Solutions to 4F10 Pattern Processing, 2004

1. *Bayes decision rule and GMMs*

   (a) For multi-class problems, we calculate all the C posterior probabilities

   $$P(\omega_1|\boldsymbol{x}), P(\omega_2|\boldsymbol{x}), ..., \ P(\omega_C|\boldsymbol{x}),$$

   find the maximum of these and assign the vector $\boldsymbol{x}$ to the corresponding class.

   Applying Bayes' Rule to this formula we need to find the class ω_j which gives

   $$\max_j p(\boldsymbol{x}|\omega_j)P(\omega_j)$$

   since the rhs denominator of Bayes' rule is independent of class and this is a frequent statement of Bayes' decision rule for minimum error.

   The key issues in approximating this calssifier is how well we can *estimate* from training data the form and parameters of both the class-conditional pdfs and the priors from training data. If there is a very large training set and a relatively few parameters to estimate and the underlying class conditional distributions are very close to e.g. Gaussian then we may be able to approximate the Bayes' classifier and the minimum error rate well. However, the usual case is that there is insufficient training data to estimate a large number of parameters and the form of the underlying pdfs may be complex, and hence the approximation may be poor. [15%]

   (b) 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). Here we are told that the data is close to Gaussian so it is probably not multi-modal and is roughly symmetric.

   For a GMM, 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 this is not so powerful and for large D (certainly for $D = 20$ or greater) then number of parameters involved can be very high and a Gaussian mixture approach (or feature reduction) is preferred, since the number of components to model the correlation structure may be rather smaller than to make up for the extra parameters a full covariance matrix.

   Computation is roughly in line with the number of parameters, although a GMM of diagonal Gaussians are slightly higher computationally per parameter, but can more

easily be approximated by not computing all the Gaussian components [latter poiunt not discussed in the course]. [20%]

(c) When E-M is used on each iteration the maximisation will guarantee an increase in the log likelihood of the data unless it is at a local maximum. Hence a graph of the log likelihood against iteration will monotonicly increase and converge. Furthermore no information about previous iterations updates is necessary (unlike the use of a momentum term).

For gradient-descent (this would be on the negative of the log likelihood function), the the step size for all parameters has to be explicitly set and there is no guarantee that the likelihood will increase. For stability need to use a small step size. The training procedure can be improved by the use of information from previous iterations or second order methods but these require more statistics to be computed than for E-M. Hence E-M is preferred for problems to which it can be applied.

Note that both are susceptible to local maxima in the log likelihood function. [10%]

(d)(i) The log likelihood function for the data

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

where the dependence on the pdf for mixture component m is explicit.

This is for any mixture model. For Gaussians, simply substitute in a Gaussian pdf expression for $p(\boldsymbol{x}_k|m)$. For instance, if the Gaussian distribution has the form $\boldsymbol{\Sigma}_m = \sigma_m^2 \mathbf{I}$ (as in lectures), then

$$l(\boldsymbol{\theta}) = \sum_{k=1}^{n} \ln \left[\sum_{m=1}^{M} c_m \frac{1}{(2\pi\sigma_m^2)^{d/2}} exp \left\{ \frac{-\|\boldsymbol{x}_k - \boldsymbol{\mu}_m\|^2}{2\sigma_m^2} \right\} \right]$$

Now, it is necessary to find the partial derivative of $l(\boldsymbol{\theta})$ with respect to the parameters of the mixture distribution. [10%]

(d)(ii) First, we note that from Bayes' since the c_m is a prior probability:

$$P(m|\boldsymbol{x}_k) = \frac{p(\boldsymbol{x}_k|m)c_m}{p(\boldsymbol{x}_k)}$$

For a particular mixture component weight c_m

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

or

$$\frac{\partial l(\boldsymbol{\theta})}{\partial c_m} = \sum_{k=1}^{n} \frac{p(\boldsymbol{x}_k|m}{p(\boldsymbol{x}_k)}$$

or

$$\frac{\partial l(\boldsymbol{\theta})}{\partial c_m} = \sum_{k=1}^{n} \frac{P(m|\boldsymbol{x}_k)}{c_m}$$

[15%]

(d)(iii) The log likelihood derivative is given above. However here we need to go back to use Lagrange multipliers to enforce the constraint.

In this case add $\lambda(\sum_{m=1}^{M} c_m - 1)$ to the log likelihood function and repeat as above and equate to zero for a maximum.

$$\frac{\partial \left(l(\boldsymbol{\theta}) + \lambda(\sum_{m=1}^{M} c_m - 1) \right)}{\partial c_m} = \sum_{k=1}^{n} \frac{P(m|\boldsymbol{x}_k)}{c_m} - \lambda$$

This implies that at the required maximum (equating to zero)

$$\hat{c}_m = \frac{1}{\lambda} \sum_{k=1}^{n} P(m|\boldsymbol{x}_k)$$

The constraint that $\sum_{m=1}^{M} c_m = 1$ gives

$$\begin{aligned} \sum_{m=1}^{M} \frac{1}{\lambda} \sum_{k=1}^{n} P(m|\boldsymbol{x}_k) &= 1 \\ \lambda &= \sum_{k=1}^{n} \sum_{m=1}^{M} P(m|\boldsymbol{x}_k) \\ &= n \end{aligned}$$

so

$$\hat{c}_m = \frac{1}{n} \sum_{k=1}^{n} P(m|\boldsymbol{x}_k)$$

[15%]

(d)(iv) The problem with direct application of this formula is that the values of the posterior $P(m|\boldsymbol{x}_k)$ depend on the component priors. However it can be used iteratively. On each iteration use the old values of the component priors to compute the posteriors and then given those values update the component priors and any other parameters of interest (not here since the means and covariances are known).

This is application of the E-M algorithm which has two stages, the first is to find the expected value of the "missing" variable. In the case of the Gaussian mixture parameters this is the mixture component that is occupied for data sample $\boldsymbol{x}_k$ i.e. the value of mixture component occupation or $P(m|\boldsymbol{x}_k)$. Given the complete data set the maximisation of the auxiliary function gives the update above. [15%]

2. *Multi-layer perceptrons*

(a) (i) The number of hidden layers determines the decision boundaries that can be generated. In choosing the number of hidden layers it should be noted that multi-layer networks are harder to train than single layer networks and that a two layer network (one hidden) with sigmoidal activation functions can model any decision boundary.

Therefore if a problem is linearly separable (rare for real problems), choose a single layer $L = 0$ network. Two layer networks $L = 1$ are most commonly used in pattern recognition (the hidden layer having sigmoidal activation functions). The number of nodes in the hidden layer needs to be great enough to be able to construct the decision boundaries for the problem but limited otherwise the number of weights will be too large to estimate reliably and performance on test-data (i.e. generalaisation) will suffer.

For classification tasks the input has the number of inputs equal to the dimensionality of the data, and the output layer normally has as many nodes as classes (1-in-C coding). [15%]

(a)(ii) The weight matrices including the biases are simply $N^{(k)} \times (N^{(k-1)} + 1)$ matrices. So we have in total

$$\sum_{k=1}^{k=L+1} N^{(k)} \times (N^{(k-1)} + 1)$$

[10%]

(b)(i)

$$y = \phi(z) = \frac{1}{1 + \exp(-z)}, \quad \frac{\partial \phi(z)}{\partial z} = \frac{\exp(-z)}{(1 + \exp(-z))^2} = \phi(z)(1 - \phi(z))$$

We would like to minimise the square error between the target of the output, $t(\mathbf{x}_p)$, and the current output value $y(\mathbf{x}_p)$.

$$E = \frac{1}{2} \sum_{p=1}^{n} (y(\mathbf{x}_p) - t(\mathbf{x}_p)'(y(\mathbf{x}_p) - t(\mathbf{x}_p)) = \sum_{p=1}^{n} E^{(p)}$$

In general we can write using the chain rule (for a single input/ouput 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

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

and the activation function is a sigmoid we have

$$\frac{\partial E}{\partial \tilde{w}_{ij}^{(k)}} = (y_i - t_i(\mathbf{x}_p)).y_i(1-y_i)x_{pj}^h$$

where x_{pj}^h is the j^{th} element of the output from the final hidden layer. [25%]

(b)(ii) The above was for one pattern. Normal batch style training first sums the contributions to the partial derivatives of E wrt the weights over all patterns to get ∇E.

$$w_{ij}^{(k)}[\tau+1] = w_{ij}^{(k)}[\tau] - \eta\frac{\partial E[\tau]}{\partial w_i^{(k)}}$$

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.

A simple approach to setting η is to make it adaptive. One of the simplest rules is to use

$$\eta[\tau+1] = \begin{cases} 1.1\eta[\tau]; & \text{if } E(\boldsymbol{\theta}[\tau]) < E(\boldsymbol{\theta}[\tau-1]) \\ 0.5\eta[\tau]; & \text{if } E(\boldsymbol{\theta}[\tau]) > E(\boldsymbol{\theta}[\tau-1]) \end{cases}$$

In words: if the previous value of $\eta[\tau]$ decreased the value of the cost function, then increase $\eta[\tau]$. [15%]

(b)(iii) The addition of a momentum term to the optimisation is common in MLP training. The update formula is

$$\tilde{\mathbf{w}}_i^{(k)}[\tau+1] = \tilde{\mathbf{w}}_i^{(k)}[\tau] + \boldsymbol{\Delta}\tilde{\mathbf{w}}_i^{(k)}[\tau]$$

where

$$\boldsymbol{\Delta}\tilde{\mathbf{w}}_i^{(k)}[\tau] = -\eta[\tau+1]\left.\frac{\partial E}{\partial \tilde{\mathbf{w}}_i^{(k)}}\right|_{\boldsymbol{\theta}[\tau]} + \alpha[\tau]\boldsymbol{\Delta}\tilde{\mathbf{w}}_i^{(k)}[\tau-1]$$

The use of the momentum term, $\alpha[\tau]$:

- smooths successive updates;
- helps avoid small local minima.

Unfortunately it introduces an additional tunable parameter to set. Also if we are lucky and hit the minimum solution we will overshoot. [15%]

(c) For the softmax

$$y_j = \frac{\exp(z_j)}{\sum_{i=1}^{n} \exp(z_i)}$$

The output is positive and the sum of all the outputs at the current level is 1, $0 \leq y_j \leq 1$.

Now the key issue here is that in calculating the derivatives for the softmax function it is necessary to conider **all** of the inputs to the the final layer activation and not just the input to a particular node.

Hence

$$\frac{\partial E}{\partial z_i} = \sum_q \frac{\partial E}{\partial y_q} \frac{\partial y_i}{\partial z_q}$$

The substitutions from above carry on as before with this modification. This increases the complexity of computing the derivatives for the final layer weights. [20%]

3. *Support Vector Machines and Kernels*

(a) The training criterion for the perceptron algorithm is the sum of the perpendicular distance from the decision boundary of all miclassified points. There is no unique solution to this cost function. For the SVM classifier the cost function is to maximum the margin, i.e.

$$\max_{\mathbf{w},b} \min \{||\mathbf{x} - \mathbf{x}_i||; \langle \mathbf{w}, \mathbf{x} \rangle + b = 0, i = 1, \ldots, m\}$$

This has the dual form which required minimising

$$\min_{\mathbf{w},b} \tau(\mathbf{w}) = \frac{1}{2}||\mathbf{w}||^2$$

subject to

$$y_i \left(\langle \mathbf{w}, \mathbf{x}_i \rangle + b \right) \geq 1$$

for all $i = 1, \ldots, m$.

When the data is not linearly separable *slack variables* must be introduced. The soft-margin classifier is obtained by minimising

$$\tau(\mathbf{w}, \boldsymbol{\xi}) = \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{m}\xi_i$$

subject to

$$\begin{aligned} y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right) &\geq 1 - \xi_i \\ \xi_i &\geq 0 \end{aligned}$$

[25%]

(b) Kernels map from the input-space to a higher dimensional feature-space. A linear classifier is then built in this feature space. This results in a non-linear decision boundary in the original feature space. The general form for the polynomial kernel is

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i.\mathbf{x}_j + 1)^d$$

[15%]

(c)(i) The kernel function is

$$\begin{aligned} k(x_i, x_j) &= \frac{1}{2} + \sum_{r=1}^{N}\cos(rx_i)\cos(rx_j) + \sin(rx_i)\sin(rx_j) \\ &= -\frac{1}{2} + \sum_{r=0}^{N}\cos(r(x_i - x_j)) \\ &= -\frac{1}{2} + \mathrm{Re}\left(\sum_{r=0}^{N}\cos(r(x_i - x_j))\right) \\ &= -\frac{1}{2} + \mathrm{Re}\left(\frac{1 - e^{i(N+1)(x_i - x_j)}}{1 - e^{i(x_i - x_j)}}\right) \end{aligned}$$

Examining the denominator

$$1 - e^{i(x_i - x_j)} = -2ie^{i(x_i - x_j)/2}\sin((x_i - x_j)/2)$$

and noting that

$$\mathrm{Re}\left(\frac{1}{ie^{i(x_i - x_j)/2}}\left(1 - e^{i(N+1)(x_i - x_j)}\right)\right) = -\sin((x_i - x_j)/2) - \sin((N + 1/2)(x_i - x_j))$$

It is then straight forward to show that

$$k(x_i, x_j) = \frac{\sin((N + 1/2)(x_i - x_j))}{2\sin((x_i - x_j)/2)}$$

[25%]

(c)(ii) The kernelised version of the rule is

$$g_i(x) = \sum_{i=1}^{m} y_i \alpha_i k(x, x_i) + b$$

Once the number of support vectors are known then the computational cost is independent of the number of training samples and N. The cost scales linearly with the number of support vectors. [15%]

(d) From the lecture notes Kesler's construct was mentioned. The multi-class problem can be converted to a 2-class problem. Consider an extended observation $\tilde{\mathbf{x}}$ which belongs to class ω_1. Then to be correctly classified

$$\tilde{\mathbf{w}}_1' \tilde{\mathbf{x}} - \tilde{\mathbf{w}}_j' \tilde{\mathbf{x}} > 0, \quad j = 2, \ldots, K$$

There are therefore $K-1$ inequalities requiring that the $K(d+1)$-dimensional vector

$$\boldsymbol{\alpha} = \begin{bmatrix} \tilde{\mathbf{w}}_1 \\ \vdots \\ \tilde{\mathbf{w}}_K \end{bmatrix}$$

correctly classifies all $K-1$ set of $K(d+1)$-dimensional samples

$$\boldsymbol{\gamma}_{12} = \begin{bmatrix} \tilde{\mathbf{x}} \\ -\tilde{\mathbf{x}} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \quad \boldsymbol{\gamma}_{13} = \begin{bmatrix} \tilde{\mathbf{x}} \\ \mathbf{0} \\ -\tilde{\mathbf{x}} \\ \vdots \\ \mathbf{0} \end{bmatrix}, \quad \ldots, \quad \boldsymbol{\gamma}_{1K} = \begin{bmatrix} \tilde{\mathbf{x}} \\ \mathbf{0} \\ \mathbf{0} \\ \vdots \\ -\tilde{\mathbf{x}} \end{bmatrix}$$

[20%]

4. *Non-Parametric Classification - Parzen Window*

(a) Parzen window density estimates make no assumptions about the form of the desnity. In contrast a Gaussian distribution assumes that the form of the density is known, so only the mean and variance are required. The Parzen window requires the storage of all points and a window function computed for each of those points. In contrast the Gaussian needs only the mean and covariance matrix to be computed. The evaluation of the PDF is then very quick, depending on the dimensionality of the training data, not the number of training samples. [20%]

(b)(i) The value of h detrmines the smoothness of the probability density estimate. The value of h should vary inversly to the number of points a typical form is

$$h_n = \frac{h}{\sqrt{n}}$$

[15%]

(b)(ii) The window function is valid PDF so

$$\int_{\mathcal{R}^d} \phi(\mathbf{x}) d\mathbf{x}; \quad \phi(\mathbf{x}) \geq 0$$

By simple analogy with the hypercube (or consider scaling each dimension)

$$\int_{\mathcal{R}^d} \phi(\frac{\mathbf{x}}{h}) d\mathbf{x} = h^d$$

Hence

$$\begin{aligned} \int_{\mathcal{R}^d} \tilde{p}(\mathbf{x}) d\mathbf{x} &= \int_{\mathcal{R}^d} \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^d} \phi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) d\mathbf{x} \\ &= \frac{1}{n} \sum_{i=1}^{n} \frac{1}{h^d} \int_{\mathcal{R}^d} \phi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) d\mathbf{x} \\ &= 1 \end{aligned}$$

[15%]

(c)(i) The form of Gaussian window function and the first order Taylor series expansion is

$$\begin{aligned} \phi\left(\frac{x - x_i}{h}\right) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(x - x_i)^2}{2h^2}\right) \\ &\approx \frac{1}{\sqrt{2\pi}} \left(1 - \frac{(x - x_i)^2}{2h^2}\right) \end{aligned}$$

The approximate Parzen window is then

$$\begin{aligned} \tilde{p}(x) &\approx \frac{1}{hn} \sum_{i=1}^{n} \left(\frac{1}{\sqrt{2\pi}} \left(1 - \frac{(x - x_i)^2}{2h^2}\right)\right) \\ &= \frac{1}{hn\sqrt{2\pi}} \sum_{i=1}^{n} \left(1 - \frac{x^2}{2h^2} + \frac{x x_i}{h^2} - \frac{x_i^2}{2h^2}\right) \end{aligned}$$

[25%]

(c)(ii) It is only necessary to store the values of b_0, b_1 and b_2. The Parzen window approximation is then simply computed as calculating the weighted sum for the Taylor series. [10%]

(c)(iii) The approximation requires that

$$\frac{x - x_i}{h} << 1$$

This requires that value of h should be large. However this results in a poor approximation for the true distribution $p(x)$. To overcome this problem the number of elements of the Taylor series expansion can be increased. [15%]