# EQ2340 - Pattern Recognition
# A.4: Code Verification and Signal Database

Tianxiao Zhao
tzh@kth.se

Zheyu Zhang
zheyuz@kth.se

October 20, 2017

## 1 Code Verification

### 1.1 Verify Forward Algorithm

The verification process is quite similar to the backward case. For the finite-duration test, we create a HMM according to the parameters in the instruction:

$$q = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \; A = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.9 & 0.1 \end{pmatrix}; \; B \sim \begin{pmatrix} \mathcal{N}(0,1) \\ \mathcal{N}(3,2) \end{pmatrix} \tag{1.1}$$

For an observed feature sequence $\underline{x} = (-0.2,\ 2.6,\ 1.3)$, we run the fixed `forward` function and get the following printout:

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

> alfaHat =

    1.0000    0.3847    0.4189
         0    0.6153    0.5811

> c =

    1.0000    0.1625    0.8266    0.0581
```

The results above are exactly the same with the ones shown in the instruction (hand calculation). This proves that our fixed `forward` function performs correctly when dealt with the finite-duration case. The code for this part is attached below.

```
% 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);
pX = prob([B1 B2], x);

% compute scaled factors with forward algorithm
mc = MarkovChain(q, A);
disp('--------- finite-duration test ----------');
[alfaHat, c] = forward(mc, pX)
```

For the infinite-duration test, we change the parameters of HMM in the previous example to:

$$q = \begin{pmatrix} 1 \\ 0 \end{pmatrix}; \; A = \begin{pmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{pmatrix}; \; B \sim \begin{pmatrix} \mathcal{N}(0,1) \\ \mathcal{N}(3,2) \end{pmatrix} \tag{1.2}$$

We first calculate the forward variable $\hat{\alpha}_t$ and factors $c_t$ by hand. To complete the computation, the scaled state-conditional probability values `pX` $= \left( \begin{smallmatrix} 1 & 0.0695 & 1 \\ 0.1418 & 1 & 0.8111 \end{smallmatrix} \right)$ are calculated by function `GaussD/prob`. The full calculation process is documented below:

$$\alpha_{1,1}^{temp} = q_1 b_1(x_1) = 1 \cdot 1 = 1, \; \alpha_{2,1}^{temp} = q_2 b_2(x_1) = 0 \cdot 0.1418 = 0 \tag{1.3}$$

$$c_1 = \alpha_{1,1}^{temp} + \alpha_{2,1}^{temp} = 1 + 0 = 1 \tag{1.4}$$

$$\hat{\alpha}_{1,1} = \alpha_{1,1}^{temp}/c_1 = 1/1 = 1,\ \hat{\alpha}_{2,1} = \alpha_{2,1}^{temp}/c_1 = 0/1 = 0 \tag{1.5}$$

$$\alpha_{1,2}^{temp} = b_1(x_2)\cdot(\hat{\alpha}_{1,1}a_{1,1} + \hat{\alpha}_{2,1}a_{2,1}) = 0.0695\cdot(1\cdot 0.9 + 0\cdot 0.1) = 0.06255 \tag{1.6}$$

$$\alpha_{2,2}^{temp} = b_2(x_2)\cdot(\hat{\alpha}_{1,1}a_{1,2} + \hat{\alpha}_{2,1}a_{2,2}) = 1\cdot(1\cdot 0.1 + 0\cdot 0.9) = 0.1 \tag{1.7}$$

$$c_2 = \alpha_{1,2}^{temp} + \alpha_{2,2}^{temp} = 0.06255 + 0.1 = 0.16255 \tag{1.8}$$

$$\hat{\alpha}_{1,2} = \alpha_{1,2}^{temp}/c_2 = 0.06255/0.16255 = 0.3848,\ \hat{\alpha}_{2,2} = \alpha_{2,2}^{temp}/c_2 = 0.1/0.16255 = 0.6152 \tag{1.9}$$

$$\alpha_{1,3}^{temp} = b_1(x_3)\cdot(\hat{\alpha}_{1,2}a_{1,1} + \hat{\alpha}_{2,2}a_{2,1}) = 1\cdot(0.3848\cdot 0.9 + 0.6152\cdot 0.1) = 0.4078 \tag{1.10}$$

$$\alpha_{2,3}^{temp} = b_2(x_3)\cdot(\hat{\alpha}_{1,2}a_{1,2} + \hat{\alpha}_{2,2}a_{2,2}) = 0.8111\cdot(0.3848\cdot 0.1 + 0.6152\cdot 0.9) = 0.4803 \tag{1.11}$$

$$c_3 = \alpha_{1,3}^{temp} + \alpha_{2,3}^{temp} = 0.4078 + 0.4803 = 0.8881 \tag{1.12}$$

$$\hat{\alpha}_{1,3} = \alpha_{1,3}^{temp}/c_3 = 0.4078/0.8881 = 0.4592,\ \hat{\alpha}_{2,3} = \alpha_{2,3}^{temp}/c_3 = 0.4803/0.8881 = 0.5408 \tag{1.13}$$

The above results can be rewritten in a matrix format:

$$\hat{\alpha} = \begin{pmatrix} 1 & 0.3848 & 0.4592 \\ 0 & 0.6152 & 0.5408 \end{pmatrix},\ c = \begin{pmatrix} 1 & 0.16255 & 0.8881 \end{pmatrix} \tag{1.14}$$

Meanwhile, we run the following code to walk through the forward calculation process. Results from the code are put below. As we can see, the results from both methods are quite close to each other. The negligible difference is mainly caused by the round-off errors during Matlab calculations. Therefore, we are sure that our `forward` function is perfectly fixed and could well handle both cases.

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

> alfaHat =

    1.0000    0.3847    0.4591
         0    0.6153    0.5409

> c =

    1.0000    0.1625    0.8881
```

The code for this part is attached below. For more details, please check `forward.m` in the folder `@MarkovChain`.

```matlab
% 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);
disp('--------- infinite-duration test ----------');
[alfaHat, c] = forward(mc, pX)
```

## 1.2 Verify log-probability

Besides, we also complete the function `@HMM/logprob` to calculate the log-probability of an observed sequence, $\log P(\underline{X} = \underline{x} \mid \lambda)$. The code for verification is attached here:

```
%% verify log-prob
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);
pX = prob([B1 B2], x);

% create a HMM
mc = MarkovChain(q, A);
hmm = HMM(mc, [B1 B2]);

% print out log-prob
disp('--------- log-probability of X ----------');
logP = logprob(hmm, x)
```

The results from the above code is as followed and they correspond to the ones in the instruction. Thus, we believe that our function `@HMM/logprob` is implemented correctly. For more details, please check `logprob.m` in the folder `@HMM`.

```
--------- log-probability of X ----------

c =

    0.3910    0.0318    0.1417    0.0581

logP =

   -9.1877
```

## 1.3 Code Diagnosis

The original code has several problems that affect the final results. Here we list them below according to the order in which we detect them.

1. The first time we run the function, we get an error report that an issue of index exceeding matrix dimensions occurs in line 64. This is a runtime error caused by line 43. There is no need to add "+10" at the end of this line. Changing it to "`numberOfStates = length(mc.InitialProb);`" could solve this issue;

2. Run the code again, and now no errors are reported. But the results are wrong both in dimension and value. The first column of `alfaHat` and `c` are correct, so the initialization process is out of suspicion. Since the number of columns is fewer than expected, then the problem should lie in a wrong count of time frames. There is a syntax error in line 40, which retrieves number of rows instead of the matrix `pX`. We change it to "`T=size(pX,2);`" accordingly;

3. Now the dimension of `alfaHat` is correct while that of `c` is not. We are sure that the problem is around the calculation of termination state in finite-duration test case. First of all, line 52-58 and line 87 are kind of redundant so we can comment these part out to keep it concise. After this, we find that there is a logical flaw in line 48 that `c` should have the same length with the number of time frames elapsed. So, here we change it to "`c = zeros(1,T);`" and correspondingly change line 90 to "`c = [c alfaHat(:,T)'*A(:,numberOfStates+1)];`" to deal with the finite-duration case;

4. Now we'll figure out why it can't give out correct values. We think there are two errors in the iterative calculation process, one in line 78 and the others in line 82. For line 78, no summation operation should be added to the matrix `B`, so we just get rid of it. For line 82, `alfaTemp` should be normalized by the factor at its corresponding time frame, not always the first one. So we change it to "`alfaTemp(j) = alfaTemp(j)/c(t);`" and now the outputs are correct.

The problems above are kind of fatal in the sense that missing one of them will definitely give out wrong outputs or report errors directly. Apart from them, there are other mild problems in this program that have little or no affects to the results. We also fix them and now we list them below.

1. Variable `Alpha_old` has nothing to do with the forward algorithm. So any line of code (line 42, 65, 71 and 83) that is involved with this variable could be commented out for simplicity;

2. In line 61, variable `initAlfaTemp` is initialized to be a fixed-sized vector which may be right with the test case but cause a runtime error when dealt with a HMM with more than 2 states. Therefore, we adapt it so that the size of variable `initAlfaTemp` could also change with input HMM.

## 2 Song Database

Before generating the song database, we read through some relevant materials about query-by-humming recognition systems. We noticed that there is a dataset called MTG-QBH [1] which includes 118 recordings of song melodies made under comprehensive variations, and it is frequently used in related research papers. Therefore we decide to modify this mature dataset and create our song database from it.

We first randomly select 10 humming melodies made by different men and women (musical experience also varies) from MTG-QBH dataset, and then divide each piece of melodies into multiple segments respectively. The duration of each snippets is around 8 seconds. According to the documentation of the MTG-QBH dataset, all those recordings are obtained in a quiet environment by a laptop's built-in microphone. Detailed information about our database can be found in Table 1.

Table 1: Information about our own database which is built on dataset MTG-QBH.

| Index | Song title | Filename prefix | Number of segments | Gender of performer | Class label |
|---|---|---|---|---|---|
| 1 | Ob la di ob la da | q9 | 15 | female | obladi |
| | | q21 | 10 | male | |
| 2 | Strangers in the night | q10 | 7 | female | strangers |
| | | q102 | 13 | male | |
| 3 | More than words | q13 | 17 | female | morethwor |
| | | q105 | 16 | male | |
| 4 | Love me tender | q18 | 11 | female | lovemetend |
| | | q108 | 10 | male | |
| 5 | Let it be | q57 | 11 | male | letitbe |
| | | q110 | 11 | female | |
| 6 | Across the universe | q12 | 16 | female | across |
| 7 | Wish you were here | q43 | 15 | female | wishyouw |
| 8 | Sweet home Alabama | q55 | 16 | male | sweethome |
| 9 | In the mood | q107 | 15 | male | inthemood |
| 10 | Enjoy the silence | q114 | 15 | male | enjoysilen |

As we can see in Table 1, each melody in our database has at least 15 examples, and some of the melodies contain recordings of both male and female performers. This setting allows us to test if our recognizer could still give right classifications for those songs performed only by one performer, and in other words, test its generalization. For different examples of the same melody, generally the variations lie in how fast it is performed (tempo), how loud it is performed (intensity) and who perform it (gender).

Before we train our recognizer, we need to divide our database into two parts, which are training set and test set. The method we are going to do this with is called k-fold cross validation: we first shuffle all the recording segments we have and partition them into $k$ parts. Each part will be regraded as testing set in turn while the other $(k-1)$ parts are used together as training set once the test set is decided. In this way, we will train and test our recognizer $k$ times and then average the test accuracy as the final result. This method could make the full use of our database for training and hence can enhance our recognizer's generalization in theory. Since each melody has different numbers of examples in the database, we may need to prune it beforehand to combat imbalanced classes.

## References

[1] Music Technology Group (UPF). Mtg-qbh: Query by humming dataset. `https://www.upf.edu/web/mtg/mtg-qbh`, 2012. [Online; accessed 19-October-2017].