CS 262 Lecture 5
Scribed by Yu Bai (SUID: 4905352) 01/2005

1. Motivation

Identifying distantly related homologs is a difficult problem, primarily because sequence identity between them is sparse. Although the traditional pairwise alignment algorithms(e.g. BLAST, Smith-Waterman) have been devised to discriminate relatively harmless differences, penalizing common or conservative changes less than radical ones, Hidden Markov Models (HMM) offer a more systemic, family-based statistical approach to describe the consensus between the sequences. It has been one of the most important tools in genome analysis and structure prediction. There are also wide HMM applications in speech recognition.

2. Outline

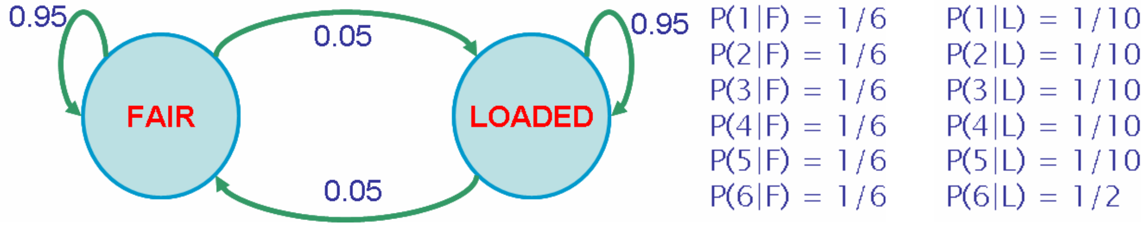
The lecture illustrated the basic ideas of Hidden Markov Model with a dishonest casino example, following the introduction to the definition and features of HMMs. The three main questions on building a HMM were reviewed and the algorithms for *decoding* and *evaluation* a HMM model were discussed.

3. Basics

Hidden Markov Model is a (customizable) statistical model that describes a series of observations by a hidden stochastic process and defines a probability distribution over possible sequences. The lecture introduced the HMM as following:

3.1 Dishonest Casino example

A casino game exists in which a player wins $1 for each roll that lands on a 1,2,3,4, or 5, and loses $2 if the roll lands on 6. With a fair die, the probability that the casino dealer wins (i.e., rolls a 6) is 1/6, and it is independent from the sequence of the previous rolls. However, the casino owner is a “dishonest” businessman and introduces a second die which is *loaded*, and consequently has a higher probability, 50%, of landing on a 6 and 10% each for other numbers. The casino dealer plays this trick carefully for not being caught, so he fetches back and forth between the fair and loaded dice every 20 rounds. (or say, 5% exchanging probability). The game can be represented as the model below:



Where P(i|F) and P(i|L) are the probability of generating number i by Fair and Loaded dice, respectively. i = {1,2,3,4,5,6}

The above diagram is an example of Hidden Markov Model. The reason that it is called “Hidden” is because normally we only observe a sequence of dice rolling, e.g.

1, 2, 6, 4, 3, 6, 5, 2, 6, 6, 4, 1, 3, 6, 6, 6, 6, 6, 6, 6, 6, 5, 4, 6, 1, 6. We cannot tell which state each rolling is in, e.g. subsequence 6, 6, 6, 6, 6, 6 may happen using the loaded dice or it can happen using the fair dice even though the later case has less probability. The state is "hidden" from the sequence, e.g. we cannot determine the sequence of states from the given sequence. Hence, it is "Hidden" Markov Model.

3.2 Definition of HMM

A Hidden Markov Model contains

- *Alphabet* $\Sigma = \{ b_1, b_2, \cdots, b_M \}$, i.e. all possible observations in a process
- A set of *States* $Q=\{1,\ldots K\}$
- Statistical parameters connecting *Alphabet* to *States* and *States* to *States*, i.e.
  - Start probability a_{0i}, $\Sigma_{(i=1..K)}\, a_{0i} =1$
  - Transition probabilities between any two *States*

    a_{ij}= transition probability from *State* i to *State* j, $\Sigma_{(j=1..K)}\, a_{ij} =1$
  - Emission probability $e_i(b) = P(x_i = b \mid \pi_i = k)$

    $e_i(b_1) + \cdots + e_i(b_M) = 1$, for all states i = 1…K

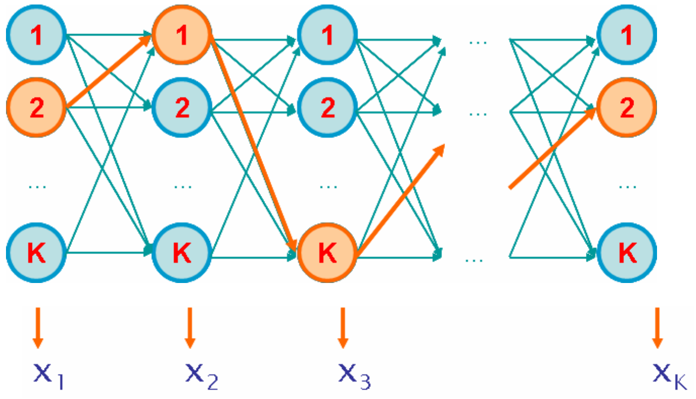
3.3 Features and notations in HMM

HMM is memory-less: at each step t, the only thing that affects future states is the current state π_t. I.e.:

$P(\pi_{t+1} = k \mid \text{"whatever happened so far"}) =P(\pi_{t+1} = k \mid \pi_1, \pi_2, \ldots, \pi_t, x_1, x_2, \ldots, x_t)= P(\pi_{t+1} = k \mid \pi_t)$

What is "Hidden": The sequence of states is hidden from the sequence of outcomes

A parse of a sequence: A sequence of *states* $\pi= \pi_1, \ldots.. \pi_N$ for a given sequence $x=x_1,\ldots x_N$



Likehood of a parse: The probability that a given parse ($\pi= \pi_1, \ldots.. \pi_N$) generates a given sequence ($x=x_1,\ldots x_N$):

$$P(x, \pi) = P(x_1, \cdots, x_N, \pi_1, \ldots\ldots, \pi_N) =$$
$$P(x_N, \pi_N \mid \pi_{N-1})\, P(x_{N-1}, \pi_{N-1} \mid \pi_{N-2})\cdots\cdots P(x_2, \pi_2 \mid \pi_1)\, P(x_1, \pi_1) =$$
$$P(x_N \mid \pi_N)\, P(\pi_N \mid \pi_{N-1}) \cdots\cdots P(x_2 \mid \pi_2)\, P(\pi_2 \mid \pi_1)\, P(x_1 \mid \pi_1)\, P(\pi_1) =$$
$$a_{0\pi 1}\, a_{\pi 1 \pi 2} \ldots\ldots a_{\pi N\text{-}1 \pi N}\, e_{\pi 1}(x_1)\cdots\cdots e_{\pi N}(x_N)$$

Using our favorite casino example, given a sequence of roll of x=1215621624, the likehood of a parse π=Fair Fair Fair Fair Fair Fair Fair Fair Fair Fair is

$a_{0Fair*}P(1 \mid Fair)\ P(Fair \mid Fair)\ P(2 \mid Fair)\ P(Fair \mid Fair)\ \cdots\ P(4 \mid Fair)=$
$1/2*(1/6)^{10} *(0.95)9=0.5x10^{-9}$

Similarly, the likehood of a parse π=Loaded Loaded Loaded Loaded Loaded Loaded Loaded Loaded Loaded is

$a_{0Loaded*}P(1|Loaded)\ P(Loaded|Loaded)\ P(2|Loaded)\ P(Loaded|Loaded)\ \cdots\ P(4|Loaded)$
$=1/2*(1/10)^8 \times (1/2)^2\ (0.95)^9=7.9x10^{-10}$

We could conclude that for above given rolls, it is 6.59 times more likely that the die is fair all the way, than that it is loaded all the way.

3.3 Main questions on HMMs (i.e. what do we learn from HMM?)

Evaluation: Given a HMM model *M*, and a sequence *x*, what is the probability of that *x* is generated from *M*? i.e. P(*x*|*M*)

Decoding: Given a HMM model *M* and a sequence *x*, find a parse(s) of the sequence π (i.e. a set of states) that maximizes P(*x*, π|*M*).

Learning: Given a HMM model *M* with unspecified transition/emission probabilities, and a sequence x, using (experimental) training data to parameterize $\theta = (e_i(.), a_{ij})$ that maximize P(*x* | θ).

4. Decoding and Evaluation algorithms

3.1 Viterbi algorithm (Decoding)

Idea of Viterbi algorithm: Given a sequence $x=x_1\ldots x_N$, and HMM model, the goal of decoding is to find a parse $\pi=\pi_1\ldots\pi_N$ such that $P(x, \pi|M)= P(x,\pi)$ is maximal. Because of the memory-less feature of HMM, the probability of the optimal parse ending at state $\pi_{i+1}=l$ only depends on the prefix optimal parse ending at state $\pi_i=k$ and the transition probability from state k to l. This is a sign of using Dynamic Programming and also the basic idea behind Viterbi algorithm:

Define $V_k(i)$= Probability of most likely sequence of states generating the first i letters and ending at state $\pi_i=k$, i.e.

$V_k(i)=\max_{(\pi 1\ldots\pi i-1)}P(x_1\ldots x_{i-1}, \pi_1\ldots\pi_{i-1}, x_i, \pi_i=k)$

Then $V_l(i+1)=\max_k P(x_{i+1}, \pi_{i+1} = l \mid \pi i = k)\ \max\{\pi_{1,\cdots,i-1}\}P(x_1\ldots x_{i-1},\pi_1,\ldots,\pi_{i-1}, x_i, \pi_i=k)$
$=e_l(x_{i+1})\ \max_k\ a_{kl}\ V_k(i)$

Algorithm: input $x=x_1\ldots x_N$

Initiation: $V_0(0)=1$; $V_k(0)=0$, for all k>0

Iteration: $V_j(i)=e_j(x_i)\ \max_k[a_{kj}V_k(i-1)]$; $Ptr_j(i)=\mathrm{argmax}_k(a_{kj}V_k(i-1))$

Termination: $P(x, \pi^*) =\max_k V_k(N)$

Traceback: $\pi_N^*=\mathrm{argmax}_k V_k(N)$; $\pi_{i-1}^* = Ptr_{\pi i}\ (i)$

There is a practical issue we need to notice, that is because the absolute value of these probabilities is very small, the underflows will cause problem in computer calculations. We therefore take log of all values: $V_l(i) =\log e_k(x_i)+\max_k[V_k(i-1) +\log a_{kl}\ ]$

Time & space:

To go over the whole Viterbi matrix, we need time O(K*N), for calculating each entry we spend another O(K) time. So in total the running time is $O(K^2N)$
We need O(KN) space to store the Viterbi matrix.

3.2 Forward algorithm (Evaluation)

Compare to Viterbi:

The evaluation problem in HMM is finding the probability of generating a sequence x given the HMM. I.e. $P(x|M)= \Sigma_\pi P(x, \pi|M) = \Sigma_\pi P(x \mid \pi) P(\pi)$. The *forward algorithm* similar to Viterbi method except that Viterbi asks for the most likely parse generating x, while *forward algorithm* asks for the sum over all possible parses generating x.

Define the *forward* probability:

$f_l(i) = P(x_1\ldots x_i, \pi_i = l) \quad = \Sigma_{\pi 1\ldots\pi i\text{-}1} P(x_1\ldots x_{i-1}, \pi_1,\ldots, \pi_{i-1}, \pi i = l)\ e_l(x_i)$

$=\Sigma_k \Sigma_{\pi 1\ldots\pi i\text{-}2} P(x_1\ldots x_{i-1}, \pi_1,\ldots, \pi_{i-2}, \pi_{i-1} = k)\ a_{kl}\ e_l(x_i)$

$=e_l(x_i)\ \Sigma_k f_k(i-1)\ a_{kl}$

Algorithm:

Initiation: $f_0(0)=1$; $f_k(0) = 0$, for all $k > 0$
Iteration: $f_l(i) = e_l(x_i)\ \Sigma_k f_k(i-1)\ a_{kl}$
Termination: $P(x) = \Sigma_k f_k(N)\ a_{k0}$; a_{k0}=the probability that the terminating state is k

Time & space:

Similar to Viterbi algorithm, Filling out the whole *forward* matrix, we need time O(K*N), for calculating each entry we spend another O(K) time. So in total the running time is $O(K^2N)$
We need O(KN) space to store the matrix.

5 Conclusions:

1. Assume a discrete-time, discrete-space dynamical system governed by a Markov chain emits a sequence of observable outputs: one output (observation) for each state in a trajectory of such states. From the observable sequence of outputs, HMM infers the most likely dynamical system. The result is a model for the underlying process
2. HMM is memory-less, therefore can not accurately describe the probability function if an event is not geometric distribution, such as the exon length in genome, which requires a long-term statistics.
3. The strategy of Dynamic Programming is applied into the decoding and evaluation algorithms of HMMs because the current state of a HMM is fully determined by the immediate prefix state. The running time of the DP-like algorithms are $O(K^2N)$ and the required space is O(KN).