Hidden Markov Models

EECS 126 (UC Berkeley)

Spring 2019

A Markov Chain that has its states hidden (or latent) is called a **Hidden Markov Model** (HMM). The X_i are state variables and belong to a state space $\mathcal{X}$ (discrete), while the Y_i are observations with $y_i \in \mathcal{Y}$, where $\mathcal{Y}$ may or may not be discrete.

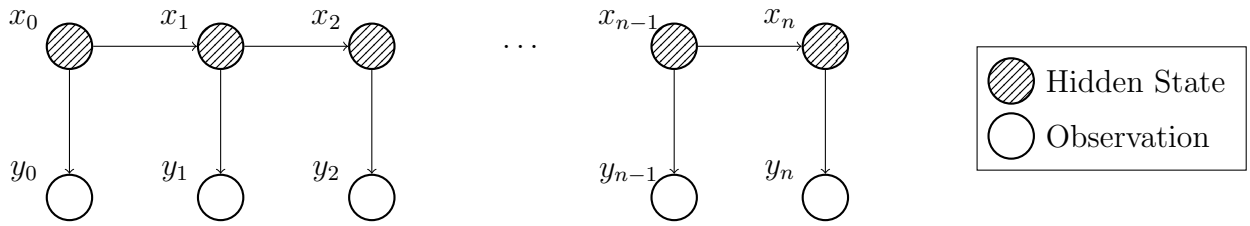

Figure 1: Graphical Model for HMMs

Due to the structure of the HMM as indicated by the above graphical model, we can simplify our probabilities a lot. This will be very useful!

Example: If we had $n = 2$, then the joint probability $\mathbb{P}(x_0, y_0, x_1, y_1)$ would be

$$\mathbb{P}(x_0, y_0, x_1, y_1) = \mathbb{P}(x_0)\mathbb{P}(y_0|x_0)\mathbb{P}(x_1|x_0, y_0)\mathbb{P}(y_1|x_1, x_0, y_0) = \mathbb{P}(x_0)\mathbb{P}(y_0|x_0)\mathbb{P}(x_1|x_0)\mathbb{P}(y_1|x_1).$$

In general, for n states and corresponding observations (excluding state 0), we have

$$\mathbb{P}(x_0, x_1, \ldots, x_n, y_0, y_1, \ldots, y_n) = \pi_0(x_0)Q(x_0, y_0)P(x_0, x_1)Q(x_1, y_1)\ldots P(x_{n-1}, x_n)Q(x_n, y_n)$$

where π_0 specifies the distribution for our initial state, Q models our transition probabilities between hidden states and observations, and P models transitions between hidden states.

HMMs can be used for inference in the following forms.

- **Filtering**: We feed in $Y_0, Y_1, \ldots, Y_T$ to our filter and want to find $\hat{X}_T$, the last hidden state. Some examples are tracking positions in real time or monitoring the current health of a patient given symptoms $\{Y_0\}_{i=0}^T$.

- **Prediction**: We feed in $Y_0, Y_1, \ldots, Y_T$ and want to predict $\hat{Y}_{T+1}$. Some examples are radar tracking, stock price predictions, or predictive coding.
- **Smoothing**: We feed in $Y_0, Y_1, \ldots, Y_T$ and want to find $\hat{X}_t$ for a choice of $t \leq T$. Some examples are inferring the cause of a car-crash or "post-mortem" analysis.
- **MLSE (Maximum Likelihood Sequence Estimation)**: We feed in $Y_0, Y_1, \ldots, Y_T$ and want to find the most likely *sequence* $\hat{X}_0, \hat{X}_1, \ldots, \hat{X}_T$ that explains our observations. In other words, we want $MAP[X^n | Y^n = y^n]$, to infer the best sequence of (hidden) states that best explains the observed sequence. This differs from smoothing, where we only care about maximizing over a single hidden state.

  Some example applications of finding the MLSE are speech recognition, auto-correction, and convolutional coding with the Viterbi algorithm. In speech recognition, we observe (or listen to) the sounds, and our goal is to find the most likely sequence of words corresponding to the sounds.

In this note, we will focus on MLSE.

The Viterbi Algorithm

The MLSE is given by

$$\begin{aligned} x^{n^*} &= \arg \max_{x^n \in \mathcal{X}^n} \mathbb{P}[X^n = x^n | Y^n = y^n] \\ &= \arg \max_{x^n \in \mathcal{X}^n} [\pi_0 Q(x_0, y_0) P(x_0, x_1) Q(x_1, y_1) \ldots P(x_{n-1}, x_n) Q(x_n, y_n)] \\ &= \arg \max_{x^n \in \mathcal{X}^n} \left[\log \pi_0(x_0) Q(x_0, y_0) + \sum_{m=1}^{n} \log \left[P(x_{m-1}, x_m) Q(x_m, y_m) \right] \right], \end{aligned}$$

where the last equation is obtained by taking a log (which is allowed because log is a monotonically increasing function). To make the expression more compact, let's define

$$\begin{aligned} d_0(x_0) &= -\log \pi_0(x_0) Q(x_0, y_0) \\ d_m(x_{m-1}, x_m) &= -\log \left[P(x_{m-1}, x_m) Q(x_m, y_m) \right]. \end{aligned}$$

Note that all the d_i's are positive. Thus, we have the following.

MLSE Estimate: Finding the maximum likelihood sequence estimate reduces to solving the following optimization problem:

$$x^{n^*} = \arg \min_{x^n} \left[d_0(x_0) + \sum_{m=1}^{n} d_m(x_{m-1}, x_m) \right].$$

Example: Say you're at a "nearly honest casino" which uses a fair die most of the time, but switches to a loaded die occasionally. We can model their transitions between the fair die (F) and the loaded die (L) as the Markov Chain in Figure 2.

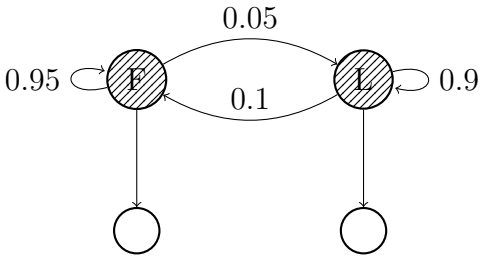

Figure 2: "Nearly Honest" Casino

Additionally, we know that for the fair die, the probability of each outcome is equally likely, so $\mathbb{P}(F = i) = \frac{1}{6}$ for $i = 1, \ldots, 6$. For the loaded die, we know that $\mathbb{P}(L = 6) = \frac{1}{2}$ and $\mathbb{P}(L = 1) = \cdots = \mathbb{P}(L = 5) = \frac{1}{10}$ so that the die is biased towards rolling a 6.

Given an observed sequence of die rolls (say $6, 6, 1, 6, 2, \ldots$), we want to infer the most likely sequence of "hidden" states (say $F, F, L, F, L, \ldots$). We can use a technique called a "trellis" diagram, which looks like this (green nodes correspond to the fair die, while red nodes correspond to the loaded die):

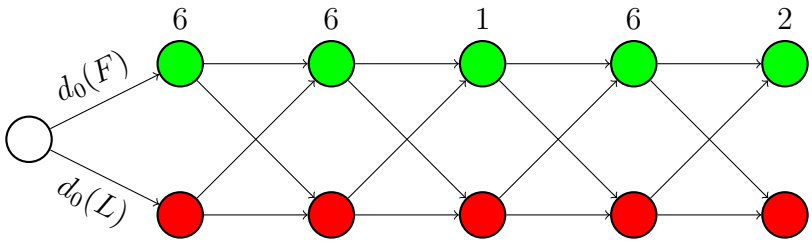

Figure 3: Trellis Diagram

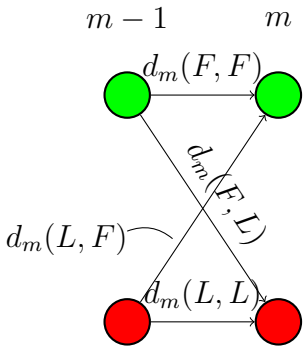

Figure 4: One "Stage" of the Trellis Diagram

To find the minimum length path from stage 0 to stage n, we need a good shortest path algorithm. Bellman-Ford seems like a good choice, especially since its dynamic programming

nature lends itself very well to such a calculation (recall that shortest paths on DAGs are best solved by dynamic programming). This technique of filling out the diagram and finding the MLSE using a dynamic programming method was first discovered by Viterbi and is referred to as the **Viterbi Algorithm**.

First, we need to calculate the edge weights, that is, all the d_m's. For illustrative purposes, we assign nice numbers for the edge weights in the trellis diagram below. In reality, we would perform the following computation for each m, where the P's are given by the Markov chain in Figure 2 and the Q's are given by the probabilities of each side for the fair and loaded die:

$$\begin{aligned} d_m(F,F) &= -\log\left[P(F,F)Q(F,Y_{m+1})\right] \\ d_m(F,L) &= -\log\left[P(F,L)Q(L,Y_{m+1})\right] \\ d_m(L,F) &= -\log\left[P(L,F)Q(F,Y_{m+1})\right] \\ d_m(L,L) &= -\log\left[P(L,L)Q(L,Y_{m+1})\right] \end{aligned}$$

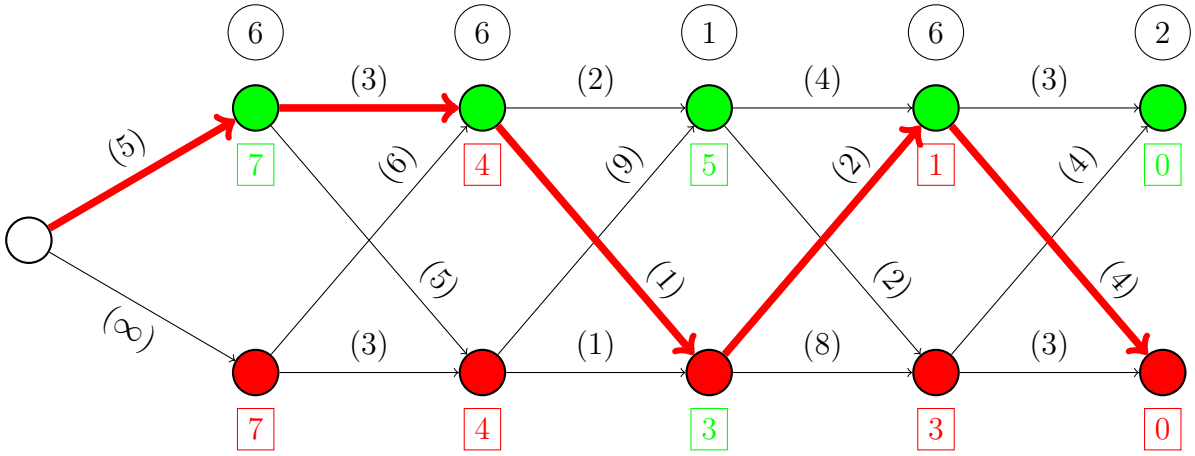

Figure 5: Obtaining the MLSE Estimate

The circled numbers above each stage are the observations, and the numbers in parentheses along every edge are the weights. The boxed numbers are the shortest path values from that node to the final stage; green numbers represent transitions to a fair die, while red numbers represent transitions to a loaded die. Our initial transition from the origin to L has distance infinity because we're told that the casino will start with a fair die.

If we work our way back through this diagram, we see that the MLSE estimate is (F, F, L, F, L).

Finally, we can do a quick analysis of how much time each of these methods take. The cost of populating the trellis is $O(N^2 n)$, where N is the number of states and n is the number of stages. If we have a populated trellis, it only takes $O(Nn)$ to find the shortest path (since we only have one node at each stage to consider). In contrast, a naive algorithm that iterates over all possible sequences takes $O(N^n)$ time; we have turned a computationally infeasible problem into a pretty efficient one!