Hidden Markov Model (Final Report of STAT 534)

Yikun Zhang
Department of Statistics,
University of Washington, Seattle
Seattle, WA 98195
yikun@uw.edu

Abstract

In this report, we are supposed to furnish some detailed information about how to train an Hidden Markov Model (HMM) by the Baum-Welch method. A 5-fold Cross-validation (CV) is applied to choose an appropriate number of states. In addition, we implement the Viterbi algorithm to calculate the most likely sequence of states for all the data. Finally, we will predict the next output and the next state given any observed sequence.

1 Introduction and Background

The hidden Markov model (HMM) is a direct extension of the (first-order) Markov chain with a doubly embedded stochastic process. The underlying Markov chain model (with state spaces) is not observable while each observation is a probabilistic function of the corresponding state.[4] The applications of HMM spread in various fields, from weather prediction[3] to speech recognition[4].

1.1 Terminologies and Notations

By definition, HMM embraces a discrete time Markov chain with a discrete state space as the hidden state model. In this project, the observed variables are discrete/categorical, so the resulting model is called the multinomial hidden Markov model. Now we introduce some notations for this HMM, which follow from Rabiner [4]. Suppose that the HMM has N distinct hidden states denoted by $S_1, ..., S_N$ and M distinct observation symbols per state denoted by $v_1, ..., v_M$.

- The state at time t: $q_t \in \{S_1, ..., S_N\}$
- The output at time t: $O_t \in \{v_1, ..., v_M\}$
- The transition probability matrix: $A = [a_{ij}]_{i,j=1}^{N}$, where

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i] \geq 0, \quad 1 \leq i, j \leq N, \qquad \sum_{j=1}^{N} a_{ij} = 1$$

- The emission probability matrix: $B = [b_i(k)]_{i=1}^{N}{}_{k=1}^{M}$, where

$$b_i(k) = P[O_t = v_k | q_t = S_i], \quad 1 \leq j \leq N, \quad 1 \leq k \leq M$$

- The initial state distribution $\pi = [\pi_i]_{i=1}^{M}$, where $\pi_i = P[q_1 = S_i], \quad 1 \leq i \leq N$

Since a multinomial HMM can be completely determined by three probability measures A, B, and π, for convenience, we use the compact notation $\lambda = (A, B, \pi)$.

1.2 The Four Basic Problems for HMMs

There are four problems that we need to solve for the HMM model in order to make it useful in real-world applications.[1]

Problem 1 Given the observation sequence $O = O_1 O_2 \cdots O_T$ and a model $\lambda = (A, B, \pi)$, how do we efficiently compute $P(O|\lambda)$, the probability of the observation sequence given the model? The Forward and Backward algorithms are developed to tackle this problem. We define the *forward probability* and *backward probability* as

$$\alpha_t(i) = P[O_{1:t}, q_t = S_i|\lambda], \quad \beta_t(i) = P(O_{(t+1):T}|q_t = S_i, \lambda). \tag{1}$$

Forward Algorithm:

1. Let $\alpha_1(i) = \pi_i b_i(O_1)$ for $1 \le i \le N$;
2. For $t = 1, ..., T-1$, compute

$$\alpha_{t+1}(i) = \sum_{j=1}^{N} \alpha_t(j) a_{ij} b_j(O_{t+1})$$

Backward Algorithm:

1. Let $\beta_T(i) = 1$ for $1 \le i \le N$;
2. For $t = T-1, T-2, ..., 1$, compute

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j)$$

Then the likelihood $P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) = \sum_{j=1}^{N} \pi_j b_j(O_1)\beta_1(j)$.

Problem 2 Given the observation sequence $O = O_1 O_2 \cdots O_T$ and a model $\lambda = (A, B, \pi)$, how do we reconstruct the most likely state sequence $Q = q_1 q_2 \cdots q_T$? Considering the possibility of state transitions with zero probability, we seek to find the single best state sequence to maximize $P(Q|O, \lambda)$ which is equivalent to maximizing $P(Q, O|\lambda)$. The **Viterbi Algorithm** is a well-known method to tackle this optimization problem.

Problem 3 How do we adjust the model parameters $\lambda = (A, B, \pi)$ to maximize $P(O|\lambda)$? With the presence of hidden variables, the direct approach to the Maximum Likelihood Estimator is intractable and an EM algorithm for HMM is proposed (**Baum-Welch Algorithm**).

Problem 4 Given the observation sequence $O = O_1 O_2 \cdots O_T$ and a model $\lambda = (A, B, \pi)$, how do we predict the next output and the next hidden state of HMM? In order to predict the most possible outcome at time point $T+1$ or more generally, at time point $T+h$, for $h = 1, 2, ...$, we resort to the maximum likelihood method. Note that

$$\begin{aligned}
P(O_{T+h}|O_{1:T}) &= \sum_{i=1}^{N}\sum_{j=1}^{N} P(O_{T+h}, q_{T+h} = S_i, q_T = S_j|O_{1:T}) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} P(O_{T+h}, q_{T+h} = S_i|q_T = S_j) \cdot P(q_T = S_j|O_{1:T}) \\
&= \sum_{i=1}^{N}\sum_{j=1}^{N} P(O_{T+h}|q_{T+h} = S_i) \cdot P(q_{T+h} = S_i|q_T = S_j) \cdot P(q_T = S_j|O_{1:T}) \\
&= \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} b_i(O_{T+h}) a_{ji}^{(h)} \alpha_T(j)}{\sum_{k=1}^{N} a_T(k)},
\end{aligned} \tag{2}$$

where $a_{ji}^{(h)}$ is the (j, i) entry of the h-step transition probability matrix, *i.e.*, A^h. In this project, we only consider the case when $h = 1$. The predicted probabilities for the next output are $P(O_{T+1} = v_k|O_{1:T}), 1 \le k \le M$ and we predict the next output of $O_{1:T}$ by

$$\hat{O}_{T+1} = \arg\max_{v_k} P(O_{T+1} = v_k|O_{1:T}), \quad 1 \le k \le M.$$

As for the prediction of the next hidden state q_{T+1}, there are several plausible approaches.

[1] We briefly summarize the problem statements in Rabiner [4] and add one more problem based on our project requirement.

For instance, we can apply the Viterbi algorithm to $O_{1:(T+1)}$ to reconstruct the most likely hidden state sequence $q_{1:(T+1)}$ and obtain the next state by examining the last element of this state sequence. Or, based on the last hidden state q_T inferred by the Viterbi algorithm, we can choose the next hidden state by $\arg\max_{S_k} P(q_{T+1} = S_k|q_T)$. If we assume that emission probabilities are nonzero, the maximum likelihood method can also be applied to predict the hidden state at time $T + h$:

$$\begin{aligned} P(q_{T+h} = S_k|O_{1:T}) &= \sum_{i=1}^{N} P(q_{T+h} = S_k, q_T = S_i|O_{1:T}) \\ &= \sum_{i=1}^{N} P(q_{T+h} = S_k|q_T = S_i) \cdot P(q_T = S_i|O_{1:T}) \\ &= \frac{\sum_{i=1}^{N} a_{ik}^{(h)} \alpha_T(k)}{\sum_{j=1}^{N} \alpha_T(j)}, \end{aligned} \tag{3}$$

where $a_{ik}^{(h)}$ is the (i, k) entry of the h-step transition probability matrix. When $h = 1$,

$$\hat{q}_{T+1} = \arg\max_{S_k} P(q_{T+1} = S_k|O_{1:T}).$$

1.3 Cross-Validation and Model Selection

Cross-validation (CV) is the most widely used method for estimating prediction error and hyperparameter tuning.[2] For hyperparameter tuning in HMM, an intuitive approach is to consider different values for the number of states and pick the one with the highest likelihood value. However, this idea tends to favor complex models and suffer the problem of overfitting, *i.e.*, the more hidden states we fit to the data, the higher likelihood value we will obtain. (As shown in Section 2.2.) To tackle this issue, we use the Akaike information criterion (AIC) and Bayesian information criterion (BIC) that penalize complex models:[1]

$$AIC = -2 \log L(\lambda|O) + 2p, \quad BIC = -2 \log L(\lambda|O) + p \log(T),$$

where T indicates the length of the observed sequence and $p = p(\lambda)$ denotes the number of independent parameters of the model. In an HMM, $p(\lambda) = N^2 - 1 + N(M - 1)$.

2 Implementation Details and Results

The training data contains $n_0 = 1000$ sequences of $T = 40$ integers from $0, 1, 2, 3$. All these sequences are independent, yielding that the log-likelihood of the whole training data is equal to the sum of log-likelihoods of individual sequences. We use all the sequences and the Baum-Welch algorithm to fit the MLE $\hat{\lambda} = (\hat{\pi}, \hat{A}, \hat{B})$ for the HMM. A Python function called `Data_preprocess` is coded to read the `train534.dat` into a `numpy` array. **All the implementations for HMM are coded in Python by myself**. Only the Python packages `numpy, time, matplotlib.pyplot`, and the `KFold` function in `sklearn.model_selection` are imported. In my codes, `M` stands for the number of states and all other variable namings follow Chen [1]. See my Python code for details.

2.1 Implementation Details

The `Forward` function:

- **Input**: Initial distribution `v` ($[N,]$ `numpy array`); Transition probability `P` ($[N, N]$ `numpy array`); Emission probability `E` ($[N, M]$ `numpy array`); Observations from HMM: `Obers` (an integer list or a $[T,]$ `numpy` array).
- **Output**: A tuple: (A `numpy` array for forward probabilities in all the time steps, a float point number for the likelihood value based on `Obsers`)
- **Implementation Details**: For the base case of iterations, I used `numpy.multiply` to compute the element-wise product of v and E. To compute the product $\alpha_t(j)a_{ij}b_j(O_{t+1})$, I applied `numpy.multiply` and `numpy.dot` to accelerate the multiplication.

The `Backward` function:

- **Input**: Same as the `Forward` function
- **Output**: A tuple: (A `numpy` array for backward probabilities in all the time steps, a float point number for the likelihood value based on `Obsers`)
- **Implementation Details**: To compute the product $a_{ij}b_j(O_{t+1})\beta_{t+1}(j)$, I also applied `numpy.multiply` and `numpy.dot` to accelerate the multiplication.

The `Baum-Welch` function:

- **Input**: Trainging set: `training`; The number of states: `M`; Initial distribution `v_0`; Transition probability `P_0`; Emission probability `E_0` (Initial values for EM); An indicator of whether the initial values for EM is provided: `init=True`; Threshold for the relative difference of log-likelihood between two iterations: `accuracy=1e-5`; The maximum number of iteration: `num_iter=10**5`; Binary variable for controlling the stopping criterion: `norm=True` (True: Use L2 norm for all parameters)
- **Output**: A tuple: `(v, P, E)` stands for (π, A, B)
- **Implementation Details**: We use the `numpy.random.rand` function to randomly initialize `(v,P,E)` and normalize them across rows. We apply `Forward` and `Backward` functions to obtain the forward and backward probabilities. Then $\gamma_t(i) = P(q_t = i|O_{1:T})$ is calculated via `numpy.multiply`. Based on this quantity, a single and double for loops are applied to update `E` and `P`. Finally, the `(v, P, E)` is normalized through rows before returning. In addition, the rows of the emission probability matrix are reordered, where the row with the largest variance (computed by `numpy.var`) is label $N-1$, etc. The rows and columns of the transition probability matrix are also reordered accordingly.
- **Remark**. By the independence assumption, the log-likelihood of the whole training set is equal to the summation of the log-likelihoods in each sequence, which can be computed from `Forward` or `Backward` functions. The stopping criterion here for the Baum-Welch algorithm is relative difference of log-likelihoods between two consecutive iterations, *i.e.*, $\frac{|L^{(k+1)}(\lambda|O)-L^{(k)}(\lambda|O)|}{|L^{(k)}(\lambda|O)|} \leq \epsilon$. In the whole project, we choose $\epsilon = 10^{-5}$.

The `Viterbi` function:

- **Input**: Same as the `Forward` function
- **Output**: A tuple: (A list storing the most likely sequence of states, a float point number for the maximum joint probability)
- **Implementation Details**: I initialize a $[M, T-1]$ `numpy` array `f` to store the optimal backtracking state for each hidden state at each time step. A double for-loop is applied to compute $\delta_t(i) = \max_{j=1,\ldots,N}[\delta_{t-1}(j)a_{ji}]b_i(O_t)$ and $\psi_t(i) = \arg\max_{j=1,\ldots,N}[\delta_{t-1}(j)a_{ji}]$.
- **Remark**. Note that when T is large, the rounding error of computing the probability can be problematic.[1] Thus, I move all the calculations to log-scale and use the recursive relation:

$$\tilde{\delta}_t(i) = \log b_i(O_t) + \max_{j=1,\ldots,N}[\tilde{\delta}_{t-1}(j) + \log a_{ji}].$$

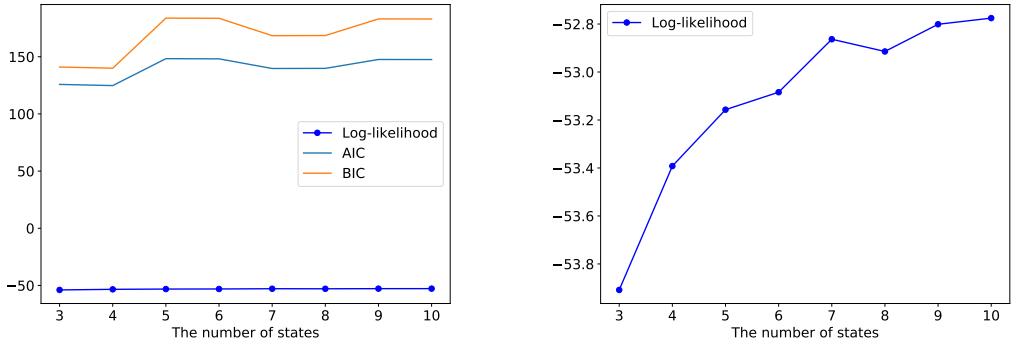
The `Predict` function:

- **Input**: Initial distribution `v` ($[N,]$ `numpy array`); Transition probability `P` ($[N, N]$ `numpy array`); Emission probability `E` ($[N, M]$ `numpy array`); Observations from HMM: `Obers` (an integer list or a $[T,]$ `numpy` array); Time step: `h=1`; An indicator of whether the Viterbi is used to predict the next hidden state: `viterbi=False`.
- **Output**: A tuple: (The next output of HMM, The next state, Output probabilities)
- **Implementation Details**: Following from Equation 2, we apply the `Forward` function to compute forward probabilities and then use `numpy.dot` twice to compute $\sum_{i=1}^{N}\sum_{j=1}^{N} b_i(O_{T+h})a_{ji}^{(h)}\alpha_T(j)$. The binary variable `viterbi` controls whether the `Viterbi` algorithm is used to construct the next hidden state. In the following prediction, we assume all the emission probabilities are nonzero and implement Equation 3 to predict the next hidden state.

As for the choice of the number of states, we implement **5-fold cross-validation**, where the training data is randomly split into 5 parts and each part has 200 sequences. For a whole 5-fold CV, one part of the training data is held out and we apply the `Baum-Welch` algorithm on the rest 800 sequences to fit the parameter $\lambda = (\pi, A, B)$. Then the `Backward`, `BIC`, and `AIC` functions are applied to compute the log-likelihood, BIC, and AIC on the hold-out 200 sequences. The candidate choices of the number of states range from 3 to 10.

2.2 Results

The plots of average log-likelihoods (the log-likelihood on the hold-out set divided by the number of sequences, *i.e.*, 200), BICs, and AICs versus the number of states are shown in Figure 1.



(a) 5-Fold CV Results

(b) Zoomed-in Log-likelihoods

Figure 1: Plots of Average Log-likelihoods, BICs, and AICs v.s. the Number of States

Figure 1 (b) verifies the claim that the log-likelihood will accrue as the number of parameters in HMM increases. Hence choosing the number of states in HMM based on the maximum value of log-likelihoods is not convincing. Figure 1 (a) indicates that the AIC and BIC reaches their minima when the number of states is 4. Since the BIC and AIC will gradually become larger as the number of states in HMM increases, we choose the number of hidden states to be **4**. In addition, we try different values of random seeds and similar curves will be produced. The 5-fold cross-validation, where the number of states ranges from 3 to 10, took 11h 23m 5s to run.

The initialization for the Baum-Welch algorithm is critical to its convergence towards the global maximum and the prediction accuracy of HMM. Thus, we apply the same 5-fold CV procedure again and use the first 39 observations in each sequence as inputs to predict the 40^{th} observation. It leads to the accuracy curve in Figure 2. The accuracy of HMM predictions attains the maximal when the random seed for the initialization is 301. We use this seed in subsequent training processes.

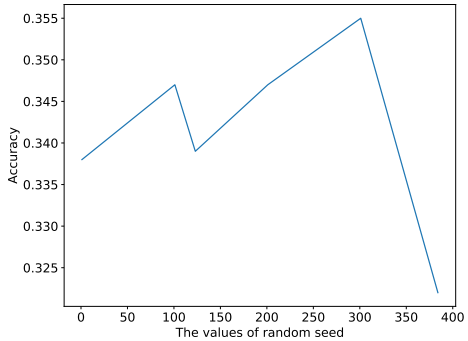


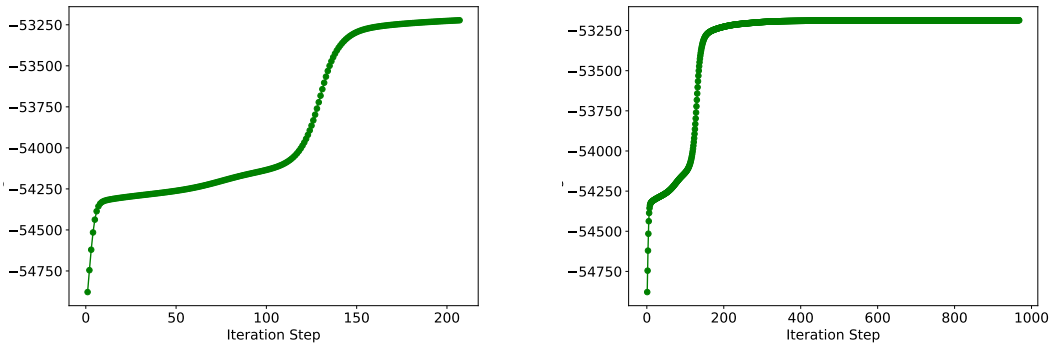
Figure 2: The Accuracy Curve for Different Random Seeds

Then we apply the `Baum-Welch` function on the whole training set (1000 sequences) to fit the parameter $\hat{\lambda} = (\hat{\pi}, \hat{A}, \hat{B})$ as follows (rounding to two decimals). The Baum-Welch algorithm on

the training set of 1000 sequences converged in 207 iterations and took 270.83 seconds to run. The learning curve of the Baum-Welch algorithm is shown in Figure 3.

$$\hat{\pi} = [0.49, 0.17, 0.20, 0.13]\,,$$

$$\hat{A} = \begin{array}{c} \text{States} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0.79 & 0.03 & 0.18 & 0.00 \\ 0.12 & 0.05 & 0.19 & 0.64 \\ 0.05 & 0.58 & 0.36 & 0.01 \\ 0.00 & 0.32 & 0.35 & 0.33 \end{pmatrix} \end{array}, \quad \hat{B} = \begin{array}{c} \text{States/Observations} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} \text{“0”} & \text{“1”} & \text{“2”} & \text{“3”} \end{array} \\ \begin{pmatrix} 0.50 & 0.04 & 0.36 & 0.10 \\ 0.41 & 0.47 & 0.08 & 0.04 \\ 0.07 & 0.27 & 0.66 & 0.00 \\ 0.06 & 0.08 & 0.03 & 0.83 \end{pmatrix} \end{array}.$$



(a) $\epsilon = 10^{-5}$ (b) $\epsilon = 10^{-8}$

Figure 3: The Learning Curves of the Baum-Welch Algorithm

Figure 3 shows that the observed log-likelihoods soars up in the first 10 iterations and then slow downs. Nevertheless, the increasing rate of log-likelihoods is accelerating from 120 to 160 iterations. Then the changes of log-likelihoods become negligible after 200 iterations. This result accidentally justifies the correctness of the choice of stopping criterion ($\epsilon = 10^{-5}$).

Then we use the `Viterbi` function to calculate the most likely sequence of states for all the data and store the results in a $n_0 \times T$ `numpy` array. The running time for the Viterbi algorithm on all the training data is 1.94 seconds, and 0.096 seconds on the test data.

As we did in selecting the random seed, we use the first 39 observations in each sequence of the training set as inputs to predict the 40^{th} observation. The accuracy on the training set is 0.355. The formulas for predicting the next output and state are given in Section 1.2.

In order to calculate the log-likelihood of the test set, we apply the `Forward` or `Backward` function 50 times and sum up all the log-likelihoods of the sequences in the test set. It turns out that the log-likelihood of the test set is -2708.648.

3 Conclusion

In this project we implement some standard ingredients of the hidden Markov model in Python and conduct 5-fold cross-validations for model selection. The most time-consuming part of HMM is the Baum-Welch algorithm. Hence in the future, we are supposed to leverage some gradient descent methods to accelerate the convergence of this EM algorithm. In addition, some preliminary explorations on the data like clustering will also enable us to figure out a more appropriate initialization for the Baum-Welch algorithm and facilitate its convergence.

References

[1] Y.-C. Chen. Lecture 9: Hidden markov model, 2018. URL `http://faculty.washington.edu/yenchic/18A_stat516/Lec9_HMM.pdf`.

[2] T. Hastie, R. Tibshirani, and J. Friedman. *The Element of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer-Verlag New York, second edition edition, 2009.

[3] D. Khiatani and U. Ghose. Weather forecasting using hidden markov model. In *2017 International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN)*, pages 220–225, Oct 2017. doi: 10.1109/IC3TSN.2017.8284480.

[4] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989. ISSN 0018-9219.