Universidad de Antioquia, Departamento de Ingeniería Electrónica y de Telecomunicaciones

Presentación de las diferentes etapas de un sistema de análisis de patrones.

Tomás Arias Vergara

9 de septiembre de 2016

1. Etapas de un sistema de análisis de patrones.

La Figura 1 muestra una representación básica de las etapas que componen un sistema de análisis de patrones general. La primera etapa consiste en la adquisición de los datos. Luego estos datos son acondicionados en la etapa de preprocesado para ser analizados adecuadamente. El siguiente paso es la extracción de características donde se obtienen representaciones de los datos adquiridos. Luego es sistema de analisis es entrenado con un clasificador o regresor, dependiendo del tipo de problema que se quiere abordar con los datos adquiridos. La etapa de toma de decisiones se refiere a como serán asginados los datos nuevos al sistema de análisis ya entrenado. Finalmente, el desempeño del clasificador o regresor es evaluado por medio de distintas técnicas. En las siguientes secciones se explican más detalles de cada etapa con un enfoque hacia el procesmaiento digital de señales de voz.

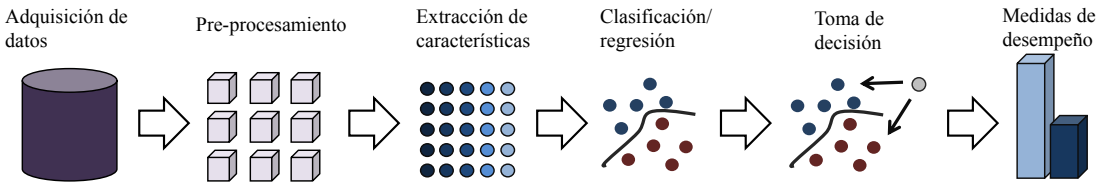

Figura 1: Metodología general en un sistema de análisis de patrones.

2. Preprocesado.

Las muestras de los datos adquiridos en el mundo real por lo general no son adecuados para ser análizados en el su estado original. Por lo general muchos de estos datos presentan ruido e inconsistencias, por lo que los datos capturados deben ser acondicionados para ser tratados adecuadamente por un sistema de análisis automático. En el procesamiento de señales de voz, esta etapa por lo general involucra distintos procedimientos que muchas veces dependen del tipo de análisis que se requiere realizar. Comenzando con la captura de datos, las grabaciones de audio pueden ser adquiridas en ambientes de ruido controlado o no, lo cual requiere de algoritmos de eliminación de ruido si es pertinente para el tipo de problema abordado. También, la distancia a la que se encuentra el micrófono o el volumen de la voz durante las grabaciones varía de persona a persona, por lo que es necesario normalizar las amplitudes en las señales de voz para que todas queden en el mismo rango de valores. Incluso el mismo dispositivo de grabación puede inducir niveles de amplitud a las grabaciones de voz, lo cual afecta el desempeño del sistema de análisis. Adicionalmente, en la etapa de preprocesado también elegimos el modelo acústico utilizado para analizar las señales de voz. A continuación se muestra un ejemplo de los pasos seguidos en la etapa de preprocesamiento para análizar señales de voz de habla continua, es decir, lecturas, monólogos, dialógos, entre otros.

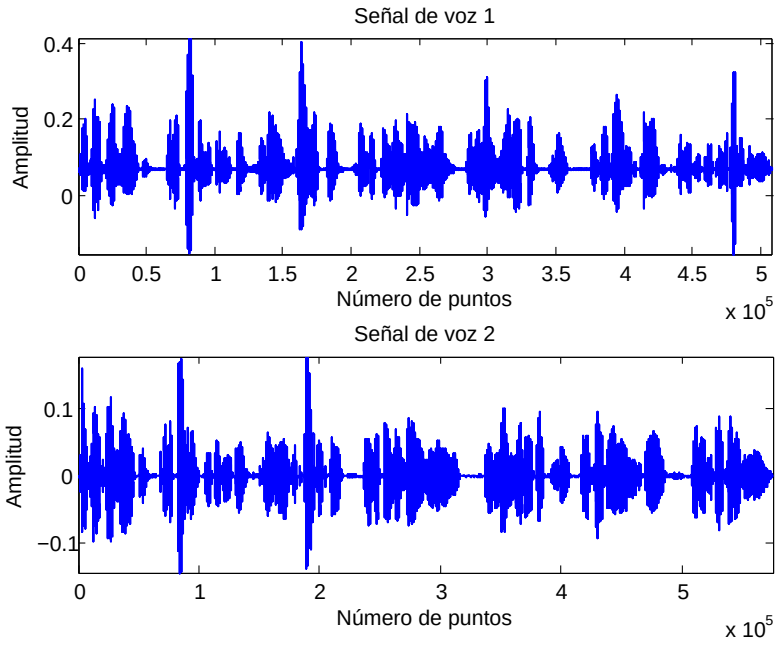

Figura 2: Señales de voz.

En la Figura 2 se muestran las grabaciones de voz de dos personas a las que se les pidió leer un texto. Como se puede observar ambas señales tienen un rango de valores de amplitud diferentes. Adicionalmente, la señal de voz (grafica superior) no está centrada

sobre cero en el eje y, a diferencia de la señal de voz (grafica inferior). Lo cual indica la presencia de un nivel DC en la señal 1.

Antes de comenzar a trabajar con estas señales de voz, el primer paso es eliminar el nivel DC de la señal. Para esto se calcula el valor promedio de la señal de audio y este valor se resta a cada punto de la señal de voz. Luego se normalizan las amplitudes para que sus valores queden en un rango que va desde -1 hasta 1. Para esto se divide cada valor de amplitud de la señal de voz entre la amplitud máxima de esta. Ambos procedimiento se pueden realizar con los siguientes comandos en Matlab.

```
%Cargar audio
[sig,fs] = audioread(['audio.wav']);

%Elminar nivel DC
sig = sig-mean(sig);

%Normalizar para valores entre -1 y 1
sig = sig./max(abs(sig));
```

El resultado de aplicar este procedimiento a las dos señales de audio se muestra en la Figura 3.

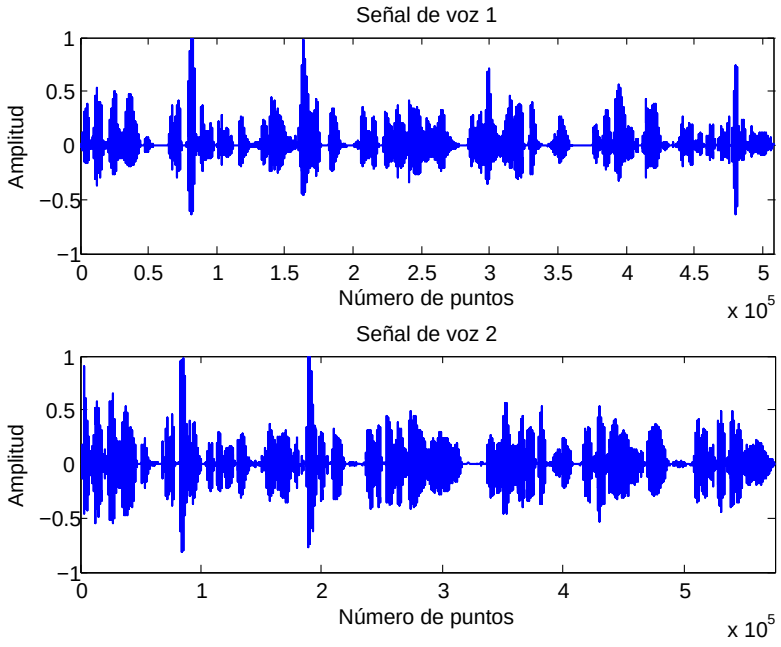

Figura 3: Señales de voz con 0 nivel DC y amplitudes normalizadas entre -1 y 1.

El siguiente paso es el enventanado de la señal. Las señales de voz producidas durante

el proceso de producción del habla, son señales no estacionarias con propiedades que cambian rápidamente en el tiempo. ésta naturaleza hace que analizar las señales de voz, por ejemplo, en el dominio de la frecuencia por medio de Fourier, sea prácticamente imposible. Por esta razón, las señales de voz son analizadas en pequeños segmentos que van de 10 ms a 120 ms, dependiendo del tipo de análisis que se quiere realizar.

3. Caracterización

La etapa de caracterización consiste en la extracción de distintas medidas con las que se busca representar los datos adquiridos. Por ejemplo, en detección de emociones por medio de señales de voz, es posible identificar emociones de alta excitación (enojo, ansiedad, felicidad y disgusto) y baja excitación (aburrimiento, neutral y tristeza), considerando diversas medidas como la energía de la señal, los coeficientes cepstrales en la escala de frecuencia de Mel, o la taza de cruces por cero, entre otras. Dado que no existen medidas estándar para lograr diferenciar entre ambos tipos de emociones, se calculan diferentes características para lograr la discriminación entre los tipos de emociones. Sin embargo, muchas de estas medidas aportan poca información acerca de la naturaleza de las emociones, por lo que dichas medidas se convierten en datos innecesarios para el sistema. Por esto, es muy común aplicar diferentes métodos de selección de características para reducir la cantidad de medidas extraídas e incluir solamente aquellas que aportan información importante al sistema.
Existen varias técnicas para selección de características tales como análisis de componentes principales (PCA, del inglés Principal Component Analysis), análisis discriminante lineal (LDA, del inglés Linear Discriminant Analysis), entre otras.

3.1. Principal Component Analysis-PCA

El objetivo principal del análisis de componentes principales es representar el espacio de características en un subespacio de menor dimensión reduciendo la redundancia y minimizando el ruido, es decir, descartar características que miden los mismo, pero con diferente rango de valores o descartar características que no aportan información. Para esto PCA se enfoca en grandes variaciones del conjunto de datos (matriz de características) para extraer los patrones que más información aportan en el espacio de características. El método PCA puede ser reducido en los siguientes pasos:

Se consideran todos los vectores de características d-dimensionales de cada muestra m (grabaciones de audio, imágenes, entre otros) sin etiquetas de clase (si la señal de voz pertenece a una persona enferma o sana, si la imagen pertenece a un objeto o un animal, etc).

Se calculan los valores promedio para cada uno de los vectores de características de las m muestras.

Se calcula la matriz de covarianza a partir de la matriz de características $d \times m$.

Se calculan los vectores propios (eigenvectores) $\in \{e_1, e_2, ..., e_d\}$ con sus correspondientes valores propios (eigenvalores) $\in \{\lambda_1, \lambda_2, ..., \lambda_d\}$.

Se organizan los eigenvalores de forma descendente y se eligen los k eigenvectores con sus correspondientes eigenvalores más altos para formar una matriz $W_{d\times k}$.

Esta nueva matriz $d \times k$ se utiliza para transformar la muestras en el nuevo subespacio. Para esto se recurre a la expresión matemática $\boldsymbol{y} = \boldsymbol{W}^T \times \boldsymbol{x}$, donde $\boldsymbol{x}$ es un vector $d \times 1$-dimensional que representa una muestra y $\boldsymbol{y}$ es la $k \times 1$ muestra transformada.

Como ejemplo se considera una base de datos de emociones clasificadas en dos grupos: emociones de baja excitación (clase 0) y emociones de alta excitación (clase 1). Para esto se consideran 534 muestras (grabaciones de voz) a las cuales se le calcularon 384 características para formar una matriz de características de dimensión 534×384. El objetivo es verificar si es posible reducir las 384 características a sólo 2. Para esto utilizamos el siguiente código en Matlab:

```
clc
clear all
close all
warning off

addpath(genpath([pwd '\sprtool']))

%Se carga una estructura que contiene la matriz de caracter\'isticas
%de cada clase y las etiquetas de cada una de las muestras.
%Eitquetas 0:emociones baja excitaci\'on; 1:emociones alta excitaci\'on
load('sample_feats.mat')

%Se normalizan las caracter\'isticas (media = 0 y desviaci\'on = 1)
X = zscore(data.X');

%Cuando se realiza reducci\'on de dimensionalidad con PCA, la matriz
%de caracter\'isticas debe tener cada muestra (grabaci\'on) en las columnas
%y cada caracter\'istica en las filas
X = X';

%PCA. Se busca reducir la dimensi\'on de los datos a 2. No se consideran
%etiquetas de clase
model = pca(X,2);

%Se extrae las componentes principales de los datos para obtener una nueva
%matriz de medidas con dos caracter\'isticas
ext_data = linproj(X,model);

%Graficar nuevo espacio de caracter\'istcas
scatter(ext_data(1,data.y==0),ext_data(2,data.y==0),'o')
hold on
scatter(ext_data(1,data.y==1),ext_data(2,data.y==1),'ro')
```

```
title('PCA para base de datos de emociones')
ylabel('PCA 2')
xlabel('PCA 1')
legend('Emociones de baja excitaci\'on','Emociones de alta excitaci\'on')
```

La salida del código anterior se puede observar en la Figura 4.

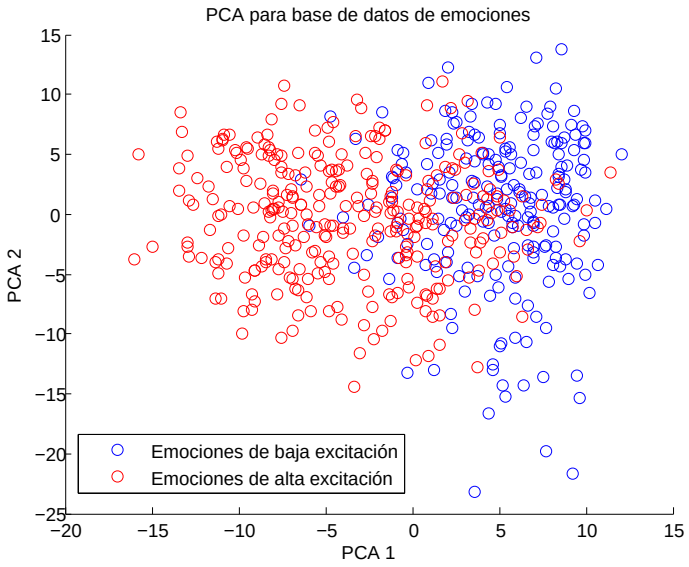

Figura 4: Análisis de componentes principales sobre una base de datos de emociones.

3.2. Linear Discriminant Analysis-LDA

Al igual que la técnica PCA, el método LDA es utilizado comúnmente como un método de reducción de características para ayudar a reducir el costo computacional y además evitar sobre ajustes de modelo al minimizar el error en la optimización de parametros. Con PCA el objetivo principal es hallar las direcciones (componentes principales) que maximizan la varianza del conjunto de datos de entrenamiento (matriz de características). Con el método LDA para cada muestra se consideran las etiquetas de clase y el objetivo principal es encontrar las direcciones (discriminantes lineales) que maximizan la separabilidad entre clases. Para realizar LDA se siguen los siguientes pasos:

Calcular las medias de los vectores d dimensionales para las distintas clases incluidas en el conjunto de datos.

Calcular las matrices de covarianza para cada clase y entre las dos clases.

Se calculan los eigenvectores $\in \{e_1, e_2, ..., e_d\}$ con sus correspondientes valores propios

(eigenvalores) $\in \{\lambda_1, \lambda_2, ..., \lambda_d\}$.

Se organizan los eigenvalores de forma descendente y se eligen los k eigenvectores con sus correspondientes eigenvalores más altos para formar una matriz $W_{d\times k}$.

Esta nueva matriz con dimensiones $d \times k$ se utiliza para transformar las muestras en el nuevo subespacio. Para esto se recurre a la expresión matemática $\boldsymbol{Y} = \boldsymbol{X} \times \boldsymbol{W}$, donde $\boldsymbol{X}$ es una matriz con $m \times k$-dimensional, que representa m muestras y $\boldsymbol{Y}$ son las $n \times k$ muestras transformadas en el nuevo subespacio.

Como ejemplo se utiliza la misma matriz de características considerada para PCA. Para realizar LDA utilizamos el siguiente código en Matlab:

```
clc
clear all
close all
warning off

%Toolbox para realizar LDA (Statistical Patter Recognition Toolbox)
addpath(genpath([pwd '\sprtool']))

%Se carga una estructura que contiene la matriz de caracter\'isticas
%de cada clase y las etiquetas de cada una de las muestras.
%Eitquetas 0:emociones baja excitaci\'on; 1:emociones alta excitaci\'on
load('sample_feats.mat')

%Se normalizan las caracter\'isticas (media = 0 y desviaci\'on = 1)
data.X = zscore(data.X');

%Cuando se realiza reducci\'on de dimensionalidad con LDA, la matriz
%de caracter\'isticas debe tener cada muestra (grabaci\'on) en las columnas
%y cada caracter\'istica en las filas
data.X = data.X';

%LDA. Se busca reducir la dimensi\'on de los datos a 2
model = lda(data,2);

%Se extrae la proyecci\'on lineal de los datos para obtener una nueva
%matriz de medidas con dos caracter\'isticas
ext_data = linproj(data,model);

%Graficar nuevo espacio de caracter\'istcas
scatter(ext_data.X(1,data.y==0),ext_data.X(2,data.y==0),'o')
hold on
scatter(ext_data.X(1,data.y==1),ext_data.X(2,data.y==1),'ro')
title('Proyecci\'on lineal considerando dos discriminadores lineales')
ylabel('LDA 2')
xlabel('LDA 1')
legend('Emociones de baja excitaci\'on','Emociones de alta excitaci\'on')
```

La salida del código anterior se puede observar en la Figura 5.

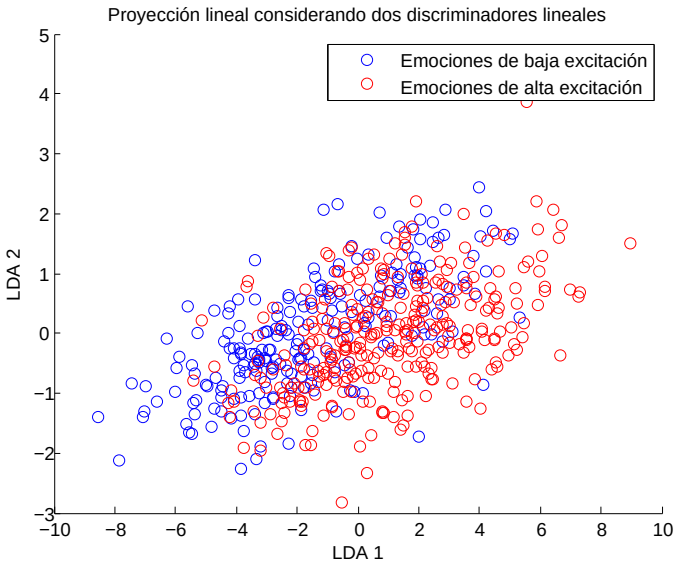

Figura 5: Análisis discriminante lineal sobre una base de datos de emociones.

3.3. PCA vs. LDA

Cuando se aplica PCA, el objetivo principal es buscar las direcciones de máxima variación del conjunto total de datos para reducir la dimensión de la matriz de características. Para esto, se calcula la matriz de covarianza de todo el conjunto de datos sin tomar en cuenta la discriminación entre clases.

Cuando se aplica el método LDA para reducción de características, cada vector de características tiene su respectiva etiqueta de clase. Es decir, en la matriz de característica cada muestra (columna) tiene una etiqueta que indica la clase a la que pertenece. Las etiquetas se utilizan para calcular la matriz de covarianza de cada clase para encontrar las componentes con máxima variación en los datos.

Dado que el método LDA considera la discriminación entre clases para reducción de características, se podría pensar que su desempeño es más alto comparado con PCA, sin embargo, un método puede resultar más útil que otro dependiendo de la aplicación.

4. Entrenamiento de un sistema de reconocimiento de patrones

En esta estapa el sistema de análisis de patrones es entrenado para tomar desiciones cuando nuevos datos son introducidos al sistema o para encontrar una función que represente

el comportamiento de nuestro conjunto de datos.

4.1. Clasificación

Los problemas de clasificación se resumen en entrenar un sistema de reconocimiento de patrones que sea capaz de discriminar entre dos o más clases. De esta manera cuando nuevos datos sean ingresados al sistema, este debe decidir a que clase pertenece el nuevo dato. Existen diversos tipos de clasificadores. Para ilustrar el funcionamiento de estos vamos a realizar clasificación con el método de los k vecinos más cercanos y una máquina de soporte vectorial.

4.1.1. K-vecinos más cercanos-K-NN

El método de clasificación K-NN (del inglés k- Nearest Neighbors) es uno de los algoritmos de clasificación más sencillos de implementar. Dado un sistema de reconocimiento de patrones entrenado con dos (o más) clases, la idea principal de los k-NN es ubicar un nuevo dato de entrada en la clase 1 o clase 2 dependiendo del número de k datos de una clase más cercanos al nuevo dato de entrada, basandose en una regla de desición por mayoría. Para ilustrar esta idea, observemos las Figuras 6 y 7. En la Figura 6 se observa lo que sucede cuando un nuevo dato de entrada (punto negro) es clasficado cuando se consideran $k = 3$ vecinos más cercanos. En este caso, dado que 2 de los 3 datos más cercanos al nuevo dato pertenecen a la clase 1 (puntos azules), este es clasificado como parte de la clase 1. Ahora, cuando $k = 5$ (Figura 7) 3 de los 5 datos más cercanos

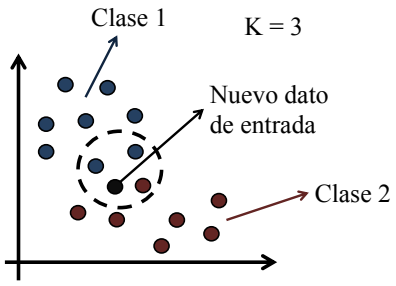

Figura 6: Ejemplo de clasificación K-NN con k = 3.

pertenecen a la clase 2 (puntos rojos), por lo tanto el nuevo dato es incluido en esta clase. De aquí podemos deducir que k es el parámetro de optmización y el rango de valores que puede tomar incluye únicamente números enteros impares. Como ejemplo se consideraran las características obtenidas en la etapa de extracción con PCA para ilustar el funcionamiento del clasificador k-NN. El siguiente código (**KNN.m**) clasifica por medio de K-NN (k=5) las características de dos tipos de emociones: de baja y alta excitación.

```
clc
```

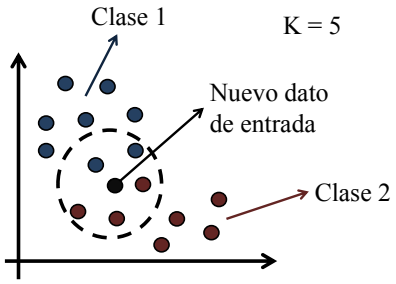

Figura 7: Ejemplo de clasificación K-NN con k = 5.

```
clear all
close all
warning off

addpath(genpath([pwd '\sprtool'])) %PCA
addpath(genpath([pwd '\netlab_gmm']) %knn

%Se carga una estructura que contiene la matriz de caracter\'isticas
%de cada clase y las etiquetas de cada una de las muestras.
%Eitquetas 0:emociones baja excitaci\'on; 1:emociones alta excitaci\'on
load('sample_feats.mat')
%PCA
X = zscore(data.X');
X = X';
model = pcasprt(X,2);
new_feats = linproj(X,model);

% Partir 70% de los datos en el conjunto de entrenamiento y 30% en el
% conjunto de prueba
cv = cvpartition(data.y,'holdout',0.3);
%Seleccionar conjunto de entrenamiento
train = new_feats(:,training(cv));
label_train = data.y(training(cv));
%Seleccionar conjunto de prueba
test = new_feats(:,test(cv));
label_test = data.y(~training(cv));

clear data
%KNN
%Entrenar modelo con k=3
k = 5;
%Los labels deben quedar en codificaci\'on  1-of-N
label_train2 = [label_train' label_train'];
label_train2(:,1) = zeros(size(label_train2,1),1);
label_train2(:,1) = label_train2(:,2)==0;
%Entrenamiento
net = knn(size(train',2),2,k,train',label_train2);
%Prueba
[y,l] = knnfwd(net,test');
```

```
figure(1)
plot(test(1,l==1),test(2,l==1),'bo','MarkerFaceColor','b')
hold on
plot(test(1,l==2),test(2,l==2),'ro','MarkerFaceColor','r')
title('Datos de prueba clasificados por el K-NN')
xlabel('Medida 1')
ylabel('Medida 2')
legend('Emociones de baja excitaci\'on','Emociones de alta excitaci\'on')

figure(2)
plot(test(1,label_test'==0),test(2,label_test'==0),'bo','MarkerFaceColor','b')
hold on
plot(test(1,label_test'==1),test(2,label_test'==1),'ro','MarkerFaceColor','r')
title('Datos de prueba con etiquetas reales')
xlabel('Medida 1')
ylabel('Medida 2')
legend('Emociones de baja excitaci\'on','Emociones de alta excitaci\'on')
```

En la Figura 8 se muestran los resultados de la clasificación (figura superior) y como debieron quedar etiquetados los datos (figura inferior). A simple vista el clasificador parece funcionar bien, sin embargo, para evaluar el desempeño se requiere analizar el porcentaje de acierto, la cantidad de datos mal clasificados, entre otros. Este análisis se describe más adelante.

4.1.2. Clasificación con Máquinas de Soporte Vectorial

Las Máquinas de Soporte Vectorial (SVM, del inglés Support Vector Machine) se utilizan por lo general para resolver problemas de clasificación biclase. Sin embargo, las SVM pueden ser extendidas para resolver problemas de clasificación multi-clase y regresión. En esta sección sólo se abordará la clasificación biclase.

La tarea de clasificación con las SVM se realiza construyendo hiperplanos de separación en espacios multi-dimensionales. Dichos hiperplanos se utilizan como el límite de decisión de pertenencia de un dato a una clase u otra. Para aclarar esta idea, en la Figura 9.A se muestra el caso más sencillo de un clasificador SVM lineal. Como se puede observar, el conjunto de datos pertenecientes a la clase 1 se puede dividir del conjunto de datos de la clase 2 utilizando una línea recta como hiperplano de separación. Luego, cuando ingresa un nuevo dato para ser clasificado, basta con identificar si este está ubicado por encima o por debajo de la línea de separación entre clases para ser asginado a un grupo. Sin embargo, en la mayoría de aplicaciones para resolver problemas de clasificación, el espacio de características no es linealmente separable, por lo tanto no es posible utilizar una SVM lineal (Figura 9.B). Para resolver este problema, las SVM utilizan unas funciones matemáticas especiales llamadas kernel. Básicamente, las funciones kernel permiten mapear (transformar) el espacio de características original en un espacio dimensional mayor y de esta manera encontrar un hiperplano de separación entre las clases. En la Figura 10 se muestra una representación gráfica del funcionamiento del kernel. Para hallar el hiperplano de separación óptimo se considera un criterio adicional

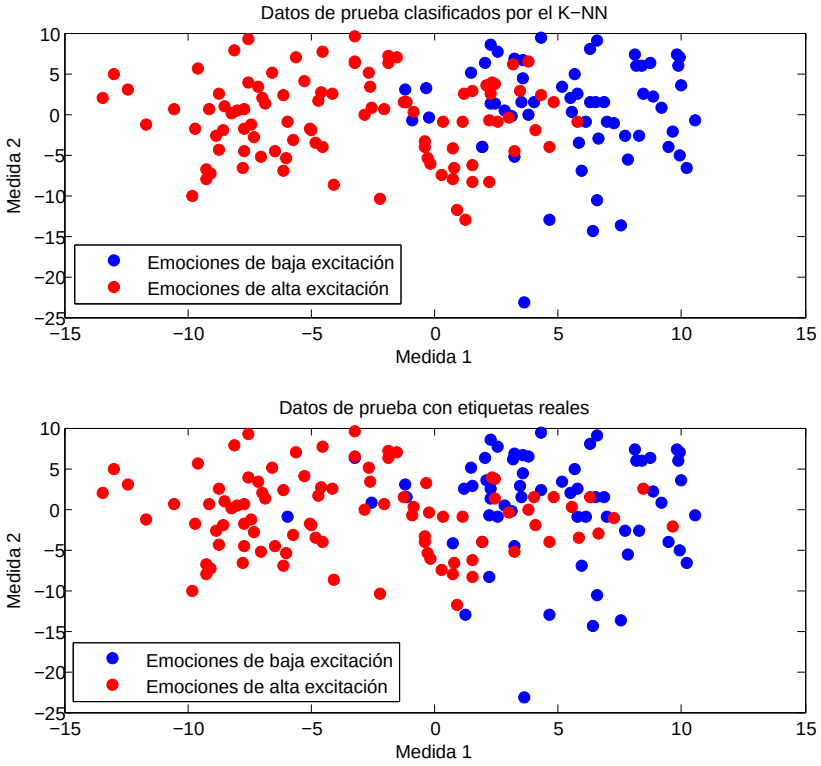

Figura 8: Datos de prueba etiquetados por el K-NN (figura superior) y datos de prueba con etiquetas reales (figura inferior).

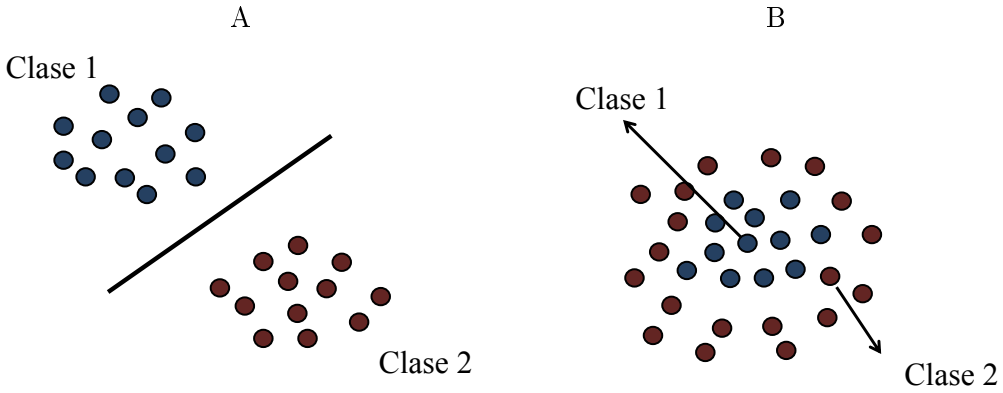

Figura 9: (A) SVM lineal. (B) Espacio de características no linealmente separable con SVM lineal.

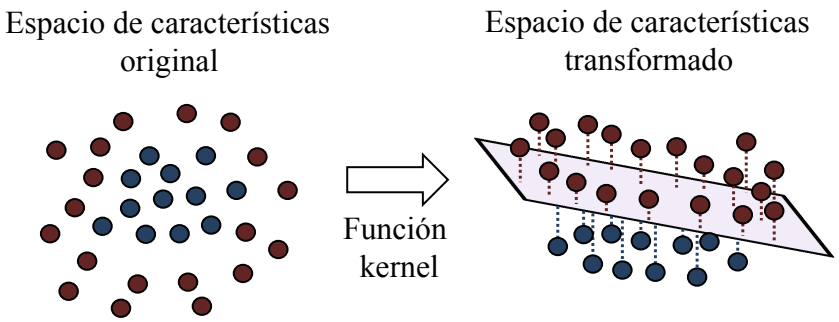

Figura 10: Representación de transformación de un espacio de características utilizando una función kernel.

que consiste en la construcción de un margen que se define como la distancia del hiperplano al punto más cercano a este. Si la SVM separa los datos de entrenamiento en dos clases distintas con un margen lo suficientemente ancho, se puede asegurar que cuando se ingresen nuevos datos para ser clasificados (datos de prueba) estos serán agrupados correctamente en la clase correspondiente. Sin embargo, ubicar todos los datos nuevos en la clase correcta sólo es posible cuando ambos grupos son linealmente separables. Este tipo de margen se conoce como margen duro (hard margin en inglés). En la Figura 11 se muestra una representación gráfica de dicho margen. Como se puede observar, el margen se construye alrededor del hiperplano de separación hasta el primer dato más cercano a este. Además, el margen tiene el mismo ancho en ambos lados del plano de desición. Todos los puntos que se encuentran sobre el margen son llamados vectores de soporte (circulos verdes en la figura). La idea principal es construir el hiperplano de separación óptimo lo más alejado posible de los puntos más cercanos a este. Esto se debe a que los datos pueden ser separados de diferentes maneras por una línea. Como se mencionó

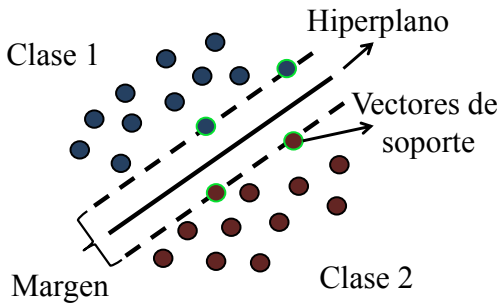

Figura 11: Margen duro para clases linealmente separables.

anteriormente, en la mayoría de los casos los datos de entrenamiento no son linealmente

separables por lo que no es posible encontrar un hiperplano óptimo e incluso si se logra construir uno, este no ofrece la mejor solución para el problema de clasificación que se busca resolver. En este tipo de situaciones es mejor construir hiperplanos que permitan cierta cantidad de error en la clasificación, es decir, se busca construir un hiperplano de separación con el mínimo número de errores de clasificación en el entrenamiento. El margen del hiperplano de separación que cumple con estas condiciones se conoce como margen blando (soft margin en inglés). En la Figura 12 se muestra una representación gráfica de dicho margen. En las SVM de margen blando, los puntos que se encuentran al

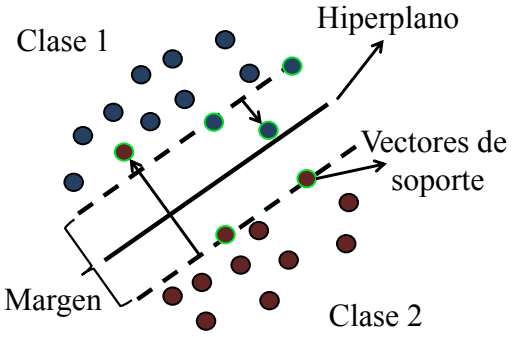

Figura 12: Margen blando para clases no linealmente separables.

otro lado del margen de desición son penalizados dependiendo de la distancia a la que se encuentran del margen. Esto permite generalizar el sistema, pues la SVM será menos sensible al ruido o datos mal etiquetados.

Para tener una idea de como afecta la penalización de los datos al sistema, en las Figura 13 se muestran seis casos en los que se varian los parámetros de ajuste C y γ con los cuales se controla el ancho del margen del hiperplano de separación. El parámetro C controla la compensación entre el tamaño del margen y la penalización de los puntos ubicados al otro lado del margen de decisión, mientras que el parámetro γ es un parámetro de ajuste que depende de la función kernel. Como se puede observar, con valores de C y γ pequeños el número de vectores de soporte es mayor (puntos marcados con círculos negros), lo que ocasiona una perdida en la generalización del sistema ya que el clasificador se adapta más a los datos de entrenamiento y por lo tanto es más sensible al ruido. Los resultados quese muestran en la Figura 13 se pueden obtener con el siguiente código en matlab (**Ejemplo_Margen_SVM.m**):

```
clc
clear
close all
warning off

addpath(genpath([pwd '\sprtool'])) %Toolbox PCA
addpath(genpath([pwd '\svmtools'])) %Toolbox SVM
```

```
%Se carga una estructura que contiene la matriz de caracteristicas
%de cada clase y las etiquetas de cada una de las muestras.
%Eitquetas 0:emociones baja excitacion; 1:emociones alta excitacion
load('sample_feats.mat')

% %PCA
X = zscore(data.X');
X = X';
model = pcasprt(X,2);
new_feats = linproj(X,model);

% % Partir 70% de los datos en el conjunto de entrenamiento y 30% en el
% % conjunto de prueba
cv = cvpartition(data.y,'holdout',0.3);
%Seleccionar conjunto de entrenamiento
train = new_feats(:,training(cv))';
label_train = data.y(training(cv))';
%Seleccionar conjunto de prueba
test = new_feats(:,test(cv))';
label_test = data.y(~training(cv))';

%Parametros de ajuste
vC = [1,100];
vnsigma = [ 1 10 100];
%Datos de entrenamiento
XdataTrainN = dataset(train,label_train);
XdataTestN = dataset(test,label_test);

iplt = 0;
for ivc=1:length(vC)
    for insigma=1:length(vnsigma)
                %Opciones para entrenar SVM con toolbox
        showplot = 'true';
        opts = svmsmoset('Display','off','MaxIter',1000000,...
            'KernelCacheLimit',10000,'TolKKT',1e-4);

        XdataTrainN = +XdataTrainN;
        iplt = iplt+1;
        subplot(1,3,insigma)
                %Entrenar SVM
        svms = svmtrain(XdataTrainN,label_train,'BoxConstraint',vC(ivc),'Kernel_Function','rbf
            'Method','SMO','AutoScale','false','showplot',showplot,'SMO_Opts',opts) ;

                %Titulo de la grafica
        title(['C = ' num2str(vC(ivc)) ' gamma = ' num2str(vnsigma(insigma))])
        legend('Clase 1','Clase 2','Vectores de soporte')
    end
end
```

5. Medidas de rendimiento

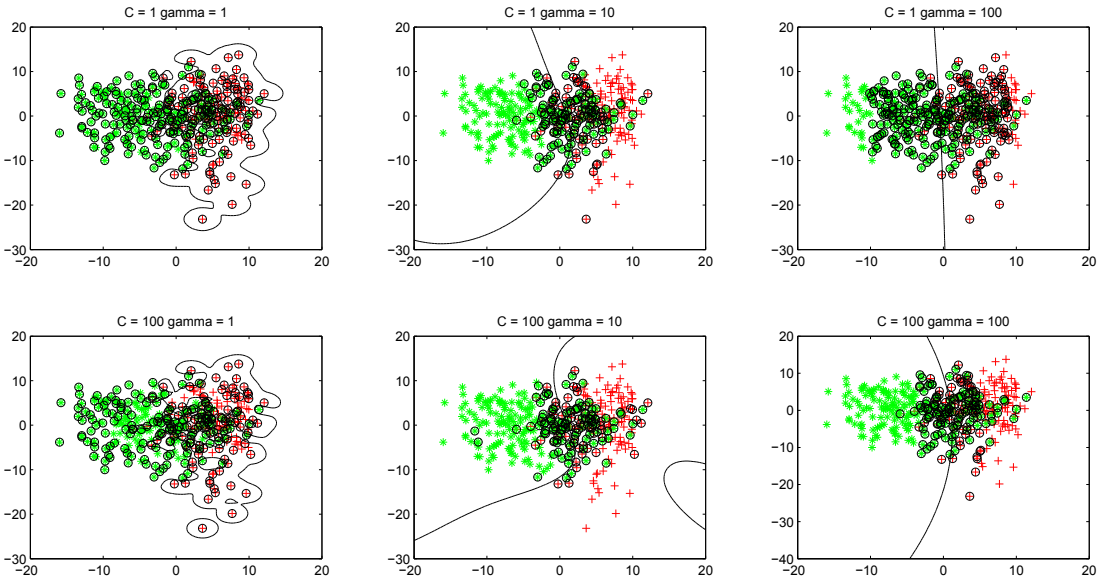

Figura 13: Ejemplo de ajuste de margen e hiperplano de decisión en SVM.

5.1. Matriz de confusión

En general para cualquier sistema de reconocimiento de patrones, se utiliza la llamada matriz de contingencia o de confusión, con la cuál se evalúa el desempeño del sistema dependiendo del número de aciertos y fallos en la etapa de clasificación de nuevos datos. Una matriz de confusión para sistemas de clasificación biclase se muestra en la Tabla 1. De acuerdo con esta matriz y tomando como referencia una de las clases en el conjunto de entrenamiento, se definen los siguientes términos:

-Verdadero positivo (TP, true positive): el número (o porcentaje) de patrones de clase 0 que el sistema clasifica correctamente como pertenecientes a la clase 0.

-Falso negativo (FN, false negative): el número (o porcentaje) de patrones de clase 0 que el sistema clasifica incorrectamente como pertenecientes a la clase 1.

-Falso positivo (FP, false positive): el número (o porcentaje) de patrones de clase 1 que le sistema clasifica incorrectamente como pertenecientes a la clase 0.

-Verdadero negativo (TN, true negative): el número (o porcentaje) de patrones de clase 1 que el sistema clasifica correctamente como pertenecientes a la clase 1.
A partir de la matriz de confusión se pueden estimar los siguientes datos:

Tasa de acierto o eficiencia (CCR, Correct Classification Rate): Esta medida es la pro-

Tabla 1: Matriz de confusión

	Clase verdadera	
Clase estimada	**Clase 0**	**Clase 1**
Clase 0	TP	FP
Clase 1	FN	TN

porción de patrones correctamente clasificados por el sistema.

$$CCR = \frac{TP + TN}{TP + FN + FP + TN} \tag{1}$$

Sensibilidad (S): La sensibilidad indica la capacidad del sistema para detectar los patrones de clase de referencia. Cuando los valores se presentan en porcentaje, la sensibilidad coincide con TP.

$$S = \frac{TP}{TP + FN} \tag{2}$$

Especificidad (E): La especificidad indica la capacidad del sistema para rechazar los patrones que no pertenecen a la clase de referencia. Cuando los valores se representan en porcentaje, la especificidad coincide con TN.

$$E = \frac{TN}{FP + TN} \tag{3}$$

Como ejemplo se compara el desempeño de los clasificadores k-NN y SVM entrenados con la base de datos de emociones de baja y alta excitación. Para la clasificación con k-NN, se utilizaron valores impares de $k \in \{1, 3, 5, 7, 9, 11\}$. Para el SVM se hizo un barrido de parámetros variando $C \in \{0{,}1, 1, 10, 100\}$ y $\gamma \in \{0{,}1, 1, 10, 100\}$. Con la siguiente función implementada en matlab, se mide el rendimiento del clasificador (**EvalRend.m**).

```
function [CMatriz, A,S,E ] = EvalRend( LR,LC,LC1,LC2 )
%Evaluacion del rendimiento del clasificador con la matriz de confusion.
%Entrada:
%LC1 = Label de la clase 1(0,1,-1,...)
%LC2 = Label de la clase 2(1,2,1,...)
%LR = Etiquetas originales de las muestras
%LC = Etiquetas asignadas a las muestras por el clasificador
%Salida:
% A = Tasa de acierto.
% S = Sensibilidad. Capacidad del sistema para detectar los patrones que
%     pertencen a la clase de referencia.
% E = Especificidad. Capacidad del sistema para rechazar los patrones que
%     no pertenecen a la clase de referencia.

%TP = True positive. Deteccion correcta. Patrones de la clase 1
%correctamente clasificados como de la clase 1.
%
%FN = False negative. Falso rechazo. Patrones de la clase 1 clasificados
```

```
%incorrectamente como pertenecientes a la clase 2.
%
%FP = False positive. Patrones de la clase 2 clasificados incorrectamente
%como de la clase 1.
%
%TN = True negative. Rechazo correcto. Patrones de la clase 2 clasificados
%correctamente como pertenecientes a la clase 2.
C1 = find(LR==LC1);
C2 = find(LR==LC2);
TP = length(find(LC(C1) == LC1));
FN = length(find(LC(C2) == LC1));
FP = length(find(LC(C1) == LC2));
TN = length(find(LC(C2) == LC2));

%Calcular porcentaje de acierto.
A = (TP + TN)/(TP+FN+FP+TN);

%Calcular sensibilidad
S = TP/(TP+FN);

%Calcular especificidad.
E = TN/(TN+FP);

CMatriz.TP =TP;
CMatriz.TN =TN;
CMatriz.FP =FP;
CMatriz.FN =FN;
end
```

Para entrenar el clasificador, se tomó aleatoriamente el 70 % (481 muestras) de los datos de la matriz de características de la base de datos de emociones. El 30 % restante (53 muestras), se tomaron como datos de prueba para evaluar la capacidad de decisión del clasificador. En la Tabla 2 se muestran los porcentajes de acierto obtenidos con k-NN. Como se puede observar, el mejor resultado se obtuvo con k=1 y k=3 (ACC = 68 %). En

Tabla 2: Porcentaje de acierto (ACC), sensibilidad (SEN) y especificidad (ESP) para el clasificador k-NN

k	**ACC (%)**	**SEN (%)**	**ESP (%)**
1	68	78	66
3	68	86	65
5	58	50	59
7	62	75	61
9	62	100	61
11	60	100	60

la Tabla 3 se muestra la matriz de confusión para k=1. En este caso se observa que 7 de 9 muestras furon clasificadas correctamente en la clase 0. Mientras que 29 de 44 muestras fueron clasificados correctamente en la clase 1. Para los resultados de clasificación con SVM, en la Tabla 4 se muestran los porcentajes de acierto obtenidos para distintos valores

Tabla 3: Matriz de confusión para k-NN con k=1

k-NN; k=1	**Clase verdadera**	
Clase estimada	**Clase 0**	**Clase 1**
Clase 0	7	2
Clase 1	15	29

de C y γ. Como se puede observar el porcentaje de acierto más alto fue de 89 %. Además los mejores resultados se obtuvieron para valores de $\gamma \geq 10$.

Tabla 4: Porcentajes de desempeño (A), sensibilidad (S) y especificidad (E) obtenidos con la SVM.

	γ											
C	**0.1**			**1**			**10**			**100**		
	A(%)	**S(%)**	**E(%)**	**A(%)**	**S(%)**	**E(%)**	**A(%)**	**S(%)**	**E(%)**	**A(%)**	**S(%)**	**E(%)**
0.1	60	60	61	83	79	87	89	81	97	81	69	100
1	58	50	60	85	80	90	89	81	97	85	75	96
10	58	50	60	79	79	80	85	80	90	89	81	97
10	58	50	60	74	70	76	85	80	90	89	81	97

5.2. Curva ROC

Una curva ROC (Receiver Operating Characteristic, en inglés) es una representación gráfica de la sensibilidad frente a la especificidad para un sistema de clasificación biclase. En general, las curvas ROC se utilizan para conocer el desempeño general de una prueba, con el calculo del área bajo la curva (AUC, Area Under the Curve). El AUC varía entre 0.5 y 1. Entre más cercano a 1 sea el valor del AUC, mejor será el desempeño del modelo.

En la Figura 14 se muestran las curvas ROC para el clasificador k-NN con k=1 y el clasificador SVM con $C = 0{,}1$ y $\gamma = 10$. Con el siguiente script escrito en matlab se entrena y guardan los resultados de los clasificadores utilizados en esta sección (**main_clasificacion.m**).

```
clc
clear
close all
warning off

addpath(genpath([pwd '\sprtool'])) %PCA
addpath(genpath([pwd '\svmtools'])) %svm
addpath(genpath([pwd '\netlab_gmm'])) %knn
addpath(genpath([pwd '\desempenno'])) %Medidas de desempe o

%Se carga una estructura que contiene la matriz de caracteristicas
%de cada clase y las etiquetas de cada una de las muestras.
%Eitquetas 0:emociones baja excitacion; 1:emociones alta excitacion
load('sample_feats.mat')
```

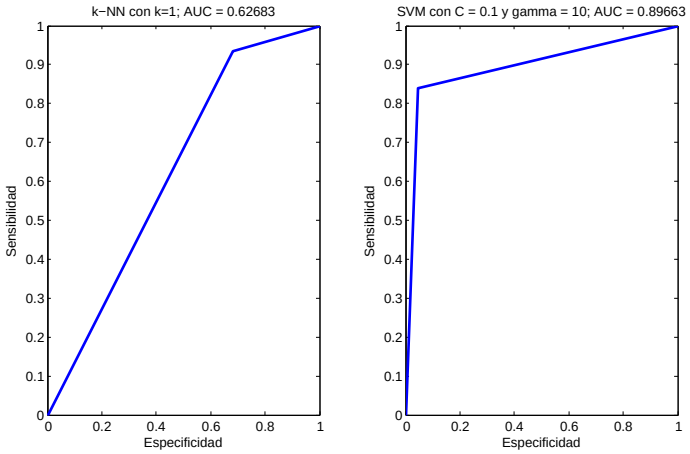

Figura 14: Curvas ROC para el clasificador k-NN y SVM.

```
% %PCA
X = zscore(data.X');
X = X';
model = pcasprt(X,2);
new_feats = linproj(X,model);

% % Partir 70% de los datos en el conjunto de entrenamiento y 30% en el
% % conjunto de prueba. La particion siempre se hace de forma aleatoria.
cv = cvpartition(data.y,'holdout',0.1);
%Seleccionar conjunto de entrenamiento
train = new_feats(:,training(cv))';
label_train = data.y(training(cv))';
%Seleccionar conjunto de prueba
test = new_feats(:,test(cv))';
label_test = data.y(~training(cv))';

%%%%%%%%%%%%%%%%CLASIFICACION CON KNN%%%%%%%%%%%%%%%%
display('Clasificacion con KNN')
%KNN
%Los labels deben quedar en codificacion  1-of-N
label_train2 = [label_train label_train];
label_train2(:,1) = zeros(size(label_train2,1),1);
k = [1,3,5,7,9,11];
for i=1:length(k)
    %Entrenamiento
    net = knn(size(train,2),2,k(i),train,label_train2);

    %clasificar datos de prueba
    [y,l] = knnfwd(net,test);

    %Cambiar etiquetas de clase (1,2) a (0,1).
    itemp = l == 1;
```

```
    l(itemp) = 0;
    l(~itemp) = 1;
    label_knn{1,i} = l;
end
for i=1:length(label_knn)
    %Matriz de confusion
    [MatConfKNN{i,1},A_knn(i,1),S_knn(i,1),E_knn(i,1)] = EvalRend(label_test,label_knn{1,i},0,
    %Curva ROC
    [x_knn{i,1},y_knn{i,1},t,auc_knn{i,1}]=perfcurve(label_test,label_knn{1,i},1);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
display('Terminado')
display('Clasificacion con SVM')
%%%%%%%%%%%%%%%%CLASIFICACION CON SVM %%%%%%%%%%%%%%%%
vC = [0.1,1,10,100];
vnsigma = [0.1,1,10,100];
XdataTrainN = dataset(train,label_train);
XdataTestN = dataset(test,label_test);

for ivc=1:length(vC)
    for insigma=1:length(vnsigma)
        % SMO conditions
        showplot = 'false';
        opts = svmsmoset('Display','off','MaxIter',1000000,...
            'KernelCacheLimit',10000,'TolKKT',1e-4);
        % Training SVM with svm toolbox
        XdataTrainN = +XdataTrainN;
        svms = svmtrain(XdataTrainN,label_train,'BoxConstraint',vC(ivc),'Kernel_Function','rbf
            'Method','SMO','AutoScale','false','showplot',showplot,'SMO_Opts',opts) ;

        %Clasificar datos de prueba
        [l, fy_test] = A_svmclassify(svms,+XdataTestN);
        label_svm{ivc,insigma} = l;
    end
end

for i=1:length(vC)
    for j=1:length(vnsigma)
        %Matriz de confusion
        [MatConfSVM{i,j},A_svm(i,j),S_svm(i,j),E_svm(i,j)] = EvalRend(label_test,label_svm{i,j
        %Curva ROC
        [x_svm{i,j},y_svm{i,j},t,auc_svm{i,j}]=perfcurve(label_test,label_svm{i,j},1);
    end
end

subplot(1,2,1)
plot(x_knn{1,1},y_knn{1,1},'LineWidth',2)
ylabel('Sensibilidad')
xlabel('Especificidad')
title(['k-NN con k=1; AUC = ' num2str(auc_knn{1,1})])
subplot(1,2,2)
plot(x_svm{1,3},y_svm{1,3},'LineWidth',2)
title(['SVM con C = 0.1 y gamma = 10; AUC = ' num2str(auc_svm{1,3})])
ylabel('Sensibilidad')
```

```
xlabel('Especificidad')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
display('Terminado')
```