**COS 424: Interacting with Data**

Lecturer: Léon Bottou
Scribes: Joshua Kroll and Gordon Stewart

Lecture # 15 - Hidden Markov Models
13 April 2010

---

# Introduction

The classifiers we've looked at up to this point ignore the sequential aspects of data. For example, in homework 2 we used the bag-of-words model to classify Reuters articles. However, a lot of data is sequential. Hidden Markov models (HMMs) allow us to model this sequentiality.

## History of HMMs

HMMs were first described in the 1960s and 70s by a group of researchers at the Institute for Defense Analyses (Baum, Petrie, Soules, Weiss). Rabiner popularized HMM methods in the 1980s, especially through their applications in speech recognition. Ferguson, at the IDA, was the first to give an account of HMMs in terms of the 3 related problems of likelihood, decoding and learning.

## HMMs and Speech Recognition

The first major application of HMMs was in speech recognition. There are two major problems in this domain: data segmentation and recognition. Speech data is represented as a waveform where the frequency and amplitude of the sound vary with time. Segmentation involves splitting a waveform into smaller pieces that correspond to individual phonemes. Recognition is the task of determining which waveform subsequences correspond to which phonemes. Segmentation and recognition are the two major tasks of HMMs in other domains as well.

Slides 10-11. Speech recognition is complicated by coarticulation. Coarticulation occurs when two phonemes are voiced simultaneously in the transition from one phoneme to another due to the physical nature of the human vocal system. This phenomenon especially complicates speech segmentation.

# Hidden Markov Models

HMMs are well described in a paper by Lawrence Rabiner [1].

Hidden Markov Models are *generative* models, unlike the *discriminative* models we've seen up to this point. Discriminative models use observed data $x$ to model unobserved variables $y$, by modeling the conditional probability distribution $P(y|x)$ and then using this to predict $y$ from $x$. In a generative model, we randomly generate observable data using hidden parameters. Because a generative model has full probability distributions for all of the variables, it can be used to simulate the value of any variable in the model. For example, in the speech recognition example above, we are asking "what is the probability of the result given the state of the world?"

Markov models are based on a Markov state machine, which is a probabilisitic state machine that obeys the Markov assumption: the transition probabilities at time $t$ in state

$s_t$ only depend on $s_{t-1}$. Additionally, we require that the model is time-invariant, in the sense that the transition probabilities

$$P_\theta(s_t|s_{t-1}) \triangleq a_{s_t,s_t-1}$$

do not depend on the time parameter $t$ (that is, the transition probabilities from state to state are fixed and depend only on the prior state, without regard to time or the path taken through the model).

Further, at each time/state $s_t$, there is a probability to emit a symbol $x_t$. This probability only depends on $s_t$ and $s_{t-1}$, and is independent of time as before. In the case of a continuous HMM, we say that $P_\theta(x_t|s_t = s)$ is distributed according to some distribution $\mathcal{N}(\mu_s, \Sigma_s)$ which depends only on the state (and possibly the prior state). In a discrete HMM, we have an alphabet of emission symbols $\mathcal{X}_c$ for each cluster $c$ in the data and we write $P_\theta(x_t \in \mathcal{X}_c|s_t = s) \triangleq b_{cs}$.

## The Ferguson Problems

Rabiner explains that HMMs can be used effectively if we can solve three problems:

1. **Likelihood** Given a specific HMM, what is the likelihood of an observation sequence? That is, can we efficiently calculate

   $$P_\theta(x_1 \dots x_T) = \sum_{s_1 \dots s_T} P(x_1 \dots x_T, s_1 \dots s_T)$$

   where $s_T$ is a possible end state. Note that on the right we have just marginalized the probability of observing a sequence over the set of allowable sequences (i.e. valid transitions which end in a valid end state).

2. **Decoding** Given a sequence of observations and an HMM, what is the most probable sequence of hidden states? That is, calculate

   $$\arg\max_{s_1 \dots s_T} P_\theta(s_1 \dots s_T|x_1 \dots x_T) = \arg\max_{s_1 \dots s_T} P_\theta(s_1 \dots s_t, x_1 \dots x_T)$$

   Note that the argmax on the right is the same as on the left because the values themselves only differ by an exogenous factor $1/P(x_1 \dots x_T)$

3. **Learning** Given an observation sequence, learn the parameters and probability distributions which maximize performance. If we knew $s_1 \dots s_T$ this would be easy; we could just compute

   $$\max_\theta \sum_{s_1 \dots s_T} P_\theta(s_1 \dots s_T) P_\theta(x_1 \dots x_T|s_1 \dots s_T)$$

   since by Bayes' theorem this effectively maximizes the probability of getting the right answer for a given observation:

   $$P_\theta(s_1 \dots s_T|x_1 \dots x_T) P_\theta(x_1 \dots x_T)$$

The idea of using these three problems to organize thinking about HMMs is due to Jack Ferguson of IDA, again according to Rabiner [1]. Thus, we call them the Ferguson problems. We will solve each of these problems below.

**Likelihood**

We'd like to compute:

$$L(\theta) \triangleq P_\theta(x_1 \dots x_T) = \sum_{s_1 \dots s_T} P(x_1 \dots x_T, s_1 \dots s_T)$$

However, we can rewrite this as:

$$L(\theta) = \sum_{s_1 \dots s_T} \prod_{t=1}^{T} a_{s_{t-1} s_t} P_\theta(x_t | s_t)$$

The number of terms in this sum is exponential in $T$ (as before, we mean the sum to run only over sequences of states which have $s_T$ as a valid end state). This is too costly to compute directly. However, we can rewrite it by factoring into something we can compute efficiently.

$$\begin{aligned}
\forall 1 \leq t \leq T, L(\theta) \triangleq P_\theta(x_1 \dots x_T) &= \sum_i P_\theta(x_1 \dots x_T, s_t = i) \\
&= \sum_i P_\theta(x_1 \dots x_t, s_t = i) P_\theta(x_{t+1} \dots x_T | x_1 \dots x_t, s_t = i) \\
&= \sum_i \underbrace{P_\theta(x_1 \dots x_t, s_t = 1)}_{\triangleq \alpha_t(i)} \underbrace{P_\theta(x_{t+1} \dots x_T | s_t = i)}_{\triangleq \beta_t(i)}
\end{aligned}$$

In the first step we are just marginalizing over states. In the second, we break up the probability into the joint probability over the observation up to time $t$, $x_1 \dots x_t$ and the state $s_t$ at time $t$ and the conditional probability of the observation after time $t$ (until the end time $T$) on the observation up to time $t$ and the state $s_t$ at time $t$. Finally, in the third step, we use the Markov assumption to note that the probability of observations after time $t$ only depend on the state $s_t$ at time $t$.

Now, we can get a recursive definition for $\alpha_t(s_t)$. This will yield an algorithm for calculating the $\alpha_t(s_t)$:

$$\begin{aligned}
\alpha_t(s_t) &= P_\theta(x_1 \dots x_t, s_t) \\
&= \sum_{s_{t-1}} P_\theta(x_1 \dots x_t, s_t, s_{t-1}) \\
&= \sum_{s_{t-1}} P_\theta(x_1 \dots x_{t-1}, s_{t-1}) P_\theta(s_t | x_1 \dots x_{t-1}, s_{t-1}) P_\theta(x_t | x_1 \dots x_{t-1}, s_{t-1}, s_t) \\
&= \sum_{s_{t-1}} \alpha_{t-1}(s_{t-1}) a_{s_{t-1} s_t} P_\theta(x_t | s_t)
\end{aligned}$$

Similarly we can get a recursive definition for $\beta_t(s_t)$, but the recursion is flipped:

$$\begin{aligned}
\beta_{t-1}(s_{t-1}) &= P_\theta(x_t \dots x_T | s_{t-1}) \\
&= \sum_{s_t} P_\theta(x_t \dots x_T | s_{t-1}, s_t) P_\theta(s_t | s_{t-1}) \\
&= \sum_{s_t} P_\theta(x_{t+1} \dots x_T | s_{t-1}, s_t) P_\theta(x_t | x_{t+1} \dots x_T, s_{t-1}, s_t) P_\theta(s_t | s_{t-1}) \\
&= \sum_{s_t} \beta_t(s_t) a_{st-1 s_t} P_\theta(x_t | s_t)
\end{aligned}$$

We could have gotten the same result by an equivalent derivation that only relies on the distributive law:

$$\begin{aligned} L(\theta) &\triangleq P_\theta(x_1 \dots x_T) = \sum_{s_1 \dots s_T} \prod_{t=1}^{T} a_{s_{t-1}s_t} P_\theta(x_t|s_t) \\ &= \sum_{s_t} \left( \left( \underbrace{\sum_{s_1 \dots s_{t-1}} \prod_{t'=1}^{t} a_{s_{t'-1}s_{t'}} P_\theta(x_{t'}|s_{t'})}_{\triangleq \alpha_t(s_t)} \right) \times \left( \underbrace{\sum_{s_{t+1} \dots s_T} \prod_{t'=t+1}^{T} a_{s_{t'-1}s_{t'}} P_\theta(x_{t'}|s_{t'})}_{\triangleq \beta_t(s_t)} \right) \right) \end{aligned}$$

Now, we can get a recursive definition by:

$$\begin{aligned} \alpha_t(s_t) &= \sum_{s_1 \dots s_{t-1}} \prod_{t'=1}^{t} a_{s_{t'-1}s_{t'}} P_\theta(x_{t'}|s_{t'}) \\ &= \sum_{s_{t-1}} P_\theta(x_t|s_t) a_{s_{t-1}s_t} \sum_{s_1 \dots s_{t-2}} \prod_{t'=1}^{t-1} a_{s_{t'-1}s_{t'}} P_\theta(x_{t'}|s_{t'}) \\ &= \sum_{s_{t-1}} \alpha_{t-1}(s_{t-1}) a_{s_{t-1}s_t} P_\theta(x_t|s_t) \end{aligned}$$

We can similarly get a recursive definition for the $\beta_t(s_t)$ in this way.

It's worthwhile noting that we can also get a derivation via the chain rule:

$$\frac{\partial L}{\partial \alpha_{t-1}} = \beta_{t-1} = \left( \frac{\partial L}{\partial \alpha_t} \right)^\top \left( \frac{\partial \alpha_t}{\partial \alpha_{t-1}} \right) = \beta_t^\top \frac{\partial \alpha_t}{\partial \alpha_{t-1}}$$

All this yields a simple algorithm that progresses forward through the model. We initialize $\alpha_0(i) = \mathbb{1}\{i = \text{Start}\}$ and then set:

$$\alpha_t(i) = \sum_j \alpha_{t-1}(j) a_{ji} P_\theta(x_t|s_t = i)$$

Once we have these, we can initialize the $\beta$ values by $\beta_T(i) = \mathbb{1}\{i \in \text{End}\}$ and then we know from our initial derivation that the likelihood is just:

$$P_\theta(x_1 \dots x_T) = \sum_i \alpha_T(i) \beta_T(i) = \sum_{i \in \text{End}} \alpha_T(i)$$

**Decoding**

We'd like to compute the most likely set of hidden states. Noting that $\max(ab, ac) = a \max(b, c)$ for $a, b, c \geq 0$, we can write:

$$\begin{aligned} \alpha_t(i) &\triangleq \sum_{s_1 \dots s_{t-1}} \prod_{t'=1}^{t} a_{s_{t'-1}s_{t'}} P_\theta(x_{t'}|s_{t'}) \\ \alpha_t^*(i) &\triangleq \max_{s_1 \dots s_{t-1}} \prod_{t'=1}^{t} a_{s_{t'-1}s_{t'}} P_\theta(x_{t'}|s_{t'} \end{aligned}$$

This leads to a very natural algorithm, called the Viterbi algorithm, to find the most likely hidden states. First, we let $\alpha_0^*(i) = \mathbb{1}\{i = \text{Start}\}$ and then calculate:

$$\alpha_t^*(i) = \max_j \alpha_{t-1}^*(j) a_{ji} P_\theta(x_t | s_t = i)$$

Then we can calculate a decoding as:

$$\max_{s_1 \dots s_T} P_\theta(s_1 \dots s_T, x_1 \dots x_T) = \max_{i \in \text{End}} \alpha_T^*(i)$$

We can think of this as a backtracking: from each state/time-step combination, we have a maximum probability over prior state/time-step combinations. By following this most likely path backwards, we can construct the most likely set of hidden states. A diagram of this is on slide 23.

**Learning**

We have a set of observations for something we'd like to model, which are of the form $X = x_1 \dots x_T$. Learning would be easy if we knew what states to associate with each observation $S = s_1 \dots s_T$. Since we don't (they're "hidden" in the model), we'll have to do something else.

We'll use expectation maximization to find the right decomposition of the likelihood that we can learn. We already know how to, given $X$, guess a distribution $Q(S|X)$ using the decoding algorithm we saw above. Now, regardless of $Q$, we know:

$$\log L(\theta) = \mathcal{L}(Q, \theta) + \mathcal{D}(Q, \theta)$$

where:

$$\begin{aligned} \mathcal{L}(Q, \theta) &= \sum_{s_1 \dots s_T} Q(S|X) \log \frac{P_\theta(S) P_\theta(X|S)}{Q(S|X)} \\ \mathcal{D}(Q, \theta) &= \sum_{s_1 \dots s_T} Q(S|X) \log \frac{Q(S|X)}{P_\theta(S|X)} \end{aligned}$$

The first of these is easy to maximize. The second is a Kullback-Leibler divergence, which we've seen before as information gain.

Let's see how we get there. First:

$$\begin{aligned} \mathcal{L}(Q, \theta) &= \sum_{s_1 \dots s_T} Q(S|X) \log \frac{P_\theta(S) P_\theta(X|S)}{Q(S|X)} \\ &= \sum_{s_1 \dots s_T} Q(S|X) \left[ \sum_t \log a_{s_{t-1} s_t} + \sum_t \log P_\theta(x_t | s_t) - \log Q(S|X) \right] \end{aligned}$$

Now, the $a_{ij}$ are probabilities so $\sum_j a_{ij} = 1$. This means that at the optimum, we have the following relation:

$$\frac{\partial \mathcal{L}}{\partial a_{ij}} = \sum + s_1 \dots s_T Q(S|X) \sum_t \frac{\mathbb{1}\{s_{t-1} = i\} \mathbb{1}\{s_t = j\}}{a_{ij}} = K_i$$

and this implies:

$$
\begin{aligned}
a_{ij} &\propto \sum_{s_1 \ldots s_T} Q(S|X) \sum_{t=1}^{T} \mathbb{1}\{s_{t-1} = i\}\mathbb{1}\{s_t = j\} \\
&\propto \sum_{t=1}^{T} \sum_{s_1 \ldots s_T} Q(S|X)\mathbb{1}\{s_{t-1} = i\}\mathbb{1}\{s_t = j\} \\
&\propto \sum_{t=1}^{T} Q(s_{t-1} = i, s_t = j | x_1 \ldots x_T) \\
&\propto \sum_{t-1}^{T} Q(s_{t-1} = i, s_t = j, x_1 \ldots x_T) \\
&\propto \sum_{t-1}^{T} \left( \underbrace{Q(x_1 \ldots x_{t-1}, s_{t-1} = i)}_{\alpha_{t-1}(j)} \underbrace{Q(s_t = j | s_{t-1} = i, \cdots)}_{a_{ij}} \right) \left( \underbrace{Q(x_t | s_t = j, \cdots)}_{P_\theta(x_t|s_t)} \underbrace{Q(x_{t+1} \ldots x_T | s_t = j, \cdots)}_{\beta_t(i)} \right)
\end{aligned}
$$

To compute this, we do not need to store $Q(S|X)$, which would be too expensive. Instead, we only need to store $\alpha_t(s), \beta_t(s)$ and the number $B_t(s) \triangleq P_\theta(x_t|s_t = s)$ for all $t$ and $s$. And this is tractable, being about the size of the model times the number of time steps in its longest path.

This yields an efficient algorithm for learning, similar to the forward algorithm from before. It works like this:

- E-Step. Here we have 3 separate steps
  1. Emission: $\forall t \forall i \, B_t(i) = P_\theta(x_t|s_t = i)$
  2. Forward Pass: This is the same as our earlier forward algorithm. Initialize $\alpha_0(i) = \mathbb{1}\{i = \text{Start}\}$ and then calculate for $t = 1 \ldots T$:
  $$\forall i \, \alpha_t(i) = \sum_j \alpha_{t-1}(j) a_{ij} B_t(i)$$
  3. Backward Pass: This is exactly analogous to the step above and the forward algorithm from before. We initialize $\beta_T(i) = \mathbb{1}\{i \in \text{End}\}$ and then calculate for $t = T \ldots 1$:
  $$\forall i \, \beta_{t-1}(i) = \sum_j \beta_t(j) a_{ij} B_t(j)$$
- M-Step. In this step, we use the following Baum-Welch formulas:
  - For a continuous HMM:
  $$
  \begin{aligned}
  a_{ij} &\leftarrow \frac{\sum_t \alpha_{t-1}(i) a_{ij} B_t(j) \beta_t(j)}{\sum_t \alpha_{t-1}(i) \beta_{t-1}(i)} \\
  \mu_i &\leftarrow \frac{\sum_t \alpha_{t-1}(i) \beta_{t-1}(i) x_t}{\sum_t \alpha_{t-1}(i) \beta_{t-1}(i)} \\
  \Sigma_i &\leftarrow \frac{\sum_t \alpha_{t-1}(i) \beta_{t-1}(i) x_t x_t^\top}{\sum_t \alpha_{t-1}(i) \beta_{t-1}(i)} - \mu_i \mu_i^\top
  \end{aligned}
  $$

– For a discrete HMM:

$$\begin{aligned} a_{ij} &\leftarrow \frac{\sum_t \alpha_{t-1}(i) a_{ij} B_t(j) \beta_t(j)}{\sum_t \alpha_{t-1}(i) \beta_{t-1}(i)} \\ b_{cs} &\leftarrow \frac{\sum_t \alpha_{t-1}(i) \beta_{t-1}(i) \mathbb{1}\{x_t \in \mathcal{X}_c\}}{\sum_t \alpha_{t-1}(i) \beta_{t-1}(i)} \end{aligned}$$

## Segmentation and Recognition

Slides 33 - 38. Assume you have an observation sequence $X = x_1, x_2, \ldots, x_T$ and a set of categories $C$. The recognition problem is to assign the most likely category $c \in C$ to $X$.

To solve the problem, train a model $W_c$ for each category $c \in C$ on the set of training observation sequences using the Baum-Welch algorithm. Then, build a prior probability distribution $P(C = c)$. Using Bayes' rule, we know that

$$P(C|x_1, x_2, \ldots, x_T) = \frac{P(X|C)P(C)}{P(X)}$$

so we can use the forward algorithm to evaluate

$$\arg\max_c P_\theta(x_1, x_2, \ldots, x_T|W_c)$$

This expression gives us the category $c$ that maximizes the probability of the observation sequence $x_1, x_2, \ldots, x_T$ in model $W_c$.

It's also possible to perform segmentation and recognition at the same time. More formally, the problem is to split a sequence $X$ into segments and simultaneously assign a category $c \in C$ to each segment. One way to solve this problem is to build models for each category $c$ and combine them into a "supermodel" that can then be trained using the forward-backward algorithm. Finite state transducers provide a more general solution to this problem of combining models.

**Note on n-gram language models.** $n-$gram language models model the probability of an emission sequence $X = x_1, \ldots, x_{m-1}, x_m$ as the product of the probabilities of the $n-$length subsequences of $X$. For example, the probability of the sequence $x, y, z$ in a bigram model would be

$$P(x|empty)P(y|x)P(z|y).$$

These probabilities can be computed from frequency data. $n-$gram language models are often used to build the prior probability distributions used to train HMMs.

## References

[1] LR Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.