4F10 Prof P Woodland pcw@eng.cam.ac.uk

Solutions to 4F10 Pattern Processing, 2006

1. *Mixture models*

(a) Gaussian mixture distributions can model arbitrary data distributions given enough mixture components. This includes (with diagonal covariance mixture components) muti-modal distributions; correlated distributions and non-symmetric distributions (a diagram can be used to illustrate this to advantage), whereas a full covariance Gaussian can only model Gaussian data but can model correlations.

Generalisation will depend on the number of parameters estimated. The number of parameters needed will be $2DM + M - 1$ where D is the dimensionality of the data and M is the number of mixture components. A full covariance model can model correlations directly (still assumes basic Gaussian structure and symmetric, unimodal). It has a symmetric covariance matrix with $D.(D+1)/2$ so the overall number including the mean is $D.(D+3)/2$. So for for large D (10's or greater) then number of parameters involved can be very high and a Gaussian mixture approach (or feature reduction) is preferred for better modelliing and improved generalisation . [25%]

(b)(i) Log likelihood of data is [10%]

$$l(\boldsymbol{\theta}) = \sum_{k=1}^{N} \ln p(x_k) = \sum_{k=1}^{N} \ln \left[\sum_{m=1}^{M} p(x_k|m) c_m \right]$$

(b)(ii) Due to the form of the log likelihood use the substitution (from Bayes' noting that the c_m is a prior probability)

$$P(m|x_k) = \frac{p(x_k|m)c_m}{p(x_k)}$$

where $P(m|x_k)$ is the posterior probability of mixture component m being associated with vector x_k and the denominator here is given by the probability density of the vector from the entire mixture distribution *i.e.* [20%]

$$p(x_k) = \sum_{m=1}^{M} c_m p(x_k|m)$$

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \theta_m} = \sum_{k=1}^{N} \frac{1}{p(x_k)} \frac{\partial \left[p(x_k|m)c_m\right]}{\partial \theta_m}$$

Now using

$$\frac{\partial \left[\ln p(x_k|m)c_m\right]}{\partial \theta_m} = \frac{1}{p(x_k|m)c_m} \frac{\partial \left[p(x_k|m)c_m\right]}{\partial \theta_m}$$

yields

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \theta_m} = \sum_{k=1}^{n} P(m|x_k) \frac{\partial \left[\ln p(x_k|m) c_m\right]}{\partial \theta_m}$$

(b)(iii)

$$\frac{\partial \left[\ln p(x_k|m) c_m\right]}{\partial \theta_m} = \frac{\partial}{\partial \theta_m} \left[\ln c_m - \frac{1}{2}\ln(2\pi) - \frac{1}{2}\ln(\sigma_m^2) - \frac{\|x_k - \mu_m\|^2}{2\sigma_m^2} \right]$$

For the mean of the m^{th} mixture component

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \mu_m} = \sum_{k=1}^{N} P(m|x_k) \frac{(x_k - \mu_m)}{\sigma_m^2}$$

and

$$\frac{\partial l(\boldsymbol{\theta})}{\partial \sigma_m} = \sum_{k=1}^{N} P(m|x_k) \left[\frac{\|x_k - \mu_m\|^2}{\sigma_m^3} - \frac{1}{\sigma_m} \right]$$

Now for gradient descent on the negative log liklihood (gradient ascent on log likelihood) adds a proportion η of the gradient for the parameter at each iteration.

Gradient descent rules are (e.g. for mean and similarly for the variance) update on iteration $t + 1$ is:

$$\mu_m^{(t+1)} = \mu_m^{(t)} + \eta \sum_{k=1}^{N} P(m|x_k) \frac{(x_k - \mu_m^{(t)})}{\sigma_m^{(t)2}}$$

Note that $P(m|x_k)$ is computed using the paramters $\boldsymbol{\theta}^{(t)}$. [25%]

(b)(iv) Compared to gradient descent EM does not require a learning rate to be set and is guranteed to reach a local minimum (gradient descent may do so). EM can require more iterations for convergence than gradient descent if well tuned. Note that both are iterative schemes and initialisation is important. There is a similar computational load for each per iteration. [20%]

2. *Bayes and error probabilities*

(a) For the two class problem we select the class with the largest posterior probability for minimum error.

$$\frac{P(\omega_1 \mid \boldsymbol{x})}{P(\omega_2 \mid \boldsymbol{x})} \underset{\omega_2}{\overset{\omega_1}{\gtrless}} 1$$

or including the priors explicitly [10%]

$$\frac{p(x|\omega_1)}{p(x|\omega_2)} \underset{\omega_2}{\overset{\omega_1}{\gtrless}} \frac{P(\omega_2)}{P(\omega_1)}$$

(b)

$$\begin{aligned} P(\text{error}) &= P(\boldsymbol{x} \in \mathcal{R}_2, \omega_1) + P(\boldsymbol{x} \in \mathcal{R}_1, \omega_2) \\ &= P(\boldsymbol{x} \in \mathcal{R}_2|\omega_1)P(\omega_1) + P(\boldsymbol{x} \in \mathcal{R}_1|\omega_2)P(\omega_2) \\ &= \int_{\mathcal{R}_2} p(\boldsymbol{x}|\omega_1)P(\omega_1)d\boldsymbol{x} + \int_{\mathcal{R}_1} p(\boldsymbol{x}|\omega_2)P(\omega_2)d\boldsymbol{x} \end{aligned}$$

[15%]

(c)(i) For the boundary points, setting up discriminant functions (log of the pdf) for each class and dropping terms independent of the class we get

$$g_i(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu}_i)'(\boldsymbol{x} - \boldsymbol{\mu}_i)$$

For the boundary $g_1(\boldsymbol{x}) = g_2(\boldsymbol{x})$. This can then be written as [20%]

$$(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)'\boldsymbol{x} = \frac{1}{2}(\boldsymbol{\mu}_1'\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2'\boldsymbol{\mu}_2)$$

(c)(ii) For the identity covariance matrix with equal priors the decision boundary is half way between the class means, and a distance d from either one. Projecting in the direction of the line joining the class means, allows us to compute the probability of error. Note that the errors for each class are equal, and using the fact of identity covariance:

$$P_e = 0.5\int_d^\infty \mathcal{N}(z;0;1)dz + 0.5\int_{-\infty}^{-d} \mathcal{N}(z;0;1)dz$$

or

$$P_e = \int_d^\infty \mathcal{N}(z;0;1)dz$$

by symmetry. This is of the form required where $a = d$. [30%]

(d) Practical reasons: class-conditional pdfs are not of correct form; parameter estimation doesn't yield ideal values (need infinite training set and optimal estimation algorithm in general); and class priors may not be accurate. Since these are all approximated then probability of error than the minimum value theoretically possible. [15%]

3. *MLPs*

(a) Node activation function is $y = \tanh(z)$. Hence taking standard differentials (not in notes ...) [15%]

$$\frac{dy}{dz} = 1 - \tanh^2(z) = 1 - y^2$$

(b) Advantage of *tanh* is due to the fact that its output range is between ± 1 rather than between zero and 1. This tends to lead to faster convergence of back-propagation training when starting from small weights evenly distributed about zero. [10%]

(c) This is from notes but with a different activation function. We would like to minimise the square error between the target of the output, $t(\mathbf{x}_p)$, and the current output value $y(\mathbf{x}_p)$.

In general we can write using the chain rule (for a single input/output pattern):

$$\frac{\partial E}{\partial \tilde{w}_{ij}^{(k)}} = \frac{\partial E}{\partial z_i^{(k)}} \frac{\partial z_i^{(k)}}{\partial \tilde{w}_{ij}^{(k)}}$$

Now, for the output layer, let y_i be the output of node i and z_i the input to the activation function

$$\frac{\partial E}{\partial z_i} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_i}$$

and since

$$\frac{\partial E}{\partial y_i} = y_i - t_i(\boldsymbol{x}_p)$$

and the activation function is tanh we have [15%]

$$\frac{\partial E}{\partial \tilde{w}_{ij}^{(k)}} = (y_i - t_i(\boldsymbol{x}_p)).(1 - y_i^2)$$

(d) In general for any weight in the network, required to calculate $\frac{\partial E}{\partial \tilde{w}_{ij}^{(k)}}$ Applying the chain rule

$$\frac{\partial E}{\partial \tilde{w}_{ij}^{(k)}} = \frac{\partial E}{\partial z_i^{(k)}} \frac{\partial z_i^{(k)}}{\partial \tilde{w}_{ij}^{(k)}} = \delta_i^{(k)} \tilde{x}_j^{(k)}$$

where

$$\frac{\partial E}{\partial z_i^{(k)}} = \delta_i^{(k)}$$

and the δ's are sometimes known as the individual "errors" (that are back-propagated).

For the output nodes the evaluation of δ_i is straightforward as shown above.

To evaluate the δ_i's for hidden layers

$$\delta_i^{(k)} = \sum_m \left[\frac{\partial E}{\partial z_m^{(k+1)}} \frac{\partial z_m^{(k+1)}}{\partial z_i^{(k)}} \right]$$

where it is assumed that only the units in layer $k+1$ are connected to units in layer k, or [25%]

$$\delta_i^{(k)} = (1 - (y_i^{(k)})^2) \sum_m \tilde{w}_{mi}^{(k+1)} \delta_m^{(k+1)}$$

(e)(i) Here we get the usual problems with gradient descent in that we get overshoot if the learning rate is too large and very slow learning if it is too small. Note that the rate will also be dependent on the number of patterns before an update is computed. [10%]

(e)(ii) For batch training sum the error derivatives over all the training patterns. This leads to an accurate determination of the error derivative and can be used with small problems (together with momentum terms and approximate second order methods). Batch update cannot be used for the largest problems/networks since updates through the entire training set will be too infrequent for reasonable convergence. So sometimes an intermediate between batch and single sample is used that can use momentum terms and adaptive learning rates. However note that updating every pattern saves memory by not requiring the accumulation of all the error derivatives and can be optimised by maths library routines (Note this answer contains information somewhat beyond the scope of lectures). [20%]

4. *PCA and Kernel PCA*

(a) PCA is based on the assumption that the directions with the greatest variance are the most "important" directions. The directions are all required to be orthogonal to one another. PCA decorrelates the elements of the feature vector as well discarding those directions with little variance. [20%]

(b) First obtain the eigenvectors and eigenvalues. Only need to consider the top 2 dimensions (the third dimensions is independent of the first 2).

$$(\lambda - 5)(\lambda - 5) - 9 = 0$$
$$(\lambda - 8)(\lambda - 2) = 0$$

Find the eigenvector associated with the first eigenvalue.

$$\begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 8 \begin{bmatrix} x \\ y \end{bmatrix}$$

The unit eigenvector is then given by $[1,1]/\sqrt{2}$. The variance in this direction is 8. As the third direction has variance of 3, the two principal components are:

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

The observation vectors are then dotted with this principal directions to get the transformed vectors. The associated covariance matrix with this data is

$$\Sigma = \begin{bmatrix} 8 & 0 \\ 0 & 3 \end{bmatrix}$$

[35%]

(c)(i) Standard PCA is restricted to performing linear projections of the data. In contrast if a kernelised version of PCA is used then non-linear projections of the data can be extracted. [15%]

(c)(ii) For a Gaussian kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{\sigma^2}\right)$$

where σ^2 determines the kernel width. The relationship to the feature-space is

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{\Phi}(\mathbf{x}_i).\mathbf{\Phi}(\mathbf{x}_j)$$

[15%]

(c)(ii) To obtain the transformed features for point $\mathbf{x}$ in the k^{th} principal direction is

$$\begin{aligned} \mathbf{v}_k.\mathbf{\Phi}(\mathbf{x}) &= \sum_{i=1}^{m} \alpha_i^{(k)} \mathbf{\Phi}(\mathbf{x}_i).\mathbf{\Phi}(\mathbf{x}) \\ &= \sum_{i=1}^{m} \alpha_i^{(k)} k(\mathbf{x}_i, \mathbf{x}) \end{aligned}$$

The advantage of using a kernel is that the dimensionality of the point in the feature-space can be very large. Using a kernel allows the dot-product in the feature-space to be represented in terms of operations in the input-space. [15%]

5. *Bayesian Networks*

(a) The probability of an earthquake being reported on the radio is dependent on whether an earthquake has occurred. The probability of the house alarm going-off is dependent on whether an earthquake has occurred and whether a burglar has entered the house. The probability of neighbour phoning the owner is dependent on whether the alarm has gone off.

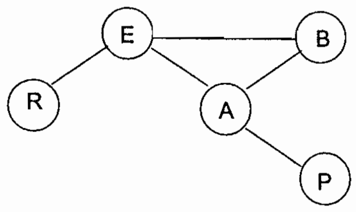
Expressing the joint distribution using the conditional independence assumptions

$$P(R, E, A, B, P) = P(E)P(B)P(R|E)P(A|B, E)P(P|A)$$

[25%]

(b)(i) Message passing allows inference to be carried out by local calculations along with messages passed between the nodes. [10%]

(b)(ii) To generate the moral graph associated with the network requires connecting the parents



There are 3 cliques associated with this BN

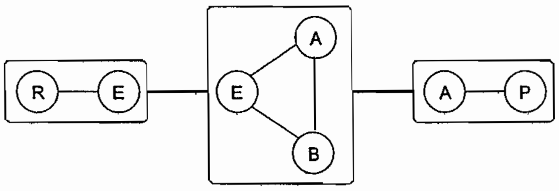
$$\begin{aligned}\mathcal{C}_1 &= \{R,E\}\\ \mathcal{C}_2 &= \{A,B,E\}\\ \mathcal{C}_3 &= \{A,P\}\end{aligned}$$

Each of these are fully connected subsets of all the variables. The separators are the *intersection* of the cliques.

$$\begin{aligned}\mathcal{S}_{12} &= \mathcal{C}_1 \cap \mathcal{C}_2 = \{E\}\\ \mathcal{S}_{23} &= \mathcal{C}_2 \cap \mathcal{C}_3 = \{A\}\end{aligned}$$

[20%]

(b)(iii) The junction tree showing the connections of the cliques is shown below.



The messages are then

$$\begin{aligned}\phi_{12}(\mathcal{S}_{12}) &= \phi_{12}(E) = \sum_R P(\mathcal{C}_1)\\ \phi_{23}(\mathcal{S}_{23}) &= \phi_{23}(A) = \sum_{E,B} P(\mathcal{C}_2)\end{aligned}$$

[20%]

(c) As whether the alarm is set-up is not observed it is necessary to use EM (other schemes could be used, but this is the one mentioned in lectures). To train the system the value of A is set-up as a latent variable. An auxiliary function is then set-up of the form

$$\mathcal{Q}(\boldsymbol{\theta}^{(k)}, \boldsymbol{\theta}^{(k+1)}) = \sum_A \sum_B P(A|R,E,B,P,\boldsymbol{\theta}^{(k)}) \log \left(P(A,R,E,B,P|\boldsymbol{\theta}^{(k+1)})\right)$$

where $\boldsymbol{\theta}^{(k)}$ are the model parameter estimates at the k^{th} iteration. This expression is then iteratively optimised. [25%]