Solutions to 4F10 Statistical Pattern Processing, 2013

1. *Mixture models*

   (a) Gaussian mixture distributions can model arbitrary data distributions given enough mixture components. This includes (with diagonal covariance mixture components) multi-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't 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

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ln p(x_k) = \sum_{i=1}^{n} \ln \left[\sum_{m=1}^{M} p(x_i|\omega_m) c_m \right]$$

   [10%]

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

$$P(\omega_m|x_i) = \frac{p(x_i|\omega_m)c_m}{p(x_i)}$$

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

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

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_m} = \sum_{i=1}^{n} \frac{1}{p(x_i)} \frac{\partial \left[p(x_i|\omega_m) c_m \right]}{\partial \theta_m}$$

   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

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \theta_m} = \sum_{i=1}^{n} P(m|x_i) \frac{\partial \left[\ln p(x_i|\omega_m) c_m \right]}{\partial \theta_m}$$

(b)(iii)

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

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

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mu_m} = \sum_{i=1}^{n} P(\omega_m|x_i)\frac{(x_i - \mu_m)}{\sigma_m^2}$$

and

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \sigma_m} = \sum_{i=1}^{n} P(\omega_m|x_i)\left[\frac{\|x_i - \mu_m\|^2}{\sigma_m^3} - \frac{1}{\sigma_m}\right]$$

Now for gradient descent on the negative log likelihood (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:

$$\mu_m^{(t+1)} = \mu_m^{(t)} + \eta \sum_{i=1}^{n} P(\omega_m|x_k)\frac{(x_i - \mu_m^{(t)})}{{\sigma_m^{(t)}}^2}$$

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

(b)(iv) Compared to gradient descent EM does not require a learning rate to be set and is guaranteed to reach a local minimum (gradient descent may do so). It does this by optimising an auxiliary function of the log likelihood that has a unique global maximum on each iteration. 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. *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 [20%]

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

(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 normally an intermediate between batch and single sample is used (a mini-batch) 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. [20%]

3. *Classifiers, Risk and SVMs*

(a) For a classifier, $f(\boldsymbol{x}, \boldsymbol{\theta})$, trained with parameters $\boldsymbol{\theta}$, the expected test error may be expressed as

$$R(\boldsymbol{\theta}) = \sum_y \int \frac{1}{2} |y - f(\boldsymbol{x}, \boldsymbol{\theta})| \, p(\boldsymbol{x}, y) d\boldsymbol{x}$$

$R(\boldsymbol{\theta})$ is called the *expected risk* [10%]

(b) Normally it is not possible to find $p(\boldsymbol{x}, y)$, so the *empirical risk* $R_{\text{emp}}(\boldsymbol{\theta})$ is used

$$R_{\text{emp}}(\boldsymbol{\theta}) = \frac{1}{2m} \sum_{i=1}^{m} |y_i - f(\boldsymbol{x}_i, \boldsymbol{\theta})|$$

It can be derived from the expected risk by replacing the density $p(x, y)$ by the empirical distribution defined by the training samples. [20%]

(c) For this problem $f(x, \theta) \in \{-1, 1\}$ and

$$\begin{aligned} R(\boldsymbol{\theta}) &= \int \frac{1}{2} |1 - f(\boldsymbol{x}, \boldsymbol{\theta})| \, p(\boldsymbol{x}, y = 1) d\boldsymbol{x} + \int \frac{1}{2} |-1 - f(\boldsymbol{x}, \boldsymbol{\theta})| \, p(\boldsymbol{x}, y = -1) d\boldsymbol{x} \\ &= \int_{\{x: f(x,\theta) = -1\}} p(\boldsymbol{x}, y = 1) d\boldsymbol{x} + \int_{\{x: f(x,\theta) = 1\}} p(\boldsymbol{x}, y = -1) d\boldsymbol{x} \\ &= P(f(\mathbf{x}, \theta) \neq \mathbf{y}) \end{aligned}$$

[10%]

(d)(i) The SVM is a linear classifier with with direction w and bias b. For the binary (two class) problem, the training samples are correctly classified if [20%]

$$y_i(\langle w, x \rangle + b) \geq 0$$

(d)(ii)To allow for training data errors, *slack variables*, $\xi_i \geq 0$, are introduced. This relaxes the previous constraint to be

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

All the points that lie outside the required margin (of +/-1) will have $\xi = 0$. For a classification error of the training data to occur $\xi > 1$, so a simple upper bound on the training errors is given by [20%]

$$N \text{ error} \leq \sum_{i=1}^{m} \xi_i$$

(d)(iii) There are now two aspects of the optimisation: maximise the margin; minimise the number of training errors. The balance of the two is commonly controlled by a variable, C. The soft-margin classifier is obtained by minimising

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

subject to

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

The dual optimisation problem for this may also be set-up, now

$$\max_{\boldsymbol{\alpha}} W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$$

subject to $0 \leq \alpha_i \leq C$ and $\sum_{i=1}^{m} \alpha_i y_i = 0$. For correctly classified points that lie on the margin, $0 \leq \alpha_i \leq C$ and $\xi_i = 0$. To determine b, pick a correctly classified point on the margin and solve $\alpha_i(u_i \langle w, x_i \rangle + b) = 0$. [20%]

4. *Decision Trees*

(a) A non-metric classifier can use properties of the data that do not have an implicit ordering, such as colour or type (e.g. type of fruit). These are useful when there is *nominal* data in addition to the numerical data. For instance descriptions that are discrete and there is no natural concept of ordering. Instances of fruit labelled by type and by weight are examples. [20%]

(b)(i) To assess the purity of the nodes a *node impurity* function at a particular node N, $\mathcal{I}(N)$ is defined

$$\mathcal{I}(N) = \phi(P(\omega_1|N), \ldots, P(\omega_K|N))$$

where there are K classification classes. The function $\phi()$ should have the following properties

- $\phi()$ is a maximum when $P(\omega_i) = 1/K$ for all i
- $\phi()$ is at a minimum when $P(\omega_i) = 1$ and $P(\omega_j) = 0, j \neq i$.

- It is symmetric function (i.e. the order of the class probabilities doesn't matter). [20%]

(b)(ii) The entropy purity function for a node N has the form

$$\mathcal{I}(N) = -\sum_{i=1}^{K} P(\omega_i|N) \log(P(\omega_i|N)) \; .$$

Calculation is straightforward, with $P(w_i|N)$ is the sample distribution of class w_i in the data at node N. The entropy satisfies all the properties of the purity function: entropy is maximum for a K-class problem if $P(w_i) = 1/K$ for all i; entropy is at a minimum if $P(w_i) = 1$ and $P(w_j) = 0$, $j \neq i$, assuming $0 \log 0 = 0$; the entropy is symmetric (because the summation is symmetric). . [20%]

(b)(iii) One way to use the purity when constructing the tree is to split each node according to the *question* that maximises the increase in purity (or minimises the impurity). The drop in impurity for a binary split is

$$\Delta\mathcal{I}(N) = \mathcal{I}(N) - \frac{n_{\mathtt{L}}}{n_{\mathtt{L}} + n_{\mathtt{R}}}\mathcal{I}(N_{\mathtt{L}}) - \frac{n_{\mathtt{R}}}{n_{\mathtt{L}} + n_{\mathtt{R}}}\mathcal{I}(N_{\mathtt{R}})$$

where:

- $N_{\mathtt{L}}$ and $N_{\mathtt{R}}$ are the left and right *descendants*;
- $n_{\mathtt{L}}$ is the *number* of the samples in the left descendant;
- $n_{\mathtt{R}}$ is the *number* of the samples in the right descendant;

A threshold is often used, so that the tree is grown until the reduction in the impurity measure fails to exceed some specified minimum. For example, we stop splitting a node N when $\Delta\mathcal{I}(N) = \mathcal{I}(N) \leq \beta$ for a threshold β. [20%]

(c) The purity function $\phi(N)$ at a node N is the least squares cost function based on the best linear regression function that can be built for that data. At node N, let $X_N = [x_1, \ldots, x_n]$ be the training observations and let $y_N = [y_1, \ldots, y_n]'$ be the real output values. The least squares cost function over the data at the N as

$$\phi(N) = \min_{w}\{\sum_{i=1}^{n}(y_i - w'x_i)^2\}$$

for which the optimum linear regression function $f(x) = \hat{w}'_N x$ can be found as

$$\hat{w}_N = (\boldsymbol{X}_N X'_N)'\boldsymbol{X}_N y_N$$

(Lecture 10)

The best question to split each node is chosen is chosen based on $\Delta\mathcal{I}(N) = \mathcal{I}(N)$ as described in the previous section, with $\phi()$ as the least squares cost. If the node is to be split, the best split is chosen, and the regression function is calculated for each child using the formula given. [20%]

5. *Parzen Window estimates*

(a) The window function is a valid PDF so

$$\int_R \phi(x)dx = 1\,;\ \phi(x) = 0$$

and

$$\int_R \phi(\frac{x}{h})dx = h$$

Hence [15%]

$$\begin{aligned}\int_R \tilde{p}(x)dx &= \int_R \frac{1}{h}\phi\left(\frac{x-x_i}{h}\right)dx \\ &= \frac{1}{n}\sum_{i=1}^{n}\int_R \frac{1}{h}\phi\left(\frac{x-x_i}{h}\right)dx = 1\end{aligned}$$

(b) The value of h determines the smoothness of the probability density estimate. The value of h should vary inversely to the number of points; a typical form is $h_n = \frac{h}{\sqrt{n}}$. [15%]

(c)

$$F_{\tilde{p}}(a) = \int_{-\infty}^{a} \tilde{p}(x)dx = \int_{-\infty}^{a} \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}\phi\left(\frac{x-x_i}{h}\right)dx = \frac{1}{n}\sum_{i=1}^{n}\int_{-\infty}^{\frac{a-x_i}{h}}\phi(v)dv = \frac{1}{n}\sum_{i=1}^{n}F_{\phi}(\frac{a-x_i}{h})$$

[20%]

(d) The derivation follows straight from the examples paper.

$$\begin{aligned}\mathcal{E}\{\tilde{p}(x)\} &= \mathcal{E}\left\{\frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}\phi\left(\frac{x-x_i}{h}\right)\right\} \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}\mathcal{E}\left\{\phi\left(\frac{x-x_i}{h}\right)\right\} \\ &= \frac{1}{h}\mathcal{E}\left\{\phi\left(\frac{x-x_i}{h}\right)\right\}\end{aligned}$$

As $\phi()$ is Gaussian distributed then

$$\begin{aligned}\mathcal{E}\left\{\mathcal{N}\left(\frac{x-x_i}{h};0,1\right)\right\} &= \int \mathcal{N}\left(\left(\frac{x-v}{h}\right);0,1\right)\mathcal{N}(v;\mu,\sigma^2)dv \\ &= \int h\mathcal{N}(x;v,h_n^2)\mathcal{N}(v;\mu,\sigma^2)dv \\ &= h\mathcal{N}(x;\mu,\sigma^2+h^2)\end{aligned}$$

Hence $\mathcal{E}\{\tilde{p}(x)\} = \mathcal{N}(x;\mu,\sigma^2+h_n^2)$ [25%]

(e)

$$\begin{aligned}\mathcal{E}\{\tilde{p}(x)\} &= \mathcal{E}\left\{\frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}\phi\left(\frac{x-x_i}{h}\right)\right\} \\ &= \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}\mathcal{E}\left\{\phi\left(\frac{x-X_i}{h}\right)\right\} = \frac{1}{h}\mathcal{E}\left\{\phi\left(\frac{x-X}{h}\right)\right\} \\ \mathcal{E}\left\{\phi\left(\frac{x-X}{h}\right)\right\} &= \int \phi\left(\frac{x-v}{h}\right)p(v)dv\end{aligned}$$

From the definition of $\phi()$ it follows that

$$\phi\left(\frac{x-v}{h}\right) = \begin{cases} \frac{1}{2} & x-h \leq v \leq x+h \\ 0 & \text{otherwise} \end{cases}$$

so that

$$\mathcal{E}\left\{\phi\left(\frac{x-x_i}{h}\right)\right\} = \int_{x-h}^{x+h}\frac{1}{2}p(v)dv = \frac{1}{2}(\int_{-\infty}^{x+h}p(v)dv - \int_{-\infty}^{x-h}p(v)dv) = \frac{1}{2}(F(x+h)-F(x-h))$$

It follows that [25%]

$$\mathcal{E}\{\tilde{p}(x)\} = \frac{1}{2h}(F(x+h)-F(x-h))$$