EQ2341 Pattern Recognition and Machine Learning Assignment 3

Hang Qin
hangq@kth.se

Chenting Zhang
chzha@kth.se

May 19, 2023

1 Introduction

In this assignment, we are going to implement the Forward Algorithm and verify our implementation. The Forward Algorithm is an iterative procedure that can calculate the scaled forward variable $\hat{\alpha}_t$ and the forward scaled factor c_t from a given observed data and the model parameters. Some specific parameters and a short data sequence are used for validation.

2 Forward Algorithm Implementation

Here we implement the forward algorithm following the steps presented in the textbook. Figure 1 shows the corresponding code. The element in the input of our forward algorithm pX is the probability $b_j(x_t)$, which is the probability of the observed data at time step t are produced by the j^{th} state.

The forward algorithm contains three parts, initialization, forward step, and termination. The comments in the code illustrate that clearly. The comments also indicate which formula from the textbook the code implements by labeling it with the formula number. In the initialization part, the scaled forward variable $\hat{\alpha}_t$ and forward scaled factor c_t at the first time step are calculated. Then in the forward step part, $\hat{\alpha}_t$ and c_t of t from 2 to T are calculated. Finally, the termination part gives the result of c_{T+1}. Although the temporary forward variable are also calculated, they are not included in the results of the forward function. The results only involve $\hat{\alpha}_t$ and c_t.

```python
def forward(self, p_x):  # p_x is a matrix of shape N, T
    # print(p_x)
    alpha_hat, c = [], []  # scaled forward variable, scaled factors (c)
    n_states = p_x.shape[0]  # N
    t_max = p_x.shape[1]  # T

    """initialization"""
    a_temp = [self.q[j] * p_x[j, 0] for j in range(n_states)]  # eq 5.42
    c.append(sum(a_temp))  # eq 5.43
    alpha_hat.append([np.divide(a, c[0]) for a in a_temp])  # eq 5.44

    """forward step"""
    for t in range(1, t_max):
        a_temp = []
        for j in range(n_states):
            con_prob = sum([np.multiply(alpha_hat[t-1][i], self.A[i, j]) for i in range(n_states)])  # eq 5.49
            a_temp.append(p_x[j, t] * con_prob)  # eq 5.50
        c_t = sum(a_temp)  # eq 5.51
        c.append(c_t)
        alpha_hat.append([a/c_t for a in a_temp])  # eq 5.52

    """termination"""
    if self.is_finite:
        c_tail = sum([alpha_hat[-1][i] * self.A[i, -1] for i in range(n_states)])  # eq 5.53
        c.append(c_tail)

    return alpha_hat, c
```

Figure 1: The forward algorithm implementation

```python
def likelihood(gauss_dist, x):
    px = []
    for i, dist in enumerate(gauss_dist):
        # for one observed data x and distribution d, the probability Pr(x|d)
        px.append(dist.prob(x))
    px = np.array(px)
    return px
```

Figure 2: The likelihood function for pX calculation

```python
def logprob(self, p_x):
    alpha_hat, c = self.stateGen.forward(p_x)
    c_log = [np.log(c_i) for c_i in c]  # 5.54
    return sum(c_log)
```

Figure 3: The logprob function in HMM class

3 Forward Algorithm Verification

To verify our forward algorithm, we test it using the setup given by the textbook. Besides, to calculate the input of the forward function, pX, we write a function called likelihood, and it can return pX from the input distribution objects and observed data. Figure 2 shows the likelihood function. The logprob function in the HMM class is also implemented following the formula 5.54 in the textbook, as shown in Figure 3. The model parameters and observed data sequence are defined as follows.

$$q = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad A = \begin{pmatrix} 0.9 & 0.1 & 0 \\ 0 & 0.9 & 0.1 \end{pmatrix} \quad x = \begin{pmatrix} -0.2 & 2.6 & 1.3 \end{pmatrix}$$

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. Utilizing our forward algorithm functions and the logprob function, the results are shown in Figure 4. The outcome is consistent with the results shown in the textbook, showing that our function works correctly and successfully implements the forward algorithm.

```
x = [-0.2  2.6  1.3]

q = [1 0]

A =
 [[0.9 0.1 0. ]
 [0.  0.9 0.1]]

alpha_hat =
 [[1.    0.385 0.419]
 [0.    0.615 0.581]]

c = [1.    0.163 0.827 0.058]

P(X = x|λ)= -9.187726979475208

Process finished with exit code 0
```

Figure 4: The verify results