PER (E.T.S. de Ingeniería Informática)
Curso 2020-2021

Práctica 0. Introducción a Octave y la tarea de clasificación MNIST

Jorge Civera Saiz, Carlos D. Martínez Hinarejos y Javier Iranzo Sánchez
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*, `gnuplot` y `octave`, 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) y Aprendizaje Automático (AA) emplean cálculos matriciales. Es el caso de *Principal Component Analysis* (PCA) y *Linear Discriminant Analysis* (LDA), así como clasificadores basados en funciones discriminantes lineales (Perceptron), o en distribuciones de probabilidad como son la Bernoulli, la multinomial y la Gaussiana 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 y AA 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: *Octave*.

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. Tarea de clasificación: MNIST

En esta sección vamos a introducir la tarea de clasificación MNIST que se utilizará como tarea de referencia para evaluar los distintos clasificadores que se estudian tanto en la asignatura de Percepción, como en la asignatura de Aprendizaje Automático del primer semestre del cuarto año.

3.1. Descripción

La base de datos MNIST[1] consiste en una colección de imágenes de dígitos manuscritos (10 clases) con unas dimensiones de 28 x 28 píxeles en escala de 256 niveles de

[1] http://yann.lecun.com/exdb/mnist

grises. Las dígitos que aparecen en las imágenes han sido normalizados en tamaño (20 x 20 píxeles) y centrados. Esta base de datos es un subconjunto de una base de datos más grande disponible desde el *National Institute of Standards and Technology* (NIST). Ha sido particionada en 60.000 imágenes de entrenamiento y 10.000 de test, que corresponde con los siguientes cuatro ficheros en formato Octave:

- Imágenes de entrenamiento: 60000 x 784 (`train-images-idx3-ubyte.mat.gz`)
- Etiquetas de clase de entrenamiento: 60000 x 1 (`train-labels-idx1-ubyte.gz`)
- Imágenes de test: 10000 x 784 (`t10k-images-idx3-ubyte.mat.gz`)
- Etiquetas de clase de test: 10000 x 1 (`t10k-labels-idx1-ubyte.mat.gz`)

En la Figura 1 se muestra una representación directa de un dígito cero que se corresponde con la segunda fila de datos del fichero `train-images-idx3-ubyte.mat.gz` que ha sido formateada a 28 filas y 28 columnas. Se puede apreciar como el tamaño del dígito está normalizado a 20 x 20 píxeles centrado sobre un fondo blanco.

La tarea MNIST ha sido cuidadosamente elaborada para que el conjunto de escritores de entrenamiento y test sea disjunto. De esta forma, no hay dígitos del mismo escritor en el entrenamiento y test. Asimismo, existen dos tipos de escritores que conviven en el conjunto de entrenamiento y en el test, que se corresponde con estudiantes de instituto y trabajadores de la oficina del censo, respectivamente. Estos últimos poseían una escritura más clara y fácil de reconocer.

En la web de la base de datos MNIST se proporcionan muchos más detalles sobre su elaboración. Asimismo, en esta misma página web se muestra una tabla de clasificadores (y preproceso aplicado) con la tasa de error conseguida sobre esta tarea y la referencia en forma de enlace a una descripción más detallada sobre el resultado conseguido por el investigador correspondiente.

MNIST es una base de datos adecuada para aquellos que desean probar técnicas de aprendizaje automático y métodos de procesamiento de patrones en datos reales dedicando un esfuerzo mínimo al procesamiento de las imágenes y el formato.

3.2. Carga de datos

Los ficheros Octave mencionados anteriormente son ficheros *ascii* comprimidos en formato Octave y están disponibles en PoliformaT.

Si examinamos `train-images-idx3-ubyte.mat.gz` desde el intérprete de comando (por ejemplo con `zless`) veremos que contiene una cabecera como esta:

```
# name: X
# type: matrix
# rows: 60000
# columns: 784
```

```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  51 159 253 159  50   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0  48 238 252 252 252 237   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0  54 227 253 252 239 233 252  57   6   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0  10  60 224 252 253 252 202  84 252 253 122   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0 163 252 252 252 253 252 252  96 189 253 167   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0  51 238 253 253 190 114 253 228  47  79 255 168   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0  48 238 252 252 179  12  75 121  21   0   0 253 243  50   0   0   0   0   0
  0   0   0   0   0   0   0   0  38 165 253 233 208  84   0   0   0   0   0   0 253 252 165   0   0   0   0   0
  0   0   0   0   0   0   0   7 178 252 240  71  19  28   0   0   0   0   0   0 253 252 195   0   0   0   0   0
  0   0   0   0   0   0   0  57 252 252  63   0   0   0   0   0   0   0   0   0 253 252 195   0   0   0   0   0
  0   0   0   0   0   0   0 198 253 190   0   0   0   0   0   0   0   0   0   0 255 253 196   0   0   0   0   0
  0   0   0   0   0   0  76 246 252 112   0   0   0   0   0   0   0   0   0   0 253 252 148   0   0   0   0   0
  0   0   0   0   0   0  85 252 230  25   0   0   0   0   0   0   0   0   7 135 253 186  12   0   0   0   0   0
  0   0   0   0   0   0  85 252 223   0   0   0   0   0   0   0   0   7 131 252 225  71   0   0   0   0   0   0
  0   0   0   0   0   0  85 252 145   0   0   0   0   0   0   0  48 165 252 173   0   0   0   0   0   0   0   0
  0   0   0   0   0   0  86 253 225   0   0   0   0   0   0 114 238 253 162   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0  85 252 249 146  48  29  85 178 225 253 223 167  56   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0  85 252 252 252 229 215 252 252 252 196 130   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0  28 199 252 252 253 252 252 233 145   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0  25 128 252 253 252 141  37   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Figura 1: Representación *ascii* de un dígito cero manuscrito de MNIST.

donde `#name` indica el nombre de la variable (matriz Octave) en la que se cargarán los datos, `#type` es el tipo de variable, `#rows` es el número de filas y finalmente `#columns` es el número de columnas. Por lo tanto el fichero `train-images-idx3-ubyte.mat.gz` contiene una matriz representando 60.000 imágenes por filas y 784 dimensiones por columnas.

En Octave cargaremos esta matriz con la siguiente orden:

```
octave:1> load train-images-idx3-ubyte.mat.gz
```

Comprobaremos que la variable X se ha cargado, así como su tamaño:

```
octave:2> size(X)
ans =

   60000     784
```

1 fila = 1 muestra

El fichero de datos de test `t10k-images-idx3-ubyte.mat.gz` cuando se cargue se instanciará en la variable `Y`.

Dado que el objetivo es evaluar la tasa de error de clasificación disponemos de las etiquetas de clase de las imágenes, tanto de entrenamiento como de test. Estas etiquetas están en los ficheros `train-labels-idx1-ubyte.gz` y `t10k-labels-idx1-ubyte.mat.gz`, respectivamente. Cargad estos ficheros para comprobar las dimensiones de las variables.

También existe la posibilidad de guardar variables a fichero mediante la función `save`. Esta función permite con opciones como `-text` (guarda en formato texto con cabecera, por omisión), `-ascii` (guarda en formato texto sin cabecera), `-z` (guarda en formato texto comprimido), o `-binary` (guarda en binario). Por ejemplo:

```
octave:3> save -z "train.txt.gz" X
```

El guardado de datos provoca pérdida de precisión, por omisión se guarda hasta el cuarto decimal. Para modificar esta precisión de guardado, se debe ejecutar previamente la función `save_precision(n)`, donde `n` es el número de cifras significativas que se guardarán en formato texto.

3.3. Visualización de dígitos

Podemos visualizar la imagen de la Fig. 1 que se corresponde con la fila 2 de la variable `X` teniendo en cuenta que es una imagen de 28 x 28 píxeles:

```
octave:4> x=X(2,:);   // te da los valores de x en las posiciones de 2 hasta la posición x
octave:5> xr=reshape(x,28,28);
octave:6> imshow((255-xr)',[])
```

También podemos visualizar las 20 primeras imágenes de la base de datos MNIST para hacernos una idea de la dificultad de esta tarea real:

```
octave:7> for i=1:20
> xr=reshape(X(i,:),28,28); imshow((255-xr)',[]); pause(1);
> end
```

4. Revisitando Octave

4.1. Funciones básicas en vectores y matrices

Como ya habéis estudiado en las prácticas de la asignatura de SIN, Octave aporta múltiples funciones para operar con vectores y matrices. En esta sección se revisan algunas de ellas que serán útiles para el proyecto de esta asignatura. Las funciones utilizadas para obtener las dimensiones de un vector o una matriz, además de la función `size`, existen las funciones `rows` y `columns` que devuelven el número de filas y columnas, respectivamente.

```
octave:8> rows(X)
ans =  60000
octave:9> columns(X)
ans =  784
```

Otra función de utilidad para estas prácticas es la función `unique` que nos permite obtener el conjunto de elementos diferentes de un vector eliminando duplicados. Por ejemplo, podemos aplicar la función `unique` al vector de etiquetas de clase de entrenamiento `xl`:

```
octave:10> labs = unique(xl)
labs =
   0
   1
   2
   3
   4
   5
   6
   7
   8
   9
```

Otra función muy útil para seleccionar un conjunto de filas o columnas en base a una condición es la función `find`. La función `find` devuelve los índices (posiciones) de aquellos elementos que cumplen la condición. Por ejemplo, se puede obtener los índices de las muestras que pertenecen a la clase del dígito 9 en el vector de etiquetas de clase de entrenamiento `xl`:

```
octave:11> ind9 = find(xl==9)
ind9 =
       5
      20
      23
      34
      44
      ...
```

Estos índices nos permiten extraer del conjunto de entrenamiento aquellas muestras (filas) que pertenecen a la clase del dígito 9:

```
octave:12> X9=X(ind9,:);
```

Recuerda como en Octave se puede indexar un subconjunto de filas (o columnas) de un vector o matriz mediante un vector.

En general, la funciones Octave se aplican por columnas, este es el caso de funciones como `sum` y `max`. Estas funciones aplicadas sobre un vector obtienen la suma y valor máximo del vector, pero cuando se aplican a matrices, devuelven la suma o valor máximo de cada columna:

```
octave:13> sum(X(:,10:15))
ans =

     0     0     0   126   470   216
```

```
octave:14> max(X(:,10:15))
ans =

     0     0     0   116   254   216
```

En este caso, aprovechamos este ejemplo para recordarte que Octave permite seleccionar rangos de columnas (o filas) mediante el operador `:`. Más detalles en el Apéndice A.2.

Adicionalmente, la función `max` también puede devolver el índice o índices del valor máximo (`argmaxv`):

```
octave:15> [maxv,argmaxv] = max(X(:,10:15))
maxv =

     0     0     0   116   254   216

argmaxv =

       1       1       1   12906   12906   12353
```

El comportamiento por defecto de estas funciones es operar por columnas (primera dimensión), comportamiento que se puede modificar proporcionando un segundo argumento que indica la dimensión sobre la que debe operar la función. Es decir, si queremos realizar una suma por filas (segunda dimensión) basta con:

```
octave:17> suma = sum(X(10:15,:),2)
suma =

   21904
   28548
   14250
   35761
   28443
   11622
```

En este ejemplo por simplicidad se ha seleccionado un subconjunto de filas (de la 10 a la 15) al cual aplicar la operación de suma.

La función `sort` por defecto sólo devuelve el vector (o matriz) ordenado de menor a mayor, pero se puede invocar para que devuelva adicionalmente el vector de índices de ordenación:

```
octave:21> [s,i] = sort(suma)
s =

   11622
   14250
```

```
   21904
   28443
   28548
   35761

i =

   6
   3
   1
   5
   2
   4
```

Si aplicamos el vector de ordenación `i` a las filas del vector `suma` obtenemos el vector ordenado `s`:

```
octave:22> suma(i,:)
ans =

   11622
   14250
   21904
   28443
   28548
   35761
```

Al igual que en todas las funciones Octave, se puede invocar la ayuda `help sort` para obtener información sobre la función correspondiente. En este caso, consulta como se podría cambiar el comportamiento por defecto de `sort` para realizar la ordenación de mayor a menor.

Además Octave dispone de otras muchas funciones básicas para operar con matrices, como son `det` para calcular el determinante de una matriz, `inv` para calcular la inversa de una matriz, `eig` que calcula valores y vectores propios, y la función `diag` que extrae la diagonal de una matriz o genera a partir de un vector una matriz diagonal. Finalmente, algunas funciones adicionales para generar matrices son `eye`, `ones` y `zeros` que devuelven la matriz identidad, todo unos y todo ceros, respectivamente.

4.2. Operadores de iteración y condicionales

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

```
octave:37> for c=labs'
```

```
> c
> end
c = 0
c =  1
c =  2
c =  3
c =  4
c =  5
c =  6
c =  7
c =  8
c =  9
```

Ejercicio 4.2.1 Utilizando las funciones que se han descrito, calcula el vector media de cada dígito y almacénalo en las columnas de una matriz `medias`.

Ejercicio 4.2.2 Realiza el mismo cálculo que en el ejercicio anterior pero haciendo uso de la función `mean`.

4.3. Funciones

Al igual que en otros lenguajes de programación, en Octave se pueden definir funciones de usuario. Como recomendación general, cada función debe estar implementada en un fichero `.m` independiente cuyo nombre coincida con el de la función. Los comentarios que aparecen al principio del fichero antes de la cabecera de la función se muestran como ayuda cuando se invoca el nombre de función precedido del comando `help`. Por ejemplo, la siguiente función `media` estaría incluida en un fichero `media.m`

```
%
% function [m] = media(X)
%
% La función media calcula a partir de X, donde las
% muestras están dispuestas por filas, su vector media
function [m] = media(X)
  m = sum(X)/rows(X);
end
```

Para poder utilizar esta función desde Octave, el directorio donde se encuentra su fichero `.m` asociado debe estar incluido en el `path`. Para ello puedes utilizar la función `addpath`, o este fichero `.m` se debe encontrar en el actual directorio de trabajo de Octave.

4.4. Scripts

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 `media.m`, podemos definir el siguiente script `script.m` que carga los datos de entrenamiento e invoca a la función `media` para mostrar las primeras 20 dimensiones del vector media resultante:

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

if (nargin!=1)
printf("Usage: script.m <trdata>\n")
exit(1);
end;

arg_list=argv();
trdata=arg_list{1};

load(trdata);

m = media(X);

m(1:20)
```

Como se puede observar, el número de argumentos del script está contenido en la variable `nargin` 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`.

Recuerda que **estos scripts se ejecutan desde el intérprete de comandos de** `bash` y que deben tener permisos de ejecución (`chmod +x script.m`).

5. Ejercicio: optimización de un clasificador lineal

Los siguiente ejercicios están pensados para poner en práctica técnicas que se utilizarán en el proyecto de la asignatura. Para estos ejercicios, se retoma la práctica del bloque 2 de SIN para plantear una optimización del código proporcionado en su momento.

5.1. Ejercicio: optimizando un clasificador lineal

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

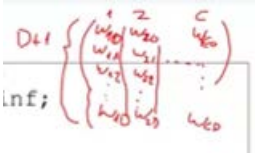
Este algoritmo estima un conjunto de pesos que minimiza el error de clasificación en el conjunto de entrenamiento. Asimismo, hay dos parámetros que controlan el comportamiento del algoritmo: el factor de aprendizaje α y el margen b.

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

w = conjunto de pesos previamente aprendidos, x = punto, linmach -> determina la clase que le asigna mayor valor a x

c = argmax $g_c(x)$

```
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
```



cstar guarda la clase más probable

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 5.1 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. Para comprobar tu versión matricial de la función `linmach.m` deberás modificar adecuadamente el script Octave `experiment.m`. Primeramente, entrena los vectores de pesos mediante Perceptron en un pequeño subconjunto (1 %) del conjunto de entrenamiento de MNIST, y después compara el error de clasificación entre la versión de SIN y la versión matricial que has implementado.

Nota: No debes utilizar operadores iterativos (`for`, `while`, etc.) para la implementación de la versión matricial de `linmach.m`, sino el producto matricial y la función `max`.

Ejercicio 5.2 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. Comprueba su correcto funcionamiento como has hecho en el anterior ejercicio.

Nota: No se deben utilizar operadores iterativos (`for`, `while`, etc.) para su implementación, sino los operadores lógicos y la función `sum` o la función `mean`.

Ejercicio 5.3 Como recordarás, el algoritmo Perceptron dispone principalmente de dos parámetros que pueden ser ajustados para minimizar el número de errores de clasificación. Estos parámetros son α, el factor de aprendizaje que controla la magnitud en que los pesos se modifican tras cada error de clasificación en el conjunto de entrenamiento, y b, el margen por el cual la función discriminante de la clase correcta debe superar al resto de clases.

En la asignatura de SIN, el ajuste de estos parámetros se realizaba a la vista del error de clasificación en el conjunto de test. Sin embargo, lo habitual es dedicar un pequeño subconjunto extraído del conjunto de entrenamiento para el ajuste de parámetros. Dicho subconjunto se conoce como conjunto de validación o *development*. Una vez realizada la exploración de los valores de los parámetros en el conjunto de validación, los parámetros que minimizan el error de clasificación en el conjunto de validación se utilizan para estimar un clasificador con todos los datos de entrenamiento y calcular el error de

clasificación en el conjunto de test, que es la cifra de error que se proporciona como tasa de error en la tarea MNIST.

Haz una copia de tu script `experiment.m` como `evaluation.m`, y modifica tu script `experiment.m` para que no necesite el conjunto de test, pero en su lugar utilice un porcentaje del conjunto de entrenamiento como conjunto de validación:

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

if (nargin!=7)
printf("Usage: experiment.m <trdata> <trlabels> <as> <bs> <maxK> <%%tr> <%%dv>\n")
exit(1);
end;

arg_list=argv();
trdata=arg_list{1};
trlabs=arg_list{2};
as=str2num(arg_list{3});
bs=str2num(arg_list{4});
K=str2num(arg_list{5});
trper=str2num(arg_list{6});
dvper=str2num(arg_list{7});

[...]
```

Por simplicidad, selecciona la parte final del conjunto de entrenamiento como conjunto de validación.

Concretamente, realiza un experimento que utilice del conjunto de entrenamiento un 5 % tanto para entrenamiento como para validación y explore los valores diversos valores de $\alpha = [0.1\ 0.01\ 0.001]$ y $b = [1.0\ 10.0\ 100.0]$. A la vista de los resultados entrena un clasificador con los parámetros que proporcionan los mejores resultados y evalúalo sobre el conjunto de test. Compara los resultados obtenidos con los resultados reportados en la tarea MNIST para el clasificador *linear classifier (1-layer NN)*.

Ejercicio 5.4 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.

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 validación 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,Edv,k]=perceptron(tr,dv,b,a,K,iw)
```

donde `tr` y `dv` son los conjuntos de entrenamiento y validación, respectivamente, y `Etr` y `Edv` son vectores que almacenan en cada iteración el número de errores de clasificación en el conjunto de entrenamiento y validación, respectivamente.

La representación gráfica del porcentaje de error de clasificación en el conjunto de entrenamiento (5 %) y validación (5 %) en función de la iteración te permitirá observar la convergencia del algoritmo visualmente. Realiza un experimento con $\alpha = 0.001$, $b = 10.0$ y $K = 200$. Puedes realizar la representación gráfica con los siguientes comandos tras invocar a la función `perceptron`:

```
plot([1:k],Etr/Ntr*100,"--xk",[1:k],Edv/Ndv*100,"-+k");
fn=sprintf("perceptron.a%.1e.b%.1e.eps",a,b);
print(fn,"-deps");
```

siendo `Ntr` y `Nte`, el número de muestras en el conjunto de entrenamiento y validación, respectivamente. La representación gráfica que deberías obtener se muestra en la Figura 2.

Ejercicio 5.5 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 del bloque 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 alcance 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 para actualizar los vectores de pesos. Implementa una versión mini-batch del algoritmo Perceptron:

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

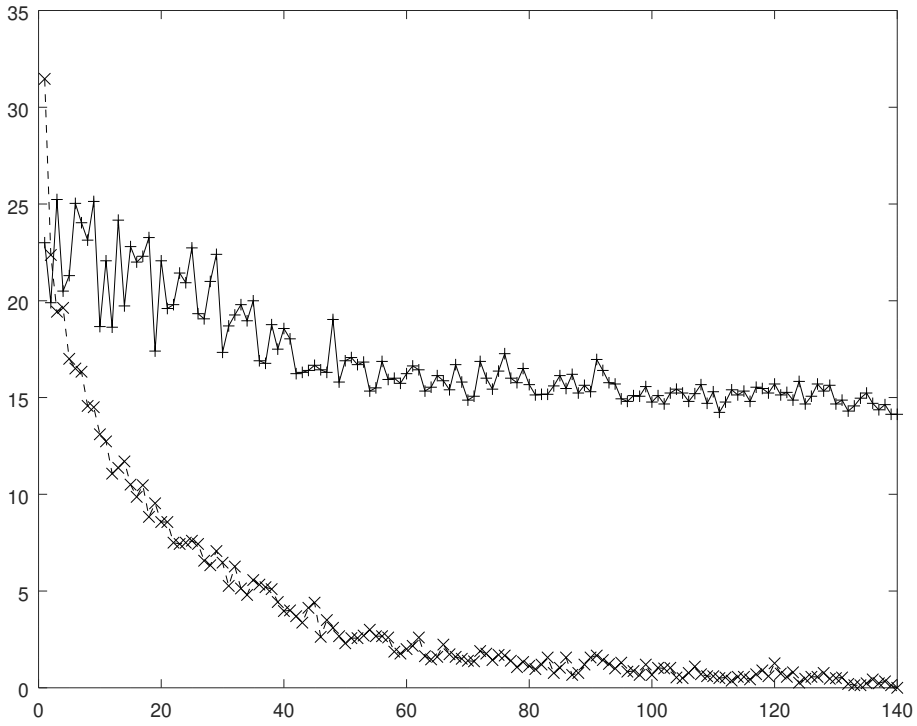


Figura 2: Evolución del porcentaje de error de clasificación (eje y) en los conjuntos de entrenamiento (abajo) y validación (arriba) en función del número de iteraciones (eje x) del algoritmo Perceptron hasta convergencia.

A. Tutorial básico sobre Octave

A.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
```

A.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
```