

Pattern Recognition Assignment 2: Speech Recognition

Hailey Yu Haihong  
Levi Leyh

May 19, 2023

Contents

<b>1</b>	<b>MarkovChain forward function.</b>	<b>1</b>
1.1	Code implementation . . . . .	1
1.2	Test of the code . . . . .	1
<b>2</b>	<b>HMM logprob function.</b>	<b>2</b>
2.1	Code implementation . . . . .	2
2.2	Test of the code . . . . .	2

# 1 MarkovChain forward function.

## 1.1 Code implementation

This is the used code:

Listing 1: MarkovChain/forward function.

```
1
2
3     def forward(self, pX):
4         """
5         Calculates the forward probabilities for a given observation sequence.
6
7         Parameters:
8         pX: array-like
9             The observation sequence (scaled).
10
11         Returns:
12         alpha: array-like, shape (T_obs, N_states)
13             Forward variable.
14         """
15         T_obs = len(pX[0])
16         N_states = len(self.q)
17
18         alpha_temp = np.zeros((N_states, T_obs))
19         c = np.zeros(T_obs)
20         alpha = np.zeros((N_states, T_obs))
21
22         # Initialization
23         #alpha_temp[:, 0] = q * np.array([dist.pdf(obs[0]) for dist in B])
24         alpha_temp[:, 0] = self.q * pX[:, 0]
25         c[0] = sum(alpha_temp[:, 0])
26         alpha[:, 0] = alpha_temp[:, 0]/(c[0])
27
28         # Recursion
29         for t in range(1, T_obs):
30             #alpha_temp[:, t] = np.array([dist.pdf(obs[t]) for dist in B]) * (alpha[:, t - 1].dot(
31                 A)[: -1])
32             alpha_temp[:, t] = pX[:, t] * (alpha[:, t - 1].T.dot(self.A)[: -1])
33             c[t] = sum(alpha_temp[:, t])
34             alpha[:, t] = alpha_temp[:, t]/(c[t])
35
36         #termination
37         if self.is_finite==True:
38             c = np.append(c, (alpha[:, t].T.dot(self.A[:,-1])))
39         return alpha, c
```

## 1.2 Test of the code

Using the test code below we have the following result

Listing 2: MarkovChain/forward function.

```
1 #test code on forward algorithm
2 q = np.array([1, 0])
3 A = np.array([[0.9, 0.1, 0], [0, 0.9, 0.1]])
4 B = [norm(0, 1), norm(3, 2)]
5 obs = [-0.2, 2.6, 1.3]
6
7 mc = MarkovChain( np.array( [1, 0] ), np.array( [[0.9, 0.1, 0], [0, 0.9, 0.1]] ) )
8
9 pX = np.zeros((2, len(obs)))
10 #normalize
11 for m in range(len(obs)):
12     scalar = np.max(np.array([norm.pdf(obs[m], 0, 1), norm.pdf(obs[m], 3, 2)]))
13     pX[0, m] = norm.pdf(obs[m], 0, 1) / scalar
14     pX[1, m] = norm.pdf(obs[m], 3, 2) / scalar
15
16 alpha, c = mc.forward(pX)
17 print("alpha_hat:\n", alpha)
18 print("c:\n", c)
```

```
alpha_hat:
[[1.          0.38470424 0.41887466]
 [0.          0.61529576 0.58112534]]
c:
[1.          0.16252347 0.82658096 0.05811253]
```

## 2 HMM logprob function.

### 2.1 Code implementation

This is the used code:

Listing 3: HMM/logprob functions

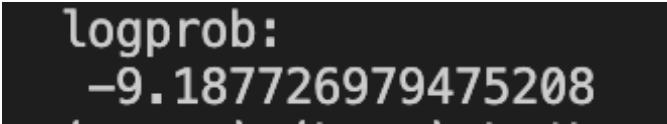
```
1 from scipy.stats import norm
2
3 def logprob(self, obs):
4     """
5     Calculates the log probability of the observation sequence given the HMM.
6
7     Parameters:
8     obs: array-like
9         The observation sequence.
10
11     Returns:
12     log_prob: float
13         The log probability of the observation sequence given the HMM.
14     """
15
16     pX = np.zeros((2, len(obs)))
17     #normalize
18     for m in range(len(obs)):
19         scalar = np.max(np.array([norm.pdf(obs[m], 0, 1), norm.pdf(obs[m], 3, 2)]))
20         #pX[0, m] = norm.pdf(obs[m], 0, 1) / scalar
21         #pX[1, m] = norm.pdf(obs[m], 3, 2) / scalar
22         for i in range(np.shape(self.stateGen.A)[0]):
23             pX[i, m] = self.outputDistr[i].prob(obs[m])
24     alpha, c = self.stateGen.forward(pX)
25     log_likelihood = np.sum(np.log(c))
26     return log_likelihood
```

### 2.2 Test of the code

Using the test code below we have the following result

Listing 4: MarkovChain/forward function.

```
1 #test code on logprob
2 g1 = GaussD( means=[0], stdevs=[1] )
3 g2 = GaussD( means=[3], stdevs=[2] )
4 h = HMM( mc, [g1, g2])
5 print("logprob:\n", h.logprob(obs))
```



```
logprob:
-9.187726979475208
```