Graphical Models - Assignment 3

James Gin, John Goodacre, Christopher Loy, Mark Neumann

December 2015

Problem 9.1

1

Entries loaded manually, see code below.

2

Using 1 to indicate a fault and 0 to indicate no fault:

$p(fuse = 1|burning = 1, quality = 0, wrinkled = 0, multipages = 0, jam = 1) = 1$

3

$p(fuse = 1|burning = 1, quality = 0, wrinkled = 0, multipages = 0, jam = 1) = 0.68113$

4

Most likely: $\{fuse = 1, drum = 0, toner = 0, paper = 0, roller = 0\}$

5

Most likely: $\{fuse = 1, drum = 0, toner = 0, paper = 0, roller = 0\}$

Code for all above exercises:

```
function ex9_1

import brml.*
```

```
[fuse, drum, toner, paper, roller, burning, quality, wrinkled, mpages,
    pjam]=assign(1:10);
nofault=1;fault=2;

variable=struct([]);
varnames={'fuse' 'drum' 'toner' 'paper' 'roller' 'burning' 'quality'
    'wrinkled' 'mpages' 'pjam'};
for v=1:10;
   variable(v).name=varnames{v}; variable(v).domain={'nofault','fault'};
end

% 1) & 2)
% For a BN associate p(i|pa(i)) with potential i: core skill nodes
pot(fuse).variables=fuse; pot(fuse).table(fault)=3/15;
    pot(fuse).table(nofault)=1-pot(fuse).table(fault);
pot(drum).variables=drum; pot(drum).table(fault)=4/15;
    pot(drum).table(nofault)=1-pot(drum).table(fault);
pot(toner).variables=toner; pot(toner).table(fault)=5/15;
    pot(toner).table(nofault)=1-pot(toner).table(fault);
pot(paper).variables=paper; pot(paper).table(fault)=8/15;
    pot(paper).table(nofault)=1-pot(paper).table(fault);
pot(roller).variables=roller; pot(roller).table(fault)=3/15;
    pot(roller).table(nofault)=1-pot(roller).table(fault);

%modules dependent on a single
pot(burning).variables=[burning fuse];
tmptable1(fault, nofault)=0; tmptable1(fault, fault)=2/3;
tmptable1(nofault, nofault)=1; tmptable1(nofault, fault)=1/3;
pot(burning).table=tmptable1;

% modules dependent on multiple

pot(wrinkled).variables=[wrinkled fuse paper];
tmptable2(fault, nofault, nofault)=1/5; tmptable2(fault, fault,
    nofault)=1/2;
tmptable2(fault, nofault, fault)=2/7; tmptable2(fault, fault, fault)=1;
tmptable2(nofault, nofault, nofault)=4/5; tmptable2(nofault, fault,
    nofault)=1/2;
tmptable2(nofault, nofault, fault)=5/7; tmptable2(nofault, fault,
    fault)=0;
pot(wrinkled).table=tmptable2;

pot(mpages).variables=[mpages paper roller];
tmptable2(fault, nofault, nofault)=0; tmptable2(fault, fault,
    nofault)=2/7;
tmptable2(fault, nofault, fault)=1/2; tmptable2(fault, fault, fault)=1;
tmptable2(nofault, nofault, nofault)=1; tmptable2(nofault, fault,
    nofault)=5/7;
tmptable2(nofault, nofault, fault)=1/2; tmptable2(nofault, fault,
    fault)=0;
```

```
pot(mpages).table=tmptable2;

pot(pjam).variables=[pjam fuse roller];
tmptable2(fault, nofault, nofault)=4/10; tmptable2(fault, fault,
    nofault)=1/2;
tmptable2(fault, nofault, fault)=1; tmptable2(fault, fault, fault)=1;
tmptable2(nofault, nofault, nofault)=6/10; tmptable2(nofault, fault,
    nofault)=1/2;
tmptable2(nofault, nofault, fault)=0; tmptable2(nofault, fault, fault)=0;
pot(pjam).table=tmptable2;

pot(quality).variables=[quality drum toner paper];
tmptable3(fault, nofault, nofault, nofault)=0; tmptable3(fault, fault,
    nofault, nofault)=1;
tmptable3(fault, nofault, fault, nofault)=1; tmptable3(fault, nofault,
    nofault, fault)=1/5;
tmptable3(fault, fault, fault, nofault)=1; tmptable3(fault, fault,
    nofault, fault)=0;
tmptable3(fault, nofault, fault, fault)=1; tmptable3(fault, fault,
    fault, fault)=1;
tmptable3(nofault, nofault, nofault, nofault)=1; tmptable3(nofault,
    fault, nofault, nofault)=0;
tmptable3(nofault, nofault, fault, nofault)=0; tmptable3(nofault,
    nofault, nofault, fault)=4/5;
tmptable3(nofault, fault, fault, nofault)=0; tmptable3(nofault, fault,
    nofault, fault)=1;
tmptable3(nofault, nofault, fault, fault)=0; tmptable3(nofault, fault,
    fault, fault)=0;
pot(quality).table=tmptable3;

pot=setpotclass(pot,'array'); % convert to cell array
drawNet(dag(pot),variable);

jointpot = multpots(pot(1:10));

symptoms = [burning quality wrinkled mpages pjam];
faults = [fault nofault nofault nofault fault];

margpot = sumpot(setpot(jointpot,symptoms,faults),setdiff(1:10,fuse));
disp(['p(fuse=fault) '
    num2str(margpot.table(fault)./sum(margpot.table))]);


% 3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% For a BN associate p(i|pa(i)) with potential i: core skill nodes
% repeat the above using a flat bayesian prior
potb(fuse).variables=fuse; potb(fuse).table(fault)=(1+3)/(2+15);
    potb(fuse).table(nofault)=1-potb(fuse).table(fault);
potb(drum).variables=drum; potb(drum).table(fault)=(1+4)/(2+15);
```

```
    potb(drum).table(nofault)=1-potb(drum).table(fault);
potb(toner).variables=toner; potb(toner).table(fault)=(1+5)/(2+15);
    potb(toner).table(nofault)=1-potb(toner).table(fault);
potb(paper).variables=paper; potb(paper).table(fault)=(1+8)/(2+15);
    potb(paper).table(nofault)=1-potb(paper).table(fault);
potb(roller).variables=roller; potb(roller).table(fault)=(1+3)/(2+15);
    potb(roller).table(nofault)=1-potb(roller).table(fault);

%modules dependent on a single
potb(burning).variables=[burning fuse];
tmptable1(fault, nofault)=(1)/(2+12); tmptable1(fault,
    fault)=(1+2)/(2+3);
tmptable1(nofault, nofault)=(1+12)/(2+12); tmptable1(nofault,
    fault)=(1+1)/(2+3);
potb(burning).table=tmptable1;

% modules dependent on multiple
potb(wrinkled).variables=[wrinkled fuse paper];
tmptable2(fault, nofault, nofault)=(1+1)/(2+5); tmptable2(fault, fault,
    nofault)=(1+1)/(2+2);
tmptable2(fault, nofault, fault)=(1+2)/(2+7); tmptable2(fault, fault,
    fault)=(1+1)/(2+1);
tmptable2(nofault, nofault, nofault)=(1+4)/(2+5); tmptable2(nofault,
    fault, nofault)=(1+1)/(1+2);
tmptable2(nofault, nofault, fault)=(1+5)/(2+7); tmptable2(nofault,
    fault, fault)=(1+0)/(2+1);
potb(wrinkled).table=tmptable2;

potb(mpages).variables=[mpages paper roller];
tmptable2(fault, nofault, nofault)=(1+0)/(2+5); tmptable2(fault, fault,
    nofault)=(1+2)/(2+7);
tmptable2(fault, nofault, fault)=(1+1)/(2+2); tmptable2(fault, fault,
    fault)=(1+1)/(2+1);
tmptable2(nofault, nofault, nofault)=(1+5)/(2+5); tmptable2(nofault,
    fault, nofault)=(1+5)/(2+7);
tmptable2(nofault, nofault, fault)=(1+1)/(2+2); tmptable2(nofault,
    fault, fault)=(1+0)/(2+1);
potb(mpages).table=tmptable2;

potb(pjam).variables=[pjam fuse roller];
tmptable2(fault, nofault, nofault)=(1+4)/(2+10); tmptable2(fault, fault,
    nofault)=1/2;
tmptable2(fault, nofault, fault)=(1+2)/(2+2); tmptable2(fault, fault,
    fault)=(1+1)/(2+1);
tmptable2(nofault, nofault, nofault)=(1+6)/(2+10); tmptable2(nofault,
    fault, nofault)=1/2;
tmptable2(nofault, nofault, fault)=(1+0)/(2+2); tmptable2(nofault,
    fault, fault)=(1+0)/(2+1);
potb(pjam).table=tmptable2;
potb(quality).variables=[quality drum toner paper];
tmptable3(fault, nofault, nofault, nofault)=(1+0)/(2+3);
    tmptable3(fault, fault, nofault, nofault)=(1+1)/(2+1);
tmptable3(fault, nofault, fault, nofault)=(1+2)/(2+2); tmptable3(fault,
    nofault, nofault, fault)=(1+1)/(2+5);
tmptable3(fault, fault, fault, nofault)=(1+1)/(2+1); tmptable3(fault,
    fault, nofault, fault)=(1+0)/(2+1);
tmptable3(fault, nofault, fault, fault)=(1+1)/(2+1); tmptable3(fault,
    fault, fault, fault)=(1+1)/(2+1);
tmptable3(nofault, nofault, nofault, nofault)=(1+3)/(2+3);
    tmptable3(nofault, fault, nofault, nofault)=(1+0)/(2+1);
tmptable3(nofault, nofault, fault, nofault)=(1+0)/(2+2);
    tmptable3(nofault, nofault, nofault, fault)=(1+4)/(2+5);
tmptable3(nofault, fault, fault, nofault)=(1+0)/(2+1);
    tmptable3(nofault, fault, nofault, fault)=(1+1)/(2+1);
tmptable3(nofault, nofault, fault, fault)=(1+0)/(2+1);
    tmptable3(nofault, fault, fault, fault)=(1+0)/(2+1);
potb(quality).table=tmptable3;

potb=setpotclass(potb,'array'); % convert to cell array
drawNet(dag(potb),variable);

jointpotb = multpots(potb(1:10));

symptoms = [burning quality wrinkled mpages pjam];
faults = [fault nofault nofault nofault fault];

%disptable(condpot(jointpotb,fuse),variable);
margpotb = sumpot(setpot(jointpotb,symptoms,faults),setdiff(1:10,fuse));

disp(['p(fuse=fault) '
    num2str(margpotb.table(fault)./sum(margpotb.table))]);

% 4)
%do max absorbtion on associated junction tree
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[~, ~, infostruct] = jtree(jointpotb);
[jtpot, jtsep] =
    jtassignpot(setpot(jointpotb,symptoms,faults),infostruct);
[jtpot, ~, ~] = absorption(jtpot, jtsep, infostruct, 'max');
jtmax=zeros(10,1);
for i=1:length(jtpot)
  [~, jtmax(jtpot{i}.variables)] = maxpot(jtpot{i});
end

fprintf('Most likely: (')
for var = [fuse drum toner paper roller]
  fprintf('%s=%s, ', variable(var).name,
      variable(var).domain{jtmax(var)})
end
```

```
fprintf(')\n')

% 5)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
symptoms = [burning pjam];
faults = [fault fault];

[~, ~, infostruct] = jtree(jointpotb);
[jtpot, jtsep] =
    jtassignpot(setpot(jointpotb,symptoms,faults),infostruct);
[jtpot, ~, ~] = absorption(jtpot, jtsep, infostruct, 'max');
jtmax=zeros(10,1);
for i=1:length(jtpot)
  [~, jtmax(jtpot{i}.variables)] = maxpot(jtpot{i});
end

fprintf('Most likely: (')
for var = [fuse drum toner paper roller]
  fprintf('%s=%s, ', variable(var).name,
      variable(var).domain{jtmax(var)})
end
fprintf(')\n')

end
```

Problem 9.9

Show that the model likelihood equation $p(D) = \prod_k \prod_j \frac{Z(u'(v_k;j))}{Z(u(v_k;j))}$ **can be written explicitly as** $p(D|M) = \prod_k \prod_j \frac{\Gamma(\sum_i (u_i(v_k;j)))}{\Gamma(\sum_i (u'_i(v_k;j)))} \prod_i \frac{\Gamma(u'_i(v_k;j))}{\Gamma(u_i(v_k;j))}$**.**

Given that the normalizing constant of the Dirichlet distribution is the Multinomial beta function defined over the hyper-parameters of the same Dirichlet distribution, we see that:

$$Z(\alpha) = \frac{\prod_{i=1}^{M} \Gamma(\alpha_i)}{\Gamma(\sum_{i=1}^{M} \alpha_i)} \tag{1}$$

where $\alpha = (\alpha_1 .... \alpha_M)$ are the real valued "pseudo-counts" for probability of observing random variables according to a particular multinomial distribution.

By direct substitution, we see that:

$$p(D) = \prod_k \prod_j \frac{Z(u'(v_k;j))}{Z(u(v_k;j))} = \prod_k \prod_j \frac{\prod_i \Gamma(u'_i(v_k;j))}{\Gamma(\sum_i (u'_i(v_k;j)))} \frac{\Gamma(\sum_i (u_i(v_k;j)))}{\prod_i \Gamma(u_i(v_k;j))} \tag{2}$$

where the various products correspond to the following: k, the number of variables, j, the number of possible parental states of each node and i, the number of hyper-parameters of the Dirichlet distribution (which will be determined by how the belief network factorises, which is the point of this entire process as through the factorisation, we have to obtain fewer hyper-parameters describing the marginal distributions between variables because we are utilising the graph structure).

Re-arranging and condensing the products over i into a single product, we achieve the final explicit model likelihood:

$$p(D|M) = \prod_k \prod_j \frac{Z(u'(v_k;j))}{Z(u(v_k;j))} = \prod_k \prod_j \frac{\Gamma(\sum_i (u_i(v_k;j)))}{\Gamma(\sum_i (u'_i(v_k;j)))} \prod_i \frac{\Gamma(u'_i(v_k;j))}{\Gamma(u_i(v_k;j))} \tag{3}$$

Problem 9.10

How many Belief Networks are in N_a?

Given an "youngest-first" ancestral ordering on 8 variables ($v_1 ... v_8$), we can see that the number of possible configurations of a belief network where each variable has at most two parents is given by:

$$\prod_{i=1}^{6} \left(1 + \begin{pmatrix} 8-i \\ 1 \end{pmatrix} + \begin{pmatrix} 8-i \\ 2 \end{pmatrix} \right) \times 2 \tag{4}$$

as if we consider the ith ancestral node, it can either have 0,1 or 2 parents. There is only one combination of this node with zero parents(as we are considering one node at a time). For the cases where it has one or two parents, there are 8 - i choose 1,2 possible permutations of the variables which are ahead of the node in the ancestral ordering. Note that for the penultimate node can either have one or zero parents, as there is only one option left to choose from in the ancestral order, so there are only two possible combinations here. We do not consider variables with a lower ancestral index than i, as these cannot be parents of i. Through this calculation, we calculate the number of 8 node trees with a maximum of two parents to be 6,288,128.

What is the computational time to find the optimal member of N_a using the Bayesian Dirichlet score, assuming that computing the BD score of any member of N_a takes 1 second and bearing in mind the decomposability of the BD score.

As we know the correct ancestral ordering, for any given variable, we know the set of its possible parents. As each node can only have a maximum of two parents, for a given node we arrive at the following expression for the number of possible configurations for a given vertex, C_v from the above equation deriving the number of BNs with this structure:

$$C_v = 1 + \binom{P_v}{1} + \binom{P_v}{2} \tag{5}$$

where P_v is the size of the parental set for vertex v. We add one to represent the configuration where the node v has no parents. Due to the fact that the likelihood function decomposes into additive log terms for the network score, we can search for the most likely state of each variable's family(including all of it's possible parental combination) independently and this will coincide with the global highest scoring belief network.

Therefore, the number of evaluations we need to make is simply the sum of the number of possible combinations of parents for each node. Given that it takes one second to score, it will take 92 seconds(91 to score all the individual vertices and an additional one to evaluate the BN where the root node is disconnected) to find the optimal member of N_a.

Estimate the time required to find the optimal member of N.

Given that there are 8! ways to order 8 distinct variables, a loose upper bound on the time required to find the best scoring BN would be $8! \times 92 = 3,709,440$ seconds. However, each ordering of the variables will contain some duplication, due to the fact that if a variable is disconnected from the BN structure, it's position in a given ancestral ordering is arbitrary. As such the actual time required is likely to be less than this upper bound.

Problem 9.13

```
function ex9_13
  close all
  import brml.*
  load('ChowLiuData')
  drawNet(ChowLiu(X));
end

function A = ChowLiu(X)
% Calculates an (undirected) ChowLiu tree for the DxN data matrix X
  import brml.*
  [D, ~] = size(X);
  K = length(unique(X));
  wij = zeros(D,D);
  for i = 1:D
    for j = 1:D
      wij(i, j) = MI(X, K, i, j);
    end
  end
  A = spantree(wij);
end

function mi = MI(X, K, i, j)
% Calculates the mutual information between variables i and j
  pi = zeros(K, 1);
  pj = zeros(K, 1);
  pij = zeros(K, K);
  for k1 = 1:K
    pi(k1) = sum(X(i, :) == k1);
    pj(k1) = sum(X(j, :) == k1);
    for k2 = 1:K
      pij(k1, k2) = double(X(i, :) == k1) * double(X(j, :) == k2)';
    end
  end
  pi = pi / sum(pi);
  pj = pj / sum(pj);
  pij = pij / sum(sum(pij));
  mi = 0;
  for k1 = 1:K
    for k2 = 1:K
      if pij(k1, k2) > 0 && pi(k1) > 0 && pj(k2) > 0
        mi = mi + pij(k1, k2) * log(pij(k1, k2) / (pi(k1) * pj(k2)));
      end
    end
  end
end
```

Code for finding the Chow-Liu tree is given above. For `ChowLiuData`, this results in the DAG shown in Figure 1.

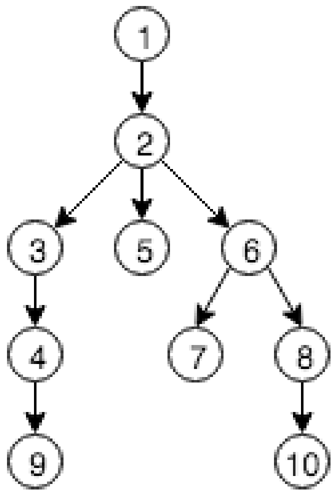

Figure 1: DAG for Problem 9.13.

Problem 10.2

1

The question asks for us to use naive Bayes to calculate the probability that a person who is 'not rich', 'married' and 'healthy' is 'content'?

We implemented this small example using the BRML toolbox and closely following DemoNaiveBayes. The implementation is given in the listing below.

Results Using this implementation we found that the probability a person who is 'not rich', 'married' and 'healthy' is content to be .... 0.70093.

2 Calculate the probability that a person who is 'not rich', 'married' is 'content'?

Results

Here we don't know if the person is healthy or not. But we can express it in terms of things we do know. Here we use $m = married$, $h = healthy$, $r = rich$

and $c = content$:

$$\begin{aligned} p(c,h|m,r) &= \frac{p(c,h,m,r)}{p(m,r)} \\ &= \frac{p(c,h,m,r)p(h)}{p(m,r)p(h)} \end{aligned}$$

But from independence assumptions in Naive Bayes, $p(m,r)p(h) = p(h,m,r)$. Therefore:

$$\begin{aligned} p(c,h|m,r) &= \frac{p(c,h,m,r)p(h)}{p(m,r,h)} \\ &= p(c|h,m,r)p(h) \end{aligned}$$

Finally:

$$\begin{aligned} p(c|m,r) &= \sum_h p(c,h|m,r) \\ &= \sum_h p(c|h,m,r)p(h) \end{aligned}$$

Calculating this gives a probability of 0.4023.

3

Naive bayes makes an assumption that the value of a particular feature is independent of the value of any other feature, given the class variable. In part 3 of this example this is not the case as clearly if for example we know that a person is younger than 20 years old, then they are not between 20-30 years old or in the other categories. The problem here is using a binary classification to set age states, where we have dependency. The solution to this is to move away from the binomial and move to multinomials and thus enable variables such as 'age' in this case to have multiple states.

```
function NaiveBayes102
function NaiveBayes102
%Naive Bayes using Bernoulli Distribution
import brml.*
xC=[1 0 1 1; % Content
    1 0 1 0;
    1 1 0 1];

xNC=[0 1 0 0 0; % Not content
    0 0 0 1 0;
    0 0 1 0 0];
```

```
% ML class priors pC = p(c=Content), pS=p(c=Not Content)
pC = size(xC,2)/(size(xC,2) + size(xNC,2)); pNC =1-pC;

mC = mean(xC')'; % ML estimates of p(x=1|c=Content)
mNC = mean(xNC')'; % ML estimates of p(x=1|c=Not content)

%1. what is the probability that a person who is 'not rich, married and
%healthy is content'

xtest=[0 1 1]'; % not rich, married, healthy

npC = pC*prod(mC.^xtest.*(1-mC).^(1-xtest)); % p(x,c=Content)
npNC = pNC*prod(mNC.^xtest.*(1-mNC).^(1-xtest)); % p(x,c=Not content)

pxC = npC/(npC+npNC); % probability that x is content
disp(['probability not rich, married, healthy is content=
    ',num2str(pxC)]);

%2. what is the probability that a person who is a person who is 'not
    rich'
%and 'married' is content? need to sum over both cases, we already have
    the
%case where the person is not rich and married, but healthy...

xtest1 = [0,1,0]'; %not healthy case

%unhealhy case
npC = pC*prod(mC.^xtest1.*(1-mC).^(1-xtest1)); % p(x,c=Content)
npNC = pNC*prod(mNC.^xtest1.*(1-mNC).^(1-xtest1)); % p(x,c=Not content)

pxC1 = npC/(npC+npNC); % probability that x is content, not healthy

pH = sum([xC(3, :) xNC(3, :)]) / (size(xC, 2) + size(xNC, 2)); pNH =
    1-pH;

%therefore the probability that he is content, given not rich and not
%married is ...
pC2 = pH * pxC + pNH * pxC1;

disp(['probability not rich, married ..is content= ',num2str(pC2)]);
```

Problem 11.1

Given the belief network described in Problem 9.1, we are presented with a set of datapoints which contain missing data for some parameters. We assume that this data is missing at random. We use the EM algorithm to learn the conditional probability tables.

We then set the evidential variables to their given states `Wrinkled = No`, `Burning = No`, `Quality = Yes` to return the probability $p(Drum = Yes) = 0.6190$.

```
function ex11p1

import brml.*
load EMprinter
[V N]=size(x);
Fuse = 1; Drum = 2; Toner = 3; Paper = 4; Roller = 5;
Burning = 6; Quality = 7; Wrinkled = 8; MultPages = 9; PaperJam = 10;
no = 1; yes = 2;

for i = 1:5
    pot{i}.variables = i;
    pot{i}.table = zeros(1,2);
end

pot{Burning}.variables = [Fuse, Burning];
pot{Burning}.table = zeros(2,2);

pot{Quality}.variables = [Drum, Toner, Paper, Quality];
pot{Quality}.table = zeros(2,2,2,2);

pot{Wrinkled}.variables = [Fuse, Paper, Wrinkled];
pot{Wrinkled}.table = zeros(2,2,2);

pot{MultPages}.variables = [Paper, Roller, MultPages];
pot{MultPages}.table = zeros(2,2,2);

pot{PaperJam}.variables = [Fuse, Roller, PaperJam];
pot{PaperJam}.table = zeros(2,2,2);

pot = setpotclass(pot, 'array');

empot=pot;
PotentialsToUpdate=1:10;
[vars numstates]=potvariables(pot);
pars.tol=0.001; pars.maxiterations=20;pars.plotprogress=1;
pars.PotentialsToUpdate=PotentialsToUpdate;
empot=EMbeliefnet(empot,x,pars);
jpot = multpots(empot);
drumPot = condpot(setpot(jpot,[Wrinkled, Burning,
    Quality],[no,no,yes]),Drum);
pDrum = drumPot.table(yes)
```

Problem 11.4

(1)

Given a distribution which is a mixture of H first-order Markov chains of length T, where the variables $h = 1, \ldots, H$ are hidden, we can derive the EM algorithm for training as follows:

We consider a single variable pair $(\mathbf{v}, h)$ where v represents the visible Markov chain variables $(v_1, \ldots, v_T)$.

Consider the KL divergence between a distribution $q(h|v)$ and $p(h|v, \theta)$ where θ are the parameters of the model. This is ≥ 0:

$$KL(q(h|v)|p(h|v,\theta)) \equiv \langle \log q(h|v) - \log p(h|v,\theta) \rangle_{q(h|v)} \geq 0$$

As $p(h|v,\theta) = p(h, v|\theta)/p(v, \theta)$ and $p(v|\theta)$ has no h dependence:

$$KL(q(h|v)|p(h|v,\theta)) \equiv \langle \log q(h|v) \rangle_{q(h|v)} - \langle \log p(h,v,\theta) \rangle_{q(h|v)} + \log p(v|\theta) \geq 0$$

Which when rearranged yields:

$$\log p(v|\theta) \geq -\langle \log q(h|v) \rangle_{q(h|v)} + \langle \log p(h,v,\theta) \rangle_{q(h|v)}$$

Therefore we have found a lower bound for the log-likelihood for a single training example.

As in this case the data are selected i.i.d from the mixture model, we can use the fact that the log likelihood of a set of N datapoints $\mathbf{V}$ is equal to the sum of the individual datapoint log likelihoods. Therefore:

$$\log p(\mathbf{V}|\theta) \geq \ L(\{q\}, \theta) \equiv -\sum_{n=1}^{N} \{\log q(h_n|v_n)\}_{q(h_n,v_n)} + \sum_{n=1}^{N} \{\log p(h_n, v_n|\theta)\}_{q(h_n,v_n)}$$

For the EM algorithm, the lower bound is maximised in alternative steps by optimising $q(h_n|v_n)$ (M-step) and then θ (E-step), with the other variable fixed. For the E-step this is equivalent to setting $q(h_n|v_n) = p(h_n, v_n|\theta)$.

In this problem:

$$p(\mathbf{v}) = \sum_{h=1}^{H} p(h) p(v_1|h) \prod_{t=1}^{T-1} p(v_{t+1}|v_t, h)$$

Consider then that the parameters θ for this model are the transition matrices $p(v_1|h)$ and $p(v_{t+1}|v_t, h)$ we then perform the following EM algorithm:

- Initialise randomly the elements of the CPTs $p(v_1|h)$, $p(v_{t+1}|v_t, h)$ and also $p(h)$
- Loop through the n data points v_n
- Perform the E-step: calculate the $p(h_n|v_n, \theta)$ given the parameters (by Bayes' rule, $p(v_n|h_n)p(h)/p(v)$)
- Perform the M-Step: Set $p(v_1|h), p(v_{t+1}|v_t|h)$ and $p(h)$ to the maximum likelihood values given the previously calculated $q(h_n|v_n)$, i.e. by counting (summing expected counts and normalising)
- End when likelihood or parameters have converged by some threshold value

(2)

We utilise the following code to run the EM algorithm on the sequences and find the following 2 clusters:

Cluster1

```
CATAGGCATTCTATGTGCTG
CCAGTTACGGACGCCGAAAG
CGGCCGCGCCTCCGGGAACG
ACATGAACTACATAGTATAA
GTTGGTCAGCACACGGACTG
CACTACGGCTACCTGGGCAA
CGGTCCGTCCGAGGCACTCG
CACCATCACCCTTGCTAAGG
CAAATGCCTCACGCGTCTCA
GCCAAGCAGGGTCTCAACTT
CATGGACTGCTCCACAAAGG
```

Cluster2

```
TGGAACCTTAAAAAAAAAAA
GTCTCCTGCCCTCTCTGAAC
GTGCCTGGACCTGAAAAGCC
AAAGTGCTCTGAAAACTCAC
CCTCCCCTCCCCTTTCCTGC
TAAGTGTCCTCTGCTCCTAA
AAAGAACTCCCCTCCCTGCC
AAAAAAACGAAAAACCTAAG
GCGTAAAAAAAGTCCTGGGT
```

The code runs multiple times in order to reduce the chance of finding only a local optimum, as EM is not guaranteed to find global optima. The best clustering generates a log-likelihood of -483.6352.

```
function ex11p4
import brml.*
load sequences
amino = 'ACGT';
for n=1:20 % convert the characters into (arbitrary) numerical values
    (from 1 to 4)
    for t = 1:20
        v{n}(t) = find(amino==sequences{n}(t));
    end
end
opts.plotprogress=2;
opts.maxit=50;
clusters = cell(100,2);
like = zeros(100,1);
for run = 1:100
[ph,pv1gh,pvgvh,loglikelihood,phgv]=mixMarkov(v,4,2,opts);

c=ones(1,20);
cluster1=[]; cluster2=[];
for n=1:20
   if phgv{n}(1)>0.5; c(n)=2;
      cluster1=[cluster1; sequences{n}];
   else
      cluster2=[cluster2; sequences{n}];
   end
end
clusters{run,1} = cluster1;
clusters{run,2} = cluster2;
like(run) = loglikelihood;

end

[y, ind] = max(like);
disp('Max Likelihood')
disp(y)
disp('Cluster1')
disp(clusters{ind,1})
disp('Cluster2')
disp(clusters{ind,2})
```

References