Graphical Methods
Assignment 4

Team Members:
- Marika Paraskevopoulou
- Eoghan Flanagan
- Jonathan Smith
- Klaudia Ludwisiak

Work Breakdown:
We approached the questions in groups of two or three, and then came together as a team to brainstorm ideas and compare answers where needed.
The following table shows in green which people took primary responsibility for which questions. It should be emphasised that for the harder questions the entire team contributed ideas.

	Eoghan	Klaudia	Maria	Jonathan
12.2				
12.4				
12.6				
23.11				
27.6				
27.9				
27.10				
27.11				

Exercise 12.2

1) Log likelihood function L is:

$$\log\left(\prod_{t=2}^{T} N\left(y_t \mid \vec{w}^T \vec{x}_{t-1}, \sigma_t^2\right)\right)$$

$$= \sum_{t=2}^{T} \frac{-1}{2\sigma_t^2}\left(y_t - \vec{w}^T \vec{x}_{t-1}\right)^2 + \text{constant w.r.t. } \vec{w}$$

Because σ_t^2 is a function of t, this is a weighted least-squares problem.

Define: $\vec{y} = \begin{bmatrix} y_2 \\ y_3 \\ \vdots \\ y_T \end{bmatrix}$, $X = \begin{bmatrix} — \vec{x}_1^T — \\ \vdots \\ — \vec{x}_{T-1}^T — \end{bmatrix}$, $W = \begin{bmatrix} 1/\sigma_2^2 & & & \\ & 1/\sigma_3^2 & & \\ & & \ddots & \\ & & & 1/\sigma_T^2 \end{bmatrix}$,

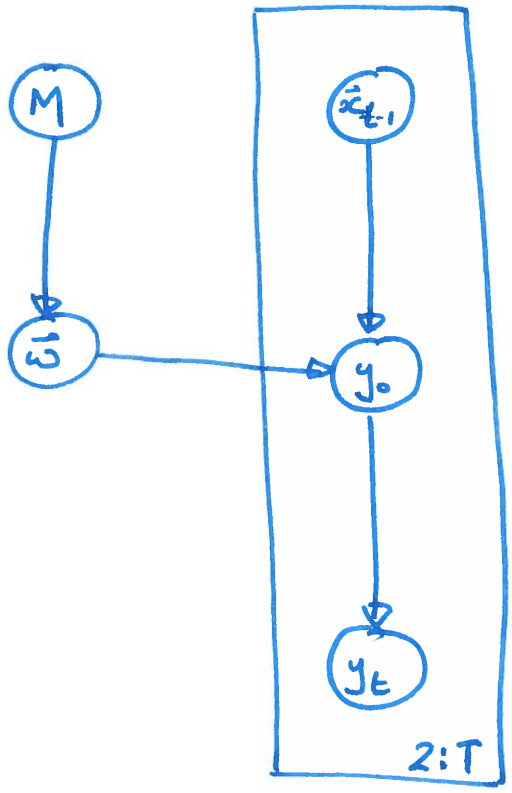
In this way the problem is of the standard form, so the standard weighted least-squares solution can be used:

$$\vec{w}^* = (X^T W X)^{-1} X^T W \vec{y}$$

This value of $\vec{w}^*$ is the maximum likelihood solution, which can be seen from the expression for (log) likelihood above.

Ex 12.2 continued

2) Taking K to be fixed, the hierarchical Bayesian Network for this model is as follows



We are interested in $P(M|D)$, where D is the data.

$$P(M|D) = \frac{P(D|M)P(M)}{P(D)}$$

$$= \int_{\omega} P(D|\vec{w}, \cancel{M}) P(\vec{w}|M)\, d\vec{w} \cdot \frac{P(M)}{P(D)}$$

(M is redundant)

$P(M)$ is constant. $P(D)$ constant with respect to $\vec{w}$

Ex 12.2 continued

So when choosing between models we only need to compare the Bayesian Likelihood term,

$$\int_{\omega} p(D|\vec{\omega})\, p(\vec{\omega}|M)$$

To evaluate this integral we can adapt equation (12.4.7) as follows:

1) Small modification to indexing to account for using $\vec{x}_{t-1}$ to predict y_t.

2) σ_t^2 not constant but is a function of t.

3) To penalise complex models we must replace the 'K' value in (12.4.7) with the number of non-zero factors used in model M. Call this #fac(M).

4) The feature function $\phi(\cdot)$ should be modified to return only the components corresponding to non-zero coefficients of the model. Eg if $\vec{x} = (1,2,3,4,5,6)$ and $M = (0,0,1,1,0,0)$ then $\phi(\vec{x}) = (3,4)$. This is needed to ensure the 'αI' term in the expression for A is consistent with the complexity penalisation.

Ex 12.2 continued

These modifications to equation (12.4.6) give:

$$2 \log P(y_2, \ldots, y_T \mid \vec{x}_1, \ldots, \vec{x}_{T-1}, M) =$$

$$-\sum_{t=2}^{T} \log(2\pi\sigma_t^2) - \sum_{t=2}^{T} \frac{y_t^2}{\sigma^2} + \vec{b}^T A^{-1} \vec{b} - \log \det(A) + \#fac(M).\log l$$

where $A \equiv I_{\#fac(M)} + \sum_{t=2}^{T} \frac{\phi(x^{t-1})\,\phi^T(x^{t-1})}{\sigma_t^2}$

$$\vec{b} \equiv \sum_{t=2}^{T} \frac{y^t \phi(x^{t-1})}{\sigma_t^2}$$

For comparing models we can ignore the constant terms, ie the ones that don't depend on M.

Giving a likelihood score as:

$$\vec{b}^T A^{-1} \vec{b} - \log \det(A) + \#fac(M).\log l$$

Ex 12.2 continued

Method: For each candidate model M, calculate the Bayesian likelihood of that model.

Select the model with the highest likelihood.

Results for part 3

(See following matlab script).

Model selected has:

$\omega_1 = 1$, $\omega_2 = 1$, $\omega_3 = 1$, $\omega_4 = 1$, $\omega_5 = 0$, $\omega_6 = 0$

Exercise 12.2

Table of Contents

Environment setup

```
close all
clear all
import brml.*
load dodder.mat
```

Constants

```
K = 6;
```

Construct each of the candidate models

```
num_models = (2^K - 1);
models = (1:num_models)';
models = de2bi(models,K,'left-msb'); % convert to binary vectors
models = models'; % transpose, to be aligned with data
```

Loop through the candidate models, calculating Bayesian Likelihoods

```
model_likelihoods = zeros(num_models,1); % init

for i = 1:num_models

    % Calculate number of factors used in this model
    num_factors = sum(models(:,i));

    % Calculate A matrix and b vector.  Doing it in a for loop is not
    % the fastest way, but is the clearest for others to understand.

    A = 0; % init
    b = 0; % init

    for t = 2:T

        % Calculate the feature vector, i.e. the x values are to be
```

```
        % considered for this model.
        feature_vec = feat(x(:,t-1),models(:,i));

        % Update A
        A = A + (feature_vec*feature_vec')/(sigma(t)^2);

        % Update b
        b = b + (y(:,t) * feature_vec)/(sigma(t)^2);

    end

    % Finally, add the identity matrix to A
    A = A + eye(num_factors);

    % Calculate Bayesian Likelihood as per (modified) equation
 (12.4.7)
    model_likelihoods(i) = b'*(A\b) - log(det(A)) + num_factors *
 log(1);

end

% Find model with max Bayesian Likelihood
[~,index] = max(model_likelihoods);

disp('Max Bayesian Likelihood model found is:')
disp(models(:,index)')
disp('w1 = 1, w2 = 1, w3 = 1, w4 = 1, w5 = 0, w6 = 0')
```

```
Max Bayesian Likelihood model found is:
     1     1     1     1     0     0

w1 = 1, w2 = 1, w3 = 1, w4 = 1, w5 = 0, w6 = 0
```

Published with MATLAB® R2016b

Filename: feat.m

Function to return the feature vector of an input vector x. In this case the feature vector is simply a subset of the original vector. The particular subset is determined by the (candidate) model. Different models have different complexity and so can make use of larger or smaller feature vectors.

INPUT: x - input column vector. Can accommodate multiple datapoints if arranged as column vectors. model - candidate model. Binary column vector.

OUTPUT: out - feature vector or matrix of output column vectors

```
function out = feat(x, model)

% check that model and original vector are same size
assert(size(x,1) == size(model,1));

% construct feature vector
out = x(model == 1,:);
```

Published with MATLAB® R2016b

Exercise 12.4

Extending the definition of the posterior Bayes Factor to three classifiers (people).

Let the three people be a, b, c.

Hypotheses are $H_{different}$ and H_{same}

Let the data be $\vec{o}_a$, $\vec{o}_b$, $\vec{o}_c$ for people a, b, c respectively.

Interested in $$\frac{P(H_{different} \mid \vec{o}_a, \vec{o}_b, \vec{o}_c)}{P(H_{same} \mid \vec{o}_a, \vec{o}_b, \vec{o}_c)}$$

Rearranging:

$$= \frac{\dfrac{P(H_{different}, \vec{o}_a, \vec{o}_b, \vec{o}_c)}{\cancel{P(\vec{o}_a, \vec{o}_b, \vec{o}_c)}}}{\dfrac{P(H_{same}, \vec{o}_a, \vec{o}_b, \vec{o}_c)}{\cancel{P(\vec{o}_a, \vec{o}_b, \vec{o}_c)}}}$$

Exercise 12.4 continued

Extending the reasoning in the notes in Section 12.6

$$= \frac{P(H_{different}) \, Z(\vec{u}+\#^a) \, Z(\vec{u}+\#^b) \, Z(\vec{u}+\#^c)}{Z(\vec{u}) \; Z(\vec{u}) \; Z(\vec{u})}$$

$$\frac{}{}$$

$$\frac{P(H_{same}) \, Z(\vec{u}+\#^a+\#^b+\#^c)}{Z(\vec{u})}$$

No prior preference among hypotheses →

$$P(H_{different}) = P(H_{same}).$$

giving

$$\frac{Z(\vec{u}+\#^a) \, Z(\vec{u}+\#^b) \, Z(\vec{u}+\#^c)}{Z(\vec{u}) \; Z(\vec{u}) \; Z(\vec{u}+\#^a+\#^b+\#^c)}$$

where: Z is the multivariate beta function

$\vec{u}$ is a (uniform) prior $(1, 1, 1)$

$\#^a, \#^b, \#^c$ are vectors of the number of times person a, b, c chose categories 1, 2, 3

Exercise 12.4 continued

From the data given in the question

$$\#^a = \begin{bmatrix} 13 \\ 3 \\ 4 \end{bmatrix} \quad \#^b = \begin{bmatrix} 4 \\ 9 \\ 7 \end{bmatrix} \quad \#^c = \begin{bmatrix} 8 \\ 6 \\ 4 \end{bmatrix}$$

Using this data (see following MatLab script)

gives $$\frac{P(H_{different} \mid \vec{O}_a, \vec{O}_b, \vec{O}_c)}{P(H_{same} \mid \vec{O}_a, \vec{O}_b, \vec{O}_c)} = 2.76$$

Exercise 12.4

Table of Contents

Environment setup

```
close all
clear all
import brml.*
```

Data

```
a = [13,3,4];
b = [4,9,7];
c = [8,8,4];

u = [1,1,1];  % Parameters of Dirichlet prior
```

Calculation of Posterior Bayes Factor

```
numerator = multi_beta(u+a) * multi_beta(u+b) * multi_beta(u+c);
denominator = multi_beta(u) * multi_beta(u) * multi_beta(u+a+b+c);

posterior_bayes_factor = numerator / denominator;
```

Display Results

```
disp(['Posterior Bayes Factor is: ',
 num2str(posterior_bayes_factor)]);
```

```
Posterior Bayes Factor is: 2.7586
```

Published with MATLAB® R2016b

Filename: multi_beta.m

Function to calcualate the multivariate beta function given a vector of parameters.

INPUT: x - input column vector or row vector of parameters.

OUTPUT: out - result

```
function out = multi_beta(x)

% Basically applying formula
% beta(z,w) = exp(gammaln(z)+gammaln(w)-gammaln(z+w))
% to more than two elements.

gamma_x = gammaln(x);
gamma_sum_x = gammaln(sum(x));

out = exp(sum(gamma_x) - gamma_sum_x);
```

Published with MATLAB® R2016b

Exercise 12.6

Part 1

For clarity, there are three parameters here representing the three degrees of freedom of $p(x,y)$, $x, y \in \{0,1\}$

These parameters are $\theta_{y=1}$, $\theta_{x=1|y=0}$, $\theta_{x=1|y=1}$

Call these θ_1, $\theta_{1|0}$, $\theta_{1|1}$

The expression for $P(\mathcal{D}|M_{y\to x})$ can be clarified as follows

$$P(\mathcal{D}|M_{y\to x}) = \int_{\theta_1}\int_{\theta_{1|0}}\int_{\theta_{1|1}} P(\theta_{1|1},\theta_{1|0})\,P(\theta_1)\prod_n P(x^n|y^n,\theta_{1|0},\theta_{1|1})\,P(y^n|\theta_y)\,d\theta_1\,d\theta_{1|0}\,d\theta_{1|1}$$

Using the relationship given in question $P(\theta_{1|1},\theta_{1|0}) = P(\theta_{1|1})P(\theta_{1|0})$ we can rearrange $P(\mathcal{D}|M_{y\to x})$ as follows:

$$P(\mathcal{D}|M_{y\to x}) = \underbrace{\int_{\theta_{1|0}} P(\theta_{1|0})\prod_{n|y=0} P(x^n|y^n=0,\theta_{1|0})\,d\theta_{1|0}}_{\text{Call this A}} \times \underbrace{\int_{\theta_{1|1}} P(\theta_{1|1})\prod_{n|y=1} P(x^n|y=1,\theta_{1|1})\,d\theta_{1|1}}_{\text{Call this B}}$$

$$\times \underbrace{\int_{\theta_1} P(\theta_1)\prod_n P(y^n|\theta_y)\,d\theta_1}_{\text{Call this C}}$$

Ex 12.6 - continued

Simplifying A by substituting in expression for prior $p(\theta_{1|0})$

$$A = \int_{\theta_{1|0}} \frac{\theta_{1|0}^{(\alpha_{1|0}-1)}(1-\theta_{1|0})^{(\beta_{1|0}-1)}\,\theta_{1|0}^{\#(x=1,y=0)}(1-\theta_{1|0})^{\#(x=0,y=0)}}{B(\alpha_{1|0},\beta_{1|0})}\,d\theta_{1|0}$$

$$= \frac{1}{B(\alpha_{1|0},\beta_{1|0})}\int_{\theta_{1|0}} \theta_{1|0}^{(\#(x=1,y=0)+\alpha_{1|0}-1)}(1-\theta_{1|0})^{(\#(x=0,y=0)+\beta_{1|0}-1)}\,d\theta_{1|0}$$

$$= \frac{B\left(\#(x=1,y=0)+\alpha_{1|0},\ \#(x=0,y=0)+\beta_{1|0}\right)}{B(\alpha_{1|0},\beta_{1|0})}$$

Similarly:

$$B = \frac{B\left(\#(x=1,y=1)+\alpha_{1|1},\ \#(x=0,y=1)+\beta_{1,1}\right)}{B(\alpha_{1|1},\beta_{1|1})}$$

$$C = \int_{\theta_1} \frac{\theta_y^{\alpha-1}(1-\theta_y)^{\beta-1}\,\theta_y^{\#(y=1)}(1-\theta_y)^{\#(y=0)}}{B(\alpha,\beta)}\,d\theta_1$$

$$= \frac{B\left(\#(y=1)+\alpha,\ \#(y=0+\beta)\right)}{B(\alpha,\beta)}$$

Ex 12-6 - Continued

Combining these expressions for A, B, C gives:

$$P(D \mid M_{y \to x}) = \frac{B(\#(x=1, y=0) + \alpha_{1|0}, \#(x=0, y=0) + \beta_{1|0})}{B(\alpha_{1|0}, \beta_{1|0})}$$

$$\times \frac{B(\#(x=1, y=1) + \alpha_{1|1}, \#(x=0, y=1) + \beta_{1|1})}{B(\alpha_{1|1}, \beta_{1|1})}$$

$$\times \frac{B(\#(y=1) + \alpha, \#(y=0) + \beta)}{B(\alpha, \beta)}$$

as required.

Exercise 12.6

Part 2

Assuming the same structure and assumptions of the reverse model as the initial model, we can simply swap the x and y labels to produce the reverse model.

So, notationally: $\theta_{y=1} \longrightarrow \theta_{x=1}$

$\theta_{x=1|y=0} \longrightarrow \theta_{y=1|x=0}$

$\theta_{x=1|y=1} \longrightarrow \theta_{y=1|x=1}$

And for hyperparameters α and β:

$\alpha_y, \beta_y \longrightarrow \alpha_x, \beta_x \quad (\text{So } P(\theta_x) = B(\theta_x | \alpha_x, \beta_x))$

$\alpha_{x=1|y=0} \longrightarrow \alpha_{y=1|x=0}$

$\alpha_{x=1|y=1} \longrightarrow \alpha_{y=1|x=1}$

$\beta_{x=1|y=0} \longrightarrow \beta_{y=1|x=0}$

$\beta_{x=1|y=1} \longrightarrow \beta_{y=1|x=1}$

With these replacements, eq 12.9.19 becomes:

$$P(\mathcal{D}|M_{x\to y}) = \frac{B(\#(y=1,x=0)+\alpha_{y=1|x=0}, \#(y=0,x=0)+\beta_{y=1|x=0})}{B(\alpha_{y=1|x=0}, \beta_{y=1|x=0})}$$

$$\times \frac{B(\#(y=1,x=1)+\alpha_{y=1|x=1}, \#(y=0,x=1)+\beta_{y=1|x=1})}{B(\alpha_{y=1|x=1}, \beta_{y=1|x=1})}$$

$$\times \frac{B(\#(x=1)+\alpha_x, \#(x=0)+\beta_x)}{B(\alpha_x, \beta_x)}$$

Scanned by CamScanner

Exercise 12.6

Part 3 ~~continued~~

So, using these expressions,

$$\frac{P(D \mid M_{y \to x})}{P(D \mid M_{x \to y})} =$$

$$\frac{B(\#(x=1,y=0)+\alpha_{x=1|y=0}, \#(x=0,y=0)+\beta_{x=1|y=0})}{B(\#(y=1,x=0)+\alpha_{y=1|x=0}, \#(y=0,x=0)+\beta_{y=1|x=0})} \times \frac{B(\alpha_{y=1|x=0}, \beta_{y=1|x=0})}{B(\alpha_{x=1|y=0}, \beta_{x=1|y=0})}$$

$$\times \frac{B(\#(x=1,y=1)+\alpha_{x=1|y=1}, \#(x=0,y=1)+\beta_{x=1|y=1})}{B(\#(y=1,x=1)+\alpha_{y=1|x=1}, \#(y=0,x=1)+\beta_{y=1|x=1})} \times \frac{B(\alpha_{y=1|x=1}, \beta_{y=1|x=1})}{B(\alpha_{x=1|y=1}, \beta_{x=1|y=1})}$$

$$\times \frac{B(\#(y=1)+\alpha_y, \#(y=0)+\beta_y)}{B(\#(x=1)+\alpha_x, \#(x=0)+\beta_x)} \times \frac{B(\alpha_x, \beta_x)}{B(\alpha_y, \beta_y)}$$

Scanned by CamScanner

Part 4

In the absence of any prior preference for $M_{y \to x}$ or $M_{x \to y}$, choose the same hyperparameter values for the two models.

In the absence of any other information, let us assume uniform prior distributions, so all α and β parameters are 1.

Using the result from part 3 on the data given:

		x = 0	x = 1
y	0	10	0
	1	10	0

gives $$\frac{P(D|M_{y \to x})}{P(D|M_{x \to y})} = \frac{B(1,11)}{B(11,11)} \frac{B(1,11)}{B(1,1)} \frac{B(11,11)}{B(1,21)}$$

$$= 0.1736$$

ie $M_{x \to y}$ is more likely.

Scanned by CamScanner

Part 4, continued

This result is sufficient to demonstrate the result that the Bayes factor gives a higher probability to the more 'deterministic' model.

Looking at the table of data:

		x = 0	x = 1
y	0	10	0
	1	10	0

If you were describing or constructing this data, x is much more informative than y.

In effect, x "determines" the data

ie $x = 0 \longrightarrow \#(y *, x = 0) = 10$ for $y = 0, 1$

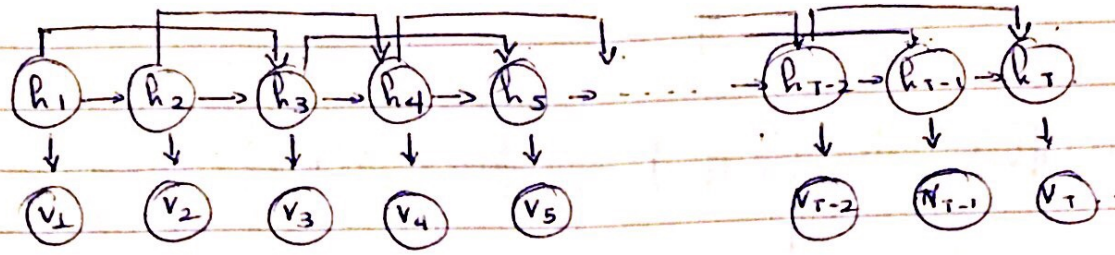
$x = 1 \longrightarrow \#(y, x = 1) = 0$ for $y = 0, 1$

as such $P(\mathcal{D} | M_{x \to y}) > P(\mathcal{D} | M_{y \to x})$

Scanned by CamScanner

23.11

Figure 1: A second order HMM.



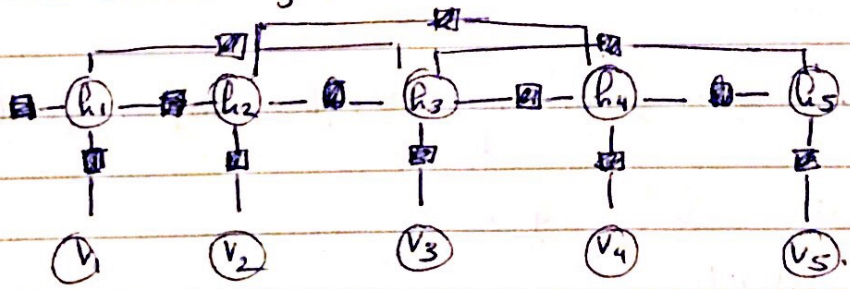
Mathematically a definition of a second order HMM is given by:

$$p(h_{1:T}, v_{1:T}) = p(h_1)\,p(v_1|h_1)\,p(h_2|h_1)\,p(v_2|h_2)\prod_{t=3}^{T} p(h_t|h_{t-1}, h_{t-2})\,p(v_t|h_t)$$

- Consider now the most likely path $h_{1:T}$ of $p(h_{1:T}|v_{1:T})$, i.e. a first order HMM. By definition is the same as the most likely state for fixed $v_{1:T}$ of $p(h_{1:T}, v_{1:T}) = \prod_t p(v_t|h_t)\,p(h_t|h_{t-1})$

Following similar procedure as in the first order HMM we can find the most likely path using the max product version of the factor graph or max-absorption on the junction tree.

Figure 2: Factor graph for a second order HMM for 5 states.



Alternatively an explicit derivation can be obtained by considering

$$\max_{h_T}\left[P(h_1)P(h_2|h_1)P(v_1|h_1)P(v_2|h_2)\prod_{t=3}^{T}P(h_t|h_{t-1},h_{t-2})P(v_t|h_t)\right] =$$

$$P(h_1)P(h_2|h_1)P(v_1|h_1)P(v_2|h_2)\max_{h_T}\left[\prod_{t=3}^{T}P(h_t|h_{t-1},h_{t-2})P(v_t|h_t)\right] =$$

$$P(h_1)P(h_2|h_1)P(v_1|h_1)P(v_2|h_2)\left[\prod_{t=3}^{T-1}P(h_t|h_{t-1},h_{t-2})P(v_t|h_t)\right]\max_{h_T}\left\{P(v_T|h_T)P(h_T|h_{T-1},h_{T-2})\right\}$$

$$\mu(h_{T-1},h_{T-2})$$

In the final step of the above expression we define the $\mu(h_{T-1},h_{T-2})$, which represents a message that carries information from the end of the chain to the penultimate timestep. Defined as: $\mu(h_{T-1},h_{T-2}) = \max_{h_T} P(h_T|h_{T-1},h_{T-2})P(v_T|h_T)$

where we also can see that the message is passing through two hidden states.

We can now continue defining the recursion (following similar steps as in the first order HMM):

$$\mu(h_{t-1},h_{t-2}) = \max_{h_t} P(v_t|h_t)P(h_t|h_{t-1},h_{t-2})\mu(h_t,h_{t-1}),\ 3\le t\le T$$

whereas for the first state we get: $\mu(h_1) = \max_{h_2} P(v_2|h_2)P(h_2|h_1)\mu(h_2,h_1)$ and $\mu(h_T,h_{T-1}) = 1$. (for $t=T$).

so that the most likely state h_1^* is given by:

$$h_1^* = \arg\max_{h_1} P(h_1)P(v_1|h_1)\mu(h_1).$$

Similarly we can found the message $\mu(h_2,h_1^*)$ and the most likely state h_2^* by: $h_2^* = \arg\max_{h_2} P(h_2|h_1^*)P(v_2|h_2)\mu(h_2,h_1^*)$.

Finally once computed, Backtracking gives: $h_t^* = \arg\max_{h_t} P(h_t|h_{t-1}^*,h_{t-2}^*)P(v_t|h_t)\mu(h_t,h_{t-1}^*)$

```
% GM Question 27.6 Gibbs Sampling adapted from demoSampleHMM
close all;
clear all;

%DEMOSAMPLEHMM demo of Gibbs sampling from a HMM versus exact result
fprintf(1,'Draw samples from p(h(1:T)|v(1:T)) for a HMM.\nUse these to
 form empirical estimates of p(h(t)|v(1:T)) and compare to the exact
 p(h(t)|v(1:T))\n\n')
import brml.*

Draw samples from p(h(1:T)|v(1:T)) for a HMM.
Use these to form empirical estimates of p(h(t)|v(1:T)) and compare to
 the exact p(h(t)|v(1:T))
```

Setup:

```
H=2; V=2; T=10;
lambda=[0.1, 1, 10, 20];
len=length(lambda);  % count4 times
c20 = 20; % count 20 times

myMean=zeros(len,c20); % initialise final result matrix

 for l=1:len
 for c= 1:c20

    A=condp(rand(H,H).^lambda(1,l));
    B=condp(rand(V,H));
    a=condp(rand(H,1));

    % draw some samples for v:
    h(1)=randgen(a);
    v(1)=randgen(B(:,h(1)));
    for t=2:T
        h(t)=randgen(A(:,h(t-1)));
        v(t)=randgen(B(:,h(t)));
    end

    % make a HMM
    [logalpha,loglik] = HMMforward(v,A,a,B); % HMM Forward Pass
    logbeta = HMMbackward(v,A,B); % HMM Backward Pass
    gamma = HMMsmooth(logalpha,logbeta,B,A,v); % exact marginal

    % single site Gibbs updating
    hsamp(:,1)=randgen(1:H,1,T);
    hv=1:T; vv=T+1:2*T; % hidden and visible variable indices

    num_samples=100; % count 100 times
    for sample=2:num_samples
        h = hsamp(:,sample-1);
```

```
        emiss=array([vv(1) hv(1)],B);
        trantm=array(hv(1),a);
        trant=array([hv(2) hv(1)],A);
        h(1) = randgen(table(setpot(multpots([trantm trant emiss]),
[vv(1) hv(2)],[v(1) h(2)])));

        for t=2:T-1
            trantm.table=A;
            trantm.variables=[hv(t) hv(t-1)];
            trant.table=A;
            trant.variables=[hv(t+1) hv(t)];
            emiss.table=B;
            emiss.variables=[vv(t) hv(t)];
            h(t) = randgen(table(setpot(multpots([trantm trant
 emiss]),[vv(t) hv(t-1) hv(t+1)],[v(t) h(t-1) h(t+1)])));
        end

        trantm.table=A;
        trantm.variables=[hv(T) hv(T-1)];
        emiss.table=B;
        emiss.variables=[vv(T) hv(T)];
        h(T) = randgen(table(setpot(multpots([trantm emiss]),[vv(T)
 hv(T-1)],[v(T) h(T-1)])));

        hsamp(:,sample)=h; % take the sample after a forward sweep
 through time
    end


    for t=1:T
        gamma_samp(:,t) = count(hsamp(t,:),H)/num_samples;
    end

    % calculate and store mean value on each iterration, rows are
 diff. lambda
    % values and columns are 1-20 repetitions.

    myMean(l,c) = mean(abs(gamma(:)-gamma_samp(:)));
    myMean20 = mean(myMean,2);
 end
 end

 % as a cell: T= [{'lambda'; lambda'}, {'myMean20'; myMean20}];
 % table doenst work with brml.  toolkit

for i=1:len
fprintf('For lambda %f the mean error over 20 x 100 runs is : %f
 \n',lambda(1,i),myMean20(i,1))
end
```

```
For lambda 0.100000 the mean error over 20 x 100 runs is : 0.036236
For lambda 1.000000 the mean error over 20 x 100 runs is : 0.035976
For lambda 10.000000 the mean error over 20 x 100 runs is : 0.211769
For lambda 20.000000 the mean error over 20 x 100 runs is : 0.072520
```

$From the table above it can be observed that the value of the Mean absolute Error for Gibbs sampling increases markedly with the order of magnitude of the parameter lambda. This effect persists when tested for different random samples. The higher the lambda parameter the more deterministic the transition matrix becomes, with the effect of fixing the modelled states. In such a way that i.e. once the h value is fiixed then it is unlikely to ever transition to another value, effectively reducing randomness and increasing mean error of this.

Published with MATLAB® R2016b

27.9

Aces + Bruisers → soccer teams.

2 : genius
1 : superb
0 : very average
-1 : muppet
-2 : worse than your grandma

rating : $a_j = \{-2,-1,0,1,2\}$ for $j = 1,..,20$. and $b_j = \{-2,-1,0,1,2\}$ ability ; player in Aces and in Bruisers

The outcome independently for each game is modeled by

$$P\left[(\text{Aces beats Bruisers}) \mid t^a, t^b, a, b\right] = \sigma\left(\sum_{i=1}^{10} \left(a_{t_i^a} - b_{t_i^b}\right)\right) \quad (1)$$

where $\sigma(x) = \dfrac{1}{1+\exp(-x)}$

10 player were selected to play in the ACES team given by $t_i^a = \{1,..,20\}$ and 10 in Bruisers given by $t_i^b \in \{1,..,20\}$.

1. Assumptions :
   - a, t^a, t^b, b are independent from each other
   - $\pi(t^a) \propto$ const. and $\pi(t^b) \propto$ const

Hence $P\left(a, b \mid \text{Aces beats Bruisers}, t^a, t^b\right) \propto$

$$\propto P\left(\text{Aces beats Bruisers} \mid a, b, t^a, t^b\right) * P\left(a, b, t^a, t^b\right) =$$

from independency assumption

$$= P\left(\text{Aces beats Bruisers} \mid a, b, t^a, t^b\right) \pi(a)\pi(b)\pi(t^a)\pi(t^b) =$$

$\pi(t^a) \propto$ const

$$= P\left(\text{Aces beats Bruisers} \mid a, b, t^a, t^b\right) \pi(a)\pi(b)$$

$\pi(t^b) \propto$ const

Thus we have concluded on :

$$P(\alpha, b \mid \text{Aces beats Bruisers}, t^{\alpha}, t^{b}) \propto P(\text{Aces beats Bruisers} \mid \alpha, b, t^{\alpha}, t^{b}) \pi(\alpha) \pi(b)$$

(MODEL 1)

Hence we can see that we have concluded in an update * for (α, b), although the derivation of $P(\alpha, b)$ is not possible. One way to handle with this is to decide to repetitively update the ability of one player say : j in Aces or Bruisers each time. If we repeatidly do this for all players in Aces and Bruisers then we will conclude in the update rule for all α and b.

Note: We update the ability of player say j conditioning upon all the other player accept this player

PROCEDURE 1.

- Start from the Aces team
- peak a player ; say "j" player, let him be : α_j, define α_{n-1} be the set of the other players accept α_j player.
- Given (MODEL 1) and the discussion above proceed to the ability update for this player

(MODEL 2) : $P(\alpha_j \mid \alpha_{n-1}, b, t^{\alpha}, t^{b}) \propto$

$$\propto P\left[(\text{Aces beats Bruisers}) \mid \alpha, b, t^{\alpha}, t^{b}\right] \pi(\alpha_j)$$

- repeat for all players in Aces and move to Bruisers team

* (MODEL 3) : Say we now peak a player from Bruisers the model will be exactly the same as (MODEL 2)

$$P(b_j \mid b_{n-1}, b, t^{\alpha}, t^{b}) \propto P\left[(\text{Aces beats Bruisers}) \mid \alpha, b, t^{\alpha}, t^{b}\right] \pi(b_j)$$

repeat this for all team players.

- Repeat PROCEDURE 1 until convergence say n times (n : must be big).

∴ PART 1: for the 10 best players for Aces and
10 best " " Bruisers

see MATLAB output

2. Given that we know the Bruisers will field numbered 1 to 10, the best team of players that Aces can field to maximise their chance of winning is to choose the 10 best players. Given that the 10 best players we have concluded for team Aces (specific) from PART 1 are somehow represent a "maximum" choice of player from Bruisers must not affect the way Aces will choose their best team (note also the independence assumptions we've made (we conclude in PART 1) because these 10 best player are represent the "global" best team for each soccer team, and this is way because the best set of players we conclude for each team. they maximise the model for each game as it described in (1). (for each team).

Therefore Aces team must choose the best 10 players hence the players that maximise (1).

Proof: $\arg\max_{\alpha} \left\{ \sigma\left(\sum_{i=1}^{10} \left(\alpha_{t_i}^{\alpha} - b_{t_i}^{b} \right) \right) \right\} =$

$$= \arg\max_{\alpha} \left\{ \frac{1}{1 + \exp\left(\sum \left(\alpha_{t_i}^{\alpha} - b_{t_i}^{b} \right) \right)} \right\} = \left(\text{hence this is} \right.$$

equivalent with maximizing $\sum \left(\alpha_{t_i}^{\alpha} - b_{t_i}^{b} \right)$ wrt α) .. =

$$= \max_{\alpha} \sum \alpha_{t_i}^{\alpha}$$

```
% Question 27.9

clear all;
close all;

import brml.*;
load('soccer.mat');

% setup
player = 20; % no. of players
g = size(game,2); % no. of games
rating = [-2,-1,0,1,2]; % rating of players' levels
r = length(rating);
probA = ones(r,player)/r; % Skill levels initially all uniform
probB = ones(r,player)/r;
loop = 100; % initialise loop, needed for probabilities to converge

  for i=1:loop
    % for each player go through all the games, find those he took
 part in.
    % for those games calculate ability update and update
 probability .
    % normalise probabilities.

    for p = 1:player
      for gg = 1:g
      % for each game access teamAces squad an check if layer p was in
 it
        if (ismember(p, game(gg).teamAces))
          probA(:,p) = probA(:,p) .*(AbilityUpdate('A',game(gg), p,
 probA, probB,rating))';
        end
       % Similarly, for each game access teamBruisers squad an check
 if layer p was in it
        if (ismember(p, game(gg).teamBruisers))
          probB(:,p) = probB(:,p) .* (AbilityUpdate('B',game(gg), p,
 probA, probB,rating))';
        end
      end
      % normalise probabilities:
      probA(:,p) = probA(:,p) ./ sum(probA(:,p));
      probB(:,p) = probB(:,p) ./ sum(probB(:,p));
    end
  end

% calculate expected ability for each player (probabilty * rating),
% sort and store values:

[rateA, playerA] = sort(probA' * rating'); % sort in ascending order
[rateB, playerB] = sort(probB' * rating');

best10A = playerA(11:end); % select last 10 values and display
```

```
best10B = playerB(11:end);

rate10A = round(rateA(11:end),4); % select corresponding bets ratings
rate10B = round(rateB(11:end),4);

save('makeTableDat','best10A','rate10A','best10B','rate10B');
% want to make a table but 'table' is a different command in the brml
% toolkit. Instead, I will save the variables of interest and make
 table in
% new script file without importing the toolkit.
```

Published with MATLAB® R2016b

```
% Load data and make table
load('makeTableDat.mat')

table([best10A],...
      [rate10A],...
      [best10B],...
      [rate10B],...
      'VariableNames',
 {'Best_Aces','Rating_A', 'Best_Bruisers','Rating_B'},...
      'RowNames',
 {'P1','P2','P3','P4', 'P5','P6','P7','P8','P9','P10'})


ans =

            Best_Aces    Rating_A    Best_Bruisers    Rating_B
            _________    ________    _____________    ________

    P1      15                0      17                    0
    P2      12           0.0001      20                    0
    P3      14           0.0059       9                    0
    P4      20                1      19                    0
    P5       1           1.0205       6                    0
    P6       7           1.0539       5               0.0051
    P7       2           1.1924      12                    1
    P8      13           1.9997      16                    1
    P9      11                2      13               1.0008
    P10     10                2       7                1.001
```

Published with MATLAB® R2016b

```
function Abil = AbilityUpdate(teamtype, game, player, probA, probB,
 rating)
% calculates likelihood function of players ability given by:
% p(Aces beats Bruisers|t_a,t_b,a,b)= sigma(sum(a(t_a)-b(t_b))
% Inputs:
% teamtype : A or B
% game: matrix from load('soccer.mat') of selected index 1 to 2o
% player: index of player in team
% probA: prior probability from team A (dimensions should be rating x
 games)
% probB: prior probability from team B (dimensions should be rating x
 games)
% rating: vector of players rating classes

% Setup and initialise variables:
% Invert to align structure with variable 'game' (from soccer.mat):
probA = probA';
probB = probB';

% make sure rating has right dimensions:
R = size(rating,2);
if R>1
rating = rating';
else R = size(rating,1);
end

% set initial matrix of ability:
Abil = zeros(1,R);

if teamtype == 'A' || teamtype == 'a' % player is in team A
  for r = 1:R
    % initialise temporary variables:
    tempsum = 0;
    tempPA = probA;
    tempPA(player,:) = zeros(1,R);
    tempPA(player,r) = 1;
    for i=1:10 % loop over all players evaluating sum(a(t_a)-b(t_b):
      tempsum = tempsum + (tempPA(game.teamAces(i),:)...
                  - probB(game.teamBruisers(i),:)) * rating;
    end
    % evaluate sigma function (as per assignment question):
        if(game.AcesWin == -1) % formula changes depending if Aces
 lose/win
          Abil(r) = 1 - 1/(1+exp(-tempsum));
        else
          Abil(r) = 1/(1+exp(-tempsum));
        end
  end

else % player is in team B
  for r = 1:R
    % initialise temporary variables:
```

```
    tempsum = 0;
    tempPB = probB;
    tempPB(player,:) = zeros(1,R);
    tempPB(player,r) = 1;
    for i=1:10 % loop over all players evaluating sum(a(t_a)-b(t_b):
      tempsum = tempsum + (probA(game.teamAces(i),:) -
 tempPB(game.teamBruisers(i),:)) * rating;
    end
    % evaluate sigma function (as per assignment question):
        if(game.AcesWin == -1) % formula changes depending if Aces
 lose/win
          Abil(r) = 1 - 1/(1+exp(-tempsum));
        else
          Abil(r) = 1/(1+exp(-tempsum));
        end
  end
end

end
```

Published with MATLAB® R2016b

Exercise 27.10

Gibbs Sampling for Disease/Symptom problem

```
% initialise random number generator and load data

rng(555);
load SymptomDisease;

% given that there are 2^50 possible symptom states we choose a high
 number
% of burn-in iterations

burn_in_iterations=50000;
total_iterations=60000;

sub_sampling_ratio=10;
counter=1;

% sample from the distribution of the provided prior probabilities
d_status=rand(50,1)<p;

for iteration=1:total_iterations

    if mod(iteration,10000)==0
        disp(['Running iteration',num2str(iteration)]);
    end



    % create a random ordering of the diseases
    % this is perhaps not strictly necessary, and we could choose
    % a random disease vector on a 'one by one' basis to update
    % but there's an intuitive appeal to making sure we update all the
    % parameters on each iteration

    update_order=randperm(50);


    for ii=1:50
        % choose the disease to update
        d_index=update_order(ii);

        % create two copies of the d_status variable
        % set the t th disease status to 0 and 1
        candidate1_d_status=d_status;
        candidate2_d_status=d_status;
        candidate1_d_status(d_index)=0;
        candidate2_d_status(d_index)=1;

        % compute the joint probabilities of the symptom
        % states given the two candidate disease vectors
```

```
        candidate1_s_probs=ones(200,1)./(ones(200,1)+exp(-
(W*candidate1_d_status+b)));
        candidate2_s_probs=ones(200,1)./(ones(200,1)+exp(-
(W*candidate2_d_status+b)));

        % compute the log likelihoods of the given symptom vector
        % s under the two candidate symptom likelihood vectors

        candidate1_s_logprobs=log(candidate1_s_probs);
        candidate1_s_loginvprobs=log(1-candidate1_s_probs);
        candidate2_s_logprobs=log(candidate2_s_probs);
        candidate2_s_loginvprobs=log(1-candidate2_s_probs);
        candidate1_s_jointloglike=sum(s.*candidate1_s_logprobs+(1-
s).*candidate1_s_loginvprobs);
        candidate2_s_jointloglike=sum(s.*candidate2_s_logprobs+(1-
s).*candidate2_s_loginvprobs);

        % calculate the rpobability of cendiate 1 from the log
 likelihoods
        prob_val=1/(1+exp(candidate2_s_jointloglike-
candidate1_s_jointloglike));

        % sample from this univariate binary distribution
        if rand(1)<prob_val
            d_status(d_index)=0;
        else
            d_status(d_index)=1;
        end
    end


    if
 (iteration>burn_in_iterations)&&(mod(iteration,sub_sampling_ratio)==0)
        output_sample(counter,:)=d_status;
        counter=counter+1;
    end


end

% output the probabilty (average over the runs) of the diseases in the
% disease vector being present.

output_p_vector=mean(output_sample)

Running iteration10000
Running iteration20000
Running iteration30000
Running iteration40000
Running iteration50000
Running iteration60000

output_p_vector =
```

```
Columns 1 through 7

  0.0240    0.9980    0.2600    1.0000    0.8510    0.1670    0.1870

Columns 8 through 14

  0.0310    0.0310    1.0000    0.0080    1.0000    1.0000    0.9990

Columns 15 through 21

  0.9710    0.6750    0.4280    0.4740    0.9970    0.9750    0.0310

Columns 22 through 28

  0.2690    1.0000    0.9200    0.1130    1.0000    0.0170         0

Columns 29 through 35

  0.9270         0    0.0040    0.2460    0.2250    1.0000    0.0070

Columns 36 through 42

  0.0050    0.9750    0.0060         0    0.0020    0.9980    0.9950

Columns 43 through 49

  1.0000    0.9990         0    0.0030    0.9990    1.0000         0

Column 50

  0.9090
```

Published with MATLAB® R2016b

Exercise 27.11

By the law of total probability, applied to the joint state of $W, \vec{b}, \vec{p}$

$$p(\vec{d} \mid \vec{s}, \mathcal{D}) = \int_{W,\vec{b},\vec{p}} p(\vec{d} \mid \vec{s}, W, \vec{b}, \vec{p}, \mathcal{D}) p(W, \vec{b}, \vec{p})$$

$$p(W, \vec{b}, \vec{p}) = p(W, \vec{b}, \vec{p} \mid \mathcal{D}) p(\mathcal{D})$$

by definition of conditional probability

therefore this equals

$$\int_{W,\vec{b},\vec{p}} p(\vec{d} \mid \vec{s}, W, \vec{b}, \vec{p}, \mathcal{D}) p(\mathcal{D}) \, p(W, \vec{b}, \vec{p} \mid \mathcal{D})$$

$$= \int_{W,\vec{b},\vec{p}} p(\vec{d} \mid \vec{s}, W, \vec{b}, \vec{p}) \, p(W, \vec{b}, \vec{p} \mid \mathcal{D})$$

as desired

$$p(W, \vec{b}, \vec{p} \mid \mathcal{D}) p(\mathcal{D}) = p(\mathcal{D} \mid W, \vec{b}, \vec{p}) p(W, \vec{b}, \vec{p})$$

(Bayes)

$$\Rightarrow p(W, \vec{b}, \vec{p} \mid \mathcal{D}) \propto p(W, \vec{b}, \vec{p}) p(\mathcal{D} \mid W, \vec{b}, \vec{p})$$

since $\mathcal{D}$ is fixed, hence $p(\mathcal{D})$ is

Scanned by CamScanner

Assuming the patient records are iid

this implies

$$p(W, \vec{b}, \vec{p}, \mathcal{D}) \propto p(W, \vec{b}, \vec{p}) \prod_{n=1}^{N} p(\vec{s}^n | \vec{d}^n, W, \vec{b}) \times p(\vec{d}^n | W, \vec{b}, \vec{p})$$

But $\vec{d}^n$ does not depend on W or $\vec{b}$

therefore this implies

$$p(W, \vec{b}, \vec{p} | \mathcal{D}) \propto p(W, \vec{b}, \vec{p}) \prod_{n=1}^{N} p(s^n | d^n, W, \vec{b}) \times p(d^n | \vec{p})$$

as stated

Scanned by CamScanner

$$p(d_i=1 \mid \vec{s}, \mathcal{D}) = \int_{w,\vec{b},\vec{p}} p(d_i=1 \mid w,\vec{b},\vec{p})\, p(w,\vec{b},\vec{p} \mid \mathcal{D})$$

By Bayes, $p(d_i=1 \mid w,\vec{b},\vec{p}) =$

$$\frac{p(d_i=1, w, \vec{b}, \vec{p}_{\setminus d_i}, \vec{s})}{p(d_i=0, w, \vec{b}, \vec{p}_{\setminus d_i}, \vec{s}) + p(d_i=1, w, \vec{b}, \vec{p}_{\setminus d_i}, \vec{s})}$$

For a given $w, \vec{b}, \vec{p}$ this can be calculated using the $p(s_j=1 \mid \vec{d}) = \sigma(w_j^T d + b_j)$

Therefore, writing $\theta = w, \vec{b}, \vec{p}$ we have

$$p(d_i=1 \mid \vec{s}, \mathcal{D}) = \int_{\theta} f(\theta)\, g(\theta)$$

where f can be calculated and g is known up to a constant

$\Rightarrow$ we can use importance sampling to calculate $p(d_i=1, \vec{s}, \mathcal{D})$. Choose a proposal distribution for $w, \vec{b}, \vec{p}$

Scanned by CamScanner

and generate draws from this

Then calculate $p(d_i = 1 | W, \vec{b}, \vec{p})$ and the corresponding weight w_j which is proportional to $p(W, \vec{b}, \vec{p}) \prod_{n=1}^{N} p(s^n | d^n, W, b) p(d^n | p)$

Then calculate $\sum_j w_j p(d_i = 1 | W, \vec{b}, \vec{p})$

$$\frac{\sum_j w_j p(d_i = 1 | W, \vec{b}, \vec{p})}{\sum_j w_j}$$

Scanned by CamScanner