EQ2341 Pattern Recognition and Machine Learning

Assignment 3: Forward Algorithm

Hanqi Yang hanqi@kth.se

Qian Zhou qianzho@kth.se

May 6, 2022

1 Introduction

In this assignment we will implement one of two very important HMM algorithms: the Forward Algorithm. This algorithm calculates conditional state probabilities, given an observed feature sequence $(x_1, ..., x_t, ...)$ and an HMM λ. The Forward Algorithm consists of three steps: initialization, forward Step, termination.

2 Forward algorithm procedure

In this section we will discuss how to implement the forward algorithm, an essential step in HMM training. The algorithm calculates conditional state probabilities, given an observed feature sequence $(x_1, ..., x_t, ...)$ and a known HMM parameter $\lambda = \{q, A, B\}$ as

$$\hat{\alpha}_{j,t} = P(S_t = j | x_1, ..., x_t, \lambda) \tag{1}$$

In our case, we only have $\lambda = \{q, A\}$ and the state-conditional output distribution for two scalar Gaussian states, so we generate a B matrix using these distributions as we did in assignment 2.

Equation 2 to 8 expose the mathematical methodology of the forward algorithm.

Initialization: When $t = 1$,

$$\alpha_{j,1} = q_j b_j(\mathbf{x}_1),\ j = 1, ..., N \tag{2}$$

$$c_1 = \sum_{k=1}^{N} \alpha_{k,1} \tag{3}$$

$$\hat{\alpha}_{j,1} = \alpha_{j,1}/c_1,\ j = 1, ..., N \tag{4}$$

Forward step: For $t = 2, ..., T$, as

$$\alpha_{j,t} = b_j(\mathbf{x}_t)(\sum_{i=1}^{N} \hat{\alpha}_{i,t-1} a_{ij}),\ j = 1, ..., N \tag{5}$$

$$c_t = \sum_{k=1}^{N} \alpha_{k,t} \tag{6}$$

$$\hat{\alpha}_{j,t} = \alpha_{j,t}/c_t,\ j = 1, ..., N \tag{7}$$

Termination: For a finite-duration HMM, the special exit condition for a finite observed sequence should be included:

$$c_{T+1} = \sum_{k=1}^{N} \hat{\alpha}_{k,T} a_{k,N+1} \tag{8}$$

Using the three steps above, the function `@MarkovChain/forward` is implemented and returns the results of the forward variables $\hat{\alpha}_{j,t}$ and the forward scale factors c_t.

Then we calculate the probability of the observed sequence $P(\mathbf{X} = \mathbf{x}|\lambda)$ using the function `@HMM/logprob`. The log-probability is obtained by applying the natural logarithm of the multiplication of the forward scale factor c_t, which corresponds to the summation of $ln(c_t)$. Equation 9 and 10 are used for infinite-duration and finite-duration HMM respectively.

$$lnP[x_1, ..., x_T|\lambda] = \sum_{t=1}^{T} lnc_t \tag{9}$$

$$lnP[x_1, ..., x_T, S_{T+1} = N + 1|\lambda] = \sum_{t=1}^{T+1} lnc_t \tag{10}$$

3 Results

In order to implement the forward algorithm, we complete the function `@MarkovChain/forward`, and `@HMM/logprob`. Then we just need a main function to test our code.

3.1 Finite duration HMM

We create a finite-duration test HMM with a Markov chain given by

$$q = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad A = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.9 & 0.1 \end{pmatrix} \tag{11}$$

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. In addition, the observed finite-duration feature sequence $x = (-0.2, 2.6, 1.3)$. Then we use our own forward implementation to calculate $\hat{\alpha}_{j,t},\ t = 1, ..., 3$ and $c_t,\ t = 1, ..., 4$, the results are

$$\hat{\alpha} = \begin{pmatrix} 1 & 0.38470424 & 0.41887466 \\ 0 & 0.61529576 & 0.58112534 \end{pmatrix} \tag{12}$$

$$c_t = \begin{pmatrix} 1 & 0.16252347 & 0.82658096 & 0.05811253 \end{pmatrix} \tag{13}$$

The results correspond with the calculation by hand. After that, we need to calculate the log-probability of an observed sequence. For the HMM and observation sequence above the result should be that $logP(\mathbf{X} = \mathbf{x}|\lambda) \approx -9.1877$, and our result is -9.18773, which is almost the same.

Therefore, our implementation of the Forward Algorithm works well for finite duration HMM.

3.2 Infinite duration HMM

For the infinite duration HMM, the parameters are given as

$$q = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \quad A = \begin{pmatrix} 0.3 & 0.7 & 0 \\ 0 & 0.5 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0.4 & 0.1 \\ 0.1 & 0.1 & 0.2 & 0.6 \end{pmatrix} \tag{14}$$

The observed finite duration feature sequence is $x = (1, 2, 4, 4, 1)$. Using our code to test, we get the results as

$$\hat{\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0.14285714 & 0.01265823 & 0 \\ 0 & 0 & 0.85714286 & 0.98734177 & 1 \end{pmatrix} \tag{15}$$

$$c_t = \begin{pmatrix} 1 & 0.35 & 0.35 & 0.56428571 & 0.09936709 \end{pmatrix} \tag{16}$$

The results correspond with what they should be as

$$\hat{\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0.143 & 0.013 & 0 \\ 0 & 0 & 0.857 & 0.987 & 1 \end{pmatrix} \tag{17}$$

$$c_t = \begin{pmatrix} 1 & 0.35 & 0.35 & 0.564 & 0.099 \end{pmatrix} \tag{18}$$

After that, we need to calculate the log-probability of an observed sequence. In this infinite HMM and observation sequence, result should be $logP(\mathbf{X} = \mathbf{x}|\lambda) \approx -4.981$.

After running our code, the result is -4.98077, which is almost the same. Therefore, our code works well also for the infinite-duration HMM.

4 Conclusion

In this assignment, we implement the forward algorithm and test them with finite HMM and infinite HMM. The results of $\hat{\alpha}$, c and the log-probability of an observed sequence are almost the same as the theoretical value. In conclusion, our implementation of the Forward Algorithm succeed.

Appendix

Python code

@MarkovChain/forward:

```
def forward(self,pX):
    T = len(pX[0, :])
    N = self.nStates
    q = self.q
    A = self.A
    c = np.zeros(T)
    alpha = np.zeros((N, T))
    alfaHat = np.zeros((N, T))
    finitestate = 0
    if N != len(A[0, :]):
        finitestate = 1

    # Initialization
    for i in range(0, N):
        alpha[i, 0] = q[i] * pX[i, 0]
        c[0] = c[0] + alpha[i, 0]
    for i in range(0, N):
        alfaHat[i, 0] = alpha[i, 0] / c[0]

    # Forward Steps
```

```
    for t in range(1, T):
        for j in range(0, N):
            alpha[j, t] = pX[j, t] * (np.transpose(
                alfaHat[:, t - 1]) @ A[:, j])
            c[t] = c[t] + alpha[j, t]
        for j in range(0, N):
            alfaHat[j, t] = alpha[j, t] / c[t]

    for t in range(1, T):
        for j in range(0, N):
            alfaHat[j, t] = alpha[j, t] / c[t]

    # Termination
    if finitestate == 1:
        tempc = c
        c = np.zeros(T + 1)
        for i in range(0, T):
            c[i] = tempc[i]
        for i in range(0, N):
            c[T] = c[T] + alfaHat[i, T - 1] * A[i, N]

    return alfaHat, c
```

@HMM/logprob:

```
def logprob(self,x):
        mc = self.stateGen
        B = self.outputDistr
        pX = GaussD.gauss_logprob(B, x)
        for j in range(0, 3):
            for i in range(0, 2):
                pX[i, j] = np.exp(pX[i, j])
        alfaHat, c = mc.forward(pX)
        return c
```