Markov Models
Probabilistic models to analyze the structure of a sequence

Based on Chapter 3 from Durbin, Eddy, Krogh, and Mitchison's book

Martin Tompa

CSE 427: Computational Biology

February 11, 2010

1 C_pG islands

We start with a motivating example, "C_pG islands". (Here, C_pG denotes the dinucleotide $5'$ CG $3'$, where the p stands for the phosphate on the DNA backbone between these two bases.) In the human genome, the C in the dinucleotide C_pG is often methylated, and from there the dinucleotide mutates to T_pG at a higher than usual rate. The result is that the dinucleotide C_pG is rarer than expected, given the independent probabilities of C and G.

However, methylation of DNA is suppressed in certain short regions, such as the promoters of genes. Therefore the frequency of C_pG is much higher in promoter regions than elsewhere in the genome. These "C_pG islands", typically hundreds or thousands of basepairs long, are important statistical signals for predicting gene starts.

This leads to two computational questions:

1. Given a chromosome sequence, how can you predict the C_pG islands?
2. Given a short sequence, how can you predict whether or not it is from a C_pG island?

We start with the second, simpler question, called the "classification" problem.

2 Markov chains

Since we are interested in dinucleotides, we need a probabilistic model of DNA sequences in which the probability of a character depends on the identity of the previous character. A DNA *Markov chain* is a directed graph with a vertex ("state") for each nucleotide A, C, G, T, and a directed edge with *transition probability* $a_{GT} = \Pr(x_i = T \mid x_{i-1} = G)$, and similarly for the other 15 dinucleotides, where $x = x_1 x_2 \ldots x_L$ is a DNA sequence.

From this definition,

$$\begin{aligned}
\Pr(x_1 x_2 \ldots x_L) &= \Pr(x_1) \Pr(x_2 \mid x_1) \Pr(x_3 \mid x_2) \ldots \Pr(x_L \mid x_{L-1}) \\
&= \Pr(x_1) a_{x_1 x_2} a_{x_2 x_3} \ldots a_{x_{L-1} x_L} .
\end{aligned}$$

To avoid the inhomogeneity of the $\Pr(x_1)$ factor, it is convenient to add a fifth state called 0 with $a_{0T} = \Pr(x_1 = T)$, etc., and to define $x_0 = 0$. With this convention, $\Pr(x_1 x_2 \ldots x_L) = a_{x_0 x_1} a_{x_1 x_2} a_{x_2 x_3} \ldots a_{x_{L-1} x_L}$.

2.1 Classification using Markov chains

Given *training data* consisting of "positive" examples (a number of $\mathrm{C}_p\mathrm{G}$ islands) and "negative" examples (a number of "background", non-$\mathrm{C}_p\mathrm{G}$-island sequences), Durbin *et al.* trained two Markov chains denoted + and − according to the following formulas:

$$a^+_{GT} = \frac{n^+_{GT}}{\sum_{s \in \{A,C,G,T\}} n^+_{Gs}} ,$$

where n^+_{GT} is the count of $\mathrm{G}_p\mathrm{T}$ in the positive training data, and similarly for each of the other 15 dinucleotides and for the negative training data. The two resulting Markov chains are shown in matrices on page 50 of Durbin *et al.* Note in these tables, for example, that $a^+_{CG} \gg a^-_{CG}$.

To combine these two Markov chains for the classification problem, use a *log likelihood ratio*:

$$\begin{aligned}
\mathrm{LLR}(x_1 x_2 \ldots x_L) &= \log_2 \frac{\Pr(x_1 x_2 \ldots x_L \mid \text{model } +)}{\Pr(x_1 x_2 \ldots x_L \mid \text{model } -)} \\
&= \log_2 \left(\frac{a^+_{x_0 x_1}}{a^-_{x_0 x_1}} \frac{a^+_{x_1 x_2}}{a^-_{x_1 x_2}} \cdots \frac{a^+_{x_{L-1} x_L}}{a^-_{x_{L-1} x_L}} \right) \\
&= \log_2 \frac{a^+_{x_0 x_1}}{a^-_{x_0 x_1}} + \log_2 \frac{a^+_{x_1 x_2}}{a^-_{x_1 x_2}} + \cdots + \log_2 \frac{a^+_{x_{L-1} x_L}}{a^-_{x_{L-1} x_L}} .
\end{aligned}$$

From the two Markov chain transition probability matrices, you can compute a 4×4 matrix with entries $\log_2 \frac{a^+_{rs}}{a^-_{rs}}$, for each of the 16 possible dinucleotides rs. Then computing $\mathrm{LLR}(x_1 x_2 \ldots x_L)$ is a simple matter of looking up and adding L entries from this matrix.

Figure 3.2 from Durbin *et al.* is a histogram of values of $\mathrm{LLR}(x)$ for *test data*, where the score $\mathrm{LLR}(x)$ is normalized by dividing by the length of the sequence x, resulting in the units "bits per nucleotide".

3 Hidden Markov models (HMMs)

We turn now to the other motivating problem from Section 1: given a chromosome sequence, how can you predict which portions are C$_p$G islands? We will combine the + and − Markov chains from Section 2.1 into a single model with 8 states A_+, C_+, G_+, T_+, A_-, C_-, G_-, T_-, with a small probability of transition from any + to any − state, and even smaller probability of transition from any − to any + state. Given a DNA sequence x generated by this probabilistic model, it is no longer possible to tell if a character C (say) came from the state C_+ or from the state C_-: the state is a "hidden variable".

This means we now need to distinguish between the sequence of states and the sequence of symbols generated. A *path* π is a sequence of states, so that $a_{jk} = \Pr(\pi_i = k \mid \pi_{i-1} = j)$, again with the convention $\pi_0 = 0$. In any state k, a symbol b is "emitted" with *emission probability* $e_k(b) = \Pr(x_i = b \mid \pi_i = k)$. In the C$_p$G island example, $e_{C_+}(C) = 1$, $e_{C_+}(A) = 0$, etc., but there can be more interesting emission probabilities in other HMMs, as we will see in the next example.

3.1 The occasionally dishonest casino example

This casino occasionally switches from using a fair die to a loaded (biased) die according to a HMM shown on page 54 of Durbin *et al.* If you were to observe just the outcomes of the die, what would be hidden is whether it is the fair or loaded die that is being used. We want to estimate this from just the outcomes of the die. Figure 3.5 shows one possible sequence of die outcomes that could have been generated from the HMM, the hidden state information that went along with that emitted sequence, and predictions (made without knowledge of the hidden state information) of which die was likely to have been used. These predictions were made according to the algorithm in the next section.

3.2 Finding the most probable state path: the Viterbi algorithm

Given an HMM and a sequence x of emitted symbols, the Viterbi algorithm calculates the most probable path π. It works by dynamic programming.

Let $v_k(i)$ be the probability of the most probable path that ends in state k when x_i is emitted. Then

$$\begin{aligned} v_0(0) &= 1, \\ v_k(0) &= 0 \text{ for } k \neq 0, \\ v_k(i) &= e_k(x_i) \max_j (v_j(i-1) a_{jk}) \text{ for } i > 0. \end{aligned}$$

To find the most probable path itself (rather than just its probability), trace backwards as in optimal alignment: if state k is the most probable when x_i is emitted, then the state j

that maximizes $v_j(i-1)a_{jk}$ is the most probable when x_{i-1} is emitted.

Go back and look at the Viterbi predictions in Figure 3.5 of Durbin *et al.* and compare them to the hidden states shown in that figure.

3.3 Computing the full distribution of state paths: the forward-backward algorithm

There are exponentially many ($2^{O(L)}$) paths of length L through an HMM, even if there are only two states as in the dishonest casino example. Because of this, the average probability of a single path is exponentially small (2^{-L} for a 2-state HMM), and the probability of even the most probable path (which is what the Viterbi algorithm computes) may not be much greater. It is not very informative to know the single most probable path, if the probability of following that path through the HMM is negligible.

Because of these considerations, we now discuss how to compute the full "posterior" probability distribution $\Pr(\pi_i = k \mid x)$, for every i and k. This is done by the combination of two algorithms, called the forward and backward algorithms. Each is a dynamic programming algorithm quite similar to the Viterbi algorithm.

3.3.1 Forward algorithm

Let $f_k(i) = \Pr(x_1 x_2 \ldots x_i \;\&\; \pi_i = k)$. Then

$$\begin{aligned} f_0(0) &= 1, \\ f_k(0) &= 0 \text{ for } k \neq 0, \\ f_k(i) &= e_k(x_i) \sum_j f_j(i-1) a_{jk} \text{ for } i > 0. \end{aligned}$$

Analogous to the start state 0, it is convenient to have a distinguished end state ε for which $a_{k\varepsilon} = \Pr(\pi_{L+1} = \varepsilon \mid \pi_L = k)$ is the probability that the path ends when in state k. Then, from the definition of $f_k(L)$,

$$\Pr(x_1 x_2 \ldots x_L \;\&\; \pi_{L+1} = \varepsilon) = \sum_k f_k(L) a_{k\varepsilon} \;. \tag{1}$$

3.3.2 How the forward algorithm contributes to the posterior probability

Let $x = x_1 x_2 \ldots x_L$. The posterior probability distribution that is our goal is

$$\Pr(\pi_i = k \mid x \;\&\; \pi_{L+1} = \varepsilon).$$

By Bayes' Theorem,

$$\begin{aligned}
&\Pr(\pi_i = k \mid x \;\&\; \pi_{L+1} = \varepsilon) \Pr(x \;\&\; \pi_{L+1} = \varepsilon) \\
&= \Pr(x \;\&\; \pi_{L+1} = \varepsilon \;\&\; \pi_i = k) \\
&= \Pr(x_1 x_2 \ldots x_i \;\&\; \pi_i = k) \Pr(x_{i+1} \ldots x_L \;\&\; \pi_{L+1} = \varepsilon \mid x_1 x_2 \ldots x_i \;\&\; \pi_i = k) \\
&= \Pr(x_1 x_2 \ldots x_i \;\&\; \pi_i = k) \Pr(x_{i+1} \ldots x_L \;\&\; \pi_{L+1} = \varepsilon \mid \pi_i = k) \\
&= f_k(i) \Pr(x_{i+1} \ldots x_L \;\&\; \pi_{L+1} = \varepsilon \mid \pi_i = k) \;.
\end{aligned}$$

Let $b_k(i) = \Pr(x_{i+1} \ldots x_L \;\&\; \pi_{L+1} = \varepsilon \mid \pi_i = k)$. Then

$$\begin{aligned}
\Pr(\pi_i = k \mid x \;\&\; \pi_{L+1} = \varepsilon) &= \frac{f_k(i) b_k(i)}{\Pr(x \;\&\; \pi_{L+1} = \varepsilon)} \\
&= \frac{f_k(i) b_k(i)}{\sum_k f_k(L) a_{k\varepsilon}} \qquad (2)
\end{aligned}$$

by Equation (1).

All that remains to compute the posterior probability via Equation (2), then, is to show how to compute $b_k(i)$. This is done by the backward algorithm.

3.3.3 Backward algorithm

The backward algorithm is dual to the forward algorithm, starting at the end of the sequence x and working backwards.

$$\begin{aligned}
b_k(L) &= a_{k\varepsilon}, \\
b_k(i) &= \sum_j a_{kj} e_j(x_{i+1}) b_j(i+1) \text{ for } i < L.
\end{aligned}$$

Note that you could calculate $b_k(i)$ using just the forward algorithm (similar to the way it is used in Equation (1)), but it would take time $\Theta(L^2)$ instead of time $\Theta(L)$ to compute it for all values of i. That is, you would lose the efficiency benefit of dynamic programming.

3.3.4 An example

Figure 3.6 of Durbin *et al.* shows the result of the forward-backward algorithm on the same sequence of die rolls given in Figure 3.5. One would predict the fair die whenever the posterior probability curve is above the horizontal line Pr(fair) = 1/2 and the loaded die whenever the curve is below that line. Notice that, unlike the Viterbi prediction in Figure 3.5, the forward-backward prediction in Figure 3.6 captures the third fair die stretch and the fourth loaded die stretch.

3.4 Parameter estimation for HMMs

The HMM training data needed to estimate the parameters a_{jk} and $e_j(b)$ consists of a set of sequences for which the path through the HMM is known. For the C$_p$G island problem, these would be genomic sequences with C$_p$G islands labeled. For the gene prediction problem, these would be genomic sequences with exons and introns labeled.

Let A_{jk} be the number of times the training sequences traverse the directed edge (j, k) of the HMM, and let $E_j(b)$ be the number of times the training sequences emit b when in state j. Then set

$$\begin{aligned} a_{jk} &= \frac{A_{jk}}{\sum_{k'} A_{jk'}} , \\ e_j(b) &= \frac{E_j(b)}{\sum_{b'} E_j(b')} . \end{aligned}$$

This method of estimation of a_{jk} is exactly the same as was used for Markov chains in Section 2.1.

What if the training data never uses state j? Then a_{jk} and $e_j(b)$ would be undefined. The solution is to modify A_{jk} and $E_j(b)$ by adding to them small "pseudocounts". A reasonable pseudocount to add to each A_{jk} is the "prior probability" of making the transition to state k. In the C$_p$G island problem, it would be reasonable to use 1/4 for each k in the same component as j, that is, either j and k are both + states or both − states.

References

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G., *Biological Sequence Analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.