Royal Institute of Technology
DD2380 - Artificial Intelligence
Marina Herrera Sarrias

06/05/2020

1 A2: Hidden Markov Models

Let X_t represent the Markov Model hidden state at time t and N the number of hidden states in the model where $X_t = x_i$ for $i = 1, 2, ...N$. Let $\mathcal{O}_t$ be the observation at time step t and K the number of events where $\mathcal{O}_t = o_k$ for $k = 1, 2, ..., K$.

Following the assignment coin model problem, in this case there are only two possible hidden states (C_1, C_2), meaning that the hidden states X_t, could either take the state "*coin 1*" or "*coin 2*" .Note that as one of the main Hidden Markov Models properties, at each time step we will not know which coin was used, so the state is hidden from us. Additionally, the observations $\mathcal{O}_t$ are the observed outcomes which could be either "*Head*" or "*Tail*" and so $\mathcal{O}_{1:t}$ will be a vector of all observations up to time t.

Question 1: This problem can be formulated in matrix form. Please specify the initial probability vector π, the transition probability matrix **A** and the observation probability matrix **B**.

In order to define a Markov model, the following probabilities need to be specified: (i) vector of initial probabilities, (ii) matrix of transition probabilities (iii) matrix of observation probabilities. The Markov model can be specified with the parameter set: $\lambda = (\mathbf{A}, \mathbf{B}, \boldsymbol{\pi})$.

(i) **The initial-state probability distribution:**

$$\boldsymbol{\pi} = P(X_1 = C_i) = [P(C_1), P(C_2)] = [0.5, 0.5]$$

(ii) **The matrix of state-transition probabilities:** holds the probabilities of transitioning from one hidden state to another hidden state. In this case, the probability of staying in a given state (C_1, C_2) or transitioning. These probabilities are independent of the passage of time.

$$\mathbf{A} = \begin{bmatrix} P(C_1|C_1) & P(C_2|C_1) \\ P(C_1|C_2) & P(C_2|C_2) \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}$$

(ii) **The matrix of observed probabilities:** Contains the probabilities of observing a particular measurement (*head*, *tail*) given that the hidden model is in a particular hidden state (C_1, C_2). Notice that each row represents a probability distribution.

$$\mathbf{B} = \begin{bmatrix} P(H|C_1) & P(T|C_1) \\ P(H|C_2) & P(T|C_2) \end{bmatrix} = \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix}$$

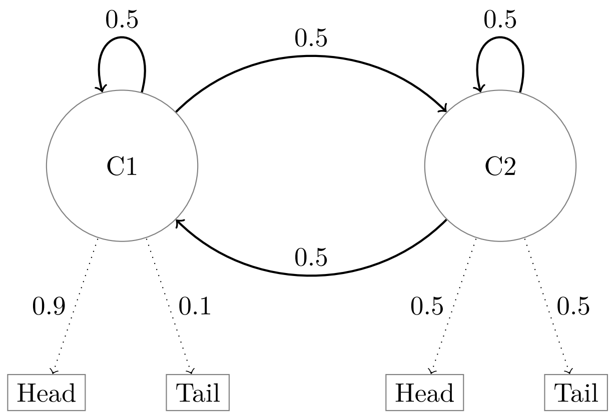

Figure 1: Hidden Markov Model

1.1 HMM 0 - Next observation distribution

We would like to determine $P(\mathcal{O}_t)$ for time step t. We know that $\mathcal{O}_t$ can only take two values $(head, tail)$. Hence, the result will be a vector of length two, representing a probability distribution for the observed outcome landing in either *head* or *tail*. The probability of selecting *coin 1* or *coin 2* will have equal probability 0.5 and this probability will remain constant at all time steps.

For computing the probability of observing each observation, i.e $P(\mathcal{O}_t = head)$ and $P(\mathcal{O}_t = tail)$, we will first need to multiply our current estimate of states $\boldsymbol{\pi}$ by the transition matrix $\mathbf{A}$.

Question 2: What is the result of this operation?

$$\boldsymbol{\pi} \cdot \mathbf{A} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}$$

In the following, the result must be multiplied with the observation matrix.

Question 3: What is the result of this operation?

$$\boldsymbol{\pi} \cdot \mathbf{A} \cdot \mathbf{B} = \begin{bmatrix} 0.5 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.9 & 0.1 \\ 0.5 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.7 & 0.3 \end{bmatrix}$$

The probability of observing a *head* is much higher than the probability of observing a *tail*.

1.2 HMM 1 - Probability of the observation sequence

This subsection covers one of the main uses of Hidden Markov Models, which is **evaluation**. Given a model $\lambda = (A, B, \pi)$ and a sequence of observation $\mathcal{O}$, we want to find $P(\mathcal{O}_{1:t}|\lambda)$. i.e. determine the likelihood for the observed sequence $\mathcal{O}$ with respect to a given model λ.

For computing this likelihood we will do it recursively by using the **Forward Algorithm** or also known as the **$\boldsymbol{\alpha}$-pass Algorithm**.

The **Forward Algorithm** iteratively estimates the probability to be in a certain state i at time t and having observed the observation sequence up to time t (for $t = 1, 2, ...T$). The matrix $\boldsymbol{\alpha}$ is of dimension T x N, where the rows represent each time step while the columns the state (*coin 1*, *coin 2*).

- We first initialize $\alpha_1(i)$ applying the product rule.

$$\alpha_1(i) = P(O_1 = o_1, X_1 = x_i) = P(O_1 = o_1|X_1 = x_i) \cdot P(X_1 = x_i) = b_i(O_1)\pi_i$$

- Next, we estimate the probabilities of the remaining $T - 1$ time steps as:

$$\alpha_t(i) = P(O_{1:t} = o_{1:t}, X_t = x_i)$$

Question 4: Why is it valid to substitute $\mathbf{O}_{1:t} = \mathbf{o}_{1:t}$ with $\mathbf{O}_t = \mathbf{o}_t$ when we condition on the state $\mathbf{X}_t = x_i$?

For computing $P(O_{1:t} = o_{1:t}, X_t = x_i)$ we will need to marginalize over all possible state sequences in $\{X_{1:t-1}\}$, this set will grow exponentially as t increases. This is where the α-pass algorithm takes advantage of the conditional independence property of the Hidden Markov Models.

$$\alpha_t(i) = \sum_{j=1}^{N} P(O_{1:t} = o_{1:t}, X_t = x_i, X_{t-1} = x_j)$$

By using the chain rule for expanding $P(O_{1:t}, X_t, X_{t-1})$ we obtain:

$$\begin{aligned} \alpha_t(i) = \sum_{j=1}^{N} & P(O_t = o_t|X_t = x_i, X_{t-1} = x_j, O_{1:t-1} = o_{1:t-1}) \\ & P(X_t = x_i|X_{t-1} = x_j, O_{1:t-1} = o_{1:t-1}) \\ & P(O_{1:t-1} = o_{1:t-1}, X_{t-1} = x_j) \end{aligned}$$

Now, we know that O_t is conditionally independent of previous observations and states, given X_t and that the current state X_t is conditionally independent of previous observations given the previous state. We can rewrite the probabilities estimate as:

$$\alpha_t(i) = P(O_t = o_t | X_t = x_i) \sum_{j=1}^{N} P(X_t = x_i | X_{t-1} = x_j)$$
$$P(O_{1:t-1} = o_{1:t-1}, X_{t-1} = x_j)$$

Finally, note that $P(O_t = o_t | X_t = x_i)$ will refer to the observation probability $b_i(o_t)$, $P(X_t = x_i | X_{t-1} = x_j)$ will refer to the transition probability from state j to state i $(a_{j,i})$ in the state-transition matrix and
$\alpha_{t-1}(j) = P(O_{1:t-1} = o_{1:t-1}, X_{t-1} = x_j)$.

$$\alpha_t(i) = b_i(o_t) \Big[\sum_{j=1}^{N} a_{j,i} \alpha_{t-1}(j) \Big]$$

Now, going back to our initial purpose of estimating the probability of having observe the observation sequence $O_{1:T}$, we obtain this by marginalizing over the hidden states distribution such that :

$$\begin{aligned} P(O_{1:T} = o_{1:T}) &= \sum_{j=1}^{N} P(O_{1:T} = o_{1:T}, X_T = x_j) \\ &= \sum_{j=1}^{N} \alpha_T(j) \end{aligned} \tag{1}$$

1.3 HMM 2 - Estimate sequence of states

This subsection covers another of the main uses of Hidden Markov Models, which is **decoding**. Given a model $\lambda = (A, B, \pi)$ and a sequence of observation $\mathcal{O}_{1:T}$, we want to find the most likely state sequence $X^*_{1:T}$ for the underlying Markov Process, where $X^*_{1:T} = (X^*_1, X^*_2, ..., X^*_T)$. We want to uncover the hidden part of the HMM.

As a common approach we will use the **Viterbi Algorithm** which is a Dynamic programming Algorithm for finding the most likely sequence of hidden states. The algorithm is structured similarly as the α-pass algorithm, with the only difference that instead of performing a sum, we will now take the maximum between all possible states.

At each time step t the Viterbi Algorithm determines the probability of the best path ending at each of the states (*coin 1* or *coin 2*). Mind that a path with optimal probability does not mean that is the optimal path. In order to find the optimal path we will need to keep track of each preceding state i.e, we trace back from the highest-scoring final state.

Question 5: How many values are stored in the matrices δ and δ^{idx} respectively?

- The matrix δ is a T x N matrix where

$$\delta_t(i) = \max_{j \in \{1,...,N\}} a_{j,i}\delta_{t-1}(j)b_i(o_t)$$

. Hence, we will only keep one value per time step corresponding to the maximum score.

- The matrix δ^{idx} is a T x N matrix which stores the state-index of the maximum $\delta_t(i)$ for $i = 1, 2, ..., N$. i.e if the algorithm determined that $\delta_t(i)$ for state i at time t to have been preceded by state k, then $\delta_t^{idx}(i) = k$.

$$\delta_t^{idx}(i) = \text{argmax}_{j \in \{1,...,N\}} a_{j,i}\delta_{t-1}(j)b_i(o_t)$$

.

Hence there will only be one value of interest per time step, corresponding to the state-index of the maximum score in $\delta_t(i)$, for $i = 1, 2, ..., N$.

1.4 HMM 3 - Estimate Model Parameters

This subsection covers another of the main uses of Hidden Markov Models, which is **learning**. Given an observation sequence $\mathcal{O}_{1:t}$ we want to estimate the set of model parameters $\lambda = (A, B, \pi)$ such that $P(\mathcal{O}_{1:t}|\lambda)$ is maximized.

We will try to approximate the initial-state probability distribution, state-transition and observed matrices by using the **Baum–Welch algorithm**. The Baum–Welch algorithm is a special case of the *Expectation-Maximization algorithm* which is used for finding the maximum-likelihood estimate of the HMM parameters given a set of observations $\mathcal{O}_{1:t}$. The Baum–Welch algorithm uses the Backward-forward algorithms (α and β-pass algorithms) to compute the statistics α and β in the E-step.

1.4.1 E-Step:

- **$\boldsymbol{\alpha}$-pass algorithm**: We need to determine the joint probability of each state i at time t and all observations up to time t, for $i = 1, 2, ..., N$ and $t = 1, 2, ..., T$.

$$\alpha_t(i) = P(O_{1:t} = o_{1:t}, X_t = x_i)$$

- **β-pass algorithm**: We need to determine the probability of observing all future observations $\mathcal{O}_{t+1:T}$ given the current state $X_t = x_i$, i.e assess the probability of a state at time t and all future observations.

$$\beta_t(i) = P(O_{t+1:T} = o_{t+1:T} | X_t = x_i)$$

Temporary variables:

- The **di-gamma** function gives us the probability of being in state i at time t and in state j at time $t+1$ given an observations sequence. We can obtain it by applying the *Bayes Theorem.*

$$\begin{aligned}
\gamma_t(i,j) &= P(X_t = x_i, X_{t+1} = x_j | O_{1:T} = o_{1:T}) \\
&= \frac{P(X_t = x_i, X_{t+1} = x_j, O_{1:T} = o_{1:T})}{P(O_{1:T} = o_{1:T})} \\
&= \frac{\alpha_t(i) a_{i,j} b_j(O_{t+1}) \beta_{t+1}(j)}{\sum_{j=1}^{N} \alpha_T(j)}
\end{aligned}$$

- The **gamma** function gives us the probability of being in state i at time t given the observed sequence.

$$\begin{aligned}
\gamma_t(i) &= P(X_t = x_i | O_{1:T} = o_{1:T}) \\
&= \frac{P(X_t = x_i, O_{1:T} = o_{1:T})}{P(O_{1:T} = o_{1:T})} \\
&= \frac{P(\alpha_t(i)\beta_t(i))}{\sum_{j=1}^{N} \alpha_T(j)}
\end{aligned}$$

We can easily obtain the gamma function by marginalizing $\gamma_t(i,j)$ over j:

$$\gamma_t(i) = \sum_{j=1}^{N} \gamma_t(i,j)$$

Question 6: Why do we need to divide by the sum over the final α values for the di-gamma function?

The denominator of the di-gamma function refers to the ”evidence” term, given by the Bayes Theorem, it is given by $P(O_{1:T} = o_{1:T})$ which is the probability estimate of having observed the sequence $O_{1:T}$ (under the model parameter set λ) as in Equation (1).

1.5 Empirical Investigation

We were provided with the files `hmm_c_N1000.in` and `hmm_c_N10000.in` containing a data sequence ($T = 1000, 10000$) generated form:

$$\mathbf{A} = \begin{bmatrix} 0.7 & 0.05 & 0.25 \\ 0.1 & 0.8 & 0.1 \\ 0.2 & 0.3 & 0.5 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.7 & 0.2 & 0.1 & 0 \\ 0.1 & 0.4 & 0.3 & 0.2 \\ 0 & 0.1 & 0.2 & 0.7 \end{bmatrix} \boldsymbol{\pi} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \tag{2}$$

Question 7: Train an HMM with the same parameter dimensions as above, i.e. **A** should be a 3 times 3 matrix, etc. Initialize your algorithm with the following matrices:

$$\mathbf{A} = \begin{bmatrix} 0.54 & 0.26 & 0.20 \\ 0.19 & 0.53 & 0.28 \\ 0.22 & 0.18 & 0.60 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.5 & 0.2 & 0.11 & 0.19 \\ 0.22 & 0.28 & 0.23 & 0.27 \\ 0.19 & 0.21 & 0.15 & 0.45 \end{bmatrix}$$
$$\boldsymbol{\pi} = \begin{bmatrix} 0.3 & 0.2 & 0.5 \end{bmatrix}$$

The estimated Euclidean distance of the true set of parameters **A**, **B**, and $\boldsymbol{\pi}$ in Eq. (2) from the initialization matrices is of 0.25431, 0.2696548 and 0.8831761 respectively.

- When $T = 1000$ the algorithm took **2040** iterations to converge to the following parameter estimates:

$$\mathbf{A} = \begin{bmatrix} 0.70 & 0.01 & 0.29 \\ 0.10 & 0.81 & 0.09 \\ 0.19 & 0.30 & 0.51 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.69 & 0.22 & 0.08 & 0.01 \\ 0.07 & 0.41 & 0.28 & 0.24 \\ 0 & 0 & 0.35 & 0.65 \end{bmatrix} \tag{3}$$

- When $T = 10000$ the algorithm took **14899** iterations to converge to the following parameter estimates:

$$\mathbf{A} = \begin{bmatrix} 0.69 & 0.05 & 0.26 \\ 0.12 & 0.75 & 0.13 \\ 0.15 & 0.26 & 0.59 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.71 & 0.19 & 0.10 & 0 \\ 0.1 & 0.42 & 0.31 & 0.17 \\ 0.03 & 0.17 & 0.19 & 0.61 \end{bmatrix} \tag{4}$$

If we compute the Euclidean-mean distance between the resulting estimated parameters From Eq. (3) and (4) and the real parameters from Eq. (2) we can verify that when training the Markov Model with the observations sequence $\mathcal{O}_{1:10000}$ the distance from the estimated model parameters **A** and **B** to the true estimated parameters is 0.06207996 and 0.05662677 while when we trained the model with the observations sequence $\mathcal{O}_{1:1000}$ the distance from the estimated model parameters **A** and **B** to the true estimated parameters is 0.03173519 and 0.0911593 respectively.

We can notice that both training sessions resulted in a small distance from the estimated and true parameters. However, when $T = 1000$ the estimated transition matrix resulted in a slightly smaller distance to the true matrix than when $T = 10000$. the Markov Model with a longer observation sequence leads to a smaller distance from the estimated observation matrix and the true parameter. A "good" choice of parameters initialization is essential for the algorithm to converge to a parameter estimate that is close to the true parameter value.

The implementation of the Baum Welch algorithm was done following the pseudo-code provided in [Stamp(2004)]. The convergence criteria established in this algorithm is achieved whenever a maximum number of iterations have been reached or when we have found the maximum likelihood ($P(O_{1:T}|\lambda)$).

At every time step a new set of parameter values is derived from the estimated number of counts of emissions and transitions by considering all possible states path. **A** and **B** will be generated, therefore what we will try to do is to keep that set of parameters that maximizes the likelihood.

An iterative method is said to be **convergent** when a sequence of approximate solutions (parameter estimations) converges for given initial approximations. However, the behavior of an algorithm will not always lead to convergence, we might encounter cases in which the approximate solutions will not settle or cases in which the algorithm will end up in a local-maximum (hill-climbing algorithm).

Question 8: Train an HMM with the same parameter dimensions as above, i.e. **A** is a $3x3$ matrix etc. The initialization is left up to you. How close do you get to the parameters above, i.e. how close do you get to the generating parameters in Eq. (2)? What is the problem when it comes to estimating the distance between these matrices? How can you solve these issues?

We will now train the HMM initializing our algorithm with the following parameter values:

$$\mathbf{A} = \begin{bmatrix} 0.05 & 0.80 & 0.15 \\ 0.68 & 0.23 & 0.09 \\ 0.01 & 0.78 & 0.21 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.00 & 0.44 & 0.08 & 0.48 \\ 0.72 & 0.05 & 0.10 & 0.13 \\ 0.32 & 0.60 & 0.05 & 0.03 \end{bmatrix} \boldsymbol{\pi} = \begin{bmatrix} 0.05 & 0.6 & 0.35 \end{bmatrix} \tag{5}$$

The average Euclidean-distance of the initial set of parameters **A**, **B**, $\boldsymbol{\pi}$ in Eq. (5) from the true set of parameters is of 0.8009589, 0.8442455 and 1.17686 respectively.

After training the HMM model we obtain the following parameter estimates:

- When $T = 1000$ the algorithm took **4803** iterations to converge to the following parameter estimates:

$$\mathbf{A} = \begin{bmatrix} 0.78 & 0.04 & 0.18 \\ 0.27 & 0.51 & 0.22 \\ 0.10 & 0.27 & 0.63 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.0 & 0.39 & 0.31 & 0.30 \\ 0.0 & 0.0 & 0.36 & 0.64 \\ 0.70 & 0.24 & 0.06 & 0.0 \end{bmatrix} \tag{6}$$

- When $T = 10000$ the algorithm took **3534** iterations to converge to the following parameter estimates:

$$\mathbf{A} = \begin{bmatrix} 0.57 & 0.27 & 0.16 \\ 0.12 & 0.73 & 0.15 \\ 0.01 & 0.31 & 0.68 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.0 & 0.56 & 0.41 & 0.03 \\ 0.08 & 0.23 & 0.21 & 0.48 \\ 0.7 & 0.20 & 0.10 & 0.0 \end{bmatrix} \tag{7}$$

Compared to the results obtained in ”Question 7” we can notice that both training sessions took less number of iterations to reach convergence. We can also mention that it is evident how sensitive the algorithm is to the choice of parameter initialization, as the resulting parameter estimates for **A** and **B** are very different to the real parameter values.

The average Euclidean distance form the estimated parameters **A** and **B** in Eq. (6) and the true parameters in Eq. (2) was of 0.2101452 and 0.8093505 respectively, while the average Euclidean distance from the results obtain in Eq. (7) and the true parameters **A** and **B** was of 0.2070527 and 0.7289388, respectively.

It is common for the training outcome of the Baum-Welch to converge to a sub-optimal local maximum. Given that the algorithm strongly depends on the initial choice of parameter values. Hence, trying to initialize the algorithm with different parameter values could be a way (not very efficient) of identifying a ”good” set of parameters for initializing the model training. Other alternative such as the one exposed in [Wang and Ju(2011)] suggest a method in which the Baum-Welch algorithm is used in the framework of the Ant Colony Optimisation, making the algorithm less sensitive to the choice of parameter initialization.

Question 9: Train an HMM with different numbers of hidden states. What happens if you use more or less than 3 hidden states? Why? Are three hidden states and four observations the best choice? If not, why? How can you determine the optimal setting? How does this depend on the amount of data you have?

We are now concerned in studying how the size of our training data and the dimensions of our observation probabilities and transition probabilities matrices, can influence on the performance of our estimates.

Referring to the size of our training data there is not an exact rule of how much data we need. However, knowing the number of variables in our model applying a rule of thumb we could consider accounting at least for 10 samples per variable. For instance, if we have 3 states, and 4 possible observations we need to estimate at least 21 variables $(3 * 3 + 4 * 3)$. Hence, it is evident that we cannot accurately estimate all of these variables, with a small number of observations.

Furthermore, deciding on an appropriate number of states for our model, relies in the knowledge we have of our problem. We might want to start with a small number and then successively increment the number of states

Question 10: Initialize your Baum-Welch algorithm with a uniform distribution. How does this effect the learning? Initialize your Baum-Welch algorithm with a diagonal $\mathbf{A}$ matrix and $\pi = [0, 0, 1]$. How does this effect the learning? Initialize your Baum-Welch algorithm with a matrices that are close to the solution. How does this effect the learning?

First, we trained the Markov model using an observation sequence of $T = 1000$ and $T = 10000$. The Baum-Welch algorithm was initialized in both cases with a set of parameters $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{\pi}$ uniformly distributed. Results showed that this kind of setting makes the algorithm get stuck in a sub-optimal maximum. Hence, initializing the parameters with an approximately uniform distribution is a better option. Also, in this setting, the length of the observed sequences do not influence on the algorithm convergence.

Second, we will now use an identity matrix for initializing the matrix of state-transition probabilities and the vector $[0, 0, 1]$ for initializing π.

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \boldsymbol{\pi} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}$$

We are now referring to the trivial case of Markov models which maps each state into itself, i.e with stationary transition probabilities. Under this scenario it will not be possible to estimate the model parameters using stationary probabilities, as we might have different transition probability matrices with the same stationary distribution.

In the E-step, the α matrix will store a single value of 1 corresponding to the last state for all time steps, which will make the gamma matrix follow the same structure, while the di-gamma function will only store values in the positions $\gamma_t(N, N)$ (Where N represents the number of hidden-states). This will make re-estimating $\mathbf{A}$ and $\mathbf{B}$ unfeasible as all operations will involve dividing by zero.

Finally, we will now initialize the Baum-Welch algorithm with matrices that are close to the solution.

$$\mathbf{A} = \begin{bmatrix} 0.68 & 0.04 & 0.28 \\ 0.07 & 0.76 & 0.17 \\ 0.19 & 0.25 & 0.56 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.65 & 0.25 & 0.07 & 0.03 \\ 0.16 & 0.34 & 0.37 & 0.13 \\ 0.08 & 0.18 & 0.12 & 0.62 \end{bmatrix} \boldsymbol{\pi} = \begin{bmatrix} 0.95 & 0.03 & 0.02 \end{bmatrix} \tag{8}$$

The estimated Euclidean distance of the set of parameters $\mathbf{A}$, $\mathbf{B}$ and $\boldsymbol{\pi}$ in Eq. (8) and the true set of parameters in Eq. (2) is of 0.0673933, 0.1242821, and 0.06164414 respectively.

- When $T = 1000$ the algorithm took **2032** iterations to converge to the following parameter estimates:

$$\mathbf{A} = \begin{bmatrix} 0.70 & 0.01 & 0.29 \\ 0.10 & 0.81 & 0.09 \\ 0.19 & 0.31 & 0.51 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.69 & 0.23 & 0.07 & 0.01 \\ 0.07 & 0.41 & 0.28 & 0.24 \\ 0 & 0 & 0.35 & 0.65 \end{bmatrix} \tag{9}$$

- When $T = 10000$ the algorithm took **14530** iterations to converge to the following parameter estimates:

$$\mathbf{A} = \begin{bmatrix} 0.69 & 0.05 & 0.26 \\ 0.11 & 0.75 & 0.14 \\ 0.15 & 0.26 & 0.59 \end{bmatrix} \mathbf{B} = \begin{bmatrix} 0.71 & 0.19 & 0.10 & 0 \\ 0.10 & 0.42 & 0.31 & 0.17 \\ 0.03 & 0.17 & 0.19 & 0.61 \end{bmatrix} \tag{10}$$

Compared to the results obtained in Eq. (3) the algorithm took slightly less iterations to converge. When training the model with $T = 1000$ the average Euclidean distance for the parameters $\mathbf{A}$ and $\mathbf{B}$ was of 0.02828427 and

0.09552549 respectively, and when $T = 10000$ the average Euclidean distance was of 0.06313438 and 0.05662677, which are almost the same than in the setting of ”Question 7”.

References

[Stamp(2004)] Mark Stamp. A revealing introduction to hidden markov models. *Science*, pages 1–20, 01 2004.

[Wang and Ju(2011)] Qingmiao Wang and Shiguang Ju. Aco-based bw algorithm for parameter estimation of hidden markov models. *IJCAT*, 41:281–286, 09 2011. doi: 10.1504/IJCAT.2011.042704.