CSE 250A. Assignment 6

Hao-en Sung (A53204772)
wrangle1005@gmail.com

November 12, 2016

7.1 Viterbi algorithm

Sol. I implement the Viterbi algorithm in C++, as shown in Code 1. Later, I use MATLAB to draw the state transition figure, as shown in Fig. 1. The drawin code is recorded as Code 2.

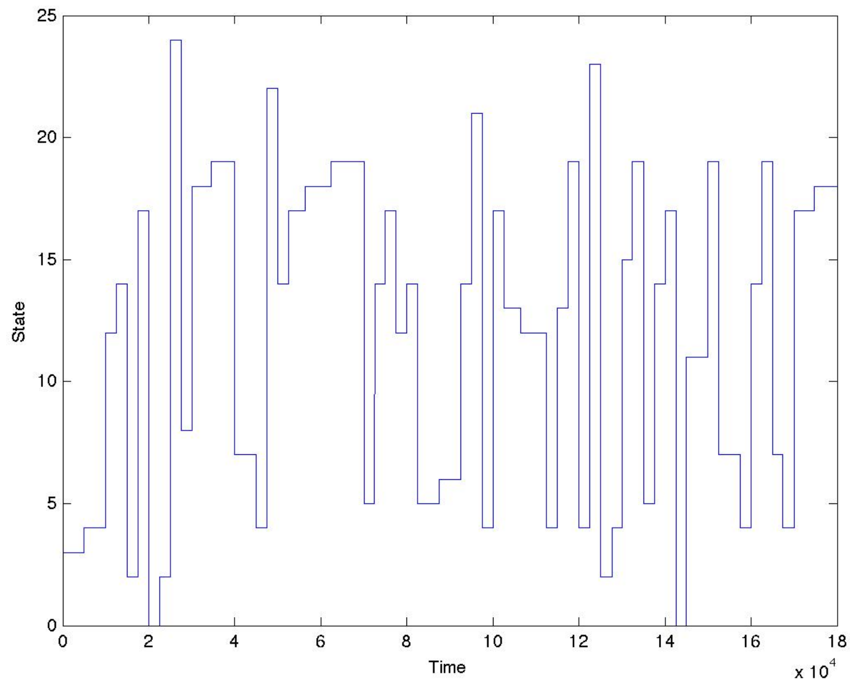

Figure 1: State Transition

□

7.2 Inference in HMMs

(a) $P(S_t = i | S_{t+1} = j, O_1, ..., O_T)$

Sol. I can first use the Bayes rule to derive the following formula.

$$P(S_t = i | S_{t+1} = j, O_1, ..., O_T) = \frac{P(S_t = i, S_{t+1} = j, O_1, ..., O_T)}{P(S_{t+1} = j, O_1, ..., O_T)}$$

For $P(S_t = i, S_{t+1} = j, O_1, ..., O_T)$, I can derive

$$\begin{aligned}
P(S_t = i, S_{t+1} = j, O_1, ..., O_T) &= \alpha_{i,t} \cdot P(S_{t+1} = j, O_{t+1}, ..., O_T | S_t = i, O_1, ..., O_t) \\
&= \frac{\alpha_{i,t}}{P(S_t = i)} \cdot P(S_t = i, S_{t+1} = j, O_{t+1}, ..., O_T) \\
&= \frac{\alpha_{i,t}}{P(S_t = i)} \cdot P(S_t = i, O_{t+1} | S_{t+1} = j, O_{t+2}, ..., O_T) \cdot \beta_{j,t+1} \cdot S_{t+2} \\
&= \frac{\alpha_{i,t}}{P(S_t = i)} \cdot P(S_t = i, O_{t+1}, S_{t+1} = j) \cdot \beta_{j,t+1} \\
&= \alpha_{i,t} \cdot P(S_{t+1} = j | S_t = i) \cdot P(O_{t+1} | S_{t+1} = j) \cdot \beta_{j,t+1}.
\end{aligned}$$

Thus, the original formula can be written as

$$\begin{aligned}
P(S_t = i | S_{t+1} = j, O_1, ..., O_T) &= \frac{\alpha_{i,t} \cdot a_{i,j} \cdot b_j(O_{t+1}) \cdot \beta_{j,t+1}}{\alpha_{j,t+1} \cdot \beta_{j,t+1}} \\
&= \frac{\alpha_{i,t}}{\alpha_{j,t+1}} \cdot a_{i,j} \cdot b_j(O_{t+1}).
\end{aligned}$$

□

(b) $P(S_{t+1}=j|S_t=i,O_1,...,O_T)$

Sol. Similar to (a), I can apply joint distribution $P(S_t=i,S_{t+1}=j,O_1,...,O_T)$ and calculate the marginalization.

$$\begin{aligned} P(S_{t+1}=j|S_t=i,O_1,...,O_T) &= \frac{P(S_t=i,S_{t+1}=j,O_1,...,O_T)}{P(S_t=i,O_1,...,O_T)} \\ &= \frac{\alpha_{i,t}\cdot P(S_{t+1}=j|S_t=i)\cdot P(O_{t+1}|S_{t+1}=j)\cdot \beta_{j,t+1}}{\alpha_{i,t}\cdot\beta_{i,t}} \\ &= \frac{\beta_{j,t+1}}{\beta_{i,t}}\cdot a_{i,j}\cdot b_j(O_{t+1}) \end{aligned}$$

□

(c) $P(S_{t-1}=i,S_t=k,S_{t+1}=j|O_1,...,O_T)$

Sol. Similar to (a) and (b), I can derive the numerator part as

$$\begin{aligned} P(S_{t-1}=i,S_t=k,S_{t+1}=j,O_1,...,O_T) &= \alpha_{i,t-1}\cdot P(S_t=k,S_{t+1}=j,O_t,...,O_T|S_{t-1}=i,O_1,...,O_{t-1}) \\ &= \alpha_{i,t-1}\cdot P(S_t=k,S_{t+1}=j,O_t,...,O_T|S_{t-1}=i) \\ &= \frac{\alpha_{i,t-1}}{P(S_{t-1}=i)}\cdot P(S_{t-1}=i,S_t=k,O_t,O_{t+1}|S_{t+1}=j,O_{t+2},...,O_T) \\ &\quad \cdot\beta_{j,t+1}\cdot P(S_{t+1}=j) \\ &= \frac{\alpha_{i,t-1}}{P(S_{t-1}=i)}\cdot P(S_{t-1}=i,S_t=k,S_{t+1}=j,O_t,O_{t+1})\cdot\beta_{j,t+1} \\ &= \alpha_{i,t-1}\cdot P(S_t=k|S_{t-1}=i)\cdot P(S_{t+1}=j|S_t=k) \\ &\qquad\quad \cdot P(O_t|S_t=k)\cdot P(O_{t+1}|S_{t+1}=j)\cdot\beta_{j,t+1} \\ &= \alpha_{i,t-1}\cdot a_{i,j}\cdot a_{j,k}\cdot b_k(O_t)\cdot b_j(O_{t+1})\cdot\beta_{j,t+1}. \end{aligned}$$

Then the original problem can be written as follows.

$$\begin{aligned} P(S_{t-1}=i,S_t=k,S_{t+1}=j|O_1,...,O_T) &= \frac{P(S_{t-1}=i,S_t=k,S_{t+1}=j,O_1,...,O_T)}{P(O_1,...,O_T)} \\ &= \frac{\alpha_{i,t-1}\cdot\beta_{j,t+1}}{\sum_k \alpha_{k,t}\cdot\beta_{k,t}}\cdot P(S_t=k|S_{t-1}=i) \\ &\qquad \cdot P(S_{t+1}=j|S_t=k)\cdot P(O_t|S_t=k)\cdot P(O_{t+1}|S_{t+1}=j) \\ &= \frac{\alpha_{i,t-1}\cdot\beta_{j,t+1}}{\sum_k \alpha_{k,t}\cdot\beta_{k,t}}\cdot a_{i,k}\cdot a_{k,j}\cdot b_k(O_t)\cdot b_j(O_{t+1}). \end{aligned}$$

□

(d) $P(S_{t-1}=i|S_{t+1}=j,O_1,...,O_T)$

Sol. Similar to previous derivations, I can first apply Bayes-rule to rewrite it as

$$\begin{aligned} P(S_{t-1}=i|S_{t+1}=j,O_1,...,O_T) &= \frac{P(S_{t-1}=i,S_{t+1}=j,O_1,...,O_T)}{P(S_{t+1}=j,O_1,...,O_T)} \\ &= \frac{\sum_k P(S_{t-1}=i,S_t=k,S_{t+1}=j,O_1,...,O_T)}{\alpha_{j,t+1}\cdot\beta_{j,t+1}}. \end{aligned}$$

When substitute the conclusion from (c), I have

$$\begin{aligned} P(S_{t-1}=i|S_{t+1}=j,O_1,...,O_T) &= \frac{\alpha_{i,t-1}}{\alpha_{j,t+1}}\cdot P(O_{t+1}|S_{t+1}=j) \\ &\quad \cdot\sum_k P(O_t|S_t=k)\cdot P(S_t=k|S_{t-1}=i)\cdot P(S_{t+1}=j|S_t=k) \\ &= \frac{\alpha_{i,t-1}}{\alpha_{j,t+1}}\cdot b_j(O_{t+1})\cdot\sum_k a_{i,k}\cdot a_{k,j}\cdot b_k(O_t). \end{aligned}$$

□

7.3 Conditional independence

Sol. I omit all calculation but record only the answer here.

False	$P(S_t \vert S_{t-1}) = P(S_t \vert S_{t-1}, O_t)$
False	$P(S_t \vert S_{t-1}) = P(S_t \vert S_{t-1}, S_{t+1})$
True	$P(S_t \vert S_{t-1}) = P(S_t \vert S_{t-1}, O_{t-1})$
False	$P(S_t \vert O_{t-1}) = P(S_t \vert O_1, ..., O_{t-1})$
True	$P(O_t \vert S_{t-1}) = P(O_t \vert S_{t-1}, O_{t-1})$
False	$P(O_t \vert O_{t-1}) = P(O_t \vert O_1, ..., O_{t-1})$
True	$P(O_1, ..., O_T) = \prod_{t=1}^{T} P(O_t \vert O_1, ..., O_{t-1})$
True	$P(S_2, S_3, ..., S_T \vert S_1) = \prod_{t=2}^{T} P(S_t \vert S_{t-1})$
True	$P(S_1, S_2, ..., S_{T-1} \vert S_T) = \prod_{t=1}^{T-1} P(S_t \vert S_{t+1})$
True	$P(O_1, ..., O_T \vert S_1, S_2, ..., S_T) = \prod_{t=1}^{T} P(O_t \vert S_t)$
False	$P(S_1, S_2, ..., S_T \vert O_1, ..., O_T) = P(S_t \vert O_t)$
False	$P(S_1, S_2, ..., S_T, O_1, ..., O_T) = P(S_t, O_t)$

Table 1: True or false

□

7.4 Belief updating

(a) Consider the discrete hidden Markov model(HMM) with hidden states S_t, observations O_t, transition matrix a_{ij}, and emission matrix b_{ik}. Let

$$q_{it} = P(S_t = i | o_1, o_2, ..., o_t)$$

denote the conditional probability that S_t is in the ith state of the HMM based on the evidence up to and including time t. Derive the recursion relation:

$$q_{jt} = \frac{1}{Z_t} b_j(o_t) \sum_i a_{ij} q_{it-1} \text{ where } Z_t = \sum_{ij} b_j(o_t) \sum_i a_{ij} q_{it-1}.$$

Justify each step in your derivation—for example, by appealing to Bayes rule or properties of conditional independence.

Sol. Consider $b_j(o_t) a_{ij} q_{it-1}$, it can actually be rewritten as

$$\begin{aligned} b_j(o_t) a_{ij} q_{it-1} &= P(o_t | S_t = j) \cdot (S_t = j | S_{t-1} = i) \cdot P(S_{t-1} = i | o_1, ..., o_{t-1}) \\ &= P(o_t | S_t = j, S_{t-1} = i, o_1, ..., o_t) \\ &\qquad \cdot P(S_t = j | S_{t-1} = i, o_1, ..., o_{t-1}) \cdot P(S_{t-1} = i | o_1, ..., o_{t-1}) \\ &= P(S_t = j, S_{t-1} = i, o_t | o_1, ..., o_{t-1}). \end{aligned}$$

It can be derived that

$$\begin{aligned} q_{jt} &= \frac{P(S_t = j, o_1, ..., o_t)}{P(o_1, ..., o_t)} \\ &= \frac{\sum_i P(S_t = j, S_{t-1} = i, o_1, ..., o_t)}{\sum_{ij} P(S_t = j, S_{t-1} = i, o_1, ..., o_t)} \\ &= \frac{\sum_i b_j(o_t) a_{ij} q_{it-1} \cdot P(o_1, ..., o_{t-1})}{\sum_{ij} b_j(o_t) a_{ij} q_{it-1} \cdot P(o_1, ..., o_{t-1})} \\ &= \frac{1}{Z_t} b_j(o_t) a_{ij} q_{it-1}. \end{aligned}$$

□

(b) Consider the dynamical system with continuous, real-valued hidden states X_t and observations Y_t, represented by the belief network shown below. By analogy to the previous problem (replacing sums by integrals), derive the recursion relation:

$$P(x_t | y_1, y_2, ..., y_t) = \frac{1}{Z_t} P(y_t | x_t) \int dx_{t-1} P(x_t | x_{t-1}) P(x_{t-1} | y_1, y_2, ..., y_{t-1}),$$

where Z_t is the appropriate normalization factor,

$$Z_t = \int dx_t P(y_t | x_t) \int dx_{t-1} P(x_t | x_{t-1}) P(x_{t-1} | y_1, y_2, ..., y_{t-1})$$

In principle, an agent could use this recursion for real-time updating of beliefs in arbitrarily complicated continuous worlds. In practice, why is this difficult for all but Gaussian random variables?

Sol. Similar to (a), I just need to replace the summation with integral as follows.

$$\begin{aligned}
P(x_t|y_1, y_2, ..., y_t) &= \frac{P(x_t, y_1, ..., y_t)}{P(y_1, ..., y_t)} \\
&= \frac{\int dx_{t-1} P(x_t, x_{t-1}, y_1, ..., y_t)}{\int dx_t \int dx_{t-1} P(x_t, x_{t-1}, y_1, ..., y_t)} \\
&= \frac{\int dx_{t-1} P(y_t|x_t, x_{t-1}, y_1, ..., y_t) \cdot P(x_t|x_{t-1}, y_1, ..., y_t) \cdot P(x_{t-1}|y_1, ..., y_t)}{\int dx_t \int dx_{t-1} P(y_t|x_t, x_{t-1}, y_1, ..., y_t) \cdot P(x_t|x_{t-1}, y_1, ..., y_t) \cdot P(x_{t-1}|y_1, ..., y_t)} \\
&= \frac{\int dx_{t-1} P(y_t|x_t) \cdot P(x_t|x_{t-1}) \cdot P(x_{t-1}|y_1, ..., y_t)}{\int dx_t \int dx_{t-1} P(y_t|x_t) \cdot P(x_t|x_{t-1}) \cdot P(x_{t-1}|y_1, ..., y_t)} \\
&= \frac{P(y_t|x_t) \int dx_{t-1} \cdot P(x_t|x_{t-1}) \cdot P(x_{t-1}|y_1, ..., y_t)}{\int dx_t P(y_t|x_t) \int dx_{t-1} P(x_t|x_{t-1}) \cdot P(x_{t-1}|y_1, ..., y_t)} \\
&= \frac{1}{Z_t} P(y_t|x_t) \int dx_{t-1} \cdot P(x_t|x_{t-1}) \cdot P(x_{t-1}|y_1, ..., y_t)
\end{aligned}$$

The reason that the Gaussian random variables are much easier for real time update is because of the mathematical property.

- If $P(\vec{X})$ and $P(\vec{Y})$ are Gaussian random variables, so is $P(\alpha\vec{X} + \beta\vec{Y})$, where α and β are linear scalar coefficients.
- If $P(\vec{X})$ and $P(\vec{Y})$ are Gaussian random variables, so are their marginal results, ex: $P(X_i)$, and conditional results, ex: $P(X_i|Y_i)$.

□

7.5 Mixture model decision boundary

(a) Compute the posterior distribution $P(y = \vec{x})$ as a function of the parameters (π_0, π_1, $\vec{\mu_0}$, $\vec{\mu_1}$, Σ_0, Σ_1) of the Gaussian mixture model.

Sol. Based on simple Bayes-rule, I have

$$\begin{aligned}
P(y = 1|\vec{x}) &= \frac{P(y = 1, \vec{x})}{\sum_i P(y = i, \vec{x})} \\
&= \frac{P(\vec{x}|y = 1) \cdot P(y = 1)}{\sum_i P(\vec{x}|y = i) \cdot P(y = i)} \\
&= \frac{(2\pi)^{-\frac{d}{2}} |\Sigma_1|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_1})^T \Sigma_1^{-1} (\vec{x}-\vec{\mu_1})} \cdot \pi_1}{(2\pi)^{-\frac{d}{2}} |\Sigma_0|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_0})^T \Sigma_0^{-1} (\vec{x}-\vec{\mu_0})} \cdot \pi_0 + (2\pi)^{-\frac{d}{2}} |\Sigma_1|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_1})^T \Sigma_1^{-1} (\vec{x}-\vec{\mu_1})} \cdot \pi_1} \\
&= \frac{|\Sigma_1|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_1})^T \Sigma_1^{-1} (\vec{x}-\vec{\mu_1})} \cdot \pi_1}{|\Sigma_0|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_0})^T \Sigma_0^{-1} (\vec{x}-\vec{\mu_0})} \cdot \pi_0 + |\Sigma_1|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_1})^T \Sigma_1^{-1} (\vec{x}-\vec{\mu_1})} \cdot \pi_1}.
\end{aligned}$$

□

(b) Consider the special case of this model where the two mixture components share the same covariance matrix: namely, $\Sigma_0 = \Sigma_1 = \Sigma$. In this case, show that your answer from part (a) can be written as:

$$P(y = 1|\vec{x}) = \sigma(\vec{w} \cdot \vec{x} + b) \text{ where } \sigma(z) = \frac{1}{1 + e^{-z}}$$

As part of your answer, you should express the parameters $(\vec{w}, b)$ of the sigmoid function explicitly in terms of the parameters $(\pi_0, \pi_1, \vec{\mu_0}, \vec{\mu_1}, \Sigma)$ of the Gaussian mixture model.

Sol. I can start from the conclusion of (a) and derive

$$\begin{aligned}
P(y = 1|\vec{x}) &= \frac{|\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_1})^T \Sigma^{-1} (\vec{x}-\vec{\mu_1})} \cdot \pi_1}{|\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_0})^T \Sigma^{-1} (\vec{x}-\vec{\mu_0})} \cdot \pi_0 + |\Sigma|^{-\frac{1}{2}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu_1})^T \Sigma^{-1} (\vec{x}-\vec{\mu_1})} \cdot \pi_1} \\
&= \frac{e^{-\frac{1}{2}(\vec{x}-\vec{\mu_1})^T \Sigma^{-1} (\vec{x}-\vec{\mu_1})} \cdot \pi_1}{e^{-\frac{1}{2}(\vec{x}-\vec{\mu_0})^T \Sigma^{-1} (\vec{x}-\vec{\mu_0})} \cdot \pi_0 + e^{-\frac{1}{2}(\vec{x}-\vec{\mu_1})^T \Sigma^{-1} (\vec{x}-\vec{\mu_1})} \cdot \pi_1} \\
&= \frac{1}{1 + \frac{\pi_0}{\pi_1} \cdot e^{-\frac{1}{2}\vec{\mu_0}^T \Sigma^{-1} \vec{\mu_0} + \frac{1}{2}\vec{\mu_1}^T \Sigma^{-1} \vec{\mu_1} + (\vec{\mu_0}-\vec{\mu_1})^T \Sigma^{-1} \vec{x}}} \\
&= \frac{1}{1 + e^{-\left[\frac{1}{2}\vec{\mu_0}^T \Sigma^{-1} \vec{\mu_0} - \frac{1}{2}\vec{\mu_1}^T \Sigma^{-1} \vec{\mu_1} - \log(\frac{\pi_0}{\pi_1}) - (\vec{\mu_0}-\vec{\mu_1})^T \Sigma^{-1} \vec{x}\right]}}.
\end{aligned}$$

That is to say,

$$\begin{aligned}
\vec{w} &= -(\vec{\mu_0} - \vec{\mu_1})^T \Sigma^{-1} \\
b &= \frac{1}{2}\vec{\mu_0}^T \Sigma^{-1} \vec{\mu_0} - \frac{1}{2}\vec{\mu_1}^T \Sigma^{-1} \vec{\mu_1} - \log(\frac{\pi_0}{\pi_1})
\end{aligned}$$

□

(c) Assume again that $\Sigma_0 = \Sigma_1 = \Sigma$. Note that in this case, the decision boundary for the mixture model reduces to a hyperplane; namely, we have $P(y = 1|\vec{x}) = P(y = 0|\vec{x})$ when $\vec{w} \cdot \vec{x} + b = 0$. Let k be a positive integer. Show that the set of points for which

$$\frac{P(y = 1|\vec{x})}{P(y = 0|\vec{x})} = k$$

is also described by a hyperplane, and find the equation for this hyperplane. (These are the points for which one class is precisely k times more likely than the other.) Of course, your answer should recover the hyperplane decision boundary $\vec{w} \cdot \vec{x} + b = 0$ when $k = 1$.

Sol. Since $P(y = 0|\vec{x}) + P(y = 1|\vec{x}) = 1$, I know that

$$\begin{aligned} P(y = 1|\vec{x}) &= \frac{k}{k+1}, \\ P(y = 0|\vec{x}) &= \frac{1}{k+1}. \end{aligned}$$

On top of that, I have

$$\begin{aligned} P(y = 0|\vec{x}) = \frac{1}{1+k} &= 1 - P(y = 1|\vec{x}) \\ &= 1 - \sigma(\vec{w} \cdot \vec{x} + b) \\ &= \sigma(-\vec{w} \cdot \vec{x} - b) \\ &= \frac{1}{1 + e^{\left[\frac{1}{2}\vec{\mu_0}^T \Sigma^{-1} \vec{\mu_0} - \frac{1}{2}\vec{\mu_1}^T \Sigma^{-1} \vec{\mu_1} - \log(\frac{\pi_0}{\pi_1}) - (\vec{\mu_0} - \vec{\mu_1})\Sigma^{-1}\vec{x}\right]}}. \end{aligned}$$

In other words,

$$\begin{aligned} k &= e^{\left[\frac{1}{2}\vec{\mu_0}^T \Sigma^{-1} \vec{\mu_0} - \frac{1}{2}\vec{\mu_1}^T \Sigma^{-1} \vec{\mu_1} - \log(\frac{\pi_0}{\pi_1}) - (\vec{\mu_0} - \vec{\mu_1})\Sigma^{-1}\vec{x}\right]} \\ \log k &= \left[\frac{1}{2}\vec{\mu_0}^T \Sigma^{-1} \vec{\mu_0} - \frac{1}{2}\vec{\mu_1}^T \Sigma^{-1} \vec{\mu_1} - \log(\frac{\pi_0}{\pi_1}) - (\vec{\mu_0} - \vec{\mu_1})\Sigma^{-1}\vec{x}\right]. \end{aligned}$$

Then, I have the hyperplane $\vec{w} \cdot \vec{x} + b = \log k$. It is easy to observe that when $k = 1$, $\vec{w} \cdot \vec{x} + b = 0$. $\square$

Appendix

```
#include <cstdio>
#include <cstdlib>
#include <cfloat>
#include <cmath>
#include <vector>
#include <string>

using namespace std;

const int NUM_STAT = 26;
const int NUM_OBSV = 2;
const int NUM_DATA = 180000;

int main() {
    vector<double> init(NUM_STAT, 0);
    { // initialStateDistribution.txt
        FILE* pf = fopen("../dat/initialStateDistribution.txt", "r");
        if (pf == NULL) {
            fprintf(stderr, "cannot open initialStateDistribution.txt");
            exit(EXIT_FAILURE);
        }

        double d;
        for (int i = 0; i < NUM_STAT; i++) {
            fscanf(pf, "%lf", &d);
            init[i] = d;
        }

        fclose(pf);
    }

    vector<vector<double>> trans(NUM_STAT, vector<double>(NUM_STAT, 0));
    { // transitionMatrix.txt
        FILE* pf = fopen("../dat/transitionMatrix.txt", "r");
        if (pf == NULL) {
            fprintf(stderr, "cannot open transitionMatrix.txt");
            exit(EXIT_FAILURE);
        }

        double d;
        for (int i = 0; i < NUM_STAT; i++) {
            for (int j = 0; j < NUM_STAT; j++) {
                fscanf(pf, "%lf", &d);
                trans[i][j] = d;
            }
        }

        fclose(pf);
    }

    vector<vector<double>> obsv(NUM_STAT, vector<double>(NUM_OBSV, 0));
    { // emissionMatrix.txt
        FILE* pf = fopen("../dat/emissionMatrix.txt", "r");
        if (pf == NULL) {
            fprintf(stderr, "cannot open emissionMatrix.txt");
            exit(EXIT_FAILURE);
        }

        double d;
        for (int i = 0; i < NUM_STAT; i++) {
            for (int j = 0; j < NUM_OBSV; j++) {
                fscanf(pf, "%lf", &d);
                obsv[i][j] = d;
            }
        }

        fclose(pf);
    }

    vector<int> data(NUM_DATA, 0);
    { // observations.txt
        FILE* pf = fopen("../dat/observations.txt", "r");
        if (pf == NULL) {
            fprintf(stderr, "cannot open observations.txt");
            exit(EXIT_FAILURE);
        }

        int d;
        for (int i = 0; i < NUM_DATA; i++) {
            fscanf(pf, "%d", &d);
            data[i] = d;
        }

        fclose(pf);
    }

    vector<vector<double>> val(NUM_DATA, vector<double>(NUM_STAT, 0));
    vector<vector<int>> rcd(NUM_DATA, vector<int>(NUM_STAT, 0));
```

```
    for (int j = 0; j < NUM_STAT; j++) {
        val[0][j] = log(init[j]) + log(obsv[j][data[0]]);
    }
    for (int i = 1; i < NUM_DATA; i++) {
        for (int j = 0; j < NUM_STAT; j++) { // next
            double mmax = -DBL_MAX;
            int idx = -1;
            for (int k = 0; k < NUM_STAT; k++) { // prev
                double tmp = val[i-1][k] + log(trans[k][j]);
                if (tmp > mmax) {
                    mmax = tmp;
                    idx = k;
                }
            }
            rcd[i][j] = idx;
            val[i][j] = mmax + log(obsv[j][data[i]]);
        }
    }

    vector<int> s(NUM_DATA, 0);
    double mmax = -DBL_MAX;
    int idx = -1;
    for (int j = 0; j < NUM_STAT; j++) {
        if (val[NUM_DATA-1][j] > mmax) {
            mmax = val[NUM_DATA-1][j];
            idx = j;
        }
    }
    s[NUM_DATA-1] = idx;
    for (int i = NUM_DATA-2; i >= 0; i--) {
        s[i] = idx = rcd[i][idx];
    }

    { // output.txt
        FILE* pf = fopen("../res/output.csv", "w");
        if (pf == NULL) {
            fprintf(stderr, "cannot write output.csv");
            exit(EXIT_FAILURE);
        }

        for (int i = 0; i < NUM_DATA; i++) {
            fprintf(pf, "%d%c", s[i], i == NUM_DATA-1 ?'\n' :',');
        }

        fclose(pf);
    }

    /*{ // for checking
        string str;
        int lst = s[0];
        str.push_back('a' + lst);
        for (int i = 1; i < NUM_DATA; i++) {
            if (lst == s[i]) {
                // do nothing
            } else {
                lst = s[i];
                str.push_back('a' + lst);
            }
        }
        printf("%s\n", str.c_str());
    }*/
}
```

Listing 1: Code for Viterbi Implementation

```
% read in data
M = csvread('../res/output.csv');

% plot figure
res = figure('visible', 'off');
plot(M);
xlabel('Time');
ylabel('State')
print -r96 % set resolution
saveas(res, '../res/state.jpg');
```

Listing 2: Code for Drawing State Transition