UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Algoritmo Perceptrón: aplicación a tareas de clasificación

DSIC
Departamento de Sistemas Informáticos y Computación

Objetivos formativos

- Implementar clasificadores lineales
- Programar el algoritmo Perceptrón
- Aplicar el algoritmo Perceptrón a tareas de clasificación

Índice

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1. Funciones discriminantes lineales

Todo clasificador puede representarse como:

$$c(x) = \arg\max_{c}\ g_c(x)$$

donde cada clase c utiliza una ***función discriminante*** $g_c(x)$ que mide el grado de pertenencia de un objeto x a la clase c

Las funciones discriminantes más utilizadas son ***lineales*** (con $\boldsymbol{x}$):

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

Con notación ***homogénea***:

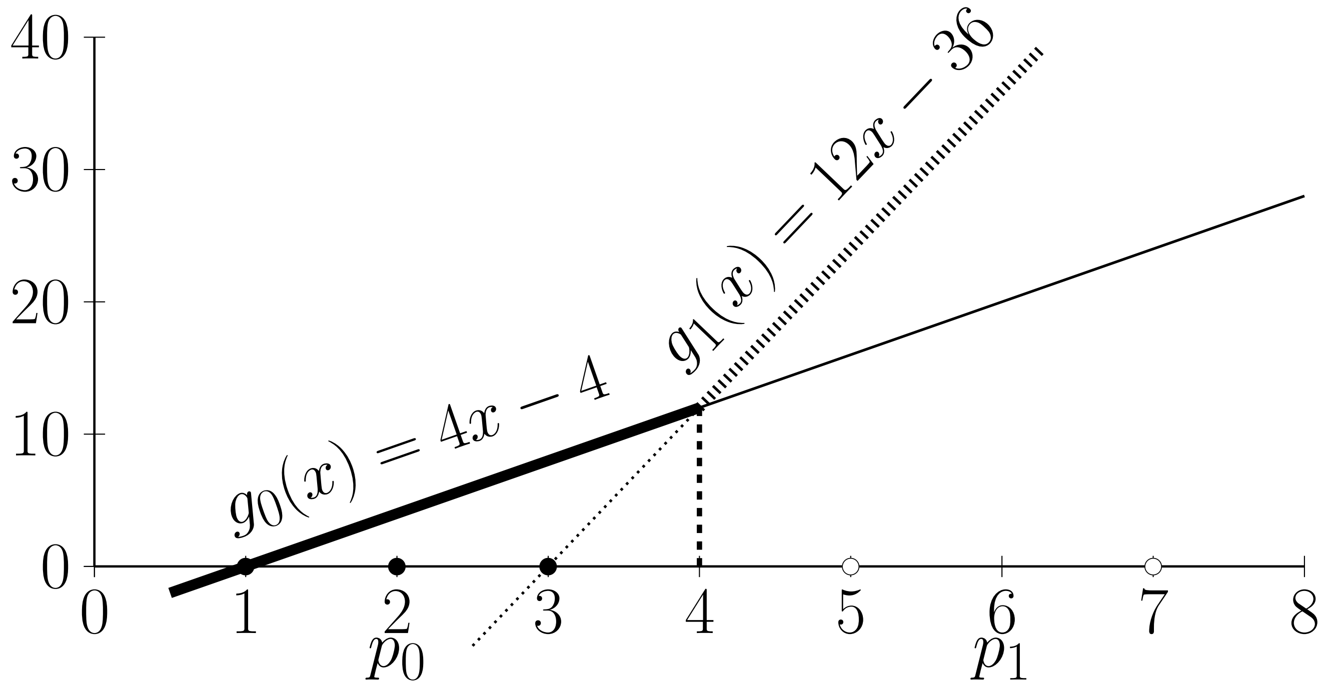
$$g_c(\mathbf{x}) = \mathbf{w}_c^t \mathbf{x} \qquad \text{donde} \quad \mathbf{x} = \begin{pmatrix} 1 \\ \boldsymbol{x} \end{pmatrix} \quad \text{y} \quad \mathbf{w}_c = \begin{pmatrix} w_{c0} \\ \boldsymbol{w}_c \end{pmatrix}$$

linmach.py

```
import numpy as np

def linmach(w,x):
    C = w.shape[1]
    cstar = None
    max_g = float('-inf')
    for c in range(C):
        g = np.dot(w[:,c], x)
        if g > max_g:
            max_g = g
            cstar = c
    return cstar
```

test_linmach.py

```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. Algoritmo Perceptrón

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

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

$$[P] = \begin{cases} 1 & \text{si} \quad P = \text{verdadero} \\ 0 & \text{si} \quad P = \text{falso} \end{cases}$$

Método:

repetir

para todo dato $\mathbf{x}_n$

$err = \texttt{falso}$

para toda clase c distinta de c_n

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

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

hasta que no queden muestras mal clasificadas
(o se llegue a un máximo de iteraciones prefijado)

perceptron.py

```python
import numpy as np

def perceptron(data, b=0.1, a=1.0, K=200):
  (N, L) = data.shape; D = L-1
  labels = np.unique(data[:, D])
  C = labels.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(labels==data[n, D])[0][0]
      err = False; 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; err = True
      if err == True:
        w[:, cn] = w[:, cn] + a*xn; E = E+1
    if E == 0:
      break
  return w,E,k
```

test_perceptron.py

```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\nk = %d' % (E, k))
```

La ejecución de este script proporciona la siguiente salida:

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

3. Aplicación a tareas de clasificación: OCR

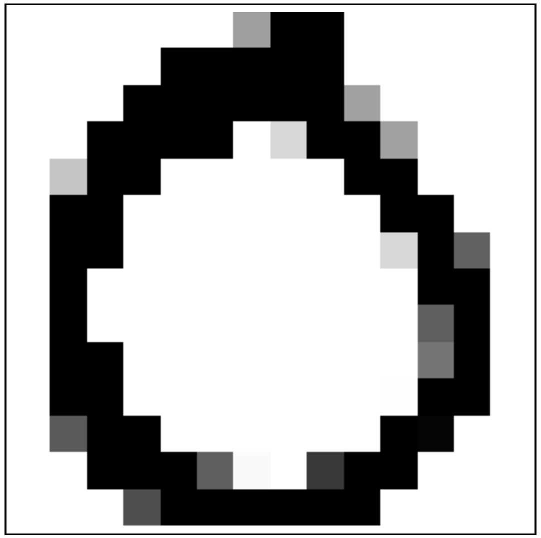
El corpus `OCR_14x14` es una matriz de 1000 filas (muestras) y 197 columnas (196 características y etiqueta de clase):

```
$ 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 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 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 0.91 1 0.99      ... 0 0
```

Cada muestra corresponde a una imagen de dígito manuscrito normalizada a 14x14 grises y leída en el orden de lectura usual:

```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. Entrenamiento

```
import numpy as np
from perceptron import perceptron

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

Cálculo de la función discriminante

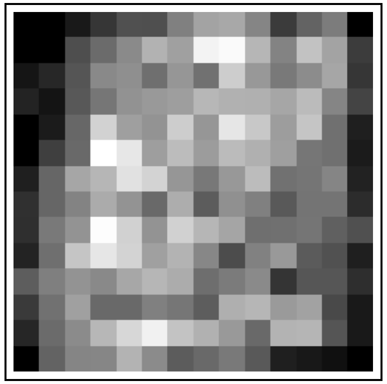
El grado de pertenencia de $\mathbf{x}$ (con $x_0 = 1$) a la clase del dígito c es $g_c(\mathbf{x}) = \mathbf{w}_c^t \mathbf{x}$, donde $\mathbf{w}_c$ viene dado por la columna c de `w`:

```
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[:,D])
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 ...
...
```

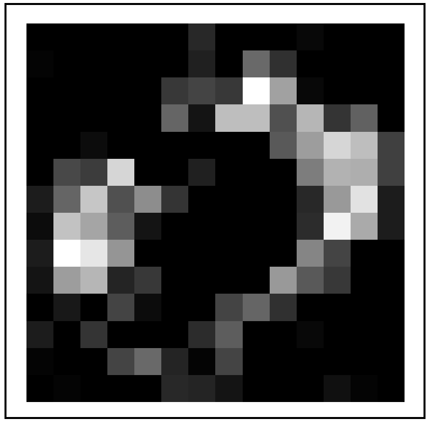
Los pesos de mayor variabilidad son más discriminativos que los pesos que varían poco.

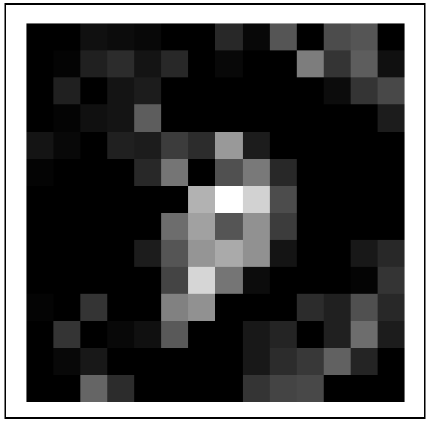
```python
import numpy as np
import matplotlib.pyplot as plt
w = np.loadtxt('percep_w')
sw = np.std(w[1:], axis=1)
I = np.reshape(sw, (14,14))
plt.imshow(I, cmap='gray')
plt.axis('off'); plt.show()
```



Los pesos de una clase c comparativamente mayores que los del resto de clases indican qué características (no negativas; p.e. grises) son “pro-c”; los menores son “‘anti-c”.

```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. Estimación del error

Estimación del error de clasificación con intervalo de confianza al 95 % mediante las muestras no empleadas en entrenamiento:

```
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. Efecto de α

```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 [0.1, 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 parámetro α, $\alpha > 0$, ***no*** tiene gran efecto sobre el comportamiento de Perceptrón.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

3.4. Efecto de b

```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 [0.1, 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 parámetro b ***sí*** tiene gran efecto.

Si las muestras son linealmente separables, escogeremos un b con el que Perceptrón converja ($E = 0$) y sea comparativamente elevado (p.e. $b = 1000$).

3.5. Entrenamiento del clasificador final

Entrenamos nuestro clasificador ***final*** con todas las muestras:

```
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)
```

Examinemos los pesos del clasificador 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. Ejercicio: aplicación a otras tareas

Sean los siguientes 4 conjuntos de datos de sendas tareas:

1. ***expressions:*** 225 expresiones faciales representadas mediante vectores 4096-D y clasificadas en 5 clases (1=sorpresa, 2=felicidad, 3=tristeza, 4=angustia y 5=disgusto).

2. ***gauss2D:*** 4000 muestras sintéticas procedentes de dos clases equiprobables de forma Gaussiana bidimensional.

3. ***gender:*** 2836 expresiones faciales representadas mediante vectores 1280-D y clasificadas por género.

4. ***videos:*** 7985 vídeos de baloncesto o no-baloncesto representados mediante vectores 2000-D extraídos de histogramas de características locales.

Actividad

1. Elabora un script `experiment.py` en Python para automatizar la aplicación del algoritmo Perceptron a otras tareas. Este script recibe como entrada los datos, y el rango de valores de α y b:

```
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));
    ...
```

Desde el intérprete de comandos ejecutaremos:

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

Actividad

Una posible salida de resultados del script sería:

```
#        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. Obtén una tabla resumen de mejores resultados aproximadamente como la siguiente:

tarea	Ete (%)	Ite (%)
OCR_14x14	4.7	[2.3, 7.1]
expressions	6.0	[0.3, 11.6]
gauss2D	10.5	[8.8, 12.2]
gender	4.6	[3.2, 6.0]
videos	27.0	[25.2, 28.8]

Examen

- El examen de laboratorio consistirá en una modificación de tu script `experiment.py` para la realización de un experimento con un conjunto de datos ya conocido o nuevo.

- El día del examen deberás entregar:
  - Script `experiment.py` original
  - Script `experiment.py` modificado
  - Resultados obtenidos y comentarios sobre los mismos