EQ2341 Pattern Recognition and Machine Learning

Assignment 1: HMM Signal Source

Hanqi Yang
hanqi@kth.se

Qian Zhou
qianzho@kth.se

April 21, 2022

Question

1. To verify your Markov chain code, calculate $P(S_t = j), j \in 1, 2$ for $t = 1, 2, 3, \ldots$ theoretically, by hand, to verify that $P(S_t = j)$ is actually constant for all t.

The infinite-duration HMM has parameters as

$$q = \begin{pmatrix} 0.75 \\ 0.25 \end{pmatrix} A = \begin{pmatrix} 0.99 & 0.01 \\ 0.03 & 0.97 \end{pmatrix} B = \begin{pmatrix} b_1(x) \\ b_2(x) \end{pmatrix}$$

The state probabilities at time $t+1$ are determined by the state probabilities at t, in combination with the transition probabilities, as

$$P[S_{t+1} = j] = \sum_{i=1}^{N} P[S_{t+1} = j | S_t = i] P[S_t = i] \tag{1}$$

Therefore, in the initial state when $t = 1$, the probability distribution is

$$P(S_1 = 1) = q(1) = 0.75 P(S_1 = 2) = q(2) = 0.25$$

When $t = 2$, the probability distribution is

$$\begin{aligned} P[S_2 = 1] &= \sum_{i=1}^{2} P[S_2 = 1 | S_1 = i] P[S_1 = i] \\ &= P[S_2 = 1 | S_1 = 1] P[S_1 = 1] + P[S_2 = 1 | S_1 = 2] P[S_1 = 2] \\ &= 0.99 \times 0.75 + 0.03 \times 0.25 = 0.75 \end{aligned}$$

$$\begin{aligned} P[S_2 = 2] &= \sum_{i=1}^{2} P[S_2 = 2 | S_1 = i] P[S_1 = i] \\ &= P[S_2 = 2 | S_1 = 1] P[S_1 = 1] + P[S_2 = 2 | S_1 = 2] P[S_1 = 2] \\ &= 0.01 \times 0.75 + 0.97 \times 0.25 = 0.25 \end{aligned}$$

We can abtain that when the state from $t = 1$ to $t = 2$, $P[S_t = j]$ remains the same. So from $t = n$ to $t = n + 1$

$$\begin{aligned} P[S_{n+1} = 1] &= \sum_{i=1}^{2} P[S_{n+1} = 1 | S_n = i] P[S_n = i] \\ &= P[S_{n+1} = 1 | S_n = 1] P[S_n = 1] + P[S_{n+1} = 1 | S_n = 2] P[S_n = 2] \\ &= 0.99 \times 0.75 + 0.03 \times 0.25 = 0.75 \end{aligned}$$

$$\begin{aligned} P[S_{n+1}=2] &= \sum_{i=1}^{2} P[S_{n+1}=2|S_n=i]P[S_n=i] \\ &= P[S_{n+1}=2|S_n=1]P[S_n=1]+P[S_{n+1}=2|S_n=2]P[S_n=2] \\ &= 0.01\times 0.75+0.97\times 0.25=0.25 \end{aligned}$$

Therefore, we can verify that $P(S_t=j)$ is actually constant for all t.

2. Use your Markov chain rand function to generate a sequence of $T=10000$ state integer numbers from the test Markov chain. Calculate the relative frequency of occurrences of $S_t=1$ and $S_t=2$. The relative frequencies should of course be approximately equal to $P(S_t)$.

We use Markov chain rand function here to generate a sequence of $T=10000$ state integer numbers and count the number of occurrences for each state. The empirical results for a single sequence and the mean of 300 different generated sequence are shown in Table 1. As we know, the theoretical calculation results are approximately 0.75 for $S_t=1$ and 0.25 for $S_t=2$, so our Markov chain rand function is quite successful.

Table 1: Empirical results with $t=10000$

State	**1 run**	**300 runs**	**Relative Frequency for 300 runs**
$S_t=1$	7358	7452.87	74.52%
$S_t=2$	2642	2548.13	25.48%

3. To verify your HMM rand method, first calculate $E[X_t]$ and $var[X_t]$ theoretically. The conditional expectation formulas $\mu_X=E[X]=E_Z[E_X[X|Z]]$ and $var[X]=E_Z[var_X[X|Z]]+var_Z[E_X[X|Z]]$ apply generally whenever some variable X depends on another variable Z and may be useful for the calculations. Then use your HMM rand function to generate a sequence of $T=10000$ output scalar random numbers $x=(x_1\ldots x_t\ldots x_T)$ from the given HMM test example. Use the standard Numpy functions *np.mean*() and *np.var*() to calculate the mean and variance of your generated sequence. The result should agree approximately with your theoretical values.

According to the question, b_1 $N(0,1)$ and b_2 $N(3,4)$.

$$\begin{aligned} E[X] &= E_Z[E_X[X|Z]] \\ &= \omega_1 E_X[X|Z=1]+\omega_2 E_X[X|Z=2] \\ &= \omega_1\mu_1+\omega_2\mu_2 \\ &= 0.75\times 0+0.25\times 3=0.75 \end{aligned} \tag{2}$$

$$\begin{aligned} var[X] &= E_Z[var_X[X|Z]]+var_Z[E_X[X|Z]] \\ &= \omega_1 var_X[X|Z=1]+\omega_2 var_X[X|Z=2] \\ &+ \omega_1(E_X[X|Z=1]-\mu)^2+\omega_2(E_X[X|Z=2]-\mu)^2 \\ &= \omega_1\sigma_1^2+\omega_2\sigma_2^2+\omega_1(\mu_1-\mu)^2+\omega_2(\mu_2-\mu)^2 \\ &= 0.75\times 1+0.25\times 4+0.75\times(0.75-0)^2-0.25\times(0.75-3)^2 \\ &= 3.4375 \end{aligned} \tag{3}$$

The mean and variance of our generated sequence are shown in Table 2. Then we compare the empirical results with the theoretical results and we can find that the results agree with the theoretical values.

Table 2: Empirical results with $t = 10000$

Metric	**1 run**	**300 runs**
Mean	0.7703	0.7536
Variance	3.4088	3.4374

4. To get an impression of how the HMM behaves, use @HMM/rand to generate a series of 500 contiguous samples X_t from the HMM, and plot them as a function of t. Do this many times until you have a good idea of what characterizes typical output of this HMM, and what structure there is to the randomness. Describe the behaviour in one or two sentences in your report. Also include one such plot in the report, labelled using title, xlabel, and ylabel to clearly show which variable is plotted along which axis. You should do this for every plot in the course project.

The series of 500 contiguous samples X_t are plotted in Figure 1. It can be seen that X_t have two states: one is around 0 with small variation, corresponding to state 1; another is around 3 with larger variation, corresponding to state 2. In addition, the amount of times the state changes is much lower than the number of times it remains the same.

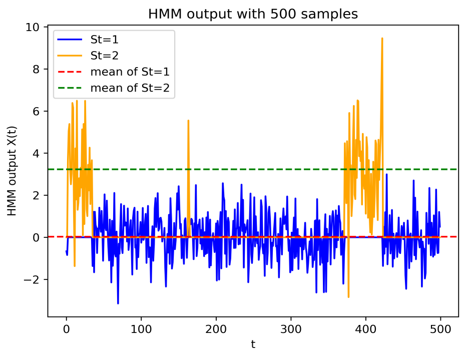

Figure 1: HMM behaviors with $\mu_1 = 0$, $\mu_2 = 3$

5. Create a new HMM, identical to the previous one except that it has $\mu_2 = \mu_1 = 0$. Generate and plot 500 contiguous values several times using @HMM/rand for this HMM. What is similar about how the two HMMs behave? What is different with this new HMM? Is it possible to estimate the state sequence S of the underlying Markov chain from the observed output variables x in this case?

The series of 500 contiguous samples X_t with $\mu_1 = \mu_2 = 0$, are plotted in Figure 2. Since both states have the same zero-mean, it is difficult to determine if an observation belongs to the first or the second state, especially at the moment the state is about to or just change. But with the original distributions in 1.4, it was much easier to differentiate them. However, because they have different variances where the second distribution has a higher value than the first, a more volatile state is more likely to be state 2.

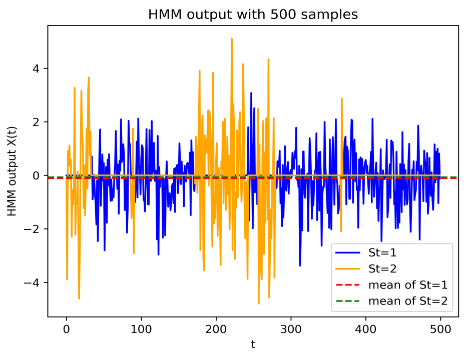

Figure 2: HMM behaviors with $\mu_1 = \mu_2 = 0$

6. Another aspect you must check is that your rand-function works for finite-duration HMMs. Define a new test HMM of your own and verify that your function returns reasonable results.

To test our rand function also works for finite-duration HMMs, we modify the initial transition matrix by adding a third column which represents the $S_t = n + 1$ state, as:

$$A = \begin{pmatrix} 0.4 & 0.5 & 0.1 \\ 0.7 & 0.3 & 0.1 \end{pmatrix}$$

As we can see in matrix A, the exit probability is 0.1. In our test, we can see the effect of a finite-duration HMM on the length of the generated sequence since the length of sequence can range from 1 to 67. After taking average of the results of 1000 calculations, we get that the expected sequence length is about 10, which is 1/0.1=10 (exactly related to exit probability), so our results are reasonable.

7. Finally, your rand function should work also when the state-conditional output distributions generate random vectors. Define a new test HMM of your own where the outputs are Gaussian vector distributions, and verify that this also works with your code. (Note that a single instance of the GaussD class is capable of generating vector output; stacking several GaussD-objects is not correct.)

To test our rand function also works when the state-conditional output distributions generate random vectors. we change the output probability distributions for 2D gaussians. Here we design the mean of vector 1 is $\mu_{11} = 0$, $\mu_{12} = 1$ and the variance is $\sigma_{11}^2 = 1$, $\sigma_{11}^2 = 4$. For vector 2 the mean is $\mu_{21} = 1$, $\mu_{22} = 2$, and the variance is $\sigma_{21}^2 = 4$, $\sigma_{21}^2 = 16$. Then we calculate the variance matrix and get the result:

$$C = \begin{pmatrix} 1.06921 & -1.23171 \\ -1.23171 & 1.41891 \end{pmatrix}$$

Therefore, we can verify that when the outputs are Gaussian vector distributions, our code works.