UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Práctica 1. Mixturas de Gaussianas
Ejercicio a realizar

Alfons Juan

DSIC

Departament de Sistemes Informàtics i Computació

Índice

1. mixgaussian-exp.py

- Importación de las librerías python necesarias:

mixgaussian-exp.py

```
1 import sys
2 import math
3 import numpy as np
4 import pickle
5 from sklearn import mixture
```

  - numpy: librería estándar de cálculo numérico
  - pickle: módulo estándar para guardar y leer objetos en fichero
  - sklearn: librería popular de aprendizaje automático (en CPU)
    - ↦ sklearn.mixture: paquete para mixturas de Gaussianas

- Lectura de parámetros de la línea de comandos:

mixgaussian-exp.py

```
7  if len(sys.argv)!=5:
8    print('Usage: %s <trdata> <trlabels> <%%trper> <%%dvper>'
     ↪  % sys.argv[0]);
9    sys.exit(1);
10
11 X= np.load(sys.argv[1])['X'];
12 xl=np.load(sys.argv[2])['xl'];
13 trper=int(sys.argv[3]);
14 dvper=int(sys.argv[4]);
```

  - **`X`** recoge datos (sin etiquetas de clase)
  - **`xl`** recoge las etiquetas de clase de los datos
  - **`trper`** es el porcentaje de datos de entrenamiento (p.e. 80)
  - **`dvper`** es el porcentaje de datos de desarrollo (p.e. 20)

- Parámetros para crear un modelo de mixturas de Gaussianas:

mixgaussian-exp.py

```
16 K=1;
17 rc=0.1;
18 seed=23;
```

  - **K** es el número de componentes
  - **rc** es factor de regularización de la matriz de covarianzas
  - **seed** es una semilla para generación de números aleatorios

- Barajado y partición train-dev de los datos:

mixgaussian-exp.py

```
20 N=X.shape[0];
21 np.random.seed(seed); perm=np.random.permutation(N);
22 X=X[perm]; xl=xl[perm];
23
24 # Selecting a subset for train and dev sets
25 Ntr=round(trper/100*N);
26 Xtr=X[:Ntr,:]; xltr=xl[:Ntr];
27 Ndv=round(dvper/100*N);
28 Xdv=X[N-Ndv:,:]; xldv=xl[N-Ndv:];
```

  - **`Ntr`** es el número de datos de training
  - **`Xtr`** son los datos de training
  - **`xltr`** son las etiquetas de los datos de training
  - **`Ndv`** es el número de datos de development
  - **`Xdv`** son los datos de development
  - **`xldv`** son las etiquetas de los datos de development

- Etiquetas, número de clases, etc.

mixgaussian-exp.py

```
30 labs=np.unique(xltr).astype(int);
31 C=labs.shape[0];
32 N,D=Xtr.shape;
33 M=Xdv.shape[0];
34 gtr=np.zeros((C,N));
35 gdv=np.zeros((C,M));
```

  - **labs** es un vector de etiquetas de clase sin repeticiones
  - **C** es el número de etiquetas de clase
  - **N** es el número de datos de entrenamiento
  - **D** es la dimensión de los datos (número de características)
  - **M** es el número de datos de development
  - **gtr** son las discriminantes de los datos de entrenamiento
  - **gdv** son las discriminantes de los datos de development

- Estandarización de los datos:

mixgaussian-exp.py

```
38 mu=np.mean(Xtr,axis=0);
39 sigma=np.std(Xtr,axis=0);
40 sigma[sigma==0]=1;
41 Xtr=(Xtr-mu)/sigma;
42 Xdv=(Xdv-mu)/sigma;
```

  - **mu** es la media (de cada característica) del training
  - **sigma** es la desviación típica (de cada característica) del training
  - **Xtr** es el training estandarizado
  - **Xdv** es el development estandarizado

- Creación, entrenamiento y evaluación de una mixtura por clase:

mixgaussian-exp.py

```
45 model=[]
46 for c,lab in enumerate(labs):
47   Xtrc=Xtr[xltr==lab];
48   Nc=Xtrc.shape[0];
49   pc=Nc/N;
50   gmm=mixture.GaussianMixture(n_components=K, reg_covar=rc,
     ↪  random_state=seed);
51   gmm.fit(Xtrc);
52   gtr[c]=math.log(pc)+gmm.score_samples(Xtr);
53   gdv[c]=math.log(pc)+gmm.score_samples(Xdv);
54   model.append((pc,gmm));
```

  - **model** es el modelo de clasificación
  - **Xtrc** son los datos de entrenamiento de la clase c
  - **Nc** es el número de datos de entrenamiento de la clase c
  - **pc** es el prior de la clase c
  - **gmm** es una mixtura de K Gaussianas ajustada con Xtrc
  - **gtr** y **gdv** son las discriminantes de los datos Xtr y Xdv

- Clasificación, cálculo de error y almacenamiento del modelo:

mixgaussian-exp.py

```
57 idx=np.argmax(gtr,axis=0);
58 etr=np.mean(np.not_equal(labs[idx],xltr))*100;
59 idx=np.argmax(gdv,axis=0);
60 edv=np.mean(np.not_equal(labs[idx],xldv))*100;
61
62 print('  K       rc    etr    edv')
63 print('--- ------- ----- -----')
64 print(f'{K:3} {rc:3.1e} {etr:5.2f} {edv:5.2f}');
65
66 filename = 'gmm.K1.rc0.1.mod'
67 pickle.dump(model, open(filename, 'wb'))
```

  - **idx** son los índices de las etiquetas de clase más probables
  - **filename** es el nombre del fichero donde se guarda el modelo

2. Tarea MNIST

- Vamos a aplicar el clasificador de mixturas de Gaussianas a MNIST proyectada mediante PCA a 20 dimensiones

- Descarga los siguientes ficheros de PoliformaT:
  - `train-images-idx3-ubyte.pca20.npz` (vectores)
  - `train-labels-idx1-ubyte.pca20.npz` (etiquetas)

- Prueba:

```
python mixgaussian-exp.py train-images-idx3-ubyte.pca20.npz
↪  train-labels-idx1-ubyte.pca20.npz 80 20
```

```
  K       rc    etr    edv
--- ------- ----- -----
  1 1.0e-01  5.78  5.78
```

3. Ejercicio

- ***Objetivo:*** entrena un clasificador basado en mixturas de gaussianas que minimice el error de clasificación en MNIST
  - Consulta la clase GaussianMixture y la sección 2.1 de la guía del usuario para familiarizarte con los parámetros ajustables
  - Entrega una memoria que describa los experimentos realizados para ajustar los parámetros del clasificador final (ya ajustado)
  - Junto con la memoria, entrega un fichero en formato `pickle` sin comprimir con el clasificador final
    - ↦ El tamaño del fichero `pickle` debe ser menor de 1 Mbyte

▸ *Ejemplo muy simple:*

- Copiamos `mixgaussian-exp.py` a `exp02.py` y lo modificamos para entrenar una mixtura de 2 componentes por clase y guardar el modelo en `gmm.K2.rc0.1.mod`:

```
python exp02.py train-images-idx3-ubyte.pca20.npz
↪  train-labels-idx1-ubyte.pca20.npz 80 20
```

```
  K       rc    etr    edv
--- ------- ----- -----
  2 1.0e-01  4.66  4.69
```

- El nuevo modelo produce menor error en dev que el modelo con una sola Gaussiana por clase (4.69 en lugar de 5.78)
- El tamaño de `gmm.K2.rc0.1.mod` es menor de 1 Mbyte

```
$ du -sh gmm.K2.rc0.1.mod
196K	gmm.K2.rc0.1.mod
```

- Si no se nos ocurre cómo mejorar el 4.69, entrenamos con todos los datos para obtener el clasificador final y lo subimos a PoliformaT junto con una memoria