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

Práctica 0. Introducción a Octave *y* MATLAB

Enrique Vidal, Jorge Civera, Francisco Casacuberta
Departamento de Sistemas Informáticos y Computación
Universitat Politècnica de València

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 como en otras asignaturas cursadas. El alumno deberá refrescar sus conocimientos y habilidades sobre dichas herramientas. Además, 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 una sesión.

2. Introducción

Varias de las técnicas empleadas en Aprendizaje Automático y Reconocimiento de Formas (RF) emplean cálculos vectoriales y matriciales, como por ejemplo en los algoritmos Perceptron o Widrow-Hoff. Por tanto, una herramienta que implemente de forma sencilla estos cálculos matriciales puede simplificar notáblemente la implementación de sistemas de RF.

Una de las herramientas comerciales más potentes en cálculo matricial es MATLAB. Asimismo 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. Entre otras, posee capacidades de cálculo numérico para solucionar problemas lineales y no lineales. También dispone de herramientas gráficas para visualizar datos y resultados. Se puede usar de forma interactiva y/o programada mediante *scripts* en un lenguaje interpretado. La sintaxis y semántica de Octave es prácticamente idéntica a MATLAB, lo que hace que los programas sean fácilmente portables entre ambas plataformas.

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 la web mencionada).

3. El algoritmo Widrow-Hoff

Como habéis visto en teoría, el algoritmo Widrow-Hoff, aprende un vector de pesos $\boldsymbol{\theta} \in \mathbb{R}^D$ que minimiza el error cuadrático:

$$q_S(\boldsymbol{\theta}) = \frac{1}{2}\sum_{n=1}^{N}(y_n - \boldsymbol{\theta}^t \mathbf{x}_n)^2. \tag{1}$$

En esta sesión introducimos la versión matricial de la Eq. 1, ya que Octave está optimizado para trabajar con operaciones matriciales:

$$q_S(\boldsymbol{\theta}) = \frac{1}{2}\|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|^2 \tag{2}$$

donde $\mathbf{X}$ es la matriz de N muestras por D dimensiones cuya n-ésima fila es $\mathbf{x}_n$, e $\mathbf{y}$ es un vector columna N-dimensional cuya n-ésima fila es y_n. La derivada de la Eq. 2 que utilizaremos en el algoritmo Widrow-Hoff es:

$$\boldsymbol{\nabla} q_S(\boldsymbol{\theta}) = -\mathbf{X}^t\,(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \tag{3}$$

Por tanto, la versión matricial del algoritmo Widrow-Hoff en base a la Eq. 3 es:

$$\begin{aligned} \boldsymbol{\theta}(1) &= \text{arbitrario} \\ \boldsymbol{\theta}(k+1) &= \boldsymbol{\theta}(k) + \rho_k\, \mathbf{X}^t\,(\mathbf{y} - \mathbf{X}\boldsymbol{\theta}(k)) \end{aligned} \tag{4}$$

Ejercicio 1. El fichero `synthdata.n100K.d10.mat.gz` del directorio de datos de la práctica 0 contiene una muestra de $N = 100{,}000$ puntos para el aprendizaje de una función de regresión lineal. Las primeras $D = 10$ columnas son los datos de entrada $\mathbf{X}$ y la columna 11 son los valores de salida $\mathbf{y}$ a predecir.

a) Implementa una función en Octave que obtenga el vector de pesos utilizando el algoritmo Widrow-Hoff de la Eq. 4:

```
function [w]=wh(data)
% Number of samples and dimensions
[N,D]=size(data)
...
```

b) Comprueba la aproximación de los pesos calculados a los pesos reales consultando el fichero `weights.n100K.d10.mat.gz` que contiene el vector de pesos utilizado para generar los valores de salida $\mathbf{y}$, así como calculando el error cuadrático medio entre la estimación $\mathbf{X}\boldsymbol{\theta}(k)$ e $\mathbf{y}$.

En cada iteración de la Eq. 4 se computa un producto de $\mathbf{X}^t(\mathbf{y} - \mathbf{X}\boldsymbol{\theta})$ con un coste asintótico de $O(N \cdot D)$. Sin embargo, si desarrollamos el producto obtenemos la expresión $\mathbf{X}^t\mathbf{y} - \mathbf{X}^t\mathbf{X}\boldsymbol{\theta}$, cuyos estadísticos suficientes $S = \mathbf{X}^t\mathbf{X}$ y $T = \mathbf{X}^t\mathbf{y}$ pueden ser calculados previamente, obteniendo una versión más eficiente (si $N > D$) de Widrow-Hoff:

$$\begin{aligned} \boldsymbol{\theta}(1) &= \text{arbitrario} \\ \boldsymbol{\theta}(k+1) &= \boldsymbol{\theta}(k) + \rho_k \ (T - S\boldsymbol{\theta}) \end{aligned} \tag{5}$$

cuyo coste asintótico es $O(D \cdot D)$.

Ejercicio 2. Implementa esta versión optimizada del algoritmo Widrow-Hoff y compara su coste temporal en el fichero `synthdata.n100K.d10.mat.gz`. Para ello, te serán de utilidad las funciones `clock` y `etime` de Octave:

```
t1=clock();
...
t2=clock();
elapsed_time = etime(t2,t1)
```

Por último, podemos obtener una solución directa del cálculo del vector de pesos igualando a cero la expresión de la derivada de la función $q_S(\boldsymbol{\theta})$ que se muestra en la Eq. 3:

$$\mathbf{X}^t\mathbf{X}\boldsymbol{\theta} = \mathbf{X}^t\mathbf{y} \tag{6}$$

Dado que $\mathbf{X}^t\mathbf{X}$ es una matriz cuadrada y habitualmente invertible, se puede despejar $\boldsymbol{\theta}$ de manera única como:

$$\boldsymbol{\theta} = (\mathbf{X}^t\mathbf{X})^{-1}\mathbf{X}^t\mathbf{y} = \mathbf{X}^\dagger\mathbf{y} \tag{7}$$

donde $\mathbf{X}^\dagger$ es la pseudoinversa de $\mathbf{X}$. Octave proporciona la función `pinv(m)` que calcula la pseudoinversa (si existe) de la matriz `m`.

Ejercicio 3. Implementa la solución directa del cálculo del vector de pesos y compárala con la versión eficiente del algoritmo Widrow-Hoff en términos temporales y en error cuadrático medio de los pesos calculados.

4. Ejercicio adicional

En el directorio de datos de la práctica 0 se encuentra el fichero `data01` que contiene datos bidimensionales de test de las clases "0" y "1". Se dispone de un clasificador lineal en dos clases definido como:

$$f(x_1, x_2) = \begin{cases} 0 & \text{si } x_1 + x_2 - 1{,}6 < 0 \\ 1 & \text{si no} \end{cases}$$

Estima el error de clasificación de dicho clasificador y la correspondiente fluctuación esperada de dicho error (llamada *"intervalo de confianza"* con un nivel de confianza o del 95 %. Recuerda que el intervalo de confianza al 95 % de una probabilidad de error empírica $\hat{p}_e$, estimada sobre un conjunto de N datos se puede calcular como

$$\hat{p}_e \pm 1.96 \sqrt{\frac{\hat{p}_e(1-\hat{p}_e)}{N}}$$

Opcionalmente se puede estimar este intervalo de confianza mediante *boostrapping*. Para ello se extraen M conjuntos de N datos muestreando con reemplazamiento el conjunto de datos original. Para cada uno de estos M conjuntos calculamos la probabilidad de error $\hat{p}_e$, y los ordenamos de mayor a menor. Finalmente, el intervalo de confianza queda definido por los valores que se encuentran en los extremos de una ventana centrada que engloba al 95 % de los valores de probabilidad de error estimados sobre los M conjuntos.

Bibliografía

Richard O. Duda, Peter E. Hart, and David G. Stork. 2000. Pattern Classification (2nd Edition). Wiley-Interscience, New York, NY, USA. Páginas 240–246.

A. 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**.

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 Octave da valor a la variable predefinida `ans`, y la muestra. Pero los resultados también pueden 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, aunque también puede emplearse la función `disp`, que muestra el contenido de la variable omitiendo su nombre:

```
octave:3> b
b =  0.83138

octave:4> ans
ans =  0.99033

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 de cada operación añadiendo (`;`) al final de la operación:

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

A.2. Operadores básicos en vectores y matrices

Para la notación matricial en Octave se usan los corchetes (`[ ]`); en su interior, las filas se separan por punto y coma (`;`) y las columnas por espacios en blanco ( ) o por comas (`,`). 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
```

Los operadores de comparación (>, <, >=, <=, ==, !=) se pueden aplicar "elemento a elemento". Como resultado se obtiene 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
```

Esos operadores se pueden emplear en la comparación de vectores y matrices de dimensiones congruentes. La matriz binaria resultante contiene los resultados de las comparaciones de los pares de elementos en la misma posición en ambas estructuras.

Para tranponer una matriz o vector se usa el operador de transposición (’):

```
octave:19> 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, mientras que para una matriz se espera una fila o secuencia de filas seguida de una columna o secuencia de columnas:

```
octave:20> v1(2)
ans =  3

octave:21> v1([2 3])
ans =
   3  -5

octave:22> v2(2)
ans =  2

octave:23> 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:24> m3([1 3], [1 4])
ans =
    1    4
    9   12
```

Para indicar todas las filas o columnas, se puede emplear (`:`):

```
octave:25> m3(:,[1 3])
ans =
    1    3
    5    7
    9   11
```

Los rangos se denotan como (`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, que por omisión es 1.

```
octave:26> m3([1 3],1:3)
ans =
    1    2    3
    9   10   11

octave:27> m3([1 3],1:2:4)
ans =
    1    3
    9   11
```

Para indicar el último índice de una dimensión se puede emplear (`end`):

```
octave:28> m3([1 3],end)
ans =
    4
   12
```

A.3. Funciones básicas en vectores y matrices

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

- `size(m)`: devuelve número de filas y columnas de la matriz (en el caso de un vector, una de las dimensiones tendrá tamaño 1)
- `eye(f,c)`, `ones(f,c)`, `zeros(f,c)`: dan la matriz identidad, todo unos y nula, respectivamente, de tamaño `f`×`c`; si se pone un solo número, da la matriz cuadrada correspondiente
- `sum(v)`, `sum(m)`: da la suma de los elementos del vector o matriz; en el caso de la 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 para columnas, 2 para filas).
- `max(v)`, `max(m)`: indica el valor máximo del vector o el vector con los máximos por columna de la matriz; si se pide que devuelva dos resultados (`[r1,r2]=max(v)`, `[r1,r2]=max(m)`), el primer resultado almacena los valores y el segundo su posición
- `det(m)`: determinante de `m`
- `eig(m)`: vector de valores propios de `m` o su versión matricial diagonal
- `diag(v)`: crear matriz diagonal con los valores de `v`
- `inv(m)`: inversa de la matriz `m` si esta es no singular
- `trace(m)`: traza de la matriz `m`

- `sort(v)`: vector ordenado con los valores del vector `v`
- `repmat(m,f,c)`: crea una matriz de `f`×`c` bloques de `m`; si `c` se omite, será de `f`×`f`
- `find(v)`, `find(m)`: se le pasa un vector o matriz e indica aquellos elementos que no son cero (índices absolutos empezando en 1 y haciendo el recorrido por cada columna y por filas ascendentes); si se le piden dos resultados (`[r1,r2]=find(v)`, `[r1,r2]=find(m)`) el primer resultado almacena fila y el segundo columna; se puede aplicar sobre resultados de operaciones lógicas a fin de verificar elementos de la matriz o vector que cumplen una condición. Por ejemplo, mediante la función (`rem`) que obtiene el resto del primer operador dividido por el segundo, se pueden obtener las posiciones de los elementos pares:

```
octave:29> [r,c]=find(rem(m3,2)==0)
r =
   1
   2
   3
   1
   2
   3

c =
   2
   2
   2
   4
   4
   4
```

A.4. Carga y salvado 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 de y salvar en ficheros. El salvado se hace mediante la orden `save`:

```
octave:30> save "m3.dat" m3
```

El fichero tiene el siguiente contenido:

```
# 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` cuyo contenido es:

```
# 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
```

Su carga definirá la matriz (`A`) como:

```
octave:31> load "maux.dat"

octave:32> A
A =
    1    2   -3
    5   -6    7
   -9   10   11
   -5    2   -1
```

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

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

En ocasiones, el salvado de datos puede provocar pérdida de precisión, pues por omisión se salva hasta el cuarto decimal. Para modificar esta precisión de salvado, se puede emplear `save_precision(n)`, donde `n` es el número de posiciones decimales (incluyendo el `.`) que se grabarán.

A.5. Funciones Octave

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

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

Porejemplo:

```
octave:34> function [m1,m2] = addsub(ma,mb)
> m1=ma+mb
> m2=ma-mb
> end

octave:35> mat1=[1,2;3,4]
mat1 =
   1   2
   3   4

octave:36> mat2=[-1,2;3,-4]
mat2 =
  -1   2
   3  -4

octave:37> addsub(mat1,mat2)
m1 =
   0   4
   6   0

m2 =
   2   0
   0   8

ans =
   0   4
   6   0

octave:38> [mr1,mr2]=addsub(mat1,mat2)
m1 =
   0   4
   6   0

m2 =
   2   0
   0   8
```

```
mr1 =
   0   4
   6   0

mr2 =
   2   0
   0   8
```

Es habitual definir las funciones en ficheros de código Octave cuyo nombre debe ser el mismo que la función con el sufijo "`.m`" (en nuestro ejemplo sería `addsub.m`). Estos ficheros deben situarse en el mismo directorio en el que se ejecuta Octave. De esa forma, se puede acceder a las funciones sin tener que definirlas cada vez.

A.6. Programas Octave

Octave se puede usar de forma *no interactiva* mediante *scripts* que son interpretados por Octave. En estos *scripts* o *"programas Octave"* 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, desde cualquier terminal podemos crear (con algún editor) el fichero `test.m` con el siguiente contenido:

```
#!/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)
```

Si desde la línea de órdenes le damos permisos de ejecución (`chmod +x test.m`), podremos ejecutarlo como cualquier programa ejecutable o *shell script*:

```
$_ ./test.m
a =
   1   2   3
   4   5   6
   7   8   9

b =
   9   8   7
   6   5   4
   3   2   1

c =
   10   10   10
   10   10   10
   10   10   10
```

```
m1 =
   10   10   10
   10   10   10
   10   10   10

m2 =
  -8  -6  -4
  -2   0   2
   4   6   8

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 el sugijo "`.m`"). En este caso, se pueden poner argumentos en la línea de órdenes y puede usarse la variable `nargin` (número de argumentos) y la función `argv()` (da una lista de los valores alfanuméricos de los argumentos recibidos). Estos argumentos están en formato de cadena de caracteres; por tanto los valores numéricos deben convertirse a formato numérico, por ejemplo empleando la función `str2num(c)`.

B. Ejercicios propuestos

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 dimensión 4×4 mediante 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 salvado 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 grabe su traspuesta en el fichero `data_trans`.