Hidden Markov Models.

Want to model sequences more realistically. Behavior of biomolecule sequence is not simply a set of *iid*'s from position to position. One way to think of this is: we have expert knowledge of some functions and structures of biomolecules. Rather than counting frequencies across all amino acid positions, let us bin positions according to these localized functions and then do frequency counts. This will lead to *position specific scoring matrices*.

There are many ways of going about this. General cadre is sometimes referred to as *machine learning*. We want to either classify some data according to some categories ("α-helical segment", or "nucleotide binding domain", e.g.). Want to convert expert knowledge into an executable algorithm. It is a natural area for probabilistic inference, and more specifically, for a Bayesian framework. Various versions exist: hidden Markov models, artificial neural

networks, Bayesian networks, graphical models, etc., roughly in increasing order of generality. We will start here with hidden Markov models (HMM's) because they are relatively simple, yet flexible enough to be very applicable for recognition and classification problems in molecular biology.

The main problems which will emerge are the following:

a.) Modeling the situation.

This usually means finding a graphical representation of how the data is to be organized. Often this is where a great deal of expertise goes, even before the data is analyzed. It is often represented in terms of graphs where the nodes of a graph might represent discrete states, and the nodes are joined by edges which represent transitions from one state to another. These transitions will be probabilistic, due to uncertain knowledge of

when we should transition from one state to another.

b.) Training the model.

This means assigning the probabilistic parameter values in the model, which must be learned from (sufficient!) reliable data.

c.) Validating the model.

Here one must compare the model of a., as parametrized in b., against independent data for the problem to see whether the combination correctly classify or align or perform whatever comparison function (to what degree, that is).

Dependencies: Markov Models.

We use discrete Markov chains. Here the dynamics will follow, instead of time, as earlier (in PAM, e.g.), the sequence position, from left

to right. So, we have, e.g., a.a. residues at positions $1, 2, \ldots, n, \ldots,$ We will be interested in calculating the probabilities of various sequences,

$$P(x_1 \ldots x_n),$$

and we can telescope this into the successive conditional probabilities:

$$\begin{aligned} P(x_1 \ldots x_n) &= P(x_n | x_1 \ldots x_{n-1}) \cdot P(x_1 \ldots x_{n-1}) \\ &= \ldots \\ &= \Pi_{i=1}^{i=n} P(x_i | x_1 \ldots x_{i-1}) \end{aligned}$$

The (first order) Markov assumption is that

$$P(x_i | x_1 \ldots x_{i-1}) = P(x_i | x_{i-1}).$$

k-th order Markov assumption is

$$P(x_i | x_1 \ldots x_{i-1}) = P(x_i | x_{i-k} \ldots x_{i-1}).$$

Such a model will assume that the observed residue at position n is not independent of the other residues viewed, but depends on what is to the left of the position, either one place or k. It localizes the dependencies.

A hidden Markov model assumes that there are two random processes (Markov chains) occurring, of which only one is observed, the other *hidden*. For discrete (e.g., finite) alphabets for the sequences we are observing, this can be viewed as the position specific scoring matrix.

It is useful to start with a dice model, the dishonest casino. A casino rolls a die and the number is observed. This is an obvious Markov chain, if the die is rolled independently each time. However, assume that there are two dice, and that the casino switches between them. Suppose one of the dice is biased, the other fair. Then there is a second Markov process, choosing which die to roll. This process is not observed (hopes the casino!). To understand the observed process (which number is visible) you need to understand the unobserved process. This example has two *main states*, "fair" or "biased". We may not know exactly what state the roll is in, but we can hope to ascertain or estimate the transition probabilities for

these main states. Each main state has its own *emission probabilities*, that is, its probability, in that state, of displaying a 1, 2, ... 5 or 6. In the casino example, these will be different.

Here is a particular version of this problem: it is very simple in that you are given the number of main states, the transition probabilities of the hidden Markov process, and the emission probabilities of the two main states. The data is the sequence of rolls which have been observed, the problem is to determine the main state chain, i.e., classify each roll into "fair" or "biased". There are 300 rolls recorded here.

Dice Data for the "Dishonest Casino"

$$P(1) = \ldots = P(5) = 0.1; P(6) = 0.5.$$

$$\begin{aligned} P(\text{fair} \to \text{fair}) &= 0.95 \\ P(\text{fair} \to \text{biased}) &= 0.05 \\ P(\text{biased} \to \text{fair}) &= 0.1 \\ P(\text{biased} \to \text{biased}) &= 0.9. \end{aligned}$$

- Data:

31511624644664424531132163116 4
15213362514454363165662656666 6
65116645313265124563666463163 6
66316232645523626666662515163 1
22255544166656656356432436413 1
51346514635341112641462625335 6
36616366646623253441366166116 3
25256246225526525226643535333 6
23312162536441443233516324363 3
66556246666263266661235524524 2

We want to come up with the most probable sequence of 300 F's and L's (for "loaded") to explain the data. There is a trio of three simple algorithms which apply here, which we describe for situations graphically the same as the dice problem. They will be modified flexibly later. The algorithms are DP algorithms, and the main requirement is to have a direction for the problem.

Let us say that we have a finite set of main states $\{\pi^{(1)}, \ldots, \pi^{(K)}\}$, and we have a Markov chain $\pi_1 \pi_2 \ldots \pi_n$ from these possible states. We have transition probabilities

$$a_{k\ell} = P(\pi^{(k)} \to \pi^{(\ell)}).$$

Suppose we have observed states $\{x^{(1)}, \ldots, x^{(M)}\}$. Each main state has its own *emission probabilities*

$$e_\ell(x^{(b)}) = P(x^{(b)} | \text{ in state } \pi^{(\ell)}).$$

We often simplify by setting

$$\pi_k = \text{“}k\text{”}; x^{(b)} = \text{“}b\text{”},$$

so that $e_\ell(x^{(b)})$ becomes $e_\ell(b)$. The first algorithm (*the Viterbi algorithm*) finds the path of maximal probability through the main states, that is $\pi^* = \text{path } \pi_1 \ldots \pi_L$ such that $P(x, \pi)$ is maximal over all paths π through the main states.

For convenience, we add two artificial main states, “Begin” (B) and “End” (E).

Given the observed sequence $x_1 \dots x_L$, with $x_0 = B$ and $x_{L+1} = E$, let $v_k(i), k = 1, \dots, K, j = 1, \dots, L$ be the maximal probability for a path $\pi_1 \dots \pi_i$ with emissions x_j in the j-th position. The basic equation is elementary:

$$v_k(i+1) = e_k(x_{i+1}) \cdot \max_\ell(v_\ell(i) a_{\ell k}).$$

We will need a traceback pointer: $\text{ptr}_i(\pi_i^* = \pi^{(s)}) = \pi_{i-1}^*$, where $\pi_{i-1}^* = \pi^{(j)}$ so that $\ell = j$ realizes the max in $\max_\ell(v_\ell(i-1)a_{\ell s})$.

Viterbi Algorithm

Initialisation:	$v_B(0) = 1$;
	$v_k(0) = 0, k > 0$.
Recursion:	$v_k(i) = e_k(x_i)$
$(i = 1, \dots L)$	$\times \max_\ell(v_\ell(i-1)a_{\ell k})$;
	$\text{ptr}_i(k) =$ as above.
Termination:	$P(x, \pi^*) = \max_k(v_k(L)a_{kE})$;
	$\pi_L^* = \text{argmax}_k(v_k(L)a_{kE})$.
Traceback:	$\pi_{i-1}^* = \text{ptr}_i(\pi_i^*)$.

Two remarks here: we have modeled the end of the sequence explicitly (a_{kE} above). This

should be set = 1 we don't model this explicitly. This is related to modeling the length of the sequence observed. Also, the computations should be done in "log space", since products of many probabilities give underflow errors computationally.

Here is the Viterbi algorithm applied to the dishonest casino problem:

For further use, we need to compute the overall probability $P(x)$ of an observed sequence. There are two ways to do this, left to right (*forward algorithm*) and right to left (*backwards algorithm*).

Set

$$f_k(i) = P(x_1 \dots x_i, \pi_i = \pi^{(k)}).$$

Forward Algorithm.

Initialization:	$f_B(0) = 1,$
	$f_k(0) = 0, k > 1;$
Recursion:	$f_k(i+1) = e_k(x_{i+1}) \sum_\ell f_\ell(i) a_{\ell k}.$
Termination:	$P(x) = \sum_k f_k(L) a_{kE}.$

Same remarks about B, E. Note convention $B = \pi^{(0)}, E = \pi^{(K+1)}$.

Finally, it turns out to be useful to know how to compute the total probability going from right to left, the *backward algorithm*. Let $b_i(k) = P(x_{i+1} \ldots x_L | \pi_i = \pi^{(k)})$.

Backward Algorithm.

Initialization:	$b_k(L) = a_{kE}$, for all k.
Recursion:	$b_k(i) = \sum_\ell e_\ell(x_{i+1}) b_\ell(i+1) a_{\ell k}$.
Termination:	$P(x) = \sum_\ell e_\ell(x_1) b_\ell(1) a_{B\ell}$.

You almost never need the termination step, because the typical application is computing the posterior probabillity

$$P(\pi_i = \pi^{(k)} | x)$$

$$= \frac{P(x_1 \ldots x_i, \pi_i = \pi^{(k)}) P(x_{i+1} \ldots x_L | \pi_i = \pi^{(k)})}{P(x)}$$

$$= \frac{f_k(i) b_k(i)}{P(x)}.$$

This gives rise to another way to *parse* or *decode* the sequence, the *maximal posterior decoding*:

$$\widehat{\pi}_i = \text{argmax}_k \, P(\pi_i = \pi^{(k)} | \, x).$$

Minus: can lead to impossible path through the hidden Markov model (may require some transitions which have probability 0).

Plus: sometimes can hear weak signal which Viterbi misses. See two examples (from DE).

Note: to begin, used transitions $B \to \pi^{(i)}$. This is equivalent to giving an initial distribution for the (hidden) states. In implementations usually referred to as the *prior distribution*.

Training HMM's.

The HMM above had several parameters: $a_{k\ell}$, $e_k(x^{(i)})$ and the prior a_{Bk}. We may also need

a_{kE}, if we are modeling length explicitly. Distinguish two cases: *paths known* and *paths unknown*.

a.) Paths known.

This means we have data of the sort we want to parse, along with the knowledge of the hidden states at each position. Then data can be sorted to give frequency estimates:

$$a_{k\ell} = \frac{\#\{k \to \ell \text{ transitions observed}\}}{\#\{\text{all transitions } k \to \ell'\}}$$

$$= \frac{A_{k\ell}}{\sum_{\ell'} A_{k\ell'}},$$

and similarly

$$e_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}.$$

Gives *maximum likelihood estimation:* maximizes the likelihood

$$P(D|\theta) = P(\text{data observed} \,|\, \text{par. vals.} = \theta),$$

or "prob. of the data, given the model".

Insufficient data: use pseudocounts, or use a smoother prior distribution, such as a Dirichlet distribution.

b.) Paths unknown.

This falls in the cadre of missing data problems: missing the state paths. Thus, must learn them from the data presented. This will be an optimization problem. Key algorithm is the *Baum-Welch training algorithm*. This is a case of the expectation maximization algorithm.

c.) Expectation maximization.

Given x, y outcomes for a model. x stands for known data, for us

$$x = \begin{cases} x_{(1)} \\ \cdots \\ x_{(N)} \end{cases}$$

$$= \begin{cases} x_{(1),1} \cdots x_{(1),L_1} \\ \cdots \\ x_{(N),1} \cdots x_{(N),L_N} \end{cases}$$

data from N sequences. For now, assume L's all equal. These would be the observed states. Collection of y variables for the missing data (similar matrix of hidden states). Idea is to iteratively improve estimations, cut off at a threshold.

Try to maximize the log-likelihood;

$$\log P(x|\theta) = \log \sum_y P(x, y|\theta),$$

sum over possible values of y. Conditioning says

$$P(x, y|\theta) = P(y|x, \theta)P(x|\theta),$$

giving

$$\log P(x|\theta) = \log P(x, y|\theta) - \log P(y|x, \theta).$$

Given estimate for the parameters $\theta = \theta^{(i)}$, seek better estimate $\theta^{(i+1)}$. Multiply by $P(y|x, \theta^{(i)})$, and sum over y's:

$$\log P(x|\theta) = \sum_y P(y|x, \theta^{(i)}) \log P(x, y|\theta)$$

$$- \sum_y P(y|x, \theta^{(i)}) P(y|x, \theta).$$

Set

$$Q(\theta, \theta^{(i)}) = \sum P(y|x, \theta^{(i)}) \log P(x, y|\theta),$$

and write

$$\log P(x|\theta) - \log P(x|\theta^{(i)})$$

$$= Q(\theta|\theta^{(i)}) - Q(\theta^{(i)}, \theta^{(i)})$$

$$+ \sum_y P(y|x, \theta^{(i)}) \log \frac{P(y|x, \theta)}{P(y|x, \theta^{(i)})}.$$

Notice last is relative entropy, so ≥ 0:

$$\log P(x|\theta) - \log P(x|\theta^{(i)}) \geq Q(\theta, \theta^{(i)}) - Q(\theta^{(i)}, \theta^{(i)}).$$

To find a $\theta^{(i+1)}$ which would raise the log-likelihood, we can maximize $Q(\theta, \theta^{(i)})$ as function of θ. Set this $= \theta^{(i+1)}$. So, compute Q (expectation step), then maximize. Iterate until below threshold for improvement

$$\log P(x|\theta^{(i+1)}) - \log P(x|\theta^{(i)}) \leq \epsilon.$$

This is an improvement! For us, the forward and backward algorithms make the necessary

calculations easy. The log-likelihood is given by

$$\log P(x|\theta) = \sum_{\pi} \log P(x, \pi|\theta),$$

where π is the path through the hidden states. These are the missing data y above. You start by fixing model parameters arbitrarily. Then we can calculate the *expected values* $A_{k\ell}, E_k(b)$. This is done as follows: given a single sequence x, and model parameters θ, the probability that $a_{k\ell}$ is used from position i to position $i+1$ is given by

$$P(\pi_i = k, \pi_{i+1} = \ell | x, \theta)$$

$$= \frac{f_k(i) a_{k\ell} e_\ell(x_{i+1}) b_\ell(i+1)}{P(x)}.$$

Now, if we have N training sequences $x_{(j)}$ as above, then the probable number of times we expect to see $a_{k\ell}$ used in the data is

$$A_{k\ell} =$$

$$\sum_{j=1}^{j=N} \frac{1}{P(x_{(j)})} \sum_i f_{(j);k} a_{k\ell} e_\ell(x_{(j);i+1}) b_{(j);\ell}(i+1).$$

Similarly, we expect a count $E_k(b)$ of b-emissions from state k given by

$$E_k(b) = \sum_j \frac{1}{P(x_{(j)})} \sum_{\{i|x_{(j);i}=b\}} f_{(j);k}(i) b_{(j);k}(i).$$

Now given our basic model, if we knew the paths through the hidden states we would get

$$P(x, \pi|\theta) = \Pi_k \Pi_b [e_k(b)]^{E_k(b,\pi)} \Pi_k \Pi_\ell a_{k\ell}^{A_{k\ell}(\pi)}.$$

Take logs, get

$$Q(\theta, \theta^{(i)}) = \sum_\pi P(\pi|x, \theta^{(i)}) \times$$

$$\left[\sum_k \sum_b E_k(b, \pi) \log e_k(b) + \sum_k \sum_\ell A_{k\ell}(\pi) \log a_{k\ell} \right].$$

Do the π sum first, get

$$Q(\theta, \theta^{(i)}) = \sum_k \sum_b E_k(b) \log e_k(b) + \sum_k \sum_\ell A_{k\ell} \log a_{k\ell}.$$

To maximize this, it turns out we just determine $\theta^{(i+1)}$ by

$$a'_{k\ell} = \frac{A_{k\ell}}{\sum_{\ell'} A_{k\ell'}}; e'_k(b) = \frac{E_k(b)}{\sum_{b'} E_k(b')}.$$

Compare this to any other choice of $a_{k\ell}$. The difference would be

$$\sum_k \sum_\ell A_{k\ell} \log \frac{a'_{k\ell}}{a_{k\ell}} = \sum_k \left(\sum_{\ell'} A_{k\ell'} \right) \sum_\ell a'_{k\ell} \log \frac{a'_{k\ell}}{a_{k\ell}}.$$

But these last summands are all relative entropies, and so are all ≥ 0 and $= 0$ iff $a'_{k\ell} = a_{k\ell}$.

Similar story for the sum with the $E_k(b)$'s.

Summarizing, we start with an estimation for the parameters, then calculate the expected number of times we would see the various transitions or emissions in our data under these parameter values, then we re-estimate the parameters using these expected number of occurrences as the new counts; iterate.

Baum-Welch Training Algorithm.

Initialization:	Take arbitrary parameter values for the model. (Take A's and E's equal to pseudocount values.)
Recursion:	For each training sequence $x_{(j)}, j = 1, \ldots, N$, calculate forward, backward variables; calculated expected values $A_{k\ell}, E_k(b)$ and sum over j. Calculate new parameters from new A's, E's. Calculate new log likelihood.
Termination:	Threshold: ΔLL small enough, or no. of iterations large.

We started from an arbitrary set of parameter values for the model. Can add a randomizer that will test the algorithm's progress against a random sample of the space of parameters for maximization.

Architecture.

Described HMM for the simplest case, along a straight line, with fixed length. Our first example will be to model the properties of a related family of proteins. Want to model length explicitly. Consider the following examples, pictorially, where we add some new connections between the nodes of our graph.