Hidden Markov Models, A Brief Tutorial

Stephen Bottos
bottos.steve@gmail.com
https://stevebottos.github.io/

November 14, 2018

Contents

1 Introduction **1**
1.1 Some Remarks 1
1.2 Key Concept and Notation Summary 2
1.3 A Simple Example 3

2 The Three Problems **5**
2.1 Evaluation 6
2.2 Decoding 7
2.3 Training 8

3 Conclusion **10**

1 Introduction

1.1 Some Remarks

Before beginning the discussion on on Hidden Markov Models (HMMs), there are a few remarks which I would like to get out of the way. First, tutorial was compiled by myself using information from two sources of literature on the subject, for details on each please see references [1] and [2]. I consider both of these sources to be quite solid, however after lengthy exposure to each I have found that there are certain elements that one may handle poorly while the other handles well in terms of capturing the intuition behind the HMM. What I have attempted to do here is adapt my own tutorial as such that the best of both can be found in one resource. All work and writing is original, in fact you'll find that I often use alternate notations to both sources.

A second remark is that, as much as I'd have liked to, I was not able to include figures to compliment my analytic derivations due to time constraints. I believe that [1] provides excellent graphical representations when necessary. On that note, the final remark is that

you'll notice the use of the **initial probability matrix**, π - which contains the probabilities of starting in any State allowed by the model - in [1] but not in [2]. This is because it is in fact not necessary, especially not within the context of this discussion, but can be included if one so chooses. As such, this parameter has been left out of this tutorial for the sake of simplicity, but it should be noted that its inclusion has little impact on the HMM (both in terms of complicating the ability to understand its mechanics and its performance). By the end of this tutorial, one should easily be able to expand their comprehension of HMMs to include the π matrix.

1.2 Key Concept and Notation Summary

I think that it would be best to present this section prior to discussing HMMs and walking through a simple example, to ensure that all key definitions and variables can be found in the same place.

- **Markov Property**. The Markov property
- M, the number of possible states belonging to the model
- N, the number of possible emission probabilities belonging to the model. For this tutorial, for simplicity's sake, we will assume there is a 1:1 relationship between the states and emission probabilities (ie: $N = M$)
- $\mathbf{o}^T$, the observation sequence $\{o(1), ..., o(T)\}$
- $\mathbf{s}^T$, the estimated hidden state $\{s(1), ..., s(T)\}$
- $\mathbf{A}$, the state transition probability matrix, having dimensions $[M \times M]$. For example,

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1M} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2M} \\ a_{11} & a_{12} & a_{13} & \dots & a_{3M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M2} & a_{M3} & \dots & a_{MM} \end{bmatrix}$$

- a_{ij}, the probability that $s(t) = j$ given that $s(t-1) = i$. Formally:

$$a_{ij} = P(s(t) = j | s(t-1) = i) \tag{1}$$

- $\mathbf{B}$, the emission probability matrix, having dimensions $[M \times M]$. For example,

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & \dots & a_{1M} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2M} \\ b_{11} & b_{12} & b_{13} & \dots & b_{3M} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{M1} & b_{M2} & b_{M3} & \dots & b_{MM} \end{bmatrix}$$

- b_{jk}, the probability that an observation $o(t) = k$ was generated by the state $s(t) = i$. Often denoted as $b_{jk}(t)$, with the addition of (t) being due to the face that we will later discuss the emission probabilities from two different points in time in the same formula. It is just included for clarity when it makes sense to do so, and omitted otherwise. Formally:

$$b_{jk}(t) = P(o(t) = k|s(t) = j) \tag{2}$$

1.3 A Simple Example

HMMs are powerful tools, commonly used in a wide range of applications from stock price prediction, to gene decoding, to speech recognition, which allow us to estimate a sequence of **states** - which are hidden from us (hence the name) - based on a series of **observations**, which are not hidden from us. This is best described with an example.

Let's say two best friends Seth and Evan have moved apart but still keep in touch via Skype briefly each day. Seth thinks it's exciting to try to guess what the weather is like at Evan's based on how Evan is dressed during their video call. There are no windows by Evan's computer, and as such, the true state - whether or not it is sunny or rainy let's say - is hidden from his friend. But, Seth does have some prior knowledge that he can use to arrive at a confident prediction regarding the weather at Evan's. Let's say that we have the following states and observations:

- Observation 1, Evan is wearing a rain jacket
- Observation 2, Evan is wearing sunglasses
- Observation 3, Evan is wearing a toque
- State 1, sunny weather
- State 2, rainy weather

Here, we have $M = 2$ states, and $N = 3$ possible observations. As such, $\mathbf{A}$ will be $[2 \times 2]$ and $\mathbf{B}$ will be $[2 \times 3]$. Note that for the actual derivations carried out in the remainder of this tutorial we assume that $N = M$ to avoid things getting cluttered. Now that that's established, let's talk about Seth's prior knowledge, which is:

- The probability of State 1 $\rightarrow$ State 1, of two sunny days in a row, is 0.6
- The probability of State 1 $\rightarrow$ State 2, of a rainy day occuring after a sunny day, is 0.4
- The probability of State 2 $\rightarrow$ State 2, of two rainy days in a row, is 0.7
- The probability of State 2 $\rightarrow$ State 1, of a rainy day occuring after a sunny day, is 0.3

... And the observations,

- The probability that Observation 1 was generated by State 1, that Evan is wearing a rain jacket while it is sunny, is 0.1

- The probability that Observation 2 was generated by State 1, that Evan is wearing sunglasses while it is sunny, is 0.6
- The probability that Observation 3 was generated by State 1, that Evan is wearing a toque while it is sunny, is 0.3
- The probability that Observation 1 was generated by State 2, that Evan is wearing a rain jacket while it is rainy, is 0.6
- The probability that Observation 2 was generated by State 2, that Evan is wearing sunglasses while it is rainy, is 0.2
- The probability that Observation 3 was generated by State 2, that Evan is wearing a toque while it is rainy, is 0.2

This scenario, described graphically, can be seen in Figure 1.

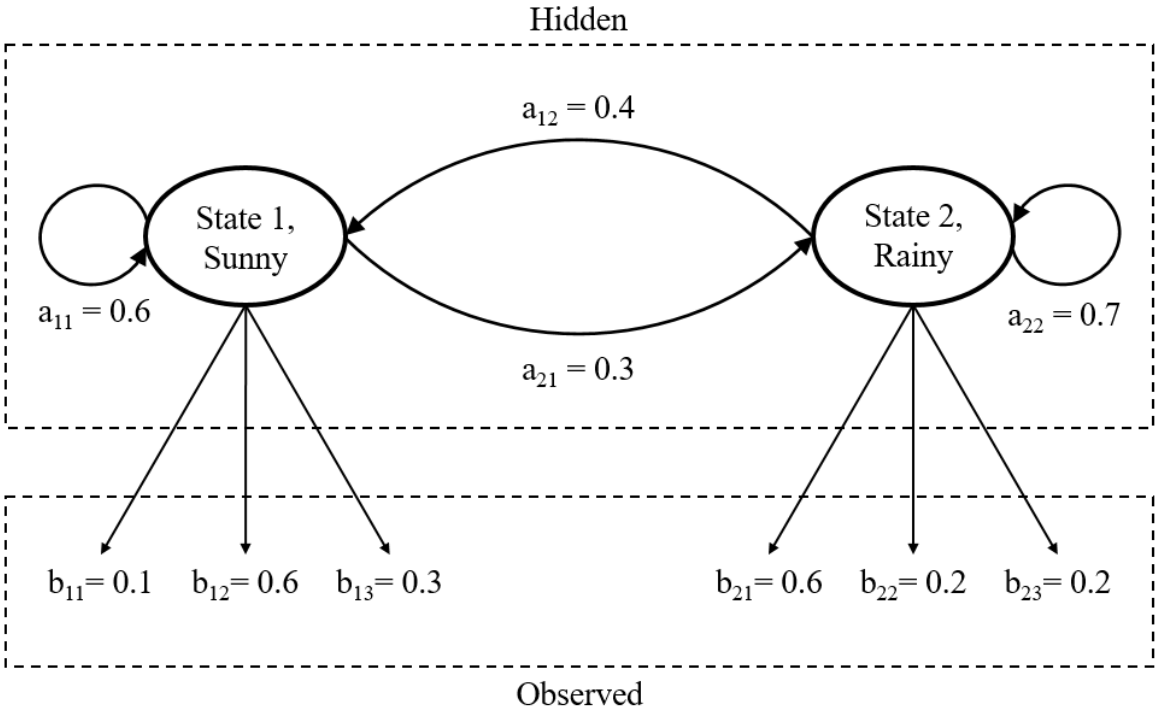

Figure 1: A graphical representation of our example.

Additionally, the **A** and **B** parameter matrices will look like so,

$$A = \begin{bmatrix} 0.6 & 0.4 \\ 0.7 & 0.3 \end{bmatrix}_{2\times 2} \quad B = \begin{bmatrix} 0.1 & 0.6 & 0.3 \\ 0.6 & 0.2 & 0.2 \end{bmatrix}_{2\times 3}$$

To demonstrate the result, I computed the most likely hidden state sequence for some made up sequence of observations for $t = 1 : 10$ using the Viterbi Algorithm, which will be discussed later. The output was as follows:

$$\mathbf{o}^{10} = \begin{bmatrix} 1 \\ 1 \\ 3 \\ 2 \\ 3 \\ 1 \\ 1 \\ 2 \\ 1 \\ 3 \end{bmatrix} \underset{\text{viterbi}}{\rightarrow} \mathbf{s}^{10} = \begin{bmatrix} 2 \\ 2 \\ 1 \\ 1 \\ 1 \\ 2 \\ 2 \\ 1 \\ 2 \\ 1 \end{bmatrix}$$

2 The Three Problems

There are three main problems that are associated with HMMs. Depending on the situation, it may be possible to solve just one

- The **Evaluation** problem addresses the question "what is the probability that this particular observation sequence was generated by this particular model". When dealing with one single HMM, one need not worry about solving the evaluation problem (however the derived formulae used in the evaluation problem are crucial to the solution of the remaining two). An excellent example of when the evaluation problem becomes important is in the application of speech recognition. To visualize, suppose that we have three HMMs - one trained to recognize the word "dog", one trained to recognize the word "cat", and one trained to recognize the word "apple". When the word "dog" is vocalized - interpreted by each HMM as an observation sequence - and evaluated then presumably, if the HMMs are trained sufficiently, the observation sequence produced by this vocalization will score a high probability of being associated with the HMM trained to recognize the word "dog", and a low probability for the other two. The algorithms of interest are the **Forward Algorithm** and the **Backward Algorithm**. Either can be used as both achieve the same thing.

- The **Decoding** problem addresses the question "given this observation sequence, what is the most likely corresponding sequence of hidden states". This differs from the evaluation problem in the sense that now we are considering a single HMM and attempting to uncover the information that is hidden from us based on the observation sequence rather than the probability that the observation sequence itself was generated by that particular HMM. The algorithm of interest is the **Viterbi Algorithm**

- The **Training** problem addresses the question "given a set of training observation sequences, and some initial guesses as to the model parameters, what are the most likely model parameters to fit this model to the training data and allow for accurate future predictions". The algorithm of interest is the **Baum Welch Algorithm**.

2.1 Evaluation

As was previously mentioned, the evaluation problem requires either the Forward Algorithm or the Backward Algorithm to solve, both achieve the same result. In order to begin to derive both algorithms, we must establish that we are given an observation sequence $\mathbf{o^T}$ as well as the model parameters $\mathbf{A}$ and $\mathbf{B}$. What we hope to determine is the probability that $\mathbf{o^T}$ was produced by the HMM (or could have been produced by the HMM) whose parameters are $\mathbf{A}$ and $\mathbf{B}$. This probability is denoted as $P(\mathbf{o^T})$ and can be expressed as follows:

$$P(\mathbf{o}^T) = \sum_{r=1}^{r_{max}} P(\mathbf{o}^T|\mathbf{s}_r^T)P(\mathbf{s}_r^T) \tag{3}$$

Here, each $\mathbf{s}_r^T$ represents its own vector of T hidden states, in some permutation, with each permutation denoted by r. In the general case, if we have c possible hidden states, and if each state sequence is T timesteps long, then the result will be a summation over $r_{max} = c^T$ possible terms. It may not be immediately obvious, but this calculation may easily grow to be incredibly computationally expensive. Thankfully, we can begin to simplify by recognizing that the **Markov Property** allows for the following:

$$P(\mathbf{o}^T|\mathbf{s}_r^T) = \prod_{t=1}^{T} P(o(t)|s_r(t)) \tag{4}$$

$$P(\mathbf{s}_r^T) = \prod_{t=1}^{T} P(s_r(t)|s_r(t-1)) \tag{5}$$

which allows us to express eq. 3 as such:

$$P(\mathbf{o}^T) = \sum_{r=1}^{r_{max}} \prod_{t=1}^{T} P(o(t)|s_r(t))P(s_r(t)|s_r(t-1)) \tag{6}$$

This still looks pretty nasty, but the interpretation is simple: the probability that we observe a particular state sequence of T observations is equal to the sum of the probability that some state produced the observation that we saw at time t, times the probability that we transitioned to the predicted state $s(t)$ from $s(t-1)$. It should be recognized that eq. 7 is simply:

$$P(\mathbf{o}^T) = \sum_{r=1}^{r_{max}} \prod_{t=1}^{T} b_{jk}(t)\, a_{ij} \tag{7}$$

which still isn't much more computationally friendly than the equation we started with. Thankfully, we can compute this recursively by implementing the Forward Algorithm or Backward Algorithm.

We'll begin our discussion on the Forward Algorithm with the introduction of the **Forward Variable**, which is defined as

$$\alpha_j(t) = \begin{cases} 0, & \text{if } t = 1,\ j \neq s(1) \\ 1, & \text{if } t = 1,\ j = s(1) \\ \left(\sum_{i=1}^{M} \alpha_i(t-1)\, a_{ij}\right) b_{jk}(t), & \text{if } t > 1 \end{cases} \tag{8}$$

And the algorithm is given as follows,

Algorithm 1 Forward Algorithm

1: **initialize** $t = 1$, $\alpha_j(1)$
2: **for** $t = 2 : T$
3: $\quad \alpha_j(t) := \left(\sum_{i=1}^{M} \alpha_i(t-1)a_{ij}\right) b_{jk}(t)$
4: **return** $P(\mathbf{o}^T) = \alpha_j(T)$, j = s(*T*)

Similar to the Forward Variable, we define the **Backward Variable** as

$$\beta_i(t) = \begin{cases} 0, & \text{if } t = T,\ i \neq s(T) \\ 1, & \text{if } t = 1,\ i = s(T) \\ \sum_{j=1}^{M} \beta_j(t+1)a_{ij}b_{jk}(t+1), & \text{if } t < T \end{cases} \tag{9}$$

And the algorithm is given as follows,

Algorithm 2 Backward Algorithm

1: **initialize** $t \leftarrow T$, $\beta_i(T)$
2: **for** $t = T - 1 : 1$
3: $\quad \beta_i(t) \leftarrow \sum_{j=1}^{M} \beta_j(t+1)a_{ij}b_{jk}(t+1)$
4: **return** $P(\mathbf{o}^T) \leftarrow \beta_i(1)$, i = s(*1*)

So, now we are able to describe $P(\mathbf{o}^T)$ in terms of $\alpha_j(T)$ or $\beta_i(1)$ - in other words, we must begin the computation for either variable at $t = 1$ and end at $t = T$ or vice-versa. In order to describe $P(\mathbf{o}^T)$ in terms of some timestep value t in the observation sequence, where t is not equal to 1 or T, then we must involve both the forward and backward variables as such:

$$P(\mathbf{o}^T) = \sum_{i=1}^{M} \alpha_i(t)\beta_i(t) = \sum_{i=1}^{M}\sum_{j=1}^{M} \alpha_i(t)\beta_j(t+1)a_{ij}b_{jk}(t+1) \tag{10}$$

This is an equality which we will work with later.

2.2 Decoding

In order to solve the decoding problem, we must infer the most likely hidden state corresponding to each observation in a given observation sequence through Maximum Likelihood Estimation. We'll define a variable which will contain the highest probability returned by the estimation up to a timestep value t as follows:

$$\delta_t(i) = \max_{s(1),\ldots,s(t)} [P(s(1), ..., s(t) = i, o(1), ..., o(t) = i] \tag{11}$$

In other words, $\delta_t(i)$ is the maximum measure of certainty, in the form of a probability, that a supposed sequence of hidden states (from $1 : t$) was generated observation sequence in

question (also from $1:t$), where $i = s(t)$. By induction, the following is also true, which is required for recursion:

$$\delta_{t+1}(j) = \max_i \left[\delta_t(i) a_{ij}\right] b_{jk}(t+1) \tag{12}$$

And, while the above equations return maximum probabilities, the actual object of interest is the inferred state at each timestep t which was responsible for maximizing the probability. We'll use the following to denote the vector that will eventually hold each most probable state, which we will later extract data from in order to obtain out most probable state sequence $\mathbf{s}^T$.

$$\mathbf{\Psi}_t(j) = \arg\max_i \left[\delta_{t-1}(i) a_{ij}\right] \tag{13}$$

Algorithm 3 Viterbi Algorithm

1: **initialize** $\delta_1(j) \leftarrow b_{jk}(1)$
2: **recurse**
3: **for** $t \leftarrow 2 : T$
4: $\delta_t(j) \leftarrow \max_i \left[\delta_{t-1}(i) a_{ij}\right] b_{jk}(t)$
5: $\mathbf{\Psi}_t(j) \leftarrow \arg\max_i \left[\delta_{t-1}(i) a_{ij}\right]$
6: **backtracking**
7: **for** $t \leftarrow T-1 : 1$
8: $s(t) \leftarrow \mathbf{\Psi}_{t+1}(j = s(t+1))$
9: **return** $\mathbf{s}^T$

2.3 Training

We've already mentioned that the method by which the training problem is solved is to implement the Baum Welch algorithm. The goal of the Baum Welch algorithm is to update some chosen initial model parameters populating the matrices $\mathbf{A}$ and $\mathbf{B}$ with estimates learned through training. This is accomplished by feeding training observation sequences to the algorithm. It is impossible to determine perfect parameters, but this is the most typical approach to establishing valid estimates. This algorithm involves "walking" through the observation sequence from $t = 1$ to $t = T$.

Recall from eq. 10 the manner in which we are able to express $P(\mathbf{o}^T)$ in terms of some point in time $t \neq 1$ or T in the sequence. We'll be using this in order to derive the formulas that we require to build the Baum Welch algorithm. We'll first define a new variable,

$$\xi_{ij}(t) = \frac{\alpha_i(t)\beta_j(t+1)a_{ij}\,b_{jk}(t+1)}{\sum_{i=1}^{M}\sum_{j=1}^{M}\alpha_i(t)\beta_j(t+1)a_{ij}\,b_{jk}(t+1)} \tag{14}$$

$$= \frac{P(s_i(t), s_j(t+1), \mathbf{o}^T)}{P(\mathbf{o}^T)}$$

Put verbally, this is simply

$$\xi_{ij}(t) = \frac{\text{probability that State } i \text{ and State } j \text{ are in the proper sequence}}{\text{probability that the sequence itself is correct}}$$

Since we'll be needing a second new variable shortly, we might as well also define the following:

$$\gamma_i(t) = \sum_{j=1}^{M} \xi_{ij}(t) \tag{15}$$

Using equations 14 and 15, we are able to infer four more important quantities, which we will only define verbally in order to avoid any unnecesarry confusion.

$$\sum_{t=1}^{T-1} \xi_{ij}(t) = \text{expected number of transitions from State } i \text{ to State } j \tag{16}$$

$$\sum_{t=1}^{T-1} \gamma_i(t) = \text{expected number of transitions from State } i \text{ in general} \tag{17}$$

If we change i to j and $T-1$ to T in eq. 17, we obtain

$$\begin{aligned} &\sum_{t=1}^{T} \gamma_j(t)\delta(o(t)=k) \\ &= \text{expected number of times that we arrive in State } j \text{ and are observing } o(t)=k \end{aligned} \tag{18}$$

Note that in eq 18 $\delta(o(t)=k)$ is used to denote the dirac delta function. Finally, we have

$$\sum_{t=1}^{T} \gamma_j(t) = \text{expected number of transitions to State } j \text{ in general} \tag{19}$$

Using these quantities, we can intuitively realize that it is possible to obtain the following, where $\hat{a}_{ij}$ and $\hat{b}_{jk}$ are the estimated parameters of the final, trained parameters a_{ij} and b_{jk}. From this point forward we will drop the (t) which was previously included in the notation of $b_{jk}(t)$:

$$\hat{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_{ij}(t)}{\sum_{t=1}^{T-1} \gamma_i(t)} \tag{20}$$

$$\hat{b}_{jk} = \frac{\sum_{t=1}^{T} \gamma_j(t)\delta(o(t)=k)}{\sum_{t=1}^{T} \gamma_j(t)} \tag{21}$$

With just a few more definitions, we will be able to fully discuss the Baum Welch algorithm.

- θ, algorithm convergence criterion
- $a_{ij,0}$, the initial guesses for each element in the state transition matrix
- $b_{ij,0}$, the initial guesses for each element in the emission probability matrix

Finally, the algorithm.

Algorithm 4 Baum Welch Algorithm

initialize $z \leftarrow 1$, $\hat{a}_{ij}(z=0) \leftarrow a_{ij,0}$, $\hat{b}_{ij}(z=0) \leftarrow b_{ij,0}$
recurse
  for $z \leftarrow z+1$
    $\hat{a}_{ij}(z) \leftarrow \hat{a}_{ij}(z-1)$ by eq. 20
    $\hat{b}_{ij}(z) \leftarrow \hat{b}_{ij}(z-1)$ by eq. 21
  until $\max_{i,j,k} \left[\hat{a}_{ij}(z) - \hat{a}_{ij}(z-1), \hat{b}_{ij}(z) - \hat{b}_{ij}(z-1)\right] < \theta$
return $a_{ij} = \hat{a}_{ij}(z), b_{ij} = \hat{b}_{ij}(z)$

3 Conclusion

This concludes my tutorial on Hidden Markov Models. I hope that you have found it informative! If you have read everything up to this point and are still lost then you're in the same position that I was when I began to study literature on HMMs. Thankfully, their actual implementation in Matlab is extremely simple - literally requiring about five lines of code if you already have trained parameters. Maybe ten if you are required to train parameters yourself. If you are interested in implementing an HMM in Python, below I'll provide the link to my website which contains a project that utlizes Python's Pomegranate package to build an HMM based on known parameters.

https://stevebottos.github.io/2018/08/13/Gaze-Tracking-System/

References

[1] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[2] R. O. Duda, P. E. Hart, D. G. Stork, *et al.*, "Pattern classification. 2nd," *Edition. New York*, vol. 55, 2001.