A.3 Algorithm Implementation

In this assignment you will implement one of two very important HMM algorithms: the Forward Algorithm and the Backward Algorithm. You will be assigned to one of these two algorithms by the teaching assistants. Read the appropriate section below and implement the associated functions.

A.3.1 The Forward Algorithm

In this section you will implement and verify a MatLab function to perform the *Forward Algorithm*. This algorithm calculates conditional state probabilities, given an observed feature sequence $(\boldsymbol{x}_1 \dots \boldsymbol{x}_t \dots)$ and an HMM λ, as

$$\hat{\alpha}_{j,t} = P(S_t = j | \boldsymbol{x}_1 \dots \boldsymbol{x}_t, \lambda).$$

You can also use the *forward scale factors* $(c_1 \dots c_t \dots)$, calculated by the algorithm, to determine the total probability $P(\underline{\boldsymbol{X}} = \underline{\boldsymbol{x}} | \lambda)$ of the observed sequence given the HMM. The Forward Algorithm is also an essential step in HMM training, and you will need it for your final recognizer.

The Forward Algorithm consists of three steps, defined by equations in Sec. 5.4:

Initialization: Eqs. (5.42)–(5.44)

Forward Step: Eqs. (5.50)–(5.52)

Termination: Eq. (5.53), only needed for finite-duration HMMs.

Implement the Forward Algorithm

In the `PattRecClasses` framework the Forward Algorithm is a method of the `MarkovChain` class, because the algorithm does not need to know anything about the type of output probability distribution used by the HMM. The Forward Algorithm needs as input only a matrix `pX` with values *proportional to* the state-conditional probability mass or density values for each state and each frame in the observed feature sequence.

The proportionality scale factors, one for each frame, must be defined and stored by the calling function, if needed. This is not the responsibility of the Forward Algorithm. The purpose of the scaling is to avoid very small probability values, which can cause numerical problems in the computations.[7]

Your task is to complete the `@MarkovChain/forward` method as specified in the function interface and comments.

[7] Another approach to numerically stable HMMs is described in the technical report `http://bozeman.genome.washington.edu/compbio/mbt599_2006/hmm_scaling_revised.pdf` by Tobias P. Mann.

Note especially that your function must work for both *infinite-duration* and *finite-duration* Markov chains, and that the output results are slightly different in these two cases. Save your finished version of the function under the same file name in the same directory.

Verify the Implementation

The only way to test your implementation is to do a calculation by hand, and compare the result with the function output. However, this tedious calculation has already been done by previous students.

Create a finite-duration test HMM with a Markov chain given by

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

and 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. Then use your own `forward` implementation to calculate $\hat{\alpha}_{j,t}$, $t = 1 \ldots 3$ and c_t, $t = 1 \ldots 4$, with an observed finite-duration feature sequence $\underline{x} = (-0.2, 2.6, 1.3)$. For this simple test example, the result of your `forward` function should be

```
alfaHat =
    1.0000    0.3847    0.4189
         0    0.6153    0.5811

c =
    1.0000    0.1625    0.8266    0.0581
```

assuming you applied `forward` directly to the scaled probabilities produced by `prob`. If you used the accompanying scale factors to undo the scaling before calling `forward`, the first three elements in `c` may differ. This is not recommended.

Since your implementation also must work for infinite-duration HMMs, it is a good idea to figure out a way to test this case as well.

Probability of a Feature Sequence

Later in the project you will need a function to calculate the log-probability of an observed sequence, $\log P(\underline{\boldsymbol{X}} = \underline{\boldsymbol{x}}|\lambda)$, given an HMM instance λ. The log-probability is used since the regular observation probability $P(\underline{\boldsymbol{X}} = \underline{\boldsymbol{x}}|\lambda)$ very often is too small to be represented in a computer. For this reason $P(\underline{\boldsymbol{X}} = \underline{\boldsymbol{x}}|\lambda)$ should never be used directly in any computations.

Complete the small MatLab function `@HMM/logprob` to perform the log-probability calculation by calling your `forward` method with the scaled conditional probability values supplied by the `OutputDistr` `prob`-method (try

`help GaussD/prob` for more information). However, do not forget to include the scale factors in the subsequent calculation. For the HMM and observation sequence above the result should be that $\log P(\underline{X} = \underline{x}|\lambda) \approx -9.1877$, using the natural logarithm, if your implementation works for this example. Your function should also work when the input is an array of HMM objects, and compute the probability of the feature sequence under each model.

Note that if your `c`-vector looks like

```
c =
    0.3910    0.0318    0.1417    0.0581
```

it is likely that your implementation of the Forward Algorithm is correct, but that you are applying it to the true, unscaled probabilities by accounting for the scaling factors already before running `forward`. This can lead to problems later on. To get numerically robust calculations of very small log-probabilities, which is vital for later parts of the project, you will have to apply `forward` to the *scaled* probability values `pX`, as given by `prob`, first—only thereafter should the scaling factors be taken into account.

Your assignment report should include

- A copy of your finished `@MarkovChain/forward` function.
- A copy of your finished `@HMM/logprob` function.

A.3.2 The Backward Algorithm

In this project step you will implement and verify a MatLab function to perform the *Backward Algorithm*. The Backward Algorithm will be needed later for HMM training.

The Backward Algorithm is used to calculate a matrix $\boldsymbol{\beta}$ of conditional probabilities of the final part of an observed sequence $(\boldsymbol{x}_{t+1} \dots \boldsymbol{x}_T)$, given an HMM λ and the state $S_t = i$ at time t. The result is known as the *backward variables*, defined through

$$\beta_{i,t} = P(\boldsymbol{x}_{t+1} \dots \boldsymbol{x}_T | S_t = i, \lambda)$$

for an infinite-duration HMM. $\beta_{i,t}$ has a slightly different interpretation for finite-duration HMMs, as explained in Sec. 5.5.

In practice it is numerically preferable to calculate the *scaled backward variables* $\hat{\beta}_{i,t}$ instead, which are proportional to the regular backward variables. The Backward Algorithm calculates these variables in two steps, defined by equations in Sec. 5.5:

Initialization: Eqs. (5.64) and (5.65) define the slightly different initializations needed for infinite-duration and finite-duration HMMs.

Backward Step: Eq. (5.70) applies to any type of HMM.

Implement the Backward Algorithm

In the `PattRecClasses` framework the Backward Algorithm is a method of the `MarkovChain` class, since the algorithm does not need to know anything about the type of output probability distribution used by the HMM. The Backward algorithm takes as input a matrix with values proportional to the state-conditional probability mass or density values for each state and each element in the observed feature sequence, along with a corresponding sequence of scale factors $(c_1 \dots c_T)$ computed by the Forward Algorithm in the previous section.

Your task is to complete the `@MarkovChain/backward` method as specified in the function interface and comments. Because of the object-oriented nature of the HMM implementation, you need not have a working Forward Algorithm in order to write the Backward Algorithm code, as seen below.

Note especially that your function must accept either an *infinite-duration* or a *finite-duration* HMM. The output has exactly the same format in both cases, although the theoretical interpretation of the output is slightly different. Save your finished version of the function under the same file name in the same directory.

Verify the Implementation

One of the most rigorous ways to test your implementation is to do a calculation by hand, and compare the result with the function output. Fortunately for you, this tedious calculation has already been carried out by previous students, and you will just use their results here.

Create a finite-duration test HMM with a Markov chain given by

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

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 $\underline{x} = (-0.2, 2.6, 1.3)$, the Forward Algorithm gives the scale factors $\underline{c} = (1, 0.1625, 0.8266, 0.0581)$. Feeding these results into the Backward algorithm, further computations show that the final scaled backward variables $\hat{\beta}_{j,t}$, $t = 1 \dots 3$ for this simple test example should be about

```
betaHat =
    1.0003    1.0393         0
    8.4182    9.3536    2.0822
```

These numerical results require that all calculations use the scaled state-conditional probability values `pX` as supplied by the `OutputDistr prob`-method (try `help GaussD/prob` for more information). If you used the

accompanying scale factors from `prob` to undo the scaling before calling `backward`, you may get different results. This is not recommended, as the unscaled values may be very small, leading to numerical problems.

Since your implementation also must work for infinite-duration HMMs, it is recommended that you figure out a way to test this case as well.

Your assignment report should include

- A copy of your finished `@MarkovChain/backward` function.