Machine Learning for Medicine TP 3

Gaussian Mixture Models and the Expectation-Maximization Algorithm

The goal of the TME is to understand the Expectation-Maximization algorithm, and to learn how to use it for practical reasons.

We will use the *scikit-learn Python* library `http://scikit-learn.org` which is already installed on the computers.

Data (some simulated data sets + data sets of TME 1)

We explore two data sets downloadable from the Machine Learning Repository (`http://archive.ics.uci.edu/ml/index.php`)

- Breast Cancer Wisconsin (Diagnostic) Data Set (`https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`)
- Mice Protein Expression Data Set (`https://archive.ics.uci.edu/ml/datasets/Mice+Protein+Expression`)

Some Theory

Notations:

X	data matrix, $N \times d$
d	dimensions
$k = 1, \ldots, K$	Gaussians
$n = 1, \ldots, N$	observations
$p(k)$	population fraction in k
$p(x_k)$	model probability in x
μ_k	K means, each a vector of length d
Σ_k	K covariance matrices, each size $d \times d$
$p(k\|n) \equiv p_{nk}$	K probabilities for each of N data points

The likelihood function takes the following form

$$\mathcal{L} = \prod_{n=1}^{N} p(x_n),$$

where we specify the model as a sum of Gaussians

$$p(x_n) = \sum_{k=1}^{K} \mathcal{N}(x_n|\mu_k, \Sigma_k)p(k),$$

$$\mathcal{N}(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}\det(\Sigma)^{1/2}} \exp\{-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)\}.$$

To simplify the computations, logarithms are used

$$\log \mathcal{N}(x|\mu, \Sigma) = -\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu) - \frac{d}{2}\log(2\pi) - \frac{1}{2}\log\det(\Sigma).$$

Expectation, or **E-step** (we suppose that we know the model, and we estimate the assignment of individual points):

$$p_{nk} \equiv p(k|n) = \frac{\mathcal{N}(x_n|\mu_k, \Sigma_k)p(k)}{p(x_n)}.$$

Maximization, or **M-step** (we suppose we know the assignment of individual points but we do not know the model):

$$\begin{aligned}
\hat{\mu}_k =& \sum_{n=1}^{N} p_{nk} x_n / \sum_{n=1}^{N} p_{nk}, \\
\hat{\Sigma}_k =& \sum_{n=1}^{N} p_{nk} (x_n - \hat{\mu}_k) \times (x_n - \hat{\mu}_k) / \sum_{n=1}^{N} p_{nk}, \\
\hat{p}(k) =& \frac{1}{N} \sum_{n=1}^{N} p_{nk}.
\end{aligned}$$

It can be proved that alternating E and M steps converges to at least a local maximum of the likelihood (convergence can be slow).

Libraries

You will need to load the following packages:

```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn import mixture
from sklearn.datasets import make_classification
from sklearn.datasets import make_blobs
from sklearn.datasets import make_moons
```

Analysis

First of all, we will do analysis on a simulated one-dimensional data. It will help to understand how the EM algorithm performs optimization. Do not forget that it can be helpful to visualize the obtained clustering if your data are 1- or 2-dimensional.

1. Construct one-dimensional simulated data as follows

```
mu1, sigma1 = 0, 0.3 # mean and standard deviation
s1 = np.random.normal(mu1, sigma1, 100)
y1 = np.repeat(0, 100)

mu2, sigma2 = 2, 0.3 # mean and standard deviation
s2 = np.random.normal(mu2, sigma2, 100)
y2 = np.repeat(1, 100)

mu = [mu1, mu2]
sigma = [sigma1, sigma2]

data = np.concatenate([s1,s2])
y = np.concatenate([y1,y2])
```

2. Here are some necessary functions, and the procedure to run EM in a one-dimensional case

```
def pr_single_comp(mu, sigma, x):
    prob = []
    for i in range(0, x.shape[0])    :
        prob.append(np.exp(-0.5*((x[i,]-mu)/sigma)**2)/sigma)
    return prob
```

```
def pr_single_normalized(mu,sigma, x):
    unnorm_prob = pr_single_comp(mu, sigma, x)
    normalization = np.sum(pr_single_comp(mu, sigma, x), axis=1)
    prob = []
    for i in range(0, len(unnorm_prob)) :
        prob.append(unnorm_prob[:][i]/normalization[i])
    return prob


def update_mu(x,mu,sigma) :
   prob = pr_single_normalized(mu,sigma,x)
   hat_mu = [0, 0]
   for i in range(0, len(prob)) :
       hat_mu = hat_mu + prob[i][:]*x[i,]
   hat_mu = hat_mu/np.sum(pr_single_normalized(mu, sigma, x), axis=0)
   return hat_mu

def update_sigma(x,mu,sigma) :
   prob = pr_single_normalized(mu,sigma,x)
   hat_sigma = [0, 0]
   for i in range(0, len(prob)) :
       hat_sigma = hat_sigma + prob[i][:]*(x[i,] - mu)**2
   hat_sigma = hat_sigma/np.sum(pr_single_normalized(mu, sigma, x), axis=0)
   return hat_sigma

mu_old = [random.uniform(-2, 2), random.uniform(0, 4)]
sigma_old = [0.3, 0.3]
NbIter = 10

# Learning procedure (optimization)
for iter in range(1, NbIter):
    hat_mu = update_mu(data,mu_old,sigma_old)
    hat_sigma = update_sigma(data,mu_old,sigma_old)
    print('iter', iter)
    print('updated mu = ',hat_mu)
    print('updated sigma = ',hat_sigma)
    mu_old = hat_mu
    sigma_old = hat_sigma + 1e-13
```

3. Test the one-dimensional case on the simulated data
4. Generate the simulated two-dimensional data from the TME 2 (blobs, moons, etc.)
5. Provide versions of the functions and of the learning procedure for the two-dimensional case
6. Test your EM code for two-dimensional data on three data sets from the TME 2 (blobs, moons, etc.)
7. Explore the EM from the `sklearn` library

   `http://scikit-learn.org/stable/modules/mixture.html#the-dirichlet-process`

   Since you perform clustering, and we do not know an optimal number of clusters, the BIC score can help to identify a model with an optimal number of components. You can use the following scheme to fix the number of clusters, and to fit a model:

```
lowest_bic = np.infty
bic = []
n_components_range = range(1, 5)
cv_types = ['spherical', 'tied', 'diag', 'full']
for cv_type in cv_types:
    for n_components in n_components_range:
        # Fit a Gaussian mixture with EM
        gmm = mixture.GaussianMixture(n_components=n_components,
         covariance_type=cv_type)
        gmm.fit(X)
        bic.append(gmm.bic(X))
        if bic[-1] < lowest_bic:
            lowest_bic = bic[-1]
            best_gmm = gmm

y_predicted = best_gmm.predict(X)
```

8. Explore what you get with `mixture.GaussianMixture()` on the three artificial data sets from the TME 2

9. Explore what results you get in a high-dimensional problem, i.e. in the Mice task and for the Breast Cancer using the `mixture.GaussianMixture()` from `sklearn`.

10. In what cases the EM provides a reasonable clustering? When the EM does not work well?