CSC412 – Assignment #3

Due: March8, 2004 in class
Worth: 10%
Late assignments not accepted.

1 EM Algorithm for Factor Analysis

In this assignment, you will implement the EM algorithm for training a factor analysis model on images of faces. You will train the factor analysis model below using maximum likelihood as a cost function and the EM algorithm as an optimizer. The faces are 28x20 pixel grayscale images, represented as 560-dimensional vectors.

- As a reminder, the factor analysis model is an unsupervised learning model for d-dimensional data $\mathbf{x}$ which uses k continuous latent variables $\mathbf{z}$:

$$\begin{aligned} p(\mathbf{z}) &= N(0, I) \\ p(\mathbf{x}|\mathbf{z}) &= N(\mu + \Lambda\mathbf{z}, \Psi) \\ p(\mathbf{x}) &= \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z} = N(\mu, \Lambda\Lambda^\top + \Psi) \end{aligned}$$

  where Λ is a $d \times k$ factor loading matrix, Ψ is a $d \times d$ diagonal matrix of sensor noise variances, μ is the data offset and I is the identity matrix.

- The maximum likelihood estimate of the offset μ is the sample mean of the data:

$$\mu = \frac{1}{N}\left[\sum_n \mathbf{x}^n\right]$$

- The E-step equations for this model are given by:

$$\begin{aligned} p(\mathbf{z}|\mathbf{x}) &= N(\beta(\mathbf{x} - \mu), I - \beta\Lambda) \\ \beta &= \Lambda^\top(\Psi + \Lambda\Lambda^\top)^{-1} \end{aligned}$$

- While the M-step updates are:

$$\begin{aligned} \Lambda^{new} &= S\beta^\top(I - \beta\Lambda + \beta S\beta^\top)^{-1} \\ \Psi^{new} &= \mathrm{diag}[S - \Lambda\beta S] \end{aligned}$$

  where S is the sample covariance of the data $\left[\sum_n (\mathbf{x}^n - \mu)(\mathbf{x}^n - \mu)^\top\right]/N$.
  (The diag[] operator sets the off-diagonal elements of a matrix to zero.)

- The average log likelihood $\ell = [\sum_n \log p(\mathbf{x}^n|\Lambda, \Psi, \mu)]/N$ is given by:

$$\ell = -\mathrm{Trace}[(\Lambda\Lambda^\top + \Psi)^{-1}S] + \log\det[(\Lambda\Lambda^\top + \Psi)^{-1}] + \text{constant}$$

2 What to do

Using the equations above, the book, the class notes, and the technical report by Zoubin Ghahramani, write code to implement the EM algorithm for maximum likelihood learning of a factor analysis model. In particular:

- Once, at the beginning of the code, compute the sample mean μ and sample covariance S of the data.
- At each iteration compute the matrix $(\Lambda\Lambda^\top + \Psi)^{-1}$ and from that, the matrix β.
- Use β, along with the sample covariance S, to update the factor loading matrix Λ and the diagonal matrix Ψ of sensor noise variances at each iteration.
- Use $(\Lambda\Lambda^\top + \Psi)^{-1}$, along with the sample covariance S to compute the average log likelihood ℓ of the data at each iteration.
- Continue iterating until convergence. You can assess convergence of EM by monitoring the log likelihood and stopping when the fractional change `(newlik-oldlik)/abs(newlik)` is very small (say less than one part in a hundred thousand.) Remember, your log likelihood should never decrease!

Train your model on some training data, and infer the mean of the hidden factors for each training case:

- Using the training data provided in `a3faces.mat`, train a factor analysis model with 2 latent variables (k=2). Restart the training at several random initializations of the parameters, and run EM to convergence each time. Keep the parameter set which gave the best average training likelihood.
- Using the parameters you learned above, for each training data point $\mathbf{x}^n$, compute the mean $\beta\mathbf{x}^n$ of the posterior distribution $p(\mathbf{z}|\mathbf{x}^n, \Lambda, \Psi, \mu)$.

Test your model on some previously unseen data and use it to build a simple face identification system:

- Using the test data in `a3testfaces.mat`, evaluate the log likelihood of each test case under the model you trained. This requires some work, since the formulas above only compute the average log likelihood.
- Set a threshold on test log likelihood equal to the minimum of the training log likelihoods, and use this to classify the test data into two classes.
- Compare your results with the given class labels, and evaluate your test error rate.

3 What to hand in

- A picture of the offset vector μ, the initial values of each column of Λ, the final values of each column of Λ and final values of the diagonal elements of Ψ.
  Each picture should represent the vector as an image, laid out in the same way that the original data was.
  Clearly label each image and indicate the grayscale axis being used in each one. (6 images total).
- A plot of the trajectory of the average training log likelihood for the run that resulted in the best parameters.
  Show iterations of EM horizontally and average log likelihood vertically.
  Clearly indicate the stopping criteria used to assess convergence of EM
- A histogram of the log likelihoods on the training cases.
  A histogram of the log likelihoods on the test cases.
  (Don't use too few bins in the histograms.)
- A 2D scatter plot having z_1 on one axis, z_2 on another axis, and with one point for each posterior mean $p(\mathbf{z}|\mathbf{x}^n)$ after convergence.
- A picture of the data cases which have the smallest, median and largest posterior values on both z_1 and z_2. (In other words, the images corresponding to the points furthest left, right, up, down and in the middle horizontally and vertically on the 2D scatter plot.) (6 plots total, clearly labeled.)
- The test error rate you achieved on the face identification problem.

4 Some hints

- The functions `reshape` and `imagesc` in Matlab are useful for making pictures of vectors. In particular, try this line to display a vector `xx`:
  `imagesc(reshape(xx,20,28)'); colormap gray; axis equal; axis off;`
  Making the window very small on the screen will reduce the aliasing effects, but of course when you print out the picture it will look very pixelated. The `colormap gray` forces Matlab to make the pictures in grayscale. The function `caxis` will tell you the grayscale axis chosen by `imagesc`.
- Try to get the EM algorithm working *without* the Ψ updates at first (ie just update Λ).
  Then when that is working, and your likelihood is always going up, try adding in the updates for Ψ.
- Initialize the model parameters by setting Λ to small random values and setting the diagonal elements of Ψ in some sensible range, for example the diagonal of the sample covariance S of the entire training set.
- Cache the intermediate quantities $(\Lambda\Lambda^\top + \Psi)^{-1}$ and βS so you don't have to recompute them each time the appear in the update or likelihood formulas.
- You might be able to speed up your code even more using the following formula, derived from the matrix inversion lemma:
$$(\Psi + \Lambda\Lambda^\top)^{-1} = \Psi^{-1}(I_x - \Lambda(I_z + \Lambda^\top\Psi^{-1}\Lambda)^{-1}\Lambda^\top\Psi^{-1})$$

5 Optional Stuff

Some fun things you can try, which you won't be graded on:

- Sort the training data according to the values taken by the posterior mean along the first factor z_1. Use a loop to visualize each training case as an image, in this sorted order. (You will find the `drawnow` command helpful inside the loop.) Sort again, but this time according to the values of the posterior mean along the second factor z_2. Cool, huh?
- "Explore" the factors by running a loop that adds a small multiple α of one factor (column of Λ) to the mean vector and displays the resulting vector as an image. Try running α from about -3 to +3 in steps of 0.1. Try this with each factor separately. Each picture you see is a completely new image synthesized by your model. What do you think the factors learned? Cool, huh?