EEEM030 Assignment 2

Speech Recognition Coursework

Yi Zhou 6567008

1. Introduction

Hidden Markov Model (HMM) framework is widely used in modelling time series data. In speech area, the Automatic Speech Recognition (ASR) systems use HMM to build the acoustic model. In this report, three basic tasks within HMM framework and their solutions are introduced. The simulation results of training a continuous-density hidden Markov model (CD-HMM) are shown and discussed.

2. Methods

2.1 Hidden Markov Model

Markov Model is a kind of Bayesian inference framework which is used to model a stochastic system. It assumes the current state is only affected by the previous state, i.e. the Markov assumption. Particularly, if the states are not observable, a Hidden Markov Model (HMM) framework is used to iteratively estimate the likelihood of the hidden state sequence. In this framework, some parameters[1] are defined as following

$$O = [o_0, o_1 \dots, o_{T-1}]$$
$$Q = [q_0, q_1 \dots q_{N-1}]$$
$$A = \begin{pmatrix} a_{11} & \cdots & a_{1(N-1)} \\ \vdots & \ddots & \vdots \\ a_{(N-1)1} & \cdots & a_{(N-1)(N-1)} \end{pmatrix}$$
$$B = b_i(o_t)$$
$$\pi = [\pi_0, \pi_1 \dots, \pi_{N-1}]$$
$$\eta = [\eta_0, \eta_1 \dots, \eta_{N-1}]$$

where Q is set of the hidden states with known length N. O is the observation sequence with length T. o_0 can be either discrete ($o_0 \in [v_0, v_1, \dots v_{k-1}]$) or continuous (drawn from N

[1] For the ease of corresponding to python codes, the index all start from zero.

distributions). A is the $(N-1)\times(N-1)$ transition matrix. Each its element a_{ij} represents the transition probability from state i to j. π and η represent the entry and exit probability respectively. B is the observation model, which can be either a $(N-1)\times(K-1)$ for discrete observations or N probability density functions for continuous observations. For a 1-D continuous univariate Gaussian case, the probability of {the observation o_t corresponds to the hidden state i} can be modelled as

$$b_i(o_t) = \frac{1}{\sqrt{2\pi}\sigma_i}\exp\frac{(x-\mu_i)^2}{2\sigma_i^2} \tag{2.1.1}$$

There are two assumptions of a first order Hidden Markov Model:

1) Markov assumption: the current state is only dependent on the previous state.

$$P(q_t|q_{T-1}, q_{T-2}\dots q_0) = P(q_t|q_{T-1}) \tag{2.1.2}$$

2) Output Independencies: the output probability of an observation is only dependent on the hidden state which produced the observation.

$$P(o_t|o_{T-1}, o_{T-2}\dots o_0, q_T, q_{T-1}\dots q_0) = P(o_t|q_t) \tag{2.1.3}$$

There are three fundamental tasks within HMM framework:

1) Estimate the likelihood of the observation sequence given the HMM model
2) Decoding the best hidden state path given the observation and the model
3) Learning model parameters given the observation sequence

In the following sections, the methods to solve these three tasks will be discussed.

2.2 Likelihood Estimation

2.2.1 Forward Algorithm

In order to estimate the likelihood of the observation sequence given the model, we can marginalize the hidden state Q, as equation 2.2.1 suggests. For every possible hidden state sequence, the transition probability and the observation probability are multiplied sequentially at each time.

$$\begin{aligned} P(O|\lambda) &= \sum_{all\ Q} P(O,Q|\lambda) = \sum_{all\ Q} P(O|Q,\lambda)P(Q|\lambda) \\ &= \sum_{all\ Q}\prod_{i=1}^{T}\pi P(o_i|q_i,\lambda)P(q_i|q_{i-1},\lambda)\eta \end{aligned} \tag{2.2.1}$$

In equation 2.2.1, the computation complexity is $O(N^T)$ which could be quite heavy if we train

a large amount of data. Furthermore, we can find there are many redundant computations since there are repetitive paths for each sequence. This inspire us to use a recursive method, given in equation 2.2.2, whose complexity is reduced to $O(TN^2)$. We define the forward likelihood $\alpha_t(i)$ as the likelihood of being start in state i after seeing the first t observations. It can be formulated as a $N \times T$ matrix and recursively computed from left to right.

$$\alpha_t(i) = \sum_{j=0}^{N-1} \alpha_{t-1}(j) a_{ji} b_i(o_t) \tag{2.2.2}$$

Finally, the likelihood of the observation sequence can be calculated according to equation 2.2.3.

$$P(O|\lambda) = \sum_{i=0}^{N-1} \alpha_{T-1}(i) \eta_i \tag{2.2.3}$$

2.2.2 Backward Algorithm

Alternatively, we can recursively update the likelihood in the reversed direction. The backward likelihood $\beta_t(i)$ is defined as the likelihood of being in state i at time t after seeing the observations from $t+1$ to the end, given the model parameters.

$$\beta_t(i) = \sum_{j=0}^{N-1} \beta_{t+1}(j) a_{ij} b_j(o_{t+1}) \tag{2.2.4}$$

Similarly, we can assembly β into a $N \times T$ matrix and recursively update its entries. Then the likelihood of the observation can be calculated using β_0 as

$$P(O|\lambda) = \sum_{i=0}^{N-1} \pi_i \beta_0(i) b_i(o_0) \tag{2.2.5}$$

2.3 Viterbi Algorithm

The task of estimating the most likely hidden state sequence path is called decoding. The Viterbi algorithm can be used to determine the underlying hidden states which give the maximum cumulative likelihood of the observations. As equation 2.3.1 shows, the process is very similar to that of the forward algorithm. The difference is Viterbi algorithm uses the max() instead sum() for the likelihood of the possible states. Finally, the likelihood of the best path is given by $P_{best} = \max_{i \in [0,N)} \nu_{T-1}(i) \eta_i$.

$$\nu_t(i) = \max_{j \in [0,N)} \nu_{t-1}(j) a_{ji} b_i(o_t) \; for \; i \in [0,N), t \in [1,T) \tag{2.3.1}$$

In the meantime, we need to record the backward path using a matrix P. $p_t(i)$ represents a pointer to the previous state at time $t-1$. When we loop to the end of the sequence, we can use this pointer to retrieve the best path.

$$p_t(i) = \underset{j\in[0,N)}{\operatorname{argmax}} v_{t-1}(j)a_{ji}b_i(o_t)\ for\ i \in [0,N), t \in [1,T) \tag{2.3.2}$$

2.4Training

To train the HMM model, we use the Baum-Welch method, which is a special case of the Expectation-Maximization(EM) algorithm. In E-step, the likelihoods are calculated using the model which is estimated in the previous iteration. In M-step, the model is re-estimated according to these likelihoods. Then, iterate this process until convergence. It should notice that such process cannot guarantee to converge to global minimum. Thus, the proper initial condition is always required.

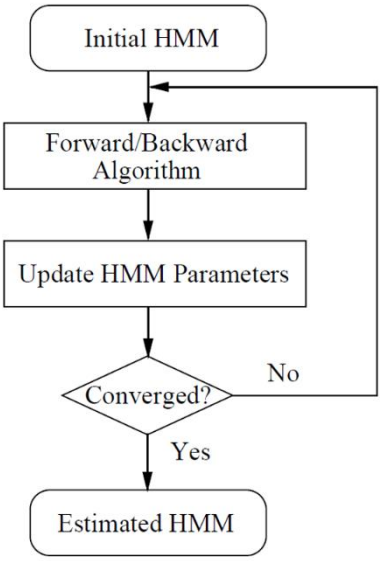


Figure 2.4.1 Process of BW training of HMM [1]

In order to update the transition probability, we need firstly estimate the transition likelihood $\xi_t(i,j)$, which interprets as the joint probability of being in state i at time t and being in state j at time $t+1$ given the observation sequence and model parameters. Since observations and transitions are independent, according to the law of joint probability, we can get the transition likelihood using equation 2.4.1.

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j|O,\lambda) = \frac{P(q_t = i, q_{t+1} = j, O|\lambda)}{P(O|\lambda)} \tag{2.4.1}$$

The joint likelihood of transition and observation can be calculated using the forward likelihood $\alpha_t(i)$ and the backward likelihood $\beta_{t+1}(j)$. The likelihood of the observation $P(O|\lambda)$ can be

either estimated by forward or backward method. Then, the $\xi_t(i,j)$ can be computed by

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}\beta_{t+1}(j)b_j(o_{t+1})}{P(O|\lambda)} \tag{2.4.2}$$

The transition probability $\hat{a}_{ij}$ can be estimated through dividing the sum of the transition likelihood from i to j over all sequence by the sum of the occupation likelihood for state i.

$$\hat{a}_{ij} = \frac{\sum_{t=0}^{T-1}\xi_t(i,j)}{\sum_{t=0}^{T-1}\gamma_t(i)} \tag{2.4.3}$$

Since every row of the transition matrix should sum up to 1, the output probability can be updated as

$$\hat{\eta}_i = 1 - \sum_{i=0}^{N-1}\hat{a}_{ij} \tag{2.4.4}$$

Next step, to estimate the output probability, we need to calculate the occupation likelihood , which is defined as the probability of being in state i at time t given the observation and model. Similarly, we use the law of joint probability to divide the equation into two parts, as shown in equation 2.4.5.

$$\gamma_t(i) = P(q_t = i|O,\lambda) = \frac{P(q_t = i, O|\lambda)}{P(O|\lambda)} \tag{2.4.5}$$

The numerator of equation 2.4.5 can be calculated using the forward and backward likelihood of state i at time t. The denominator is the same as in equation 2.4.2. Then the likelihood can be calculated by

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} \tag{2.4.6}$$

Specially, $\gamma_0(i)$ corresponds to the estimation of entry probability $\hat{\pi}_i$.

For the discrete observation model, the probability of observing v_k at state i can be calculated by

$$\hat{b}_i(v_k) = \frac{\sum_{t=0}^{T-1}\gamma_t(i)\delta(o_t - v_k)}{\sum_{t=0}^{T-1}\gamma_t(i)} \tag{2.4.6}$$

The denominator represents the sum of occurrence probability of state i when v_k is observed. The $\delta()$ is the Dirac function which returns 1 at 0 and 0 elsewhere. The denominator represents the sum of occurrence probability of state i over all the sequence.

For the continuous observation model , the output probability densities are modelled by Gaussian distributions. Then, the mean and variance are updated as

$$\hat{\mu}_i = \frac{\sum_{t=0}^{T-1} \gamma_t(i) o_t}{\sum_{t=0}^{T-1} \gamma_t(i)} \tag{2.4.7}$$

$$\hat{\Sigma}_i = \frac{\sum_{t=0}^{T-1} \gamma_t(i)(o_t - \mu_i)(o_t - \mu_i)^T}{\sum_{t=0}^{T-1} \gamma_t(i)} \tag{2.4.8}$$

Finally, following the process in figure 2.4.1, the model parameters are iteratively updated until convergence.

3. Result Analysis

3.1 Initial Values

In this coursework, the task is to train an HMM model for a 1D continuous observation sequence, whose values are given in table 3.1.1.

t	0	1	2	3	4	5	6
o_t	1.8	2.6	2.7	3.3	4.4	5.4	5.2

Table 3.1.1 The observation sequence

Table 3.1.2 gives the initial guess of the state transition probability matrix. Where the entry probabilities π_i are given by the first row and exit probabilities η are given by the last row. The four elements in the middle are the transition probabilities a_{ij} between states.

0	0.93	0.07	0
0	0.74	0.21	0.05
0	0.08	0.90	0.02
0	0	0	0

Table 3.1.2 The state transition matrix

According to the table above, a state topology plot is drawn as figure 3.1.1. The small gray circles denote the entry and exit points. The white circles lablled 1 and 2 denote the two discrete states. Each solid vector indicates a state transition, which bounded with a state trasition probability. The output models are denoted by two squares lablelled b1 and b2 respectively.

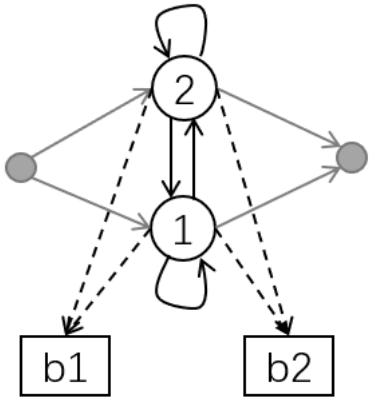


Figure 3.1.1 The state topology plot

The output density functions are modelled by two Gaussian density functions, whose parameters are given in table 3.1.3. In figure 3.1.2, the blue and orange curves depict the two density functions respectively. The observations and their corresponding probabilities are shown by the red and blue dots. We can find that the initial parameters are pretty good guesses. For the first four and the last two observations, we can clearly assign them into different models according to their probabilities. The ambiguity occurs at the fifth observation, where the probabilities are pretty close to each other.

State	1	2
Mean	3.00	5.00
Variance	1.21	0.25

Table 3.1.3 The initial values of the Gaussian density functions

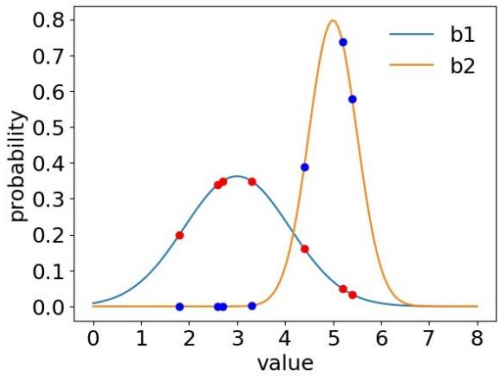


Figure 3.1.2 The output density functions

3.2 Likelihood Estimation

In this section, the likelihood of the observations is estimated using the given initial model parameters.

Figure 3.2.1 shows the estimated forward likelihood. It suggests the likelihood of being in state i at time t after seeing the observations before t.

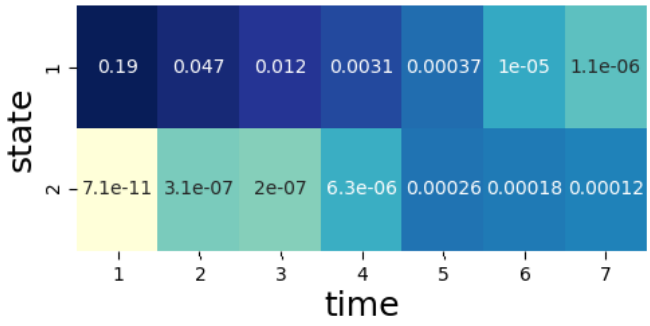


Figure 3.2.1 The forward likelihood

The likelihood of the observation sequence given by the forward method is

$$P_f(O|\lambda) = 2.462970481849691\text{e} - 06$$

Figure 3.2.2 shows the estimated backward likelihood. It suggests the likelihood of being in state i at time t after seeing all the observations after $t + 1$.

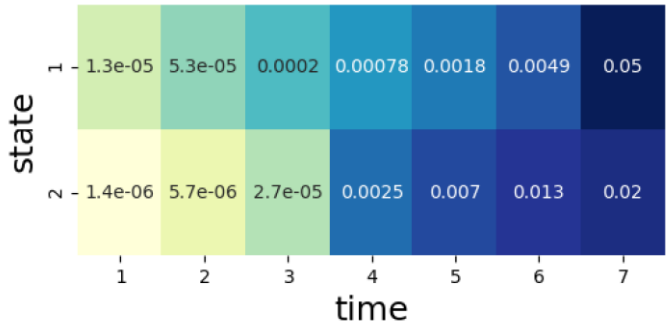


Figure 3.2.2 The backward likelihood

The observation likelihood calculated by the backward method is

$$P_b(O|\lambda) = 2.4629704818496913\text{e} - 06$$

We can find both two methods give the same result.

3.3 Training

In this section, the HMM model is trained following the process depicted in figure 2.4.1.

The occupation likelihood in the first iteration is shown in figure 3.3.1.

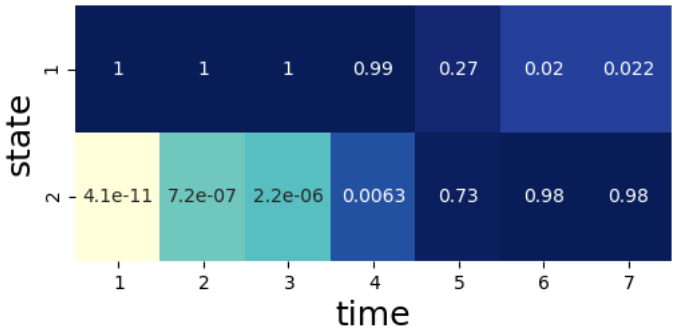
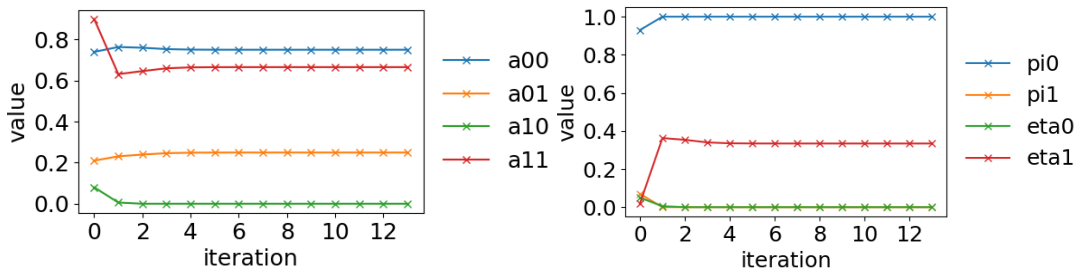


Figure 3.3.1 The occupation likelihood

The max iteration is set as 20 and a threshold of the sum of difference in mean is set as 1e-6. It takes 12 iterations to converge into the final solution. Figure 3.3.2 shows the convergence plot of some model parameters including the state transition probabilities, the entry probabilities and the exit probabilities. Since there are only one training sample, the entry probability of starting from state 0 converges to 1. Because there only a few real transitions begin from state 1, the output probability of ending in state 1 converges to about 0.3. It can be concluded that the model tends to overfit the training samples if the number of samples is not large enough.



(a) (b)

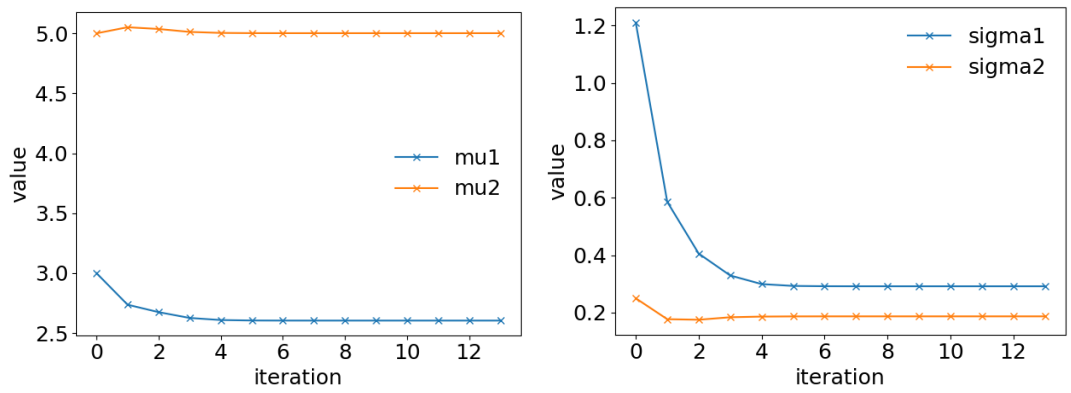
Figure 3.3.2 The convergence of the model parameters:

(a) The state transition probabilities (b) The entry probabilities and the exit probabilities

The means and variances of the final observation models are given in the table below. In figure 3.3.3, we can find the initial values are quite close to the optimized results, so that the model parameters converge very quickly. The shape of the observation model $o_1(v_k)$ does not change a lot, while the observation model $o_2(v_k)$ adjusts its mean to left and reduce its spread to better fit the observation sequences. Figure 3.3.4 (a) better illustrate the adjustment process graphically.

State	1	2
Mean	2.60400091	5.00124716
Variance	0.29181143	0.18711869

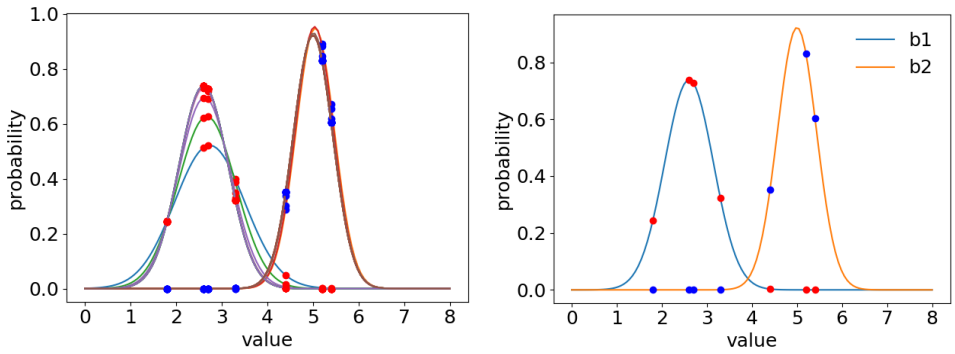
Table 3.3.1 The parameters of the estimated output model



(a) (b)

Figure 3.3.3 The convergence of the model parameters:
(a)The means and (b) the variances of the Gaussian output probabilities

Figure 3.3.4 (b) shows the curves of the converged output probability density functions. The dots represent the observation sequences. Compared to the initial distributions, the fourth observation now have a relatively high probability in the Gaussian curve of state 1, while a very low probability in the distribution of state 2. All the observations can be clearly assigned one of the hidden states. We can infer that the most possible path should be [0, 0, 0, 0, 1, 1, 1].



(a) (b)

Figure 3.3.4 The distribution of the Gaussian output models:
(a)the whole process and (b) the best fit

Finally, we use the Viterbi method to retrieve the best state transition path. As illustrated in figure 3.3.5, the best path is [0, 0, 0, 0, 1, 1, 1], which is consistent with the inference from the Gaussian curves. The best path likelihood is 0.00011576329920463527.

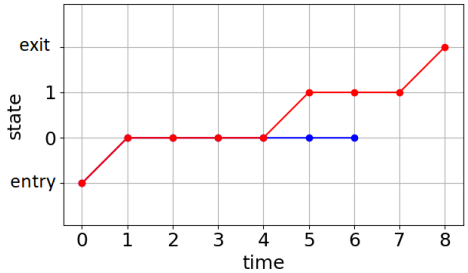


Figure 3.3.5 The best state transition path (in red)

4. Conclusion

In this report, a continuous-density hidden Markov model is trained using the Baum-Welch methods. The simulation results show a fast and good convergence. It can be concluded that the Hidden Markov Model is a strong tool in modelling time series data with hidden states.

Reference

[1] S. J. Young, et al.,The HTK Book, Cambridge Univ. Eng. Dept.(v3.2), 2002 [http://htk.eng.cam.ac.uk/].
[2] Jurafsky, D. and Martin, J. (2014). Speech and language processing. [India]: Dorling Kindersley Pvt, Ltd.
[3] Jackson, P. (2019). *Speech & Audio Processing & Recognition*.

Appendix

Forward Likelihood Data

Time	Alpha 1	Alpha 2
0	1.860275071764466714e-01	7.123396394303055561e-11
1	4.673180782619031776e-02	3.095024504436310764e-07
2	1.208399524646727036e-02	1.990437964790388943e-07
3	3.124704636602149317e-03	6.254296319308691286e-06
4	3.731722995743262943e-04	2.570312027072975750e-04
5	9.957651209727423329e-06	1.794316252195002196e-04
6	1.066232842837537257e-06	1.204829419853906938e-04

Backward Likelihood Data

Time	Beta 1	Beta 2
0	1.323981877267003840e-05	1.431371364767285705e-06
1	5.270433207692120704e-05	5.698253078765066816e-06
2	2.038204235212215214e-04	2.739540945503825920e-05
3	7.832621964722463054e-04	2.479489948342161396e-03
4	1.758887042986387881e-03	7.028728575745334800e-03
5	4.909528961264239891e-03	1.345405583824032386e-02
6	5.000000000000000278e-02	2.000000000000000042e-02

Occupancy Probability Data

1.00000000e+00	9.99999284e-01	9.99997786e-01	9.93703755e-01	2.66494433e-01	1.98489496e-02	2.16452623e-02
4.13980829e-11	7.16055391e-07	2.21394708e-06	6.29624471e-03	7.33505567e-01	9.80151050e-01	9.78354738e-01

$(ProjectPath)/model.py

```
import numpy as np
from estimate import ContinuousHMM

p = np.array([0.93, 0.07])
a = np.array([[0.74, 0.21], [0.08, 0.9]])
o = np.array([1.8, 2.6, 2.7, 3.3, 4.4, 5.4, 5.2])
e = np.array([0.05, 0.02])
mu = np.array([3.0, 5.0])
sigma = np.array([1.21, 0.25])
c = ContinuousHMM(p, e, a, mu, sigma, o)

if __name__ == "__main__":
prob_forward = c.getForwardProb()
prob_backward = c.getBackwardProb()
    c.doTraining()
best_path, best_path_prob = c.getDecodingPath()
```

$(ProjectPath)/HMM/estimate.py

```
import numpy as np


def norm(mu, sigma, x):
    return 1.0 / np.sqrt(2 * np.pi * sigma) * np.exp(-1.0 / 2 * (x - mu) **
2 / sigma)


class ContinuousHMM:
```

```
    def __init__(self, entry_prob, exit_prob, a, mu, sigma, observation,
max_iter=20 , threshold =1e-08):
        self.entry_prob = np.asarray(entry_prob)
        self.exit_prob = np.asarray(exit_prob)
        self.a = np.asarray(a)
        self.mu = np.asarray(mu)
        self.sigma = np.asarray(sigma)
        self.observation = np.asarray(observation)
        self.N = a.shape[0]
        self.T = observation.shape[0]
        self.forward = np.zeros([self.N, self.T])
        self.backward = np.zeros([self.N, self.T])
        self.max_iter = max_iter
        self.threshold = threshold

    def getForwardProb(self):
        forward = np.zeros([self.N, self.T])
        for i in range(self.N):
            forward[i, 0] = self.entry_prob[i] * norm(self.mu[i],
self.sigma[i], self.observation[0])
        for t in range(1, self.T):
            for i in range(self.N):
                for j in range(self.N):
                    forward[i, t] += forward[j, t - 1] * self.a[j, i] \
                                     * norm(self.mu[i], self.sigma[i],
self.observation[t])
        forward_prob = sum(forward[:, -1] * self.exit_prob)
        self.forward = forward
        return forward_prob

    def getBackwardProb(self):
        backward = np.zeros([self.N, self.T])
        for i in range(self.N):
            backward[i, -1] = self.exit_prob[i]
        for t in range(self.T - 2, -1, -1):
            for i in range(self.N):
                for j in range(self.N):
                    backward[i, t] += backward[j, t + 1] * self.a[i, j] \
                                      * norm(self.mu[j], self.sigma[j],
self.observation[t + 1])

        backward_prob = 0
        for i in range(self.N):
            backward_prob += self.entry_prob[i] * backward[i, 0] \
```

```
                            * norm(self.mu[i], self.sigma[i],
self.observation[0])
        self.backward = backward
        return backward_prob

    def getDecodingPath(self):
        viterbi = np.zeros([self.N, self.T])
        back_pointer = np.zeros([self.N, self.T], dtype=int)
        best_path = np.zeros(self.T, dtype=int)
        for i in range(self.N):
            viterbi[i, 0] = self.entry_prob[i] * norm(self.mu[i],
self.sigma[i], self.observation[0])
        for t in range(1, self.T):
            for i in range(self.N):
                state_list = []
                for j in range(self.N):
                    state_list.append(viterbi[j, t - 1] * self.a[j, i])
                back_pointer[i, t] = np.argmax(state_list)
                viterbi[i, t] = viterbi[back_pointer[i, t], t - 1] *
self.a[back_pointer[i, t], i] \
                                * norm(self.mu[i], self.sigma[i],
self.observation[t])
        best_path_pointer = np.argmax(viterbi[:, -1] * self.exit_prob)
        best_path_prob = viterbi[best_path_pointer, -1] *
self.exit_prob[best_path_pointer]
        best_path[-1] = best_path_pointer
        for t in range(self.T - 1, 0, -1):
            best_path[t - 1] = back_pointer[best_path[t], t]
        return best_path, best_path_prob, back_pointer

    def doTraining(self):
        for ind in range(self.max_iter):
            norm1 = self.getForwardProb()
            norm2 = self.getBackwardProb()
            ksi = np.zeros([self.N, self.N, self.T - 1])
            gamma = np.zeros([self.N, self.T])
            for t in range(0, self.T - 1):
                for i in range(self.N):
                    for j in range(self.N):
                        ksi[i, j, t] = self.forward[i, t] * self.a[i, j] \
                                       * norm(self.mu[j], self.sigma[j],
self.observation[t + 1]) \
                                       * self.backward[j, t + 1] / norm1
            for t in range(self.T):
```

```
                for i in range(self.N):
                    gamma[i, t] = self.forward[i, t] * self.backward[i, t] /
norm1
            for i in range(self.N):
                for j in range(self.N):
                    #self.a[i, j] = sum(ksi[i, j, :]) /
sum(sum(ksi[i, :, :]))
                    self.a[i, j] = sum(ksi[i, j, :]) / sum(gamma[i, :])
                self.exit_prob[i] = 1 - sum(self.a[i, :])
            mu_old = np.copy(self.mu)
            for i in range(self.N):
                self.mu[i] = sum(gamma[i, :] * self.observation) /
sum(gamma[i, :])
                self.sigma[i] = sum(gamma[i, :] * (self.observation -
mu_old[i]) ** 2) / sum(gamma[i, :])
            self.entry_prob = gamma[0, :]
            print(self.mu)
            print(self.sigma)
            if sum(mu_old - self.mu) < self.threshold
:
                print(ind)
                break
        return data1, data2, data3, data4
```