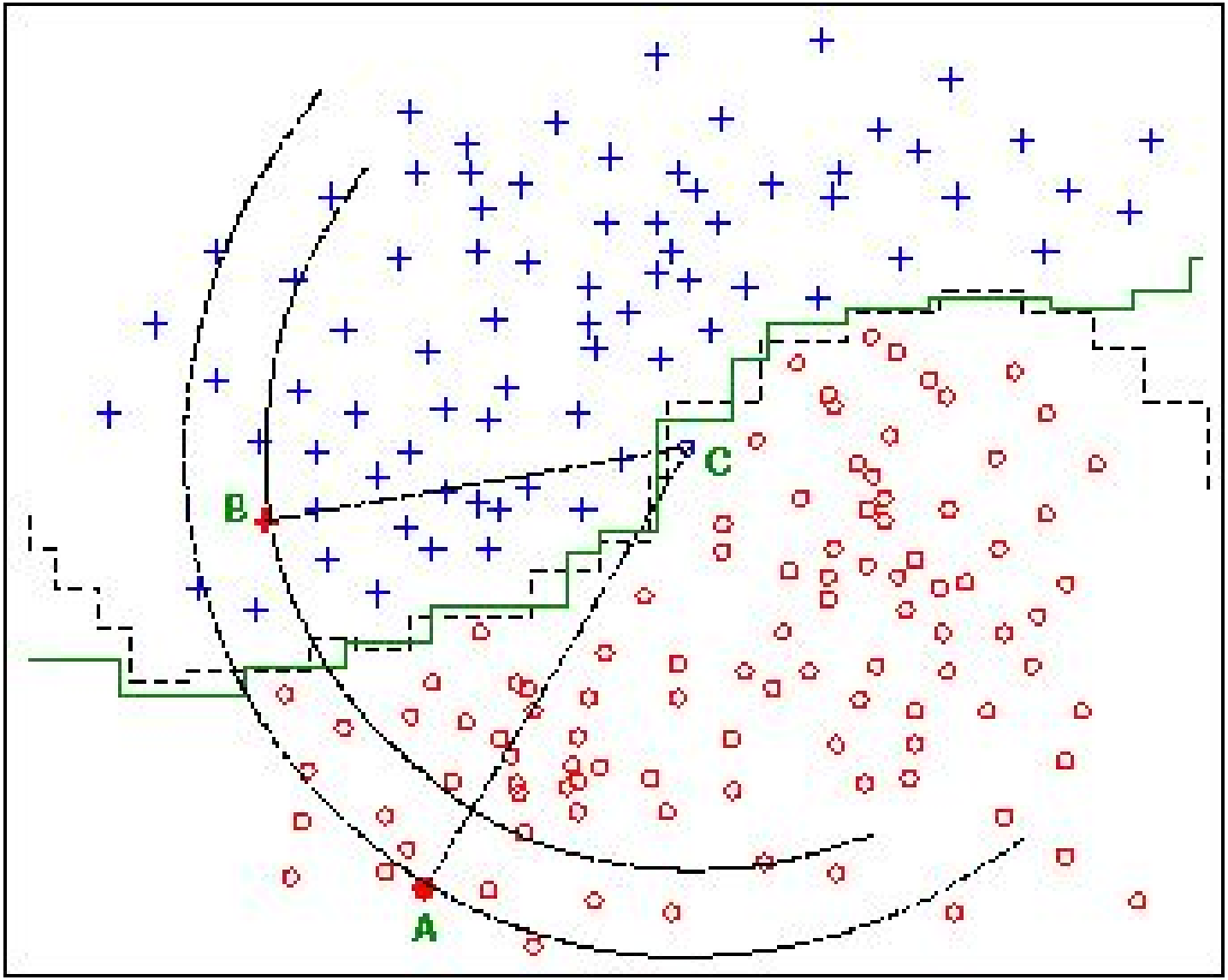


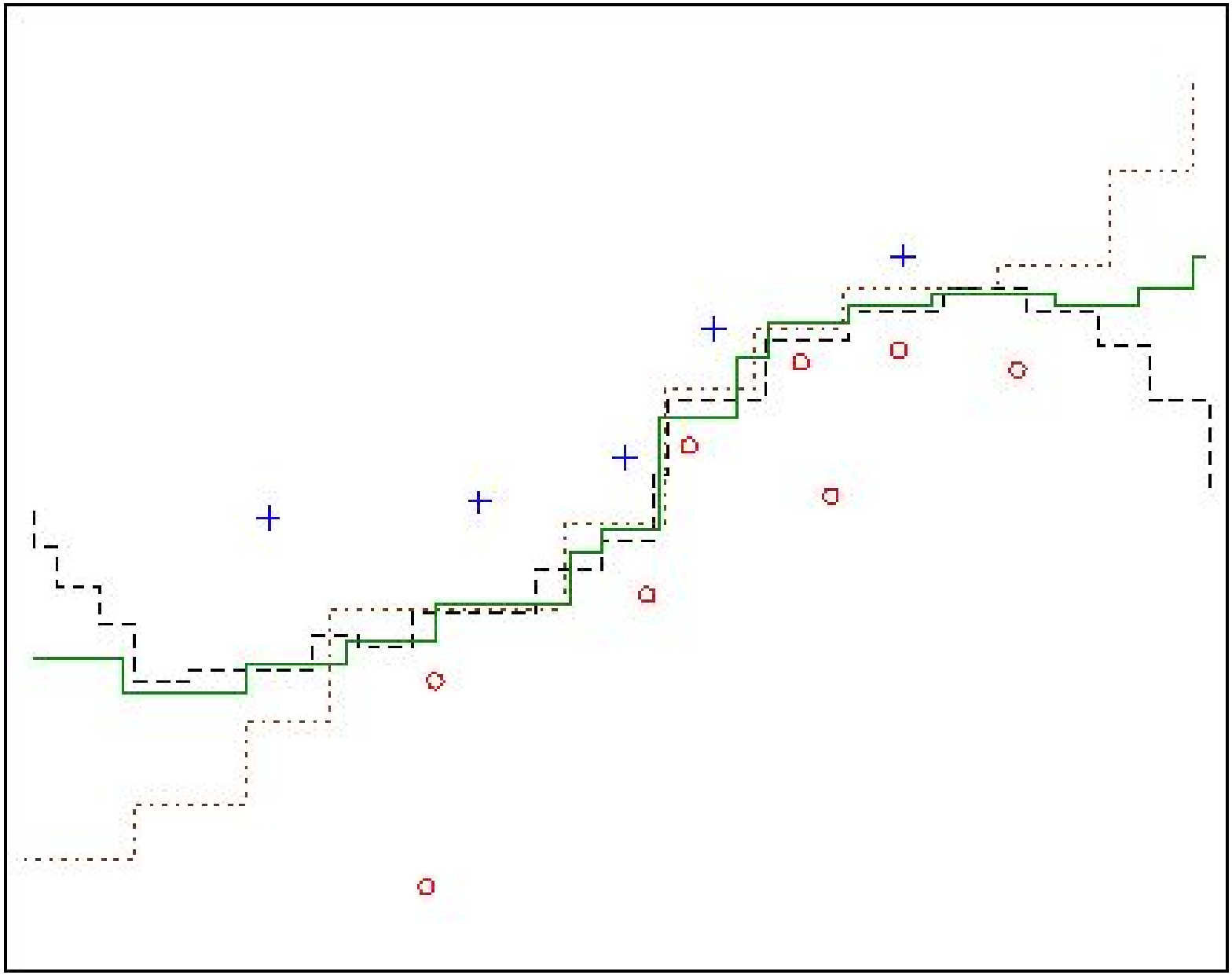
Representación del conjunto condensado S_C tal como ha quedado tras añadir los dos primeros prototipos

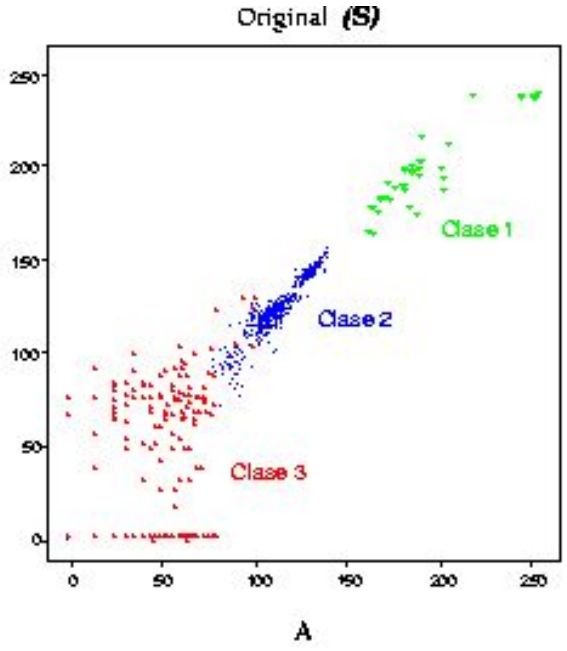
S_{aux}

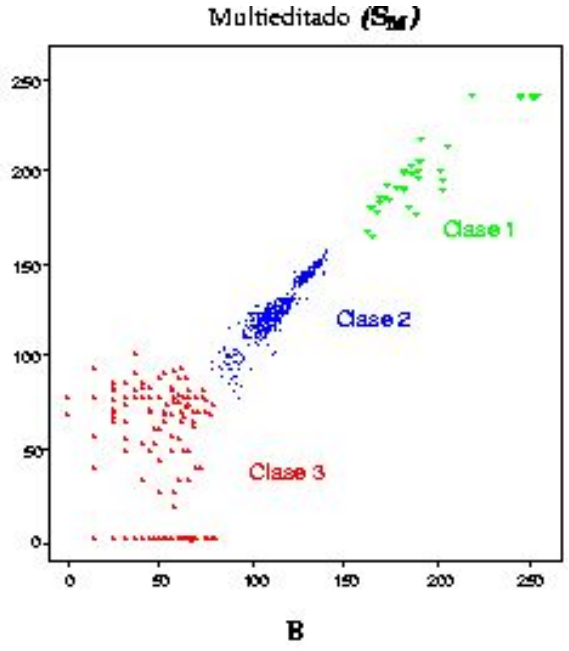


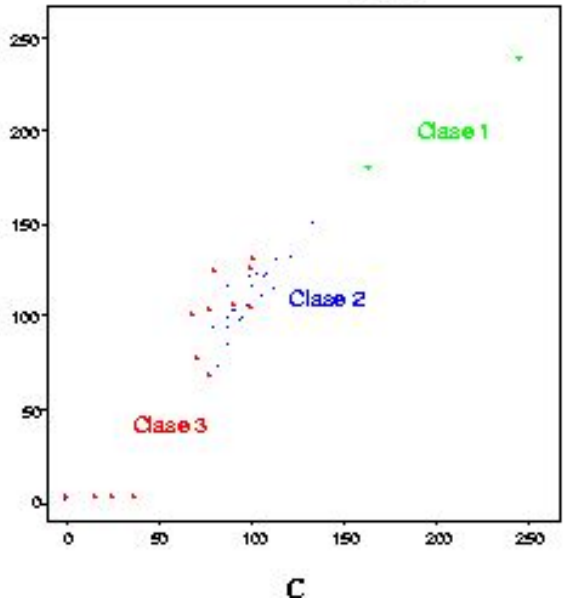
Conjunto de prototipos que falta por procesar. Además de A y B se han eliminado de S_{aux} los prototipos indicados con círculos rellenos

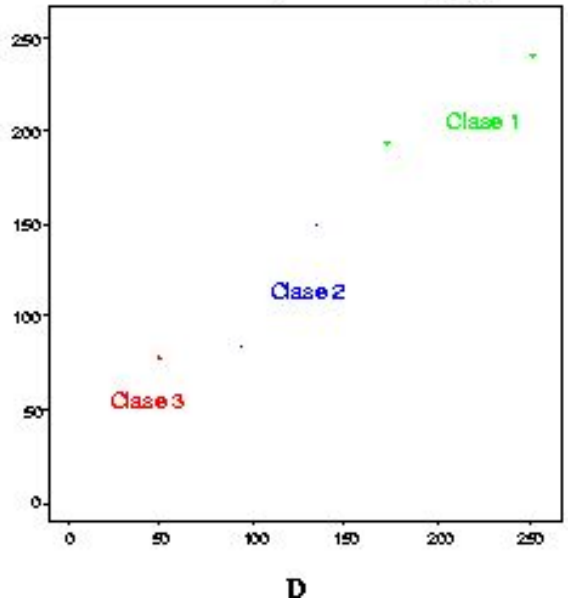

B
C
A




Original (S)
Clase 1
Clase 2
Clase 3
A


Multieditado (S_M)
Clase 1
Clase 2
Clase 3
B


Condensado (S_C)
Clase 1
Clase 2
Clase 3
C


Multieditado y condensado (S_{MC})
Clase 1
Clase 2
Clase 3
D

Condensado de Hart

- Requiere un conjunto previamente editado (para asegurar consistencia).
- No produce un conjunto minimal, sólo un **conjunto *reducido***. En general reducción *severa* → drástica disminución de costo computacional de 1-NN
- Fronteras de decisión no son tan cercanas a la frontera óptima de Bayes como las del clasificador 1-NN sobre conjunto editado → ligera **pérdida de precisión**
- Procedimiento de selección de prototipos de S_{aux} es aleatorio → se obtienen diferentes conjuntos condensados a partir del mismo conjunto de prototipos (**conjunto condensado no es único**)

Métodos de aprendizaje adaptativo

- **LVQ** (***L****earning* ***V****ector* ***Q****uantization*) o aprendizaje por cuantificación vectorial, propuestos por Kohonen

- **DSM** (***D****ecision* ***S****urface* ***M****apping*) o construcción de superficies de decisión, propuesto por Geva y Sitte

Métodos de aprendizaje adaptativo

- Fija a priori la cantidad de prototipos del conjunto de aprendizaje resultante n_p
- El conjunto resultante no tiene porque estar incluido en el conjunto Inicial.
- Heurística sencilla, rapidez de cálculo
- Dificultad para establecer valores adecuados de los parámetros.

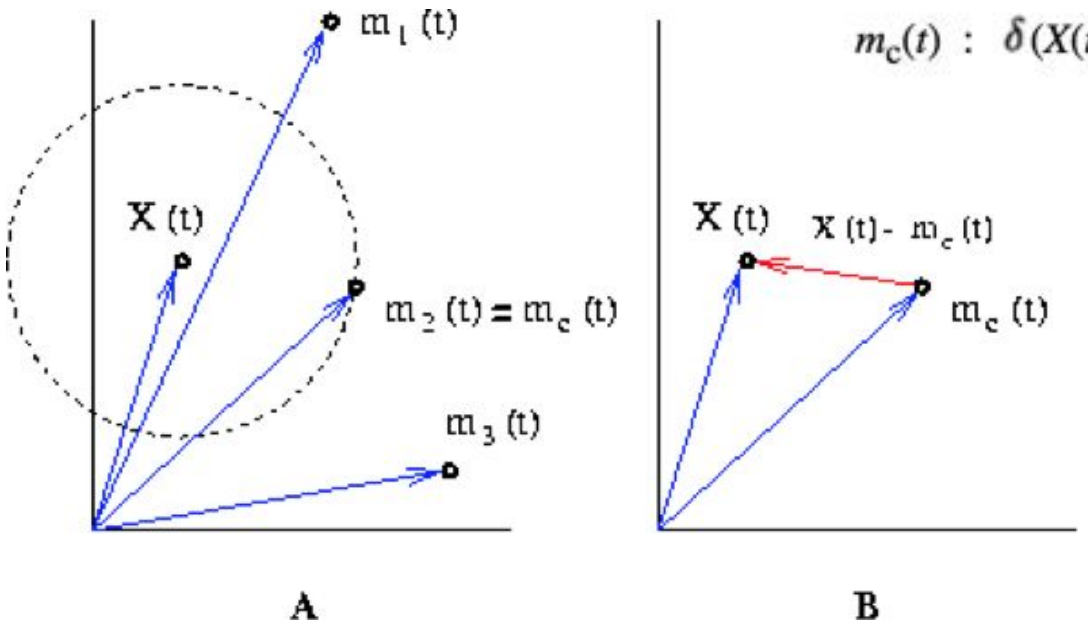
Aprendizaje competitivo y cuantificación vectorial

- Supongamos que se dispone de:
  - $X = X(t),\ t = 1, 2,...$ Una secuencia de patrones (uno por paso de tiempo)
  - $\{m_i(t) : m_i(t),\ i = 1, 2,..., n_p\}$, Un conjunto fijo de **vectores de referencia** o **prototipos** que se modifican durante el aprendizaje.
  - $\{m_i(0), i = 1, 2,..., n_p\}$ ha sido inicializado de alguna forma.
  - Actualizo $m_c(t)$ el más cercano a $X(t)$

Cuantificación Vectorial

$$m_c(t+1) \longleftarrow m_c(t) + \alpha(t)\,[X(t) - m_c(t)]$$
$$m_i(t+1) \longleftarrow m_i(t) \quad \text{para } i \neq c$$

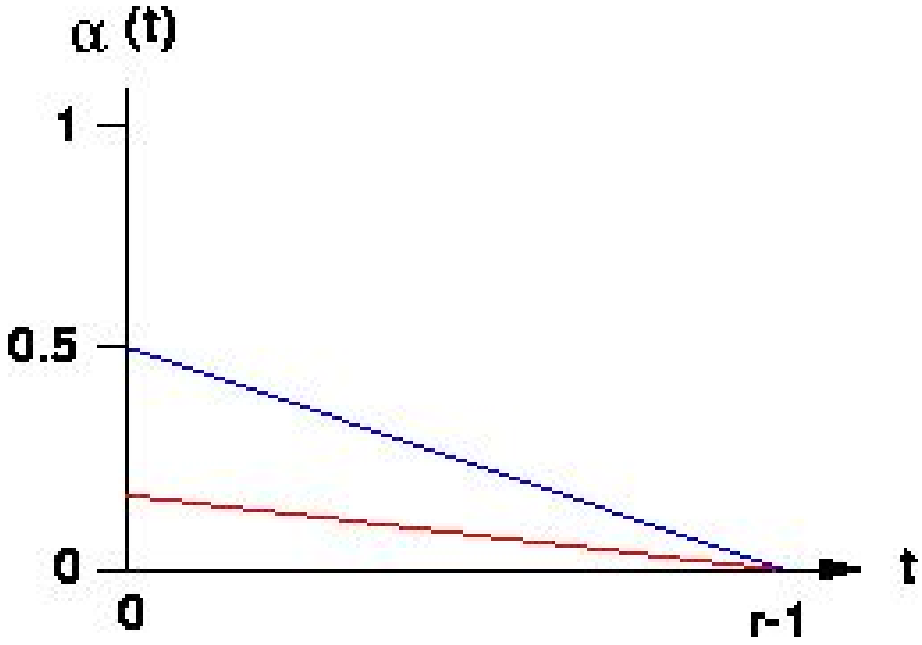
α(t) secuencia monótona *decreciente* de coeficientes escalares : $0 < \alpha(t) < 1$

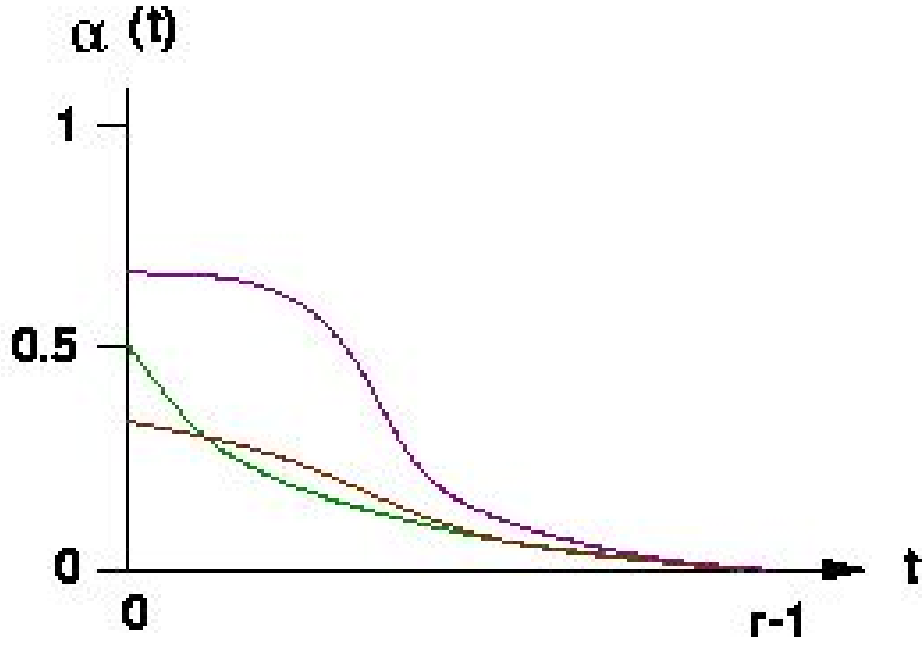


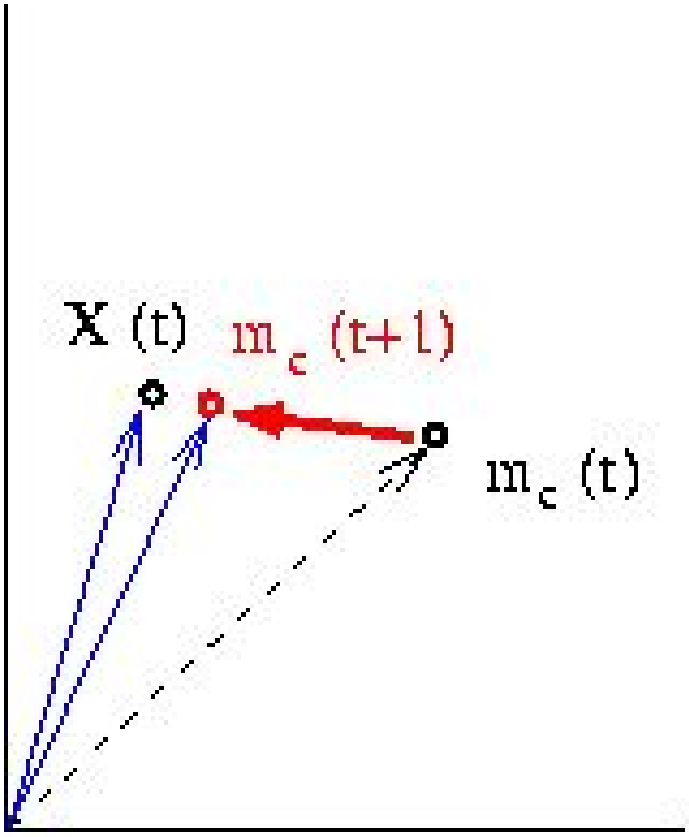
$$m_c(t) : \delta(X(t), m_c(t)) = \min_{i=1\ldots N_p} \{\delta(X(t), m_i(t))\}$$

Con esta estrategia se modifica un único prototipo (m_c) en cada paso t de aprendizaje

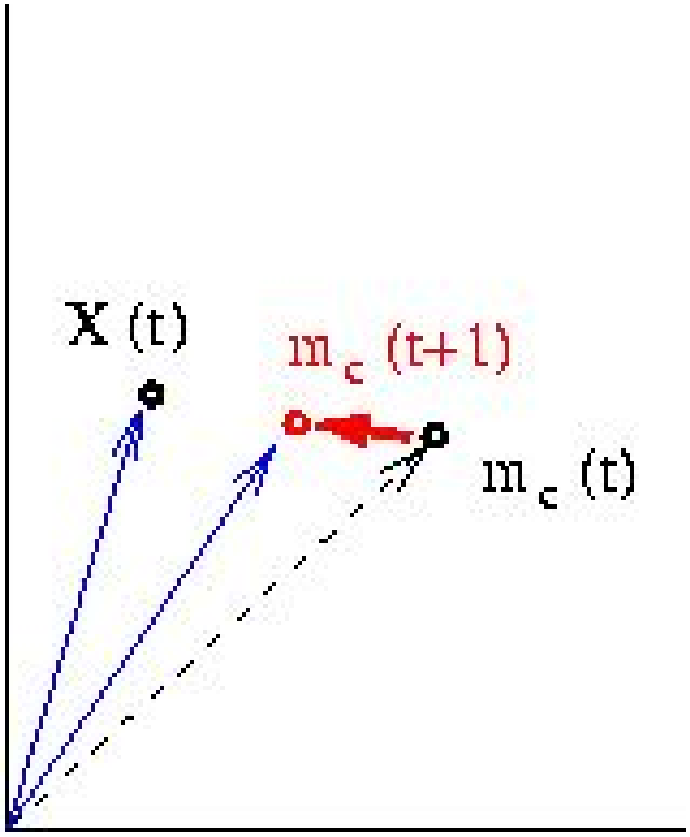
Función de Ganancia o Razón de Aprendizaje







A ($\alpha(t) = 0.75$)



B ($\alpha(t) = 0.4$)

Aprendizaje por cuantificación vectorial

Learning vector quantization (LVQ)

Extensión de Aprendizaje competitivo donde los prototipos están **etiquetados**

Inicialización

- Determinación de n_{pi} (número de referencias por clase)
  - proporcional a n_i (diferentes prob. a priori)
  - n_{pi} sea el mismo para todas las clases

- Seleccionan los prototipos de $S_{LVQ}(0)$:
  - Para cada clase, se procesan *secuencialmente* sus prototipos.
  - Se añaden a $S_{LVQ}(0)$ si la clasificación k-NN es correcta.

Aprendizaje por cuantificación vectorial

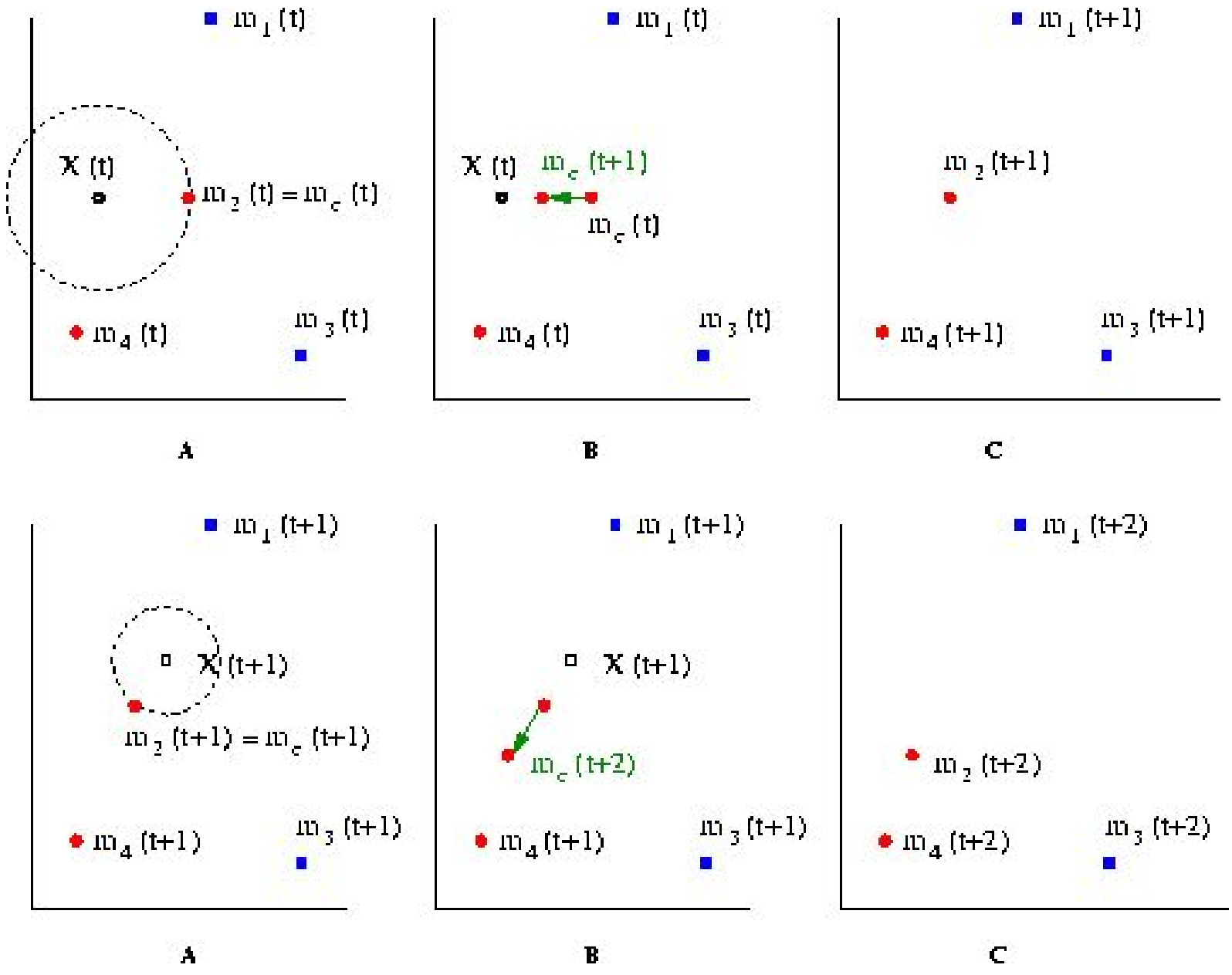
Learning vector quantization (LVQ)

$$S_{\mathrm{LVQ}}(0), S_{\mathrm{LVQ}}(1), \ldots, S_{\mathrm{LVQ}}(r-1) = S_{\mathrm{LVQ}}$$

$$m_c(t+1) \leftarrow m_c(t) + \alpha(t)[X(t) - m_c(t)] \quad \{Premio\}$$

$$m_c(t+1) \leftarrow m_c(t) - \alpha(t)[X(t) - m_c(t)] \quad \{Castigo\}$$

Premio: Si la clase de $m_c(t)$, coincide con la $X(t)$,
Castigo: En otro caso, $m_c(t)$ se aleja de $X(t)$.


$m_1(t)$
$X(t)$
$m_2(t) = m_c(t)$
$m_4(t)$
$m_3(t)$
A
$m_1(t)$
$X(t)$
$m_c(t+1)$
$m_c(t)$
$m_4(t)$
$m_3(t)$
B
$m_1(t+1)$
$m_2(t+1)$
$m_4(t+1)$
$m_3(t+1)$
C
$m_1(t+1)$
$X(t+1)$
$m_2(t+1) = m_c(t+1)$
$m_4(t+1)$
$m_3(t+1)$
A
$m_1(t+1)$
$X(t+1)$
$m_c(t+2)$
$m_4(t+1)$
$m_3(t+1)$
B
$m_1(t+2)$
$m_2(t+2)$
$m_4(t+2)$
$m_3(t+2)$
C

Aprendizaje adaptativo (S, N_p, r)

Entradas

$S \in A$	Conjunto de entrenamiento original.
$N_p \in \mathbb{N}$	Numero de prot. del cto. de referencia.
$r \in \mathbb{N}$	Numero de pasos de aprendizaje.

Salidas

$S_{LVQ} \in A$	Conjunto de referencia resultante.

Auxiliares

$Z(t) \in S$	Un prototipo del conjunto S.

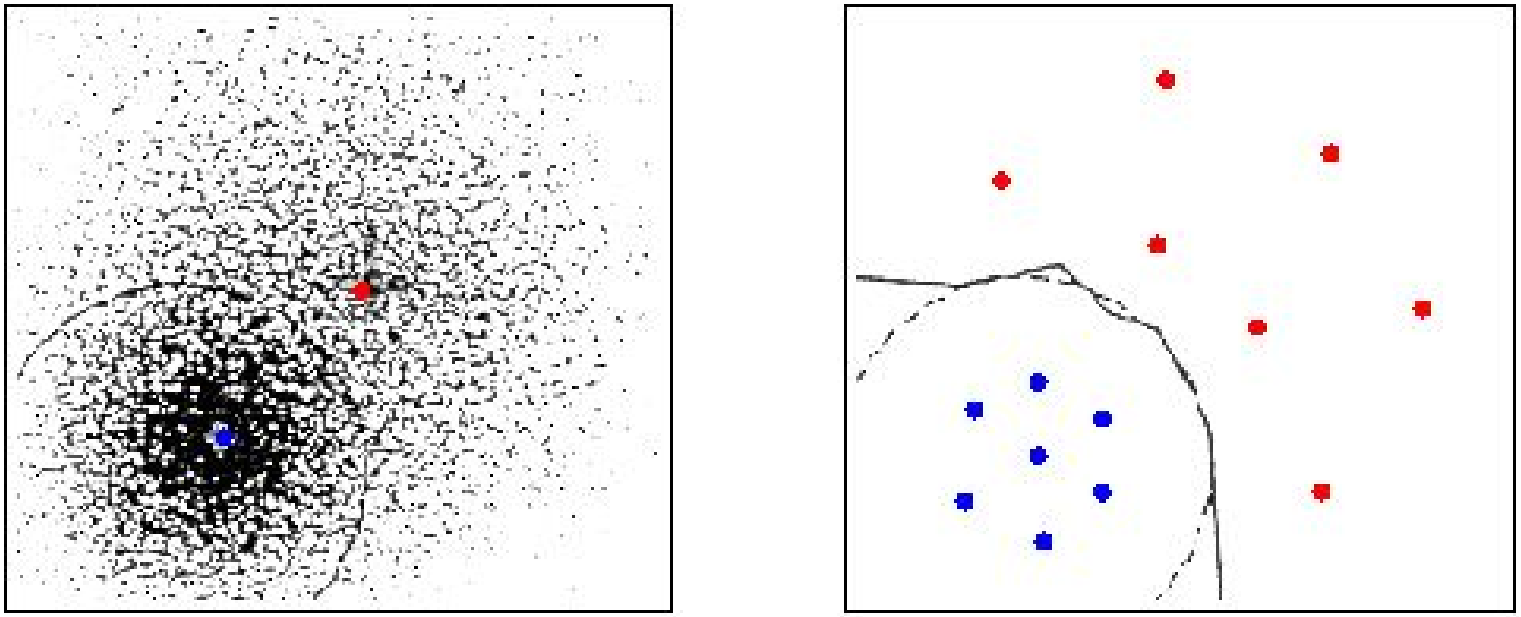
Algoritmo

$S_{LVQ}(0) \leftarrow$ **Inicializar** (S, N_p) $\{Inicializacion\}$
$t \leftarrow 0$
Repetir $\{$ *Aprendizaje* $\}$
  $Z(t) \leftarrow$ **Extraer** (S)
  $S_{LVQ}(t+1) \leftarrow$ **Corregir** $(Z(t), S_{LVQ}(t))$
  $t \leftarrow t+1$
Hasta $((t = r)$ O $(convergencia))$
$S_{LVQ} \leftarrow S_{LVQ}(t-1)$

LQV-1

- Tiende a mover los prototipos hacia prototipos de aprendizaje de su misma clase y a alejarlos de los de otra clase
- Recomendable fijar un valor pequeño para $\alpha(0)$, bastante menor que 0.1 (0.02 ó 0.03).
- Número de pasos de aprendizaje es suficiente con presentar un número de prototipos $50n_p < r < 200n_p$.
- *No es tan importante el valor de* ***r*** *si el conjunto inicial es de buena calidad (previamente editado).*

LQV-1



A B

A) Conjunto original de prototipos, *S*.
B) Conjunto de prototipos $S_{\text{LVQ - 1}}$ construido a partir de los de A) (7 por clase).

Método LVQ-1 Optimizado (OLVQ-1)

$$\alpha_c(t) = \frac{\alpha_c(t-1)}{1 \pm \alpha_c(t-1)}$$

- Cada prototipo del conjunto de referencia tiene su razón de aprendizaje $\alpha_c(t)$
- La función $\alpha(t)$ de LVQ-1 se sustituye por N_p funciones $\alpha_i(t)$

El signo del denominador es:

➔ **Positivo** si Clase($m_c(t)$) = Clase($X(t)$)

$\alpha_c(t)$ *decrece* si X(t) es clasificado correctamente (misma clase).
En prototipos situados *centros* de los agrupamientos *decrece rápidamente*.

➔ **Negativo** si Clase($m_c(t)$) ≠ Clase ($X(t)$).

$\alpha_c(t)$ *crece* si X(t) es clasificado incorrectamente (distinta clase).
En prototipos cercanos a las *fronteras* $\alpha_c(t)$ crece rápidamente (lo que significa que se alejan muy pronto hacia el centro del agrupamiento).

➔ $\alpha_{cmáx} = 0.3, \quad 30n_p < r < 50n_p$

➔ Se desestabiliza para valores altos de r

LVQ-2.1, LVQ-3

- Otras versiones propuestas del aprendizaje LVQ son los llamados métodos LVQ-2.1 y LVQ-3.
- La principal diferencia respecto a LVQ-1 es que modifican más de un prototipo. LVQ-2.1 modifica dos prototipos (el más cercano de la misma clase y el más cercano de distinta clase)
- Sin embargo, requieren más parámetros que especificar.
- El desempeño es similar, idea: usar métodos con menos parámetros

Aprendizaje de Superficies de Decisión

(Decision Surface Mapping - DSM)

Adaptive Nearest Neighbor Pattern Classification

Shlomo Geva and Joaquin Sitte

Abstract—**We describe a variant of nearest neighbor pattern classification (NN) [1] and supervised learning by learning vector quantization (LVQ) [2], [3]. The decision surface mapping method, which we call DSM, is a fast supervised learning algorithm, and is a member of the LVQ family of algorithms. A relatively small number of prototypes are selected from a training set of correctly classified samples. The training set is then used to adapt these prototypes to map the decision surface separating the classes. This algorithm is compared with NN pattern classification, learning vector quantization (LVQ1) [2], and a two-layer perceptron trained by error backpropagation [4]. When the class boundaries are sharply defined (i.e., no classification error in the training set) the DSM algorithm outperforms these methods with respect to error rates, learning rates, and the number of prototypes required to describe class boundaries.**

I. Introduction

The nearest neighbor (NN) method assigns an unclassified sample vector to the class of the nearest of a set of correctly classified prototypes, or codebook vectors. Cover and Hart have shown that in a large sample, the error of this rule is bounded above by twice the Bayes probability of error [1].

Learning vector quantization (LVQ1, LVQ2, LVQ2.1, and LVQ3), described by Kohonen [2], [3], is a nearest neighbor classification method in which a fixed number of prototype vectors are progressively modified to cover the input space. The LVQ family of algorithms is concerned with optimal placement of these prototypes, so as to reflect the probability distribution of the training samples. The adaptive decision surface mapping (DSM) algorithm is a variation of the LVQ method, but we have dropped the requirement that the prototypes reflect the probability distribution of the classes. Instead, the algorithm adapts the prototype vectors to

rectly classified, that is, the training sample is of the same class as the nearest prototype, no modifications are applied. When misclassification occurs, modifications take place to apply both punishment and reward.

The punishment step takes the nearest neighbor prototype, which, in this case, is of the wrong class, and moves it away from the training sample, along the line connecting the two vectors

$$\vec{m}_w(t+1) = \vec{m}_w(t) - \alpha(t)\left[\vec{x}(t) - \vec{m}_w(t)\right]. \qquad (1)$$

The reward step searches for the nearest correct prototype and moves it towards the training sample, along the line connecting the two vectors

$$\vec{m}_c(t+1) = \vec{m}_c(t) + \alpha(t)\left[\vec{x}(t) - \vec{m}_c(t)\right]. \qquad (2)$$

The term α is a scalar gain factor, monotonically decreasing with time. For the cases discussed below, we have found that very good results are obtained when α starts from a value of 0.3 or less, and linearly decreases to 0, at a rate consistent with the desired training limit (number of presentations). The algorithm is not very sensitive to initial values of α, but if alpha starts too small, training takes longer.

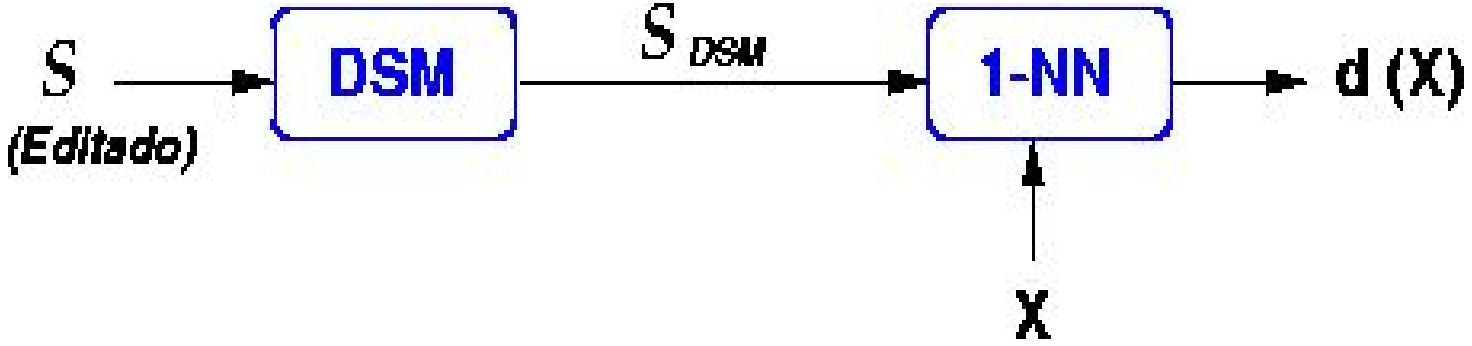
In the earlier stages of the training process α is relatively large; therefore the process is allowed to rapidly modify prototypes to remove large classification errors caused by the initial conditions. In later stages, as α decreases, a more refined adaptation takes place to correct smaller classification errors or to arrive at a compromise configuration where errors are minimized.

The algorithm modifies prototypes only on misclassification, and since errors are more likely to occur with samples near class boundaries, it rearranges prototypes, in pairs, on each side of a class boundary, to correct or at least reduce the magnitude of these errors.

It is possible that a configuration eliminating all classification errors on the training set could be arrived at before α reaches a value of 0. In that instance training is complete.

DSM is different from all the variants of LVQ. In LVQ1 modifications are applied at each presentation, either to punish an incorrect classification or to reward a correct one. LVQ2 modifies

Aprendizaje de Superficies de Decisión



- Sólo se corrige si hay error de clasificación.
- Se **castiga** al prototipo más cercano (el inductor del error).
- Se **premia** al prototipo más cercano de la misma clase que $Z(t)$.

$$m_c(t+1) \leftarrow m_c(t) + \alpha(t)[X(t) - m_c(t)] \qquad \{Premio\}$$

$$m_w(t+1) \leftarrow m_w(t) - \alpha(t)[X(t) - m_w(t)] \qquad \{Castigo\}$$

Corrección DSM $(Z(t),\ S_{DSM}(t))$

Entradas

$Z(t) \in S$	Un prototipo de aprendizaje de S.
$S_{DSM}(t) \in A$	Cto. de prototipos en el paso t.

Salidas

$S_{DSM}(t+1) \in A$	Cto. de prototipos corregido para $t+1$.

Auxiliares

$m_w(t), m_c(t) \in S_{DSM}(t)$	Dos prototipos de $S_{DSM}(t)$
$X(t) \in P$	Patron asociado a $Z(t)$.

Algoritmo

$X(t) \leftarrow$ **Patron** $(Z(t))$
$m_w(t) \leftarrow$ **1-NN** $(X(t),\ S_{DSM}(t))$ { *Prototipo mas cercano* }

Si (**Clase** $(Z(t)) \neq$ **Clase** $(m_w(t))$ entonces

$m_c(t) \leftarrow$ **1-NNClase** $(Z(t),\ S_{DSM}(t))$
{ *Prototipo mas cercano de igual clase que* $Z(t)$ }
$m_w(t+1) \leftarrow m_w(t) - \alpha(t)[X(t) - m_w(t)]$ { *castigo* }
$m_c(t+1) \leftarrow m_c(t) + \alpha(t)[X(t) - m_c(t)]$ { *premio* }

Fin-si
Devolver $(S_{DSM}(t+1))$

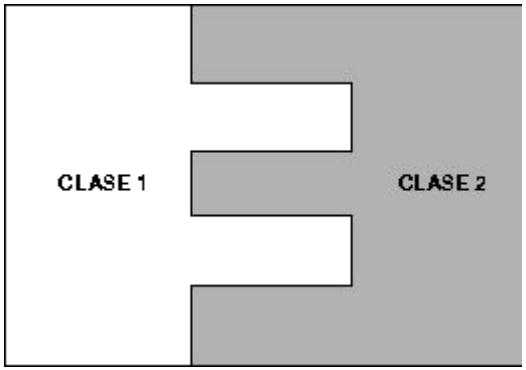
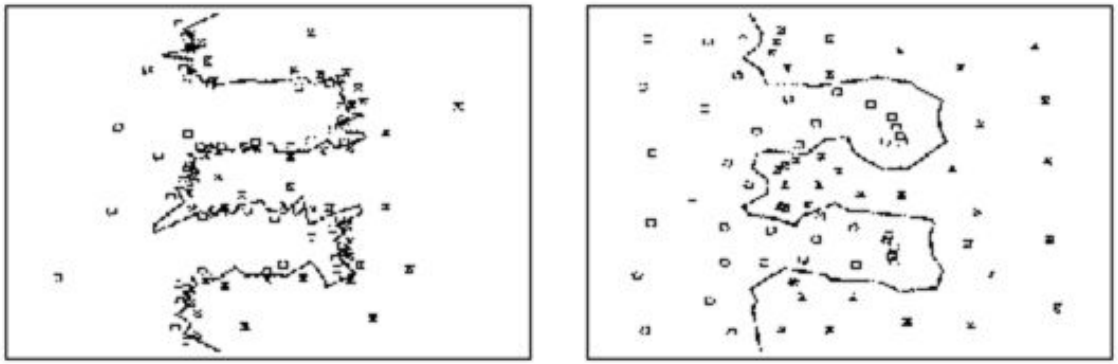
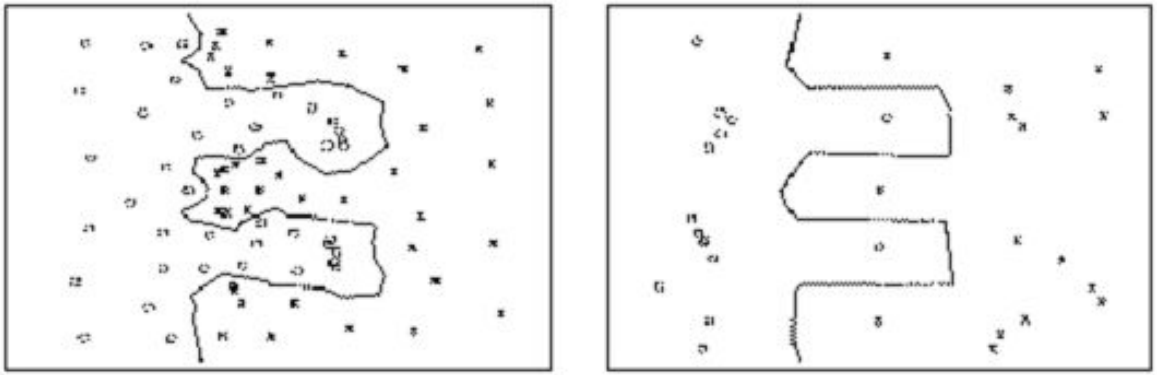


Figura 59: a) Condensado de Hart. b) LVQ-1



A B

Figura 67: A) LVQ-1. B) DSM



A B

Tabla 11: Error de clasificación 1-NN para diferentes valores de N_p

N_p	DSM	LVQ-1
6	7.14	19.00
8	3.82	19.55
9	1.86	14.64
10	0.43	12.34
20	0.43	4.44
24	0.41	3.06
50	0.49	2.51
250	0.79	1.84