# Lab 9. Clasificación mediante SVM y Validación cruzada

[Lab 9. Clasificación mediante SVM y Validación cruzada 1](#_Toc5603)

[Objetivos 1](#_Toc5604)

[1) Parte 1: Crear las K divisiones para la validación cruzada 1](#_Toc5605)

[2) Parte 2: Clasificación usando SVM y validación cruzada. Cálculo de las métricas de rendimiento 2](#_Toc5606)

[3) EXTENSIÓN A LA PARTE 1 5](#_Toc5607)

## Objetivos

En esta práctica vamos a trabajar con el clasificador de las máquinas de vectores de soporte (*Support Vector Machine*: SVM). El método de evaluación del clasificador en esta práctica va a ser el de la validación cruzada en vez de la partición entre conjuntos de entrenamiento y test (*holdout*).

El script general para esta práctica será main\_lab9.m, y en él se irá completando el código para realizar los ejercicios planteados en la práctica.

Para esta práctica emplearemos el mismo *dataset* que en la práctica anterior: un *dataset* con datos de masas detectadas en mamografías para determinar si las masas que se detectaron son benignas o malignas[[1]](#footnote-1). En él se han eliminado los patrones con *missing values* (patrones en los que no existían todos los datos) y sus datos se normalizaron para que cada variable del conjunto tuviese media 0 y desviación típica 1.

Cada elemento del *dataset* (variable de Matlab X) se define mediante 5 variables. Las clases se indican en la variable de Matlab Y (Y(i)==0 indica que la masa descrita por el patrón X(i,:) es benigna; Y(i)==1 indica que la masa descrita por el patrón X(i,:) es maligna). Estas variables de Matlab se obtienen tras cargar el archivo mammographic\_data\_norm.mat. Como siembre, los patrones estarán contenidos en las columnas de las matrices de datos (es decir, cada patrón está en una columna distinta de la variable X).

**/\*-------------------CÓDIGO---------------------------------\*/**

**% ============================================================**

%% PRELIMINAR: CARGA CONJUNTO DE DATOS

load mammographic\_data\_norm.mat;

% X contiene los patrones de entrenamiento (cada patrón en una columna): Patrones de dimensión 5

% Y contiene la clase del patrón (1==maligna; 0==benigna)

% Número de patrones (elementos) y de variables por cada patrón en este dataset

[num\_patrones, num\_variables] = size(X);

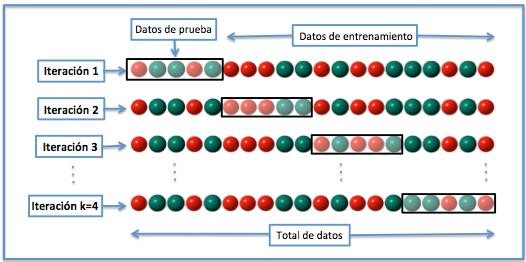
**% ============================================================**

/\*---------------------------------------------------------------------------------------------\*/

## 1) Parte 1: Crear las K divisiones para la validación cruzada

Medir el error de un clasificador utilizando los mismos datos que se usaron para entrenarlo no es una buena práctica. En prácticas anteriores se ha dividido el conjunto de datos en dos conjuntos disjuntos: uno para el entrenamiento (de aproximadamente el 70% de los datos) y otro para realizar el test (con el 30% de datos restantes). En esta práctica vamos a realizar la evaluación mediante la técnica de la **validación cruzada con *K-folds***.

Lo que hace *K-folds* es dividir el conjunto original en *K* partes (en el caso de esta práctica se ha fijado ***K* = 10**). Posteriormente se realizará el entrenamiento mediante SVM utilizando 9 de estas partes (conjunto de entrenamiento) y 1 para la clasificación (conjunto de test). Este proceso se repetirá 10 veces, tomando en cada una de ellas un conjunto de test diferente, de forma que todas las partes se hayan utilizado para entrenamiento y test al menos una vez. Finalmente, **el resultado (*accuracy*, *FScore*, etc.) será una media de los resultados obtenidos en cada una de las 10 iteraciones**.



*Figura 1. Ejemplo de validación cruzada con k-folds con k=4*

En esta práctica los índices para el *K-folds* se crean mediante la función:

**indices\_folds = crear\_k\_fold(Y, k)**

**Abre dicha función y complétala**. Después de llamar a la función desde el script principal main\_lab9.m, se imprimen por pantalla las proporciones de elementos de cada clase que hay en cada división. Esto sirve como comprobación de que se ha realizado bien y para comparar las proporciones de cada división con las proporciones del conjunto de datos original. **Idealmente las proporciones deberían ser similares**.

/\*-------------------------CÓDIGO-----------------------------------------------------\*/

**% ============================================================**

**FUNCION CREAR KFOLDS**

function indices\_folds = crear\_k\_fold(Y, k)

% Esta función crea un vector (matriz fila) con tantos elementos como

% patrones que contiene el índice del fold en el que estará el correspondiente

% patrón. Esta asignación se realiza aleatoriamente.

% Es decir, en la posición i-ésima del vector indices\_folds estará el número

% de fold (ENTRE 1 Y K) en el que estará el patrón i-ésimo. LA FUNCIÓN ESTÁ

% DISEÑADA PARA PROBLEMAS DE DOS CLASES.

%

% INPUT

% - Y: Vector que indica las clases de los patrones.

% - k: Número de folds que hay que generar.

%

% OUTPUT

% indices\_folds: vector (matriz fila) con la misma longitud que Y.

% Contiene el índice del fold en el que estará el correspondiente patrón.

% Es decir, en la posición i-ésima del vector de salida estará el número

% de fold en el que estará el patrón i-ésimo.

%

nPorClase = length(Y)/k;

modulo = mod(length(Y),k);

v = [];

for i=1:nPorClase

v = horzcat(v, randperm(k)); %%concatena el trozo con el anterior y randperm, hace un random con permutaciones

end

if(modulo ~= 0)

v = horzcat(v, randperm(modulo));

end

indices\_folds = v;

% m = zeros(1,length(Y));

%

% for i=1:length(Y)

% p = 0;

%

% while(p == 0)

% r = randi([1 k],1,1); %%randi: aleatorios

% en = find(m==r);

% nRepetido = length(en);

% if((nRepetido < nPorClase))

% element = modulo;

% while(element ~=0)

% m(i) = r;

% element = element-1;

% end

%

% m(i)=r;

% p = 1;

% end

% end

count0 = 1;

count1 = 1;

inds0=find (Y==0);

inds1=find (Y==1);

clase0 = mod(randperm(length(inds0)),k)+1;%% se suma 1 porque va desde 0 hasta k-1

clase1 =mod(randperm(length(inds1)),k)+1;

for i = 1 : length(Y)

if Y(i) == 0

indices\_folds(i) = clase0(count0);

count0 = count0 + 1;

elseif Y(i) ==1

indices\_folds(i) = clase1(count1);

count1 = count1 + 1;

end

end

end

**% ============================================================**

/\*------------------------------------------------------------------------------------------\*/

/\*--------------------------------CÓDIGO----------------------------------------------\*/

**% ============================================================**

MAIN:

k = 10;

% Crea las K divisiones para realizar la validación cruzada

indices\_folds = crear\_k\_fold(Y, k);

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\*\*\*\*\*\*\* COMPROBACIÓN DE PROPORCIONES EN LOS FOLDS \*\*\*\*\*\*\*\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\n');

fprintf('COMPLETE DATASET: %4.2f (Class 0) - %4.2f (Class 1)\n', 100\*sum(Y==0)/length(Y), 100\*sum(Y==1)/length(Y));

fprintf('\n');

for num\_fold=1:max(indices\_folds)

indsFolds = indices\_folds==num\_fold;

numClass0\_k = sum(Y(indsFolds)==0);

numClass1\_k = sum(Y(indsFolds)==1);

fprintf('FOLD %d: %4.2f (Class 0) - %4.2f (Class 1)\n', num\_fold, 100\*numClass0\_k/sum(indsFolds), 100\*numClass1\_k/sum(indsFolds));

end

fprintf('\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\n');

fprintf('\n');

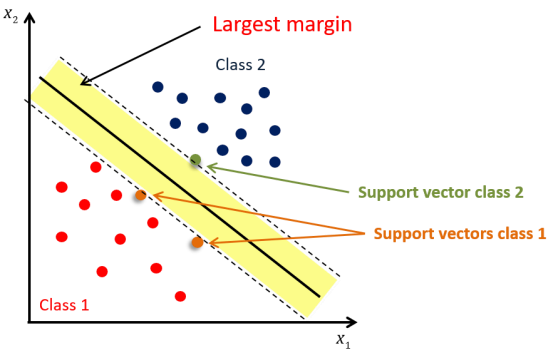
fprintf('\n');

**% ============================================================**

/\*-----------------------------------------------------------------------------------------------\*/

## 2) Parte 2: Clasificación usando SVM y validación cruzada. Cálculo de las métricas de rendimiento

SVM es un clasificador lineal que intenta separar las clases en el espacio de características mediante un hiperplano que tenga los mayores márgenes posibles con las clases (ver Figura 2).



*Figura 2. Clasificación SVM con margen máximo*

Para clasificar con SVM usaremos la librería LibSVM[[2]](#footnote-2) (incluída con el código de esta práctica). Sus funciones principales son **svmtrain** y **svmpredict**, que se usan para entrenar el SVM y clasificar con él, respectivamente. A continuación, se copia la **documentación para estas dos funciones** provista en el fichero README de la librería:

**¡¡¡CUIDADO!!! En esta práctica se han renombrado los ficheros svmtrain y svmpredict a svmtrain\_2 y svmpredict\_2 para que no hubiera**

**“colisión” con función svmtrain de Matlab.**

Usage

=====

**model = svmtrain(training\_label\_vector, training\_instance\_matrix [, 'libsvm\_options']);**

-training\_label\_vector: **An m by 1 vector** of training labels

(type must be double).

-training\_instance\_matrix: **An m by n matrix of m training instances with n features.**

-libsvm\_options: A **string** of training options in the same format as that of LIBSVM.

**[predicted\_label] = svmpredict(testing\_label\_vector, testing\_instance\_matrix, model [, 'libsvm\_options']);**

**Or**

**[predicted\_label, accuracy, decision\_values/prob\_estimates] = svmpredict(testing\_label\_vector, testing\_instance\_matrix, model [, 'libsvm\_options']);**

-testing\_label\_vector: **An m by 1 vector** of prediction labels. If labels of test data are unknown, simply use any random values. (type must be double)

-testing\_instance\_matrix: **An m by n matrix of m testing instances with n features.**

-model: The output of svmtrain.

-libsvm\_options: A string of testing options in the same format as that of LIBSVM.

Las libsvm\_options que se usan en **svmtrain** están ya indicadas en el script main\_lab\_9, por lo que no es necesario que os preocupéis de ello. **TEN CUIDADO CON LAS DIMENSIONES DE LAS MATRICES DE DATOS (X) Y DE CLASES (Y): ES POSIBLE QUE TENGAS QUE TRANSPONERLAS PARA UTILIZARLAS CON SVMTRAIN Y SVPREDICT.**

En un proceso iterativo en el que se realizarán **K iteraciones** (una por cada *fold*) se realizará la siguiente secuencia:

1. Se entrena el clasificador SVM con los patrones de los K-1 *folds* correspondientes.
2. Se utiliza el SVM entrenado para clasificar los patrones del *fold* correspondiente a la iteración actual.
3. Se calculan las métricas de error para el resultado de la clasificación de los elementos del *fold* actual y se almacenan en un vector para poder calcular su media y desviación típica al terminar el proceso.

Al final del proceso se calcularán **la media y desviación típica** de cada una de las métricas de error calculadas anteriormente, que se considerarán como el resultado final para la evaluación del clasificador evaluado mediante validación cruzada.

**Se pide que calcules el *accuracy* y el *FScore* de la clasificación de los elementos de cada *fold* y los almacenes en sendos vectores**. El objetivo es que al final del bucle puedas imprimir por pantalla los resultados de cada *fold* y la media y desviación típica de los resultados de todos los folds.

Recordamos que el accuracy y FScore se pueden calcular de la siguiente manera, a partir de los resultados de la matriz de confusión.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **Clase predicha** | |
| 1 | 0 |
| **Clase real** | 1 | TP | FN |
| 0 | FP | TN |

*TP TN*+

* **Accuracy**: *accuracy* = *TP TN*+ + +*FP FN*

*TP*

* **Precision**: *precision* = *TP*+ *FP*

*TP*

* **Recall**: *recall* = *TP*+ *FN*

*precision recall*·

* **FScore**: *FScore* =2

*precision*+*recall*

/\*-------------------------------CÓDIGO----------------------------------------------------------\*/

**% ============================================================**

addpath('libsvm-3.20/matlab');

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% Clasificación usando Validación Cruzada

% Parámetro C de SVM. PROBAR CON DIFERENTES Y VER SU IMPACTO EN LA

% CLASIFICACIÓN

C\_val = 1;

% Inicialización de vectores de resultados

accuracies = zeros(1, num\_fold);

FScores = zeros(1, num\_fold);

for num\_fold = 1:max(indices\_folds)

% Obtención de los índices asignados al num\_fold-ésimo fold.

% indsTest es un vector donde estén las posiciones donde indices\_folds

% valga num\_fold

% indsTrain es un vector donde están las posiciones donde indices\_folds

% tenga un valor distinto de num\_fold

% ====================== YOUR CODE HERE ======================

indsTest=find(indices\_folds == num\_fold);

indsTrain=find(indices\_folds ~= num\_fold);

% ============================================================

% Separación entre conjuntos de test (con los patrones que

% están en el fold num\_fold) y entrenamiento (todos los demás)

% SE TRANSPONEN LAS Y PORQUE LIBSVM ESPERA LOS VECTORES CON LAS CLASES

% COMO VECTORES COLUMNA. LAS X NO SE TRANSPONEN PORQUE LOS PATRONES DEL

% CONJUNTO DE DATOS TIENEN QUE ESTAR EN FILAS

X\_test\_fold = X(indsTest, :);

Y\_test\_fold = Y(indsTest)';

X\_train\_fold = X(indsTrain, :);

Y\_train\_fold = Y(indsTrain)';

% Entrenamiento de SVM

opt\_string=['-t 0 -e 0.001 -c ' num2str(C\_val)];

% ====================== YOUR CODE HERE ======================

model = svmtrain\_2(Y\_train\_fold, X\_train\_fold, opt\_string);

% ============================================================

% Test de SVM

% ====================== YOUR CODE HERE ======================

predict = svmpredict\_2(Y\_test\_fold, X\_test\_fold, model);

% ============================================================

% Cálculo del accuracy para este fold

% ====================== YOUR CODE HERE ======================

TP = 0;

TN = 0;

FP = 0;

FN = 0;

for i = 1: length(Y\_test\_fold)

if Y\_test\_fold(i) == predict(i) && Y\_test\_fold(i) == 1

TP = TP +1;

elseif Y\_test\_fold(i) ~= predict(i) && Y\_test\_fold(i) == 1

FN = FN + 1;

elseif Y\_test\_fold(i) ~= predict(i) && Y\_test\_fold(i) == 0

FP = FP + 1;

elseif Y\_test\_fold(i) == predict(i) && Y\_test\_fold(i) == 0

TN = TN +1;

end

end

% El accuracy de cada fold se acumula en un vector para poder calcular

% su media y desviación típica fuera del bucle

accuracies(num\_fold) = (TP+TN)/(TP+TN+FP+FN);

% ============================================================

% Cálculo del F-Score para este fold

% ====================== YOUR CODE HERE ======================

precision = TP/(TP+FP);

recall = TP/(TP+FN);

% El FScore de cada fold se acumula en un vector para poder calcular

% su media fuera del bucle

FScores(num\_fold) = 2\*((precision\*recall)/(precision+recall));

% ============================================================

end

% \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

% B: Mostramos resultados

fprintf('\n\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\*\*\*\*\*\*\* RESULTADOS DE LA CLASIFICACIÓN CON VALIDACIÓN. CRUZADA \*\*\*\*\*\*\*\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\n');

for num\_fold=1:max(indices\_folds)

fprintf('FOLD %d: Accuracy = %4.2f - FScore = %4.2f (Class 1)\n', num\_fold, accuracies(num\_fold), FScores(num\_fold));

end

fprintf('\n');

fprintf('AVERAGE ACCURACY = %4.2f - STD ACCURACY = %4.2f \n', mean(accuracies), std(accuracies));

fprintf('AVERAGE FSCORE = %4.2f - STD FSCORE = %4.2f \n', mean(FScores), std(FScores));

fprintf('\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n');

fprintf('\n');

fprintf('\n');

fprintf('\n');

**% ============================================================**

/\*-----------------------------------------------------------------------------------------------\*/

## 3) EXTENSIÓN A LA PARTE 1

Al imprimir por pantalla las proporciones de las clases en cada *fold*, es posible que estas no sean similares. En ese caso, y teniendo en cuenta que esta función funcionará con problemas de dos clases, **intenta modificar la función para que las proporciones de cada clase fuesen similares en cada *fold* a las del *dataset* completo.**

1. Descrito en <https://archive.ics.uci.edu/ml/datasets/Mammographic+Mass>

   [↑](#footnote-ref-1)
2. <https://www.csie.ntu.edu.tw/~cjlin/libsvm/>

   [↑](#footnote-ref-2)