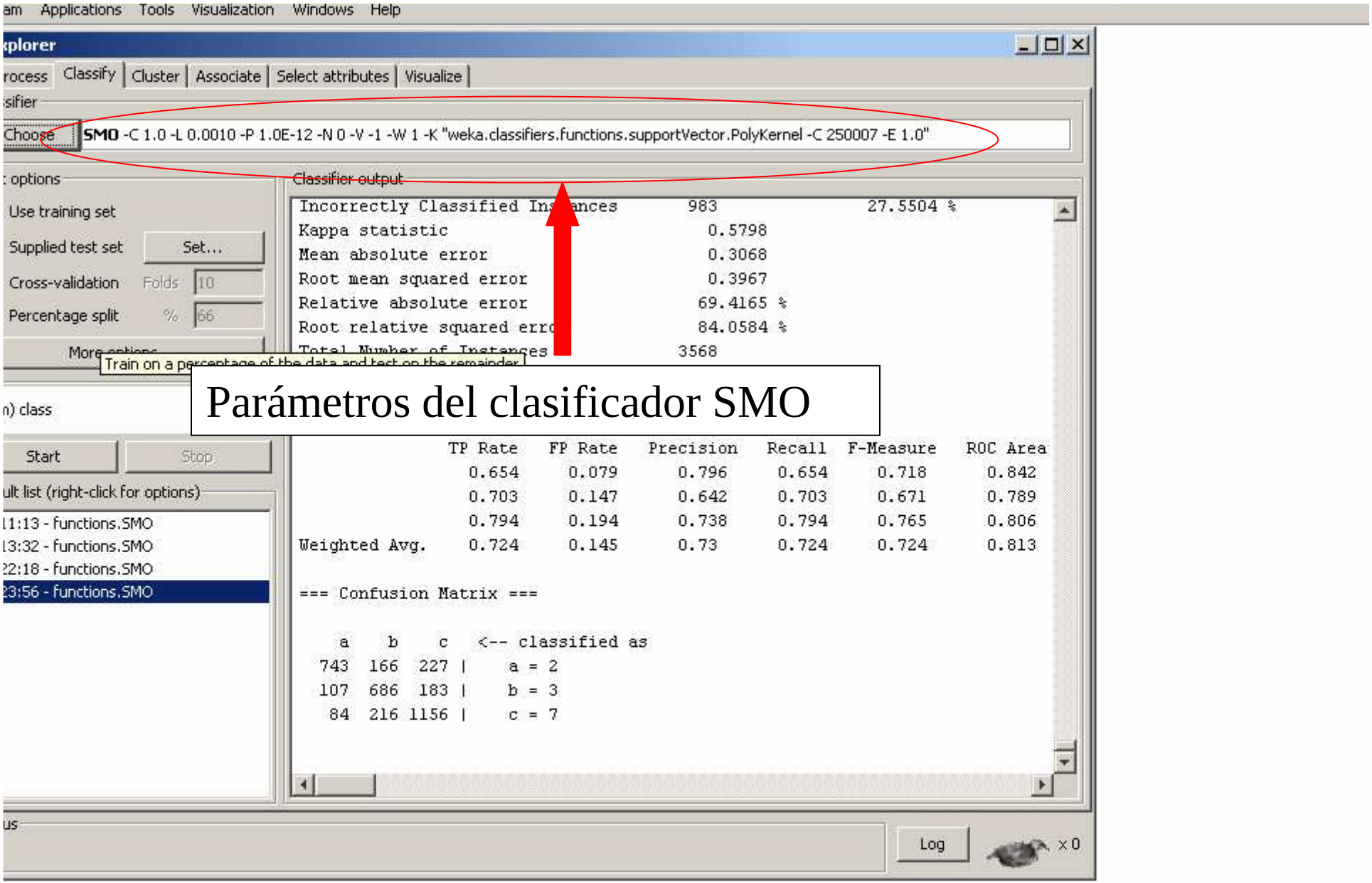
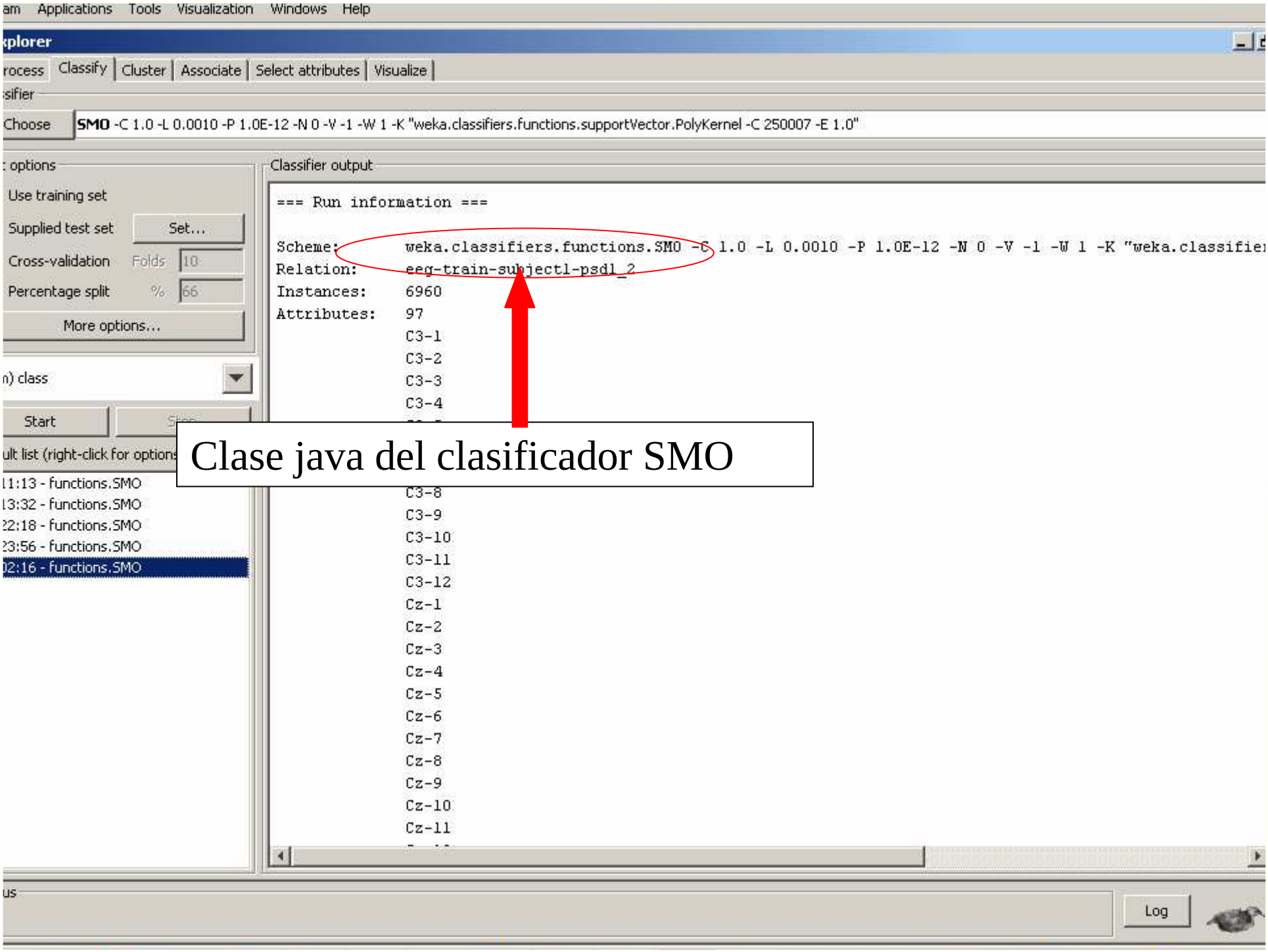
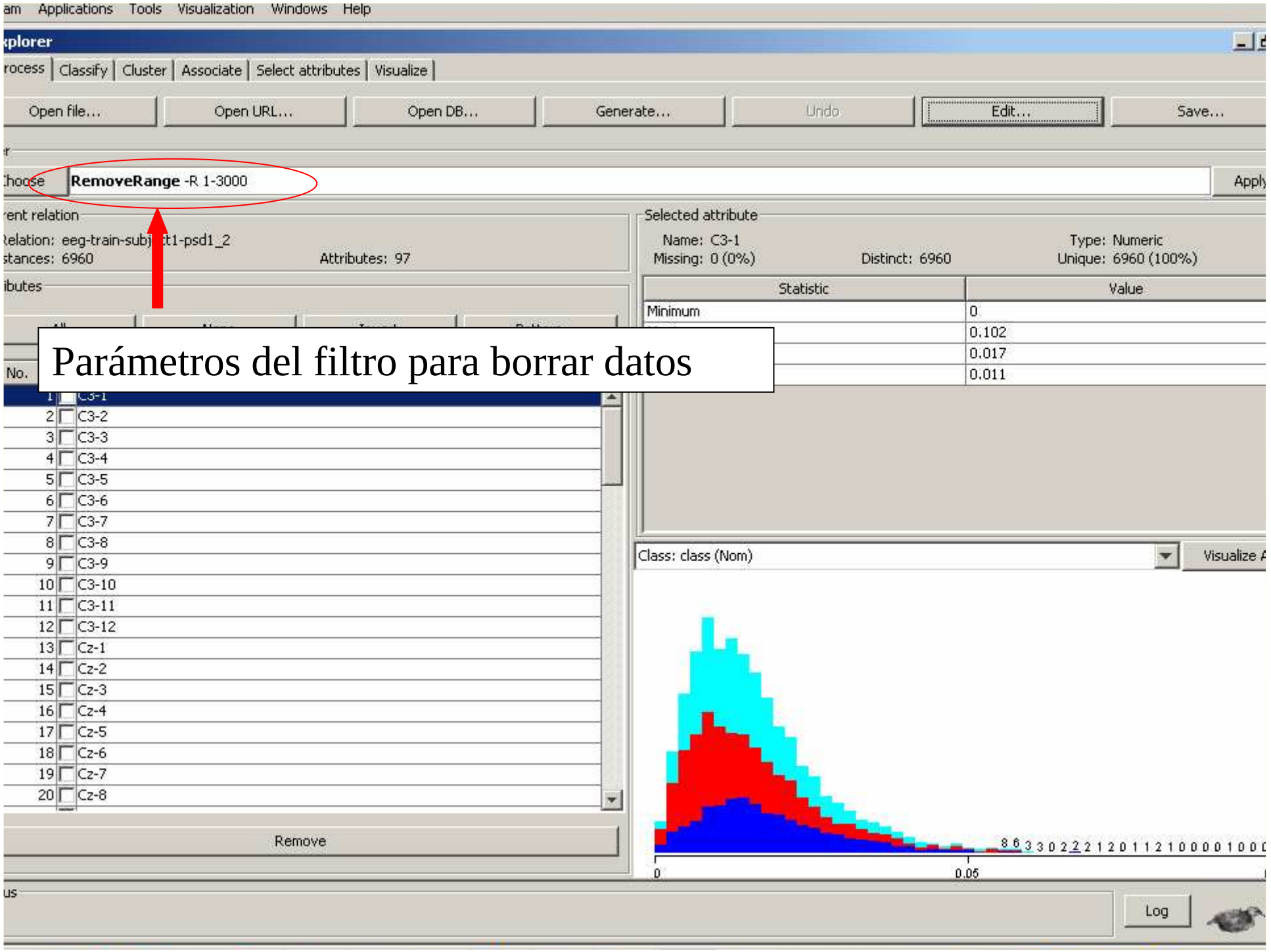
Uso de Weka desde un script

Script para hacer una curva de aprendizaje

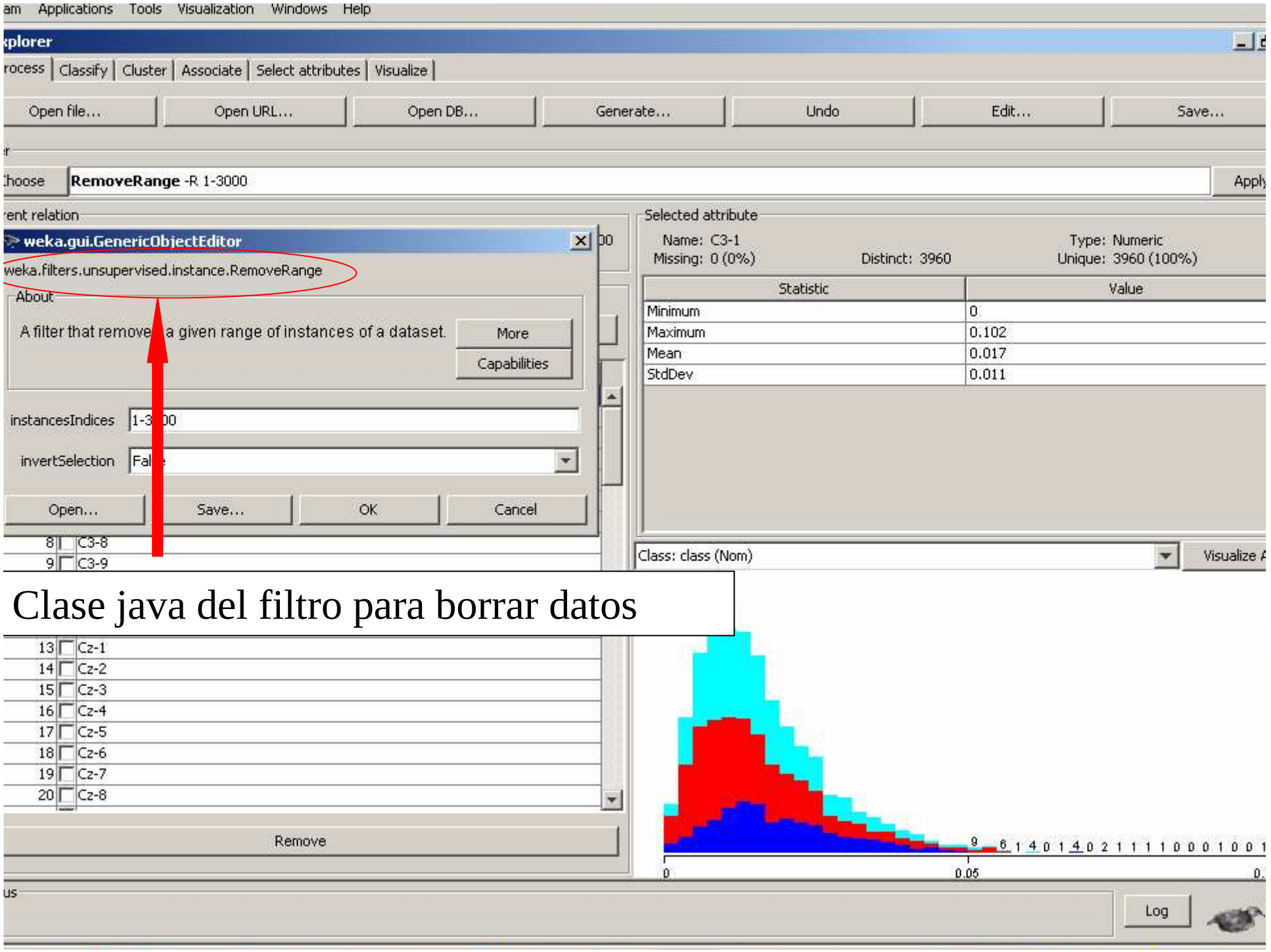
- Cómo usar Weka desde la línea de comandos para, por ejemplo, hacer una curva de aprendizaje
- Probar con:
  - 20% de los datos, validación cruzada de 10
  - 40%
  - 60%
  - 80%
  - 100%
- Útil, porque para hacer pruebas, normalmente no es necesario usar todos los datos
- Usaremos un script que:
  - Utiliza un filtro que borra cierta cantidad de datos y
  - Le pasa un clasificador SMO (SVM)


Applications Tools Visualization Windows Help
Classify
Cluster
Associate
Select attributes
Visualize
Choose
SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
Classifier output
Incorrectly Classified Instances 983 27.5504 %
Kappa statistic 0.5798
Mean absolute error 0.3068
Root mean squared error 0.3967
Relative absolute error 69.4165 %
Root relative squared error 84.0584 %
Total Number of Instances 3568
Parámetros del clasificador SMO
TP Rate FP Rate Precision Recall F-Measure ROC Area
0.654 0.079 0.796 0.654 0.718 0.842
0.703 0.147 0.642 0.703 0.671 0.789
0.794 0.194 0.738 0.794 0.765 0.806
Weighted Avg. 0.724 0.145 0.73 0.724 0.724 0.813
=== Confusion Matrix ===
a b c <-- classified as
743 166 227 | a = 2
107 686 183 | b = 3
84 216 1156 | c = 7
Start
Stop
Log
x 0


Applications Tools Visualization Windows Help
Classify Cluster Associate Select attributes Visualize
Choose
SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
Start
Classifier output
=== Run information ===
Scheme: weka.classifiers.functions.SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifier
Relation: eeg-train-subject1-psd1_2
Instances: 6960
Attributes: 97
C3-1
C3-2
C3-3
C3-4
C3-8
C3-9
C3-10
C3-11
C3-12
Cz-1
Cz-2
Cz-3
Cz-4
Cz-5
Cz-6
Cz-7
Cz-8
Cz-9
Cz-10
Cz-11
Clase java del clasificador SMO
11:13 - functions.SMO
13:32 - functions.SMO
22:18 - functions.SMO
23:56 - functions.SMO
02:16 - functions.SMO
Log



Parámetros del filtro para borrar datos



Clase java del filtro para borrar datos

Opciones de SMO

- Ejemplo, si ejecutamos en la linea de comandos, o ponemos la siguiente linea en un archivo .bat, podremos ver las opciones de SMO:

```
java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"  weka.classifiers.functions.SMO -h
```

Opciones de SMO

- -h: ayuda
- -info: resumen del clasificador
- -t: conjunto de aprendizaje
- -T: conjunto de test
- -x: número de folds para validación cruzada
- -no-cv: no hagas validación cruzada
- -split-percentage: divide en entrenamiento y test
- -preserver-order: preserva el orden de los datos en split
- -C: Complejidad
- -L: Tolerancia
- -P: Epsilon
- Etc …

Opciones de RemoveInstances

- -i: fichero de entrada
- -o: fichero de salida
- -R: rango de instancias a borrar
- -V: si en lugar de borrar, se quiere seleccionar

Curva de aprendizaje para BCI, sujeto 1

- Vamos a hacerlo con los 3960 datos de las sesiones 1 y 2:
  - 20%: 792 datos
  - 40%: 1584 datos
  - 60%: 2376 datos
  - 80%: 3168 datos
  - 100%: 3960 datos
- Aplicaremos SMO con validación cruzada de 10, para cada subconjunto de datos
- En este dominio (BCI), tendría más sentido hacerlo con sesiones 1 y 2 para entrenar, y la 3 para hacer el test (en lugar de usar validación cruzada de las sesiones 1+2).

```
java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"
      weka.filters.unsupervised.instance.RemoveRange -R 1-792 -V -i "train_subject1_psd0102.arff" -o
      "modificado.arff"

java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"  weka.classifiers.functions.SMO -C 1.0 -L
      0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
      -t "modificado.arff" -x 10 > resultados20.txt


java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"
      weka.filters.unsupervised.instance.RemoveRange -R 1-1584 -V -i "train_subject1_psd0102.arff" -o
      "modificado.arff"

java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"  weka.classifiers.functions.SMO -C 1.0 -L
      0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
      -t "modificado.arff" -x 10 > resultados40.txt


java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"
      weka.filters.unsupervised.instance.RemoveRange -R 1-2376 -V -i "train_subject1_psd0102.arff" -o
      "modificado.arff"

java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"  weka.classifiers.functions.SMO -C 1.0 -L
      0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
      -t "modificado.arff" -x 10 > resultados60.txt


java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"
      weka.filters.unsupervised.instance.RemoveRange -R 1-3168 -V -i "train_subject1_psd0102.arff" -o
      "modificado.arff"

java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"  weka.classifiers.functions.SMO -C 1.0 -L
      0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
      -t "modificado.arff" -x 10 > resultados80.txt


java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"
      weka.filters.unsupervised.instance.RemoveRange -R 1-3960 -V -i "train_subject1_psd0102.arff" -o
      "modificado.arff"

java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"  weka.classifiers.functions.SMO -C 1.0 -L
      0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
```

Curva de aprendizaje con validación cruzada

- 20%: 98.8636 %
- 40%: 94.6338 %
- 60%: 90.7407 %
- 80%: 86.7109 %
- 100%: 83.0303 %

- Datos no desordenados …
- Ocurre que los datos están parcialmente agrupados por clase

Para desordenar un fichero de datos (randomizar)

```
java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-
   6\weka.jar"
   weka.filters.unsupervised.instance.Randomize -S 42 -
   i "train_subject1_psd0102.arff" -o "randomizado.arff"
```

```
java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"
    weka.filters.unsupervised.instance.Randomize -S 42 -i
    "train_subject1_psd0102.arff" -o "randomizado.arff"


java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"
    weka.filters.unsupervised.instance.RemoveRange -R 1-792 -V -
    i "randomizado.arff" -o "modificado.arff"
java -Xmx1024m -cp "C:\Program Files (x86)\Weka-3-6\weka.jar"
    weka.classifiers.functions.SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N
    0 -V -1 -W 1 -K
    "weka.classifiers.functions.supportVector.PolyKernel -C
    250007 -E 1.0" -t "modificado.arff" -x 10 > resultados20.txt

...
```

Curva de aprendizaje con validación cruzada (y datos previamente desordenados)

- 20%: 75 %
- 40%: 77.7146 %
- 60%: 78.2828 %
- 80%: 78.851 %
- 100%: 78.1566 %

- Con el 40% de los datos parece suficiente
- Nota: con los datos del BCI, esta curva de aprendizaje sería mas conveniente calcular los porcentajes de acierto haciendo test (opción –T) con la sesión 3, en lugar de hacer validación cruzada con 01+02 (opción –x)

Curva de aprendizaje entrenando con 0102 y haciendo el test con sesión 03

- Se usa la opción **-T fichero-test** en vez de **-x 10**

- 20%: 70.796 %
- 40%: 72.0572 %
- 60%: 72.1132 %
- 80%: 72.5897 %
- 100%: 72.5897 %

También se puede hacer desde C, perl, …

```
system("java -cp \"C:\\Archivos de programa\\Weka-3-
   4\\weka.jar\"
   weka.filters.unsupervised.attribute.Remove -R 2-4
-i \"C:\\Archivos de programa\\Weka-3-
   4\\data\\weather.arff\"
-o \"C:\\Archivos de programa\\Weka-3-
   4\\data\\weather-bis.arff\"");
```

(tal vez haya que poner el carácter de escape \)

Más información (Weka y línea de comandos)

- http://weka.sourceforge.net/wekadoc/index.php/en:Primer
- http://maya.cs.depaul.edu/~classes/ect584/WEKA/classify.html

Máquinas de Vectores de Soporte, teoría y práctica

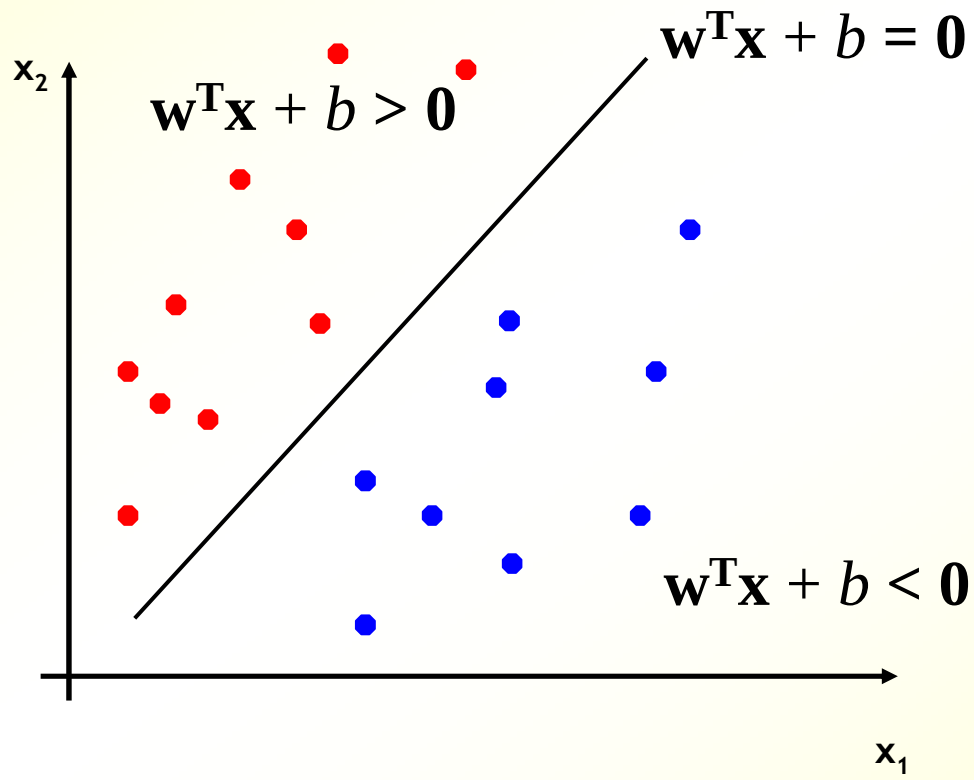
SVMs: Support Vector Machines

Support Vector Machines

- Casos:
  - Modelo lineal:
    - **Los datos son separables linealmente (hard margin)**
    - Los datos no son separables linealmente (soft margin)
  - Modelo no lineal (kernels)

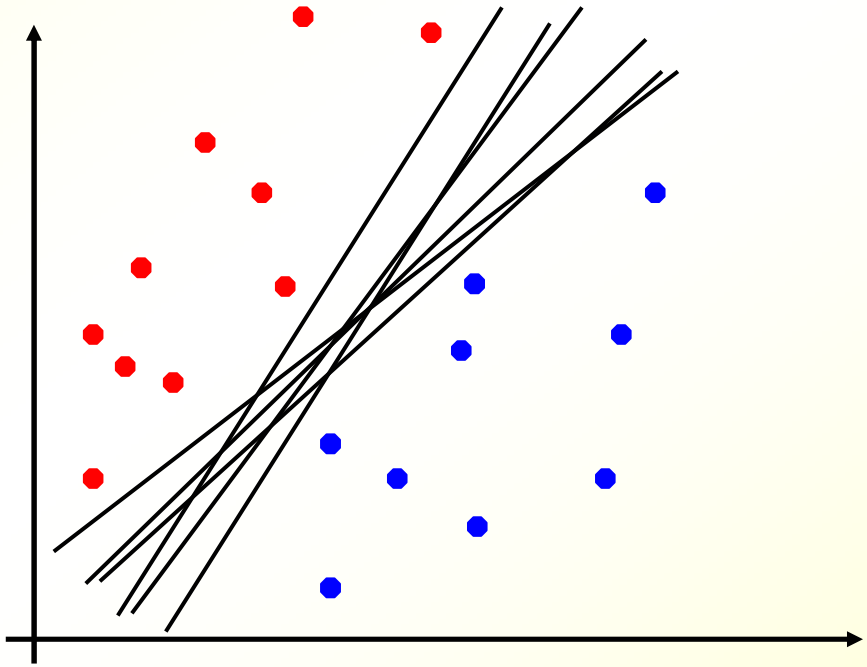
Clasificación lineal biclase

$$w^T x+b = w_1 * x_1 + w_2 * x_2 + b = 0$$

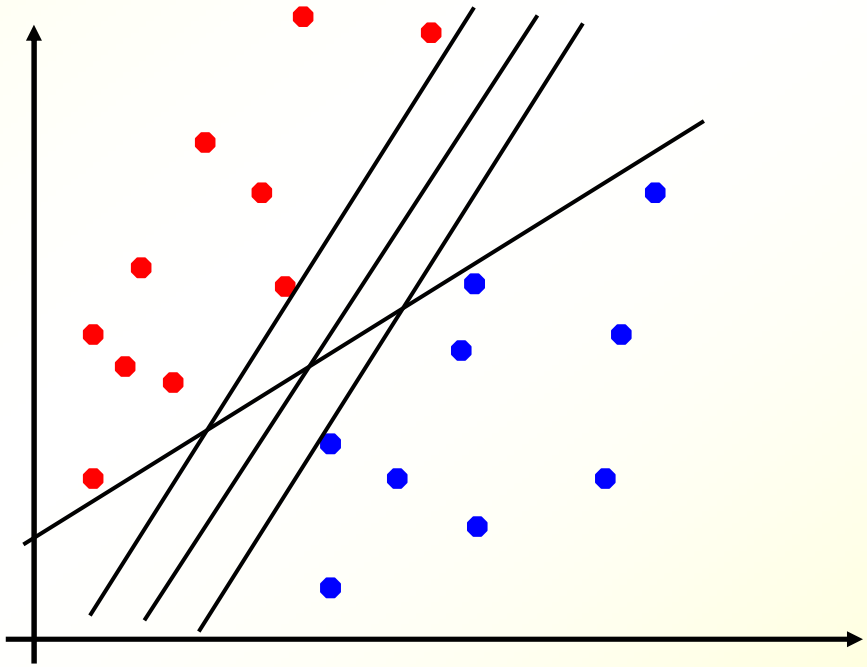


$$f(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$$

¿Qué hiperplano elegir?



¿Qué hiperplano elegir?

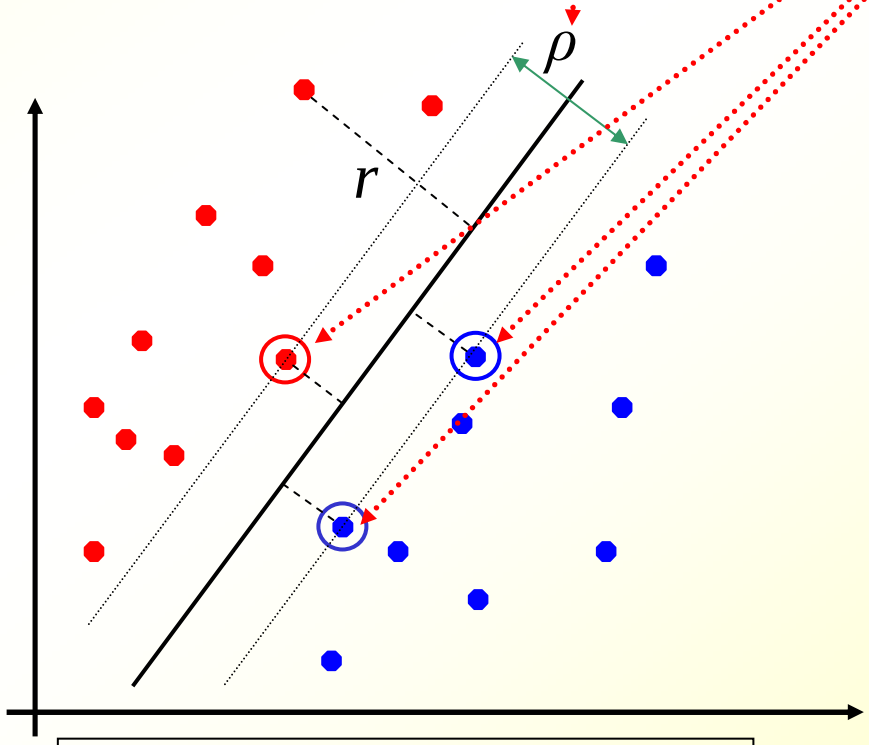


Maximización del margen

El hiperplano que maximiza el margen es también el que generaliza mejor

Margen $\rho = \frac{2}{\|\mathbf{w}\|}$

Vectores de soporte



$\mathbf{w}^T\mathbf{x}+b = w_1*x_1+w_2*x_2+b$

Maximización del margen

- El algoritmo de entrenamiento de las SVMs:
  - Encuentra los vectores de soporte
  - Maximiza el margen $\rho = \frac{2}{\|\mathbf{w}\|}$
  - O lo que es lo mismo, minimiza: $|w|^2 = w^T w$
  - Si los datos no son linealmente separables, el problema no tiene solución (hard margin)

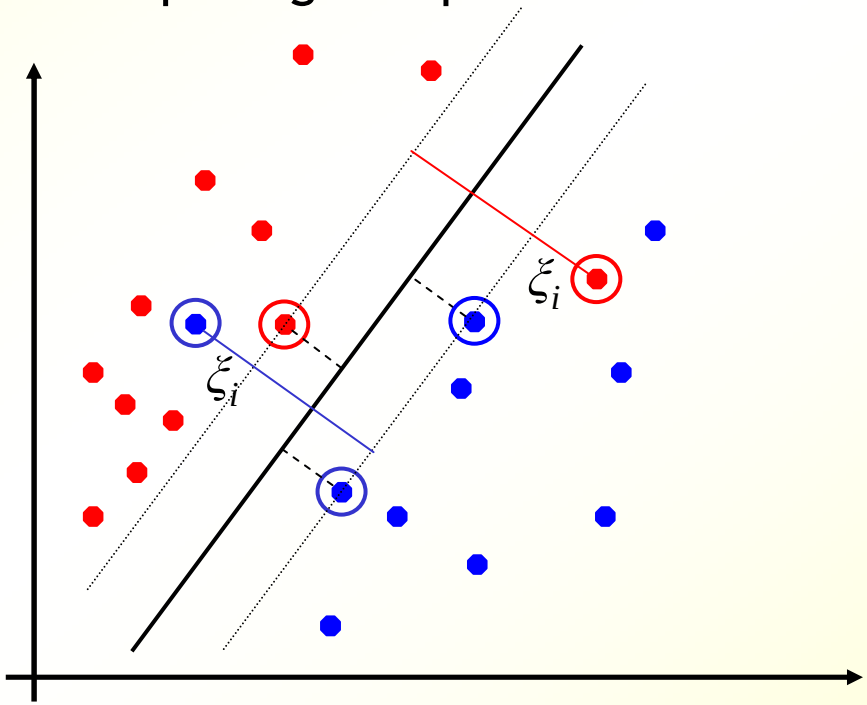
Support Vector Machines

- Casos:
  - Modelo lineal:
    - Los datos son separables linealmente (hard margin)
    - **Los datos no son separables linealmente (soft margin)**
  - Modelo no lineal (kernels)

Soft Margin, slack variables

Algunos datos pueden estar al otro lado del plano, debido a por ejemplo, el ruido

Solución: permitir que algunos puntos estén mal clasificados



Minimizar: $\mathbf{w}^T\mathbf{w} + C^*\Sigma\xi_i$

Soft Margin, slack variables

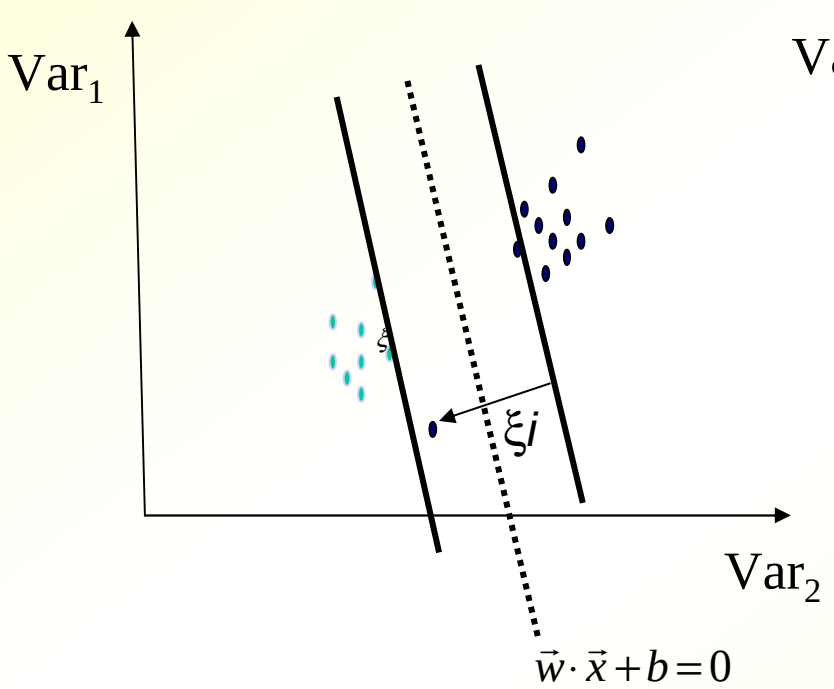
- Antes:
  - Maximizar el márgen $\rho = \frac{2}{\|\mathbf{w}\|}$
  - O lo que es lo mismo, minimizar $|\mathbf{w}|^2 = \mathbf{w}^T\mathbf{w}$
- Ahora: permitir "márgenes blandos" con variables de holgura:
  - Minimizar: $\mathbf{w}^T\mathbf{w} + C*\Sigma\xi_i$
- El parámetro *C* controla el peso que le damos a uno o a otro objetivo y permite controlar el sobreaprendizaje:
  - Si *C* tiene un valor grande …
  - Si *C* tiene un valor pequeño …

Soft Margin, slack variables

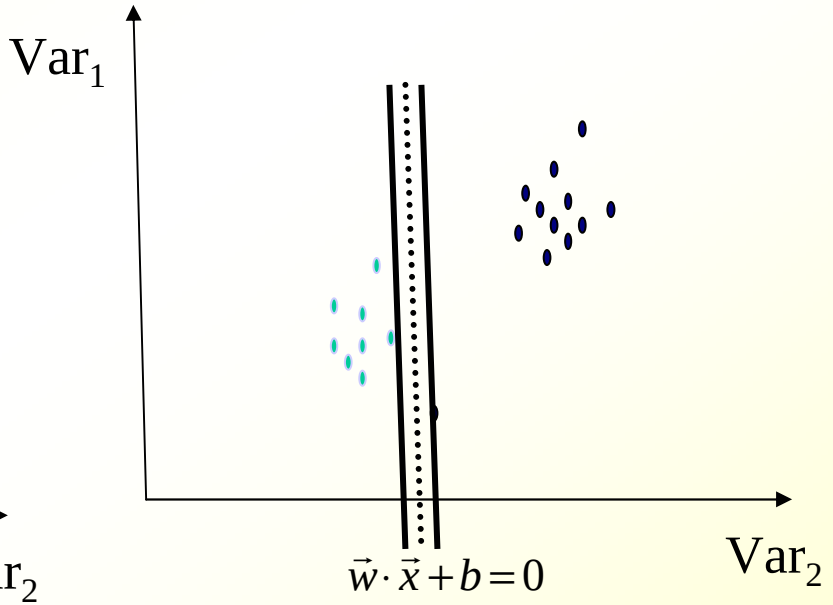
- Antes:
  - Maximizar el márgen $\rho = \frac{2}{\|\mathbf{w}\|}$
  - O lo que es lo mismo, minimizar $|\mathbf{w}|^2 = \mathbf{w}^T\mathbf{w}$
- Ahora: permitir "márgenes blandos":
  - Minimizar: $\mathbf{w}^T\mathbf{w} + C*\Sigma\xi_i$
- El parámetro *C* controla el peso que le damos a uno o a otro componente y permite controlar el sobreaprendizaje:
  - Si *C* tiene un valor grande, se le da mucha importancia a que todos los datos de entrenamiento se clasifiquen correctamente (slack variables tienden a cero). Si hay ruido, puede haber overfitting
  - Si *C* tiene un valor pequeño ocurre lo contrario. Si es demasiado pequeño, puede ocurrir que haya demasiados datos de entrenamiento mal clasificados (muchas slack variables con valores altos)
  - Si *C* es muy grande, el resultado es similar al de hard margin

Soft Margin vs. Hard Margin

Minimizar: $\mathbf{w}^T\mathbf{w} + C^*\Sigma\xi_i$



Soft Margin SVM (para un valor apropiado del coste *C)*



Hard Margin SVM (o Soft Margin con *C* muy grande)

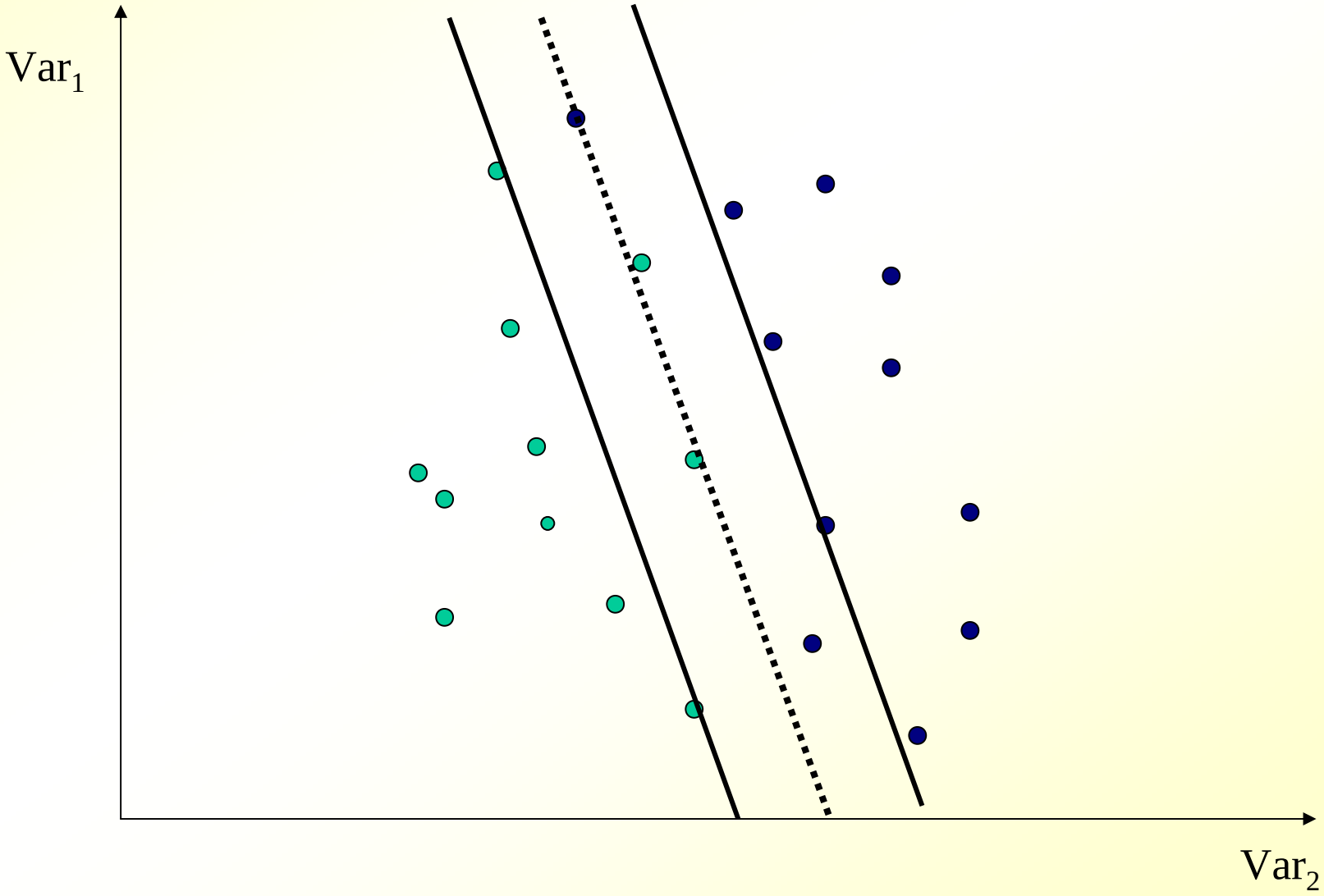
Soft Margin vs. Hard Margin

- Soft Margin es más robusta al ruido y a los outliers
- Soft Margin siempre tiene una solución (Hard Margin no, si los datos no son linealmente separables)
- Pero Soft Margin requiere estimar el coste (el parámetro *C*)

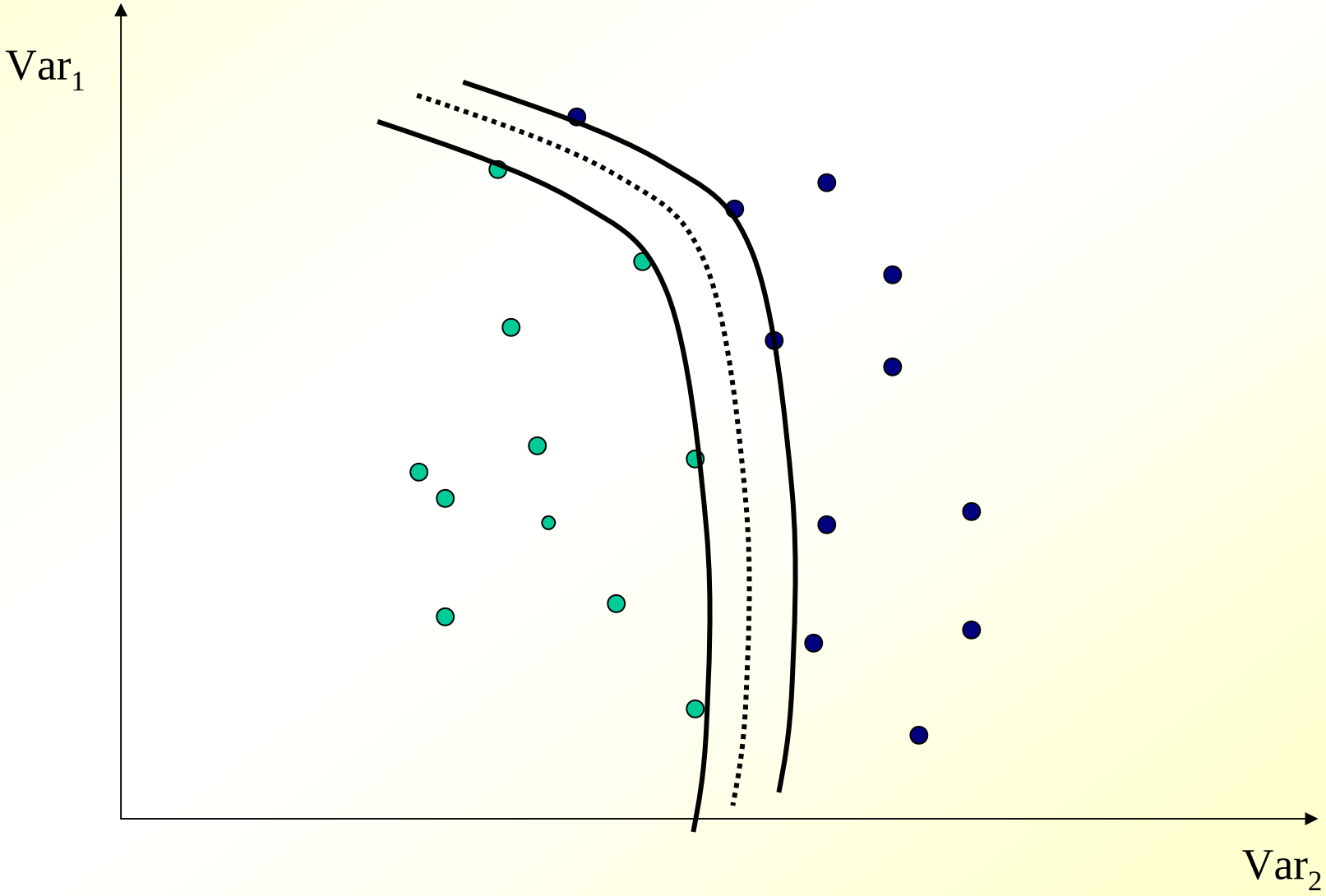
Support Vector Machines

- Casos:
  - Modelo lineal:
    - Los datos son separables linealmente
    - Los datos no son separables linealmente (soft margin)
  - **Modelo no lineal (kernels)**

Modelo lineal vs. Modelo no lineal

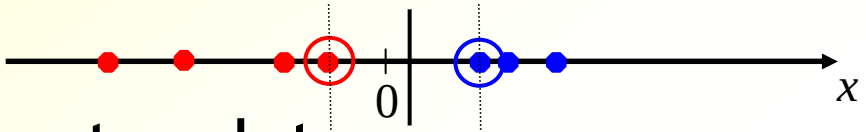


*Modelo lineal vs. **Modelo no lineal***

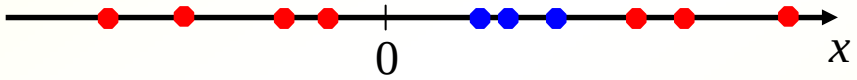


Clasificación no lineal biclase

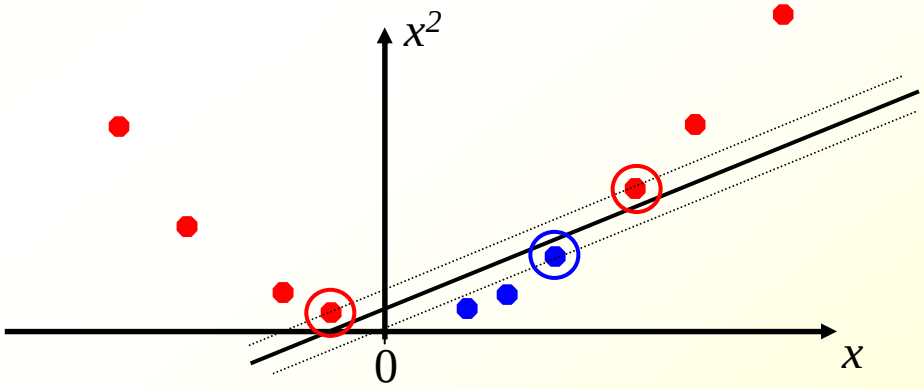
- Caso lineal en una dimensión:



- Pero estos datos no son separables linealmente

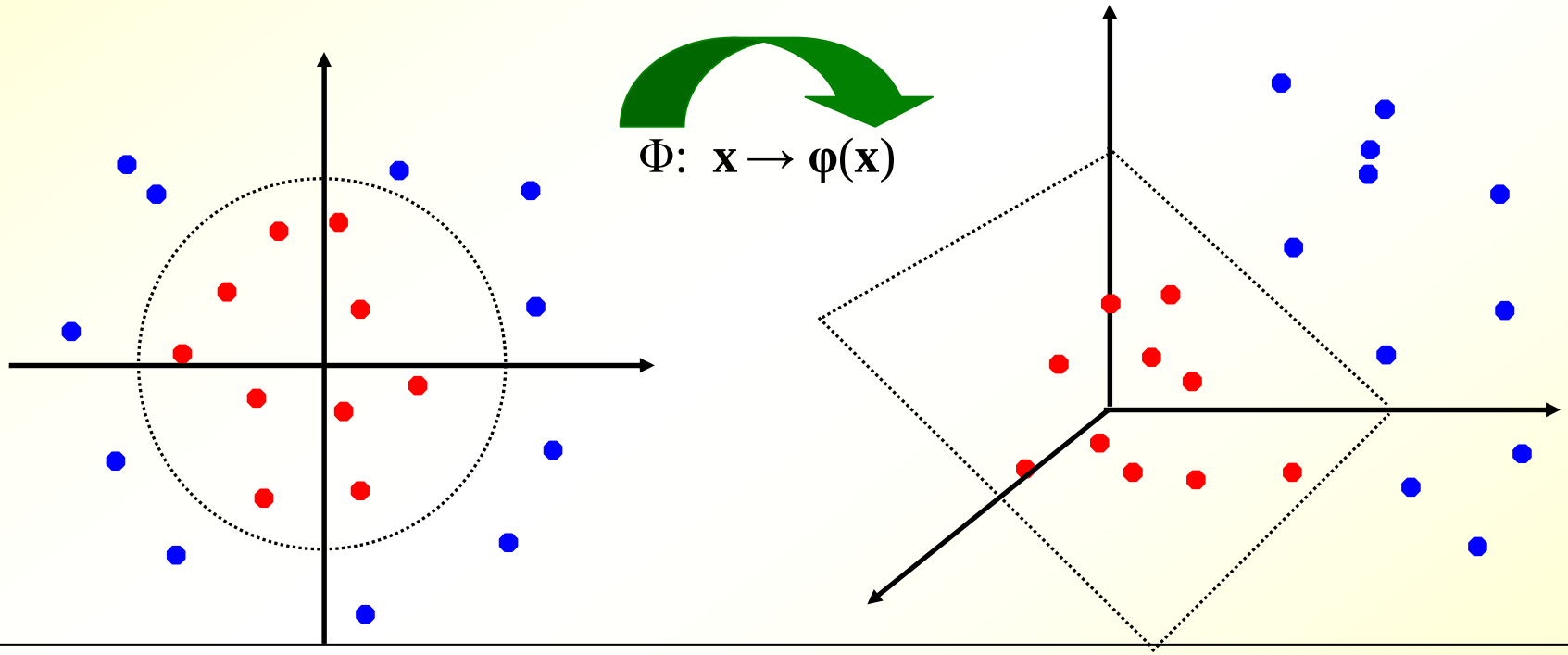


- ¿Y si los proyectamos a dos dimensiones?



Proyecciones

Proyección de dos a tres dimensiones



Datos no separables linealmente en dos dimensiones, pero si en tres dimensiones, con la proyección adecuada

Es decir, una separación lineal en el espacio proyectado corresponde a una separación circular en el espacio original

Kernels

- Es decir, se reduce el caso no lineal al lineal mediante la proyección. El algoritmo de optimización a utilizar es el mismo
- Kernel: es una función que realiza la proyección de manera **implícita**
- Tipos:
  - Lineal: no hace proyección
  - Polinómico: proyección al espacio de atributos producto
  - Gaussiano: proyección a infinitas dimensiones
  - Otros …

Kernel polinómico $K(x \cdot z) = (x \cdot z + 1)^p$

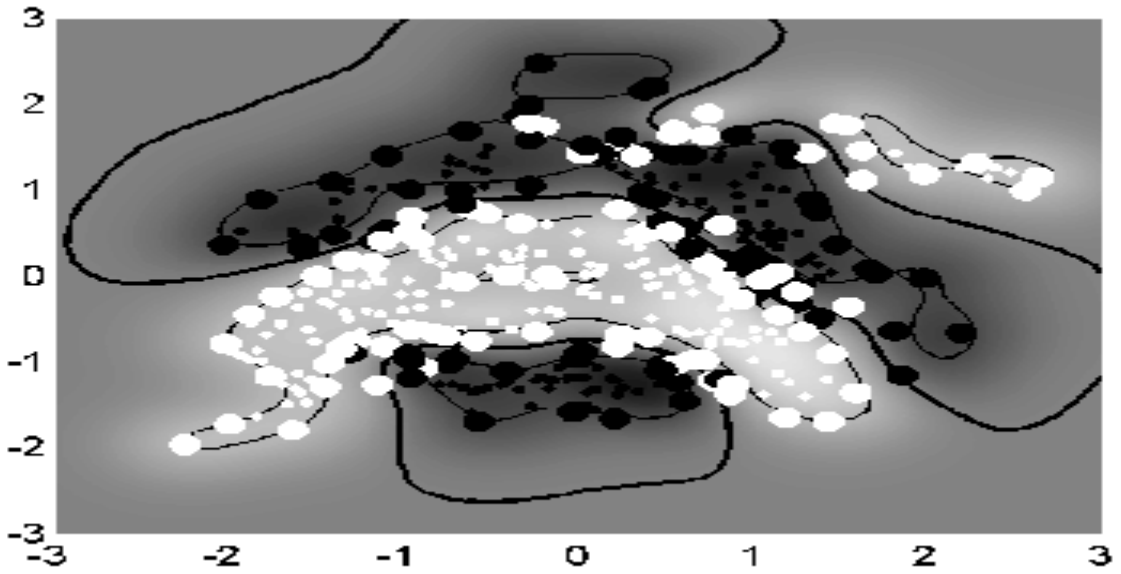
- Un parámetro *p* (dos, si es $(x^*z+b)^p$)
- p = 1: proyección lineal (o sea, sin proyección)
- p = 2: proyección cuadrática.
  - Ej: si partimos de un espacio de datos de dos dimensiones, se proyecta a un espacio de 6 dimensiones donde los atributos son:

    $$[1,\ x_1^2, \sqrt{2}\, x_1 x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2]$$

  - El tipo de separaciones son cónicas: parábolas, elipses, circunferencias, hipérbolas
- p = n: el espacio proyectado consta de atributos producto (monomios de grado n)

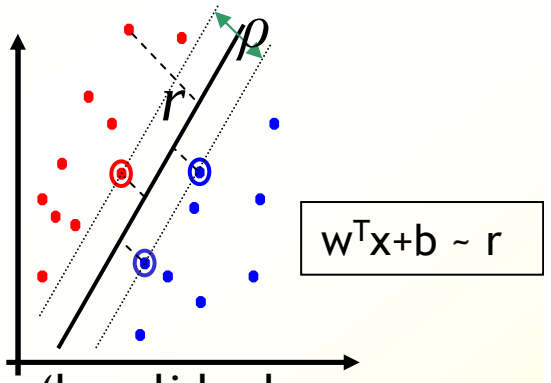
Kernel gaussiano $K(x \cdot z) = \exp(-\|x - z\| / 2\sigma^2)$

- Proyecta a infinitas dimensiones
- Es de los kernels más genérico
- Un parámetro: sigma
- Para ciertos valores de sigma, es casi equivalente a separadores lineales

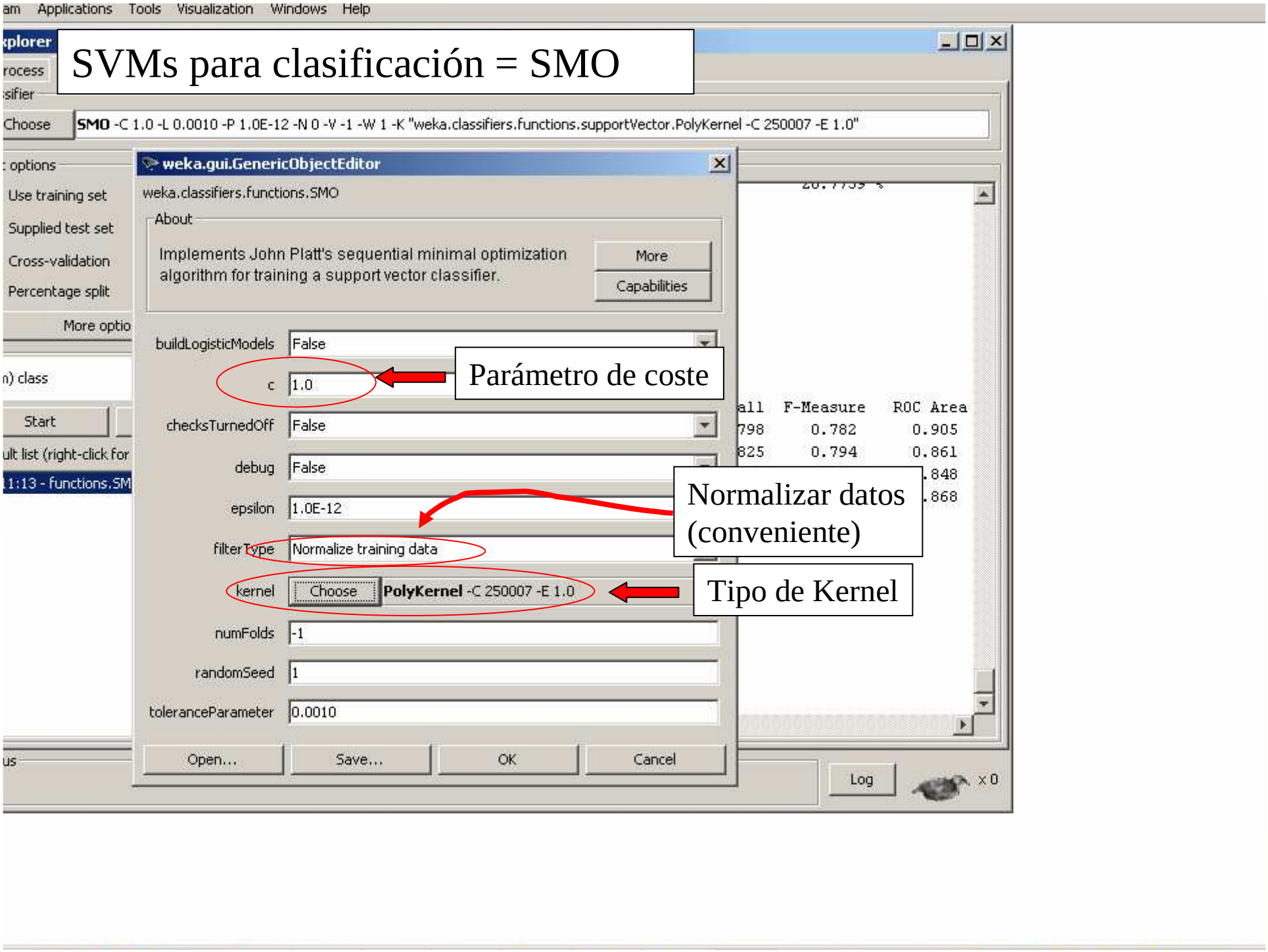


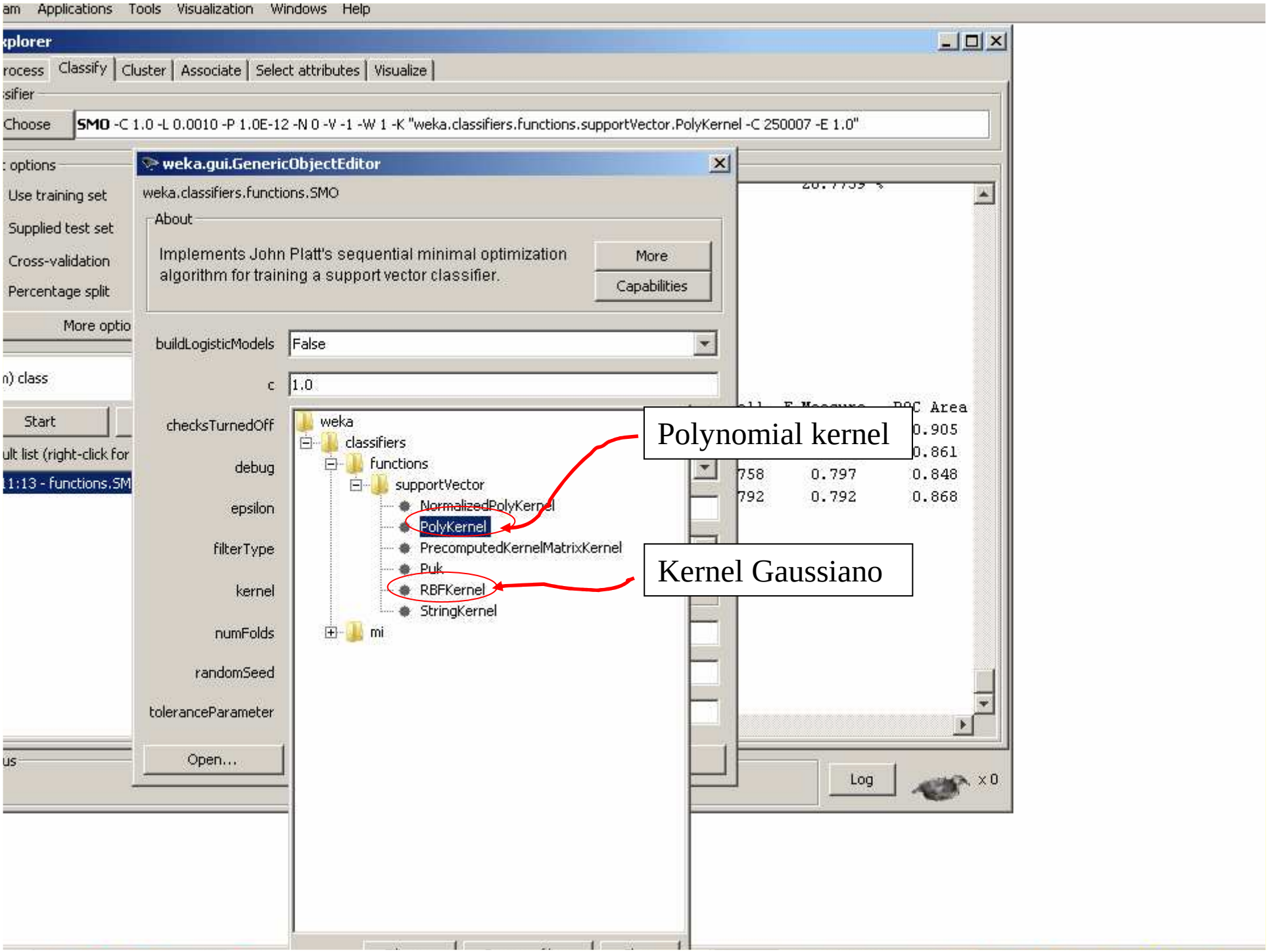
Clasificación con multiples clases

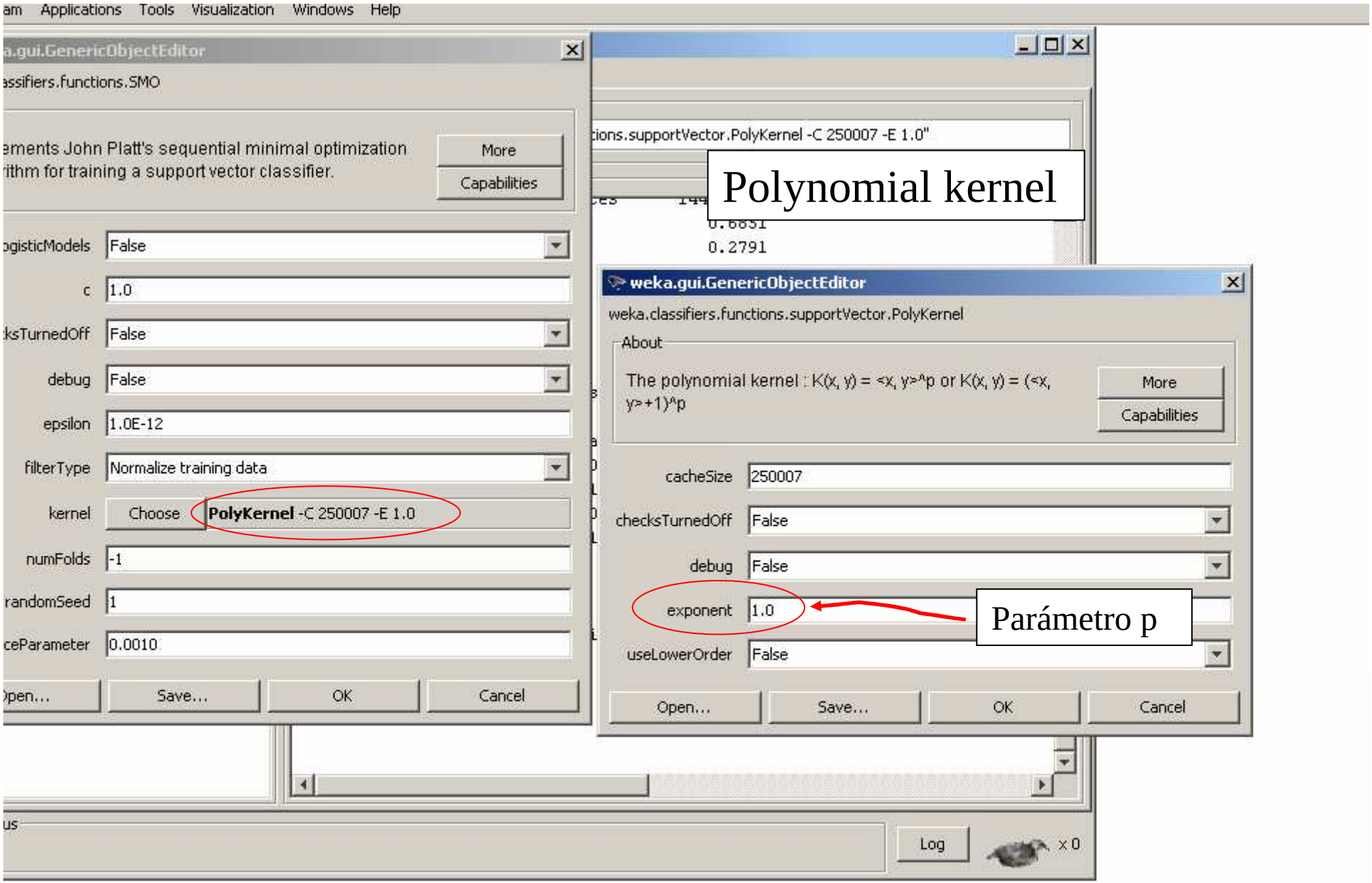
- One-versus-all:
  - Ej con 3 clases, generar:
    - Separador 1 vs. 2+3
    - Separador 2 vs. 1+3
    - Separador 3 vs. 1+2
    - En test, escoger el separador que de mayor valor (la salida de una SVM es la distancia al hiperplano)

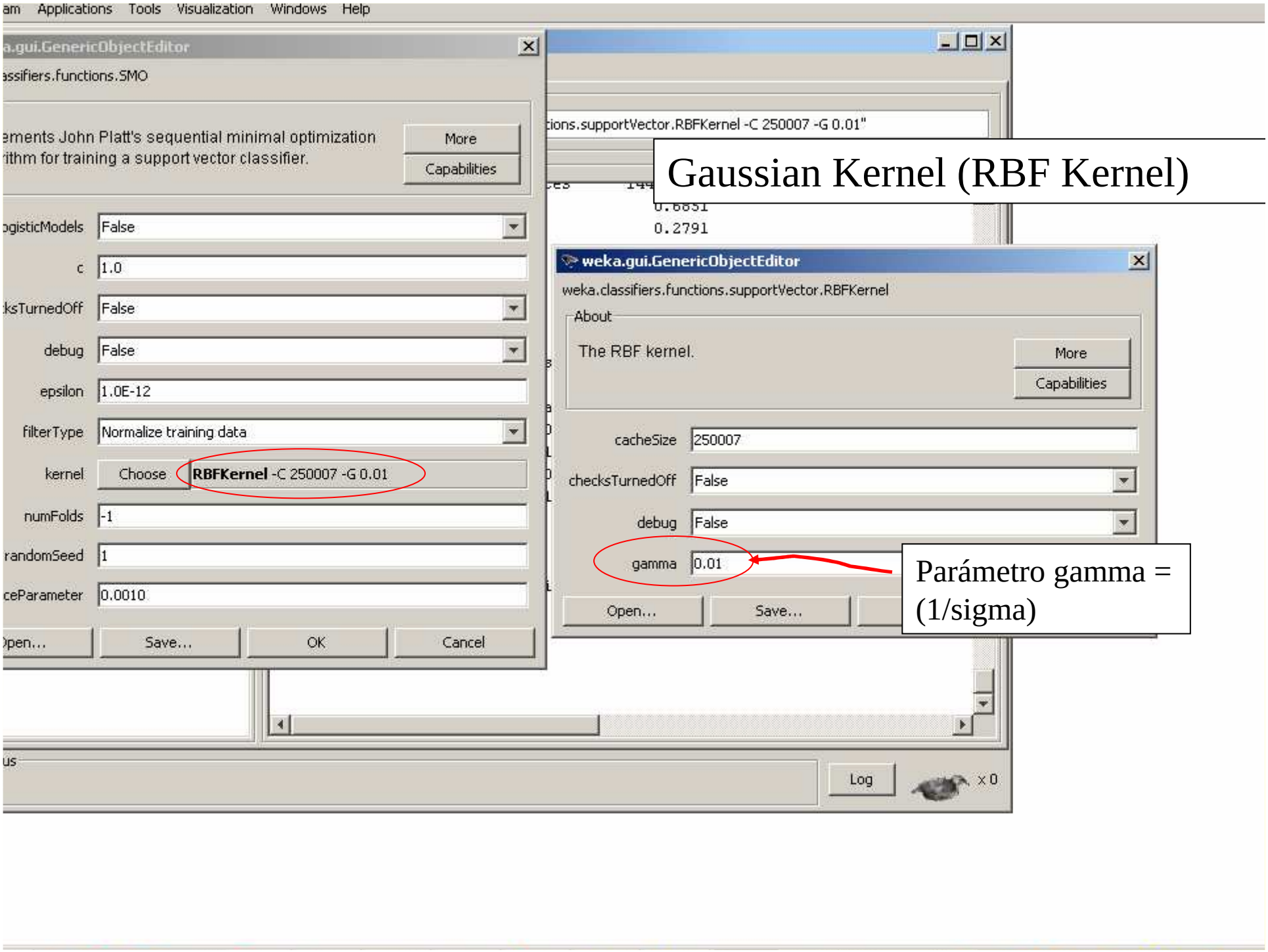


- One-versus-one: (solución adoptada por SMO de Weka)
  - Ej con 3 clases, generar:
    - Separador 1 vs. 2
    - Separador 1 vs. 3
    - Separador 2 vs. 3
    - En test, por votación
- Hay SVMs que permiten multiclase

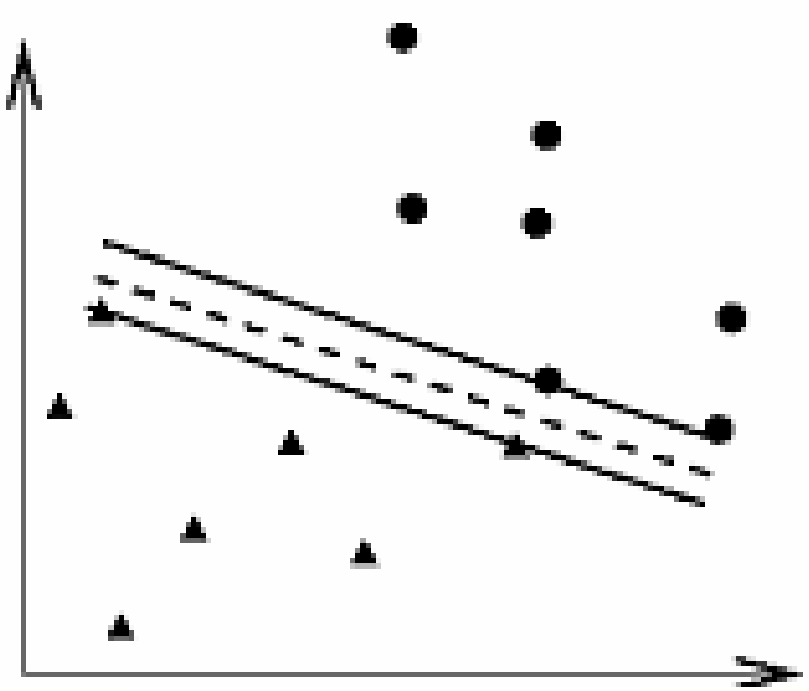

SVMs para clasificación = SMO
Applications Tools Visualization Windows Help
Choose
SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
weka.gui.GenericObjectEditor
weka.classifiers.functions.SMO
About
Implements John Platt's sequential minimal optimization algorithm for training a support vector classifier.
More
Capabilities
buildLogisticModels
False
c
1.0
Parámetro de coste
checksTurnedOff
False
debug
False
epsilon
1.0E-12
filterType
Normalize training data
Normalizar datos (conveniente)
kernel
Choose
PolyKernel -C 250007 -E 1.0
Tipo de Kernel
numFolds
-1
randomSeed
1
toleranceParameter
0.0010
Open...
Save...
OK
Cancel
Use training set
Supplied test set
Cross-validation
Percentage split
Start
F-Measure
ROC Area
0.782
0.905
0.794
0.861
Log


Applications Tools Visualization Windows Help
Classify
Cluster
Associate
Select attributes
Visualize
Choose
SMO -C 1.0 -L 0.0010 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -C 250007 -E 1.0"
weka.gui.GenericObjectEditor
weka.classifiers.functions.SMO
About
Implements John Platt's sequential minimal optimization algorithm for training a support vector classifier.
More
Capabilities
buildLogisticModels
False
c
1.0
checksTurnedOff
debug
epsilon
filterType
kernel
numFolds
randomSeed
toleranceParameter
Open...
weka
classifiers
functions
supportVector
NormalizedPolyKernel
PolyKernel
PrecomputedKernelMatrixKernel
Puk
RBFKernel
StringKernel
mi
Polynomial kernel
Kernel Gaussiano
Use training set
Supplied test set
Cross-validation
Percentage split
Start
Log
x 0

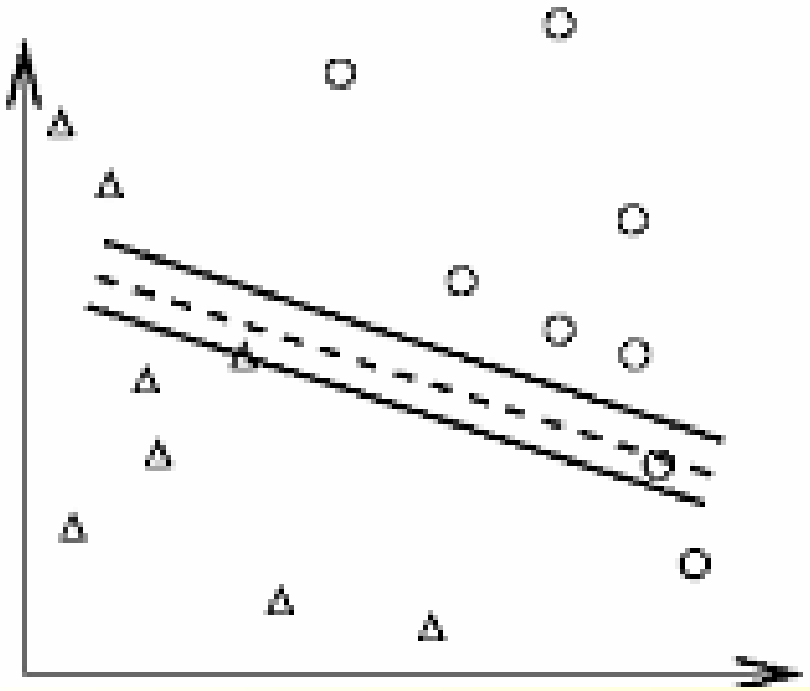

Applications
Tools
Visualization
Windows
Help
ssifiers.functions.SMO
ements John Platt's sequential minimal optimization
ithm for training a support vector classifier.
More
Capabilities
ogisticModels
False
c
1.0
ksTurnedOff
False
debug
False
epsilon
1.0E-12
filterType
Normalize training data
kernel
Choose
PolyKernel -C 250007 -E 1.0
numFolds
-1
randomSeed
1
ceParameter
0.0010
Open...
Save...
OK
Cancel
ions.supportVector.PolyKernel -C 250007 -E 1.0"
Polynomial kernel
0.6851
0.2791
weka.gui.GenericObjectEditor
weka.classifiers.functions.supportVector.PolyKernel
About
The polynomial kernel : K(x, y) = <x, y>^p or K(x, y) = (<x, y>+1)^p
More
Capabilities
cacheSize
250007
checksTurnedOff
False
debug
False
exponent
1.0
Parámetro p
useLowerOrder
False
Open...
Save...
OK
Cancel
Log
x 0


Applications Tools Visualization Windows Help
a.gui.GenericObjectEditor
assifiers.functions.SMO
ements John Platt's sequential minimal optimization
ithm for training a support vector classifier.
More
Capabilities
ogisticModels False
c 1.0
ksTurnedOff False
debug False
epsilon 1.0E-12
filterType Normalize training data
kernel Choose RBFKernel -C 250007 -G 0.01
numFolds -1
randomSeed 1
ceParameter 0.0010
Open...
Save...
OK
Cancel
tions.supportVector.RBFKernel -C 250007 -G 0.01"
0.6851
0.2791
weka.gui.GenericObjectEditor
weka.classifiers.functions.supportVector.RBFKernel
About
The RBF kernel.
More
Capabilities
cacheSize 250007
checksTurnedOff False
debug False
gamma 0.01
Open...
Save...
Log
x 0
Gaussian Kernel (RBF Kernel)
Parámetro gamma = (1/sigma)

Model selection (Determinación de parámetros). Overfitting.



Entrenamiento



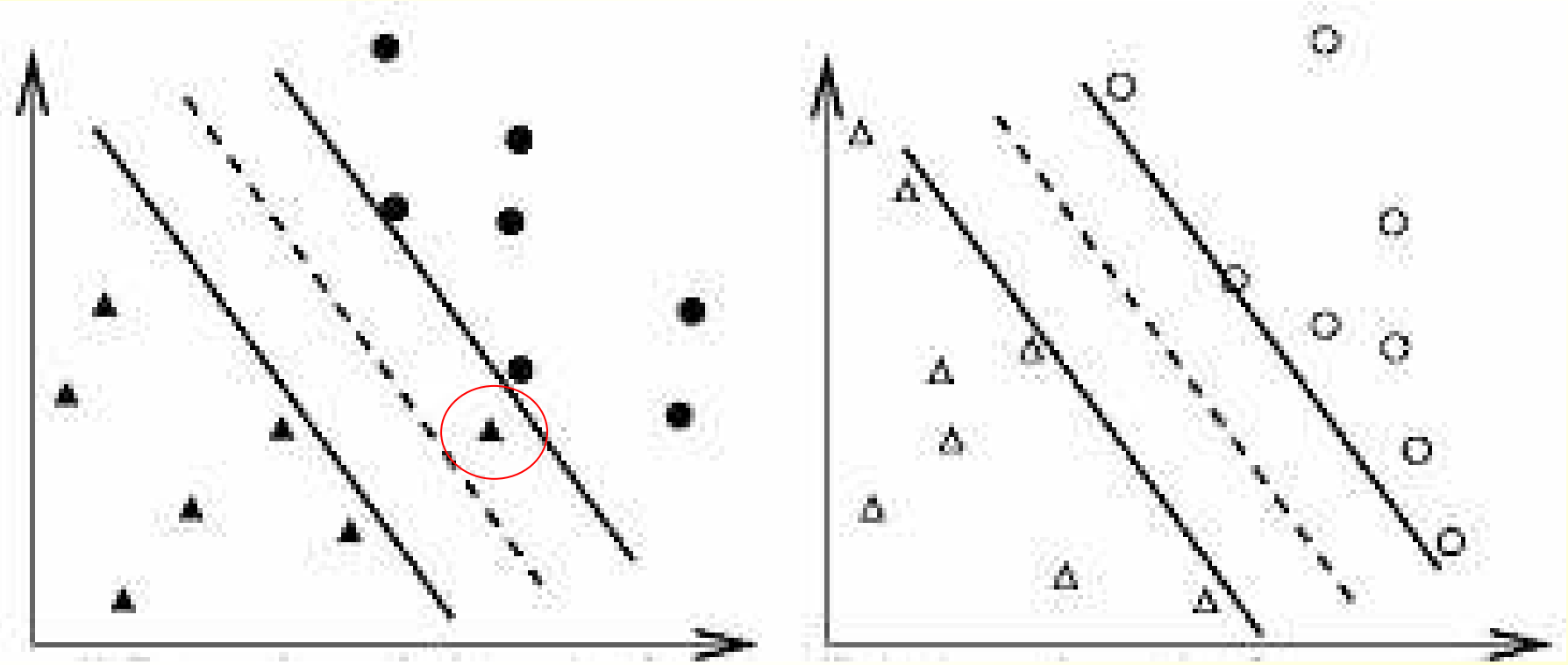
Test

EJEMPLO DE SOBREAPRENDIZAJE

Hard Margin, C grande,

¿Tal vez sería mejor utilizar otros parámetros?

Model selection (Determinación de parámetros). Sin overfitting.



Entrenamiento **Test**

EJEMPLO DE BUENA GENERALIZACIÓN

Soft Margin, C pequeña

Metodología para el uso de SVMs

1. Normalizar (escalado) los atributos
2. Encontrar los parámetros más apropiados
   - Parámetro *C* (coste, complejidad):
     - Valor grande: complejidad mayor, posible sobreaprendizaje
     - Valor pequeño: complejidad menor, tal vez mejor generalización, tal vez underfitting (infra-aprendizaje)
   - Kernel:
     - Lineal: ninguno
     - Polinómico: exponente (p)
     - Gaussiano: gamma
3. Construir el modelo con los mejores parámetros y todo el conjunto de entrenamiento

- Nota: probar primero con un kernel lineal, y después con uno Gaussiano.

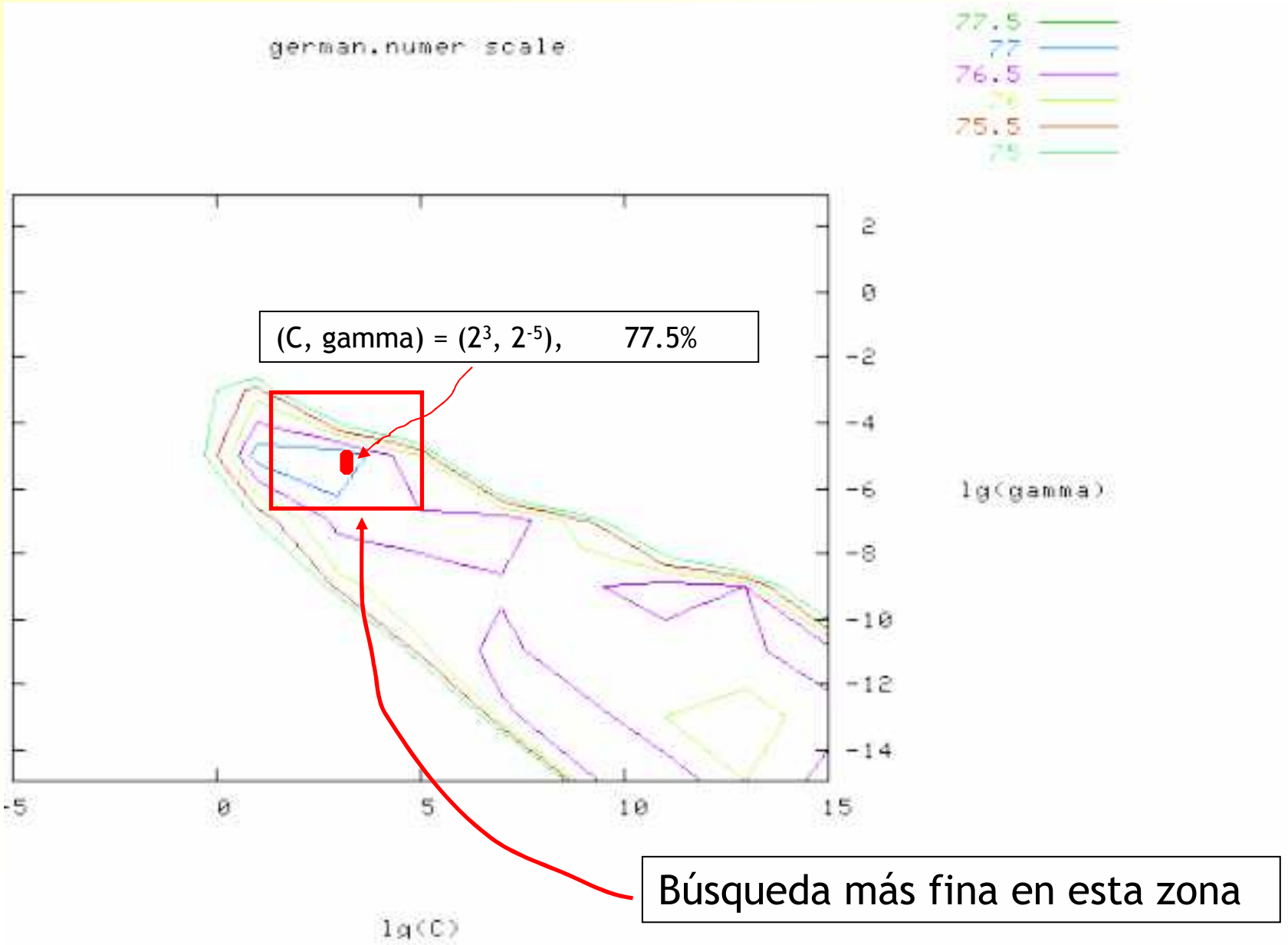
Metodología. Normalización de los datos (escalado)

- Sea un dato x = (x1, x2, …, xn, clase)
- Hay que normalizar cada atributo para que estén todos en el mismo rango y no tenga unos mas peso que otros
- Lo siguiente normaliza los atributos al rango [0,1]
- Atributo normalizado $x_i' = (x_i - x_{min})/(x_{max} - x_{min})$
- SMO de Weka ya lo hace por omisión

Metodología. Selección de parámetros

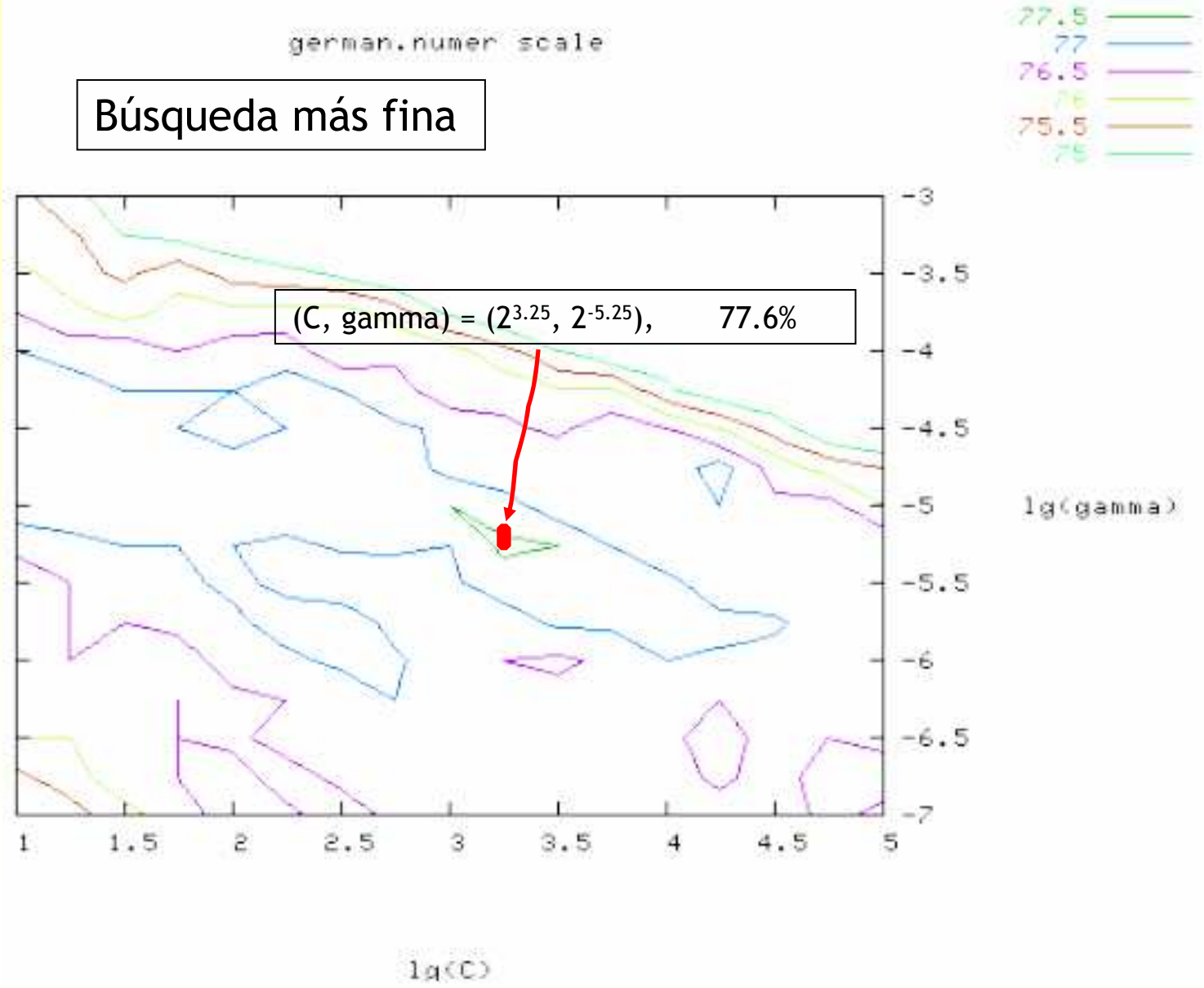
- Ejemplo, para el **Kernel Gaussiano** (RBF kernel), 2 parámetros: C y gamma
- Búsqueda en grid ("enrejado"), hacer validación cruzada para cada combinación de valores (C, gamma)
- Buen rango de búsqueda (logarítmico):
  - C = 2^{-5}, 2^{-3}, 2^{-1}, 2^{3}, …, 2^{15} ,(10 valores en total)
  - Gamma = 2^{-15} , 2^{-13}, …, 2^{3} ,(8 valores en total)

  - Costoso: ¡hay que hacer 80 validaciones cruzadas! (con los datos del BCI me llevó 3.5 horas)
  - Una vez identificada una buena zona, hacer una búsqueda en grid más fina en la zona deseada con pasos de 0.25
  - Por último habrá que aprender el modelo con esos parámetros y TODO el conjunto de entrenamiento



2: Loose grid search on $C = 2^{-5}, 2^{-3}, \ldots, 2^{15}$ and $\gamma = 2^{-15}, 2^{-13}, \ldots, 2^{3}$.

Búsqueda más fina



3: Fine grid-search on $C = 2^1, 2^{1.25}, \ldots, 2^5$ and $\gamma = 2^{-7}, 2^{-6.75}, \ldots, 2^{-3}$.

Uso de libsvm en Weka en Windows

- Descargaremos la librería de:

  http://www.csie.ntu.edu.tw/~cjlin/libsvm/

  Download libsvm 2.9
- Descomprimir el archivo .zip
- Buscar el archivo libsvm.jar
- Crear un fichero .jar, añadiendo la localización de libsvm.jar en el CLASSPATH

Fichero batch para libsvm en Weka (Windows)

```
@echo off
javaw -Xmx1024m -classpath "C:\Program Files
   (x86)\Weka-3 6\weka.jar;C:\Users\aler\libsvm-
   2.9\java\libsvm.jar" weka.gui.Main
```

Uso de liblinear en Weka en Windows

- LibLinear es un separador lineal basado en SVM, muy rápido y con solo el parámetro de coste *C*
- La página de liblinear está en:

  http://www.csie.ntu.edu.tw/~cjlin/liblinear/
- Pero el liblinear en java para Weka está en:

  http://www.bwaldvogel.de/liblinear-java/

- Hay que añadir liblinear en el CLASSPATH, en el fichero bat

Fichero batch para liblinear en Weka (Windows)

```
@echo off
javaw -Xmx1024m -classpath "C:\Program Files
   (x86)\Weka-3 6\weka.jar;C:\Users\aler\libsvm-
   2.9\java\libsvm.jar;C:\Users\aler\libsvm-
   2.9\java\liblinear-1.5.jar" weka.gui.Main
```

LIBLINEAR y LIBSVM EN WINDOWS

- Ambos tienen ejecutables en Windows, escritos en C++, que son más rápidos.
- También tienen utilidades escritas en python para hacer la búsqueda de parámetros
- No es necesario que utiliceis estos ejecutables