Assignment 3

Per Emil Hammarlund, Albert Öst

2019-05-05

Contents

Implementation of forward pass and log prob

Forward pass

The forward pass was implemented using the following code:

```
function [alphaHat, c]=forward(mc,pX)
%---------------------------------------------------------
%Code Authors:
% Albert Öst
% Per Emil Hammarlund
%---------------------------------------------------------

% Get the number of observations
T = size(pX, 2);

% Get the number of hidden states in the markov chain
ns = nStates(mc);

% That means that the alhpas will be a T by ns matrix of probabilities
alphaHat = zeros(ns, T);

%-------------------- continue code from here, and delete error message

% Check if the markov chain is finite or not
isFinite = finiteDuration(mc);
if isFinite
    transProbs = mc.TransitionProb(:, 1: end - 1);
    c = zeros(T + 1, 1);
else
    transProbs = mc.TransitionProb;
    c = zeros(T, 1);
end

alphaHat(:,1) = mc.InitialProb .* pX(:,1);
c(1) = sum(alphaHat(:,1));

alphaHat(:,1) = alphaHat(:,1) ./ c(1);

for t=2:T
    alphaHat(:,t) = pX(:,t) .* (alphaHat(:,t - 1)' * transProbs)';
    c(t) = sum(alphaHat(:,t));
    alphaHat(:,t) = alphaHat(:,t) ./ c(t);
end

if isFinite
    c(T + 1) = sum(alphaHat(:,T) .* mc.TransitionProb(:,end));
end


end
```

Implementation of log prob

The log prob was implemented using the following code:

```
%-----------------------------------------------------
%Code Authors:
% Albert öst
% Per Emil Hammarlund
%-----------------------------------------------------

function logP=logprob(hmm,x)
hmmSize=size(hmm);%size of hmm array
T=size(x,2);%number of vector samples in observed sequence
logP=zeros(hmmSize);%space for result
for i=1:numel(hmm)%for all HMM objects
    %Note: array elements can always be accessed as hmm(i),
    %regardless of hmmSize, even with multi-dimensional array.
    %
    %logP(i)= result for hmm(i)
    %continue coding from here, and delete the error message.
    [pX, logS] = prob(hmm(i).OutputDistr, x);

    % Scale the probs
    pX = (ones(size(pX, 1), 1) * exp(logS)) .* pX;

    [~, c] = hmm(i).StateGen.forward(pX);


    logP(i) = sum(log(c));

end;
```

Verification of forward pass

To verify the forward pass, the model:

$$q = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$
$$A = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.9 & 0.1 \end{pmatrix}$$

With the state conditional output:

$$g_1 = \mathcal{N}(0, 1)$$
$$g_2 = \mathcal{N}(3, 2)$$

Was constructed using the following code:

```
mc = MarkovChain([1; 0], [0.9 0.1 0; 0 0.9 0.1]);
g1 = GaussD('Mean', 0, 'StDev', 1);
g2 = GaussD('Mean', 3, 'StDev', 2);
```

And pX was calculated by:

```
pX = prob([g1 g2], x);
```

Now that the required parameters and model where complete, the function:

```
[alphaHat, c] = mc.forward(pX)
```

Could now be tested, which gave the following output:

```
alphaHat =

   1.000000000000000   0.384704237490574   0.418874656659074
                   0   0.615295762509426   0.581125343340926

c =

          1.000000000000000
          0.162523466100529
          0.826580955035720
          0.058112534334093
```

Which is the same values as what was desired in the lab instruction.

Validation of log prob

The same observations and model where used once again, but this time a HMM was also constructed:

```
h = HMM(mc, [g1 g2]);
```

The **logprob** function was then tested with:

```
logP = h.logprob(x)
```

Which gave the following output:

```
logP =

  -9.187726979475208
```

Which was the same value as what was desired in the lab instruction.