# Lab 10. Evaluación de modelos mediante curva ROC y reducción de dimensionalidad con PCA

[Lab 10. Evaluación de modelos mediante curva ROC y reducción de dimensionalidad con PCA 1](#_Toc5117)

[Objetivos 1](#_Toc5118)

[1) Parte 1: Evaluación de modelos con curvas ROC 1](#_Toc5119)

[2) Parte 2: Reducción de dimensionalidad mediante Análisis de Componentes Principales (PCA) 4](#_Toc5120)

## Objetivos

Esta práctica se divide en dos partes diferenciadas:

En **primer lugar**, se evaluará el rendimiento de un clasificador (en este caso, usaremos el clasificador de regresión logística) mediante su **curva ROC**. En esta parte de la práctica emplearemos el mismo *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). En este caso, los datos **NO ESTÁN NORMALIZADOS**. Se espera que **el alumnado compruebe las diferencias en los resultados en el experimento de clasificación con y sin normalización**. Esta parte se encuentra en la carpeta 1\_Lab10\_ROC\_curve, y el script principal para la misma será main\_ROCCurve.m.

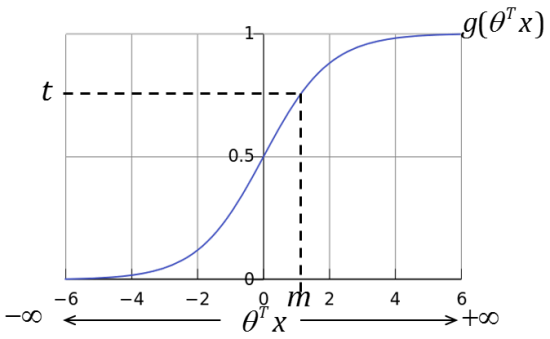
En segundo lugar, se programará el algoritmo del Análisis de Componentes Principales (PCA) y se probará con el conjunto de datos [Iris Flower dataset,](https://en.wikipedia.org/wiki/Iris_flower_data_set) que caracteriza tres tipos de flores (Iris setosa, Iris virginica e Iris versicolor) mediante cuatro variables: longitud y anchura de sépalos y de pétalos. Esta parte se encuentra en la carpeta 2\_Lab10\_PCA, y el script principal se llama main\_PCA.m.

En ambos casos, los datos están contenidos en la variable de Matlab X y sus clases se indican en la variable de Matlab Y. Como siempre, 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).

## 1) Parte 1: Evaluación de modelos con curvas ROC

Existen clasificadores cuya salida no es la clase a la que pertenece el vector de características del patrón de entrada, sino la probabilidad que dicho patrón pertenezca a la clase positiva. El clasificador de regresión logística es uno de ellos.

En estos casos, para asignar una clase a un vector de características es necesario determinar un valor denominado **umbral de decisión**, tal que **cuando la salida del clasificador es mayor que dicho umbral, la clase asignada será la positiva**. Normalmente este umbral vale 0.5.



*Figura 1. Función sigmoidal*

Esto permite obtener métricas de rendimiento de la clasificación como las que vimos en las prácticas anteriores, basadas en la **matriz de confusión** del clasificador.

Como ya vimos en teoría, el *accuracy* no siempre es una buena medida de rendimiento del clasificador, sobre todo cuando el conjunto de datos de test está muy desequilibrado (es decir, hay muchos más elementos de una clase que de otra).

Una forma más independiente y completa para evaluar el rendimiento de clasificadores es mediante el **análisis ROC** (*Receiver Operating Characteristic*). Dada una matriz de confusión:

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

Recordemos que:

*TN*

* **Tasa de verdaderos negativos**: *TNR*= *FP TN*+

*FP*

* **Tasa de falsos positivos**: *FPR*= *FP TN*+

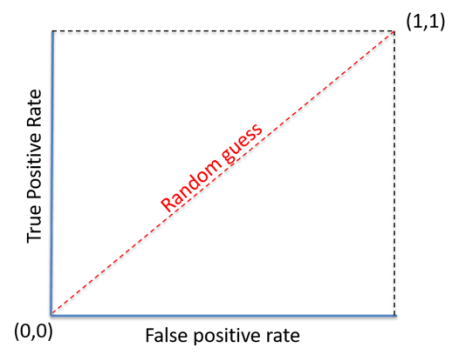
*FN*

* **Tasa de falsos negativos**: *FNR*= *TP FN*+

*TP*

* **Tasa de verdaderos positivos**: *TPR*= *TP FN*+

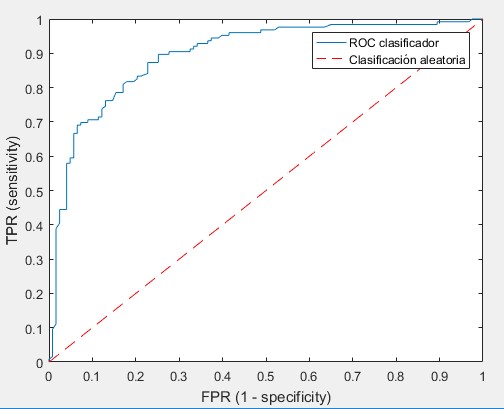
Con esto, se puede crear una gráfica que representa el rendimiento de un clasificador mediante un punto en el espacio ROC (ver Figura 2), en el que los puntos (0,0) y (1,1) corresponden respectivamente al clasificador que predice todo como clase negativa y clase positiva, respectivamente.



*Figura 2. Espacio ROC*

Por lo tanto, desde el punto de vista del aprendizaje, se podrían descartar clasificadores que no estén dentro de la superficie formada por los puntos (0,0), (1,1) y (0,1).

En el caso de nuestro clasificador, es evidente que, **para umbrales de decisión diferentes**, las clases asignadas a los patrones pueden variar y, por lo tanto, **la matriz de confusión también puede ser diferente**. Por lo tanto, variando el umbral de decisión tendremos diferentes valores de FPR y TPR (**diferentes puntos en el espacio ROC**), pudiendo crear una curva que evalúe lo bueno que es un clasificador particular calculando el **área bajo la curva ROC** (Figura 3).



*Figura 3. Curva ROC*

En el código que se adjunta con esta práctica, la curva ROC se calcula en la función fGetROC cuyos parámetros de entrada son:

* **Y\_test**: Vector que contiene las clases reales (0 o 1) de los elementos del conjunto de test.
* **Y\_test\_hat**: Vector que contiene las salidas del clasificador. Estas salidas corresponden con la probabilidad de los patrones clasificados de pertenecer a la clase positiva.
* **umbrales\_decision**: Vector con los diferentes umbrales de decisión que se utilizarán para generar los valores de FPR y TPR.

Y devolverá dos vectores correspondientes a la FPR y a la TPR obtenidas con cada uno de los umbrales de decisión de entrada.

Se pide que completes esta función para que calcule la FPR y TPR de cada umbral de decisión.

Finalmente, en el script principal se calculará el **área bajo la curva ROC**, que sirve como

métrica que “resume” la curva, permitiendo comparar las curvas obtenidas por diferentes clasificadores.

|  |  |
| --- | --- |
| **TAREA: Además, compara los resultados de la clasificación con los datos originales (sin normalizar) y los datos normalizados (usar la normalización** | |
| **por tipificación). ¿Qué sucede? ¿Sabrías indicar por qué?** |  |

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

%% PRELIMINAR: CARGA CONJUNTO DE DATOS Y PARTICIÓN TRAIN-TEST

load mammographic\_data.mat;

% X contiene los patrones de entrenamiento (dimensión 5)

% Y contiene la clase del patrón

% NORMALIZACIÓN DE LOS DATOS:

% La clasificación se probará SIN y CON normalización

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

medias = mean(X);

desviacion = std(X);

[f,c] = size(X);

for i=1:f

for j=1:c

X(i,j) = (X(i,j) - medias(j))/desviacion(j);

end

end

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

% 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;

% Partición de los datos en conjuntos de entrenamiento y test.

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

%num\_patrones\_test = num\_patrones - num\_patrones\_train;

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: ENTRENAMIENTO DEL CLASIFICADOR Y CLASIFICACIÓN DEL CONJUNTO DE TEST

% La función fClassify\_LogisticReg implementa el clasificador de la regresión

% logística. Ábrela y completa el código

alpha = 2;

% ENTRENAMIENTO:

theta = fEntrena\_LogisticReg(X\_train, Y\_train, alpha);

% CLASIFICACIÓN DEL CONJUNTO DE TEST

Y\_test\_hat = fClasifica\_LogisticReg(X\_test, theta);

Y\_test\_asig = Y\_test\_hat>=0.5;

% Muestra matriz de confusión

figure;

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

%% PARTE 2: CURVA ROC

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

% Llamada a fGetROC. Los umbrales de decisión que se usen tienen que estar

% entre 0 y 1, PERO EN ORDEN DECRECIENTE para que la curva quede bien

% representada

umbrales = linspace(1,0,101);

[FPR,TPR] = fGetROC(Y\_test, Y\_test\_hat, umbrales);

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

figure; plot([0,FPR], [0,TPR]);

hold on;

line(0:0.01:1, 0:0.01:1, 'Color', 'red', 'LineStyle', '--');

legend('ROC clasificador', 'Clasificación aleatoria');

xlabel('FPR (1 - specificity)');

ylabel('TPR (sensitivity)');

hold off;

% Calcula el área bajo dicha curva

AUC = trapz(FPR, TPR);

fprintf('\n\*\*\*\*\*\*\nArea bajo la curva = %1.4f \n', AUC);

% Para comprobar resultado: La curva debe ser similar

%[FPR\_2,TPR\_2,T,AUC] = perfcurve(Y\_test, Y\_test\_hat, 1);

%figure; plot(FPR\_2,TPR\_2);

%hold on;

%line(0:0.01:1, 0:0.01:1, 'Color', 'red', 'LineStyle', '--');

%legend('ROC clasificador', 'Clasificación aleatoria');

%xlabel('FPR (1 - specificity)');

%ylabel('TPR (sensitivity)');

%hold off;

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

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

**FUNCION FGETROC**

function [vFPR, vTPR] = fGetROC(Y\_test, Y\_test\_hat, umbrales\_decision)

% Esta función obtiene los valores de Tasa de Falsa Aceptación (FPR) y Tasa

% de Verdadera Aceptación (TPR) que definen la curva ROC, para ciertos

% umbrales de decisión. Además, calcula el área bajo dicha curva. Esta

% función está diseñada para problemas de clasificación binaria (.

%

% INPUT

% - Y\_test: Vector que contiene las clases reales (0 o 1) de los

% elementos del conjunto de test.

% - Y\_test\_hat: Vector que contiene las salidas del clasificador. Estas

% salidas corresponden con la probabilidad de los patrones clasificados

% de pertenecer a la clase positiva.

% - umbrales\_decision: Vector con los diferentes umbrales de decisión que

% se utilizarán para generar los valores de FPR y TPR.

%

% OUTPUT

% - vFPR: Vector con el mismo número de elementos que umbrales\_decision

% que contiene los valores de FPR para cada umbral de decisión utilizado.

% - vTPR: Vector con el mismo número de elementos que umbrales\_decision

% que contiene los valores de TPR para cada umbral de decisión utilizado.

% - AUC: Valor del área que se encuentra bajo la curva ROC.

%

vFPR = zeros(1, length(umbrales\_decision));

vTPR = zeros(1, length(umbrales\_decision));

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

for i=1:length(umbrales\_decision)

umbral = umbrales\_decision(i);

predicted\_label = Y\_test\_hat >= umbral;

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 j=1:length(Y\_test)

oneOne(1,j) = Y\_test(1,j) & predicted\_label(1,j);

zeroOne(1,j) = ~Y\_test(1,j) & predicted\_label(1,j);

oneZero(1,j) = Y\_test(1,j) & ~predicted\_label(1,j);

zeroZero(1,j) = ~(Y\_test(1,j) | predicted\_label(1,j));

end

TP = length(find(oneOne));

FN = length(find(oneZero));

FP = length(find(zeroOne));

TN = length(find(zeroZero));

vTPR(i) = TP/(TP + FN);

vFPR(i) = FP/(FP + TN);

end

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

end

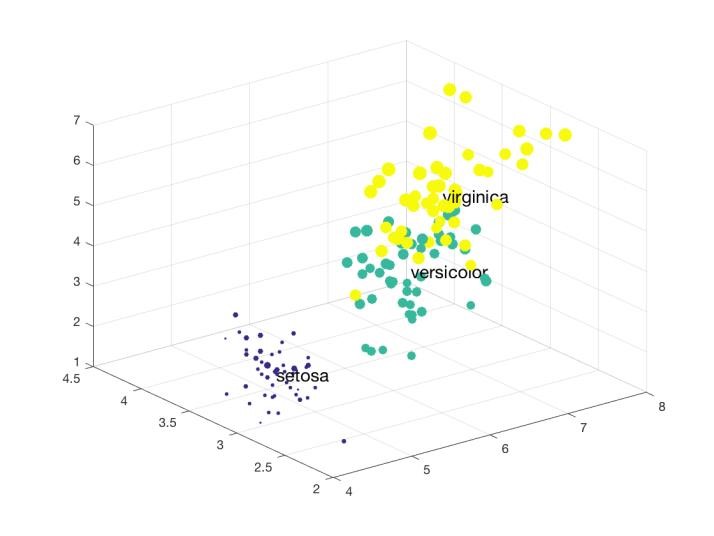
**% ============================================================**

## 2) Parte 2: Reducción de dimensionalidad mediante Análisis de Componentes Principales (PCA)

El código adjunto de la práctica main\_PCA.m realiza la lectura de un fichero de datos que contiene el set de datos de las flores iris que es muy utilizado en comprobación de algoritmos de clasificación.

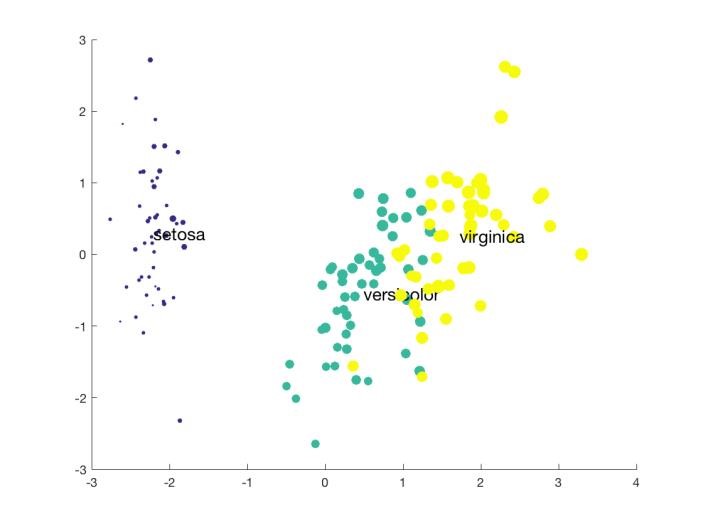
Este código tiene que realizar una reducción de la dimensionalidad por medio de un análisis de componentes principales (PCA) para eliminar variables que puedan introducir redundancia en los datos. Este código está estructurado en tres partes.

1. – Lectura del fichero de datos.
2. – Plot tridimensional de las variables en las que cada coordenada se corresponde con una variable, el tamaño de los puntos con la cuarta variable y el color nos indica la clase a la que pertenece cada muestra. En la figura siguiente se ve dicha gráfica.



*Figura 4. Plot tridimensional de los datos Iris*

1. – Análisis de componentes principales de los datos para reducirlos a una dimensionalidad *k*.
2. – Plot de los resultados de la PCA sobre un mapa de dos dimensiones, siempre y cuando *k=2*.



*Figura 5. Plot en dos dimensiones de los datos Iris*

En el código de la práctica falta por programar el algoritmo del PCA, por lo que habrá que completarlo con las instrucciones necesarias. En el código se indica con comentarios los pasos del PCA y dónde hay que escribir las instrucciones para llevarlos a cabo, de acuerdo al algoritmo visto en clase. Estos pasos, y algunas funciones útiles para llevarlos a cabo son:

1. – Normalización de los datos.
2. – Cálculo de la matriz de covarianza (**función útil:** **cov**).
3. – Obtención de los autovalores y autovectores de la matriz de covarianza (**función útil: eig** o **svd**).
4. – Ordenación de los autovectores en función del valor sus autovalores asociados de mayor a menor.
5. – Selección de los autovectores que tienen los k autovalores mayores. En caso de que k=0, hay que seleccionar un valor de k tal que permita explicar el 95% de la variabilidad de los datos.
6. – Obtención de los nuevos datos.

**El resultado tiene que ser similar al de la Figura 5**, aunque puede ocurrir que los ejes estén invertidos, en función de cómo se obtengan los autovectores.

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Visualizacion de datos por medio de una pca %%

% Programación de un script que realiza una pca sobre unos datos de entrada

% Este script realiza la lectura de un fichero de datos y realiza una pca,

% la seleccion del numero de variables se realizará o bien de forma

% automática o se insertará por código.

%% Parámetros

% Selección del número de componentes principales.

k = 2;

% Carga de los datos

load('datos\_iris.mat');

%% Ploteo de los datos originales

% Se crea un scatter de los datos en 3D en el cual la posicion es

% indiciativa de las tres primeras propiedades de la flor, el tamaño del

% punto de la cuarta propiedad y el color de la clase.

figure()

for i=1:length(tiposIris)

% Se aisla cada grupo de datos

datos\_in = X(Y==i, :);

% Se añade al scatter cada grupo de datos

scatter3(datos\_in(:,1), datos\_in(:,2), datos\_in(:,3), (datos\_in(:,4)\*50), Y(Y==i), 'filled');

% Se marca con un texto el tipo de planta de cada grupo

text(mean(datos\_in(:,1)), mean(datos\_in(:,2)), mean(datos\_in(:,3)), tiposIris(i), 'FontSize', 14);

hold on

end

%% Programa de la PCA

% En ese apartado se programará la PCA de acuerdo al algortimo visto en

% clase.

% 1. Normalización de los datos. La normalización consistirá en la

% tipificación: restar a cada variable la media de esa variable en todos

% los vectores de características (en este caso: la media de toda la columa

% correspondiente) y dividirla entre la desviación típica de esa variable

% en todos los vectores de características

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

medias = mean(X);

desviacion = std(X);

[f,c] = size(X);

X\_noNorm = X;

for i=1:f

for j=1:c

X(i,j) = (X(i,j) - medias(j))/desviacion(j);

end

end

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

% 2. Cálculo de la matriz de covarianza

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

S = cov(X);

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

% 3. Obtención de los autovalores y autovectoresvectores de la matriz de

% covarianza de los datos.

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

[autovectores, autovalores\_diag] = eig(S);

autovalores = diag(autovalores\_diag);

autovalores = autovalores';

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

% 4. Ordenación de los autovectores en función de sus valores singulares

% asociados de mayor a menor.

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

[autovalores\_ordenados, pos] = sort(autovalores, 'descend');

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

% 5. Seleccionar los k primeros autovectores

% 5.a. En el casode que k sea 0 se selecciona un valor de k tal que permita

% explicar el 95% de la variabilidad de los datos

if(k == 0)

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

total = sum(autovalores);

umbral = 0.95\* total;

i = 0;

sumParcial = 0;

while sumParcial < umbral

part = autovalores\_ordenados(i);

sumParcial = sumParcial + part;

i = i + 1;

end

k = i;

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

end

% 5.b. Seleccionar los k primeros autovectores (después de ordenarlos en el

% paso 4)

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

posVectoresSeleccionados = pos(1:k);

vectoresSeleccionados = autovectores(:,posVectoresSeleccionados);

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

% 6. Obtenemos los nuevos datos

% Es lo mismo que aplicar la matriz de giro a los datos para obtener los

% valores con las nuevas corrdenadas.

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

[f,c] = size(X);

DatosPCASol = zeros(f,k);

for i=1:f

DatosPCASol(i,:) = X(i,:) \* vectoresSeleccionados;

end

DatosPCASol2 = X\*vectoresSeleccionados;

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

%% Ploteo del resultado de la PCA cuando k = 2

% Se va a crear un subplot para cada variable, la posicion de los puntos

% se determina por medio de las coordenadas PCA, el radio del punto va

% a depender en cada caso de una de las variables originales.

if(k==2)

figure();

% Recorremos los subplots para plotear en cada uno una cosa

for i=1:length(tiposIris)

index = Y==i;

% Ploteamos los datos como puntos

scatter(DatosPCASol(index,1), DatosPCASol(index,2), X\_noNorm(index,4)\*50, Y(Y==i),'filled');

% Añadimos el texto correspondiente a cada grupo

text(mean(DatosPCASol(index,1)), mean(DatosPCASol(index,2)), tiposIris(i), 'FontSize', 14);

hold on

end

end

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

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

   [↑](#footnote-ref-1)