
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

# Algorisme Perceptró: aplicació a tasques de classificació

DSIC
Departament de Sistemes Informàtics i Computació

# Objectius formatius

- Implementar classificadors lineals
- Programar l'algorisme Perceptró
- Aplicar l'algorisme Perceptró a tasques de classificació

# Índex

# 1 Funcions discriminants lineals

Tot classificador pot representar-se com:

$$c(x) = \arg\max_c \; g_c(x)$$

on cada classe $c$ utilitza una ***funció discriminant*** $g_c(x)$ que mesura la pseudo-probabilitat de pertinença d'un objecte $x$ a $c$

Les funcions discriminants més utilitzades són ***lineals*** (amb $\boldsymbol{x}$):

$$g_c(\boldsymbol{x}) = \boldsymbol{w}_c^t \boldsymbol{x} + w_{c0} \qquad \text{on} \quad \boldsymbol{x} = \begin{pmatrix} x_1 \\ \vdots \\ x_D \end{pmatrix} \quad \text{y} \quad \boldsymbol{w_c} = \begin{pmatrix} w_{c1} \\ \vdots \\ w_{cD} \end{pmatrix}$$

Amb notació ***homogènia***:

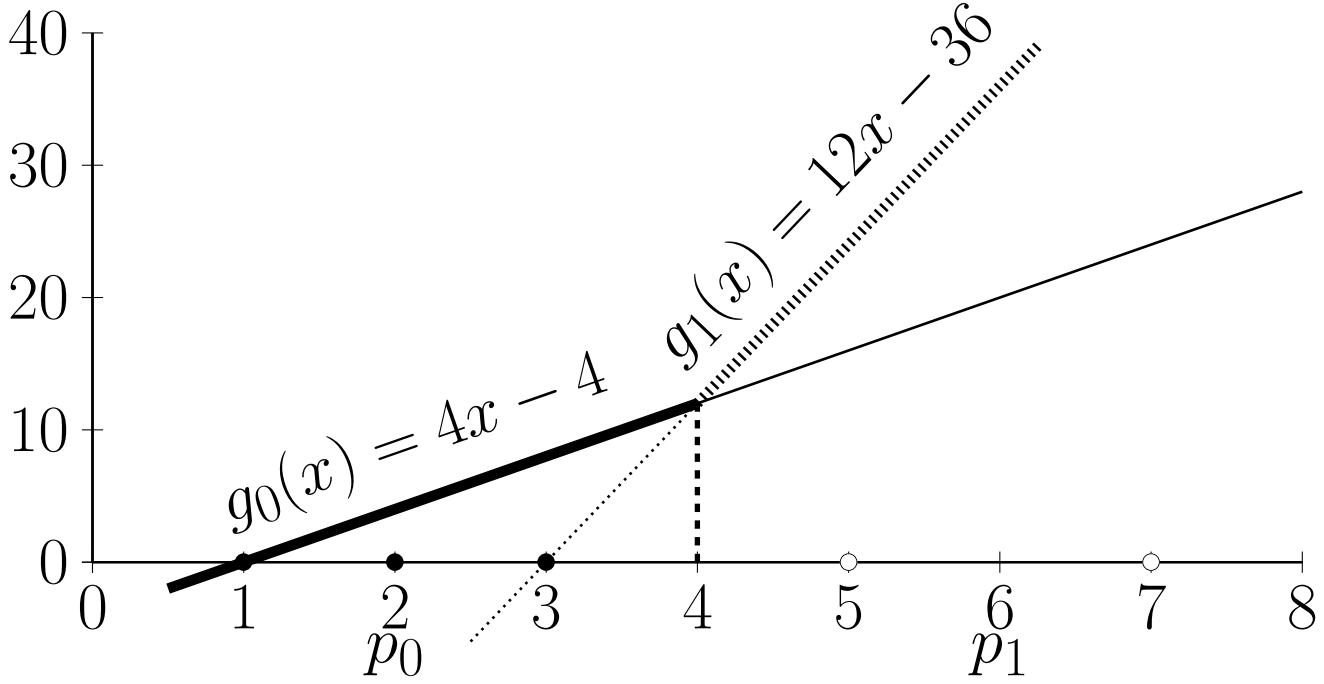
$$g_c(\mathbf{x}) = \mathbf{w}_c^t \mathbf{x} \qquad \text{on} \quad \mathbf{x} = \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix} \quad \text{y} \quad \mathbf{w}_c = \begin{pmatrix} w_{c0} \\ \boldsymbol{w}_c \end{pmatrix}$$

# linmach.py

```
import math
import numpy as np

def linmach(w,x):
  C = w.shape[1]; cstar=1; max=float('-inf');
  for c in range(C):
    g=np.dot(w[:,c],x);
    if g>max:
      max=g; cstar=c;
  return cstar;
```

# test_linmach.py

```
#!/usr/bin/python

import numpy as np
from linmach import linmach

w=np.array([[-4,-36],[4,12]]);
for x in range(1,9):
  y=np.array([1, x]);
  print('c(%d)=%d' % (x,linmach(w,y)));
```




```
c(1)=0
c(2)=0
c(3)=0
c(4)=0
c(5)=1
c(6)=1
c(7)=1
c(8)=1
```

# 2 Algorisme Perceptró

***Entrada:*** $\{(\mathbf{x}_n, c_n)\}_{n=1}^{N}$, $\{\mathbf{w}_c\}_{c=1}^{C}$, $\alpha \in \mathbb{R}^{>0}$ y $b \in \mathbb{R}$

***Eixida:*** $$\{\mathbf{w}_c\}^* = \underset{\{\mathbf{w}_c\}}{\arg\min} \sum_n \left[ \max_{c \neq c_n} \mathbf{w}_c^t \mathbf{x}_n + b > \mathbf{w}_{c_n}^t \mathbf{x}_n \right]$$

***Mètode:***

$$[P] = \begin{cases} 1 & \text{si } P = \text{vertader} \\ 0 & \text{si } P = \text{fals} \end{cases}$$

**repetir**

**per a tota** dada $\mathbf{x}_n$

$err = \texttt{fals}$

**per a tota** classe $c$ diferent de $c_n$

**si** $\mathbf{w}_c^t \mathbf{x}_n + b > \mathbf{w}_{c_n}^t \mathbf{x}_n$: $\mathbf{w}_c = \mathbf{w}_c - \alpha \cdot \mathbf{x}_n$; $err = \texttt{vertader}$

**si** $err$: $\mathbf{w}_{c_n} = \mathbf{w}_{c_n} + \alpha \cdot \mathbf{x}_n$

**fins** que no queden mostres mal classificades
(o s'arribe a un màxim d'iteracions prefixat)

# perceptron.py

```
import numpy as np

def perceptron(data,b=0.1,a=1.0,K=200):
  (N,L)=data.shape; D=L-1;
  labs=np.unique(data[:,L-1]); C=labs.size;
  w = np.zeros((L,C));
  for k in range(1,K+1):
    E=0;
    for n in range(N):
     xn=np.concatenate(([1],data[n,:D]));
     cn=np.where(labs==data[n,L-1])[0][0];
     er=0; g=np.dot(w[:,cn],xn);
     for c in range(C):
      if c != cn and np.dot(w[:,c],xn) + b > g:
          w[:,c] = w[:,c] - a*xn; er=1;
     if er==1:
        w[:,cn] = w[:,cn] + a*xn; E=E+1;
    if E==0:
      break;
  return w,E,k;
```

# test_perceptron.py

```
#!/usr/bin/python

import numpy as np
from perceptron import perceptron

data=np.array([[0, 0, 0], [1, 1, 1]]);
w,E,k=perceptron(data);
print(w);
print('E=%d k=%d' % (E,k));
```

L'execució d'aquest script proporciona la següent eixida:

```
[[ 1. -1.]
 [-1.  1.]
 [-1.  1.]]
E=0 k=3
```

# 3 Aplicació a tasques de classificació: OCR

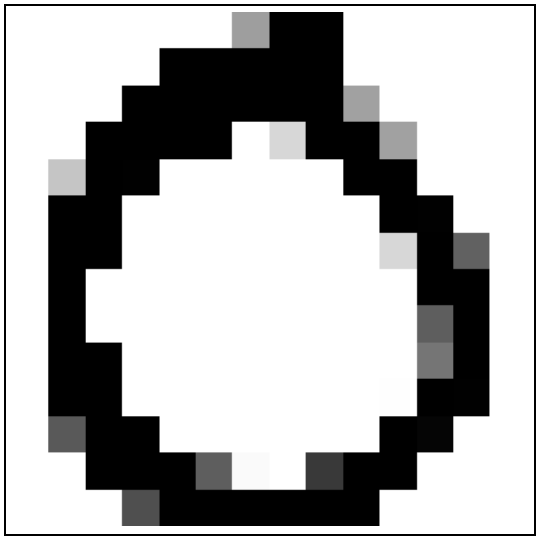
El corpus `OCR_14x14` és una matriu de 1000 files (mostres) i 197 columnes (196 característiques i etiqueta de classe):

```
$ head -n 3 OCR_14x14
```

```
 0 0 0 0 0 0 0 0 0.62 1 0 0 0 0 0 0 0 0 0 0 0.63 1 1 1 1   ... 0 0
 0 0 0 0 0 0 0.38 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0  ... 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.91 1 0.99     ... 0 0
```

Cada mostra correspon a una imatge de dígit manuscrit normalitzada a 14x14 grisos i llegida en l'ordre de lectura usual:

```
#!/usr/bin/python

import numpy as np
import matplotlib.pyplot as plt

data=np.loadtxt('OCR_14x14');
N,L=data.shape; D=L-1;
I=np.reshape(data[1,:D],(14,14));
plt.imshow(I, cmap='gray_r');
plt.axis('off'); plt.show();

np.random.seed(23);
perm=np.random.permutation(N);
data=data[perm];
for n in range(N):
  I=np.reshape(data[n,:D],(14,14));
  plt.imshow(I, cmap='gray_r');
  plt.axis('off'); plt.show();
```



# 3.1 Entrenament

```
#!/usr/bin/python

import numpy as np
from perceptron import perceptron

data=np.loadtxt('OCR_14x14');
N,L=data.shape; D=L-1;
labs=np.unique(data[:,L-1]); C=labs.size;

np.random.seed(23); perm=np.random.permutation(N);
data=data[perm];

NTr=int(round(.7*N)); train=data[:NTr,:];
w,E,k=perceptron(train);

np.savetxt('percep_w',w,fmt='%.2f');
print(w);
```

```
[[-38.   -34.   -36.   ... -32.   -50.   -36.  ]
 [  0.     0.     0.   ...   0.     0.     0.  ]
 [  0.     0.     0.   ...   0.     0.     0.  ]
 ...
 [  0.     0.     0.23 ...   0.54  -0.77  -1.  ]
 [  0.     0.     0.96 ...   0.    -0.96   0.  ]
 [  0.     0.     0.   ...   0.     0.     0.  ]]
```

# Càlcul de la funció discriminant

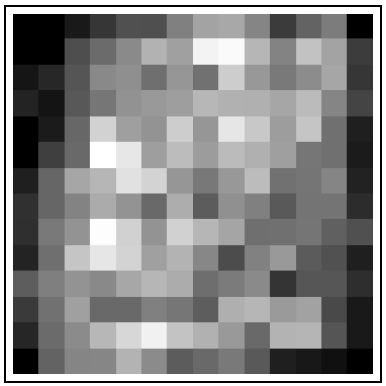
El grau de pertinença de $\mathbf{x}$ (amb $x_0 = 1$) a la classe del dígit $c$ és $g_c(\mathbf{x}) = \mathbf{w}_c^t \mathbf{x}$, on $\mathbf{w}_c$ ve donat per la columna $c$ de $\mathtt{w}$:

```
#!/usr/bin/python

from __future__ import print_function
import numpy as np

data=np.loadtxt('OCR_14x14'); w=np.loadtxt('percep_w');
N,L=data.shape; D=L-1;
labs=np.unique(data[:,L-1]); C=labs.size;

for n in range(N):
  for c in range(C):
      xn=np.concatenate(([1],data[n,:D]));
      print('g_%d(x_%d)=%.0f ' % (c,n,np.dot(w[:,c],xn)),end='');
  print('');
```

```
g_0(x_0)=-687 g_1(x_0)=-882 g_2(x_0)=-854 g_3(x_0)=-789 ...
g_0(x_1)=-519 g_1(x_1)=-655 g_2(x_1)=-553 g_3(x_1)=-588 ...
g_0(x_2)=-730 g_1(x_2)=-877 g_2(x_2)=-914 g_3(x_2)=-785 ...
...
```

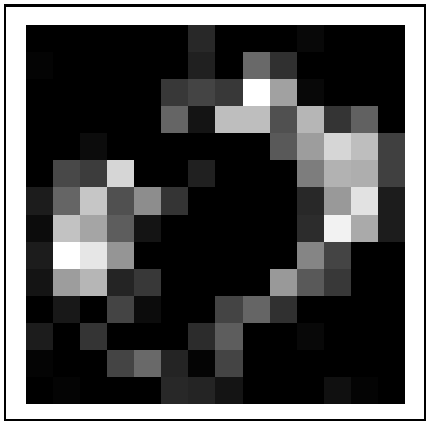
Els pesos de major variabilitat són més discriminatius que els pesos que varien poc.

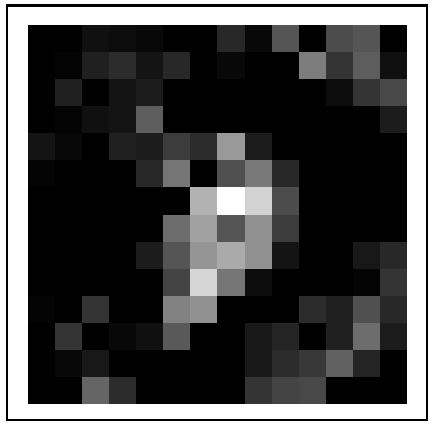
```
#!/usr/bin/python
import numpy as np
import matplotlib.pyplot as plt
w=np.loadtxt('percep_w');
sw=np.std(w[1:],axis=1);
I=np.reshape(sw,(14,14));
plt.imshow(I, cmap='gray');
plt.axis('off'); plt.show();
```



Els pesos d'una classe $c$ comparativament majors que els de la resta de classes indiquen característiques (no neg.; p.e. grisos) "pro-$c$"; els menors "anti-$c$".

```
#!/usr/bin/python
import numpy as np
import matplotlib.pyplot as plt
w=np.loadtxt('percep_w');
D,C=w.shape;
mw=np.mean(w[1:],axis=1);
for c in range(C):
  wc=w[1:,c];
  pw=np.maximum(0,wc-mw);
  I=np.reshape(pw,(14,14));
  plt.imshow(I, cmap='gray');
  plt.axis('off'); plt.show();
  nw=np.minimum(0,wc-mw);
  I=np.reshape(nw,(14,14));
  plt.imshow(I, cmap='gray_r');
  plt.axis('off'); plt.show();
```





# 3.2 Estimació de l'error

Estimació de l'error de classificació amb interval de confiança al 95% mitjançant les mostres no emprades en entrenament:

```
#!/usr/bin/python
import math; import numpy as np
from linmach import linmach
from confus import confus
data=np.loadtxt('OCR_14x14');
N,L=data.shape; D=L-1;
labs=np.unique(data[:,L-1]);C=labs.size;
np.random.seed(23);
perm=np.random.permutation(N);
data=data[perm]; NTr=int(round(.7*N));
M=N-NTr; test=data[NTr:,:];
w=np.loadtxt('percep_w');rl=np.zeros((M,1));
for m in range(M):
  tem=np.concatenate(([1],test[m,:D]));
  rl[m]=labs[linmach(w,tem)];
ner,m=confus(test[:,L-1].reshape(M,1),rl);
print('ner=%d'%ner);print(m);
per=ner/M; print('per=%.3f'%per);
r=1.96*math.sqrt(per*(1-per)/M);
print('r = %.3f' % r);
print('I=[%.3f, %.3f]'%(per-r,per+r));
```

```
ner = 17
[[27.  0.  0.  0. ...]
 [ 0. 25.  0.  0. ...]
 [ 1.  0. 34.  0. ...]
 [ 0.  0.  1. 26. ...]
 [ 0.  0.  1.  0. ...]
 [ 1.  0.  0.  0. ...]
 [ 0.  0.  1.  0. ...]
 [ 0.  0.  0.  0. ...]
 [ 1.  1.  0.  2. ...]
 [ 0.  0.  0.  0. ...]]
per = 0.057
r = 0.026
I=[0.031, 0.083]
```

# 3.3 Efecte de $\alpha$

```
#!/usr/bin/python
import numpy as np; from perceptron import perceptron;
from linmach import linmach; from confus import confus
data=np.loadtxt('OCR_14x14');
N,L=data.shape; D=L-1; labs=np.unique(data[:,L-1]); C=labs.size;
np.random.seed(23); perm=np.random.permutation(N); data=data[perm];
NTr=int(round(.7*N)); train=data[:NTr,:]; M=N-NTr; test=data[NTr:,:];
print('#       a   E   k Ete');
print('#------- --- --- ---');
for a in [.1,1,10,100,1000,10000,100000]:
  w,E,k=perceptron(train,a=a); rl=np.zeros((M,1));
  for n in range(M):
    rl[n]=labs[linmach(w,np.concatenate(([1],test[n,:D])))];
  nerr,m=confus(test[:,L-1].reshape(M,1),rl);
  print('%8.1f %3d %3d %3d' % (a,E,k,nerr));
```

```
#       a   E   k Ete
#------- --- --- ---
     0.1   0  11  14
     1.0   0  12  17
    10.0   0  10  15
   100.0   0  10  15
  1000.0   0  10  15
 10000.0   0  10  15
100000.0   0  10  15
```

El paràmetre $\alpha$, $\alpha > 0$, ***no*** té gran efecte sobre el comportament de Perceptró.

# 3.4 Efecto de $b$

```
#!/usr/bin/python
import numpy as np; from perceptron import perceptron;
from linmach import linmach; from confus import confus
data=np.loadtxt('OCR_14x14');
N,L=data.shape; D=L-1; labs=np.unique(data[:,L-1]); C=labs.size;
np.random.seed(23); perm=np.random.permutation(N); data=data[perm];
NTr=int(round(.7*N)); train=data[:NTr,:]; M=N-NTr; test=data[NTr:,:];
print('#       b   E   k Ete');
print('#------- --- --- ---');
for b in [.1,1,10,100,1000,10000,100000]:
  w,E,k=perceptron(train,b); rl=np.zeros((M,1));
  for n in range(M):
    rl[n]=labs[linmach(w,np.concatenate(([1],test[n,:D])))];
  nerr,m=confus(test[:,L-1].reshape(M,1),rl);
  print('%8.1f %3d %3d %3d' % (b,E,k,nerr));
```

```
#       b   E   k Ete
#------- --- --- ---
     0.1   0  12  17
     1.0   0  11  14
    10.0   0  12  18
   100.0   0  17  14
  1000.0   0 123  14
 10000.0 162 200  11
100000.0 538 200  29
```

El paràmetre $b$ ***sí*** té gran efecte.

Si les mostres són linealment separables, escollirem un $b$ amb el qual Perceptró convergisca ($E = 0$) i siga comparativament elevat (p.e. $b = 1000$).

# 3.5 Entrenament del classificador final

Entrenem el nostre classificador ***final*** amb totes les mostres:

```
#!/usr/bin/python
import numpy as np
from perceptron import perceptron
data=np.loadtxt('OCR_14x14');
N,L=data.shape;
np.random.seed(23); perm=np.random.permutation(N); data=data[perm];
w,E,k=perceptron(data,1000,0.1);
np.savetxt('OCR_14x14__w',w,fmt='%.2f');
print(w);
```

Examinem els pesos del classificador final:

```
[[-1993.1   -1848.2   -1949.3   ... -1913.1   -2222.5   -1944.6  ]
 [    0.       0.       0.    ...     0.       0.       0.    ]
 [    0.       0.       0.    ...     0.       0.       0.    ]
 ...
 [  -50.009   -32.206    6.783 ...   -46.971   -41.912   -38.398]
 [   -5.712    -4.96    11.649 ...    -7.594    -4.861    -4.627]
 [    0.       0.       0.    ...     0.       0.       0.    ]]
```

# 4 Exercici: aplicació a altres tasques

Siguen els següents 4 conjunts de dades de sengles tasques:

1. ***expressions:*** 225 expressions facials representades amb vectors 4096-D i classificades en 5 classes (1=sorpresa, 2=felicitat, 3=tristesa, 4=angoixa i 5=disgust).

2. ***gauss2D:*** 4000 mostres sintètiques procedents de dues classes equiprobables de forma Gaussiana bidimensional.

3. ***gender:*** 2836 expressions facials representades mitjancant vectors 1280-D i classificades per gènere.

4. ***videos:*** 7985 vídeos de bàsquet/no-bàsquet descrits amb vectors 2000-D extrets d'histogrames de característiques locals.

# Activitat

1. Elabora un script `experiment.py` en Python per a automatitzar l'aplicació de l'algorisme Perceptró a altres tasques. Aquest script rep com a entrada les dades, i el rang de valors de $\alpha$ i $b$:

```
#!/usr/bin/python
import sys; import math; import numpy as np
from perceptron import perceptron; from confus import confus
from linmach import linmach
if len(sys.argv)!=4:
  print('Usage: %s <data> <alphas> <bs>' % sys.argv[0]);
  sys.exit(1);
data=np.loadtxt(sys.argv[1]);
alphas=np.fromstring(sys.argv[2],sep=' ');
bs=np.fromstring(sys.argv[3],sep=' ');
...
for a in alphas:
  for b in bs:
    w,E,k=perceptron(train,b,a); rl=np.zeros((M,1));
    ...
```

Des de l'intèrpret de comandos executarem

```
$ ./experiment.py OCR_14x14 '.1 1 10 100 1000 10000' '0.1'
```

# Activitat

Una possible eixida de resultats del script seria:

```
#       a          b    E    k  Ete  Ete (%)     Ite (%)
#------- --------- --- ---- ---- -------- ----------
     0.1       0.1    0   11   14      4.7 [2.3, 7.1]
     1.0       0.1    0   12   17      5.7 [3.1, 8.3]
    10.0       0.1    0   10   15      5.0 [2.5, 7.5]
   100.0       0.1    0   10   15      5.0 [2.5, 7.5]
  1000.0       0.1    0   10   15      5.0 [2.5, 7.5]
 10000.0       0.1    0   10   15      5.0 [2.5, 7.5]
```

2. Obtín una taula de resultats semblant a la següent:

| tasca | Ete (%) | Ite (%) |
|---|---|---|
| OCR_14x14 | 4.7 | [2.3, 7.1] |
| expressions | 3.0 | [0.0, 7.1] |
| gauss2D | 10.5 | [8.8, 12.2] |
| gender | 4.6 | [3.2, 6.0] |
| videos | 27.0 | [25.2, 28.8] |

# Examen

- L'examen de laboratori consistirà en una modificació del teu script `experiment.py` per a la realització d'un experiment amb un conjunt de dades ja conegut o nou.

- El dia de l'examen hauràs de lliurar:
  - Script `experiment.py` original
  - Script `experiment.py` modificat
  - Resultats obtinguts i comentaris sobre els mateixos