

Pair HMMs and CRFs

David Hall (dlwh@)

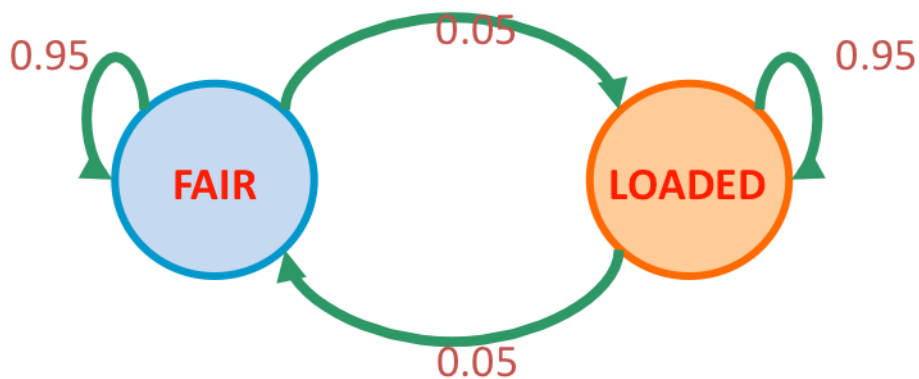
February 4, 2009

1 Notation Recap

Recall that an HMM is a tuple (Σ, Q, A, a_0, e) where,

- $\Sigma = \{b_1, \dots, b_M\}$ is the alphabet. (e.g. ACTG)
- $Q = \{1, \dots, K\}$ is a set of states. (CpG island or not)
- $A = (a_{ij})$ is a $K \times K$ matrix of transition probabilities between states.
- a_{0i} is a K -dimensional vector of initial state probabilities.
- e_{ib_k} are emission probabilities. (a $K \times M$ matrix)

An example is from the casino example, where we have two states (FAIR and LOADED) and transitions between the states with probabilities labeled on the arcs. Not shown are emissions, which would be the die rolls.



2 Pair HMMs

There are two kinds of Pair HMMs, one that we won't really be using, and one that we will be using.

2.1 Dual-Emission HMMs

The basic idea here is to emit a pair of observations at each state. So you would formally define it as $(\Sigma_1 \times \Sigma_2, Q, A, a_0, e)$, for some two alphabets Σ_1 and Σ_2 . In the casino example, this would be like if there were two games being played, and the dealer would alternate between both games being fair or both games being unfair. Doing learning, inference, and decoding here is much the same as it is in normal HMMs. If you treat each pair of states as a single state, then the algorithms work exactly the same. As you add more states triple-hmms, ..., K-HMMS you end up exploding your state space to $O(|\Sigma|^K)$.

Practical applications of this kind of HMM are doing alignments where gaps aren't allowed, or anywhere where you always get an emission from both states.

2.2 Pair-HMMs with η

A more relevant (for our purposes) HMM allows η (the empty character) in both alphabets. That is $(\Sigma_1 \cup \{\eta\}) \times (\Sigma_2 \cup \{\eta\}), Q, A, a_0, e$. The interpretation of η in the casino example would be that the dealer may with some probability decide to skip playing one game at some step. That is, he may decide not to flip a coin in one game 60% of the time when he's in "FAIR" mode, and you will receive no information that he decided to do that. For simplicity, we assume at most one of the emissions at any step is an η .

2.2.1 Decoding

Thus, figuring out π in Pair-HMMs is more complicated, since we don't know if a state in the sequence is expressed or not. Because of this, normal Viterbi and normal posterior decoding won't work. However, we can modify them.

Specifically, consider single-sequence Viterbi, defined as

$$\begin{aligned} V_k(i) &= \max_{\pi_1, \dots, \pi_{i-1}} P(x_1, \dots, x_{i-1}, \pi_1, \pi_{i-1}, x_i, \pi_i = k) \\ &= e_{kx_i} \max_j a_{jk} V_j(i-1) \end{aligned}$$

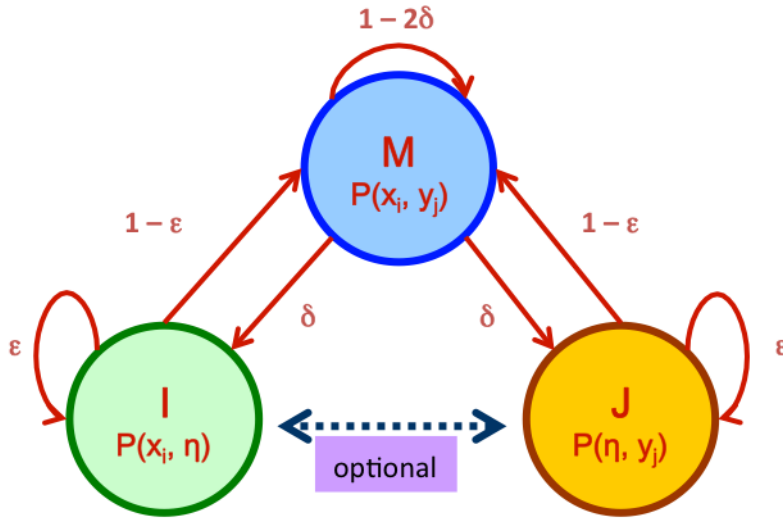
. This won't work for Pair-HMMs because the first i π states may not generate the first i observations of both sequences x and y . So, we have to keep track of the possibility that either of the strings could actually have an η at any point. We revise Viterbi to another DP that builds a square matrix. (It should look really familiar by now!)

$$\begin{aligned} V_M(i, j) &= P(x_i, y_j) \max \begin{cases} (1 - 2\delta)V_M(i-1, j-1) \\ (1 - \epsilon)V_I(i-1, j-1) \\ (1 - \epsilon)V_J(i-1, j-1) \end{cases} \\ (1) \quad V_I(i, j) &= Q(x_i) \max \begin{cases} \delta V_M(i-1, j) \\ \epsilon V_I(i-1, j) \end{cases} \\ V_J(i, j) &= Q(y_j) \max \begin{cases} \delta V_M(i, j-1) \\ \epsilon V_J(i, j-1) \end{cases} \end{aligned}$$

where we make the assumption (and definition) that $\forall c \in \Sigma P(\epsilon, c) = P(c, \epsilon) = Q(c)$, and $P(x_i, y_j)$ is the probability of emitting the two characters given the state. δ is probability of opening a gap, and η is the

probability of extending a gap. The other probabilities are set so that the transitions sum to one. The interpretation of these three states is that $V_M(i, j)$ represents the probability of aligning $(x_{1:i}, y_{1:j})$ ending in state M (a match state), V_I the probability of aligning $(x_{1:i}, y_{1:j})$ ending in I (the x-gap state), and V_J the probability of aligning $(x_{1:i}, y_{1:j})$ ending in a gap state J (the y gap state). Therefore, at the end of your recurrence, just choose the max of these three values for position (M,N).

Pictorially, we represent this algorithm by a three-state HMM, one for each recurrence:



$$\forall c \in \Sigma, P(\eta, c) = P(c, \eta) = Q(c)$$

Note that a similar transformation can be applied to forward/backward. See Durbin for more information.

2.2.2 Pair Viterbi as Needleman-Wunsch

The reason this DP should look so familiar is that it's equivalent to Needleman-Wunsch with affine gaps. Let's start by dividing through by $\prod_i Q(x_i) \prod_i Q(y_j)$, which is similar to the cost of doing no alignment at all (deleting everything):

$$V_M(i, j) = \frac{P(x_i, y_j)}{Q(x_i)Q(y_j)} \max \begin{cases} (1 - 2\delta)V_M(i - 1, j - 1) \\ (1 - \epsilon)V_I(i - 1, j - 1) \\ (1 - \epsilon)V_J(i - 1, j - 1) \end{cases}$$

$$V_I(i, j) = \max \begin{cases} \delta V_M(i - 1, j) \\ \epsilon V_I(i - 1, j) \end{cases}$$

$$V_J(i, j) = \max \begin{cases} \delta V_M(i, j - 1) \\ \epsilon V_J(i, j - 1) \end{cases}$$

These scores are multiplicative, but alignment was additive. Just take logs, and remove terms that won't affect the final result:

$$\begin{aligned} \log V_M(i, j) &= \log \frac{P(x_i, y_i)}{Q(x_i)Q(y_i)} + \max \begin{cases} \log V_M(i-1, j-1) \\ \log V_I(i-1, j-1) \\ \log V_J(i-1, j-1) \end{cases} \\ \log V_I(i, j) &= \max \begin{cases} \log \delta + \log V_M(i-1, j) \\ \log \epsilon + \log V_I(i-1, j) \end{cases} \\ \log V_J(i, j) &= \max \begin{cases} \log \delta + \log V_M(i, j-1) \\ \log \epsilon + \log V_J(i, j-1) \end{cases} \end{aligned}$$

And now, through the magic of renaming, we have a perfect mapping:

$$\begin{aligned} (2) \quad M(i, j) &= S(x_i, y_j) + \max \begin{cases} M(i-1, j-1) \\ I(i-1, j-1) \\ J(i-1, j-1) \end{cases} \\ I(i, j) &= \max \begin{cases} d + M(i-1, j) \\ e + I(i-1, j) \end{cases} \\ J(i, j) &= \max \begin{cases} e + M(i, j-1) \\ e + J(i, j-1) \end{cases} \end{aligned}$$

That ends us with the mapping:

$$\begin{aligned} (3) \quad d &= \log \delta \\ e &= \log \epsilon \\ S(i, j) &= \log \frac{P(x_i, y_j)}{Q(x_i)Q(y_j)} \end{aligned}$$

3 Conditional Random Fields

Thus far, we've looked at HMMs, which represent a distribution $P(\pi, x)$ for an observed sequence x and a parse π . However, usually we are not interested in the distribution over x , since we already know what x is. Instead, we mostly want to know $P(\pi|x)$. For instance if we're using Viterbi, we want $\arg \max_{\pi} P(\pi|x) = \arg \max P(x, \pi)$, and if we're using posterior decoding, we want the full posterior $P(\pi|x)$. The key insight behind CRFs, then, is that we don't need to spend any time learning the full joint distribution $P(\pi, x)$. Instead, we are perfectly happy learning $P(\pi|x)$. This conditioning is precisely why we call them conditional random fields.

3.1 Definition

Formally, a *Conditional Random Field* is a probability distribution over the hidden parse π given x :

$$(4) \quad \begin{aligned} P(\pi|x) &= \frac{1}{Z(x)} \exp\left(\sum_i^{|x|} \mathbf{w}^T \mathbf{F}(\pi_i, \pi_{i-1}, x, i)\right) \\ Z(x) &= \sum_{\pi} \exp\left(\sum_i^{|x|} \mathbf{w}^T \mathbf{F}(\pi_i, \pi_{i-1}, x, i)\right) \end{aligned}$$

$Z(x)$ is called the *partition function*. It acts as a normalization constant to ensure that everything sums to 1. \mathbf{F} is an arbitrary function with a domain state \times state \times observations \times observations \times positions $\rightarrow \mathfrak{R}^n$, and \mathbf{w} is a weights vector also in \mathfrak{R}^n . Recall that $a^T b$ is the dot product, which is just $\sum_i a_i b_i$. Thus, the weights influence how much impact the features have on the overall probability, and each feature gets its own weight. A higher weight means that that feature is more influential in determining the probability of a sequence, while a lower weight means it has a lower.

3.2 HMMs vs. CRFs

It's actually possible to map an HMM as a CRF. First, recall that, for an HMM:

$$\begin{aligned} \log P(x, \pi) &= \log P(\pi_0) + \sum_{i=1}^{|x|} \log P(\pi_i | \pi_{i-1}) + \log P(x_i | \pi_i) \\ &= \log a_{0\pi_0} + \sum_{i=1}^{|x|} \log a_{\pi_{i-1}\pi_i} + \log e_{\pi_i x_i} \end{aligned}$$

Then, suppose we define a (very large!) weights vector using all of these the log probabilities. We'd get

$$(5) \quad \mathbf{w} = \begin{pmatrix} \log a_{01} \\ \vdots \\ \log a_{0K} \\ \vdots \\ \log a_{KK} \\ \log e_{1b_1} \\ \vdots \\ \log e_{Kb_M} \end{pmatrix} \in \mathfrak{R}$$

. Now define a feature function \mathbf{F} with the same dimension as \mathbf{w} . Intuitively, for the mapping to work, we want to include each weight that applies to that position. That is, we want the vector \mathbf{F} to be 1 where the values of x, π at position π are the right values for that weight, and 0 everywhere else. Define $1\{\text{condition}\}$ to be the indicator function: 1 wherever the condition is true and 0 otherwise. Then we can define \mathbf{F} as:

$$(6) \quad \mathbf{F}(\pi_i, \pi_{i-1}, x, i) = \begin{pmatrix} 1\{i = 1, \pi_i = 1\} \\ \vdots \\ 1\{i = 1, \pi_i = K\} \\ 1\{\pi_{i-1} = 1, \pi_i = 1\} \\ \vdots \\ 1\{\pi_{i-1} = K, \pi_i = K\} \\ \vdots \\ 1\{x_i = b_1, \pi_i = 1\} \\ \vdots \\ 1\{x_i = b_M, \pi_i = K\} \end{pmatrix} \in \mathfrak{R}$$

Combining these two, we can compute the log probability of the sequence as:

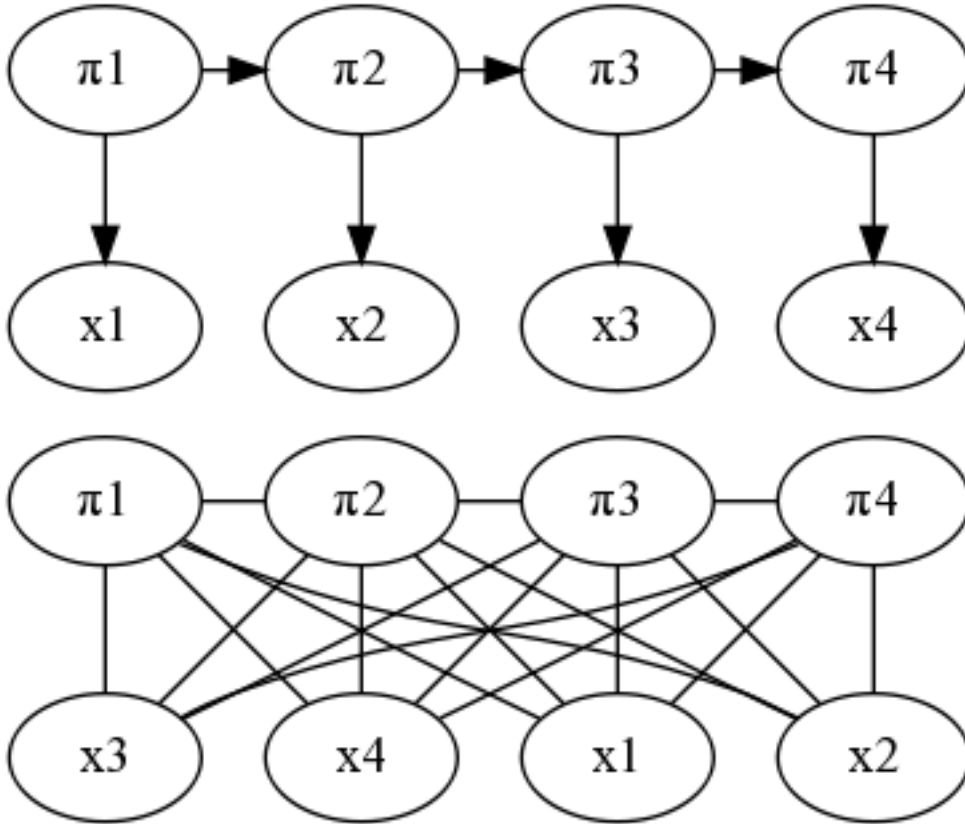
$$(7) \quad \begin{aligned} \log P(x, \pi) &= \sum_i^{|x|} \mathbf{w}^T \mathbf{F}(\pi_i, \pi_{i-1}, x, i) \\ \log P(\pi|x) &= \frac{P(x, \pi)}{\sum_{\pi'} P(x, \pi')} \end{aligned}$$

Thus, we can treat HMMs as a kind of CRF, with features for the initial “start” state, transition features from one state to another, and emission features for each state to the observed word.

However, it’s important to note that CRFs are more powerful than HMMs, for two primary reasons:

1. HMM parameters are constrained by requirements like $\sum_i a_{0i} = 1$. CRFs, on the other hand, can have arbitrary vectors \mathbf{w} .
2. CRFs can have many more kinds of features than HMMs. In particular, HMM can only generate each observation x_i once. However, CRFs don’t model x , and so we don’t need to worry about that. For instance, it’s fine to have a feature $1\{x_{i-1} = b_j, \pi_i = \ell\}$ (that is, one that is active for the previous observation and the current state, which aren’t allowed in HMMs. Alternatively, think about the casino example. We could have a feature that looks at the previous 100 dice roles to make a decision about transitioning between a FAIR die and a LOADED die.

A graphical representation is helpful. In the below, each state in the HMM is connected only to its own emission, its parent, and its child. In the CRF, there can be arbitrary communication between each state and every observed element of the sequence.



3.3 Inference and Decoding

So we have CRFs; how do we use these things? For Viterbi, using Equation 4 we can derive:

$$\begin{aligned}
 \arg \max_{\pi} P(\pi|x) &= \arg \max_{\pi} \frac{1}{Z(x)} \exp\left(\sum_i^{|\mathbf{x}|} \mathbf{w}^T \mathbf{F}(\pi_i, \pi_{i-1}, x, i)\right) \\
 (8) \qquad \qquad \qquad &= \arg \max_{\pi} \exp\left(\sum_i^{|\mathbf{x}|} \mathbf{w}^T \mathbf{F}(\pi_i, \pi_{i-1}, x, i)\right) \\
 &= \arg \max_{\pi} \sum_i^{|\mathbf{x}|} \mathbf{w}^T \mathbf{F}(\pi_i, \pi_{i-1}, x, i)
 \end{aligned}$$

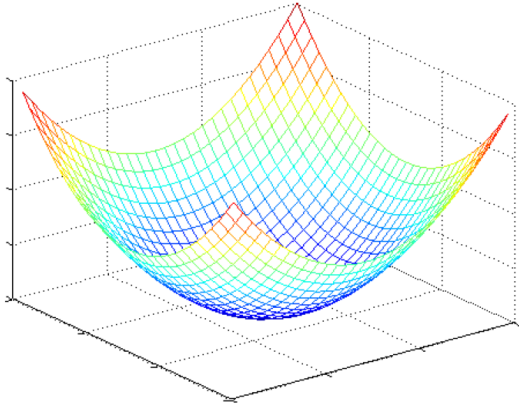
From there, it's straightforward to extend Viterbi to this new setting:

$$(9) \qquad \qquad \qquad V_k(i) = \max_j \mathbf{w}^T \mathbf{F}(k, j, x, i) + V_j(i-1)$$

This is basically the same as the HMM Viterbi, except that we calculate the feature/weight dot product instead of the different emission and transmission probabilities.

3.4 Learning

So far we've assumed we know what \mathbf{w} is. In HMMs, this wasn't too complicated, since you could just count, smooth, and divide to learn the parameters. Here, we're not estimating probabilities, so we can't just count and divide. Instead, we use an optimization method to find a global minimum that finds the best fit of \mathbf{w} to the observed values. The function $-\log P(\pi|x, \mathbf{w})$ is a differentiable, convex function of \mathbf{w} . In there are only two weights, you can represent the space as a "bowl" with a single minimum point as in the picture below:



Because of convexity, minimizing this function (which is the same as maximizing the probability of π) can be easily learned via gradient descent or any optimization method. This is analogous to reaching the bottom of a valley by following the steepest way down at every point. Let $g(w) = -\log P(\pi|x, \mathbf{w})$. Then gradient descent has the form:

$$(10) \quad w' := w - \nabla g(w)$$

, where $\nabla g(w)$ is the gradient of g evaluated at w .

