PER (E.T.S. de Ingeniería Informática)
Curso 2018-2019

Práctica 0. Introducción a Octave

Jorge Civera Saiz, Carlos D. Martínez Hinarejos
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

Índice

1. Trabajo previo a la sesión de prácticas

Para la realización de esta práctica se supone que se ha adquirido previamente experiencia en el uso de *shell scripts*, *awk* y *gnuplot*, tanto en Sistemas Inteligentes (SIN) como en otras asignaturas cursadas. El alumno deberá haber leído de forma detallada la totalidad del boletín práctico, para poder centrarse en profundidad en la parte que hay que desarrollar en el laboratorio, la cual durará dos sesiones.

2. Introducción

Varias de las técnicas empleadas dentro del área de Reconocimiento de Formas (RF) emplean cálculos matriciales. Es el caso de *Principal Component Analysis* (PCA) y *Linear Discriminant Analysis* (LDA), así como clasificadores Bernoulli, multinomial y gaussiano entre otros. Por tanto, la aplicación de una herramienta que implemente de forma sencilla estos cálculos matriciales puede ayudar a obtener más rápidamente los sistemas de RF que hacen uso de estas funcionalidades.

Una de las herramientas comerciales más potentes en cálculo matricial es MATLAB, pero existe una herramienta de código libre que presenta capacidades semejantes: *GNU Octave*.

GNU Octave es un lenguaje de alto nivel interpretado definido inicialmente para computación numérica. Posee capacidades de cálculo numérico para solucionar problemas lineales y no lineales, así como otros experimentos numéricos. Posee capacidades gráficas para visualizar y manipular los datos. Se puede usar de forma interactiva o no (empleando ficheros que guarden programas a interpretar). Su sintaxis y semántica es muy semejante a MATLAB, lo que hace que los programas de éste sean fácilmente portables a Octave.

Octave está en continuo crecimiento y puede descargarse y consultarse su documentación y estado en su web `http://www.gnu.org/software/octave/`. Aunque está definido para funcionar en GNU/Linux, también es portable a otras plataformas como OS X y MS-Windows (los detalles pueden consultarse en su web).

3. Órdenes básicas de *Octave*

Octave puede ejecutarse desde la línea de órdenes o desde el menú de aplicaciones del entorno gráfico. Para ejecutar desde la línea de órdenes se abre un terminal y se escribe:

```
octave
```

Generalmente, obtenemos una salida similar a:

```
GNU Octave, version 3.8.2
Copyright (C) 2014 John W. Eaton and others.
This is free software; see the source code for copying conditions.
There is ABSOLUTELY NO WARRANTY; not even for MERCHANTABILITY or
FITNESS FOR A PARTICULAR PURPOSE.  For details, type 'warranty'.

Octave was configured for "x86_64-redhat-linux-gnu".

Additional information about Octave is available at http://www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/get-involved.html

Read http://www.octave.org/bugs.html to learn how to submit bug reports.
```

```
For information about changes from previous versions, type 'news'.

octave:1>
```

La última línea tendrá un cursor indicando que se esperan órdenes de Octave. Estamos pues en el modo **interactivo**.

3.1. Aritmética básica

Octave acepta a partir de este momento expresiones aritméticas sencillas (operadores `+`, `-`, `*`, `/` y `^`, este último para exponenciación), funciones trigonométricas (`sin`, `cos`, `tan`, `arcsin`, `arccos`, `arctan`), logaritmos (`log`, `log10`), exponencial neperiana (`e^n` ó `exp(n)`) y valor absoluto (`abs`). Como respuesta a estas expresiones da valor a la variable `ans`, mostrándola, pero también puede asignarse a otras variables. Por ejemplo:

```
octave:1> sin(1.71)
ans =  0.99033
octave:2> b=sin(2.16)
b =  0.83138
```

Para consultar el valor de una variable, basta con escribir su nombre:

```
octave:3> b
b =  0.83138
octave:4> ans
ans =  0.99033
```

Aunque también puede emplearse la función `disp`, que muestra el contenido de la variable omitiendo su nombre:

```
octave:5> disp(b)
 0.83138
```

Las variables pueden usarse en otras expresiones:

```
octave:6> c=b*ans
c =  0.82334
```

Puede evitarse que se muestre el resultado en cada operación añadiendo `;` al final de la operación:

```
octave:7> d=ans*b*5;
octave:8> d
d =  4.1167
```

3.2. Operadores básicos en vectores y matrices

La notación matricial en Octave es entre corchetes ([ ]); en su interior, las filas se separan por punto y coma (;) y las columnas por espacios en blanco o coma (,). Por ejemplo, para crear un vector fila de dimensión 3, un vector columna de dimensión 2 y una matriz de 3×2, se puede hacer:

```
octave:9> v1=[1 3 -5]
v1 =

   1   3  -5

octave:10> v2=[4;2]
v2 =

   4
   2

octave:11> m=[3,-4;2 1;-5 0]
m =

   3  -4
   2   1
  -5   0
```

Siempre y cuando las dimensiones de los elementos vectoriales y matriciales implicados sean apropiados, sobre ellos se pueden aplicar operadores de suma (+), diferencia (-) o producto (*). El operador potencia ^ puede aplicarse sobre matrices cuadradas. Los operadores producto, división y potencia tienen la versión "elemento a elemento" (.*, ./, .^). Por ejemplo:

```
octave:12> mv=v1*m
mv =

   34   -1

octave:13> mx=m.*5
mx =

   15  -20
   10    5
  -25    0

octave:14> v3=v1*v2
error: operator *: nonconformant arguments (op1 is 1x3, op2 is 2x1)
```

```
octave:15> v3=v1*[3;5;6]
v3 = -12
octave:16> mvv=[3;5;6]*v1
mvv =

    3    9  -15
    5   15  -25
    6   18  -30
```

Se pueden aplicar operadores de comparación como `>`, `<`, `>=`, `<=`, `==` y `!=`, que retornan una matriz binaria con 1 en las posiciones en las que se cumple la condición y 0 en caso contrario:

```
octave:17> m>=0
ans =

   1   0
   1   1
   0   1

octave:18> m!=0
ans =

   1   1
   1   1
   1   0
```

Se pueden emplear también en la comparación de vectores y matrices de dimensiones congruentes, dando la matriz binaria resultado de comparar los elementos en la misma posición en ambas estructuras. Por ejemplo:

```
octave:19> v1<[2 1 -3]
ans =

   1   0   1
```

También existe el operador de transposición (`'`):

```
octave:20> m2=m'
m2 =

   3   2  -5
  -4   1   0
```

El indexado de los elementos se hace entre paréntesis. Para vectores puede indicarse una posición o lista de posiciones:

```
octave:21> v1(2)
ans =  3
octave:22> v1([2 3])
ans =

   3  -5
octave:23> v2(2)
ans =  2
```

Mientras que para una matriz se espera un vector de índices para seleccionar filas o columnas:

```
octave:24> m3=[1 2 3 4;5 6 7 8;9 10 11 12]
m3 =

    1    2    3    4
    5    6    7    8
    9   10   11   12

octave:25> m3([2 3],[2 4])
ans =

    6    8
   10   12
```

Para indicar una fila completa (o una columna completa), se pueden emplear los dos puntos (`:`):

```
octave:26> m3([2 3],:)
ans =

    5    6    7    8
    9   10   11   12
```

Para indicar un rango de filas o columnas, se emplea `i:f`, donde `i` es el índice inicial y `f` el final. Se puede emplear la notación `i:inc:f`, donde `inc` indica el incremento (por omisión es 1).

```
octave:27> m3([2 3],2:4)
ans =

    6    7    8
   10   11   12

octave:28> m3([1 3],1:2:4)
ans =
```

```
    1    3
    9   11
```

El uso de vectores de índices es una funcionalidad muy útil, pues permite seleccionar aquellas filas o columnas de interés para nuestro propósito, incluso reordenar las filas o columnas de una matriz. Por ejemplo:

```
octave:29> m3(:,[4 3 2 1])
ans =

    4    3    2    1
    8    7    6    5
   12   11   10    9
```

Para indicar el último índice de una dimensión se puede emplear la palabra `end`:

```
octave:30> m3([1 3],end)
ans =

    4
   12
```

3.3. Funciones básicas en vectores y matrices

Octave aporta múltiples funciones para operar con vectores y matrices. Las más importantes son:

- `size(m)`: devuelve el número de filas y columnas de la matriz (en el caso de un vector, una de las dimensiones tendrá tamaño 1)
- `rows(m)`, `columns(m)`: devuelve el número de filas y columnas de la matriz, respectivamente.
- `eye(f,c)`, `ones(f,c)`, `zeros(f,c)`: dan la matriz identidad, todo unos y todo ceros, respectivamente, de tamaño `f`×`c`; si `c` se omite, será de `f`×`f`
- `sum(v)`, `sum(m)`: devuelve la suma de los elementos del vector o matriz; en el caso de una matriz, devuelve el vector resultante de las sumas por columnas; si se le pasa un segundo argumento `sum(m,n)`, éste indica la dimensión a sumar (1 por columnas, 2 por filas).
- `max(v)`, `max(m)`: indica el valor máximo del vector `v` o el vector con los máximos por columna de la matriz `m`; si se pide que devuelva dos resultados `[r1,r2]=max(v)`, el primer resultado almacena el máximo y el segundo, la posición de este máximo. Esto es extrapolable al caso de matrices `[r1,r2]=max(m)`, siendo `r1` un vector de máximos y `r2` un vector de posiciones de máximos por columnas.

- `unique(v)`: devuelve los valores diferentes del vector `v` en orden creciente.
- `det(m)`: determinante de `m`.
- `eig(m)`: vector de valores propios de `m` o su versión.
- `diag(m)`: devuelve la diagonal de una matriz `m` en forma de vector.
- `inv(m)`: inversa de la matriz `m` si ésta es no singular
- `sort(v)`: devuelve el vector ordenado con los valores del vector `v` de menor a mayor. Si se invoca con dos resultados `[s,i]=sort(v)`, entonces `s` es el vector ordenado e `i` es el vector de índices de ordenación. Es decir, `i` aplicado a `v` nos devuelve `s`: `s=v(i)`
- `find(v)`, `find(m)`: recibe un vector o matriz y devuelve los índices (posiciones) de aquellos elementos que no son cero (índices absolutos empezando en 1 y haciendo el recorrido por cada columna y por filas ascendentes en caso de matrices).

```
octave:31> v = [0 2 0 2 1 0]
v =

   0   2   0   2   1   0

octave:32> find(v)
ans =

   2   4   5
```

Es muy útil para aplicarla combinada con operaciones lógicas a fin de buscar elementos de un vector o matriz que cumplen una condición.

```
octave:33> find(v==2)
ans =

   2   4
```

En este caso, sabemos que en las posiciones 2 y 4 del vector hay un valor igual a 2. Observa que `find()` se ejecuta sobre el resultado de `v==2`.

```
octave:34> v==2
ans =

   0   1   0   1   0   0
```

Otro caso de uso de `find()` es la selección de filas o columnas que cumplen una condición. Por ejemplo, selecciona las filas de la matriz cuya última columna es un 2.

```
octave:35> A=[1 3 5 1;5 7 9 1; 0 2 4 2; 4 6 8 2]
A =

   1   3   5   1
   5   7   9   1
   0   2   4   2
   4   6   8   2
```

Primero generamos un vector de índices que nos indica que filas tienen un `2` en su última columna.

```
octave:36> vi=find(A(:,end)==2)
vi =

   3
   4
```

El vector de índices `vi` se utiliza para seleccionar las filas cuya última columna es un 2.

```
octave:37> A2=A(vi,:)
A2 =

   0   2   4   2
   4   6   8   2
```

- `repmat(m,f,c)`: crea una matriz de `f`×`c` bloques de `m`; si `c` se omite, será de `f`×`f`

```
octave:38> m=[1 2; 3 4]
m =

   1   2
   3   4

octave:39> M=repmat(m,2,3)
M =

   1   2   1   2   1   2
   3   4   3   4   3   4
   1   2   1   2   1   2
   3   4   3   4   3   4
```

Se puede entender `M` como una matriz `2 x 3` donde cada elemento es la matrix `m`.

- `randperm(N)`: devuelve un vector de permutación aleatoria de `N` elementos

```
octave:40> v=randperm(5)
v =

   2   1   4   5   3
```

Se puede utilizar para permutar aleatoriamente (barajar) las filas o columnas de un vector o matriz.

```
octave:41> A=[0 1;2 3; 4 5; 6 7; 8 9]
A =

   0   1
   2   3
   4   5
   6   7
   8   9

octave:42> A(v,:)
ans =

   2   3
   0   1
   6   7
   8   9
   4   5
```

3.4. Carga y guardado de datos

La introducción de datos de forma manual no es apropiada para grandes cantidades de datos. Por tanto, Octave aporta funciones que permiten cargar y guardar en ficheros. El guardado se hace mediante la orden `save`:

```
octave:43> save "m3.dat" m3
```

El fichero tiene el siguiente formato:

```
# Created by Octave 3.0.5, DATE <user@machine>
# name: m3
# type: matrix
# rows: 3
# columns: 4
 1 2 3 4
 5 6 7 8
 9 10 11 12
```

Los ficheros de datos a cargar deben seguir este formato, indicando en la línea `# name:` el nombre de la variable en la que se cargarán los datos. Por ejemplo, ante un fichero `maux.dat` con contenido:

```
# Created by Octave 3.0.5, DATE <user@machine>
# name: A
# type: matrix
# rows: 4
# columns: 3
 1 2 -3
 5 -6 7
 -9 10 11
 -5 2 -1
```

Se cargaría con:

```
octave:44> load "maux.dat"
octave:45> A
A =

    1    2   -3
    5   -6    7
   -9   10   11
   -5    2   -1
```

La orden `save` puede usarse con opciones como `-text` (guarda en formato texto con cabecera, por omisión), `-ascii` (guarda en formato texto sin cabecera), `-z` (guarda en formato comprimido), o `-binary` (guarda en binario). Por ejemplo:

```
octave:33> save -ascii "m3woh.dat" m3
```

En ocasiones, el guardado de datos puede provocar pérdida de precisión, pues por omisión se guarda hasta el cuarto decimal. Para modificar esta precisión de guardado, se puede ejecutar previamente `save_precision(n)`, donde `n` es el número de cifras significativas que se guardarán en formato texto (`-text`).

3.5. Funciones Octave

En Octave se pueden definir funciones que hagan tareas más específicas y complejas. Las funciones Octave pueden recibir varios parámetros y pueden devolver varios valores de retorno (que pueden incluir vectores y matrices).
La sintaxis básica de una función en Octave es:

```
function [ lista_valores_retorno ] = nombre ( [ lista_parametros ] )
  cuerpo
end
```

Un posible ejemplo sería:

```
octave:46> function [madd,msub] = addsub(ma,mb) madd=ma+mb; msub=ma-mb; end
octave:47> mat1=[1,2;3,4]
mat1 =

   1   2
   3   4

octave:48> mat2=[-1,2;3,-4]
mat2 =

  -1   2
   3  -4

octave:49> [mr1,mr2]=addsub(mat1,mat2)
mr1 =

   0   4
   6   0

mr2 =

   2   0
   0   8
```

Estas funciones se definen habitualmente en ficheros `.m` (de código Octave) cuyo nombre debe ser el mismo que la función que incluyen (en nuestro ejemplo, `addsub.m`), y que deben situarse en el mismo directorio en el que se ejecuta Octave. De esa forma, se puede acceder a las funciones sin necesidad de definirlas cada vez.

3.6. Programas Octave

Octave se puede usar de forma **no interactiva** escribiendo *scripts* que son interpretados por Octave y donde se pueden emplear las mismas instrucciones que en el modo interactivo. Por ejemplo, suponiendo que tenemos en el directorio actual el fichero `addsub.m` con la función previamente definida:

```
function [madd,msub] = addsub(ma,mb)
  madd=ma+mb;
  msub=ma-mb;
end
```

desde la terminal editamos el siguiente *script* en el fichero `prueba.m`:

```
#!/usr/bin/octave -qf

a=[1 2 3;4 5 6;7 8 9];
b=[9 8 7;6 5 4;3 2 1];
c=a+b;

[d,e]=addsub(a,b)

disp(c)
```

Desde la línea de comandos hay que darle permisos de ejecución (`chmod +x prueba.m`) y se puede ejecutar como cualquier programa o *script*: `./prueba.m`. La salida será:

```
d =

   10   10   10
   10   10   10
   10   10   10

e =

  -8  -6  -4
  -2   0   2
   4   6   8

   10   10   10
   10   10   10
   10   10   10
```

Estos programas también pueden ejecutarse desde la línea interactiva de Octave escribiendo su nombre (sin `.m`):

```
octave:50> prueba
d =

   10   10   10
   10   10   10
   10   10   10

e =

  -8  -6  -4
  -2   0   2
   4   6   8
```

```
   10   10   10
   10   10   10
   10   10   10
```

En este caso, se pueden añadir argumentos en la línea de órdenes y pueden usarse la variable `nargin` (número de argumentos) y la función `argv()` (devuelve la lista de los argumentos pasados). Estos argumentos están en formato de cadena, y pueden convertirse a formato numérico en caso necesario empleando la función `str2num(c)`.

Por ejemplo, un script `exp.m` para realizar un experimento que recibe como parámetros el nombre del fichero de entrenamiento y de test realizando un barrido de un parámetro `k` de un valor mínimo a un valor máximo con un determinado incremento (`step`):

```
#!/usr/bin/octave -qf

if (nargin!=5)
 printf("Usage: exp.m <train> <test> <mink> <stepk> <maxk>\n");
 exit(1);
end

arg_list=argv();
train=arg_list{1};
test=arg_list{2};
mink=str2num(arg_list{3});
stepk=str2num(arg_list{4});
maxk=str2num(arg_list{5});

for k=mink:stepk:maxk
...
```

Como es de esperar de cualquier lenguaje de programación, Octave dispone de operadores de iteración (`for`, `while`) y condicionales (`if`, `switch`). Sin embargo, el operador `for` permite iterar sobre los valores de un vector:

```
octave:51> for i=[3 -4 1 5] i end
i =  3
i = -4
i =  1
i =  5
```

De igual forma se puede iterar sobre una secuencia de valores:

```
octave:52> s=0; for i=1:2:9 s=s+i; end; s
s =  25
```

4. Ejercicios

Los siguientes ejercicios propuestos están preparados para poner en práctica los conceptos presentados en este boletín, de forma que se puedan afrontar las siguientes dos prácticas que hacen uso de ellos. Estos ejercicios no es necesario entregarlos.

Primeramente, se plantean una serie de ejercicios básicos que un usuario con algunos conocimientos de Octave podría obviar. Seguidamente, se retoma la práctica 2 de SIN para plantear una optimización del código proporcionado en su momento. Finalmente, se complementa la formación en Octave con una serie de ejercicios avanzados cuya resolución preparara al alumno adecuadamente para las dos prácticas siguientes.

4.1. Ejercicios básicos

1. Realiza el producto escalar de los vectores $v_1 = (1, 3, 8, 9)$ y $v_2 = (-1, 8, 2, -3)$.

2. *a*) Obtén la matriz de dimensiones 4×4 a partir del producto de los vectores v_1 y v_2
   b) Calcula su determinante
   c) Calcula sus valores propios.

3. Sobre la matriz del ejercicio 2:

   a) Calcula su submatriz 2×2 formadas por las filas 1 y 3 y las columnas 2 y 3.
   b) Súmale la matriz todo unos a la submatriz resultante.
   c) Calcula el determinante de la matriz resultante.
   d) Calcula la inversa de la matriz resultante.

4. Sobre la matriz inversa resultado del ejercicio 3:

   a) Calcula su máximo valor y su posición.
   b) Calcula las posiciones (fila y columna) de los elementos mayores que 0.
   c) Calcula la suma de cada columna.
   d) Calcula la suma de cada fila.

5. Salva la matriz inversa resultante del ejercicio 3 con hasta 7 dígitos decimales; comprueba que se ha guardado correctamente.

6. Define una función Octave que reciba una matriz y devuelva la primera fila y la primera columna de esa matriz.

7. Implementa un *script* Octave que lea una matriz de un fichero `data` y guarde su traspuesta en el fichero `data_trans`.

4.2. Ejercicio: optimizando un clasificador lineal

En PoliformaT se encuentra el fichero `SINLab2.tgz` para este ejercicio. En la práctica 2 de la asignatura de SIN se estudia una implementación en Octave del algoritmo Perceptron:

```
function [w,E,k]=perceptron(data,b,a,K,iw)
  [N,L]=size(data); D=L-1;
  labs=unique(data(:,L)); C=numel(labs);
  if (nargin<5) w=zeros(D+1,C); else w=iw; end
  if (nargin<4) K=200; end;
  if (nargin<3) a=1.0; end;
  if (nargin<2) b=0.1; end;
  for k=1:K
    E=0;
    for n=1:N
      xn=[1 data(n,1:D)]';
      cn=find(labs==data(n,L));
      er=0; g=w(:,cn)'*xn;
      for c=1:C; if (c!=cn && w(:,c)'*xn+b>g)
        w(:,c)=w(:,c)-a*xn; er=1; end; end
      if (er)
        w(:,cn)=w(:,cn)+a*xn; E=E+1; end; end
    if (E==0) break; end; end
endfunction
```

Este algoritmo estima un conjunto de pesos `w` que minimiza el error de clasificación `E` de un conjunto de muestras etiquetadas `data`. Asimismo, hay dos parámetros que controlan el comportamiento del algoritmo: el factor de aprendizaje `a` y el margen `b`.

También se disponía de una implementación del clasificador para funciones discriminantes lineales en la función Octave `linmach.m`:

```
function cstar=linmach(w,x)
  C=columns(w); cstar=1; max=-inf;
  for c=1:C
    g=w(:,c)'*x;
    if (g>max) max=g; cstar=c; endif; end
endfunction
```

Esta función dada una matriz de pesos `w`, donde los pesos de cada clase están dispuestos por columnas, y una muestra de test `x`, devuelve la etiqueta de clase `cstar` en la que se clasifica la muestra `x`.

Ejercicio 1. Implementa una versión *matricial* de la función `linmach.m` que, en lugar de recibir una única muestra de test, reciba un conjunto de muestras de test dispuestas

por filas en una matriz `x`, y devuelva un vector de etiquetas de clase `cstar`, donde cada elemento (fila) es la clasificación de una muestra de test.

No se deben utilizar operadores iterativos (`for`, `while`, etc.) para su implementación, sino el producto matricial y la función máximo `[r1,r2]=max(v)`. Para comprobar tu versión matricial de la función `linmach.m`, utiliza el conjunto de datos `OCR_14x14.gz` y modifica adecuadamente el script Octave `experiment.m`:

```
#!/usr/bin/octave -qf

load("OCR_14x14.gz");
[N,L]=size(data); D=L-1;
ll=unique(data(:,L));
C=numel(ll); rand("seed",23);
data=data(randperm(N),:);
[w,E,k]=perceptron(data(1:round(.7*N),:));
M=N-round(.7*N); te=data(N-M+1:N,:);
rl=zeros(M,1);
for m=1:M
  tem=[1 te(m,1:D)]';
  rl(m)=ll(linmach(w,tem)); end
[nerr m]=confus(te(:,L),rl)
```

Ejercicio 2. La función `confus` proporciona una estimación del error empírico y de la matriz de confusión al comparar la etiqueta de clase real `te(:,L)` con la etiqueta de clase estimada `rl` por el clasificador. Reemplaza la llamada a `confus` por tu propia estimación del error empírico (por simplicidad, no calcules la matriz de confusión) sin utilizar operadores iterativos (`for`, `while`, etc.). Pista: utiliza los operadores lógicos.

Ejercicio 3. La función `perceptron.m` devuelve el vector de pesos estimado para cada clase, la iteración `k` en la que se detuvo el proceso de entrenamiento y el número de errores de clasificación en el conjunto de entrenamiento en esa iteración k:

```
function [w,E,k]=perceptron(data,b,a,K,iw)
  [N,L]=size(data); D=L-1;
  labs=unique(data(:,L)); C=numel(labs);
  if (nargin<5) w=zeros(D+1,C); else w=iw; end
  if (nargin<4) K=200; end;
  if (nargin<3) a=1.0; end;
  if (nargin<2) b=0.1; end;
  for k=1:K
    E=0;
    for n=1:N
      xn=[1 data(n,1:D)]';
      cn=find(labs==data(n,L));
      er=0; g=w(:,cn)'*xn;
```

```
      for c=1:C; if (c!=cn && w(:,c)'*xn+b>g)
        w(:,c)=w(:,c)-a*xn; er=1; end; end
      if (er)
        w(:,cn)=w(:,cn)+a*xn; E=E+1; end; end
    if (E==0) break; end; end
endfunction
```

Un estudio empírico muy interesante consiste en analizar la evolución del número de errores de clasificación tanto en el conjunto de entrenamiento como en el conjunto de test tras cada iteración durante el proceso de entrenamiento. Para ello es necesario modificar la función `perceptron.m` de forma que su declaración sea:

```
function [w,Etr,Ete,k]=perceptron(tr,te,b,a,K,iw)
```

donde `tr` y `te` son los conjuntos de entrenamiento y test, respectivamente, y `Etr` y `Ete` son vectores que almacenan en cada iteración el error de clasificación en el conjunto de entrenamiento y test, respectivamente.

La representación gráfica del número de errores de clasificación en el conjunto de entrenamiento y test en función de la iteración te permitirá observar el fenómeno del sobreentrenamiento.

Ejercicio 4. El algoritmo Perceptron optimiza el vector de pesos de cada clase para minimizar el número de errores de clasificación en el conjunto de entrenamiento. Este algoritmo aplica la conocida técnica de descenso por gradiente para optimizar los vectores de pesos restando o sumando una cantidad que es proporcional a la muestra mal clasificada:

```
w(:,c)=w(:,c)-a*xn;          w(:,cn)=w(:,cn)+a*xn;
```

Esta cantidad es básicamente el gradiente de la función objetivo respecto a los pesos. En la versión del algoritmo Perceptron estudiada en la práctica 2 de SIN, la modificación (suma del gradiente) de los vectores de pesos se realiza tras cada muestra, si es necesario. Sin embargo, existen otras aproximaciones:

- Batch: Acumula el gradiente de todas las muestras del conjunto de entrenamiento y lo suma al vector de pesos al final de cada iteración.
- Mini-batch: Acumula el gradiente de un subconjunto de M muestras y lo suma al vector de pesos.

La aproximación basada en mini-batch, utilizada habitualmente para entrenar redes neuronales, busca un descenso por gradiente más estable y que lleve a un mejor óptimo local del vector de pesos. Además, la aproximación mini-batch permite estudiar el error de clasificación en función del número de muestras M utilizado. Implementa una versión mini-batch del algoritmo Perceptron:

```
function [w,Etr,Ete,k]=perceptron(tr,te,b,a,K,M,iw)
```

4.3. Ejercicios adicionales

En PoliformaT se encuentra el fichero `videosTr.gz`. Este fichero contiene vectores de 2000 características extraídos de vídeos de baloncesto o no-baloncesto de Youtube. Cada fila contiene 2001 componentes, 2000 características más la etiqueta de clase (0 ó 1). Puedes consultar la cabecera del fichero `videosTr.gz` para saber el nombre de la variable que es creada en `Octave` al cargar ese fichero. A continuación, calcula utilizando operaciones matriciales:

1. El módulo de cada vector de características.

$$|\mathbf{x}| = \sqrt{\sum_{d=1}^{D} x_d^2}$$

2. El reordenamiento de los vectores de características por valor de su módulo de mayor a menor.

3. Los vectores de características de módulo unitario

$$\mathbf{x}_u = \frac{\mathbf{x}}{|\mathbf{x}|}$$

4. La distancia Euclídea del primer vector de características al resto de vectores de características:

$$L_2(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{d=1}^{D} (\mathbf{x}_d - \mathbf{y}_d)^2}$$

5. La media $\overline{\mathbf{x}_c}$ y matriz de covarianzas Σ_c de cada clase.

$$\overline{\mathbf{x}_c} = \frac{1}{N_c} \sum_{n:c_n=c} \mathbf{x}_n \qquad \Sigma_c = \frac{1}{N_c} \sum_{n:c_n=c} (\mathbf{x}_n - \overline{\mathbf{x}_c})(\mathbf{x}_n - \overline{\mathbf{x}_c})^t$$

   donde N_c es el número de vectores de la clase c.

6. La normalización de los vectores de características utilizando media y desviación típica de su clase:

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - \overline{\mathbf{x}_c}}{\sigma_c}$$

   donde σ_c es la diagonal de la matriz de covarianzas Σ_c.