Pattern Recognition Assignment 2: Speech Recognition

Hailey Yu Haihong
Levi Leyh

May 19, 2023

Contents

1 MarkovChain forward function.

1.1 Code implementation

This is the used code:

Listing 1: MarkovChain/forward function.

```python

    def forward(self, pX):
        """
        Calculates the forward probabilities for a given observation sequence.

        Parameters:
        pX: array-like
            The observation sequence (scaled).

        Returns:
        alpha: array-like, shape (T_obs, N_states)
            Forward variable.
        """
        T_obs = len(pX[0])
        N_states = len(self.q)

        alpha_temp = np.zeros((N_states, T_obs))
        c = np.zeros(T_obs)
        alpha = np.zeros((N_states, T_obs))

        # Initialization
        #alpha_temp[:, 0] = q * np.array([dist.pdf(obs[0]) for dist in B])
        alpha_temp[:, 0] = self.q * pX[:, 0]
        c[0] = sum(alpha_temp[:, 0])
        alpha[:, 0] = alpha_temp[:, 0]/(c[0])

        # Recursion
        for t in range(1, T_obs):
            #alpha_temp[:, t] = np.array([dist.pdf(obs[t]) for dist in B]) * (alpha[:, t - 1].dot(
                A)[:-1])
            alpha_temp[:, t] = pX[:, t] * (alpha[:, t - 1].T.dot(self.A)[:-1])
            c[t] = sum(alpha_temp[:, t])
            alpha[:, t] = alpha_temp[:, t]/(c[t])

        #termination
        if self.is_finite==True:
            c = np.append(c, (alpha[:, t].T.dot(self.A[:,-1])))
        return alpha, c
```

1.2 Test of the code

Using the test code below we have the following result

Listing 2: MarkovChain/forward function.

```python
#test code on forward algorithm
q = np.array([1, 0])
A = np.array([[0.9, 0.1, 0], [0, 0.9, 0.1]])
B = [norm(0, 1), norm(3, 2)]
obs = [-0.2, 2.6, 1.3]

mc = MarkovChain( np.array( [1, 0] ), np.array( [[0.9, 0.1, 0], [0, 0.9, 0.1]] ) )

pX = np.zeros((2, len(obs)))
#normalize
for m in range(len(obs)):
    scalar = np.max(np.array([norm.pdf(obs[m], 0, 1), norm.pdf(obs[m], 3, 2)]))
    pX[0, m] = norm.pdf(obs[m], 0, 1) / scalar
    pX[1, m] = norm.pdf(obs[m], 3, 2) / scalar

alpha, c = mc.forward(pX)
print("alpha_hat:\n", alpha)
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

```
from scipy.stats import norm

    def logprob(self, obs):
        """
        Calculates the log probability of the observation sequence given the HMM.

        Parameters:
        obs: array-like
            The observation sequence.

        Returns:
        log_prob: float
            The log probability of the observation sequence given the HMM.
        """

        pX = np.zeros((2, len(obs)))
        #normalize
        for m in range(len(obs)):
            scalar = np.max(np.array([norm.pdf(obs[m], 0, 1), norm.pdf(obs[m], 3, 2)]))
            #pX[0, m] = norm.pdf(obs[m], 0, 1) / scalar
            #pX[1, m] = norm.pdf(obs[m], 3, 2) / scalar
            for i in range(np.shape(self.stateGen.A)[0]):
                pX[i, m] = self.outputDistr[i].prob(obs[m])
        alpha, c = self.stateGen.forward(pX)
        log_likelihood = np.sum(np.log(c))
        return log_likelihood
```

2.2 Test of the code

Using the test code below we have the following result

Listing 4: MarkovChain/forward function.

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