EQ2340 Pattern Recognition Project Assignment 3 Report

Feiyang Liu (liuf@kth.se)
Baoqing She (baoqing@kth.se)

***Abstract*—HMM[1](Hidden Markov Model) is widely used to characterize signals in many fields nowadays, due to its superiority, such as simple mathematical structure, describing complex feature-vector sequences, automatically adapted among others. We want to create and analyze HMM in our project.**

I. Introduction

In this report, We are supposed to implement and verify backward algorithm which is used for HHM training. However, we implement both backward and forward algorithm. All of our simulations are based on MATLAB, and all relevant source codes are attached in zip archive.

II. Theoretical Knowledge

The Backward Algorithm is used to calculate a matrix β of conditional probabilities of the final part of an observed sequence $(x_{t+1}...x_T)$, given an HMM and the state $S_t = i$ at time t. The result is known as the backward variables, defined through

$$\beta_{i,t} = P(x_{t+1}...x_T|S_t = i, \lambda) \tag{1}$$

This forward algorithm calculates conditional state probabilities, given an observed feature sequence $(x_1...x_t...)$ and an HMM ,as

$$\hat{\alpha}_{i,t} = P(S_t = i|x_1...x_t\lambda) \tag{2}$$

III. implement algorithm

A. Backward Algorithm

Initialization Using Eq (5.64) and (5.65) in [1] to define the slightly different initializations needed for infinite-duration and finite-duration HMMs.

Backward Step: Using Eq. (5.70) in [1] to apply to any type of HMM.

B. Forward Algorithm

Initialization: Using Eqs. (5.42), (5.44) in [1]

Forward Step: Using Eqs. (5.50), (5.52) in [1]

Termination: Using Eq. (5.53) in [1], only needed for finite-duration HMMs.

IV. verify algorithm

To verify that our backward algorithm, we assume

$$q = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3}$$

$$A = \begin{bmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.9 & 0.1 \end{bmatrix} \tag{4}$$

where the state-conditional output distribution is a scalar Gaussian with mean $\mu 1 = 0$ and standard deviation $\sigma 1 = 1$ for state 1, and another Gaussian with $\mu 2 = 3$ and $\sigma 2 = 2$ for state 2. Previous calculations show that, for this HMM and the observation sequence $x = (0.2, 2.6, 1.3)$. Feeding these results into the forward algorithm, the result should be:

$$alfaHat = \begin{bmatrix} 1.0000 & 0.3847 & 0.4189 \\ 0 & 0.6153 & 0.5811 \end{bmatrix} \tag{5}$$

$$c = \begin{bmatrix} 1.0000 & 0.1625 & 0.8266 & 0.0581 \end{bmatrix} \tag{6}$$

Our simulation result is shown in (7) (8), which are identical to the calculated results.

$$\hat{alfaHat} = \begin{bmatrix} 1.0000 & 0.3847 & 0.4189 \\ 0 & 0.6153 & 0.5811 \end{bmatrix} \tag{7}$$

$$\hat{c} = \begin{bmatrix} 1.0000 & 0.1625 & 0.8266 & 0.0581 \end{bmatrix} \tag{8}$$

In the previous step, the Forward Algorithm gives the scale factors $c = (1, 0.1625, 0.8266, 0.0581)$. Feeding these results into the Backward algorithm, further computations show that the final scaled backward variables for this simple test example should be about

$$betaHat = \begin{bmatrix} 1.0003 & 1.0393 & 0 \\ 8.4182 & 9.3536 & 2.0822 \end{bmatrix} \tag{9}$$

Our simulation result $\hat{betaHat}$ is shown in (9), which is identical to betaHat. Hence, our backward algorithm is verified.

$$\hat{betaHat} = \begin{bmatrix} 1.0003 & 1.0393 & 0 \\ 8.4182 & 9.3536 & 2.0822 \end{bmatrix} \tag{10}$$

We also change the transition matrix A into a square matrix B to test infinite HMM. We still can get results from the algorithm we implemented. Hence, our algorithm works for both infinite and finite HMM.

$$B = \begin{bmatrix} 0.9 & 0.1 \\ 0.7 & 0.3 \end{bmatrix} \tag{11}$$

References

[1] Arne Leigon and Gustav Eje Henter. Pattern Recognition Fundamental Theory and Exercise Problems

```
function betaHat=backward(mc,pX,c)

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

q = mc.InitialProb;
a = mc.TransitionProb;
%%define a logical, if yes then finity, if no then infinity
decide = (size(a,1) ~= size(a,2));

n = size(mc.InitialProb,1); %number of states

beta = zeros(n,T);
betaHat = zeros(n,T);

%initialize
%if is finity
if decide
    beta(:,T) = a(:,n+1);
    betaHat(:,T) = beta(:,T) / (c(T) * c(T+1));
%if is in-finity
else
    beta(:,T) = 1;
    betaHat(:,T) = 1 / c(T);
end

%backward
for t = (T - 1) : (-1) : 1
    sigma =   a(:,1:n) * (pX(:,t+1) .* betaHat(:,t+1));
    betaHat(:,t) = sigma / c(t);
end

end
```

Fig. 1

```
function [alfaHat, c]=forward(mc,pX)


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

q = mc.InitialProb;
a = mc.TransitionProb;
%%define a logical, if yes then finity, if no then infinity
decide = (size(a,1) ~= size(a,2));

n = size(mc.InitialProb,1);%number of states

alfa_temp = zeros(n,T);
alfaHat = zeros(n,T);
c = zeros(1,T);

%initialize
alfa_temp(:,1) = pX(:,1) .* q;
c(1) = sum(alfa_temp(:,1));
alfaHat(:,1) = alfa_temp(:,1) ./ c(1);

for t = 2 : T
    sigma =   (alfaHat(:,t-1)' * a(:,1:n))';
    alfa_temp(:,t) = pX(:,t) .* sigma;
    c(t) = sum(alfa_temp(:,t));
    alfaHat(:,t) = alfa_temp(:,t) ./ c(t);

end
 %if finite then formulate the terminal
    if decide
        exit = alfaHat(:,t)' * a(:,n+1);
        c = [c exit];
    end
end
```

Fig. 2