Graphical Models: Assignment 3

Jesús Herrera, I-Horng Huang, Tomas Jakab, Antonio Remiro, Nicholas Williams

12 December 2016

Contributions

jesus.herrera.16@ucl.ac.uk: 9.13, 10.3, 23.4
i-horng.huang.16@ucl.ac.uk: 23.15, 23.17
tomas.jakab.16@ucl.ac.uk: 7.4, 7.17, 23.8
antonio.remiro.16@ucl.ac.uk: 9.1, 10.5, 23.16
nicholas.williams.16@ucl.ac.uk: 9.9, 9.10, 9.14

Exercise 7.4

We formulate the problem of finding the optimal x_t, y_t sequence as MDP. We use policy iteration algorithm to compute optimal policy. We set the discounting factor γ to 0.95 in order to ensure algorithm convergence. The optimal sequence of states is then obtained by following the optimal policy from the start state $(1, 13)$ to the parking bay of the airplane at $(8, 4)$. This is done by function `getPath`. The solution for both parts of this exercise differs only in the definition of transition matrix. The optimal sequences are shown in the output of our code at the end of this exercise.

1. We define the deterministic transition matrix $p(s_{t+1}|s_t, d_t)$ as follows:

```
for x = 1:Gx
        for y = 1:Gy
                p(st(x,y),st(x,y),stay)=1; % can stay in same state
                if validgridposition(x+1,y,Gx,Gy)
                        p(st(x+1,y),st(x,y),right)=1;
                end
                if validgridposition(x-1,y,Gx,Gy)
                        p(st(x-1,y),st(x,y),left)=1;
                end
                if validgridposition(x,y+1,Gx,Gy)
                        p(st(x,y+1),st(x,y),up)=1;
                end
                if validgridposition(x,y-1,Gx,Gy)
                        p(st(x,y-1),st(x,y),down)=1;
                end
        end
end
```

2. The non-deterministic transition matrix $p(s_{t+1}|s_t, d_t)$ is defined as follows:

```
for x = 1:Gx
  for y = 1:Gy
    p(st(x,y),st(x,y),stay)=1; % can stay in same state
    if validgridposition(x+1,y,Gx,Gy)
      if ~validgridposition(x,y+1,Gx,Gy)
        p(st(x+1,y),st(x,y),right)=1;
      else
        p(st(x+1,y),st(x,y),right)=0.9;
        p(st(x,y+1),st(x,y),right)=0.1;
      end
    end
    if validgridposition(x-1,y,Gx,Gy)
```

```
          p(st(x-1,y),st(x,y),left)=1;
        end
        if validgridposition(x,y+1,Gx,Gy)
          p(st(x,y+1),st(x,y),up)=1;
        end
        if validgridposition(x,y-1,Gx,Gy)
          p(st(x,y-1),st(x,y),down)=1;
        end
    end
end
```

Code:

```
function ex74

close all ;

run BRMLtoolkit/setup.m ;
import brml.*

load airplane.mat

Gx = size(U, 1); Gy = size(U, 2);  % two dimensional grid size
S = Gx*Gy; % number of states on grid
st = reshape(1:S,Gx,Gy); % assign each grid point a state

A = 5;  % number of action (decision) states
[stay up down left right] = assign(1:A); % actions (decisions)
p = zeros(S,S,A); % initialise the transition p(xt|xtm,dtm) ie p(x(t)|x(t-1),d(t
    -1))

disp('--------------------------------------------------------------------------------')
disp('part 1')
disp('--------------------------------------------------------------------------------')
% make a deterministic transition matrix on a 2D grid:
for x = 1:Gx
        for y = 1:Gy
                p(st(x,y),st(x,y),stay)=1; % can stay in same state
                if validgridposition(x+1,y,Gx,Gy)
                        p(st(x+1,y),st(x,y),right)=1;
                end
                if validgridposition(x-1,y,Gx,Gy)
                        p(st(x-1,y),st(x,y),left)=1;
                end
                if validgridposition(x,y+1,Gx,Gy)
                        p(st(x,y+1),st(x,y),up)=1;
                end
                if validgridposition(x,y-1,Gx,Gy)
                        p(st(x,y-1),st(x,y),down)=1;
                end
        end
end
% define utilities
u = U(:) ;
figure(1); imagesc(reshape(u,Gx,Gy)); colorbar; title('utilities');

gam = 0.95; % discount factor
opts.plotprogress=1;
opts.maxiterations=30; opts.tol=0.001; % termination critria

figure(2);
[val,dstar] = myMDPSolve(p,u,gam,opts) ;
```

```
figure(3); imagesc(reshape(val,Gx,Gy));
figure(4); imagesc(reshape(dstar, Gx, Gy)) ;

dstar = reshape(dstar,Gx,Gy) ;
startx = 1 ;
starty = 13 ;

endx = 8 ;
endy = 4 ;

path = getPath(dstar, startx, starty, endx, endy) ;
gridPath = zeros(Gx,Gy) ;
gridPath(sub2ind([Gx, Gy], path(:,1), path(:,2))) = 1 ;
figure(5); imagesc(gridPath)
disp('Optimal sequence x(t), y(t):') ;
disp(path)

disp('-----------------------------------------------------------------------')
disp('part 2')
disp('-----------------------------------------------------------------------')
for x = 1:Gx
  for y = 1:Gy
    p(st(x,y),st(x,y),stay)=1; % can stay in same state
    if validgridposition(x+1,y,Gx,Gy)
      if ~validgridposition(x,y+1,Gx,Gy)
        p(st(x+1,y),st(x,y),right)=1;
      else
        p(st(x+1,y),st(x,y),right)=0.9;
        p(st(x,y+1),st(x,y),right)=0.1;
      end
    end
    if validgridposition(x-1,y,Gx,Gy)
      p(st(x-1,y),st(x,y),left)=1;
    end
    if validgridposition(x,y+1,Gx,Gy)
      p(st(x,y+1),st(x,y),up)=1;
    end
    if validgridposition(x,y-1,Gx,Gy)
      p(st(x,y-1),st(x,y),down)=1;
    end
  end
end

figure(6);
[val,dstar] = myMDPSolve(p,u,gam,opts) ;

figure(7); imagesc(reshape(val,Gx,Gy));
figure(8); imagesc(reshape(dstar, Gx, Gy)) ;

dstar = reshape(dstar,Gx,Gy) ;

path = getPath(dstar, startx, starty, endx, endy) ;
gridPath = zeros(Gx,Gy) ;
gridPath(sub2ind([Gx, Gy], path(:,1), path(:,2))) = 1 ;
figure(9); imagesc(gridPath)
disp('Optimal sequence x(t), y(t):') ;
disp(path)


% -----------------------------------------------------------------------
function [val, dstar]=myMDPSolve(tran,util,gam,opts)
```

```
% -----------------------------------------------------------------------
%MDPSOLVE solve a Markov Decision Process
% val=MDPsolve(tran, util, gam, opts)
% tran : transition probability matrix
% util : utility matrix
% gam : discounting factor
import brml.*
[xt xtm dtm]=assign(1:3); % assign the variables x(t), x(t-1), d(t-1) to some numbers
% define the domains of the variables
S = size(tran,1); A = size(tran,3);

% define the transition p(x(t)|x(t-1),d(t-1))
tranpot=array;
tranpot.variables=[xt xtm dtm]; tranpot.table=tran;
% setup the value v(x(t))
valpot=array;
valpot.variables=xt; valpot.table=ones(S,1); % initial values
oldvalue=valpot.table;

% Policy Iteration:
pdstar=zeros(S,S);
for policyloop=1:opts.maxiterations
    % Policy evaluation: get the optimal decisions as a function of the state:
    [tmppot dstar] = maxpot(sumpot(multpots([tranpot valpot]),xt),dtm);
    for x1=1:S
        for x2=1:S
            pdstar(x1,x2) = tran(x2,x1,dstar(x1));
        end
    end
    valpot.table = (eye(S)-gam*pdstar)\util;
    if mean(abs(valpot.table-oldvalue))<opts.tol; break; end % stop if converged
    oldvalue=valpot.table;
    if opts.plotprogress;       bar(valpot.table); drawnow; end
end
val=valpot.table;

% -----------------------------------------------------------------------
function positions = getPath(dstar, startx, starty, endx, endy)
% -----------------------------------------------------------------------
import brml.*
positions = [startx, starty] ;
x = startx ;
y = starty ;
[stay up down left right] = assign(1:5); % actions (decisions)
while ~(x == endx && y == endy)
        % [stay up down left right]
        action = dstar(x ,y) ;
        if action == up
                y = y + 1 ;
        elseif action == down
                y = y - 1 ;
        elseif action == left
                x = x - 1 ;
        elseif action == right
                x = x + 1 ;
        end
        positions = [positions; x, y] ;
end
```

Output:

```
part 1
--------------------------------------------------------------------
Optimal sequence x(t), y(t):
     1    13
     1    12
     1    11
     2    11
     3    11
     4    11
     5    11
     6    11
     7    11
     8    11
     9    11
    10    11
    11    11
    12    11
    13    11
    14    11
    15    11
    15    10
    15     9
    15     8
    15     7
    14     7
    13     7
    12     7
    11     7
    10     7
     9     7
     8     7
     7     7
     6     7
     5     7
     4     7
     4     6
     4     5
     4     4
     5     4
     6     4
     7     4
     8     4

--------------------------------------------------------------------
part 2
--------------------------------------------------------------------
Optimal sequence x(t), y(t):
     1    13
     1    14
     2    14
     3    14
     4    14
     5    14
     6    14
     7    14
     8    14
     9    14
    10    14
    11    14
    12    14
    13    14
    14    14
```

15	14
16	14
16	13
16	12
16	11
16	10
15	10
15	9
15	8
15	7
14	7
13	7
12	7
11	7
10	7
9	7
8	7
7	7
6	7
5	7
4	7
4	6
4	5
4	4
5	4
6	4
7	4
8	4

Exercise 7.17

We formulate the problem as temporary bounded MDP. The set of decisions is $D = \{down, same, up\}$. Since $T = 50$, the set of possible states X is finite, $X = \{-49, -48, \ldots, 0, \ldots, 48, 49\}$. The utility function is non-zero only at $t = 50$, therefore we only define

$$u_{50}(x) = \begin{cases} 1 & \text{if } x \in \{-25, 25\} \\ 0 & \text{otherwise} \end{cases}.$$

Our task is to compute

$$U(d_1|x_1) = \sum_{x_2} \max_{d_2} \sum_{x_3} \max_{d_3} \sum_{x_4} \cdots \max_{d_{49}} \sum_{x_{50}} p(x_2|x_1, d_1) p(x_3|x_2, d_2) \cdots p(x_{50}|x_{49}, d_{49}) u_{50}(x_{50})$$

for every d_1 and $x_1 = 0$. $U(d_1|x_1)$ can be rewritten as

$$U(d_1|x_1) = \sum_{x_2} p(x_2|x_1, d_1) \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2) \max_{d_3} \sum_{x_4} \cdots \max_{d_{48}} \sum_{x_{49}} p(x_{49}|x_{48}, d_{48}) \max_{d_{49}} \sum_{x_{50}} p(x_{50}|x_{49}, d_{49}) u_{50}(x_{50}).$$

This can be efficiently solved by simple message passing. Let us formulate messages:

$$\begin{aligned}
v_{49\leftarrow 50}(x_{49}) &= \max_{d_{49}} \sum_{x_{50}} p(x_{50}|x_{49}, d_{49}) u_{50}(x_{50}) \\
v_{48\leftarrow 49}(x_{48}) &= \max_{d_{48}} \sum_{x_{49}} p(x_{49}|x_{48}, d_{48}) v_{49\leftarrow 50}(x_{49}) \\
&\vdots \\
v_{2\leftarrow 3}(x_2) &= \max_{d_2} \sum_{x_3} p(x_3|x_2, d_2) v_{3\leftarrow 4}(x_3).
\end{aligned}$$

Finally, we have

$$U(d_1|x_1) = \sum_{x_2} p(x_2|x_1, d_1) v_{2\leftarrow 3}(x_2).$$

The exact values are numerically computed by Matlab code as shown below. The optimal expected reward of taking the actions *down*, *same*, *up* at timestep 1 is 0.7684, 0.7658, and 0.7684 respectively.

Code:

```
function ex717

run BRMLtoolkit/setup.m ;
import brml.*

[down, same, up] = assign(1:3) ;
% we start at 0 with offset 50
nStates = 49 + 49 + 1 ;
% define the transition matrix p(x_t| x_t-1, d_t-1)
ptable = zeros(nStates, nStates, 3) ;
for state=2:nStates-1
  ptable(state-1, state, down) = 0.8 ;
  ptable(state, state, down) = 0.1 ;
  ptable(state+1, state, down) = 0.1 ;

  ptable(state, state, same) = 0.8 ;
  ptable(state-1, state, same) = 0.1 ;
  ptable(state+1, state, same) = 0.1 ;

  ptable(state+1, state, up) = 0.8 ;
  ptable(state, state, up) = 0.1 ;
  ptable(state-1, state, up) = 0.1 ;
end

% utility u(x_50)
utable = zeros(nStates, 1) ;
utable(50+25) = 1 ;
utable(50-25) = 1 ;
upot = array([50], utable) ;

% assign x(t), x(t-1), d(t-1)
[xt, xtm, dtm] = assign([50, 49, 100]) ;
pot = array([xt xtm dtm], ptable) ;
% multiply with utilities
pot = pot * upot ;
% sum over x_50
pot = sumpot(pot, xt) ;
% max over d
pot = maxpot(pot, dtm) ;

% do message passing
for x=(49:-1:3)
  [xt, xtm, dtm] = assign([x, x-1, 100]) ;
  % multiply with previous message u_t<t+1
  pot = array([xt xtm dtm], ptable) * pot ;
  % sum over x_t
  pot = sumpot(pot, xt) ;
  % max over d
  pot = maxpot(pot, dtm) ;
end

% finally compute U(d_1|x_1)
[xt, xtm, dtm] = assign([2, 1, 100]) ;
pot = array([xt xtm dtm], ptable) * pot ;
pot = sumpot(pot, xt) ;

% U(d_1|x_1=0)
disp('Expected rewards for actions down, same, up:');
```

```
disp(pot.table(50,:))
```

Output:

```
Expected rewards for actions down, same, up:
    0.7684    0.7658    0.7684
```

Exercise 9.1

1. The belief network presented by StopPress is equivalent to:

$$p(B|F)p(Q|D,T,P)p(W|P,F)p(M|P,R)p(PJ|F,R)p(F)p(D)p(T)p(P)p(R),$$

where F represents fuse assembly malfunction, D drum unit, T toner out, P poor paper quality, R worn roller, B burning smell, Q poor print quality, W wrinkled pages, M multiple pages fed and PJ paper jam. To learn the table entries we can do so by counting the number of times a given variable is in state 1 for each combination of its parental states:

$$\begin{aligned}
p(B=1|F=1) &= \frac{2}{3},\\
p(B=1|F=0) &= 0,\\
p(Q=1|D=1,T=1,P=1) &= 1,\\
p(Q=1|D=1,T=1,P=0) &= 1,\\
p(Q=1|D=1,T=0,P=1) &= 1,\\
p(Q=1|D=1,T=0,P=0) &= 1.
\end{aligned}$$

The last 4 entries imply $p(Q=1|D=1)=1$. Similarly,

$$\begin{aligned}
p(Q=1|D=0,T=1,P=1) &= 1,\\
p(Q=1|D=0,T=1,P=0) &= 1,\\
p(Q=1|D=0,T=0,P=1) &= \frac{1}{5},\\
p(Q=1|D=0,T=0,P=0) &= 0.\\
p(W=1|P=1,F=1) &= 1,\\
p(W=1|P=1,F=0) &= \frac{2}{7},\\
p(W=1|P=0,F=1) &= \frac{1}{2},\\
p(W=1|P=0,F=0) &= \frac{1}{5},\\
p(M=1|P=1,R=1) &= 1,\\
p(M=1|P=1,R=0) &= \frac{2}{7},\\
p(M=1|P=0,R=1) &= \frac{1}{2},\\
p(M=1|P=0,R=0) &= 0,\\
p(PJ=1|F=1,R=1) &= 0,\\
p(PJ=1|F=1,R=0) &= \frac{1}{2},\\
p(PJ=1|F=0,R=1) &= 1,\\
p(PJ=1|F=0,R=0) &= \frac{2}{5}.
\end{aligned}$$

Similarly, based on counting, for the parent variables: $p(F=1)=\frac{1}{5}$, $p(D=1)=\frac{4}{15}$, $p(T=1)=\frac{1}{3}$, $p(P=1)=\frac{8}{15}$ and $p(R=1)=\frac{1}{5}$. These five CPTs then complete the full distribution specification.

2. We use the computed maximum likelihood tables and the BRML Toolbox to program the belief network. Our code for the implemented function `demoPrinter.m` is presented below:

```
function demoPrinter
% DEMOPRINTER: Programs the belief network for Exercise 9.1 using the calculated
% maximum likelihood tables.
import brml.*
% Variable order is arbitary
[fuse drum toner paper roller burning quality wrinkled multpages jam]=assign(1:10)
    ;

yes=1; no=2; % define states, starting from 1. yes means there is a fault/
    diagnosis

variable(fuse).name='fuse'; variable(fuse).domain = {'yes','no'};
variable(drum).name='drum'; variable(drum).domain = {'yes','no'};
variable(toner).name='toner'; variable(toner).domain = {'yes','no'};
variable(paper).name='paper'; variable(paper).domain = {'yes','no'};
variable(roller).name='roller'; variable(roller).domain = {'yes','no'};
variable(burning).name='burning'; variable(burning).domain = {'yes','no'};
variable(quality).name='quality'; variable(quality).domain = {'yes','no'};
variable(wrinkled).name='wrinkled'; variable(wrinkled).domain = {'yes','no'};
variable(multpages).name='multpages'; variable(multpages).domain = {'yes','no'};
variable(jam).name='jam'; variable(jam).domain = {'yes','no'};

pot(fuse).variables=fuse;
pot(fuse).table(yes)=0.2;
pot(fuse).table(no)=0.8;

pot(drum).variables=drum;
pot(drum).table(yes)=(4/15);
pot(drum).table(no)= 1 - pot(drum).table(yes);

pot(toner).variables=toner;
pot(toner).table(yes)=(1/3);
pot(toner).table(no)= 1 - pot(toner).table(yes);

pot(paper).variables=paper;
pot(paper).table(yes)=(8/15);
pot(paper).table(no)= 1 - pot(paper).table(yes);

pot(roller).variables=roller;
pot(roller).table(yes)=0.2;
pot(roller).table(no)= 0.8;

pot(burning).variables=[burning fuse];
pot(burning).table(yes,yes)=(2/3);
pot(burning).table(no,yes)=(1/3);
pot(burning).table(yes,no)=0;
pot(burning).table(no,no)=1;

pot(quality).variables=[quality drum toner paper];
pot(quality).table(yes,yes,yes,yes)=1;
pot(quality).table(yes,yes,yes,no)=1;
pot(quality).table(yes,yes,no,yes)=1;
pot(quality).table(yes,yes,no,no)=1;
pot(quality).table(yes,no,yes,yes)=1;
pot(quality).table(yes,no,yes,no)=1;
pot(quality).table(yes,no,no,yes)=0.2;
pot(quality).table(yes,no,no,no)=0;
pot(quality).table(no,yes,yes,yes)=0;
pot(quality).table(no,yes,yes,no)=0;
pot(quality).table(no,yes,no,yes)=0;
pot(quality).table(no,yes,no,no)=0;
pot(quality).table(no,no,yes,yes)=0;
```

```
pot(quality).table(no,no,yes,no)=0;
pot(quality).table(no,no,no,yes)=0.8;
pot(quality).table(no,no,no,no)=1;

pot(wrinkled).variables=[wrinkled fuse paper];
pot(wrinkled).table(yes,yes,yes)=1;
pot(wrinkled).table(yes,yes,no)=0.5;
pot(wrinkled).table(yes,no,yes)=(2/7);
pot(wrinkled).table(yes,no,no)=0.2;
pot(wrinkled).table(no,yes,yes)=0;
pot(wrinkled).table(no,yes,no)=0.5;
pot(wrinkled).table(no,no,yes)=1-pot(wrinkled).table(yes,no,yes);
pot(wrinkled).table(no,no,no)=0.8;

pot(multpages).variables=[multpages paper roller];
pot(multpages).table(yes,yes,yes)=1;
pot(multpages).table(yes,yes,no)=(2/7);
pot(multpages).table(yes,no,yes)=0.5;
pot(multpages).table(yes,no,no)=0;
pot(multpages).table(no,yes,yes)=0;
pot(multpages).table(no,yes,no)=1-pot(multpages).table(yes,yes,no);
pot(multpages).table(no,no,yes)=0.5;
pot(multpages).table(no,no,no)=1;

pot(jam).variables=[jam fuse roller];
pot(jam).table(yes,yes,yes)=0;
pot(jam).table(yes,yes,no)=0.5;
pot(jam).table(yes,no,yes)=1;
pot(jam).table(yes,no,no)=0.4;
pot(jam).table(no,yes,yes)=1;
pot(jam).table(no,yes,no)=0.5;
pot(jam).table(no,no,yes)=0;
pot(jam).table(no,no,no)=0.6;

pot=setpotclass(pot,'array');
% do inference:
jointpot = multpots(pot([fuse drum toner paper roller ... % joint distribution
                         burning quality wrinkled multpages jam]));
disptable(jointpot,variable);
drawNet(dag(pot),variable);

disp('p(fuse|burn=yes, jam=yes, wrinkled=no, multpages=no, quality=no):')
disptable(condpot(setpot(jointpot,[burning jam wrinkled multpages quality], ...
                         [yes yes no no no]), fuse),variable);
```

Running the `demoprinter.m` function outputs :

```
p(fuse|burn=yes, jam=yes, wrinkled=no, multpages=no, quality=no):
fuse    =yes    1.000000e+00
fuse    =no     0
```

The probability that there is a fuse assembly malfunction is 1.

3. Consider our belief network. Using a flat Beta prior $\alpha = \beta = 1$ for all conditional probability tables, the marginal probability tables are given by,

$$p(F=1|\mathcal{V}) = \frac{1+\#(F=1)}{2+N} = \frac{1+3}{2+15} = \frac{4}{17},$$

$$p(D=1|\mathcal{V}) = \frac{1+\#(D=1)}{2+N} = \frac{1+4}{2+15} = \frac{5}{17},$$

$$p(T=1|\mathcal{V}) = \frac{1+\#(T=1)}{2+N} = \frac{1+5}{2+15} = \frac{6}{17},$$

$$p(P=1|\mathcal{V}) = \frac{1+\#(P=1)}{2+N} = \frac{1+8}{2+15} = \frac{9}{17},$$

$$p(R=1|\mathcal{V}) = \frac{1+\#(R=1)}{2+N} = \frac{1+3}{2+15} = \frac{4}{17},$$

Therefore, we now have for our conditional probability tables:

$$p(B=1|F=1,\mathcal{V}) = \frac{1+\#(B=1,F=1)}{2+\#(B=1,F=1)+\#(B=0,F=1)} = \frac{1+2}{2+2+1} = \frac{3}{5},$$

$$p(B=1|F=0,\mathcal{V}) = \frac{1+\#(B=1,F=0)}{2+\#(B=1,F=0)+\#(B=0,F=0)} = \frac{1+0}{2+0+12} = \frac{1}{14},$$

$$p(Q=1|D=1,T=1,P=1,\mathcal{V}) = \frac{1+1}{2+1} = \frac{2}{3},$$

$$p(Q=1|D=1,T=1,P=0,\mathcal{V}) = \frac{1+1}{2+1} = \frac{2}{3},$$

$$p(Q=1|D=1,T=0,P=1,\mathcal{V}) = \frac{1+1}{2+1} = \frac{2}{3},$$

$$p(Q=1|D=1,T=0,P=0,\mathcal{V}) = \frac{1+1}{2+1} = \frac{2}{3},$$

$$p(Q=1|D=0,T=1,P=1,\mathcal{V}) = \frac{1+1}{2+1} = \frac{2}{3},$$

$$p(Q=1|D=0,T=1,P=0,\mathcal{V}) = \frac{1+2}{2+2} = \frac{3}{4},$$

$$p(Q=1|D=0,T=0,P=1,\mathcal{V}) = \frac{1+1}{2+5} = \frac{2}{7},$$

$$p(Q=1|D=0,T=0,P=0,\mathcal{V}) = \frac{1+0}{2+3} = \frac{1}{5},$$

$$p(W=1|P=1,F=1,\mathcal{V}) = \frac{1+1}{2+1} = \frac{2}{3},$$

$$p(W=1|P=1,F=0,\mathcal{V}) = \frac{1+2}{2+7} = \frac{1}{3},$$

$$p(W=1|P=0,F=1,\mathcal{V}) = \frac{1+1}{2+2} = \frac{1}{2},$$

$$p(W=1|P=0,F=0,\mathcal{V}) = \frac{1+1}{2+5} = \frac{2}{7},$$

$$p(M=1|P=1,R=1,\mathcal{V}) = \frac{1+1}{2+1} = \frac{2}{3},$$

$$p(M=1|P=1,R=0,\mathcal{V}) = \frac{1+2}{2+7} = \frac{1}{3},$$

$$p(M=1|P=0,R=1,\mathcal{V}) = \frac{1+1}{2+2} = \frac{1}{2},$$

$$p(M=1|P=0,R=0,\mathcal{V}) = \frac{1+0}{2+5} = \frac{1}{7},$$

$$p(PJ=1|F=1,R=1,\mathcal{V}) = \frac{1+0}{2+1} = \frac{1}{3},$$

$$p(PJ=1|F=1,R=0,\mathcal{V}) = \frac{1+1}{2+2} = \frac{1}{2},$$

$$p(PJ=1|F=0,R=1,\mathcal{V}) = \frac{1+2}{2+2} = \frac{3}{4},$$

$$p(PJ=1|F=0,R=0,\mathcal{V}) = \frac{1+4}{2+10} = \frac{5}{12}.$$

Running `demoPrinter.m` with our new probability tables gives:

```
p(fuse|burn=yes, jam=yes, wrinkled=no, multpages=no, quality=no):
fuse    =yes    6.186152e-01
fuse    =no     3.813848e-01
```

The probability of there being a fuse assembly malfunction is now 0.619.

4. An associated junction tree can be obtained from the belief network (see Figure 1).

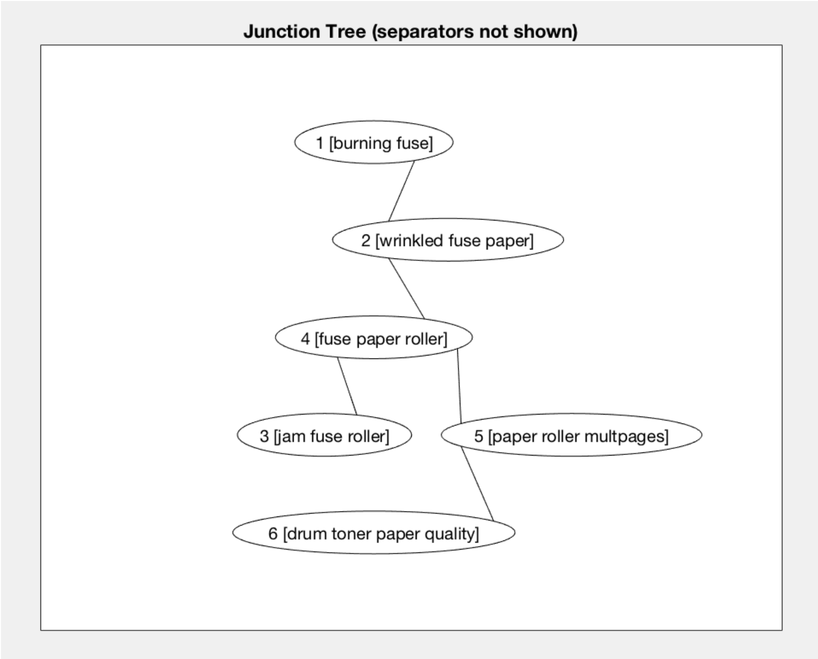


Figure 1: Junction Tree corresponding to the Printer Nightmare Belief Network.

The following lines of code are added to `demoPrinter.m`. The BRML Toolbox provides the `absorption.m` function, which can be utilised to implement max-absorption by calling the function with the flag ‘max’.

```
condi_pot = condpot(setpot(jointpot,[burning quality wrinkled multpages jam], ...
                                    [yes no no no yes]));

% setup the Junction Tree
[jtpot, jtsep, infostruct]=jtree(condi_pot);


% do full round of max-absorption
[jtpot2, jtsep2] = absorption(jtpot, jtsep, infostruct, 'max');
for i=1:length(jtpot2)    % find max over each clique
    [newpot JTmaxstate(jtpot2{i}.variables)] = maxpot(jtpot2{i}, [], 0);
end
disp(['Max-absorbtion on JT has optimal state: ', num2str(JTmaxstate), ...
      '. With probability ' num2str(newpot.table)])
```

Using the flat Beta prior and running the code above outputs the following:

```
Max-absorbtion on JT has optimal state: 1  2  2  2  2. With probability 0.24706
```

Effectively, this indicates that the most likely joint diagnosis corresponds to all parent variables set to 2 except Fuse which is set to 1. In other words, to only the fuse malfunctioning and all other diagnostic ‘variables’ working properly (with probability 0.247 (3 s.f.)).

5. The code utilised to solve this question is presented below, with some adjustments made with respect to Part 4.

```
condi_pot_ii = condpot(setpot(jointpot,[burning jam], ...
                               [yes yes]));

% setup the Junction Tree
[jtpot, jtsep, infostruct]=jtree(condi_pot_ii);


% do full round of max-absorption
[jtpot2, jtsep2] = absorption(jtpot, jtsep, infostruct, 'max');
for i=1:length(jtpot2)    % find max over each clique
    [newpot JTmaxstate(jtpot2{i}.variables)] = maxpot(jtpot2{i}, [], 0);
end
disp(['Max-absorbtion on JT has optimal state: ', num2str(JTmaxstate), ...
      '. With probability ' num2str(newpot.table)])
```

In this occasion, only `burning` and `jam` are set to `yes` (there is a fault). Other child variable values are unknown. Using the flat Beta prior and running the code above outputs the following:

```
Max-absorbtion on JT has optimal state: 1  2  2  1  2  0  2  1  2. With probability 1
```

Here the values for the last four variables are irrelevant; note how a vector entry has a zero - this indicates that a value has not been assigned for this entry. The last three values `2 1 2` correspond to `quality`, `wrinkled` and `multpages`: these are child variables. The first five entries, `1 2 2 1 2`, represent the most likely state of the diagnostic variables. This corresponds to a fuse assembly malfunction and poor paper quality, with all other diagnostic units functioning properly. The max-absorption method can be used to compute this efficiently once we have obtained a junction tree. Passing messages in both forward and backward directions across all separators, according to a valid schedule, the most likely joint state can be read off from maximising the state of the clique potentials.

Exercise 9.9

We use the result that the normalisation constant of Dirichlet distribution is:

$$Z(u) = \frac{\prod \Gamma(\mathbf{u})}{\Gamma(\sum(\mathbf{u}))} = \frac{\prod_{i=1}^{n} \Gamma(u_i)}{\Gamma(\sum_{i=1}^{n}(u_i))}$$

where $\mathbf{u} = (u_1, ..., u_n)$ and Γ denotes the Gamma function. Then, by simple algebaic manipulations, we have that:

$$\begin{aligned}
p(\mathcal{D}|M) &= \prod_k \prod_n p(v_k^n|\mathrm{pa}(v_k^n)) = \prod_k \prod_j \frac{Z(\mathbf{u}'(v_k;j))}{Z(\mathbf{u}(v_k;j))} \\
&= \prod_k \prod_j \frac{\frac{\prod \Gamma(\mathbf{u}'(v_k;j))}{\Gamma(\sum(\mathbf{u}'(v_k;j)))}}{\frac{\prod \Gamma(\mathbf{u}'(v_k;j))}{\Gamma(\sum(\mathbf{u}'(v_k;j)))}} \\
&= \prod_k \prod_j \frac{\prod \Gamma(\mathbf{u}'(v_k;j))}{\Gamma(\sum(\mathbf{u}'(v_k;j)))} \frac{\Gamma(\sum(\mathbf{u}'(v_k;j)))}{\prod \Gamma(\mathbf{u}'(v_k;j))} \\
&= \prod_k \prod_j \frac{\Gamma(\sum(\mathbf{u}'(v_k;j)))}{\Gamma(\sum(\mathbf{u}'(v_k;j)))} \frac{\prod \Gamma(\mathbf{u}'(v_k;j))}{\prod \Gamma(\mathbf{u}'(v_k;j))} \\
&= \prod_k \prod_j \frac{\Gamma(\sum_i u_i(v_k;j))}{\Gamma(\sum_i u_i'(v_k;j))} \frac{\prod_i \Gamma(u_i(v_k;j))}{\prod_i \Gamma(u_i'(v_k;j))} \\
&= \prod_k \prod_j \frac{\Gamma(\sum_i u_i(v_k;j))}{\Gamma(\sum_i u_i'(v_k;j))} \prod_i \left[\frac{\Gamma(u_i(v_k;j))}{\Gamma(u_i'(v_k;j))}\right]
\end{aligned}$$

Exercise 9.10

1. A belief network should be thought of as directed acyclic graph or DAG but where the nodes are random variables (i.e. has some additional probability information). If we restrict to belief networks of 8 nodes then we know by question 9.14 that there are at least 2^{28} such graphs. If we further restrict to those with 2 parents we have an enormous reduction on this number. This is best calcuated using the **Network Scoring** technique where

the ancestral order is given and the constraint that each variable has maximally two parents is assumed. To do this we assume an ancestral ordering on 8 nodes. Without loss of generality, say, $x_1, x_2, x_3, x_4.x_5, x_6, x_7, x_8$ with Dirichlet hyperparameters set to unity and limit the number of parents of each variable to be at most two. In this situation we are able to generate all possible graph structures.

We can also do this algebraically. If $x_1, x_2, x_3, x_4.x_5, x_6, x_7, x_8$ is the ancestral ordering then x_8 has possibly parenting over the 7 choices $x_1, x_2, x_3, x_4.x_5, x_6, x_7$. By considering no parents, one parent or two parents, we have:

$$\binom{7}{0} + \binom{7}{1} + \binom{7}{2}$$

Likewise, x_7 has possibly parenting over the 6 choices $x_1, x_2, x_3, x_4.x_5, x_6$:

$$\binom{6}{0} + \binom{6}{1} + \binom{6}{2}$$

In total, by summing over 8 variables, we have:

$$\sum_{i=1}^{8}\sum_{j=0}^{2}\binom{i}{j} = \sum_{i=1}^{8}\binom{i}{0} + \binom{i}{1} + \binom{i}{2} = 91$$

2. The overall time is

$$|a| \times \Big(1 + \frac{|a| \times (|a|+1)}{2}\Big),$$

where a denotes the number of variables in the ancestral ordering.

Exercise 9.13

For this exercise we are given a matrix $\mathbf{X}$ of dimension $D \times N$ in which each column represents a training (multivariate) data point. We need to create the function *ChowLiu.m* which takes matrix $\mathbf{X}$ as input and return a sparse matrix $\mathbf{A}$ which represents the adjacency matrix of the graph that generates the Maximum Spanning Tree.

We use the following algorithm to solve this task:

1. We need to calculate the mutual information between each pair of variables. In this case we are given 10 variables $\{1...10\}$. This will generate a 10×10 symmetric matrix in which the ij_{th} element will contain the mutual information between the variables. These will represent the weights.

$$w_{i,j} = MI(X_i, X_j)$$

```
weights =
    0.9460    0.1001    0.0064    0.0030    0.0059    0.0057    0.0001    0.0030    0.0018    0.0000
    0.1001    0.9850    0.0410    0.0083    0.0140    0.0252    0.0001    0.0038    0.0037    0.0002
    0.0064    0.0410    1.4194    0.2773    0.0025    0.0048    0.0014    0.0022    0.0090    0.0007
    0.0030    0.0083    0.2773    1.5471    0.0034    0.0010    0.0007    0.0021    0.0782    0.0005
    0.0059    0.0140    0.0025    0.0034    1.4580    0.0017    0.0032    0.0013    0.0007    0.0004
    0.0057    0.0252    0.0048    0.0010    0.0017    1.5419    0.1825    0.1104    0.0023    0.0179
    0.0001    0.0001    0.0014    0.0007    0.0032    0.1825    0.9451    0.0245    0.0003    0.0038
    0.0030    0.0038    0.0022    0.0021    0.0013    0.1104    0.0245    1.4863    0.0054    0.0728
    0.0018    0.0037    0.0090    0.0782    0.0007    0.0023    0.0003    0.0054    1.4162    0.0015
    0.0000    0.0002    0.0007    0.0005    0.0004    0.0179    0.0038    0.0728    0.0015    0.9451
```

2. Since the matrix calculated with the mutual information as the weight is a symmetric matrix, we will take into account only the upper triangular part of the matrix (without considering the diagonal).

```
new_mat =
         0    0.1001    0.0064    0.0030    0.0059    0.0057    0.0001    0.0030    0.0018    0.0000
         0         0    0.0410    0.0083    0.0140    0.0252    0.0001    0.0038    0.0037    0.0002
         0         0         0    0.2773    0.0025    0.0048    0.0014    0.0022    0.0090    0.0007
         0         0         0         0    0.0034    0.0010    0.0007    0.0021    0.0782    0.0005
         0         0         0         0         0    0.0017    0.0032    0.0013    0.0007    0.0004
         0         0         0         0         0         0    0.1825    0.1104    0.0023    0.0179
         0         0         0         0         0         0         0    0.0245    0.0003    0.0038
         0         0         0         0         0         0         0         0    0.0054    0.0728
         0         0         0         0         0         0         0         0         0    0.0015
         0         0         0         0         0         0         0         0         0         0
```

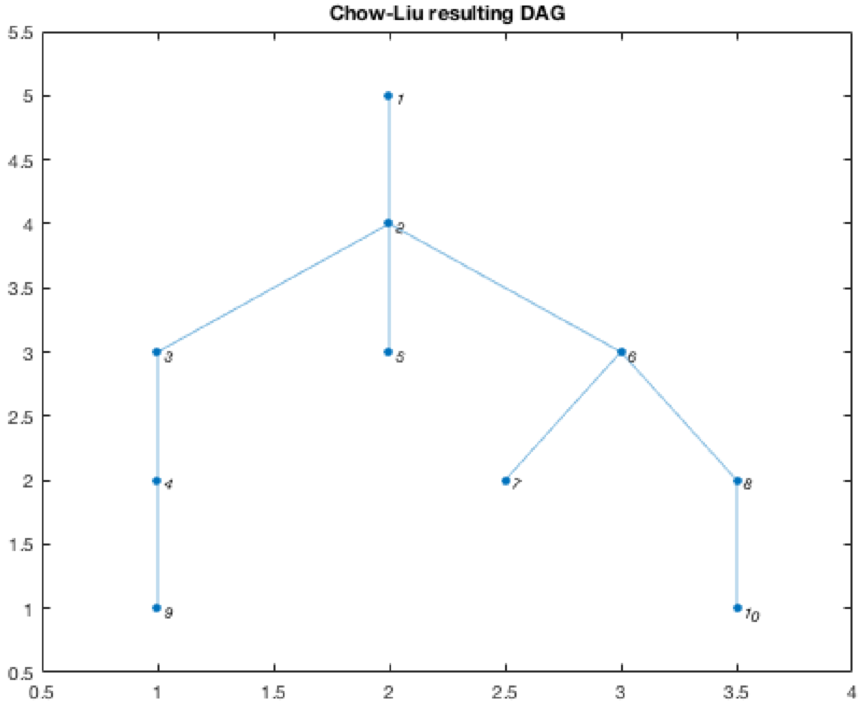


Figure 2: resulting DAG: Chow-Liu tree

3. We then generate a sparse matrix and sort the non-zero elements in *descending* order (below a list of indexes of sorted items).

3,4	1,6	5,6
6,7	8,9	9,10
6,8	3,6	3,7
1,2	7,10	5,8
4,9	2,8	4,6
8,10	2,9	5,9
2,3	4,5	4,7
2,6	5,7	3,10
7,8	1,8	4,10
6,10	1,4	5,10
2,5	3,5	7,9
3,9	6,9	2,10
2,4	3,8	1,7
1,3	4,8	2,7
1,5	1,9	1,10

4. Once we have the sorted indexes we can proceed to make use of the *BRML Toolbox* function *spantree.m*. As instructed in the function documentation: to find a maximum weight spanning tree based on an edge-list, call this routine with the edges ordered heaviest first. This sorted edge-list was calculated in the step above.

5. We have all we need in order to get the plot requested with edges oriented away from variable 1 as seen in Figure 2.

Below the code to generate our function *ChowLiu.m*

```
function A = ChowLiu(X)
%ChowLiu returns a sparse matrix representing the adjacency matrix of the
% resulting graph (Maximum Spanning Tree)
% Input X: is the Matrix with the training data
% Output A: sparse matrix representing the adjacency matrix, can be used to
% graph the tree.

```

```
% Import all the libraries needed, in this case only BRML Toolbox. We have
% also used a package created by the University of Manchester
% http://www.cs.man.ac.uk/~pococka4/MIToolbox.html
import brml.*

% create the weight matrix with the Mutual Information between variables
weights = zeros(10,10);
for i = 1:10
    for j = 1:10
        weights(i,j) = mi(X(i,:)',X(j,:)');
    end
end

% seller the upper triangular matrix and exclude the main diagonal
new_mat = triu(weights,1);
% get the indexes for the non-zero values
[i,j,s] = find(new_mat);
% sort the edges in descending order based on the value
new_weights = sortrows([i j s], -3);
% prepare data for the spantree.m function
indexes_sorted = new_weights(:,[1 2])
% generate the sparse Matrix A to be returned by the function
[A dum1 dum2] = spantree(indexes_sorted)
end
```

We use our function to create the picture requested

```
% use our function to create the sparse matrix. We load the data given by
% the exercise in advance.
result_chow_liu = ChowLiu(ChowLiuData);
% Generate a graph based on the result, which will be used for the plot
G = graph(result_chow_liu);
% plot the tree
plot(G,'Layout','layered');
title('Chow-Liu resulting DAG')
```

Exercise 9.14

Let G be a graph with $N > 1$ nodes. In this question we need to prove that the number of DAGs are bounded as follows:

$$\prod_{n=1}^{N} 2^{n-1} = 2^{N(N-1)/2} < |\text{DAGs}| < N! 2^{N(N-1)/2}$$

We have another obvious bounding. Consider:

$$\text{undirected graphs} < |\text{DAGs}| < \text{directed graphs}$$

We start by treating the inequality on the left, the lower bound. First note that by Gauss' arithmetic summation we have:

$$\begin{aligned}\prod_{n=1}^{N} 2^{n-1} &= 2^0 + 2^1 + 2^2 + ... + 2^{N-1}\\ &= 2^{1+2+3+...+(N-1)}\\ &= 2^{((N-1)+1)(N-1)/2}\\ &= 2^{N(N-1)/2}\end{aligned}$$

In a graph, each edge is uniquely determined by the two nodes at each end, if we do not allow for loops. Therefore the number of edges is equal to the number of size 2 subsets selected from the set of N nodes. This is $\binom{N}{2}$.

$$\binom{N}{2} = \frac{N!}{2!(N-2)!} = \frac{N(N-1)}{2}$$

So, by deciding whether or not to include an edge or not, we find that we have

$$2^{N(N-1)/2}$$

undirected graphs, as required.

We now treat the inequality on the right, the upper bound.

We ignore of a moment cyclic graphs and concentrate on inserting ordering to our undirected graphs. By the construction, we have N choices, then $N-1$ choices at the subsequent step, ..., down to 2 and then 1 choice. In other words,

$$N \times N - 1 \times ... \times 2 \times 1 = N!$$

Therefore, we are bounded above by:

$$N!2^{N(N-1)/2}$$

Together, we have:

$$\prod_{n=1}^{N} 2^{n-1} = 2^{N(N-1)/2} < |\text{DAGs}| < N!2^{N(N-1)/2}$$

Exercise 10.3

Each document given to us is represented as a row vector in the matrices given, we have: 6 training samples for *Politics* and 7 training samples for *Sports*.

We solved this exercise based on the code given in the *BRML Toolbox* for solving a Naive Bayes classifier. With the subtle difference that in the example provided in that case, the training samples are represented by column vectors. Hence we modified the code as shown below:

```
function demoNaiveBayes_10_3
%demoNaiveBayes_10_3 Naive Bayes to solve problem 10.3
import brml.*

xP=[1 0 1 1 1 0 1 1; % Politics
    0 0 0 1 0 0 1 1;
    1 0 0 1 1 0 1 0;
    0 1 0 0 1 1 0 1;
    0 0 0 1 1 0 1 1;
    0 0 0 1 1 0 0 1];

xS=[1 1 0 0 0 0 0 0; % Sport
    0 0 1 0 0 0 0 0;
    1 1 0 1 0 0 0 0;
    1 1 0 1 0 0 0 1;
    1 1 0 1 1 0 0 0;
    0 0 0 1 0 1 0 0;
    1 1 1 1 1 0 1 0];

pP = size(xP,2)/(size(xP,2) + size(xS,2)); pS =1-pP; % ML class priors pP = p(c=P), pS=p(c=S)

mP = mean(xP); % ML estimates of p(x=1|c=P)
mS = mean(xS); % ML estimates of p(x=1|c=S)

xtest=[1 0 0 1 1 1 1 0]'; % test point

npP = pP*prod(mP.^xtest.*(1-mP).^(1-xtest)); % p(x,c=P)
npS = pS*prod(mS.^xtest.*(1-mS).^(1-xtest)); % p(x,c=S)

pxP = npP/(npP+npS); % probability that x is about Politics
disp(['probability x is about Politics = ',num2str(pxP)]);
```

Below we can see the result after running the code:

```
>> demoNaiveBayes_10_3
probability x is about Politics = 0.41219
```

Exercise 10.5

1. Our training dataset consists of a set of pairs (x^n, c^n), $n = 1, ..., N$. Recall that each x^n is a D-dimensional vector. We denote x_i^n as the i-th component of x^n, where $x_i^n \in \{0, 1\}$. We wish to estimate $p(x_i = 1|c)$ for each attribute of the two classes. Denote $\theta_i^c = p(x_i = 1|c)$ and hence, $p(x_i = 0|c) = 1 - \theta_i^c$. Then, the probability of observing a given x is written,

$$p(x|c) = \prod_{i=1}^{D} p(x_i|c) = \prod_{i=1}^{D} (\theta_i^c)^{x_i}(1 - \theta_i^c)^{1-x_i},$$

making the Naive Bayes conditional independence assumptions. Above, since $x_i^n \in \{0, 1\}$ and each i term contributes a factor θ_i^c if $x_i = 1$ or $1 - \theta_i^c$ if $x_i = 0$. Hence, given that the training data is i.i.d. generated, the log likelihood of the attributes and the class labels is,

$$L = \sum_n \log p(x^n, c^n) = \sum_n \log p(c^n) \prod_i p(x_i^n|c^n) = \Big\{ \sum_{i,n} x_i^n \log \theta_i^{c^n} + (1 - x_i^n) \log(1 - \theta_i^{c^n}) \Big\} + n_0 \log p(c = 0) + n_1 \log p(c = 1), \tag{1}$$

where n_0 is the number of data points that are not spam $(c = 0)$ and n_1 is the number of data points that are spam $(c = 1)$. In terms of the parameters,

$$L = \sum_{i,n} \{[[x_i^n = 1, c^n = 0]] \log \theta_i^0 + [[x_i^n = 0, c^n = 0]] \log(1 - \theta_i^0) + [[x_i^n = 1, c^n = 1]] \log \theta_i^1 + [[x_i^n = 0, c^n = 1]] \log(1 - \theta_i^1)\}$$

$$+ n_0 \log p(c = 0) + n_1 \log p(c = 1),$$

where $[[x_i^n = 1, c^n = 0]] = 1$ if $x_i^n = 1, c^n = 0$ and $[[x_i^n = 1, c^n = 0]] = 0$ otherwise, etc. By differentiating L with respect to θ_i^c and setting this expression equal to zero, one can find the optimal Maximum Likelihood parameter θ_i^c:

$$\theta_i^c = p(x_i = 1|c) = \frac{\sum_{n=1}^{N} [[c^n = c, x_i^n = 1]]}{\sum_{n=1}^{N} [[c^n = c, x_i^n = 1]] + [[c^n = c, x_i^n = 0]]} = \frac{\sum_{n=1}^{N} [[c^n = c, x_i^n = 1]]}{\sum_{n=1}^{N} [[c^n = c]]} = \frac{count(x_i = 1|c)}{count(c)}.$$

Here $\sum_{n=1}^{N} [[c^n = c]] = count(c)$ corresponds to the number of times we encounter the label c in the training set. Similarly, $\sum_{n=1}^{N} [[c^n = c, x_i^n = 1]] = count(x_i = 1|c)$ is the number of times $x_i = 1$ for label c. $p(x_i = 1|c = 1)$, $p(x_i = 1|c = 0)$, $i = 1, ..., D$ can be obtained from the previous expression. In similar fashion, optimising equation (1) with respect to $p(c)$:

$$p(c) = \frac{\sum_{n=1}^{N} [[c^n = c]]}{N} = \frac{count(c)}{N},$$

i.e. the number of times class c occurs divided by the total number of data points. Parameter $p(c = 1)$, the probability of the class being spam, can be calculated by counting occurrences as above to fully specify the system. Note that other probabilities e.g. $p(x_i = 0|c = 1)$ are given by the normalisation requirement in this binary class case.

2. Counting the number of occurrences in the training data, as shown above, corresponds to maximum likelihood learning of the table entries $p(c)$, $p(x_i|c)$, for a fully observed dataset. Given such trained model, $p(x, c)$, we can then use Bayes' Rule to form a classifier for a novel input vector x^*:

$$p(c|x^*) = \frac{p(x^*|c)p(c)}{p(x^*)} = \frac{p(x^*|c)p(c)}{\sum_c p(x^*|c)p(c)},$$

In the binary class case, a novel input x^* can be classified as spam (class 1) if $p(c = 1|x^*) > p(c = 0|x^*)$. Using Bayes'rule, this expression becomes,

$$\frac{p(x^*|c = 1)p(c = 1)}{p(x^*)} > \frac{p(x^*|c = 0)p(c = 0)}{p(x^*)}.$$

Taking logs,

$$\log p(x^*|c = 1) + \log p(c = 1) - \log p(x^*) > \log p(x^*|c = 0) + \log p(c = 0) - \log p(x^*).$$

Dropping the normalisation constant, $\log p(x^*)$, from both sides and denoting x_i^* as the i-th component of x^*, from the definition of the classifier:

$$\sum_i \log p(x_i^*|c=1) + \log p(c=1) > \sum_i \log p(x_i^*|c=0) + \log p(c=0).$$

We can therefore classify $x*$ as spam $(c=1)$ if,

$$\sum_i \{x_i^* \log \theta_i^{c=1} + (1-x_i^*)\log(1-\theta_i^{c=1})\} + \log p(c=1) > \sum_i \{x_i^* \log \theta_i^{c=0} + (1-x_i^*)\log(1-\theta_i^{c=0})\} + \log p(c=0),$$

using our previous definition of θ_i^c. Note that this expression can also be expressed as: classify x^* as spam $(c=1)$ if $\sum_i w_i x_i^* + a > 0$ for some choice of weights w_i and constant a i.e. w specifies a hyperplane in the attribute space and x^* is classified as spam if it lies on the positive side of the hyperplane.

3. Consider trying to classify a new e-mail containing the word 'viagra'. In the spam training data, the word 'viagra' never appears. This means that for this particular word x_i, $p(x_i, c=1) = 0$. Therefore, ML estimates for Naive Bayes will make the extremely confident classification $p(c=1|x_i) = p(spam|viagra) = 0$ (note also $p(notspam|viagra) = 0$), which contrasts with observation. This suggests potential problems for utilising Naive Bayes on sparse data.

Possible ways of countering this effect may involve smoothing methods e.g. Laplace smoothing. This method solves the previous problem by giving 'viagra' a small non-zero probability/small number of frequency counts for both classes, so that posterior probabilities don't suddenly drop to zero. Another option to smooth the probabilities is a Bayesian approach: using priors on the probabilities $p(x_i = s|c) = \theta_s^i(c)$, hence discouraging extreme values.

Spammers can easily fool a Naive Bayes spam filter utilising a technique known as 'Bayesian poisoning'. This involves introducing lots of 'hammy' words into e-mails; these may be collected from 'legitimate' sources such as news or literary works. Each of these words is treated independently and again and again count towards the probability of 'not-spam'. Other options for fooling the NB spam filter could be replacing text for pictures, or repacling 'viagra' for 'v!agra' or 'viagraa' (these words will be encountered less frequently by the spam filter).

Exercise 23.4

Given the string `rgenmonleunosbpnntje vrancg` typed with 'stubby fingers', the most likely correct English sentence intended is `the monkey is on the branch`. `NMax` (the number of most probable joint states displayed) is set to 700; `the monkey is on the branch` is the 659th most likely probable state but the most likely **correct** English sentence.

We are asked to get the value of $\log p(h_{1:27}|v_{1:27})$ at each of the iterations (for each of the sequences generated). In order to achieve this we need to take into account the messages passed from the leaves to a specific root that we can choose, in this case we have chosen the very first node, once we have reached the root. We then make the calculation and print the result. We show the code to produce the results:

```
function demoHMMbigram
%DEMOHMMBIGRAM demo of HHM for the bigram typing scenario
import brml.*
load freq % http://www.data-compression.com/english.shtml
l = {'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z',' '};
load typing % get the A transition and B emission matrices
% figure(1); imagesc(A); set(gca,'xtick',1:27); set(gca,'xticklabel',l); set(gca,'ytick',1:27); set(gca,'yticklabel',l)
% colorbar; colormap hot; title('transition')
% figure(2); imagesc(B); set(gca,'xtick',1:27); set(gca,'xticklabel',l); set(gca,'ytick',1:27); set(gca,'yticklabel',l)
% colorbar; colormap hot; title('emission')
ph1=condp(ones(27,1)); % uniform first hidden state distribution

s = 'rgenmonleunosbpnntje vrancg';  Nmax=659; % observed sequence
v=double(s)-96; v=replace(v,-64,27); % convert to numbers

% find the most likely hidden sequences by defining a Factor Graph:
T = length(s);
hh=1:T; vv=T+1:2*T;
```

```
empot=array([vv(1) hh(1)],B);
prior=array(hh(1),ph1);
pot{1} = multpots([setpot(empot,vv(1),v(1)) prior]);
for t=2:T
    tranpot=array([hh(t) hh(t-1)],A);
    empot=array([vv(t) hh(t)],B);
    pot{t} = multpots([setpot(empot,vv(t),v(t)) tranpot]);
end
FG = FactorGraph(pot);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% BEGIN: Exercise 23.4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Max-Prod calculation
[maxstate maxval mess]=maxNprodFG(pot,FG,Nmax);

% Insert code to calculate the log-likelihood
[dum1 fact2var]=FactorConnectingVariable(hh(1),FG); % can choose any of the variable nodes
tmpmess = multpots(mess(fact2var));
FGloglik = log(sum(tmpmess.table));

for n=1:Nmax
    maxstatearray(n,:)= horzcat(maxstate(n,1:length(s)).state);
end

strs=char(replace(maxstatearray+96,123,32)); % make strings from the decodings
[r c] = size(strs);

% Print both the strings and the loglikelihood of each one
for i = 1:r
    disp([num2str(i) ' & ' strs(i,:) ' & ' num2str(FGloglik(i)) ' \\'])
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% END: Exercise 23.4
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fid=fopen('brit-a-z.txt','r'); % see http://www.curlewcommunications.co.uk/wordlist.html for Disclaimer and Copyright
w=textscan(fid,'%s'); w=w{1}; % get the words from the dictionary

% discard those decodings that are not in the dictionary:
% (An alternative would be to just compute the probability of each word in
% the dictionary to generate the observed sequence.)
for t=1:Nmax
    str = strs(t,:); % current string
    spac = strfind(str,' '); % chop the string into words
    spac = [spac length(str)+1]; % find the spaces
    start=1; val=1;
    for i=1:length(spac) % go through all the words in the string
        wd{i} = str(start:(spac(i)-1));
        start=spac(i)+1;
        if isempty(find(strcmp(wd{i},w))) % check if word is in the dictionary
            val=0; break
        end
    end
    if val; disp([num2str(t) ': ' str]);end
end
```

And below are the results printed for each iteration, until the 659*th* which is the one generating our final sentence.

1	the monorinos on the veanch	-92.2057
2	the monorinos on the beanch	-92.2702
3	the monorinos on the branch	-92.2726
4	then onorinos on the veanch	-92.3498
5	the monowinos on the veanch	-92.4014
6	then onorinos on the beanch	-92.414
7	then onorinos on the branch	-92.4168
8	the monowinos on the beanch	-92.4644
9	the monowinos on the branch	-92.4681
10	the monorinos on the franch	-92.511
11	then onowinos on the veanch	-92.5373
12	the monorinos on the veanct	-92.5728
13	then onowinos on the beanch	-92.5995
14	then onowinos on the branch	-92.6013
15	the monorinos on the beanct	-92.6326
16	the monorinos on the branct	-92.6348
17	then onorinos on the franch	-92.6481
18	the monowinos on the franch	-92.6973
19	then onorinos on the veanct	-92.7097
20	the monowinos on the veanct	-92.7568
21	then onorinos on the beanct	-92.7685
22	then onorinos on the branct	-92.7707
23	the monorinis on the veanch	-92.7949
24	then inorinos on the veanch	-92.7979
25	the monowinos on the beanct	-92.8192
26	the monowinos on the branct	-92.822
27	then onowinos on the franch	-92.8335
28	the monorinis on the beanch	-92.8541
29	the monorinis on the branch	-92.8562
30	then inorinos on the beanch	-92.8584
31	then inorinos on the branch	-92.8607
32	the monorinos on the franct	-92.8724
33	then onowinos on the veanct	-92.8977
34	the minorinos on the veanch	-92.9136
35	then onorinis on the veanch	-92.932
36	then onowinos on the beanct	-92.9554
37	then onowinos on the branct	-92.9578
38	the minorinos on the beanch	-92.9758
39	the minorinos on the branch	-92.9794
40	the monowinis on the veanch	-92.9849
41	then inowinos on the veanch	-92.9888
42	then onorinis on the beanch	-92.9976
43	then onorinis on the branch	-92.9994
44	then onorinos on the franct	-93.0097
45	the monorinos on the cranch	-93.0401
46	the monorinos on the tanch	-93.0451
47	the monowinis on the beanch	-93.0474
48	the monowinis on the branch	-93.0505
49	then inowinos on the beanch	-93.0525
50	the monorinos on the ceanch	-93.054
51	then inowinos on the branch	-93.0554
52	the monory is on the veanch	-93.0583
53	the monowinos on the franct	-93.0637
54	the monorinis on the franch	-93.0969
55	then inorinos on the franch	-93.1
56	the minowinos on the veanch	-93.1075
57	the monory is on the beanch	-93.1179
58	the monory is on the branch	-93.1201
59	then onowinis on the veanch	-93.1253
60	the monorinis on the veanct	-93.1558

61	then inorinos on the veanct	-93.1584
62	the minowinos on the beanch	-93.165
63	the minowinos on the branch	-93.1676
64	then onorinos on the cranch	-93.1767
65	then onorinos on the tanch	-93.1817
66	then onowinis on the beanch	-93.185
67	then onowinis on the branch	-93.1875
68	then onorinos on the ceanch	-93.1922
69	then onory is on the veanch	-93.1965
70	then onowinos on the franct	-93.2018
71	the minorinos on the franch	-93.2185
72	the monorinis on the beanct	-93.2232
73	the monorinis on the branct	-93.2263
74	then inorinos on the beanct	-93.2281
75	then inorinos on the branct	-93.23
76	the monowinos on the cranch	-93.2363
77	then onorinis on the franch	-93.241
78	the monowinos on the tanch	-93.2422
79	the monorinos on the ranch	-93.2434
80	the monowinos on the ceanch	-93.2485
81	the monorinos on the granch	-93.2597
82	then onory is on the beanch	-93.2622
83	then onory is on the branch	-93.2645
84	the monory os on the veanch	-93.273
85	the minorinos on the veanct	-93.2834
86	the monowinis on the franch	-93.2898
87	the monorinod on the veanch	-93.2936
88	then inowinos on the franch	-93.2944
89	then onorinis on the veanct	-93.3042
90	the monory os on the beanch	-93.3328
91	the monory os on the branch	-93.3356
92	the minorinos on the beanct	-93.3446
93	the minorinos on the branct	-93.3465
94	the monowinis on the veanct	-93.3514
95	the monorinod on the beanch	-93.3539
96	then inowinos on the veanct	-93.3543
97	the monorinod on the branch	-93.3561
98	the monory is on the franch	-93.3585
99	then onorinis on the beanct	-93.3633
100	then onorinis on the branct	-93.3651
101	then onowinos on the cranch	-93.372
102	then onowinos on the tanch	-93.3768
103	then onorinos on the ranch	-93.378
104	then onowinos on the ceanch	-93.3835
105	then inorinis on the veanch	-93.3897
106	then onorinos on the granch	-93.3945
107	the monorinos on the cranct	-93.4062
108	the mo orinos on the veanch	-93.4094
109	the minowinos on the franch	-93.4101
110	then onory os on the veanch	-93.4107
111	the monorinos on the tanct	-93.4132
112	the monowinis on the beanct	-93.415
113	the monowinis on the branct	-93.417
114	then inowinos on the beanct	-93.4187
115	the monorinos on the ceanct	-93.42
116	then inowinos on the branct	-93.4207
117	the monory is on the veanct	-93.4227
118	then onowinis on the franch	-93.4293
119	the monowinos on the ranch	-93.4311
120	then onorinod on the veanch	-93.4332

121	the monowinos on the granch	-93.4473
122	then inorinis on the beanch	-93.4534
123	then inorinis on the branch	-93.4556
124	the monley is on the veanch	-93.4579
125	the onorinos on the veanch	-93.4603
126	the monorinis on the franct	-93.4658
127	then inorinos on the franct	-93.4695
128	the mo orinos on the beanch	-93.4754
129	the minowinos on the veanct	-93.4759
130	then onory os on the beanch	-93.4773
131	the mo orinos on the branch	-93.4781
132	then onory os on the branch	-93.479
133	the monowinod on the veanch	-93.4869
134	the monory is on the beanct	-93.4892
135	the monory is on the branct	-93.4913
136	then onowinis on the veanct	-93.4965
137	then onorinod on the beanch	-93.4989
138	then onorinod on the branch	-93.5008
139	then onory is on the franch	-93.5034
140	the minorinis on the veanch	-93.513
141	the monley is on the beanch	-93.523
142	the onorinos on the beanch	-93.5241
143	the monley is on the branch	-93.526
144	the onorinos on the branch	-93.527
145	the monorunos on the veanch	-93.5311
146	the minowinos on the beanct	-93.5392
147	the minowinos on the branct	-93.5411
148	the monowinod on the beanch	-93.5484
149	the monowinod on the branch	-93.5508
150	then onorinos on the cranct	-93.5514
151	then o orinos on the veanch	-93.5533
152	then onorinos on the tanct	-93.5565
153	then onowinis on the beanct	-93.5582
154	then onowinis on the branct	-93.5599
155	then onorinos on the ceanct	-93.5622
156	then onory is on the veanct	-93.5652
157	then onowinos on the ranch	-93.5728
158	the minorinis on the beanch	-93.5748
159	the monory os on the franch	-93.5753
160	the minorinis on the branch	-93.5767
161	then inowinis on the veanch	-93.5839
162	the monleinos on the veanch	-93.586
163	the minorinos on the franct	-93.5862
164	then onowinos on the granch	-93.5896
165	the monorunos on the beanch	-93.5917
166	the monorunos on the branch	-93.5933
167	the monorinod on the franch	-93.5957
168	then onley is on the veanch	-93.5999
169	the monowinos on the cranct	-93.6014
170	the mo owinos on the veanch	-93.6033
171	then onorinis on the franct	-93.6063
172	the monowinos on the tanct	-93.6072
173	the monorinos on the ranct	-93.609
174	the monowinos on the ceanct	-93.6144
175	then o orinos on the beanch	-93.6173
176	then o orinos on the branch	-93.619
177	the monorinos on the grianct	-93.6261
178	then onowinod on the veanch	-93.6267
179	then onory is on the beanct	-93.6291
180	then onory is on the branct	-93.6308
181	the monorinis on the cranch	-93.6387
182	the monory os on the veanct	-93.6393
183	then inorinos on the cranch	-93.6414
184	the monorinis on the tanch	-93.6438
185	then inorinos on the tanch	-93.6463
186	then inowinis on the beanch	-93.648
187	the monorinis on the ceanch	-93.6497
188	the monorinowhon the veanch	-93.65
189	then inowinis on the branch	-93.6503
190	the monleinos on the beanch	-93.6511
191	the onowinos on the veanch	-93.6526
192	the monleinos on the branch	-93.653
193	then inorinos on the ceanch	-93.6532
194	the monorinos on the ves ch	-93.6549
195	then inory is on the veanch	-93.6559
196	the momorinos on the veanch	-93.6581
197	the monowinis on the franct	-93.6612
198	the monorinod on the veanct	-93.6637
199	then inowinos on the franct	-93.6646
200	then onley is on the beanch	-93.6673
201	then onley is on the branch	-93.6692
202	the mo owinos on the beanch	-93.6707
203	the mo owinos on the branch	-93.6731
204	then onorunos on the veanch	-93.6749
205	the monley os on the veanch	-93.6794
206	the mo ley is on the veanch	-93.6822
207	then onowinod on the beanch	-93.6919
208	then onowinod on the branch	-93.694
209	then inorinis on the franch	-93.7001
210	the monory os on the beanct	-93.7046
211	the minowinis on the veanch	-93.706
212	the monory os on the branct	-93.7062
213	the monorinowhon the beanch	-93.7142
214	the monorinowhon the branch	-93.7159
215	the onowinos on the beanch	-93.7168
216	the mo orinos on the franch	-93.7174
217	then onory os on the franch	-93.7186
218	the onowinos on the branch	-93.7195
219	the monorinos on the bes ch	-93.7201
220	then inory is on the beanch	-93.721
221	then inory is on the branch	-93.7231
222	the momorinos on the beanch	-93.7234
223	the momorinos on the branch	-93.7256
224	the monorinod on the beanct	-93.7272
225	the monorinod on the branct	-93.7295
226	the monory is on the franct	-93.7326
227	then onleinos on the veanch	-93.7336
228	then onorunos on the beanch	-93.7383
229	then onorunos on the branch	-93.7401
230	then onorinod on the franch	-93.7429
231	the monley os on the beanch	-93.7438
232	the monley os on the branch	-93.7456
233	the mo ley is on the beanch	-93.7468
234	then onowinos on the cranct	-93.7481
235	the mo ley is on the branch	-93.7488
236	then o owinos on the veanch	-93.75
237	then onowinos on the tanct	-93.753
238	then onorinos on the ranct	-93.754
239	then onowinos on the ceanct	-93.7586
240	the minorinos on the cranch	-93.764

241	then inorinis on the veanct	-93.7651
242	the monley is on the franch	-93.7677
243	the onorinos on the franch	-93.7687
244	the minorinos on the tanch	-93.7697
245	then onorinos on the grantc	-93.7704
246	the minowinis on the beanch	-93.7717
247	the minowinis on the branch	-93.7732
248	the minorinos on the ceanch	-93.7755
249	the minory is on the veanch	-93.778
250	the mo orinos on the veanct	-93.7824
251	then onorinis on the cranch	-93.7832
252	the minowinos on the franct	-93.7837
253	then onory os on the veanct	-93.7842
254	the mo prinos on the veanch	-93.7862
255	then onorinis on the tanch	-93.7884
256	the monowinod on the franch	-93.7923
257	then onorinis on the ceanch	-93.7938
258	then onorinowhon the veanch	-93.7946
259	then onleinos on the beanch	-93.7955
260	then onleinos on the branch	-93.797
261	then onorinos on the ves ch	-93.799
262	then omorinos on the veanch	-93.8015
263	then onowinis on the franct	-93.8028
264	the monowinos on the ranct	-93.8045
265	then onorinod on the veanct	-93.8059
266	the mo leinos on the veanch	-93.8105
267	then o owinos on the beanch	-93.8122
268	then o owinos on the branch	-93.8137
269	then onley os on the veanch	-93.8191
270	the minorinis on the franch	-93.8201
271	the monowinos on the grantc	-93.8211
272	then o ley is on the veanch	-93.8226
273	then inorinis on the beanct	-93.827
274	then inorinis on the branct	-93.8287
275	the monley is on the veanct	-93.83
276	the onorinos on the veanct	-93.831
277	the monowinis on the cranch	-93.8336
278	then inowinos on the cranch	-93.8359
279	the monorunos on the franch	-93.8364
280	the monowinis on the tanch	-93.8386
281	the monorinis on the ranch	-93.8397
282	the minory is on the beanch	-93.8408
283	then inowinos on the tanch	-93.842
284	the minory is on the branch	-93.8426
285	then inorinos on the ranch	-93.843
286	the monowinis on the ceanch	-93.8449
287	the monowinowhon the veanch	-93.8451
288	the mo orinos on the beanct	-93.8456
289	then onory os on the beanct	-93.8465
290	the mo orinos on the branct	-93.8473
291	then inowinos on the ceanch	-93.8479
292	then onory os on the branct	-93.8483
293	the mo prinos on the beanch	-93.8491
294	the monowinos on the ves ch	-93.8505
295	the mo prinos on the branch	-93.8513
296	the mo mey is on the veanch	-93.8522
297	the momowinos on the veanch	-93.8534
298	the monowinod on the veanct	-93.8564
299	then onorinowhon the beanch	-93.8579
300	the monorinis on the granch	-93.8581

301	then onorinowhon the branch	-93.8597
302	then o orinos on the franch	-93.8611
303	then inorinos on the granch	-93.8614
304	then onorinos on the bes ch	-93.863
305	then omorinos on the beanch	-93.8654
306	the monorinos on the ctanch	-93.8663
307	then omorinos on the branch	-93.8672
308	then onorinod on the beanct	-93.8689
309	then onorinod on the branct	-93.8707
310	then onory is on the franct	-93.8734
311	the mo leinos on the beanch	-93.8743
312	then inory os on the veanch	-93.8757
313	the monldinos on the veanch	-93.8762
314	the mo leinos on the branch	-93.8766
315	the monorinos on the feanch	-93.8788
316	then onley os on the beanch	-93.8843
317	the minorinis on the veanct	-93.8849
318	then onley os on the branch	-93.8861
319	then o ley is on the beanch	-93.8878
320	then o ley is on the branch	-93.8894
321	then inowinis on the franch	-93.8939
322	the monley is on the beanct	-93.895
323	the onorinos on the beanct	-93.8959
324	the monleinos on the franch	-93.897
325	then inorinod on the veanch	-93.8974
326	the monley is on the branct	-93.8976
327	the onorinos on the branct	-93.8986
328	the mo ley os on the veanch	-93.9
329	the monorunos on the veanct	-93.9029
330	the monory is on the cranch	-93.9059
331	then onley is on the franch	-93.9092
332	the monowinowhon the beanch	-93.9102
333	the monory is on the tanch	-93.9109
334	the monowinowhon the branch	-93.9128
335	the mo owinos on the franch	-93.914
336	the monowinos on the bes ch	-93.9157
337	the mo mey is on the beanch	-93.9171
338	the monory is on the ceanch	-93.9179
339	the momowinos on the beanch	-93.9188
340	the mo mey is on the branch	-93.9197
341	the momowinos on the branch	-93.9209
342	the monowinod on the beanct	-93.9226
343	the monowinod on the branct	-93.9244
344	the inorinos on the veanch	-93.9261
345	then o orinos on the veanct	-93.9276
346	then o prinos on the veanch	-93.9304
347	then onowinod on the franch	-93.9361
348	the monorinow on the veanch	-93.9382
349	then inory os on the beanch	-93.9401
350	the monldinos on the beanch	-93.9404
351	then inory os on the branch	-93.9428
352	the monldinos on the branch	-93.9434
353	then onowinos on the ranct	-93.9476
354	the minorinis on the beanct	-93.9496
355	the monory os on the franct	-93.9501
356	the minorinis on the branct	-93.9516
357	then o leinos on the veanch	-93.9533
358	the minowinos on the cranch	-93.9584
359	then inowinis on the veanct	-93.959
360	the monorinowhon the franch	-93.9604

361	the monleinos on the veanct	-93.9614
362	then inorinod on the beanch	-93.9618
363	the onowinos on the franch	-93.9631
364	the mo ley os on the beanch	-93.9637
365	then inorinod on the branch	-93.9641
366	the minowinos on the tanch	-93.9648
367	then onowinos on the grancт	-93.9653
368	the minorinos on the ranch	-93.9657
369	the mo ley os on the branch	-93.9659
370	then inory is on the franch	-93.967
371	the monorunos on the beanct	-93.968
372	the momorinos on the franch	-93.9688
373	the monorunos on the branct	-93.9696
374	the minowinos on the ceanch	-93.9709
375	the monorinod on the franct	-93.9718
376	then onley is on the veanct	-93.9746
377	the mo owinos on the veanct	-93.9773
378	then onowinis on the cranch	-93.9777
379	then onorunos on the franch	-93.9803
380	the minorinos on the granch	-93.9814
381	the mo meinos on the veanch	-93.9817
382	then onowinis on the tanch	-93.9827
383	then onorinis on the ranch	-93.9838
384	the monley os on the franch	-93.9854
385	the momprinos on the veanch	-93.9873
386	the inorinos on the beanch	-93.9884
387	the mo ley is on the franch	-93.989
388	then onowinis on the ceanch	-93.9895
389	then onowinowhon the veanch	-93.9897
390	then o orinos on the beanct	-93.9902
391	the inorinos on the branch	-93.9913
392	then o orinos on the branct	-93.9925
393	then o prinos on the beanch	-93.9939
394	then onowinos on the ves ch	-93.9947
395	then o prinos on the branch	-93.9957
396	then o mey is on the veanch	-93.9966
397	the minory os on the veanch	-93.9974
398	then omowinos on the veanch	-93.9977
399	then onowinod on the veanct	-94.0005
400	then onorinis on the granch	-94.0021
401	the monorinow on the beanch	-94.0025
402	the monorinow on the branch	-94.0043
403	then onorinos on the ctanch	-94.0093
404	the monorinis on the cranct	-94.0122
405	the mo orinis on the veanch	-94.0139
406	the minowinis on the franch	-94.0145
407	then inorinos on the cranct	-94.0152
408	then o leinos on the beanch	-94.0165
409	the minorinod on the veanch	-94.0177
410	the monorinis on the tanct	-94.0183
411	then onldinos on the veanch	-94.0187
412	then o leinos on the branch	-94.0191
413	then onorinos on the feanch	-94.0206
414	then inorinos on the tanct	-94.0217
415	then inowinis on the beanct	-94.0234
416	the monorinis on the ceanct	-94.0247
417	the monorinowhon the veanct	-94.0249
418	then inowinis on the branct	-94.0252
419	the monleinos on the beanct	-94.0258
420	the onowinos on the veanct	-94.0273

421	the monleinos on the branct	-94.0275
422	then inorinos on the ceanct	-94.0277
423	the monorinos on the ves ct	-94.0293
424	the mo pey is on the veanch	-94.0299
425	then inory is on the veanct	-94.0303
426	the momorinos on the veanct	-94.0319
427	the monowinis on the ranch	-94.0355
428	then onley is on the beanct	-94.0376
429	then inowinos on the ranch	-94.0381
430	then onleinos on the franch	-94.0389
431	then onley is on the branct	-94.0393
432	then o ley os on the veanch	-94.0406
433	the mo owinos on the beanct	-94.0409
434	the mo owinos on the branct	-94.0425
435	then onorunos on the veanct	-94.0441
436	the mo meinos on the beanch	-94.0453
437	the mo meinos on the branch	-94.0466
438	then onory is on the cranch	-94.0471
439	the lonorinos on the veanch	-94.0476
440	the monley os on the veanct	-94.0486
441	the momprinos on the beanch	-94.05
442	the mo ley is on the veanct	-94.0514
443	the momprinos on the branch	-94.0517
444	then onowinowhon the beanch	-94.0522
445	the monowinis on the granch	-94.0524
446	then onory is on the tanch	-94.0532
447	then onowinowhon the branch	-94.0543
448	then o owinos on the franch	-94.055
449	then inowinos on the granch	-94.0555
450	then onowinos on the bes ch	-94.057
451	then o mey is on the beanch	-94.0583
452	then onory is on the ceanch	-94.059
453	the minory os on the beanch	-94.0593
454	then omowinos on the beanch	-94.0596
455	then o mey is on the branch	-94.0602
456	the monowinos on the ctanch	-94.0609
457	the minory os on the branch	-94.0617
458	then omowinos on the branch	-94.0623
459	then onowinod on the beanct	-94.064
460	the onorinis on the veanch	-94.0644
461	then onowinod on the branct	-94.0659
462	then inley is on the veanch	-94.0668
463	the monory pe on the veanch	-94.0696
464	the mo mey os on the veanch	-94.0714
465	the monowinos on the feanch	-94.0724
466	then inorinis on the franct	-94.0731
467	the mo orinis on the beanch	-94.0783
468	the minowinis on the veanct	-94.0788
469	then onorinow on the veanch	-94.0797
470	the mo orinis on the branch	-94.08
471	the minorinod on the beanch	-94.0811
472	then onldinos on the beanch	-94.0818
473	the minorinod on the branch	-94.0827
474	then onldinos on the branch	-94.0836
475	the monorinos on the fanch	-94.0848
476	the minory is on the franch	-94.0855
477	the monorinowhon the beanct	-94.0879
478	the monorinowhon the branct	-94.0894
479	the onowinos on the beanct	-94.0902
480	the mo orinos on the franct	-94.0904

481	then inowinod on the veanch	-94.0916
482	then onory os on the franct	-94.092
483	the onowinos on the branct	-94.0928
484	the monorinos on the bes ct	-94.0933
485	the mo pey is on the beanch	-94.0942
486	then inory is on the beanct	-94.0944
487	the mo prinos on the franch	-94.0948
488	the mo pey is on the branch	-94.0961
489	then inory is on the branct	-94.0962
490	the momorinos on the beanct	-94.0964
491	the mo ldinos on the veanch	-94.0979
492	the momorinos on the branct	-94.0982
493	then onorinowhon the franch	-94.1026
494	then onleinos on the veanct	-94.1033
495	then o ley os on the beanch	-94.105
496	the monory is on the ranch	-94.1061
497	then o ley os on the branch	-94.1065
498	then onorunos on the beanct	-94.1081
499	then omorinos on the franch	-94.1089
500	then onorunos on the branct	-94.1099
501	the lonorinos on the beanch	-94.1116
502	then onorinod on the franct	-94.1121
503	the monley os on the beanct	-94.1127
504	the lonorinos on the branch	-94.1136
505	the monley os on the branct	-94.1143
506	the mo ley is on the beanct	-94.1155
507	the mo leinos on the franch	-94.1167
508	the mo ley is on the branct	-94.1175
509	the inowinos on the veanch	-94.1177
510	then o owinos on the veanct	-94.1188
511	the monory os on the cranch	-94.1222
512	the monory is on the granch	-94.1225
513	then o meinos on the veanch	-94.1232
514	then onley os on the franch	-94.1259
515	the onorinis on the beanch	-94.1264
516	the monory os on the tanch	-94.1281
517	then omprinos on the veanch	-94.1283
518	the onorinis on the branch	-94.1285
519	then inley is on the beanch	-94.129
520	then o ley is on the franch	-94.1299
521	then inley is on the branch	-94.1306
522	the monowinow on the veanch	-94.1311
523	the monory pe on the beanch	-94.132
524	the monorunis on the veanch	-94.1328
525	the monory pe on the branch	-94.1335
526	the mo mey os on the beanch	-94.134
527	the monory os on the ceanch	-94.135
528	the minorinos on the cranct	-94.1354
529	the mo mey os on the branch	-94.1362
530	then inorunos on the veanch	-94.1365
531	the monley is on the franct	-94.1389
532	the onorinos on the franct	-94.1398
533	the minorinos on the tanct	-94.1409
534	the monosinos on the veanch	-94.1418
535	the minowinis on the beanct	-94.1426
536	then onorinow on the beanch	-94.1435
537	the minowinis on the branct	-94.1443
538	the monorinod on the cranch	-94.1453
539	then onorinow on the branch	-94.1465
540	the minorinos on the ceanct	-94.1478

541	the minory is on the veanct	-94.1499
542	the monorinod on the tanch	-94.1508
543	the monodinos on the veanch	-94.1529
544	the monowinowhon the franch	-94.1538
545	then onorinis on the cranct	-94.1544
546	then inowinod on the beanch	-94.1549
547	the monorinod on the ceanch	-94.1563
548	then o orinis on the veanch	-94.1566
549	then inowinod on the branch	-94.157
550	the mo prinos on the veanct	-94.1579
551	the minowinos on the ranch	-94.1585
552	the mo peinos on the veanch	-94.1589
553	then onorinis on the tanct	-94.1605
554	the mo mey is on the franch	-94.1609
555	the mo ldinos on the beanch	-94.1614
556	the momowinos on the franch	-94.162
557	the mo ldinos on the branch	-94.163
558	the monowinod on the franct	-94.1649
559	then onorinis on the ceanct	-94.1662
560	then onorinowhon the veanct	-94.1667
561	then onleinos on the beanct	-94.1674
562	then onleinos on the branct	-94.1691
563	then onorinos on the ves ct	-94.171
564	then o pey is on the veanch	-94.1716
565	then omorinos on the veanct	-94.1734
566	the minowinos on the granch	-94.1755
567	then onowinis on the ranch	-94.177
568	the mo leinos on the veanct	-94.1801
569	the inowinos on the beanch	-94.1806
570	then inory os on the franch	-94.1817
571	the monldinos on the franch	-94.1822
572	then o owinos on the beanct	-94.1824
573	the inowinos on the branch	-94.1831
574	then o owinos on the branct	-94.1842
575	the minley is on the veanch	-94.1848
576	then o meinos on the beanch	-94.187
577	then o meinos on the branch	-94.1884
578	the monkey is on the veanch	-94.1891
579	then onley os on the veanct	-94.1901
580	the monleinis on the veanch	-94.1906
581	the minorinis on the franct	-94.1908
582	then omprinos on the beanch	-94.1918
583	then o ley is on the veanct	-94.1932
584	then inleinos on the veanch	-94.1937
585	then omprinos on the branch	-94.194
586	then onowinis on the granch	-94.1944
587	the monowinow on the beanch	-94.1951
588	the monowinow on the branch	-94.1966
589	the monorunis on the beanch	-94.197
590	the monorunis on the branch	-94.1985
591	then inorunos on the beanch	-94.1995
592	then inorunos on the branch	-94.201
593	then onowinos on the ctanch	-94.2022
594	then inorinod on the franch	-94.2033
595	the monosinos on the beanch	-94.2044
596	the mo ley os on the franch	-94.2047
597	the monowinis on the cranct	-94.2056
598	the monosinos on the branch	-94.2066
599	the mo owinis on the veanch	-94.2076
600	then inowinos on the cranct	-94.2086

601	the monorunos on the franct	-94.209
602	then onory pe on the veanch	-94.2101
603	the minowinod on the veanch	-94.2107
604	the monowinis on the tanct	-94.2112
605	then o mey os on the veanch	-94.2122
606	the monorinis on the ranct	-94.2124
607	the minory is on the beanct	-94.2133
608	then onowinos on the feanch	-94.2133
609	then inowinos on the tanct	-94.2145
610	the minory is on the branct	-94.2151
611	then inorinos on the ranct	-94.2155
612	the monodinos on the beanch	-94.2169
613	the monowinis on the ceanct	-94.2181
614	the monowinowhon the veanct	-94.2183
615	the monodinos on the branch	-94.2193
616	then o orinis on the beanch	-94.2207
617	then inowinos on the ceanct	-94.221
618	the mo prinos on the beanct	-94.2219
619	then o orinis on the branch	-94.2226
620	the mo peinos on the beanch	-94.2229
621	the monowinos on the ves ct	-94.223
622	the mo prinos on the branct	-94.2238
623	the mo mey is on the veanct	-94.2245
624	the mo peinos on the branch	-94.2248
625	the momowinos on the veanct	-94.2255
626	then onorinos on the fanch	-94.2271
627	the mompey is on the veanch	-94.2297
628	then onorinowhon the beanct	-94.2302
629	the monorinis on the granct	-94.2304
630	the inorinos on the franch	-94.2316
631	then onorinowhon the branct	-94.2322
632	then o orinos on the franct	-94.2328
633	then inorinos on the granct	-94.2332
634	then onorinos on the bes ct	-94.2349
635	then o pey is on the beanch	-94.2355
636	then o prinos on the franch	-94.2359
637	then o pey is on the branch	-94.237
638	then omorinos on the beanct	-94.2373
639	the monorinos on the ctanct	-94.2382
640	then o ldinos on the veanch	-94.2386
641	then omorinos on the branct	-94.2389
642	the lonowinos on the veanch	-94.2417
643	the monorinow on the franch	-94.2437
644	the mo leinos on the beanct	-94.2444
645	then inorinis on the cranch	-94.2451
646	the mo pey os on the veanch	-94.2456
647	then inory os on the veanct	-94.2459
648	the monldinos on the veanct	-94.246
649	the mo leinos on the branct	-94.2464
650	then onory is on the ranch	-94.2473
651	the monorinos on the feanct	-94.2477
652	the monorinos on the geanch	-94.2486
653	the minley is on the beanch	-94.2487
654	the minley is on the branch	-94.2503
655	then inorinis on the tanch	-94.2507
656	the monkey is on the beanch	-94.253
657	then onley os on the beanct	-94.2538
658	the monleinis on the beanch	-94.2543
659	**monkey is on the branch**	**-94.2547**

Exercise 23.8

We assume that all the states have the same initial probability.

1. The probability that a first-order Markov chain generated sequence $v_{1:T}$ is in general given by

$$p(v_{1:T}) = p(v_1) \prod_{t=2}^{T} p(v_t|v_{t-1}).$$

   We computed the probability that the Markov chain `pnew` generated the sequence S using Matlab (see the code below). The resulting probability is $8.1781 \cdot 10^{-13}$.

2. Similarly the probability of the sequence S given the Markov chain `qnew` is $8.6147 \cdot 10^{-24}$.

   It really make sense that S has a higher probability under `pnew` compared with `qnew`. The transitions with high probability from `pnew` ($A \to C$, $C \to G$, $G \to T$, $T \to A$) appear more frequently in the sequence S then the high probability transitions from `qnew` ($T \to G$, $G \to C$, $C \to A$, $A \to T$).

3. Yes, we agree with the solution found as it supports our intuition about the problem. We expect the EM algorithm to uncover the original two Markov chains we used to generate data and assign the highest posterior probabilities to such a Markov chain that is close to the one originally used to generate a particular sequence. The result returned in `phgv` indicates posterior probability $p(h|v)$ that a sequence v was generated by a Markov chain h. As we checked with our code, all sequences generated by `pnew` have the probability $p(h|v) = 1.0$ for $h = 2$ and all sequences generated by `pnew` have the probability $p(h|v) = 1.0$ for $h = 1$. Note that the model assignment can be arbitrary as the EM algorithm is randomly initialized. When we examine the transition matrix for $h = 2$ we can see that is similar to `pnew`:

```
pnew =

  0.0250    0.0250    0.0250    0.9250
  0.9250    0.0250    0.0250    0.0250
  0.0250    0.9250    0.0250    0.0250
  0.0250    0.0250    0.9250    0.0250

estimated model for h = 2
  0.0477    0.0323    0.0108    0.9132
  0.8912    0.0216    0.0215    0.0342
  0.0345    0.9191    0.0323    0.0342
  0.0265    0.0270    0.9355    0.0184
```

   The EM successfully find Markov chain that is close the the one used for generating original sequences and assigned the highest posterior probability to it. The same is true for the $h = 1$, which is close to `qnew`:

```
qnew =

    0.0250    0.9250    0.0250    0.0250
    0.0250    0.0250    0.9250    0.0250
    0.0250    0.0250    0.0250    0.9250
    0.9250    0.0250    0.0250    0.0250

estimated model for h = 1
    0.0286    0.9160    0.0220    0.0243
    0.0286    0.0367    0.9256    0.0350
    0.0260    0.0184    0.0358    0.9245
    0.9169    0.0289    0.0165    0.0162
```

4. The hidden sequence $h^p_{1:16}$ has log probability -16.0462 and the hidden sequence $h^q_{1:16}$ has log probability -25.7757, therefore the $h^p_{1:16}$ should be prefered. Moreover, it makes intuitively sense. As we know from the part 1. and 2. `pnew` has higher probability that it generated S than `qnew`. In this case, the emission distribution is just adding noise, because the observed symbols that corresponds to the hidden ones are still generated with the highest probability.

Code:

```
function ex238

run BRMLtoolkit/setup.m ;
import brml.* ;

[A, C, G, T] = assign(1:4) ;
symbols = 'ACGT' ;

% p(v(t)|v(t-1))
p = zeros(4, 4) ;
p(C,A) = 1 ;
p(G,C) = 1 ;
p(T,G) = 1 ;
p(A,T) = 1 ;

pnew = 0.9*p + 0.1*ones(4)/4 ;

q = zeros(4, 4) ;
q(G,T) = 1 ;
q(C,G) = 1 ;
q(A,C) = 1 ;
q(T,A) = 1 ;

qnew = 0.9*q + 0.1*ones(4)/4 ;

S = [A, A, G, T, A, C, T, T, A, C, C, T, A, C, G, C] ;

disp('%------------------------------------------------------------------------')
disp('% part 1')
disp('%------------------------------------------------------------------------')
pnewS = pmarkov(pnew, S) ;
disp('probability of the sequence S given pnew') ;
disp(pnewS) ;

disp('%------------------------------------------------------------------------')
disp(' part 2')
disp('%------------------------------------------------------------------------')
qnewS = pmarkov(qnew, S) ;
disp('probability of the sequence S given qnew') ;
disp(qnewS) ;

disp('%------------------------------------------------------------------------')
disp('% part 3')
disp('%------------------------------------------------------------------------')
v = cell(1,200) ;
for i=1:100
  v{i} = gens(pnew, 16) ;
end
```

```
for i=101:200
  v{i} = gens(qnew, 16) ;
end

opts.maxit=10; opts.plotprogress=1;
[ph,pv1gh,pvgvh,loglikelihood,phgv] = mixMarkov(v,4,2,opts) ;

classes = zeros(1,200) ;
for i=1:numel(phgv)
  [~, classes(i)] = max(phgv{i}) ;
end
if all(classes(1:100) == 1)
  disp('all instances generated by pnew belong to h = 1') ;
  pnew
  disp('estimated model for h = 1')
  disp(pvgvh(:,:,1))
end
if all(classes(1:100) == 2)
  disp('all instances generated by pnew belong to h = 2') ;
  pnew
  disp('estimated model for h = 2')
  disp(pvgvh(:,:,2))
end
if all(classes(101:200) == 1)
  disp('all instances generated by qnew belong to h = 1') ;
  qnew
  disp('estimated model for h = 1')
  disp(pvgvh(:,:,1))
end
if all(classes(101:200) == 2)
  disp('all instances generated by qnew belong to h = 2') ;
  qnew
  disp('estimated model for h = 2')
  disp(pvgvh(:,:,2))
end

disp('%
  ------------------------------------------------------------------------------'
  )
disp('% part 4')
disp('%
  ------------------------------------------------------------------------------'
  )
disp('observed sequence');
disp(converts(S, symbols));

% emission distribution
pe = eye(4,4) * 0.6 + 0.1 ;
ph1 = condp(ones(4,1));
[viterbimaxstate logprob] = HMMviterbi(S,pnew,ph1,pe) ;
disp('hidden sequence given pnew') ;
disp(converts(viterbimaxstate, symbols)) ;
logprob
[viterbimaxstate logprob] = HMMviterbi(S,qnew,ph1,pe) ;
disp('hidden sequence given pnew') ;
disp(converts(viterbimaxstate, symbols)) ;
logprob


% ------------------------------------------------------------------------------
function pS = pmarkov(p, s)
% compute probability of a sequence s given the transition matrix p
```

```
n = size(p, 1) ;
% initial state probability
pS = log(1/n) ;
for i=2:numel(s)
  pS = pS + log(p(s(i), (s(i-1)))) ;
end
pS = exp(pS) ;


function s = gens(p, l)
% generate random sequence given the transition matrix p
import brml.*
n = size(p, 1) ;
s = zeros(1, l) ;
s(1) = randgen(ones(1,n)/n) ;
for j=2:l
  s(j) = randgen(p(:,s(j-1))) ;
end


function c = converts(s, symbols)
% convert numbers to symbols
c = char(zeros(size(s))) ;
for i=1:numel(s)
  c(i) = symbols(s(i)) ;
end
```

Output:

```
% -------------------------------------------------------------------------------
% part 1
% -------------------------------------------------------------------------------
probability of the sequence S given pnew
   8.1781e-13

% -------------------------------------------------------------------------------
 part 2
% -------------------------------------------------------------------------------
probability of the sequence S given qnew
   8.6147e-24

% -------------------------------------------------------------------------------
% part 3
% -------------------------------------------------------------------------------
all instances generated by pnew belong to h = 2

pnew =

    0.0250    0.0250    0.0250    0.9250
    0.9250    0.0250    0.0250    0.0250
    0.0250    0.9250    0.0250    0.0250
    0.0250    0.0250    0.9250    0.0250

estimated model for h = 2
    0.0212    0.0139    0.0185    0.9193
    0.9443    0.0083    0.0159    0.0260
    0.0159    0.9612    0.0265    0.0234
    0.0186    0.0166    0.9392    0.0312

all instances generated by qnew belong to h = 1

qnew =
```

```
    0.0250    0.9250    0.0250    0.0250
    0.0250    0.0250    0.9250    0.0250
    0.0250    0.0250    0.0250    0.9250
    0.9250    0.0250    0.0250    0.0250

estimated model for h = 1
    0.0193    0.9198    0.0266    0.0233
    0.0165    0.0294    0.9122    0.0284
    0.0220    0.0241    0.0319    0.9147
    0.9421    0.0267    0.0293    0.0336

% ---------------------------------------------------------------------------
% part 4
% ---------------------------------------------------------------------------
observed sequence
AAGTACTTACCTACGC
hidden sequence given pnew
ACGTACGTACGTACGT

logprob =

  -16.0462

hidden sequence given pnew
ATGCATGCATGCATGC

logprob =

  -25.7757
```

Exercise 23.15

1.

$$\begin{aligned}
\alpha(h_t, c_t) &= p(h_t, c_t, v_{1:t}) \\
&= \sum_{h_{t-1}, c_{t-1}} p(v_t|v_{1:t-1}, h_t, c_t, c_{t-1}) p(h_t|v_{1:t-1}, h_{t-1}, c_t, c_{t-1}) p(c_t|v_{1:t-1}, h_{t-1}, c_{t-1}) p(v_{1:t-1}, h_{t-1}, c_{t-1}) \\
&= \sum_{h_{t-1}, c_{t-1}} p(v_t|h_t) p(h_t|h_{t-1}, c_t) p(c_t, c_{t-1}) p(v_{1:t-1}, h_{t-1}, c_{t-1}) \\
&= p(v_t|h_t) \sum_{h_{t-1}, c_{t-1}} p(h_t|h_{t-1}, c_t) p(c_t|c_{t-1}) \alpha_{t-1}(h_{t-1}, c_{t-1})
\end{aligned}$$

2.

$$\begin{aligned}
\frac{\alpha(h_t, c_t)}{p(v_t|h_t)} &= \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t) \sum_{c_{t-1}} p(c_t|c_{t-1}) \alpha_{t-1}(h_{t-1}, c_{t-1}) \\
&= \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t) p(c_t|c_{t-1} = 1) \alpha_{t-1}(h_{t-1}, c_{t-1} = 1) + \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t) p(c_t|c_{t-1} > 1) \alpha_{t-1}(h_{t-1}, c_{t-1} > 1) \\
&= \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t) p(c_t|c_{t-1} = 1) \alpha_{t-1}(h_{t-1}, c_{t-1} = 1) + \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t) \sum_{c_{t-1}=2}^{D_{max}} p(c_t|c_{t-1}) \alpha_{t-1}(h_{t-1}, c_{t-1})
\end{aligned}$$

- the latent variable counter c_t is sampled from $p_{dur}(c_t)$ with maximum duration D_{max} and decreases by 1 per time step, until $c_{t-1} = 1$, which is when re-sampling happens again.

3.

$$\frac{\alpha(h_t, c_t)}{p(v_t|h_t)} = \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t) p(c_t|c_{t-1} = 1) \alpha_{t-1}(h_{t-1}, c_{t-1} = 1) + \mathbb{I}[2 \le c_t \le D_{max}] \sum_{h_{t-1}} p(h_t|h_{t-1,c_t}) \alpha_{t-1}(h_{t-1}, c_{t-1})$$

- since state h_t wont change for the whole duration where $2 \le c_t \le D_{max}$, then $p(h_t|h_{t-1}, c_t)$ is the same for all duration.

- $\mathbb{I}[2 \leq c_t \leq D_{max}] = \mathbb{I}[c > 1] = \mathbb{I}[c \neq D_{max}]$
- $c_{t-1} \neq 1$ for right half of the RHS, thus for any given $c_t = c$, the right half's $c_{t-1} = c + 1$, therefore

$$\frac{\alpha(h_t, c_t)}{p(v_t|h_t)} = \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t = c)p(c_t = c|c_{t-1} = 1)\alpha_{t-1}(h_{t-1}, c_{t-1} = 1) + \mathbb{I}[c \neq D_{max}] \sum_{h_{t-1}} p(h_t|h_{t-1}, c)\alpha_{t-1}(h_{t-1}, c+1)$$

4. for $c_t = 1, p(h_t|h_{t-1}, c_t) = p_{tran}(h_t|h_{t-1})$

$$\begin{aligned}
\alpha_t(h_t, 1) &= p(v_t|h_t) \sum_{h_{t-1}} p(h_t|h_{t-1}, c_t = 1)p(c_t = 1|c_{t-1} = 1)\alpha_{t-1}(h_{t-1}, c_{t-1} = 1) \\
&+ \mathbb{I}[1 \neq D_{max}] \sum_{h_{t-1}} p(h_t|h_{t-1}, 1)\alpha_{t-1}(h_{t-1}, 1 + 1) \\
&= p(v_t|h_t) \sum_{h_{t-1}} p_{tran}(h_t|h_{t-1})p_{dur}(1)\alpha_{t-1}(h_{t-1}, c_{t-1} = 1) \\
&+ \mathbb{I}[1 \neq D_{max}] \sum_{h_{t-1}} p_{tran}(h_t|h_{t-1})\alpha_{t-1}(h_{t-1}, 2)
\end{aligned}$$

and for $h_t = h$, we arrive at the proof. For $c_t > 1, p(h_t|h_{t-1}, c_t) = \delta(h_t, h_{t-1})$, i.e. the hidden state h_t doesn't change.

$$\alpha(h_t, c) = p(v_t|h_t = h)\{p_{dur}(c)\alpha_{t-1}(h_{t-1}, c) + \mathbb{I}[c \neq D_{max}]\alpha_{t-1}(h, c + 1)\}$$

5. Complexity of filtered inference in duration model is $O(TH^2D_{max})$

- dimension of $c_t \bigotimes h_t$ is $D_{max}H$
- naively computational complexity is $O(TH^2D_{max}^2)$
- if, however, this is computed using the Forward-Backward recursion, then we only have to go through c_t once, hence complexity is reduced to $O(TH^2D_{max})$ - see below.

6.

$$\begin{aligned}
p(h_t, c_t, v_{1:T}) &= p(h_t, c_t, v_{1:t}, v_{t+1:T}) \\
&= p(h_t, c_t, v_{1:t})p(v_{t+1:T}|h_t, c_t, v_{1:t}) \\
&= \alpha(h_t, c_t)\beta(h_t, c_t)
\end{aligned}$$

Beta recursion:

$$\beta(h_{t-1}, c_{t-1}) = \sum_{h_t, c_t} p(v_t|h_t)p(h_t|h_{t-1}, c_t)p(c_t|c_{t-1})\beta(h_t, c_t)$$

Hence,

$$\beta(h_t, c_t) = \frac{\beta(h_{t-1}, c_{t-1})}{\sum_{h_t, c_t} p(v_t|h_t)p(h_t|h_{t-1}, c_t)p(c_t|c_{t-1})},$$

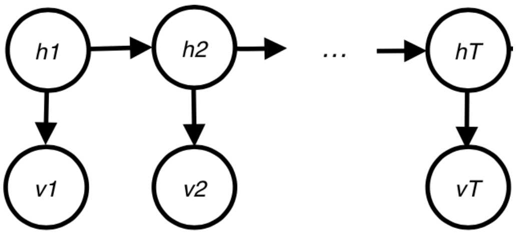
so the smoothed posterior is

$$p(h_t, c_t|v_{1:T}) = \gamma(h_t, ct) = \frac{\alpha(h_t, c_t)\beta(h_t, c_t)}{\sum_{h_t, c_t} \alpha(h_t, c_t)\beta(h_t, c_t)},$$

where $\alpha(h_t, c_t)$ is given in part 4.

Exercise 23.16

1. The HMM with for the problem ($L+1$ states) can be visualised as:



Filtering: For this Hidden Markov Model, we want to obtain $p(h_t|v_{1:t})$. To do so, we must first compute the joint marginal $p(h_t, v_{1:t})$ from which the conditional marginal $p(h_t|v_{1:t})$ can be obtained by normalisation. A recursion for $p(h_t, v_{1:t})$ is obtained by considering:

$$p(h_t, v_{1:t}) = \sum_{h_{t-1}} p(h_t, h_{t-1}, v_{1:t-1}, v_t) = \sum_{h_{t-1}} p(v_t|h_t)p(h_t|h_{t-1})p(h_{t-1}, v_{1:t-1}),$$

from the CI assumptions of the model. Hence if we define, $\alpha(h_t) = p(h_t, v_{1:t})$, the equation above gives the $\alpha - recursion$:

$$\alpha(h_t) = p(v_t|h_t) \sum_{h_{t-1}} p(h_t|h_{t-1})\alpha(h_{t-1}), \quad t > 1$$

where $p(v_t|h_t)$ is a corrector term and $\sum_{h_{t-1}} p(h_t|h_{t-1})\alpha(h_{t-1})$ is a predictor term, with $\alpha(h_1) = p(h_1, v_1) = p(v_1|h_1)p(h_1)$. This recursion propagates forward $\alpha(h_{t-1})$ for each timestep giving a new 'prior' at t, which is then modulated by observation v_t. We must, therefore, compute an $\alpha(h_t)$ for each t, $t > 1$. We have then $T-1$ 'instances' (for a text of T characters). Each $\alpha(h_t)$ is added across each latent state and there are $L+1$ of these. Hence, filtering can be computed with $O((T-1)(L+1)) \approx O(TL)$ operations using α-recursion.

Smoothing: We want to compute $p(h_t|v_{1:u})$ for $t < u$. In similar fashion as for filtering, we must first compute $p(h_t, v_{1:u})$. In the parallel method (analogous to message passing on factor graphs), a smoothed posterior is partitioned into past and future contributions:

$$p(h_t, v_{1:u}) = p(h_t, v_{1:t}, v_{t+1:u}) = p(h_1, v_{1:t})p(v_{t+1:u}|h_t, v_{1:t}) = \alpha(h_t)\beta(h_t),$$

where $p(h_t, v_{1:t})$ is term for the 'past' and $p(v_{t+1:u}|h_t, v_{1:t})$ for the 'future' (where $v_{1:t}$ can be removed - the past from the future is d-separated by h_t). Another recursion can be obtained:

$$p(v_{t:u}|h_{t-1}) = \sum_{h_t} p(v_t, v_{t+1:u}, h_t|h_{t-1}) = \sum_{h_t} p(v_t|h_t)p(v_{t+1:u}|h_t)p(h_t|h_{t-1}).$$

Defining $\beta(h_t) = p(v_{t+1:u}|h_t)$, the equation above gives the $\beta - recursion$ ("backward" versus the "forward" $\alpha - recursion$):

$$\beta(h_{t-1}) = \sum_{h_t} p(v_t|h_t)p(h_t|h_{t-1})\beta(h_t), \quad 2 \le t \le T$$

for T instances with $\beta(h_T) = 1$. We must, therefore, compute a $\beta(h_{t-1})$ for each t, $2 \le t \le T$. We have $T-1$ instances. Each $\beta(h_{t-1})$ is added across each latent state and there are $L+1$ of these. Additionally, since the α and β recursions are independent and can thus run in parallel, smoothing can be computed with $O(T-1)(L+1)) \approx O(TL)$ operations using α-recursion and β-recursion.

Viterbi: Recall that the Viterbi algorithm is a special case of the N-max-product algorithm. In similar fashion as for filtering and smoothing, a $\mu - recursion$ can be defined for the Viterbi algortithm:

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

with $\mu(h_T) = 1$. As in the max-product algorithm, one obtains a most likely state h_1^* and once computed, h_t^* can be obtained through backtracking. For $2 \le t \le T$, one must store the latent variable that maximises $\mu(h_{t-1})$ for each instance. Hence, we compute μ-recursion $T-1$ times. In addition, for each recursion computation, one must maximise across $L+1$ latent states. Hence, the Viterbi algorithm can be computed with $O((T-1)(L+1)) \approx O(TL)$ operations using μ-recursion.

2. The code developed to calculate the smoothed probability p_t is presented below.

```
import brml.*

% target pattern
pat = [3 4 2 4 1 3 2]; % reverse - start at T
pat_len = length(pat); % pattern length
% text
numbers = [1 2 1 3 2 3 2 4 2 3 3 1 4 4 2 4 1 3 1 4 2 2 3 1 4 2 3 3 1 2 3 4];
text_len = length(numbers) % text length

% tau parameter
t = 0.5;
% emmission probability
e_prob = 0.9;
dom_size = 4; % s domain size

% transition matrix
phghm(1,1) = t;
phghm(pat_len+1,1) = 1-t;
for i = 1:pat_len
    phghm(i,i+1) = 1;
    ci(i) = i; % column index
end

% initially
initial = phghm(1,:)';

% emmission
pvgh = zeros(dom_size, pat_len);
pvgh(sub2ind(size(pvgh),pat,ci)) = e_prob - (1/30);
% uniform prob over 4
pvgh = [(repmat((1/dom_size),dom_size, 1)) pvgh+(1/30)];

% inference (forward and backward)
[alph, loglik] = HMMforward(numbers, phghm,initial, pvgh);
bet = HMMbackward(numbers, phghm,pvgh);
[phtgV1T, phthtpgV1T] = HMMsmooth(alph,bet, pvgh, phghm,numbers); % smoothing
% returns smooth posterior and pair

smoothed(2,:) = phtgV1T(pat_len+1,:); % insert smoothed posterior_prob
smoothed(1,:) = 1:text_len;
```

The variable `smoothed` contains the smoothed probabilities for each position t.

```
>> demo_stringmatcher

smoothed =

  Columns 1 through 10

    1.0000    2.0000    3.0000    4.0000    5.0000    6.0000    7.0000    8.0000    9.0000   10.0000
    0.0000    0.0000    0.0038    0.0000    0.9916    0.0000    0.0000    0.0000    0.0014    0.0013

  Columns 11 through 20

   11.0000   12.0000   13.0000   14.0000   15.0000   16.0000   17.0000   18.0000   19.0000   20.0000
    0.0037    0.0000    0.0024    0.0001    0.0001    0.0000    0.0543    0.0000    0.0000    0.0000

  Columns 21 through 30

   21.0000   22.0000   23.0000   24.0000   25.0000   26.0000   27.0000   28.0000   29.0000   30.0000
    0.0000    0.9456    0.0000    0.0000    0.0000    0.0041    0.0152    0.0000    0.0003    0.8386

  Columns 31 through 32

   31.0000   32.0000
    0.0043    0.0162
```

Exercise 23.17

Standard recursion:

$$\alpha(h_t) = p(v_t|h_t) \sum_{h_{t-1}} p(h_t|h_{t-1})\alpha(h_{t-1})$$

Normalized recursion:

$$\hat{\alpha}(h_t) = \frac{1}{z_t} p(v_t|h_t)p(h_t|h_{t-1})\hat{\alpha}(h_{t-1}),$$

where

$$z_t = \sum_{h_t} p(v_t|h_t)p(h_t|h_{t-1})\hat{\alpha}(h+t-1)$$

Show that:

$$\alpha(h_t) = \hat{\alpha}(h_t) \prod_{\tau=1}^{t} z_\tau$$

Right hand side:

$$\hat{\alpha}(h_t) \prod_{\tau=1}^{t} z_\tau = \prod_{\tau=1}^{t} z_\tau \frac{1}{z_t} p(v_t|h_t)p(h_t|h_{t-1})\hat{\alpha}(h_{t-1})$$

Expanding the last term recursively:

$$\begin{aligned}
\hat{\alpha}(h_{t-1}) &= \frac{1}{z_{t-1}} p(v_{t-1}|h_{t-1})p(h_{t-1}|h_{t-2})\hat{\alpha}(h_{t-2}) \\
\hat{\alpha}(h_{t-2}) &= \frac{1}{z_{t-2}} p(v_{t-2}|h_{t-2})p(h_{t-2}|h_{t-3})\hat{\alpha}(h_{t-3}) \\
&\vdots \\
\hat{\alpha}(h_2) &= \frac{1}{z_2} p(v_2|h_2)p(h_2|h_1)\hat{\alpha}(h_1) \\
\hat{\alpha}(h_1) &= \frac{1}{z_1} p(v_1|h_1)p(h_1)
\end{aligned}$$

incurring all the normalization terms:

$$\frac{1}{z_t}\frac{1}{z_{t-1}}\frac{1}{z_{t-2}}\cdots\frac{1}{z_2}\frac{1}{z_1} = \frac{1}{\prod_{\tau=1}^{t} z_\tau},$$

which is then canceled with the product term.

Now look at the Left hand side:

$$\alpha(h_t) = p(v_t|h_t) \sum_{h_{t-1}} p(h_t|h_{t-1})\alpha(h_{t-1})$$

Again expanding the last term,

$$\begin{aligned}
\alpha(h_{t-1}) &= p(v_{t-1}|h_{t-1}) \sum_{h_{t-2}} p(h_{t-1}|h_{t-2})\alpha(h_{t-2}) \\
\alpha(h_{t-2}) &= p(v_{t-2}|h_{t-2}) \sum_{h_{t-3}} p(h_{t-2}|h_{t-3})\alpha(h_{t-3}) \\
&\vdots \\
\alpha(h_2) &= p(v_2|h_2) \sum_{h_2} p(h_2|h_1)\alpha(h_1) \\
\alpha(h_1) &= p(v_1|h_1)p(h_1).
\end{aligned}$$

Thus by message passing and getting rid of the summation term, we can see that the LHS == RHS and that therefore, $\log p(v_{1:T}) = \log \prod_{\tau=1}^{T} z_\tau = \sum_{\tau=1}^{T} \log(z_\tau)$.

References

Questions 10.5 and 23.16 make reference to material discussed in:

- Barber, D. Bayesian Reasoning and Machine Learning, Cambridge University Press, 2012.