UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Máquinas de vectores soporte

Alfons Juan

DSIC

Departament de Sistemes Informàtics i Computació

Índice

1. Recordatorio de teoría de SVM

- Sea $\phi(\mathbf{x};\boldsymbol{\theta},\theta_0)$, una discriminante lineal obtenida mediante el método SVM, donde $\boldsymbol{\theta}$ es el vector de pesos óptimo calculado como:

$$\boldsymbol{\theta} = \sum_{m \in \mathcal{V}} c_m \, \alpha_m \, \mathbf{x}_m$$

  siendo c_m, la etiqueta de clase, α_m, el multiplicador de Lagrange óptimo, y $\mathcal{V}$, el conjunto de vectores soporte.

- El peso umbral θ_0, por las condiciones de KKT se calcula para cualquier vector soporte $m \in \mathcal{V}$ correcto ($\alpha_m < \mathcal{C}$) como:

$$\theta_0 \; = \; c_m - \boldsymbol{\theta}^t \mathbf{x}_m$$

- El margen es $\frac{2}{\|\boldsymbol{\theta}\|}$ y la tolerancia de margen ζ_m para un vector soporte $m \in \mathcal{V}$ se calcula como:

$$\zeta_m = 1 - c_m(\boldsymbol{\theta}^t \mathbf{x}_m + \; \theta_0)$$

▸ *Caso bidimensional:* $\mathbf{x}, \boldsymbol{\theta} \in \mathbb{R}^2$

▹ Ecuación de la recta de separación asociada a ϕ:

$$\phi(\mathbf{x}; \boldsymbol{\theta}, \theta_0) = 0 \quad \Rightarrow \quad \theta_1 x_1 + \theta_2 x_2 + \theta_0 = 0 \quad \Rightarrow \quad x_2 = -\frac{\theta_1}{\theta_2} x_1 - \frac{\theta_0}{\theta_2}$$

▹ Ecuaciones de las rectas que definen las fronteras del margen:

$$\phi(\mathbf{x}; \boldsymbol{\theta}, \theta_0) = +1 \;\Rightarrow\; \theta_1 x_1 + \theta_2 x_2 + \theta_0 = +1 \;\Rightarrow\; x_2 = -\frac{\theta_1}{\theta_2} x_1 - \frac{\theta_0 - 1}{\theta_2}$$

$$\phi(\mathbf{x}; \boldsymbol{\theta}, \theta_0) = -1 \;\Rightarrow\; \theta_1 x_1 + \theta_2 x_2 + \theta_0 = -1 \;\Rightarrow\; x_2 = -\frac{\theta_1}{\theta_2} x_1 - \frac{\theta_0 + 1}{\theta_2}$$

2. Aprendizaje y clasificación con libsvm

- ***Aprendizaje:*** obtiene los ***multiplicadores de Lagrange*** óptimos, $\{\alpha_n\}$, asociados a los vectores de aprendizaje $\{\boldsymbol{x}_n\}$.

- ***Clasificación:*** mediante discriminante lineal de ***vector de pesos*** $\boldsymbol{\theta}$ y ***umbral*** θ_0, derivados de su ***soporte,*** esto es, vectores de aprendizaje de multiplicadores no-nulos.

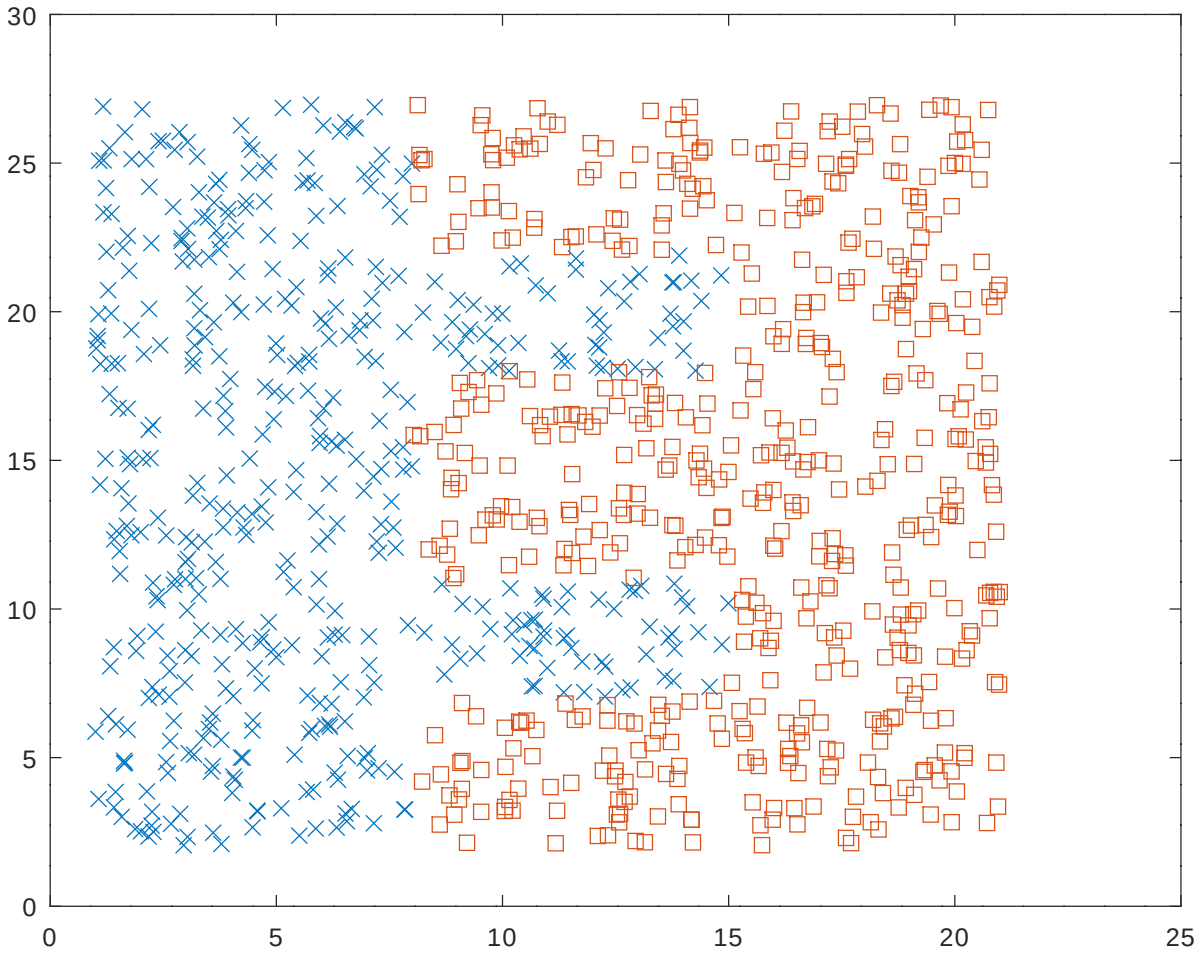
- ***libsvm:*** implementa el aprendizaje y clasificación mediante las librerías ***svmtrain.mex*** y ***svmpredict.mex***, respectivamente.

  - Multiplicadores en ***sv_coef:*** los positivos asociados a vectores soporte de la clase $+1$ y los negativos a la -1; esto es, en realidad corresponden al producto del verdadero multiplicador por la etiqueta de clase del vector soporte asociado, $c_n\alpha_n$.

- ***Corpus hart:*** no linealmente separable en su espacio de representación original, pero sí en un espacio transformado mediante un ***kernel de base radial*** (RBF), de mayor dimensión.

hart.sh

```
#!/usr/bin/octave
load("hart/tr.dat"); load("hart/trlabels.dat");
X1=X(xl==1,:); X2=X(xl==2,:);
plot(X1(:,1),X1(:,2),"x",X2(:,1),X2(:,2),"s"); print -color hart.eps
```



▸ Aprendizaje: svmtrain

octave

```
svmtrain
```

```
Usage: model = svmtrain(training_label_vector, training_instance_matrix,
↪  'libsvm_options');
libsvm_options:
-s svm_type : set type of SVM (default 0)
→0 -- C-SVC→→(multi-class classification)
→1 -- nu-SVC→→(multi-class classification)
→2 -- one-class SVM
→3 -- epsilon-SVR→(regression)
→4 -- nu-SVR→→(regression)
-t kernel_type : set type of kernel function (default 2)
→0 -- linear: u'*v
→1 -- polynomial: (gamma*u'*v + coef0)^degree
→2 -- radial basis function: exp(-gamma*|u-v|^2)
→3 -- sigmoid: tanh(gamma*u'*v + coef0)
→4 -- precomputed kernel (kernel values in training_instance_matrix)
-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/num_features)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 100)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates : whether to train a SVC or SVR model for probability estimates,
↪  0 or 1 (default 0)
-wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
-v n: n-fold cross validation mode
-q : quiet mode (no outputs)
```

▸ *Aprendizaje (cont.):* kernel RBF y coste $C = 1$

harte.sh

```
#!/usr/bin/octave
load("hart/tr.dat"); load("hart/trlabels.dat"); X1=X(xl==1,:); X2=X(xl==2,:);
res=svmtrain(xl,X,'-t 2 -c 1'); whos res; fieldnames(res)
for [val,key]=res
  printf("%s: %s ",key,typeinfo(val));
  if isscalar(val) val
  elseif ismatrix(val) printf("%d %d\n",rows(val),columns(val));
  else printf("\n"); end; end
```

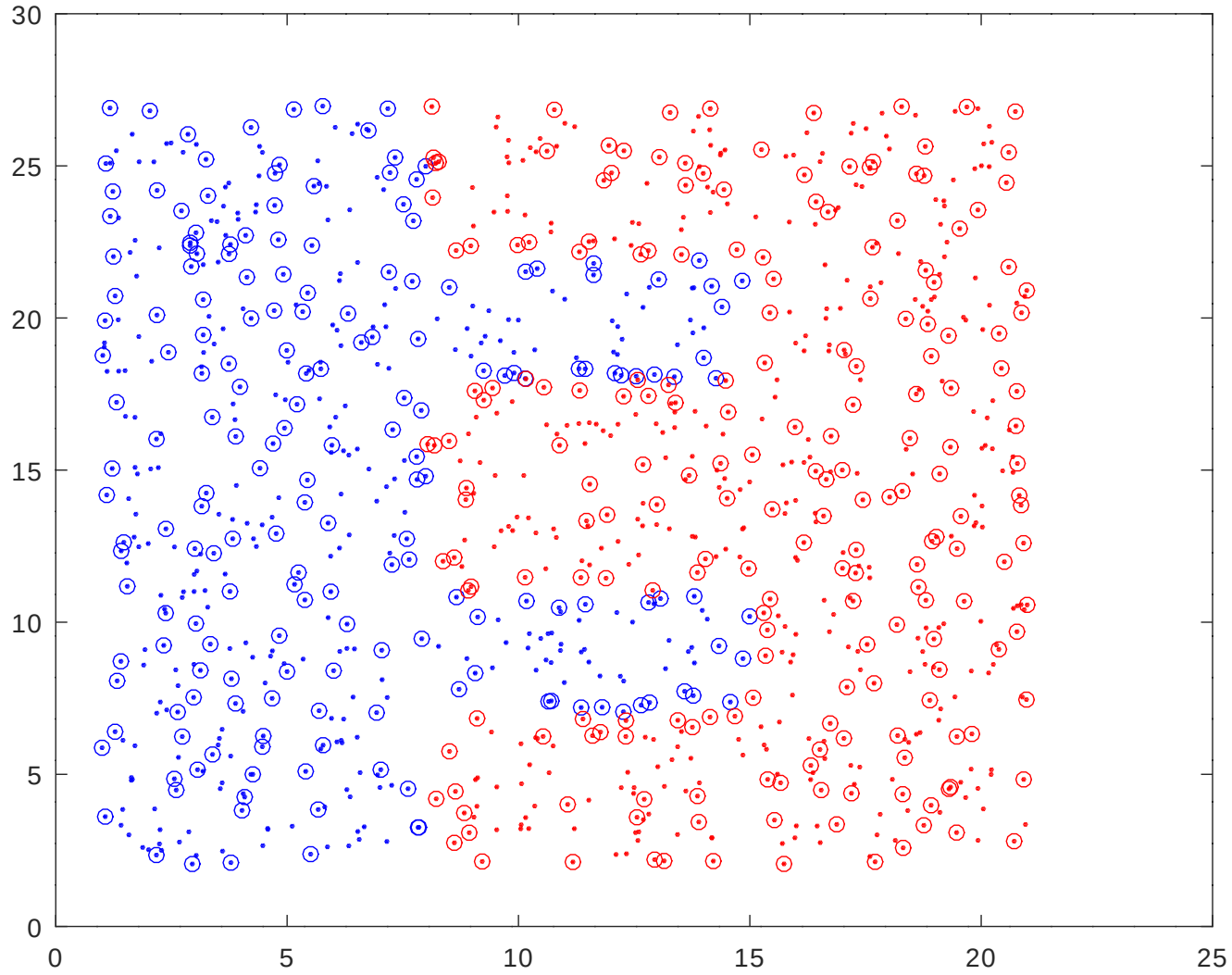
harte.out

```
.*.*
optimization finished, #iter = 2298
nu = 0.174579
obj = -108.403744, rho = -0.096434
nSV = 398, nBSV = 91
Total nSV = 398
Variables in the current scope:
   Attr Name        Size                     Bytes  Class
   ==== ====        ====                     =====  =====
        res         1x1                      19224  struct
Total is 1 element using 19224 bytes
ans =
{
  [1,1] = Parameters
  [2,1] = nr_class
  [3,1] = totalSV
  [4,1] = rho
  [5,1] = Label
  [6,1] = sv_indices
  [7,1] = ProbA
  [8,1] = ProbB
  [9,1] = nSV
  [10,1] = sv_coef
  [11,1] = SVs
}
Parameters: matrix 5 1
nr_class: scalar val =  2
totalSV: scalar val =  398
rho: scalar val = -0.096434
Label: matrix 2 1
sv_indices: matrix 398 1
ProbA: matrix 0 0
ProbB: matrix 0 0
nSV: matrix 2 1
sv_coef: matrix 398 1
SVs: sparse matrix 398 2
```

▸ *Aprendizaje (cont.):* vectores soporte

hartep.sh

```
#!/usr/bin/octave
load("hart/tr.dat"); load("hart/trlabels.dat"); X1=X(xl==1,:); X2=X(xl==2,:);
res=svmtrain(xl,X,'-t 2 -c 1 -q'); Xsv=X(res.sv_indices,:);
xlsv=xl(res.sv_indices); Xsv1=Xsv(xlsv==1,:); Xsv2=Xsv(xlsv==2,:);
plot(X1(:,1),X1(:,2),"b.",Xsv1(:,1),Xsv1(:,2),"bo",
     X2(:,1),X2(:,2),"r.",Xsv2(:,1),Xsv2(:,2),"ro"); print -color hartep.eps
```



▸ Clasificación: svmpredict

octave

```
svmpredict
```

```
Usage: [predicted_label, accuracy, decision_values/prob_estimates] =
↪  svmpredict(testing_label_vector, testing_instance_matrix, model,
↪  'libsvm_options')
       [predicted_label] = svmpredict(testing_label_vector,
       ↪  testing_instance_matrix, model, 'libsvm_options')
Parameters:
  model: SVM model structure from svmtrain.
  libsvm_options:
    -b probability_estimates: whether to predict probability estimates,
    ↪  0 or 1 (default 0); one-class SVM not supported yet
    -q : quiet mode (no outputs)
Returns:
  predicted_label: SVM prediction output vector.
  accuracy: a vector with accuracy, mean squared error, squared
  ↪  correlation coefficient.
  prob_estimates: If selected, probability estimate vector.
```

harte2.sh

```
#!/usr/bin/octave
load("hart/tr.dat"); load("hart/trlabels.dat");
res=svmtrain(xl,X,'-t 2 -c 1 -q');
load("hart/ts.dat"); load("hart/tslabels.dat");
svmpredict(yl,Y,res,' ');
```

harte2.out

```
Accuracy = 98.1% (981/1000) (classification)
```

3. Ejercicios

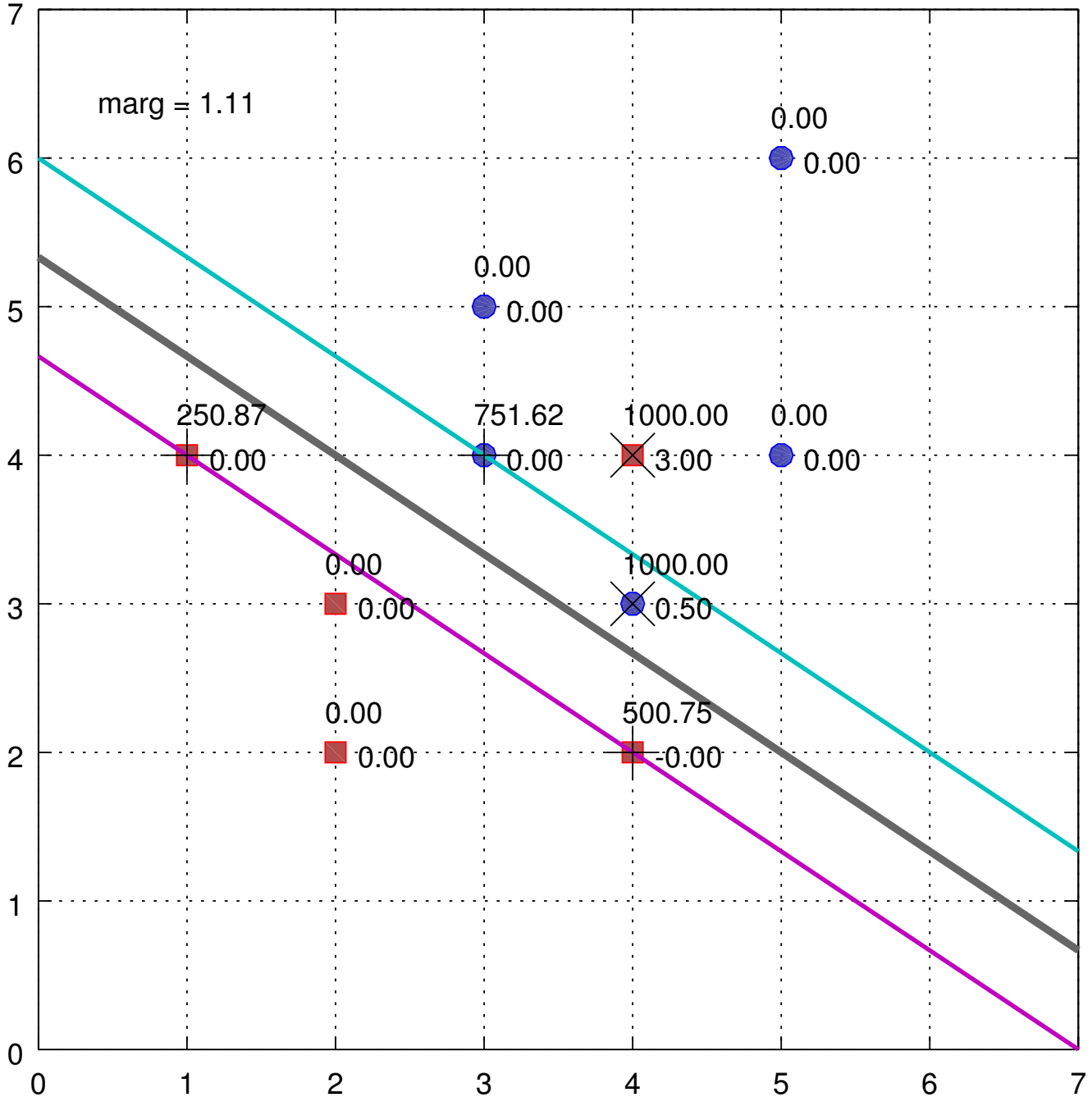
3.1. Ejercicio 1 (0.4 puntos)

- ***Tarea mini:*** dos conjuntos de datos de entrenamiento 2D
  - trSep.dat, trSeplabels.dat: linealmente separable (sin kernel)
  - tr.dat, trlabels.dat: no linealmente separable

- Para cada uno de los dos conjuntos de la tarea mini:
  1. Obtén los SVM sin kernel con C grande ($C = 1000$).
  2. Halla los multiplicadores; vectores soporte; vector de pesos y umbral de la discriminante lineal; y margen correspondiente.
  3. Calcula los parámetros de la frontera lineal de separación.
  4. Representa los datos, vectores soporte y frontera.

- Para el conjunto no-separable, prueba valores relevantes de C y:
  1. Determina los valores de tolerancia de margen, ζ.
  2. Marca los vectores soporte "erróneos" en la gráfica.

- ***Pista:*** gráfica “ideal” para el no-separable (con $C = 1000$)



3.2. Ejercicio 2 (0.4 puntos)

- ***Tarea MNIST:*** las SVMs bajan del $1\,\%$ de error
  `http://devres.zoomquiet.top/data/20160422121512/index.html`
- Estima el error de las SVMs en MNIST, en función de $\mathcal{C}$ (`-c` 1, 10, 100) y el kernel (`-t` 0, 1, 2, 3), con sus parámetros específicos:
  - Polinómico (`-t` 1): grado del polinomio (`-d` 1, 2, 3, 4, 5, ...)
  - Gaussiano (`-t` 2): gamma (`-g` 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, ...)
  - Sigmoide (`-t` 3): gamma (`-g` 1e-1, 1e-2, 1e-3, 1e-4, 1e-5, ...)
- Recomendaciones:
  - Normaliza los datos en $[0, 1]$ (dividiendo por 255).
  - Como en mixturas, haz un script que explore los parámetros indicados arriba para unos pocos valores de PCA ($D = 50, 100, 200$).
  - Por eficiencia, utiliza una partición entrenamiento-validación reducida; por ejemplo, $90\,\%$-$10\,\%$ con 6000 muestras en total.
- Describe brevemente los resultados obtenidos con base en una representación adecuada de los mismos, gráfica o tabular.

3.3. Ejercicio 3 (0.2 puntos)

- Estima el error en MNIST a partir de los conjuntos oficiales, con los mejores valores hallados en el ejercicio 2. Acompaña la estimación de un intervalo de confianza al $95\,\%$ y discute los resultados obtenidos, comparándolos con los obtenidos con mixturas y los publicados en la página oficial de MNIST .