Markov Chain Models

(Slides courtesy of Dr. Mark Craven)
Motivation for Markov Models in Computational Biology

• there are many cases in which we would like to represent the statistical regularities of some class of sequences
  – genes
  – various regulatory sites in DNA (e.g. where RNA polymerase and transcription factors bind)
  – proteins in a given family
• Markov models are well suited to this type of task
A Markov Chain Model

transition probabilities

\[
\begin{align*}
\Pr(x_i = a \mid x_{i-1} = g) &= 0.16 \\
\Pr(x_i = c \mid x_{i-1} = g) &= 0.34 \\
\Pr(x_i = g \mid x_{i-1} = g) &= 0.38 \\
\Pr(x_i = t \mid x_{i-1} = g) &= 0.12
\end{align*}
\]
Markov Chain Models

- A Markov chain model is defined by
  - A set of states
    - Some states emit symbols
    - Other states (e.g., the begin state) are silent
  - A set of transitions with associated probabilities
    - The transitions emanating from a given state define a distribution over the possible next states
Markov Chain Models

• given some sequence $x$ of length $L$, we can ask how probable the sequence is given our model

• for any probabilistic model of sequences, we can write this probability as

\[
\Pr(x) = \Pr(x_L, x_{L-1}, \ldots, x_1)
\]
\[
= \Pr(x_L \mid x_{L-1}, \ldots, x_1) \Pr(x_{L-1} \mid x_{L-2}, \ldots, x_1) \ldots \Pr(x_1)
\]

• key property of a (1st order) Markov chain: the probability of each $x_i$ depends only on the value of $x_{i-1}$

\[
\Pr(x) = \Pr(x_L \mid x_{L-1}) \Pr(x_{L-1} \mid x_{L-2}) \ldots \Pr(x_2 \mid x_1) \Pr(x_1)
\]
\[
= \Pr(x_1) \prod_{i=2}^{L} \Pr(x_i \mid x_{i-1})
\]
The Probability of a Sequence for a Given Markov Chain Model

\[ \Pr(cggt) = \Pr(c) \Pr(g \mid c) \Pr(g \mid g) \Pr(t \mid g) \]
Markov Chain Models

• can also have an end state; allows the model to represent
  – a distribution over sequences of different lengths
  – preferences for ending sequences with certain symbols
Markov Chain Notation

- the transition parameters can be denoted by $a_{x_{i-1}x_i}$ where
  $$a_{x_{i-1}x_i} = \Pr(x_i \mid x_{i-1})$$

- similarly we can denote the probability of a sequence $x$ as
  $$a_{Bx_1} \prod_{i=2}^{L} a_{x_{i-1}x_i} = \Pr(x_1) \prod_{i=2}^{L} \Pr(x_i \mid x_{i-1})$$

where $a_{Bx_1}$ represents the transition from the begin state
Example Application

• CpG islands
  – CG dinucleotides are rarer in eukaryotic genomes than expected given the marginal probabilities of C and G
  – but the regions upstream of genes are richer in CG dinucleotides than elsewhere – *CpG islands*
  – useful evidence for finding genes
• could predict CpG islands with Markov chains
  – one to represent CpG islands
  – one to represent the rest of the genome
Estimating the Model Parameters

• given some data (e.g. a set of sequences from CpG islands), how can we determine the probability parameters of our model?

• one approach: maximum likelihood estimation
  – given a set of data $D$
  – set the parameters $\theta$ to maximize

  $\Pr(D \mid \theta)$

  – i.e. make the data $D$ look likely under the model
Maximum Likelihood Estimation

• suppose we want to estimate the parameters $\Pr(a)$, $\Pr(c)$, $\Pr(g)$, $\Pr(t)$
• and we’re given the sequences
  accgcgctta
  gcttagtgac
  tagccgttac
• then the maximum likelihood estimates are

$$
\begin{align*}
\Pr(a) &= \frac{6}{30} = 0.2 \\
\Pr(c) &= \frac{9}{30} = 0.3 \\
\Pr(g) &= \frac{7}{30} = 0.233 \\
\Pr(t) &= \frac{8}{30} = 0.267
\end{align*}
$$
Maximum Likelihood Estimation

• suppose instead we saw the following sequences
  gccgcgcttg
  gcttggtggc
  tggcgcgtttgc

• then the maximum likelihood estimates are

\[
\begin{align*}
\Pr(a) &= \frac{0}{30} = 0 \\
\Pr(c) &= \frac{9}{30} = 0.3 \\
\Pr(g) &= \frac{13}{30} = 0.433 \\
\Pr(t) &= \frac{8}{30} = 0.267
\end{align*}
\]

Do we really want to set this to 0?
A Bayesian Approach

• instead of estimating parameters strictly from the data, we could start with some prior belief for each
• for example, we could use *Laplace estimates*

\[
Pr(a) = \frac{n_a + 1}{\sum_i (n_i + 1)}
\]

• where \( n_i \) represents the number of occurrences of character \( i \)
• using Laplace estimates with the sequences

\[
gccgcgcgttg \quad Pr(a) = \frac{0 + 1}{34}
\]

\[
gcttggtggc
\]

\[
tggccgttgc \quad Pr(c) = \frac{9 + 1}{34}
\]
A Bayesian Approach

• a more general form: \( m\)-estimates

\[
\Pr(a) = \frac{n_a + p_a m}{\left( \sum_i n_i \right) + m}
\]

- prior probability of \( a \)
- number of “virtual” instances

• with \( m=8 \) and uniform priors

\[
gccgcgcttg
\]

\[
gcttgggtggc
\]

\[
tggccgttgtgc
\]

\[
\Pr(c) = \frac{9 + 0.25 \times 8}{30 + 8} = \frac{11}{38}
\]
Estimation for 1\textsuperscript{st} Order Probabilities

- to estimate a 1\textsuperscript{st} order parameter, such as \(\Pr(c|g)\), we count the number of times that \(g\) follows the history \(c\) in our given sequences

- using Laplace estimates with the sequences

\[
gccgcgccttg \quad gcttggtggc \quad tggccgttgc
\]

\[
\begin{align*}
\Pr(a \mid g) &= \frac{0 + 1}{12 + 4} \\
\Pr(a \mid c) &= \frac{0 + 1}{7 + 4} \\
\Pr(c \mid g) &= \frac{7 + 1}{12 + 4} \\
\Pr(g \mid g) &= \frac{3 + 1}{12 + 4} \\
\Pr(t \mid g) &= \frac{2 + 1}{12 + 4}
\end{align*}
\]
Markov Chain Models

- can also have an end state; allows the model to represent
  - a distribution over sequences of different lengths
  - preferences for ending sequences with certain symbols
A Simple HMM

- given say a \( T \) in our input sequence, which state emitted it?
Hidden State

• we’ll distinguish between the *observed* parts of a problem and the *hidden* parts
• in the Markov models we’ve considered previously, it is clear which state accounts for each part of the observed sequence
• in the model above, there are multiple states that could account for each part of the observed sequence
  – this is the hidden part of the problem
The Parameters of an HMM

• as in Markov chain models, we have transition probabilities

\[ a_{kl} = \Pr(\pi_i = l \mid \pi_{i-1} = k) \]

probability of a transition from state \( k \) to \( l \)

\( \pi \) represents a path (sequence of states) through the model

• since we’ve decoupled states and characters, we might also have emission probabilities

\[ e_k(b) = \Pr(x_i = b \mid \pi_i = k) \]

probability of emitting character \( b \) in state \( k \)
A Simple HMM

$a_{13}$: probability of a transition from state 1 to state 3

e_2(A): probability of emitting character $A$ in state 2
Three Important Questions

• How likely is a given sequence?
  the Forward algorithm

• What is the most probable “path” for generating a given sequence?
  the Viterbi algorithm

• How can we learn the HMM parameters given a set of sequences?
  the Forward-Backward (Baum-Welch) algorithm
How Likely is a Given Sequence?

- the probability that the path is taken and the sequence is generated: $x_1 \ldots x_L$

$$\Pr(x_1 \ldots x_L, \pi_0 \ldots \pi_N) = a_{0\pi_1} \prod_{i=1}^{L} e_{\pi_i} (x_i) a_{\pi_i \pi_{i+1}}$$

(assuming begin/end are the only silent states on path)
How Likely Is A Given Sequence?

\[
Pr(AAC, \pi) = a_{01} \times e_1(A) \times a_{11} \times e_1(A) \times a_{13} \times e_3(C) \times a_{35} \\
= 0.5 \times 0.4 \times 0.2 \times 0.4 \times 0.8 \times 0.3 \times 0.6
\]
How Likely is a Given Sequence?

• the probability over all paths is:

\[
\Pr(x_1...x_L) = \sum_{\pi} \Pr(x_1...x_L, \pi_0...\pi_N)
\]

• but the number of paths can be exponential in the length of the sequence...
• the Forward algorithm enables us to compute this efficiently
How Likely is a Given Sequence: The Forward Algorithm

• define \( f_k(i) \) to be the probability of being in state \( k \) having observed the first \( i \) characters of \( x \)

• we want to compute \( f_N(L) \), the probability of being in the end state having observed all of \( x \)

• can define this recursively
The Forward Algorithm

• because of the Markov property, don’t have to explicitly enumerate every path – use dynamic programming instead

• e.g. compute $f_4(i)$ using $f_2(i-1)$, $f_4(i-1)$
The Forward Algorithm

• initialization:

\[ f_0(0) = 1 \]

probability that we’re in start state and have observed 0 characters from the sequence.

\[ f_k(0) = 0, \quad \text{for } k \text{ that are not silent states} \]
The Forward Algorithm

• recursion for emitting states ($i = 1 \ldots L$):

$$f_l(i) = e_l(i) \sum_k f_k(i-1)a_{kl}$$

• recursion for silent states:

$$f_l(i) = \sum_k f_k(i)a_{kl}$$
The Forward Algorithm

- termination:

\[ Pr(x) = Pr(x_1...x_L) = f_N(L) = \sum_k f_k(L)a_{kN} \]

probability that we’re in the end state and have observed the entire sequence
Forward Algorithm Example

- given the sequence $x = \text{TAGA}$
Forward Algorithm Example

- given the sequence $x = TAGA$
- initialization

\[ f_0(0) = 1 \]
\[ f_1(0) = 0 \]
\[ f_5(0) = 0 \]

- computing other values

\[ f_1(1) = e_1(T) \times (f_0(0) \times a_{01} + f_1(0)a_{11}) = 0.3 \times (1 \times 0.5 + 0 \times 0.2) = 0.15 \]
\[ f_2(1) = 0.4 \times (1 \times 0.5 + 0 \times 0.8) \]
\[ f_1(2) = e_1(A) \times (f_0(1) \times a_{01} + f_1(1)a_{11}) = 0.4 \times (0 \times 0.5 + 0.15 \times 0.2) \]

... ...

\[ \Pr(TAGA) = f_5(4) = (f_3(4) \times a_{35} + f_4(4)a_{45}) \]
Forward Algorithm Note

• in some cases, we can make the algorithm more efficient by taking into account the minimum number of steps that must be taken to reach a state

• e.g. for this HMM, we don’t need to initialize or compute the values

\[ f_3(0), \ f_4(0), \ f_5(0), \ f_5(1) \]
Three Important Questions

- How likely is a given sequence?
- What is the most probable “path” for generating a given sequence?
- How can we learn the HMM parameters given a set of sequences?
Finding the Most Probable Path: The Viterbi Algorithm

• define $v_k(i)$ to be the probability of the most probable path accounting for the first $i$ characters of $x$ and ending in state $k$

• we want to compute $v_N(L)$, the probability of the most probable path accounting for all of the sequence and ending in the end state

• can define recursively

• can use DP to find $v_N(L)$ efficiently
Finding the Most Probable Path: The Viterbi Algorithm

- initialization:

\[
\nu_0(0) = 1 \\
\nu_k(0) = 0, \quad \text{for } k \text{ that are not silent states}
\]
The Viterbi Algorithm

• recursion for emitting states \((i = 1 \ldots L)\):

\[
\nu_l(i) = e_l(x_i) \max_k \left[ \nu_k(i - 1)a_{kl} \right]
\]

\[
\text{ptr}_l(i) = \arg \max_k \left[ \nu_k(i - 1)a_{kl} \right]
\]

keep track of most probable path

• recursion for silent states:

\[
\nu_l(i) = \max_k \left[ \nu_k(i)a_{kl} \right]
\]

\[
\text{ptr}_l(i) = \arg \max_k \left[ \nu_k(i)a_{kl} \right]
\]
The Viterbi Algorithm

- termination:

\[ \Pr(x, \pi) = \max_k \left( v_k(L) a_{kN} \right) \]

\[ \pi_L = \arg\max_k \left( v_k(L) a_{kN} \right) \]

- traceback: follow pointers back starting at \( \pi_L \)
Forward & Viterbi Algorithms

- Forward/Viterbi algorithms effectively consider all possible paths for a sequence
  - Forward to find probability of a sequence
  - Viterbi to find most probable path
- consider a sequence of length 4…
Three Important Questions

• How likely is a given sequence?
• What is the most probable “path” for generating a given sequence?
• How can we learn the HMM parameters given a set of sequences?
Learning Parameters

• if we know the state path for each training sequence, learning the model parameters is simple
  – no hidden state during training
  – count how often each parameter is used
  – normalize/smooth to get probabilities
  – process is just like it was for Markov chain models

• if we don’t know the path for each training sequence, how can we determine the counts?
  – key insight: estimate the counts by considering every path weighted by its probability
Learning without Hidden State

- learning is simple if we know the correct path for each sequence in our training set

- estimate parameters by counting the number of times each parameter is used across the training set
Learning with Hidden State

- if we don’t know the correct path for each sequence in our training set, consider all possible paths for the sequence

- estimate parameters through a procedure that counts the expected number of times each parameter is used across the training set
Learning Parameters: The Baum-Welch Algorithm

- *a.k.a* the Forward-Backward algorithm
- an *Expectation Maximization* (EM) algorithm
  - EM is a family of algorithms for learning probabilistic models in problems that involve hidden state
- in this context, the hidden state is the path that best explains each training sequence
Learning Parameters: The Baum-Welch Algorithm

- algorithm sketch:
  - initialize parameters of model
  - iterate until convergence
    - calculate the expected number of times each transition or emission is used
    - adjust the parameters to maximize the likelihood of these expected values
The Expectation Step

• we want to know the probability of producing sequence $x$ with the $i$ th symbol being produced by state $k$ (for all $x$, $i$, and $k$)
The Expectation Step

- the forward algorithm gives us $f_k(i)$, the probability of being in state $k$ having observed the first $i$ characters of $x$
The Expectation Step

- the *backward algorithm* gives us $b_k(i)$, the probability of observing the rest of $x$, given that we’re in state $k$ after $i$ characters.
The Expectation Step

- putting forward and backward together, we can compute the probability of producing sequence $x$ with the $i$ th symbol being produced by state $q$
The Expectation Step

• first, we need to know the probability of the \( i \) th symbol being produced by state \( k \), given sequence \( x \)

\[
Pr(\pi_i = k \mid x)
\]

• given this we can compute our expected counts for state transitions, character emissions
The Expectation Step

- the probability of producing $x$ with the $i$th symbol being produced by state $k$ is

\[
\Pr(\pi_i = k, x) = \Pr(x_1 \ldots x_i, \pi_i = k) \times \Pr(x_{i+1} \ldots x_L | \pi_i = k)
\]

- the first term is $f_k(i)$, computed by the forward algorithm

- the second term is $b_k(i)$, computed by the backward algorithm
The Backward Algorithm

- initialization:

\[ b_k(L) = a_{kN} \]

for states with a transition to \textit{end} state
The Backward Algorithm

• recursion \((i = L \ldots l)\):

\[
b_k(i) = \sum_l \begin{cases} a_{kl} b_l(i), & \text{if } l \text{ is silent state} \\ a_{kl} e_l(x_{i+1}) b_l(i + 1), & \text{otherwise} \end{cases}
\]
The Backward Algorithm

- termination:

\[
Pr(x) = Pr(x_1...x_L) = \sum_l \begin{cases} 
    a_{0l} b_l(0), & \text{if } l \text{ is silent state} \\
    a_{0l} e_l(x_1) b_l(1), & \text{otherwise}
\end{cases}
\]
The Expectation Step

• now we can calculate the probability of the \( i \) th symbol being produced by state \( k \), given \( x \)

\[
\Pr(\pi_i = k \mid x) = \frac{\Pr(\pi_i = k, x)}{\Pr(x)}
\]

\[
= \frac{f_k(i)b_k(i)}{\Pr(x)}
\]

\[
= \frac{f_k(i)b_k(i)}{f_N(L)}
\]
The Expectation Step

- now we can calculate the expected number of times letter \( c \) is emitted by state \( k \)
- here we’ve added the superscript \( j \) to refer to a specific sequence in the training set

\[
 n_{k,c} = \sum_{x^j} \left[ \frac{1}{\text{Pr}(x^j)} \sum_{\{i|x_i^j = c\}} f_k^j(i) b_k^j(i) \right]
\]

sum over sequences
sum over positions where \( c \) occurs in \( x \)
The Expectation Step

- and we can calculate the expected number of times that the transition from \( k \) to \( l \) is used

\[
\sum_j \sum_i f_k^j(i) a_{kl} e_l(x_{i+1}^j) b_l^j(i + 1) / \Pr(x^j)
\]

- or if \( l \) is a silent state

\[
\sum_j \sum_i f_k^j(i) a_{kl} b_l^j(i) / \Pr(x^j)
\]
The Maximization Step

- Let $n_{k,c}$ be the expected number of emissions of $c$ from state $k$ for the training set.
- estimate new emission parameters by:

$$e_k(c) = \frac{n_{k,c}}{\sum_{c'} n_{k,c'}}$$

- just like in the simple case
- but typically we’ll do some “smoothing” (e.g. add pseudocounts)
The Maximization Step

- let $n_{k \rightarrow l}$ be the expected number of transitions from state $k$ to state $l$ for the training set
- estimate new transition parameters by:

$$\alpha_{kl} = \frac{n_{k \rightarrow l}}{\sum_{m} n_{k \rightarrow m}}$$
The Baum-Welch Algorithm

- initialize the parameters of the HMM
- iterate until convergence
  - initialize $n_{k,c}$, $n_{k\rightarrow l}$ with pseudocounts
  - **E-step**: for each training set sequence $j = 1 \ldots n$
    - calculate $f_k(i)$ values for sequence $j$
    - calculate $b_k(i)$ values for sequence $j$
    - add the contribution of sequence $j$ to $n_{k,c}$, $n_{k\rightarrow l}$
  - **M-step**: update the HMM parameters using $n_{k,c}$, $n_{k\rightarrow l}$
Baum-Welch Algorithm Example

• given
  – the HMM with the parameters initialized as shown
  – the training sequences TAG, ACG

• we’ll work through one iteration of Baum-Welch
Baum-Welch Example (Cont)

- determining the forward values for TAG

\[ f_0(0) = 1 \]
\[ f_1(1) = e_1(T) \times a_{01} \times f_0(0) = 0.4 \times 1 = 0.4 \]
\[ f_1(2) = e_1(A) \times a_{11} \times f_1(1) = 0.4 \times 0.8 \times 0.4 = 0.128 \]
\[ f_2(2) = e_2(A) \times a_{12} \times f_1(1) = 0.1 \times 0.2 \times 0.4 = 0.008 \]
\[ f_2(3) = e_2(G) \times (a_{12} \times f_1(2) + a_{22} \times f_2(2)) = 0.4 \times (0.0008 + 0.0256) = 0.01056 \]
\[ f_3(3) = a_{23} \times f_2(3) = 0.9 \times 0.01056 = 0.009504 \]

- here we compute just the values that represent events with non-zero probability

- in a similar way, we also compute forward values for ACG
Baum-Welch Example (Cont)

- determining the backward values for TAG

\[ b_3(3) = 1 \]
\[ b_2(3) = a_{23} \times b_3(3) = 0.9 \times 1 = 0.9 \]
\[ b_2(2) = a_{22} \times e_2(G) \times b_2(3) = 0.1 \times 0.4 \times 0.9 = 0.036 \]
\[ b_1(2) = a_{12} \times e_2(G) \times b_2(3) = 0.2 \times 0.4 \times 0.9 = 0.072 \]
\[ b_1(1) = a_{11} \times e_1(A) \times b_1(2) + a_{12} \times e_2(A) \times b_2(2) = \]
\[ 0.8 \times 0.4 \times 0.072 + 0.2 \times 0.1 \times 0.036 = 0.02376 \]
\[ b_0(0) = a_{04} \times e_1(T) \times b_1(1) = 1.0 \times 0.4 \times 0.02376 = 0.009504 \]

- here we compute just the values that represent events with non-zero probability
- in a similar way, we also compute backward values for ACG
Baum-Welch Example (Cont)

• determining the expected emission counts for state 1

\[ n_{1,A} = \frac{f_1(2)b_1(2)}{f_3(3)} + \frac{f_1(1)b_1(1)}{f_3(3)} + 1 \]

\[ n_{1,C} = \frac{f_1(2)b_1(2)}{f_3(3)} + 1 \]

\[ n_{1,G} = 1 \]

\[ n_{1,T} = \frac{f_1(1)b_1(1)}{f_3(3)} + 1 \]

*note that the forward/backward values in these two columns differ; in each column they are computed for the sequence associated with the column*
Baum-Welch Example (Cont)

• determining the expected transition counts for state 1 (not using pseudocounts)

\[
\begin{align*}
    n_{1\rightarrow 1} &= \frac{f_1(1)a_{11}e_1(A)b_1(2)}{f_3(3)} + \frac{f_1(1)a_{11}e_1(C)b_1(2)}{f_3(3)} \\
    n_{1\rightarrow 2} &= \frac{f_1(1)a_{12}e_2(A)b_2(2) + f_1(2)a_{12}e_2(G)b_2(3)}{f_3(3)} + \frac{f_1(1)a_{12}e_2(C)b_2(2) + f_1(2)a_{12}e_2(G)b_2(3)}{f_3(3)}
\end{align*}
\]

• in a similar way, we also determine the expected emission/transition counts for state 2
Baum-Welch Example (Cont)

• determining probabilities for state 1

\[ e_1(A) = \frac{n_{1,A}}{n_{1,A} + n_{1,C} + n_{1,G} + n_{1,T}} \]

\[ e_1(C) = \frac{n_{1,C}}{n_{1,A} + n_{1,C} + n_{1,G} + n_{1,T}} \]

\[ a_{11} = \frac{n_{1\rightarrow1}}{n_{1\rightarrow1} + n_{1\rightarrow2}} \]

\[ a_{12} = \frac{n_{1\rightarrow2}}{n_{1\rightarrow1} + n_{1\rightarrow2}} \]
Markov Models Summary

• we considered models that varied in terms of order, in/homogeneity, hidden state
• three DP-based algorithms for HMMs: Forward, Backward and Viterbi
• we discussed three key tasks: learning, classification and segmentation
• the algorithms used for each task depend on whether there is hidden state (correct path known) in the problem or not
Comments on Markov Models

• there are many successful applications in computational biology
  – gene recognition and associated subtasks
  – protein family modeling
  – motif modeling
  – etc.

• there are many variants of the models/algorithms we considered here (some of these are covered in BMI/CS 776)
  – fixed length motif models
  – semi-markov models
  – stochastic context free grammars
  – Gibbs sampling for learning parameters