# Lab 7. K-Nearest Neighbours

[Lab 7. K-Nearest Neighbours 1](#_Toc6398)

[Objetivos 1](#_Toc6399)

[1) Parte 1: Partición del dataset en conjuntos de entrenamiento y de test 1](#_Toc6400)

[2) Parte 2: k-Vecinos más cercanos 2](#_Toc6401)

[3) Parte 3: Evaluación del rendimiento del clasificador 4](#_Toc6402)

## Objetivos

En esta práctica vamos a trabajar con el método de clasificación de los k vecinos más cercanos (k-Nearest Neighbours, kNN). Además, vamos a simular un experimento de clasificación y calcular algunas métricas de error del mismo.

El script general para esta práctica será main\_lab7.m, y en él se irán llamando a funciones que realizarán los ejercicios planteados en la práctica.

Para esta práctica emplearemos un *dataset* con datos de correos electrónicos para determinar si son spam o no (descrito en [https://archive.ics.uci.edu/ml/datasets/Spambase)](https://archive.ics.uci.edu/ml/datasets/Spambase). Cada elemento del *dataset* (variable de Matlab X) se define mediante 57 variables. Las clases se indican en la variable de Matlab Y. Estas variables de Matlab se obtienen tras cargar el archivo spambase\_data.mat.

Como convención, en esta práctica cada patrón (el vector de características que define cada correo electrónico) está contenido en cada fila de la matriz de datos

## 1) Parte 1: Partición del dataset en conjuntos de entrenamiento y de test

Cuando se está evaluando un clasificador en aprendizaje supervisado, es necesario utilizar un conjunto de entrenamiento para entrenar el modelo de clasificación y un conjunto de test, que se clasificará con el modelo generado para evaluar su rendimiento.

La diferencia es que las clases son **conocidas en el caso del conjunto de entrenamiento**

(notadas en las ecuaciones de la parte de teoría mediante *y*( )*i* ), mientras que **no las utilizaremos en la clasificación de los elementos** (solo en la comparación entre los resultados de la clasificación y la clase real).

En la primera parte del script main\_lab7.m se carga el dataset y se realiza la partición del mismo en conjuntos de entrenamiento y test.

**Se pide que expliques, mediante comentarios en el código, cada una de las líneas de código siguientes:**

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

num\_patrones\_train = round(p\_train\*num\_patrones);

ind\_permuta = randperm(num\_patrones);

inds\_train = ind\_permuta(1:num\_patrones\_train); inds\_test = ind\_permuta(num\_patrones\_train+1:end);

X\_train = X(inds\_train, :);

Y\_train = Y(inds\_train);

X\_test= X(inds\_test, :);

Y\_test = Y(inds\_test);

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

%% PARTE 1: CARGA CONJUNTO DE DATOS Y PARTICIÓN TRAIN-TEST

load spambase\_data.mat;

% X contiene los patrones de entrenamiento (cada columna corresponde a un patrón)

% Y contiene la clase de cada patrón

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

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

% Parámetro que indica el porcentaje de patrones que se utilizarán en

% el conjunto de entrenamiento

p\_train = 0.7;

% En la siguiente sección de código se realiza la partición de los datos en

% entrenamiento y test. Indica qué realiza cada línea de código mediante

% comentarios.

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

num\_patrones\_train = round(p\_train\*num\_patrones);

ind\_permuta = randperm(num\_patrones);

inds\_train = ind\_permuta(1:num\_patrones\_train);

inds\_test = ind\_permuta(num\_patrones\_train+1:end);

X\_train = X(inds\_train, :);

Y\_train = Y(inds\_train);

X\_test= X(inds\_test, :);

Y\_test = Y(inds\_test);

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

## 2) Parte 2: k-Vecinos más cercanos

El clasificador de los k vecinos más cercanos (*k Nearest Neighbours*, kNN) asigna a cada elemento del conjunto de test (es decir, el elemento cuya clase queremos conocer) a la clase más repetida de entre los *k* elementos más cercanos del conjunto de entrenamiento en el espacio de características (ver Figura en página siguiente).

La búsqueda de los elementos más cercanos se realiza mediante una métrica de distancia (puede ser la Euclídea, Manhattan, Chebychev, distancia del coseno, etc., como se vio en la teoría). En esta práctica utilizaremos la distancia Euclídea.

*n*

*d*(**x y**, ) = (*x yi* − *i* )2 ,

*i*=1

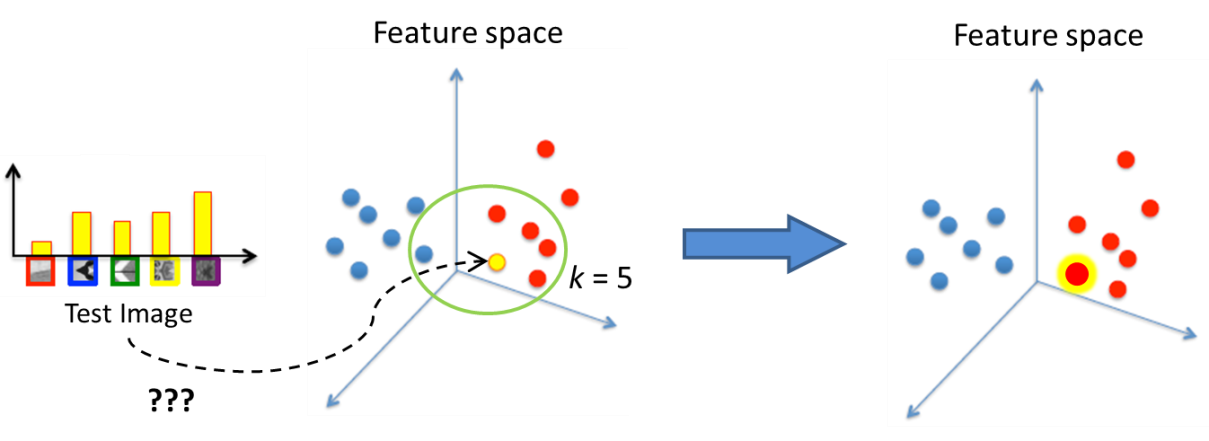
donde *n* es el número de características que definen cada elemento.

Como ejemplo, vamos a suponer un problema de clasificación con 2 clases, un conjunto de entrenamiento de 6 elementos **x x x x x x**1, 2, 3, 4, 5, 6, cuyas clases son *c*(**x**1) =*c*(**x**2) =*c*(**x**3) = 0 y *c*(**x**4) =*c*(**x**5) =*c*(**x**6) =1, y el parámetro *k*=3. Dado un ejemplo **x***t* cuyas distancias a los elementos del conjunto de entrenamiento son:

*d*(**x x***t* , 1) = 0,2 *d*(**x x***t* , 2) = 0,3 *d*(**x x***t* , 2) = 0,05 *d*(**x x***t* , 4) = 0,1

*d*(**x x***t* , 5) = 0,15 *d*(**x x***t* , 6) = 0,4

Los *k*=3 elementos más cercanos son **x**3, **x**4 y **x**5 , cuyas clases son, respectivamente, *c*(**x**3) = 0 y *c*(**x**4) =*c*(**x**5) =1. Por lo tanto, la clase que el clasificador asignará a **x***t* es *c*(**x***t* ) =1. Se muestra a continuación un ejemplo gráfico (para *k*=5):



**Tendrás que completar la función fClassify\_kNN.m, en las partes indicadas con**

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

%% PARTE 2: ALGORITMO DE LOS K VECINOS MÁS CERCANOS

% La función fClassify\_kNN ejecuta el algoritmo de kNN. Ábrela y completa

% el código

k=3;

Y\_test\_asig = fClassify\_kNN(X\_train, Y\_train, X\_test, k);

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

Específicamente para cada elemento del conjunto de test tendrás que:

1. Calcular la distancia Euclídea entre dicho elemento y todos los elementos del conjunto de entrenamiento. **Función útil: pdist2**.
2. Ordenar las distancias en orden ascendente, así como sus clases. **Función útil: sort**.
3. Obtener las clases correspondientes a las *k* primeras distancias.
4. Asignar el elemento de test a la clase más repetida dentro de las *k* primeras.

Esta función recibe los siguientes parámetros:

* **X\_train:** Matriz de tamaño **n\_Train** x f que contiene los datos del conjunto de entrenamiento. n es el número de elementos en el conjunto de entrenamiento y f es el número de características.
* **Y\_train:** Vector **de longitud n\_Train que contiene el número de clase (en este caso, 0 o 1) de cada elemento** (es decir, cada fila) en X\_train.
* **X\_test:** Matriz de tamaño **mx**f con los datos del conjunto de test. **m** es el número de elementos en el conjunto de test y f es el número de características.
* **k**: Número de vecinos que considerará el algoritmo.

La función devolverá un vector de longitud m que contenta el número de clase asignada por el clasificador de cada elemento de X\_test.

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

function Y\_assign = fClassify\_kNN(X\_train, Y\_train, X\_test, k)

% This function implements the kNN classification algorithm with the

% eucludean distance

%

% INPUT

% - X\_train: Matrix (n\_Train x m), where n\_Train is the number of

% training elements and m is the number of features (the length of the

% feature vector)

% - Y\_train: The classess of the elements in the training set. It is a

% vector of length n\_Train with the number of the class.

% - X\_test: matrix (n\_t x m), where n\_t is the number of elements in the test

% set.

% - k: Number of nearest neighbours to consider in order to make an

% assignation

%

% OUTPUT

% A vector with length n\_t, with the classess assigned by the algorithm

% to the elements in the training set.

%

numElemTest = size(X\_test, 1);

numElemTrain = size(X\_train, 1);

% Allocate space for the output

Y\_assign = zeros(1, numElemTest);

% for each element in the test set...

for i=1:numElemTest

element = X\_test(i,:);

% 1 - Compute the Euclidean distance of the i-th test element to all the

% training elements

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

distances = pdist2(element,X\_train,'euclidean');

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

% 2 - Order distances in ascending order, and also the elements in

% Y\_train, to know the classess in the k-nearest-neighbours

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

[ordered\_distances,inds] = sort(distances);

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

% 3 - Take the k first classes of the training set

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

classes = Y\_train(inds(1,1:k));

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

% 4 - Assign to the i-th element the most frequent class

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

n\_0 = sum(find(classes == 0));

n\_1 = sum(find(classes == 1));

if n\_0 > n\_1

Y\_assign(1,i) = 0;

else

Y\_assign(1,i) = 1;

end

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

end

end

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

## 3) Parte 3: Evaluación del rendimiento del clasificador

Una vez que el clasificador ha asignado las clases a cada uno de los elementos del conjunto de test es necesario compararlas con las clases reales de dichos elementos, con el fin de **evaluar lo bueno que es el clasificador** (o los datos con los que se describen los objetos que queremos clasificar).

Los resultados de una clasificación se pueden resumir mediante una **matriz de confusión**. En ella, cada columna corresponde a las predicciones de las clases y cada fila corresponde a las clases reales. Por ejemplo, en la siguiente matriz:

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

TP, TN, FP, FN corresponden respectivamente, al número de **verdaderos positivos**, **verdaderos negativos**, **falsos positivos** y **falsos negativos**. Así, por ejemplo, si FN=10, significa que el clasificador ha clasificado como negativos (clase 0) 10 elementos que realmente eran positivos (pertenecían a la clase 1) Algunas métricas de rendimiento son:

* **Error global:** Describe la tasa de elementos mal clasificados:

*error* = *FP*+ *FN*

*TP TN*+ + +*FP FN*

* **Tasa de falsos positivos (falsa aceptación):** Es la tasa de elementos que se clasifican como positivos (de la clase 1) pero en realidad son negativos (clase 0) de entre los elementos que realmente son negativos.

*FP FP*

*FPR* = =

*N FP TN*+

* **Tasa de falsos negativos (falso rechazo):** Es la tasa de elementos que se clasifican como negativos (de la clase 0) pero en realidad son positivos (clase 1) de entre los elementos que realmente son positivos.

*FN FN*

*FNR* = =

*P TP*+ *FN*

* **Precisión:** Es la fracción de verdaderos positivos entre el total de elementos que el clasificador ha predicho como positivos

*TP TP*

*precision* = =

*pred* \_ *positive TP* + *FP*

* **Recall:** Fracción de los verdaderos positivos entre los elementos que son realmente positivos

*TP TP*

*recall* = = *P TP*+ *FN*

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

%% PARTE 3: EVALUACIÓN DEL RENDMIENTO DEL CLASIFICADOR

% Muestra matriz de confusión

plotconfusion(Y\_test, Y\_test\_asig);

oneOne = zeros(1,length(Y\_test));

zeroOne = zeros(1,length(Y\_test));

oneZero = zeros(1,length(Y\_test));

zeroZero = zeros(1,length(Y\_test));

for i=1:length(Y\_test)

oneOne(1,i) = Y\_test(1,i) & Y\_test\_asig(1,i);

zeroOne(1,i) = ~Y\_test(1,i) & Y\_test\_asig(1,i);

oneZero(1,i) = Y\_test(1,i) & ~Y\_test\_asig(1,i);

zeroZero(1,i) = ~(Y\_test(1,i) | Y\_test\_asig(1,i));

end

TP = length(find(oneOne));

FN = length(find(oneZero));

FP = length(find(zeroOne));

TN = length(find(zeroZero));

% Error--> Error global

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

error = (FP + FN) / (TP+TN+FP+FN);

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

fprintf('\n\*\*\*\*\*\*\nError global = %1.4f%% (classification)\n', error\*100);

% Tasa de falsa aceptación

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

FPR = FP / (FP+TN);

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

fprintf('\n\*\*\*\*\*\*\nTasa de falsa aceptación = %1.4f%% (classification)\n', FPR\*100);

% Tasa de falso rechazo

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

FNR = FN / (TP + FN);

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

fprintf('\n\*\*\*\*\*\*\nTasa de falso rechazo = %1.4f%% (classification)\n', FNR\*100);

% Precision

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

precision = TP / (TP+FP);

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

fprintf('\n\*\*\*\*\*\*\nPrecision = %1.4f%% (classification)\n', precision\*100);

% Recall

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

recall = sum(Y\_test\_asig==1 & Y\_test==1)/sum(Y\_test==1);

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

fprintf('\n\*\*\*\*\*\*\nRecall = %1.4f%% (classification)\n', recall\*100);

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