APR (E.T.S. de Ingeniería Informática)
Curso 2019-2020

Práctica 1. Mixturas de gaussianas

Jorge Civera, Francisco Casacuberta, Enrique Vidal
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 previamente el alumnado ha adquirido experiencia en el uso de *octave*, *shell scripts* y *gnuplot*, tanto en la asignatura de Sistemas Inteligentes como en la asignatura de Percepción. Si no es el caso, el alumnado puede refrescar sus conocimientos y habilidades sobre dichas herramientas recurriendo al tutorial incluido en el anexo de este documento. Este tutorial es el mismo que se utilizó en la asignatura de Percepción.

Además, debería haber leído de forma detallada la totalidad de este boletín, para poder centrarse en profundidad en el trabajo a desarrollar en el laboratorio, el cual durará tres sesiones.

2. Tarea de clasificación: MNIST

2.1. Introducción

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 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`)

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

```
  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   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  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.

En la Figura 1 se muestra una representación 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.

2.2. 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
```

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
```

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.

2.3. 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:3> x=X(2,:);
octave:4> xr=reshape(x,28,28);
octave:5> 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:6> for i=1:20
> xr=reshape(X(i,:),28,28); imshow((255-xr)',[]); pause(1);
> end
```

3. Clasificador multinomial

En la asignatura de Percepción se implementó un clasificador multinomial para un tarea de clasificación de dos clases (*ham* y *spam*). Por esta razón y dado que los datos de entrada son enteros positivos, antes de trabajar con clasificadores gaussianos, realizaremos un primer experimento con un clasificador multinomial con el que se está más familiarizado.

La función Octave que implementa la estimación de parámetros multinomiales, suavizado por Laplace, y clasificación y estimación del error tanto en el conjunto de entrenamiento como de test, está disponible en PoliformaT en el fichero `multinomial.m`. Se puede descargar en tu directorio local y observar que es una generalización a C clases de la implementación que se realizó en Percepción. Se puede invocar esta función desde Octave cargando previamente los ficheros de datos:

```
octave:7> load train-labels-idx1-ubyte.mat.gz
octave:8> load t10k-images-idx3-ubyte.mat.gz
octave:9> load t10k-labels-idx1-ubyte.mat.gz
octave:10> multinomial(X,xl,Y,yl,[1.0e-01 1.0e-02 1.0e-05 1.0e-10])

    eps tr-err te-err
------- ------ ------
1.0e-01 36.058 34.860
1.0e-02 29.123 27.770
1.0e-05 17.705 16.440
1.0e-10 17.458 16.320
```

Asimismo, en PoliformaT, se ha dejado un script Octave `multinomial-exp.m` ejecutable desde terminal que recibe como entrada los ficheros de entrenamiento y test, y un vector de valores de epsilons de suavizado, y devuelve las tasas de error:

```
$ ./multinomial-exp.m train-images-idx3-ubyte.mat.gz \
train-labels-idx1-ubyte.mat.gz t10k-images-idx3-ubyte.mat.gz \
t10k-labels-idx1-ubyte.mat.gz "[1.0e-01 1.0e-02 1.0e-05 1.0e-10]"
    eps tr-err te-err
------- ------ ------
1.0e-01 36.058 34.860
1.0e-02 29.123 27.770
1.0e-05 17.705 16.440
1.0e-10 17.458 16.320
```

Como se puede observar, la primera columna es el valor de epsilon utilizado en el suavizado de Laplace, mientras que la segunda y tercera columnas son la tasas de error de clasificación en entrenamiento y test, respectivamente. Si comparamos el error en test que obtiene nuestro clasificador multinomial con otros clasificadores lineales[2] del

[2]Recuerda que el clasificador multinomial es lineal, es decir, define una frontera de decisión lineal.

principio de la tabla de la web de MNIST, veremos que está alejado de la tasa de error que se consigue con *linear classifier (1-layer NN)* sin preproceso, que es la tasa de error más elevada que se reporta. Más abajo en esta misma tabla podemos observar como los clasificadores basados en k vecinos más cercanos[3] obtienen mejores tasas de error.

Sin embargo, nos vamos a centrar en la sección de la tabla referente a *Non-Linear Classifiers*, y más concretamente en el resultado de la fila *40 PCA + quadratic classifier*. Como se estudió en Percepción, el clasificador gaussiano es un clasificador cuadrático, es decir, define una frontera de decisión que es una curva cuadrática y más compleja que la lineal. Por ello y dado que ya conoces la teoría del clasificador gaussiano, la siguiente sección la dedicaremos a su implementación y a los problemas prácticos asociados. Seguidamente y después de ver el clasificador gaussiano, dedicaremos la última sección al clasificador basado en mixturas de gaussianas.

4. Clasificador gaussiano

Antes de abordar la implementación del clasificador gaussiano es necesario recordar que, tanto el clasificador multinomial como el clasificador gaussiano, son instanciaciones del clasificador Bayes:

$$\begin{aligned} c^*(x) &= \operatorname*{argmax}_{c=1,\dots,C} P(c \mid x) \\ &= \operatorname*{argmax}_{c=1,\dots,C} P(c)\, p(x \mid c) \\ &= \operatorname*{argmax}_{c=1,\dots,C} \log P(c) + \log p(x \mid c) \end{aligned}$$

donde la f.d. condicional $p(x \mid c)$ se modeliza mediante una distribución concreta. En esta sección, la f.d. condicional $p(x \mid c)$ será una distribución gaussiana D-dimensional:

$$p(\mathbf{x} \mid c) \sim \mathcal{N}_D(\boldsymbol{\mu}_c, \Sigma_c), \quad c = 1, \dots, C$$

Por tanto:

$$\begin{aligned} c^*(\boldsymbol{x}) &= \operatorname*{argmax}_{c=1,\dots,C} \log P(c) + \log p(\boldsymbol{x} \mid c) \\ &= \operatorname*{argmax}_{c=1,\dots,C} \log P(c) - \frac{D}{2}\log 2\pi - \frac{1}{2}\log |\Sigma_c| - \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_c)^t \Sigma_c^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_c) \\ &= \operatorname*{argmax}_{c=1,\dots,C} \log P(c) - \frac{1}{2}\log |\Sigma_c| - \frac{1}{2}\boldsymbol{x}^t\Sigma_c^{-1}\boldsymbol{x} + \boldsymbol{\mu}_c^t\Sigma_c^{-1}\boldsymbol{x} - \frac{1}{2}\boldsymbol{\mu}_c^t\Sigma_c^{-1}\boldsymbol{\mu}_c \\ &= \operatorname*{argmax}_{c=1,\dots,C} \log P(c) - \frac{1}{2}\boldsymbol{x}^t\Sigma_c^{-1}\boldsymbol{x} + \boldsymbol{\mu}_c^t\Sigma_c^{-1}\boldsymbol{x} + \left(-\frac{1}{2}\log |\Sigma_c| - \frac{1}{2}\boldsymbol{\mu}_c^t\Sigma_c^{-1}\boldsymbol{\mu}_c\right) \end{aligned}$$

[3]Recuerda que son clasificadores lineales a trozos, pero que pueden definir fronteras de decisión arbitráreamente complejas con suficientes muestras de entrenamiento. En MNIST tenemos 6000 muestras por clase.

Expresado en términos de función discriminante cuadrática con $\boldsymbol{x}$:

$$c^*(\boldsymbol{x}) = \underset{c=1,\ldots,C}{\operatorname{argmax}}\ g_c(\boldsymbol{x}) = \underset{c=1,\ldots,C}{\operatorname{argmax}} \log P(c) + \boldsymbol{x}^t\, W_c\, \boldsymbol{x} + \boldsymbol{w}_c^t\, \boldsymbol{x} + w_{c0}$$

donde

$$W_c = -\frac{1}{2}\Sigma_c^{-1} \tag{1}$$

$$\boldsymbol{w}_c = \Sigma_c^{-1}\boldsymbol{\mu}_c \tag{2}$$

$$w_{c0} = -\frac{1}{2}\log\,|\Sigma_c| - \frac{1}{2}\boldsymbol{\mu}_c^t\Sigma_c^{-1}\boldsymbol{\mu}_c \tag{3}$$

Para simplificar la implementación hemos dejado el término $\log P(c)$ fuera de la componente constante de la probabilidad condicional gaussiana.

La estimación máximo verosímil de los parámetros del clasificador gaussiano son ampliamente conocidos:

$$\begin{aligned}
\hat{P}(c) &= \frac{N_c}{N} \\
\hat{\boldsymbol{\mu}}_c &= \frac{1}{N_c}\sum_{n:c_n=c} \boldsymbol{x}_n \\
\hat{\Sigma}_c &= \frac{1}{N_c}\sum_{n\,:\,c_n=c} (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_c)(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_c)^t
\end{aligned}$$

A continuación se muestra una implementación en Octave del clasificador gaussiano `gaussian.m` disponible en PoliformaT. Primero, se explica la parte del código correspondiente a la estimación de parámetros:

```
 1 function [teerr] = gaussian(X,xl,Y,yl)
 2
 3 classes=unique(xl);
 4 N=rows(X);
 5 M=rows(Y);
 6 D=columns(X);
 7
 8 for c=classes'
 9   ic=find(c==classes);
10   idx=find(xl==c);
11   Xc=X(idx,:);
12   Nc=rows(Xc);
13   pc(ic)=Nc/N;
14   muc=sum(Xc)/Nc;
```

```
15   mu(:,ic)=muc';
16   sigma{ic}=((Xc-muc)'*(Xc-muc))/Nc;
17   % Smoothing with identity matrix
18   sigma{ic}=alpha*sigma{ic}+(1-alpha)*eye(D);
19 end
```

Las líneas 3-6 obtienen el vector de etiquetas de clases `[0 1 ... 9]`, el número de muestras de entrenamiento y test, y la dimensionalidad de los datos.

A continuación sigue el bucle (líneas 10-19) donde se estimará los parámetros de cada clase, teniendo en cuenta que `ic` es el índice (o posición) de la clase `c` dentro del vector `classes` y `idx` es el vector de índices a muestras de la clase `c` sobre las filas de la matriz `X`. Recuerda que `ic` es un índice que tomará valores en `[1 2 ... 10]`.

En las líneas 13 y 14 se estima la probabilidad a priori y media de la clase `c`. El vector de probabilidades a priori `pc` será un vector fila, y la matriz de medias `mu` dispone la media de cada clase como un vector columna.

En la línea 16 se estima la matriz de covarianzas de la clase `c` y se almacena en la posición `ic` de un vector de celdas. Estos vectores de celdas permiten almacenar tipos diferentes en el mismo vector, pero en nuestro caso nos será útil para crear un vector de matrices de covarianzas e indexarlo fácilmente.

Además, la línea 18 implementa el suavizado *flat smoothing* con la matriz identidad:

$$\tilde{\Sigma}_c = \alpha \cdot \hat{\Sigma}_c + (1 - \alpha) \cdot I$$

Seguidamente, se muestra el código que estima la función discriminante de cada clase, la clasificación y la estimación del error.

```
20 for c=classes'
21   ic=find(c==classes);
22   gte(:,ic)=log(pc(ic))+compute_pxGc(mu(:,ic),sigma{ic},Y);
23 end
24
25 [~,idy]=max(gte');
26 teerr=mean(classes(idy)!=yl)*100;
```

Las líneas 20-23 básicamente calculan $\log P(c) + p(\boldsymbol{x} \mid c)$ siendo la función auxiliar `compute_pxGc` la que estima para el conjunto de test, la log probabilidad condicional modelizada mediante una distribución gaussiana. En la línea 25 obtenemos el índice de la clase `[1 2 ... 10]` cuyo valor de su función discriminante es máximo para cada muestra. Finalmente, la línea 26 es el cálculo de la probabilidad de error empírica, es decir, el número de errores promedio sobre el conjunto de test.

La función auxiliar `compute_pxGc` (ver abajo) calcula la log probabilidad condicional para la gaussiana con media `mu` y matriz de covarianzas `sigma` de cada muestra en `X`. La línea 28 se corresponde con el término cuadrático que pre y posmultiplica `X` por la Ec. 1, la línea 29 es el término lineal, que multiplica `X` por la Ec. 2, y las líneas 30 y 31 son el término constante

```
27 function [pxGc] = compute_pxGc(mu,sigma,X)
28   qua=-0.5*sum((X*pinv(sigma)).*X,2);
29   lin=X*(mu'*pinv(sigma))';
30   cons=-0.5*logdet(sigma);
31   cons=cons-0.5*mu'*pinv(sigma)*mu;
32   pxGc=qua+lin+cons;
33 end
```

Como puedes observar, para calcular la inversa de la matriz de covarianzas utilizamos la función pseudoinversa `pinv` en lugar de la inversa convencional `inv`. Esto se debe a que no se puede calcular la inversa de una matriz cuyo rango no es completo, pero sí la pseudoinversa. Es decir, nuestra matriz de covarianzas para MNIST es singular, y por tanto tiene filas (o columnas) que son linealmente dependientes unas de otras. Puedes comprobarlo rápidamente ejecutando `rank(cov(X,1))` y viendo que el rango es inferior a la dimensionalidad de los datos $D = 784$. Esto se debe mayormente a que tenemos muchas dimensiones que son siempre cero, al ser parte del fondo sobre el que está centrado el dígito.

Como consecuencia de que la matriz de covarianzas sea singular, su determinante, que se calcula en la línea 30, será cero y su logaritmo `-Inf` imposibilitando el cálculo de esta log probabilidad. Es por ello que la función `logdet` reemplaza el logaritmo de cero por el logaritmo de la constante más pequeña representable en Octave `eps = 2.2204e-16`. Además, podrás comprobar que la función `logdet` no calcula `log(det(sigma))` directamente, sino que para evitar valores excesivamente grandes (`Inf`) del determinante, este cálculo se realiza de forma robusta como la suma de logaritmos de los valores propios de la matriz de covarianzas[4].

Al igual que el clasificador multinomial, el clasificador gaussiano se puede invocar directamente desde Octave:

```
octave:11> gaussian(X,xl,Y,yl,1.0)
ans =  14.280
```

Se puede observar como el clasificador gaussiano mejora ligeramente la tasa de error del clasificador, pero queda lejos de la tasa de error que se puede conseguir combinando PCA con un clasificador cuadrático, en nuestro caso sería un clasificador gaussiano.

Ejercicio 4.1 Realiza un experimento para evaluar el error del clasificador gaussiano (sin suavizado, es decir, $\alpha = 1{,}0$) en función del número de componentes PCA a las cuales se proyectan los datos originales. Representa gráficamente los resultados obtenidos en forma de una curva.

Ejercicio 4.2 A la vista de los resultados obtenidos en el ejercicio anterior, repite el experimento anterior ajustando el factor de suavizado. Añade a la gráfica del ejercicio

[4] https://www.adelaide.edu.au/mathslearning/play/seminars/evalue-magic-tricks-handout.pdf

anterior una curva de error de clasificación en función del número de componentes PCA por cada valor α utilizado. Compara los resultados obtenidos con los reportados en la tarea MNIST.

5. Clasificador basado en mixturas de gaussianas

Las mixturas de distribuciones de probabilidad, en nuestro caso condicionada a la clase $p(\mathbf{x} \mid c)$, se desarrollan introduciendo una variable oculta k que indica la componente de la mixtura que genera la muestra $\mathbf{x}$

$$p(\mathbf{x} \mid c) = \sum_{k=1}^{K} p(\mathbf{x}, k \mid c) = \sum_{k=1}^{K} p(k \mid c)\, p(\mathbf{x} \mid k, c) \tag{4}$$

donde $p(k \mid c)$ es la probabilidad a priori de la componente condicionada a la clase, y $p(\mathbf{x} \mid k, c)$ es una distribución de probabilidad condicionada no sólo a la clase, sino también a la componente dentro de esa clase. En nuestro caso $p(\mathbf{x} \mid k, c)$ está modelizada mediante una distribución gaussiana

$$p(\mathbf{x} \mid k, c) \sim \mathcal{N}_D(\boldsymbol{\mu}_{ck}, \Sigma_{ck}), \quad c = 1, \ldots, C \quad k = 1, \ldots, K$$

pero podría haber sido modelizada mediante una distribución multinomial o cualquier otra.

Observad como la componente de cada clase tendrá su propia media y matriz de covarianzas. El conjunto de parámetros de este modelo de mixturas es:

$$\boldsymbol{\theta} = (P_1, \cdots, P_C, \boldsymbol{\theta}_1, \cdots \boldsymbol{\theta}_C)$$

donde

$$\boldsymbol{\theta}_c = (p_{c1}, \cdots, p_{cK}, \mu_{c1} \cdots \mu_{cK}, \Sigma_{c1}, \cdots, \Sigma_{cK})$$

La estimación de estos parámetros sería trivial si para cada muestra supiéramos que componente la generó (z_n):

$$\hat{P}(c) = \frac{N_c}{N} \tag{5}$$

$$\hat{p}_{ck} = \frac{\sum\limits_{n\,:\,c_n=c \wedge z_n=k} 1}{N_c} = \frac{N_{ck}}{N_c} \tag{6}$$

$$\hat{\boldsymbol{\mu}}_{ck} = \frac{1}{N_{ck}} \sum_{n\,:\,c_n=c \wedge z_n=k} \boldsymbol{x}_n \tag{7}$$

$$\hat{\Sigma}_{ck} = \frac{1}{N_{ck}} \sum_{n\,:\,c_n=c \wedge z_n=k} (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ck})(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ck})^t \tag{8}$$

donde N_{ck} es el número de muestras de la componente k de la clase c.

Sin embargo, como has estudiado en teoría, el algoritmo EM de manera iterativa nos permite estimar los parámetros de un modelo (paso M) donde hay variables ocultas, calculando una distribución de probabilidad sobre las mismas (paso E).

El paso E en la iteración t de un modelo de mixturas dada la estimación actual de los parámetros $\boldsymbol{\theta}(t)$ es:

$$\begin{aligned} z_{nk}^{(t)} &= p(z_n = k \mid \mathbf{x}_n, c_n = c; \boldsymbol{\theta}(t)) \\ &= \frac{p(k \mid c) \cdot p(\mathbf{x} \mid k,\, c)}{\sum_{k'=1}^{K} p(k' \mid c) \cdot p(\mathbf{x} \mid k',\, c)} \end{aligned} \tag{9}$$

Es decir, la probabilidad de la muestra $\mathbf{x}$ de acuerdo a la distribución de probabilidad de la componente k de la clase c. También, se puede interpretar como el grado de pertenencia de la muestra $\mathbf{x}$ a la componente k de la clase c. Desde el punto de vista del paso M, que ahora detallaremos, z_{nk} es el peso o contribución parcial de una muestra a una componente, de forma que una misma muestra puede contribuir parcialmente a la estimación de los parámetros de varias componentes.

En el paso M se estiman los parámetros del modelo teniendo una estimación del grado de pertenencia de cada muestra de entrenamiento a cada componente. La estimación de la probabilidad a priori de cada clase queda fuera de la estimación por el algoritmo EM (ver Ec. 5). El resto de parámetros siguen una estimación similar a las Eqs. 6, 7 y 8, pero teniendo en cuenta la idea de que una muestra contribuye parcialmente acorde a z_{nk} a la estimación de los parámetros de la componente k en la clase c: En el paso M se estiman los parámetros del modelo teniendo una estimación del grado de pertenencia de cada muestra de entrenamiento a cada componente. La estimación de la probabilidad a priori de cada clase queda fuera de la estimación por el algoritmo EM (ver Eq. 5). El resto de parámetros siguen una estimación similar a las Ecs. 6, 7 y 8, pero teniendo en cuenta la idea de que una muestra contribuye parcialmente acorde a z_{nk} a la estimación de los parámetros de la componente k en la clase c:

$$\hat{p}_{ck}^{(t+1)} = \frac{1}{N_c} \sum_{n\,:\,c_n=c} z_{nk}^{(t)} \tag{10}$$

$$\hat{\boldsymbol{\mu}}_{ck}^{(t+1)} = \frac{1}{\sum\limits_{n\,:\,c_n=c} z_{nk}^{(t)}} \sum_{n\,:\,c_n=c} z_{nk}^{(t)}\, \boldsymbol{x}_n \tag{11}$$

$$\hat{\Sigma}_{ck}^{(t+1)} = \frac{1}{\sum\limits_{n\,:\,c_n=c} z_{nk}^{(t)}} \sum_{n\,:\,c_n=c} z_{nk}^{(t)} (\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ck})(\boldsymbol{x}_n - \hat{\boldsymbol{\mu}}_{ck})^t \tag{12}$$

Finalmente, antes de pasar a la implementación es necesario tratar la inicialización del algoritmo EM. La manera más habitual de inicializar el algoritmo EM es definir una inicialización de los parámetros del modelo $\boldsymbol{\theta}^{(0)}$ que dependen de la variable oculta, pero

también sería posible definir una inicialización de la distribución de probabilidad sobre la variable oculta.

En nuestro caso, inicializaremos la probabilidad a priori, la media y matriz de covarianza de cada componente. Seguidamente se muestra el código utilizado para la inicialización:

```
1 sigma=cell(C,K);
2 for c=classes'
3   ic=find(c==classes);
4   pkGc{ic}(1:K)=1/K;
5   idc=find(xl==c);
6   Nc=rows(idc);
7   mu{ic}=X(idc(randperm(Nc,K)),:)';
8   sigma(ic,1:K)=cov(X(idc,:),1)/K;
9 end
```

La línea 1 define una matriz de celdas $C \times K$ para almacenar las matrices de covarianza de cada una de las C clases y K componentes por clase. En el bucle desde la línea 2 a la línea 9 se inicializan los parámetros de cada clase. La línea 4 inicializa la probabilidad de cada componente como una distribución uniforme. La línea 7 define la media de cada componente como una muestra aleatoria de la clase c, y la matriz de covarianzas es inicialmente común a todas las componentes y estimada como la matriz de covarianzas de la clase dividida por el número de componentes. Como puedes deducir está inicialización es arbitrárea, pero ha demostrado funcionar bien en la práctica.

Al igual que en el clasificador gaussiano haremos uso de una función auxiliar para calcular la probabilidad de cada muestra en una componente, que se corresponde con el logaritmo del numerador de la Ec. 9, es decir, $\log p(k \mid c) + \log p(\mathbf{x} \mid k, c)$: Al igual que en el clasificador gaussiano haremos uso de una función auxiliar para calcular la probabilidad de cada muestra en una componente, que se corresponde con el logaritmo del numerador de la Eq. 9, es decir, $\log p(k \mid c) + \log p(\mathbf{x} \mid k, c)$:

```
10 function [zk] = compute_zk(pkGc,mu,sigma,X)
11   D=columns(X);
12   cons=log(pkGc);
13   cons=cons-0.5*D*log(2*pi);
14   cons=cons-0.5*logdet(sigma);
15   cons=cons-0.5*mu'*pinv(sigma)*mu;
16   lin=X*(mu'*pinv(sigma))';
17   qua=-0.5*sum((X*pinv(sigma)).*X,2);
18   zk=qua+lin+cons;
19 end
```

Observarás que es casi idéntica a la función auxiliar del clasificador gaussiano a excepción de la línea 13 que se incluye porque vamos a hacer una estimación de la probabilidad a posteriori y no solamente una clasificación como en el clasificador gaussiano.

Una vez han sido inicializados los parámetros, el algoritmo EM ejecuta los pasos E y M de manera iterativa hasta convergencia (o un número máximo de iteraciones). En nuestra implementación la condición de convergencia es que el incremento relativo de la log verosimilitud entre dos iteraciones no supere cierto umbral:

$$\frac{\mid L(\boldsymbol{\theta}; \boldsymbol{X})^{(t+1)} - L(\boldsymbol{\theta}; \boldsymbol{X})^{(t)} \mid}{\mid L(\boldsymbol{\theta}; \boldsymbol{X})^{(t)} \mid} < \epsilon$$

donde

$$L(\boldsymbol{\theta}; \boldsymbol{X}) = \sum_n \log p(c_n) + \log p(\mathbf{x}_n \mid c_n)$$

donde $p(\mathbf{x}_n \mid c_n)$ se define en la Ec. 4. De manera similar, $p(\mathbf{x}_n \mid c_n)$ es el denominador de la Ec. 9, es decir, que calcularemos $p(\mathbf{x}_n \mid c_n)$ como el sumatorio sobre las K componentes invocando la función `compute_zk` para cada componente.

En cada iteración del algoritmo EM se ejecutan los pasos E y M por cada clase, estimando z_{nk} y $\boldsymbol{\theta}^{(t+1)}$, respectivamente:

```
20   for c=classes'
21
22     % E step: Estimate zk
23     ic=find(c==classes);
24     idc=find(xl==c);
25     Nc=rows(idc);
26     Xc=X(idc,:);
27     zk=[];
28     for k=1:K
29       zk(:,k)=compute_zk(pkGc{ic}(k),mu{ic}(:,k),sigma{ic,k},Xc);
30     end
31
32     % Robust computation of znk and log-likelihood
33     maxzk=max(zk,[],2);
34     zk=exp(zk-maxzk);
35     sumzk=sum(zk,2);
36     zk=zk./sumzk;
37     L=L+Nc*log(pc(ic))+sum(maxzk+log(sumzk));
38
39     % M step: parameter update
40     % HERE YOUR CODE FOR PARAMETER ESTIMATION
41
42   end
43
44   % Likelihood divided by the number of training samples
45   L=L/N;
```

Las líneas 22 a 36 definen el paso E, mientras que las líneas de código que implementarás para el paso M deben ir a partir de la línea 40. En cuanto al paso E, la línea 29 estima la log probabilidad de cada muestra de la clase c para cada componente k dando lugar a cada vector columna de la matriz `zk`. Seguidamente en las líneas 33 a 36 se realiza la estimación de z_{nk} de la Ec. 9, pero de manera *robusta* para manejar valores pequeños de log probabilidad[5].

$$
\begin{aligned}
z_{nk}^{(t)} &= \frac{p(k \mid c) \cdot p(\mathbf{x} \mid k,\, c)}{\sum_{k'=1}^{K} p(k' \mid c) \cdot p(\mathbf{x} \mid k',\, c)} \\
z_{nk}^{(t)} &= \frac{\frac{p(k|c)\cdot p(\mathbf{x}|k,\, c)}{\max_{k''} p(k''|c)\cdot p(\mathbf{x}|k'',\, c)}}{\sum_{k'=1}^{K} \frac{p(k'|c)\cdot p(\mathbf{x}|k',\, c)}{\max_{k''} p(k''|c)\cdot p(\mathbf{x}|k'',\, c)}} \\
z_{nk}^{(t)} &= \frac{\exp\left(\log\left(\frac{p(k|c)\cdot p(\mathbf{x}|k,\, c)}{\max_{k''} p(k''|c)\cdot p(\mathbf{x}|k'',\, c)}\right)\right)}{\sum_{k'=1}^{K} \exp\left(\log\left(\frac{p(k'|c)\cdot p(\mathbf{x}|k',\, c)}{\max_{k''} p(k''|c)\cdot p(\mathbf{x}|k'',\, c)}\right)\right)} \\
z_{nk}^{(t)} &= \frac{\exp\left(\log p(k \mid c) + \log p(\mathbf{x} \mid k,\, c)\right) - \max_{k''} \log p(k'' \mid c) + \log p(\mathbf{x} \mid k'',\, c))}{\sum_{k'=1}^{K} \exp\left(\log p(k' \mid c) + \log p(\mathbf{x} \mid k',\, c) - \max_{k''} \log p(k'' \mid c) + \log p(\mathbf{x} \mid k'',\, c)\right)}
\end{aligned}
$$

Es decir, para cada muestra se divide la log probabilidad de cada componente por la log probabilidad de aquella componente con máxima log probabilidad. Después se calcula la exponenciación y se normaliza por la suma del total de las componentes.

Sobre el código, la línea 33 calcula la componente de máxima log probabilidad, restamos el máximo de la log probabilidad en la línea 34, calculamos la suma para todas las componentes en la línea 35 y normalizamos en la línea 36. Nótese que estas operaciones se hacen a la vez para todas las muestras de la clase c aprovechando la capacidad de Octave de trabajar con matrices y vectores de manera más eficiente.

La log verosimilitud para las muestras de la clase c se calcula en la línea 37 aprovechando el cálculo robusto realizado

$$
L(\boldsymbol{\theta}; \boldsymbol{X}) = \sum_{c} L(\boldsymbol{\theta}_c; \boldsymbol{X}_c)
$$

donde

$$
\begin{aligned}
L(\boldsymbol{\theta}_c; \boldsymbol{X}_c) &= \sum_{n\,:\,c_n = c} \log P(c_n) + \log p(\mathbf{x}_n \mid c_n) \\
&= N_c \, \log P(c_n) + \sum_{n\,:\,c_n = c} \log \sum_{k=1}^{K} p(\mathbf{x}_n, k \mid c_n)
\end{aligned}
$$

siendo $p(\mathbf{x}_n \mid c_n)$ el denominador de z_{nk} y que se calcula en la línea 35. Sin embargo, debido al cálculo robusto a $p(\mathbf{x}_n \mid c_n)$ le ha sido restado el máximo de cada componente,

[5] https://en.wikipedia.org/wiki/LogSumExp

y por tanto para compensar debemos sumar el máximo de cada componente como se observa en la línea 37. En la línea 45, una vez se han procesado todas las muestras, se puede observar como la log verosimilitud se normaliza por el número de muestras, pero es algo opcional.

En cuanto al paso M, será una implementación directa de las Ecs. 10, 11 y 12 aprovechando las operaciones matriciales en Octave. Ver Ejercicio 5.1 abajo.

La clasificación de cada muestra es un cálculo muy similar a la estimación de la log verosimilitud, ya que la función discriminante que se calcula es la probabilidad de cada muestra de acuerdo a los parámetros de cada una de las clases involucradas.

$$\begin{aligned} c^*(\boldsymbol{x}) &= \underset{c=1,\ldots,C}{\operatorname{argmax}} \ \log P(c) + \log p(\boldsymbol{x} \mid c) \\ &= \underset{c=1,\ldots,C}{\operatorname{argmax}} \ \log P(c) + \log \sum_{k=1}^{K} p(\boldsymbol{x}, k \mid c) \end{aligned}$$

Finalmente. el fichero `mixgaussian.m` de PoliformaT contiene la implementación descrita anteriomente. Por favor, dedica unos minutos a leer el código y comprobar que comprendes la implementación ya realizada.

Ejercicio 5.1 Implementa el paso M de estimación de los parámetros que gobiernan la mixtura de gaussianas y que se detallan en las Ecs. 10, 11 y 12.

Ejercicio 5.2 Para realizar una primera comprobación del correcto funcionamiento del clasificador de mixturas de gaussianas, replica el experimento realizado en el ejercicio 4.2 pero con mixturas de gaussianas de 1 componente y utilizando el mejor valor α.

Ejercicio 5.3 Realiza un experimento donde se evalúe el error de clasificación en función del número de componentes por mixtura para un número de componentes PCA y un valor α fijo. Representa gráficamente los resultados obtenidos en forma de una curva. Genera curvas de resultados adicionales utilizando diferente número de componentes PCA. Compara los resultados obtenidos con los reportados en la tarea MNIST.

Pista: Las tasas de error en MNIST con proyección a 30 componentes de PCA, y 1, 2 y 5 componentes por mixtura en convergencia con suavizado α =0.9 son 3.99 %, 3.72 % y 2.95 %, respectivamente.

A. Tutorial sobre Octave

A.1. Introducción

Varias de las técnicas empleadas en Aprendizaje Automático y Reconocimiento de Formas (RF) emplean cálculos vectoriales y matriciales, como por ejemplo en las implementaciones de clasificadores basados en la distribución gaussiana. 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).

A.2. 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.2.1. 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.2.2. 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.2.3. 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.2.4. 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.2.5. 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.2.6. 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)`.

A.3. 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`.