Lab 3: Expectation Maximisation

Machine Learning:
Pattern Recognition

In this lab we will build a probabilistic classifier and estimate its parameters. In order to illustrate how the complexity of learning changes when we do not observe all the variables, we will start with the model depicted in figure 1. We will then switch to the graphical model depicted in figure 2 and compare the two models' performance.

Read through the complete exercise handout before starting your implementation: the hints at the end, and the instructions for the subsequent exercises will help you for the current one.

Exercise 1: Gaussian Distribution

In this model, we want to find the posterior probability $p(\mathcal{C}_k|\mathbf{x}_n)$ for all data points x_n. We get this from Bayes' rule

$$p(\mathcal{C}_k|\mathbf{x}_n) = \frac{p(\mathbf{x}_n|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\mathbf{x}_n|\mathcal{C}_j)p(\mathcal{C}_j)}, \tag{1}$$

where we can see from the graph that the data elements are conditionally independent given the class and we model the class-conditional probability with a normal distribution

$$p(\mathbf{x}_n|\mathcal{C}_k) = \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{2}$$

$$= \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp -\frac{1}{2}(\mathbf{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}_n - \boldsymbol{\mu}_k) \tag{3}$$

Load the data in banana.mat, which consists of two classes A and B. Split the data in train and test sets and plot the data, in order to get a feel for it. Just by looking at the data, do you think the model will perform well?

Do the training, which consists of finding the prior probabilities for each class, $p(\mathcal{C}_k)$, and the parameters of the class-conditional distributions, the means $\boldsymbol{\mu}_k$ and covariance matrices $\boldsymbol{\Sigma}_k$. Include how you compute those in your report. Now use Bayes' rule to compute the most likely class label for each data point in the test set. Include the resulting confusion matrix and error rate in you report.

Hint: The probability of a data set X under a Gaussian PDF with mean MU and covariance SIGMA can be computed in Matlab using the function P = mvnpdf(X,MU,SIGMA).

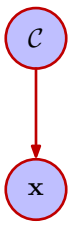

Figure 1: Bayesian classifier with Normally distributed observations. The class labels $\mathcal{C}$ are discrete variables in one-of-K format. The observations $\mathbf{x}$ are multivariate and continuous, and their distribution is described with a Gaussian probability density function.

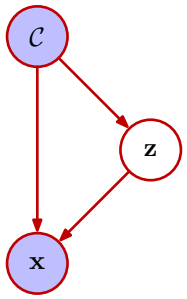

Figure 2: Graphical model of a Bayesian classifier with observations described by a Mixture of Gaussians (MOG). Each class has a number of mixture elements, which are each modelling part of the data. Which datapoints belong to which mixture element is controlled by the latent variable $\mathbf{z}$.

Exercise 2: Mixtures of Gaussians

In this model, the class-conditional distributions $p(\mathbf{x}|\mathcal{C}_k)$ are modelled by a mixture of Gaussians, so that the total data-likelihood

$$p(\mathbf{x}_n, \mathcal{C}_k) = p(\mathcal{C}_k) \sum_j p(\mathbf{x}_n|z_j, \mathcal{C}_k) p(z_j|\mathcal{C}_k), \tag{4}$$

where

$$p(\mathbf{x}_n|z_j, \mathcal{C}_k) = \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk}). \tag{5}$$

Since the latent variables controlling the mixture components are not observed, we need to optimise the log-likelihood with the EM algorithm. You are provided with some code to help you pull this off:

- em_mog(X,C,verbose) implements the iterative loop of the EM algorithm. The parameters are X, the $N \times D$ data matrix where each row is a data element, C the number of mixture components and verbose, a parameter that controls how much debugging information the function should print.

  If verbose is set to 2, the function will additionally plot the data and the contours of the mixture elements as they are updated. This requires the mixture elements to be stored in a cell array with structure elements PI, MU and SIGMA (see line 56 in em_mog.m for more details). If you don't use this (or if you modify the code,) you are not constrained to this format.

- plot_gauss(MU,SIGMA) plots a countour of a Gaussian distribution in two dimensions. It takes two parameters: MU, the mean of the PDF, and SIGMA, its covariance.

The code provided requires a number of functions in order to work, implementing the intialisation, E-step and M-step of the EM algorithm. Those functions are:

- MOG = init_mog(X,C) which creates a structure and fills in the initial parameter values. The parameters are X, the data matrix, and C, the desired number of mixture components.

- [Q LL] = mog_E_step(X,MOG) which implements the E-step of the algorithm. The parameters are X, the data, and MOG the current parameter values. The function does not change the parameters but computes Q, a matrix that contains the probability of each mixture component given the data $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$, and LL, the log-likelihood of the data set under the mixture model.

- MOG = mog_M_step(X,Q,MOG) wich takes as inputs X, the data matrix, Q, the corresponding responsibilities, and MOG the current parameter values. The function returns the updated parameter values.

Use the same data in banana.mat, and implement the required functions listed above. Include pseudo-code in your report, explaining how you compute the responsibilities $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$ and what the updates are that maximise the complete expected log-likelihood.

Optimise the model parameters for the classifier with mixture-of-Gaussians distributions. Use the same train and test sets as for exercise 1 and compare the performance of both classifiers. Discuss the differences you found.

Optional Extra: The contour plots of the distributions do not reflect the priors ($p(\mathbf{z}|\mathcal{C}_l)$) and therefore do not give you an exact idea of the probability density under the model. In order to visualise that, you could plot the probability density under both models in three dimensions.

Hints

1. Include a check in your M-step to avoid singularities. You can check this by computing c = cond(SIGMA): if c becomes larger than a given threshold (say, 10^{10}), do not perform the parameter update.
2. The responsibilities $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$ are stored in the matrix Q in the E-step of the algorithm. You know the size of this matrix in advance: if you create a matrix of zeros of that size and fill in the values, your code will be much more efficient than if your grow the matrix as you compute the values.

Discussion of the Implementation

Discuss the following with your teammate and include your thoughts in your report:

1. What is a sensible way to initialise the parameters of your model?
2. How many components should you try? Having plotted the data, can you guess what number of components should be optimal? How does your classification accuracy evolve as the number of components change?
3. Do the results depend on the initialisation?
4. Why is hint 1 important?

Exercise 3: Log-Probabilities

As we have seen in the class, one problem that occurs often when doing probabilistic modelling, is that since probabilities lay in the range $[0, \ldots, 1]$, the joint probability of multiple variables becomes very small very fast. This is a problem, because computers can only represent numbers with a finite precision and too small a probability will result in an underflow, hence introducing numerical errors in an otherwise mathematically correct algorithm.

The solution that is typically used is then to use the logarithm of probabilities, rather than the probabilities themselves. The products of probabilities then become sums of log-probabilities:

$$\ln[p(a)p(b)] = \ln p(a) + \ln p(b) \tag{6}$$

However a solution must be found when sums are involved. How do we compute

$$\ln[p(a) + p(b)] \tag{7}$$

when $\ln p(a)$ and $\ln p(b)$ are known? The solution is not to do the following:

$$\ln[p(a) + p(b)] = \ln[\exp(\ln p(a)) + \exp(\ln p(b))]. \tag{8}$$

As an example to illustrate this, try to compute $\ln[p(a) + p(b)]$ when $\ln p(a) = -1000$ and $\ln p(b) = -1001$. (Really, do try the above approach before reading on. The correct solution is -999.69.) We've seen that we

can solve this problem as follows:

$$\ln[p(a) + p(b)] = \ln[\exp(\ln p(a)) + \exp(\ln p(b))] \tag{9}$$

$$= \ln\left[\exp(\ln p(a))\left(1 + \frac{\exp(\ln p(b))}{\exp(\ln p(a))}\right)\right] \tag{10}$$

$$= \ln p(a) + \ln[1 + \exp(\ln p(b) - \ln p(a))], \tag{11}$$

which is now easy to compute, because either the difference between $\ln p(a)$ and $\ln p(b)$ is small (and then the exponent can be computed without numerical problems) or it is large (and then the exponent of the difference goes to zero, so that the $\ln(1+0) \to \ln(1) = 0$. You do still need to pay attention that you take the largest value out of the ln, so that the difference goes to large negative values rather than large positive values.

The above discussion is pretty important: as we start using larger data sets, the numerical issues become dominant. In effect, you should never use probabilities in your code and always work with log-probabilities. The products then become sums, and the sums become the function explained above. This has the added advantage that ln and exp are pretty expensive operations. If you use distributions from the exponential family, such as the Gaussian, computing the log-probability is computationally slightly cheaper than computing the probability. A final benefit is that sometimes the code is easier to debug, because it is clear what errors are due to implementation bugs and what errors are due to the limited precision of the machine. Have a look at the lecture notes for more information on this topic.

Assignment

You may have noticed such numerical problems in the previous exercise, especially when using larger numbers of mixture components. In this exercise, we will change the E-step of the previous exercise and compute everything with log-probabilities.

1. First implement a function called logsumexp that computes the sum of two log-probabilities as described above. Make sure that this function takes the largest value out of the logarithm. Write the function so that both scalars and matrices can be used as arguments.

2. Discuss how we can use this function to compute sums of more than two probabilities using log-probabilities.

3. Check your implementation by computing $\ln[p(a) + p(b)]$ when $\ln p(a) = -1000$ and $\ln p(b) = -1001$.

4. Now convert the E-step of the previous exercise. Use the function lmvnpdf(X,MU,SIGMA) which is functionally equivalent to log mvnpdf(X,MU,SIGMA) but never computes the logarithm or exponent and is therefore numerically stable:

$$\mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp -\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu}) \tag{12}$$

$$\ln \mathcal{N}(\mathbf{x}_n; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{1}{2}\left(d \ln 2\pi + \ln|\boldsymbol{\Sigma}| + (\mathbf{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right) \tag{13}$$

Notice, however, that only the function mog_E_step is changed, and that it must still return $p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$, not $\ln p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$. In the adapted function, you compute $\ln p(\mathbf{z}|\mathbf{x}, \boldsymbol{\theta})$, but you need to exponentiate it before returning the value. However now the computation does not suffer from the underflows and is therefore more precise.

Include the pseudocode for logsumexp in your report, and write down what the E-step becomes when using log-probabilities. Discuss how this affects your results.