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

Proyecto de prácticas. Reconocimiento de dígitos manuscritos: MNIST

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

Índice

1. Objetivos

El principal objetivo de este proyecto de prácticas es la implementación y evaluación de diversos clasificadores estudiados en teoría sobre la tarea real de reconocimiento de dígitos manuscritos MNIST.

Este objetivo principal se subdivide en subobjetivos básicos que se pueden entender como fases o hitos de este proyecto:

1. Implementar la técnica de reducción de dimensionalidad PCA.
2. Visualizar los vectores de proyección PCA.
3. Realizar proyecciones lineales mediante vectores de proyección PCA.
4. Evaluar el efecto que tiene PCA sobre la tasa de error del clasificador basado en vecinos más cercanos.
5. Implementar el clasificador multinomial y gaussiano.
6. Resolver los problemas prácticos que nos podemos encontrar durante el desarrollo del clasificador multinomial y gaussiano.
7. Evaluar los clasificadores multinomial y gaussiano en función de sus parámetros.

La evaluación del proyecto de la asignatura consta de hasta 2 puntos en ejercicios obligatorios, y hasta 1 punto en ejercicios opcionales que el alumnado podrá elegir libremente. En total, la nota de prácticas constituye 3 puntos de la nota final de la asignatura.

El presente boletín debe leerse en su integridad previamente a la asistencia a la práctica para tener una visión completa del proyecto. Para realizar el proyecto de esta práctica se asume que el alumno ha realizado hasta el ejercicio 5.3 de la anterior práctica.

2. Principal Component Analysis

Para la implementación de la técnica de reducción de dimensionalidad PCA es necesario la utilización de la función `eig`. Esta función calcula los valores (y opcionalmente los vectores propios) de una matriz. Para más información ejecuta `help eig` desde Octave.

La función `eig` no garantiza que los vectores propios que devuelve estén ordenados de mayor a menor por valor propio asociado. Por ello, será necesaria la utilización de la función `sort` (ver `help sort`).

Ejercicio 2.1 Implementa la función `pca` que recibe como entrada los datos de entrenamiento `X` dispuestos por filas y devuelve el vector media `m` y la matriz de proyección `W` donde los vectores propios están dispuestos por columnas de mayor a menor valor propio asociado:

```
function [m,W]=pca(X)
  %% Aquí el código necesario
end
```

Recuerda que esta función debe guardarse en un fichero `pca.m`. Los pasos a seguir para implementar la función `pca` son:

1. Calcula la media `m` de los datos de entrenamiento `X`.
2. Resta la media a todos los datos de entrenamiento para obtener la matriz `Xm`.
3. Calcula la matriz de covarianza de dimensiones $D \times D$ como el producto de la transpuesta de la matriz `Xm'` por la matriz `Xm`.
4. Calcula los valores y vectores propios de la matriz de covarianzas resultante utilizando la función `eig`.
5. Ordena los vectores propios de mayor a menor valor propio asociado mediante la función `sort` para así definir la matrix `W`.

Ejercicio 2.2 Comprueba la correcta implementación de PCA representando gráficamente los 5 primeros eigenvectores. Recuerda que la visualización de vectores fue descrita en la práctica anterior. La representación gráfica de los 5 primeros vectores propios del conjunto de entrenamiento de MNIST es la siguiente:

3. Clasificador basado en vecinos más cercanos

En esta sección vamos a estudiar una implementación básica de un clasificador basado en vecinos más cercanos. También estudiaremos el efecto que tiene la técnica de reducción de dimensionalidad PCA sobre la tasa de error de un clasificador basado en vecinos más cercanos.

Para ello se dispone en PoliformaT de dos ficheros necesarios para el uso de este clasificador: `knn.m` y `L2dist.m`. Estos ficheros deben descargarse en el directorio de trabajo.

El fichero `knn.m` contiene la siguiente función:

```
function [err]=knn(X,xl,Y,yl,k)
....
end
```

que implementa un clasificador por `k` vecinos más cercanos. Más concretamente, esta función devuelve el error de clasificación de este clasificador usando `X` como muestras de entrenamiento (por filas) e `Y` como muestras de test (por filas). Las etiquetas de clase de

entrenamiento y test almacenadas en `xl` y `yl`, respectivamente, se proporcionan tal cual se cargan de fichero, es decir, como un vector columna.

El parámetro `k` indica que para la clasificación de cada muestra de test se considerarán las `k` muestras de entrenamiento más cercanas a esta muestra de test. De forma que la clase mayoritaria entre las `k` muestras de entrenamiento más cercanas define la clase de la muestra de test.

El fichero `L2dist.m` incluye una función que implementa el cálculo de distancia L2 (Euclídea):

```
function D = L2dist(X,Y)
....
endfunction
```

Esta función devuelve la matriz `D` de distancia L2 entre las N muestras de `X` y las M muestras de `Y`. La función `L2dist(X,Y)` se invoca desde la función `knn(X,xl,Y,yl,k)`.

Ejercicio obligatorio (0.5 puntos). Estudia el comportamiento del clasificador basado en el vecino más cercano (`k`=1) en combinación con PCA con el objetivo de obtener una gráfica como la que aparece en la Figura 1.

Para ello, te proporcionamos la primera parte del script `pca+knn-exp.m` que a partir del conjunto de datos `trdata` y `trlabels` evalúa un rango de valores de dimensionalidad de PCA `ks` dedicando un porcentaje a entrenamiento y otro a validación:

```
 1 #!/usr/bin/octave -qf
 2
 3 if (nargin!=5)
 4 printf("Usage: pca+knn-exp.m <trdata> <trlabels> <ks> <%%trper> <%%dvper>\n")
 5 exit(1);
 6 end;
 7
 8 arg_list=argv();
 9 trdata=arg_list{1};
10 trlabs=arg_list{2};
11 ks=str2num(arg_list{3});
12 trper=str2num(arg_list{4});
13 dvper=str2num(arg_list{5});
14
15 load(trdata);
16 load(trlabs);
17
18 N=rows(X);
19 rand("seed",23); permutation=randperm(N);
20 X=X(permutation,:); xl=xl(permutation,:);
21
```

```
22 Ntr=round(trper/100*N);
23 Ndv=round(dvper/100*N);
24 Xtr=X(1:Ntr,:); xltr=xl(1:Ntr);
25 Xdv=X(N-Ndv+1:N,:); xldv=xl(N-Ndv+1:N);
26 ...
```

Los resultados necesarios para realizar la Figura 1 se obtienen con la siguiente ejecución (desde el intérprete de comandos de `bash`):

```
./pca+knn-exp.m train-images-idx3-ubyte.mat.gz train-labels-idx1-ubyte.mat.gz \
"[1 2 5 10 20 50 100 200 500]" 90 10
```

Recuerda que la barra (\) no se debe escribir y únicamente indica que el comando sigue en la siguiente línea. Recuerda incluir en este script el código necesario para calcular la tasa de error del clasificador basado en el vecino más cercano sin aplicar PCA.

Las líneas 8-13 capturan los parámetros de entrada del script: `trdata` es el fichero con el conjunto de datos (muestras por filas); `trlabs`, fichero con el conjunto de etiquetas de clase (por filas); `ks`, vector con el rango de dimensiones PCA a evaluar; `trper`, porcentaje del conjunto de datos para entrenamiento; y `dvper` para validación. Las líneas 18-20 realizan el barajado por filas de los datos y etiquetas de clase. Las líneas 22-25 realizan la partición en entrenamiento y validación.

Completa el script `pca+knn-exp.m` con el código necesario para calcular PCA del conjunto de entrenamiento, realizar la proyección a k dimensiones de los conjuntos de entrenamiento y validación, y efectuar la llamada al clasificador basado en el vecino más cercano (`k`=1) con estos conjuntos para obtener la tasa de error en el conjunto de validación que se representa gráficamente en la Figura 1. Recuerda que también debes calcular la tasa de error con la dimensionalidad original de los datos.

A la vista de los resultados obtenidos en el ejercicio anterior, entrena un clasificador final que utilice el conjunto de datos de entrenamiento (incluyendo el conjunto de validación) con los mejores parámetros y calcule la tasa de error en el conjunto de test.

Para ello, te proporcionamos la primera parte del script `pca+knn-eva.m` que a partir del conjunto de datos de entrenamiento `trdata` y `trlabels`, el conjunto de test `tedata` y `telabels`, y el valor óptimo de la dimensionalidad de PCA `k`, debe calcular la tasa de error en el conjunto de test aplicando PCA y sin aplicar PCA.

```
1 #!/usr/bin/octave -qf
2
3 if (nargin!=5)
4 printf("Usage: pca+knn-eva.m <trdata> <trlabels> <tedata> <telabels> <k>\n")
5 exit(1);
6 end;
7
8 arg_list=argv();
9 trdata=arg_list{1};
```

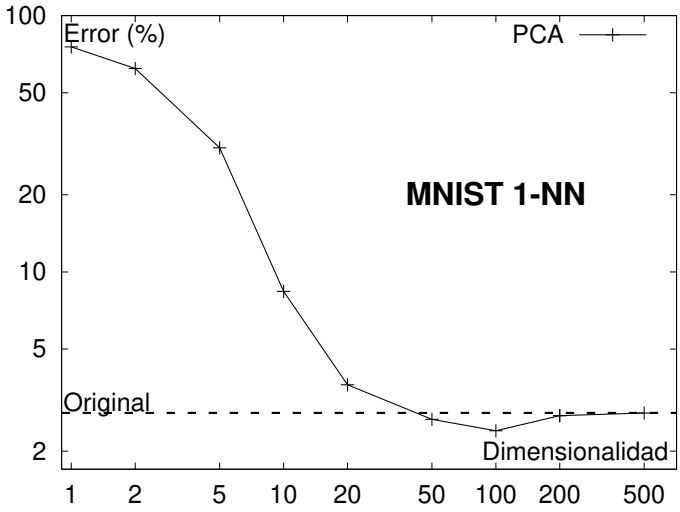

Figura 1: Resultados comparativos entre PCA y espacio original en MNIST utilizando el clasificador de 1 vecino más cercano (1-NN).

```
10 trlabs=arg_list{2};
11 tedata=arg_list{3};
12 telabs=arg_list{4};
13 k=str2num(arg_list{5});
14
15 load(trdata);
16 load(trlabs);
17 load(tedata);
18 load(telabs);
19 ...
```

Compara tus resultados con los aportados en la web de MNIST para *K-Nearest Neighbors*.

Ejercicios opcionales A continuación se plantean una serie de ejercicios opcionales con los cuales se puede sumar hasta 0.5 puntos a la nota de prácticas:

- Como se puede observar en la web de MNIST para *K-Nearest Neighbors* se proporcionan resultados con distancia *L3* que mejoran los resultados de la distancia *Euclidean (L2)* para la dimensionalidad original de los datos. Implementa otras distancias de la familia L_p:

$$d_p(\boldsymbol{a}, \boldsymbol{b}) = \left(\sum_{i=1}^{D} |a_i - b_i|^p \right)^{\frac{1}{p}}$$

al menos *L0*, *L1* y *L3*, y estudia su comportamiento en un clasificador basado en el vecino más cercano (k=1) combinado con una reducción de dimensionalidad PCA (0.25 puntos).

- La distancia Euclídea ponderada suele proporcionar resultados que mejoran los conseguidos por la distancia Euclídea no ponderada. Más concretamente, la distancia Mahalanobis-diagonal por clase presenta un compromiso entre complejidad de implementación y buenos resultados. Implementa esta distancia y estudia su comportamiento en un clasificador basado en el vecino más cercano (k=1) combinado con una reducción de dimensionalidad PCA (0.5 puntos).

  Nota: Dado que para algunas dimensiones obtendrás varianzas igual a cero, será necesario que suavices dichas varianzas mediante flat smoothing con la varianza de la gaussiana estandarizada (ver Tema 5 de teoría).

- Implementa el algoritmo de edición de Wilson y estudia su comportamiento en un clasificador basado en el vecino más cercano (k=1) combinado con una reducción de dimensionalidad PCA (0.5 puntos).

4. Implementación de un clasificador multinomial

Antes de abordar la implementación del clasificador multinomial es necesario recordar que, tanto el clasificador multinomial como el clasificador gaussiano que se verá en la siguiente sección, son instanciaciones del clasificador de 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, ya sea multinomial, gaussiana, u otra cualquiera. En esta sección, la f.d. condicional $p(x \mid c)$ será una distribución multinomial D-dimensional:

$$\begin{aligned} c^*(\boldsymbol{x}) &= \operatorname*{argmax}_{c=1,\dots,C} \log P(c) + \log p(x \mid c) \\ &= \operatorname*{argmax}_{c=1,\dots,C} \log P(c) + \log \frac{x_+!}{x_1! \cdots x_D!} \prod_{d=1}^{D} p_{cd}^{x_d} \\ &= \operatorname*{argmax}_{c=1,\dots,C} \log P(c) + \log \frac{x_+!}{x_1! \cdots x_D!} + \sum_{d=1}^{D} x_d \log p_{cd} \end{aligned}$$

Eliminando términos constantes y expresado en términos de función discriminante lineal:

$$\begin{aligned} c^*(\boldsymbol{x}) &= \operatorname*{argmax}_{c=1,\ldots,C} g_c(\boldsymbol{x}) \\ &= \operatorname*{argmax}_{c=1,\ldots,C} \boldsymbol{w}_c\, \boldsymbol{x} + w_{c0} \end{aligned}$$

donde

$$\begin{aligned} \boldsymbol{w}_c &= \log \boldsymbol{p}_c \\ w_{c0} &= \log p(c) \end{aligned}$$

La estimación máximo-verosímil de los parámetros del clasificador multinomial es:

$$\hat{p}(c) = \frac{N_c}{N} \qquad\qquad \hat{\boldsymbol{p}}_c = \frac{1}{\sum\limits_{n\,:\,c_n=c} \sum\limits_{d} x_{nd}} \sum_{n\,:\,c_n=c} \boldsymbol{x}_n$$

En el caso de la tarea MNIST, $\boldsymbol{x}_n$ es el vector de niveles de gris de la imagen n y x_{nd}, el nivel de gris del píxel d de la imagen n.

Antes de realizar un experimento, es necesario suavizar los parámetros correspondientes a los prototipos multinomiales pues existen píxeles cuyo nivel de gris es siempre cero. En este caso se propone aplicar el *suavizado de Laplace* (ver Tema 5 de teoría) a nuestros prototipos multinomiales:

$$\tilde{\boldsymbol{p}}_{cd} = \frac{\hat{\boldsymbol{p}}_{cd} + \epsilon}{\sum\limits_{d} (\hat{\boldsymbol{p}}_{cd} + \epsilon)} \qquad \epsilon > 0$$

Ejercicio obligatorio (0.5 puntos). Implementa la función `multinomial` que recibe como entrada los conjuntos de datos y etiquetas de clase de entrenamiento, `Xtr` y `xltr`, y los conjuntos de datos y etiquetas de clase de validación, `Xdv` y `xldv`, un vector de posibles valores `epsilon` de suavizado de Laplace; y devuelve las tasas de error de clasificación en el conjunto de entrenamiento `etr` y validación `edv` para cada uno de los valores en el vector `epsilons`.

```
function [etr,edv]=multinomial(Xtr,xltr,Xdv,xldv,epsilons)
  %% Aquí el código necesario
endfunction
```

Recuerda que esta función debe guardarse en un fichero `multinomial.m`. Realiza un experimento en MNIST para estudiar la evolución de la tasa de error en el conjunto de validación en función del valor `epsilon` utilizado con el objetivo de obtener una gráfica como la mostrada en la Figura 2.

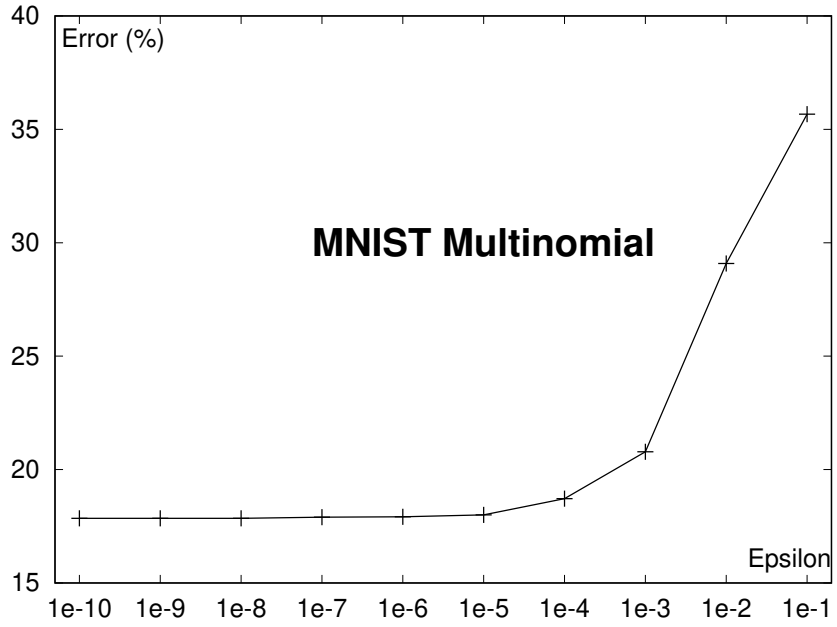

Figura 2: Error del clasificador multinomial en función del parámetro de suavizado ϵ.

Para ello debes implementar un script `multinomial-exp.m`, muy similar al script `pca+knn-exp.m` que invoca a la función `multinomial`. Dedica un 90 % de los datos de entrenamiento de MNIST al conjunto de entrenamiento y un 10 % al conjunto de validación.

A la vista de los resultados obtenidos en el conjunto de validación, entrena un clasificador final con los datos de entrenamiento (incluyendo el conjunto de validación) y el mejor valor de ϵ de suavizado. Estima el error en el conjunto de test con este clasificador final. Integra la realización de esta estimación en un script `multinomial-eva.m`, que también será similar a `pca+knn-eva.m`. Compara el resultado obtenido con los reportados en la web de MNIST.

5. Implementación de un clasificador gaussiano

En esta sección, la f.d. condicional $p(x \mid c)$ del clasificador de Bayes será una distribución gaussiana D-dimensional:

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

Por tanto:

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

Eliminando términos constantes y expresado como función discriminante cuadrática con $\boldsymbol{x}$:

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

donde

$$\begin{aligned}
W_c &= -\frac{1}{2}\Sigma_c^{-1} \\
\boldsymbol{w}_c &= \Sigma_c^{-1}\boldsymbol{\mu}_c \\
w_{c0} &= \log P(c) - \frac{1}{2}\log\ |\Sigma_c| - \frac{1}{2}\boldsymbol{\mu}_c^t\Sigma_c^{-1}\boldsymbol{\mu}_c
\end{aligned}$$

La estimación máximo-verosímil de los parámetros del clasificador gaussiano es ampliamente conocida:

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

El suavizado con la matriz identidad (*flat smoothing*) es:

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

Ejercicio obligatorio (1 punto). Implementa la función `gaussian` que recibe como entrada los conjuntos de datos y etiquetas de clase de entrenamiento, `Xtr` y `xltr`, los conjuntos de datos y etiquetas de clase de validación, `Xdv` y `xldv`, un vector de posibles valores `alphas` para realizar el suavizado con la matriz identidad; y devuelve las tasas de error de clasificación en el conjunto de entrenamiento `etr` y validación `edv` para cada uno de los valores `alphas`.

```
function [etr,edv]=gaussian(Xtr,xltr,Xdv,xldv,alphas)
  %% Aquí el código necesario
endfunction
```

Recuerda que esta función debe guardarse en un fichero `gaussian.m`. Algunos puntos a tener en cuenta de cara a la implementación:

- La implementación de la estimación de los parámetros de la gaussiana de cada clase, probabilidad a priori, media y matriz de covarianzas se deberá realizar seleccionando del conjunto de entrenamiento las muestras de cada clase.
- Las medias (una por cada clase) es recomendable almacenarlas en una matriz `mu` donde las medias están dispuestas por columnas.
- Para almacenar las matrices de covarianzas es recomendable utilizar un vector de celdas `sigma{c}`.
- Es recomendable implementar una función auxiliar `gc(pc(c),mu(:,c),sigma{c},X)` que calcule la probabilidad conjunta de cada muestra en `X` dados los parámetros de la clase c, es decir, $p(\boldsymbol{x}, c)$.

Si realizas un experimento (90 % entrenamiento, 10 % validación) sin suavizado ($\alpha = 1{,}0$), mediante el script `gaussian-exp.m` que debes implementar, observarás que Octave imprime varios avisos "`warning: matrix singular to machine precision`". Además observarás que la tasa de error de clasificación es del 90 %, es decir, se clasifican todas las muestras en la misma clase.

Esto se debe a que las matrices de covarianzas estimadas a partir de los datos de cada clase son singulares, es decir, el rango de estas matrices no es completo ya que existen filas (o columnas) que son linealmente dependientes unas de otras. Puedes calcular el rango de la matriz de covarianzas utilizando la función `rank` y comprobar que el rango es inferior a la dimensionalidad de los datos. En el caso de MNIST, esto se debe a que tenemos filas (o columnas) de la matriz de covarianzas que son todo cero, al ser estas filas (o columnas) parte del fondo de la imagen sobre el que está centrado el dígito manuscrito.

Como consecuencia de que las matrices de covarianzas sean singulares se plantean dos problemas de estimación de la probabilidad condicional gaussiana:

1. El determinante de un matriz singular es cero y su logaritmo será `-Inf`.
2. No es posible calcular la inversa de una matriz singular.

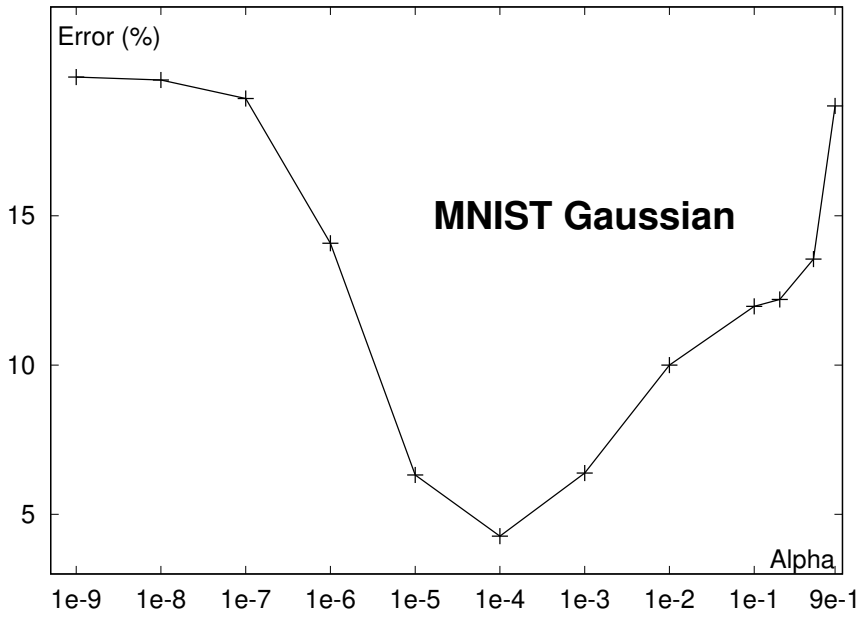

Figura 3: Error del clasificador gaussiano en función del parámetro de suavizado α.

El primer problema lo resolveremos reemplazando el cero por el valor más pequeño representable en nuestra máquina, que es la constante `realmin` en Octave. Adicionalmente, para que el cálculo del logaritmo del determinante sea más robusto utilizaremos la propiedad que dice que el determinante de una matriz es igual al producto de sus valores propios[1]. Por tanto, el logaritmo del determinante será la suma del logaritmo de los valores propios:

$$\log |\Sigma| = \sum_{d=1}^{D} \log \lambda_d$$

El segundo problema lo solventaremos reemplazando la inversa `inv` por la pseudoinversa `pinv` que sí que se puede calcular para una matriz singular.

Implementa las dos soluciones a la singularidad de las matrices de covarianzas. Es recomendable que implementes estas dos soluciones en una función `logdet(X)` que calcule el logaritmo del determinante de una matriz. Recuerda que si existe un valor propio que sea cero, no podrás calcular el logaritmo porque el determinante es cero y deberás tratarlo adecuadamente. Realiza un experimento con la misma partición que la del ejercicio anterior pero variando el valor de α de suavizado como se muestra en la Figura 3.

A la vista de los resultados obtenidos en el conjunto de validación, entrena un clasificador final con los datos de entrenamiento (incluyendo el conjunto de validación) y el mejor valor de α de suavizado. Estima el error en el conjunto de test con este clasificador

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

final. Integra la realización de esta estimación en un script `gaussian-eva.m`. Compara el resultado obtenido con los reportados en la web de MNIST, especialmente en la sección *Non-Linear Classifiers*.

Ejercicio opcional (0.5 puntos). Se puede observar como el clasificador gaussiano mejora la tasa de error del clasificador multinomial, pero según la web de MNIST queda lejos de la tasa de error conseguida con *40 PCA + quadratic classifier*. En nuestro caso, el clasificador cuadrático será un clasificador gaussiano. Realiza un experimento para evaluar el error del clasificador gaussiano con diferentes valores de α en función del número de componentes PCA a las cuales se proyectan los datos originales. Representa gráficamente los resultados obtenidos donde cada curva es un valor constante de α y el eje x representa la variación en el número de componentes PCA. Finalmente, entrena un clasificador final con la mejor combinación de parámetros y evalúa en el conjunto de test para comparar el resultado con el proporcionado en la web de MNIST.

6. Ejercicios opcionales

En esta sección, se plantean ejercicios opcionales adicionales que junto con los anteriores opcionales pueden sumar hasta 1 punto a la nota de prácticas:

- Implementación y evaluación de un clasificador Bernoulli al igual que se ha hecho con el clasificador multinomial. En este caso será necesario convertir previamente las imágenes que originalmente están en escala de grises a blanco y negro (binarización). Para ello debes aplicar un umbral de binarización, es decir, el nivel de la escala gris por encima del cual un píxel se considera negro y por debajo, blanco (0.5 puntos).
- Combinación de clasificadores fuertes mediante Bagging: clasificadores de vecinos más cercanos (0.5 puntos).

Por parte del alumnado se puede proponer la implementación de otros clasificadores distintos a los propuestos en los ejercicios opcionales que requerirán la valoración y aprobación del profesorado de la asignatura.