EQ2340 - Pattern Recognition
A.3: Algorithm Implementation - Backward Algorithm

Tianxiao Zhao
tzh@kth.se

Zheyu Zhang
zheyuz@kth.se

October 11, 2017

Verify the Implementation

In this assignment, we implement the Backward Algorithm of the `MarkovChain` class, and verify our implementation using the following two tests.

Finite-duration Test

For the finite-duration HMM, we define a Markov chain with the following parameters:

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

According to previous calculations, for this HMM and an observation sequence $\underline{x} = (-0.2,\ 2.6,\ 1.3)$, the Forward Algorithm gives scale factors $\underline{c} = (1,\ 0.1625,\ 0.8266,\ 0.0581)$. We feed these quantities into our code below and run it to start the test:

```
% finite-duration test
q = [1; 0];
A = [0.9 0.1 0; 0 0.9 0.1];
x = [-0.2 2.6 1.3];
B1 = GaussD('Mean', 0, 'StDev', 1);
B2 = GaussD('Mean', 3, 'StDev', 2);

mc = MarkovChain(q, A);
pX = prob([B1 B2], x);
c = [1 0.1625 0.8266 0.0581];

disp('--------- finite-duration test ----------');
betaHat = backward(mc, pX, c)
```

The final printout is pasted below. This computation corresponds to the result provided by the instruction, which means that our implementation passes the finite-duration test.

```
> --------- finite-duration test ----------

> betaHat =

    1.0003    1.0393         0
    8.4182    9.3536    2.0822
```

Infinite-duration Test

Now, we change the parameters as following to define a infinite-duration HMM:

$$q = \begin{pmatrix} 1 \\ 0 \end{pmatrix} ; \; A = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix} ; \; B \sim \begin{pmatrix} \mathcal{N}(0,1) \\ \mathcal{N}(3,2) \end{pmatrix} \tag{0.2}$$

This infinite-duration HMM also generates an observation sequence $\underline{x} = (-0.2,\ 2.6,\ 1.3)$. To get the corresponding scale factors, we also complete the `@MarkovChain/forward` function (`forward.m` in the folder `@MarkovChain`) and feed the scaled state-conditional probability values `pX` $= \begin{pmatrix} 1 & 0.0695 & 1 \\ 0.1418 & 1 & 0.8111 \end{pmatrix}$ (calculated by function `GaussD/prob`) into it. The scale factors are $\underline{c} = (1,\ 0.1625,\ 0.8881)$.

We first calculate the scaled backward variables $\hat{\beta}_{j,t}$ for $t = 1, 2, 3$ by hand. The calculation process is documented below:

$$\hat{\beta}_{j,3} = 1/c_3 = 1/0.8881 = 1.1260,\ j = 1, 2 \tag{0.3}$$

$$\hat{\beta}_{1,2} = \frac{1}{c_2} \cdot \sum_{j=1}^{N} a_{1,j} b_{j,3} \hat{\beta}_{j,3} = \frac{1}{0.1625} \cdot (0.9 \cdot 1 \cdot 1.126 + 0.1 \cdot 0.8111 \cdot 1.126) = 6.7983 \tag{0.4}$$

$$\hat{\beta}_{2,2} = \frac{1}{c_2} \cdot \sum_{j=1}^{N} a_{2,j} b_{j,3} \hat{\beta}_{j,3} = \frac{1}{0.1625} \cdot (0.1 \cdot 1 \cdot 1.126 + 0.9 \cdot 0.8111 \cdot 1.126) = 5.7512 \tag{0.5}$$

$$\hat{\beta}_{1,1} = \frac{1}{c_1} \cdot \sum_{j=1}^{N} a_{1,j} b_{j,2} \hat{\beta}_{j,2} = 1 \cdot (0.9 \cdot 0.0695 \cdot 6.7983 + 0.1 \cdot 1 \cdot 5.7512) = 1.0004 \tag{0.6}$$

$$\hat{\beta}_{2,1} = \frac{1}{c_1} \cdot \sum_{j=1}^{N} a_{2,j} b_{j,2} \hat{\beta}_{j,2} = 1 \cdot (0.1 \cdot 0.0695 \cdot 6.7983 + 0.9 \cdot 1 \cdot 5.7512) = 5.2233 \tag{0.7}$$

In a more compact format, the above results can be rewritten into a matrix:

$$\hat{\beta} = \begin{pmatrix} 1.0004 & 6.7983 & 1.1260 \\ 5.2233 & 5.7512 & 1.1260 \end{pmatrix} \tag{0.8}$$

Then we run the following code to compute the backward variables in practice:

```
% infinite-duration test
q = [1; 0];
A = [0.9 0.1; 0.1 0.9];
x = [-0.2 2.6 1.3];
B1 = GaussD('Mean', 0, 'StDev', 1);
B2 = GaussD('Mean', 3, 'StDev', 2);
pX = prob([B1 B2], x);

% compute scaled factors with forward algorithm
mc = MarkovChain(q, A);
[¬, c] = forward(mc, pX);

disp('--------- infinite-duration test ----------');
betaHat = backward(mc, pX, c)
```

The output of this run is attached below. As we can see, the results are quite close to the analytical ones, but have a ~ 0.001 level of absolute error. This may be caused by an accumulation of round-off errors during the calculations. This acceptable error level proves that our implementation could also handle the infinite-duration test cases.

```
> --------- infinite-duration test ----------

> betaHat =

    1.0000    6.7973    1.1260
    5.2223    5.7501    1.1260
```