Assignment 4

Yann Debain debain@kth.se

Harsha Holenarasipura Narasanna harshahn@kth.se

May 19, 2019

1 Introduction

The goal of this assignment is to implement and verify the backward algorithm to train the HMM in Matlab.

The code is enclosed into the package **PattRecClasses-2**, you will find in this report the different tests implemented to verify that the code is correct (the backward algorithm is in the Annexe section).

2 Verify the backward algorithm

2.1 Finite HMM

For the given test example, the parameters of the HMM source are :

$$q = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \quad A = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.9 & 0.1 \end{pmatrix}; \quad B = \begin{pmatrix} b_1(x) = \mathcal{N}(0,1) \\ b_2(x) = \mathcal{N}(3,2) \end{pmatrix}$$

We assume that the observed feature sequence is $x = [-0.2, 2.6, 1.3]$ and $c = \begin{pmatrix} 1 & 0.1625 & 0.8266 & 0.0581 \end{pmatrix}$.

The results of the backward algorithm for this feature sequence is given in the paper. We have implemented the backward algorithm by following the different steps in the course and obtained the following outputs :

$$\hat{\beta} = \begin{pmatrix} 1.0003 & 1.0393 & 0 \\ 8.4182 & 9.3536 & 2.0822 \end{pmatrix}$$

The outputs are consistent with the results in the paper, thus we can assume the backward algorithm for a finite HMM is well implemented.

2.2 Infinite HMM

For the infinite-duration HMM, we remove the end state in the transition matrix and we will consider the model from the exercise 5.1. The parameters of the HMM source become :

$$q = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}; \quad A = \begin{pmatrix} 0.3 & 0.7 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{pmatrix}; \quad B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0.4 & 0.1 \\ 0.1 & 0.1 & 0.2 & 0.6 \end{pmatrix}$$

We assume that the observed feature sequence is $x = [1, 2, 4, 4, 1]$.

Thanks to the forward algorithm we have c that allows us to use the backward algorithm (after verified that the forward algorithm has been well implemented) : $c = \begin{pmatrix} 1 & 0.35 & 0.35 & \frac{79}{140} & \frac{157}{1580} \end{pmatrix}$

Results computed by hand

$$\hat{\beta} = \begin{pmatrix} 1 & \frac{52}{157} & \frac{28}{157} & \frac{840}{157} & \frac{1580}{157} \\ \frac{1073}{1099} & \frac{20}{7} & \frac{260}{157} & \frac{140}{157} & \frac{1580}{157} \\ \frac{576}{1099} & \frac{5760}{1099} & \frac{480}{157} & \frac{280}{157} & \frac{1580}{157} \end{pmatrix}$$

Results with Matlab

$$\hat{\beta} = \begin{pmatrix} 1 & 0.3312 & 0.1783 & 5.3503 & 10.0637 \\ 0.9763 & 2.8571 & 1.6561 & 0.8917 & 10.0637 \\ 0.5241 & 5.2411 & 3.0573 & 1.7834 & 10.0637 \end{pmatrix}$$

We can verify :

— $\frac{1073}{1099} \simeq 0.9763$; $\frac{576}{1099} \simeq 0.5241$
— $\frac{52}{157} \simeq 0.3312$; $\frac{20}{7} \simeq 2.8571$; $\frac{5760}{1099} \simeq 5.2411$
— $\frac{28}{157} \simeq 0.1783$; $\frac{260}{157} \simeq 1.6561$; $\frac{480}{157} \simeq 3.0573$
— $\frac{840}{157} \simeq 5.3503$; $\frac{140}{157} \simeq 0.8917$; $\frac{280}{157} \simeq 1.7834$
— $\frac{1580}{157} \simeq 10.0637$

The outputs are consistent with the results computed by hand, thus we can assume the backward algorithm for an infinite HMM is well implemented too.

3 Conclusion

As a conclusion, we have seen how to implement the backward algorithm to train the HMM.

For a given set of parameters, we have verified if the codes have been well implemented for the different case of HMM (finite/infinite duration).

4 Annexes - Code

backward algorithm

```
function betaHat=backward(mc,pX,c)
%betaHat=backward(mc,pX,c)
%calculates scaled observation probabilities,
%using the backward algorithm, for a given single MarkovChain,
%to be used when the MarkovChain is part of a HMM object.
%
%Input:
%mc= MarkovChain object
%pX= matrix with state-conditional likelihood values,
%   without considering the Markov depencence between sequence samples.
%       pX(j,t)= P( X(t)= observed x(t) | S(t)= j )
%       (must be pre-calculated externally)
%   NOTE: pX may be arbitrarily scaled, as defined externally,
%   i.e. it may not be a properly normalized probability density or mass.
%
%c=row vector with observation probabilities, obtained from Forward method:
%       c(t)=P[x(t) | x(1)...x(t-1),HMM]; t=1..T
%       c(1)*c(2)*..c(t)=P[x(1)..x(t)| HMM]
%   If the HMM has Finite Duration, the last element includes
%   the probability that the HMM ended at exactly the given sequence length, i.e.
%   c(T+1)=P( S(T+1)=N+1| x(1)...x(T-1), x(T)  )
%Thus, for an infinite-duration HMM:
%   length(c)=T
%   prod(c)=P( x(1)..x(T) )
%and, for a finite-duration HMM:
%   length(c)=T+1
%   prod(c)=P( x(1)..x(T), S(T+1)=END )
%
%Result:
%betaHat=matrix with scaled backward probabilities:
%       betaHat(j,t)=beta(j,t)/(c(t)*..c(T)), where
%       beta(j,t)=P[x(t+1)...x(T)| S(t)=j] for an infinite-duration HMM without END state
%       beta(j,t)=P[x(t+1)...x(T),S(T+1)=END| S(t)=j] for a finite-duration HMM
%
%NOTE:
%For an infinite-duration HMM:
%   P[S(t)=j|x(1)..x(T)]= alfaHat(j,t)*betaHat(j,t)*c(t)
%For a finite-duration HMM with separate END state:
%   P[S(t)=j|x(1)..x(T), S(T+1)=END]= alfaHat(j,t)*betaHat(j,t)*c(t)
%
%Ref: Arne Leijon (200x) Pattern Recognition.
%
%Arne Leijon 2007-08-16 tested (changed definition of scale factors)
%           2009-09-29 tested (changed test for finite duration)

T = size(pX,2); %Number of observations
```

```
N = nStates(mc);

A  = mc.TransitionProb;

betaHat = zeros(N, T);

isFinite = (N ~= size(A, 2));

% First iteration
if isFinite
    betaHat(:, T) = A(:, N+1)./(c(T)*c(T+1));
else
        betaHat(:, T) = ones(N,1)./c(T);
end

% Loop
for t = T-1:-1:1
    betaHat(:, t) = A(:, 1:N)*(pX(:, t+1).*betaHat(:, t+1))./c(t);
end


end
```

Tests

```
close all;
clear all;

addpath('PattRecClasses-2')

%% Test forward - Finite MC

q = [1; 0];
A = [0.9, 0.1, 0; 0, 0.9, 0.1];

b1 = GaussD('Mean', 0, 'StDev', 1);
b2 = GaussD('Mean', 3, 'StDev', 2);

B = [b1; b2];

mc = MarkovChain(q, A);

x = [-0.2, 2.6, 1.3];
pX = prob(B, x);

c = [1, 0.1625, 0.8266, 0.0581];

betaHat_f = backward(mc, pX, c);

%% Test forward - Infinite MC

q = [1; 0; 0];
A = [0.3, 0.7, 0; 0, 0.5, 0.5; 0, 0, 1];

```

```
30 b1 = DiscreteD([1, 0, 0, 0]);
31 b2 = DiscreteD([0, 0.5, 0.4, 0.1]);
32 b3 = DiscreteD([0.1, 0.1, 0.2, 0.6]);
33
34 B = [b1; b2; b3];
35
36 mc = MarkovChain(q, A);
37
38 x = [1, 2, 4, 4, 1];
39 pX = prob(B, x);
40
41 [~, c_i] = forward(mc, pX);
42
43 betaHat_i = backward(mc, pX, c_i);
```