Modeling biological sequences

- Ability to generate DNA sequences of a certain type
  - Not exact alignment to previously known gene
  - Preserving "properties" of type, not identical sequence
- Ability to recognize DNA sequences of a certain type
  - What (hidden) state is most likely to have generated observations
  - Find set of states and transitions that generated a long sequence
- Ability to learn distinguishing characteristics of each type
  - Training our generative models on large datasets
  - Learn to classify unlabelled data

Scoring probability of states & observations

What is the likelihood of

\[ x = 1, 2, 1, 5, 6, 2, 1, 6, 2, 4 \]

emission transition emission transition emission

\[ p = \frac{1}{2} \times P(1 \mid \text{Fair}) P(\text{Fair}, 3 \mid \text{Fair}) P(2 \mid \text{Fair}) P(\text{Fair}, 2 \mid \text{Fair}) P(1 \mid \text{Fair}) \]

\[ = \frac{1}{2} \times \left( \frac{1}{10} \right)^2 \times \left( \frac{1}{2} \right)^2 \times \left( \frac{1}{6} \right)^2 \times \left( 0.95 \right)^2 \times \left( 0.05 \right)^2 \]

\[ = 2.8 \times 10^{-10} \]

The three main questions on HMMs

1. Evaluation
   
   GIVEN a HMM M, and a sequence x,
   
   FIND Prob[ x | M ]

2. Decoding
   
   GIVEN a HMM M, and a sequence x,
   
   FIND the sequence \( \pi \) of states that maximizes \[ P( x, \pi \mid M ) \]

3. Learning
   
   GIVEN a HMM M, with unspecified transition/emission probs., and a sequence x,
   
   FIND parameters \( \theta = (e_i(\cdot)), a_{ij} \) that maximize \[ P( x \mid \theta ) \]
Markov Chains for CpG islands: (1) Training

- **Training Set:**
  - set of DNA sequences w/ known CpG islands
- **Derive two Markov chain models:**
  - ‘+’ model: from the CpG islands
  - ‘-’ model: from the remainder of sequence
- **Transition probabilities for each model:**

\[
\sum_{s}^{+} \frac{c_{st}^{+}}{c_{st}} = t' \quad s't \quad s \quad t \\
\sum_{s}^{-} \frac{c_{st}^{-}}{c_{st}} = t' \quad s't \quad s \quad t
\]

- is the number of times letter \( t \) followed letter \( s \) inside the CpG islands
- is the number of times letter \( t \) followed letter \( s \) outside the CpG islands

Markov Chains for CpG islands: (2) Model comparison

- Evaluate \( P(x, \pi^+) \)
  - \( a_0, C+, A+, T+, G+, C+, G+, T+ \)
- Evaluate \( P(x, \pi^-) \)
  - \( b_0, C-, A-, T-, G-, C-, G-, T- \)

HMM for CpG islands

- Build a single model that combines both Markov chains:
  - ‘+’ states: A+, C+, G+, T+
    - Emit symbols: A, C, G, T in CpG islands
  - ‘-’ states: A-, C-, G-, T-
    - Emit symbols: A, C, G, T in non-islands
- Emission probabilities distinct for the ‘+’ and the ‘-’ states
  - Infer most likely set of states, giving rise to observed emissions
  - ‘Paint’ the sequence with + and - states

Probability of given path \( p \) & observations \( x \)

\[
P(p, x) = (a_{0, C})^* (a_{C, G}^+) (a_{G, C}^-) (a_{C, G}^+) (a_{G, 0})
\]

Evaluation

Finding most likely state path

Problem 1: Decoding

How can we find the most likely path?
Finding the most likely path

- Find path $\pi^*$ that maximizes total joint probability $P(x, \pi)$
- $P(x, \pi) = a_0^{\pi_1} \prod \phi_i(x_i) \times a_{\pi_i,\pi_{i+1}}$ start emission transition

The Viterbi Algorithm

<table>
<thead>
<tr>
<th>State</th>
<th>1</th>
<th>2</th>
<th>...</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>x_1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x_2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x_K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input: $x = x_1 \ldots x_N$

Initialization:
- $V_0(0) = 1, V_i(0) = 0$, for all $k > 0$

Iteration:
- $V_i(i) = e_k(x_i) \times \max_j a_{\pi_i,\pi_j} V_j(i-1)$

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

Traceback:
- Follow max pointers back

Running time and space:
- Time: $O(K^2N)$
- Space: $O(KN)$

Problem 2: Evaluation

Find the likelihood a sequence is generated by the model

Simple: Given the model, generate some sequence $x$

Complex: Given $x$, was it generated by the model?

Given a sequence $x$, what is the probability that $x$ was generated by the model (using any path)?
- $P(x) = \sum_{\pi} P(x, \pi)$

- Challenge: exponential number of paths
- (cheap) alternative:
  - Calculate probability over maximum (Viterbi) path $\pi^*$
- (real) solution:
  - Calculate sum iteratively using dynamic programming
The Forward Algorithm – derivation
Define the forward probability:

\[ f(t) = P(x_t, \ldots, x_N, \pi_t = l) \]

\[ = \sum_{\pi_{t-1}} P(x_{t-1}, \ldots, x_N, \pi_{t-1}, \pi_t = l) \theta_l(x_t) \]

\[ = \sum_{\pi_{t-1}} f(t-1) a_{l\pi_{t-1}} \theta_l(x_t) \]

\[ = \theta_l(x_t) \sum_{\pi_{t-1}} f(t-1) a_{l\pi_{t-1}} \]

Calculate total probability \[ \Sigma_t P(x_t, \pi_t) \] recursively

- Assume we know \( f(t) \) for the previous time step (i-1)
- Calculate \( f_k(i) = \theta_k(x_i) \times \sum_j (f_j(i-1) \times a_{jk}) \)

The Forward Algorithm

Input: \( x = x_1 \ldots x_N \)

Initialization:

\( f_0(0) = 1, f_k(0) = 0, \text{ for all } k > 0 \)

Iteration:

\( f_k(i) = \theta_k(x_i) \times \sum_j (f_j(i-1) \times a_{jk}) \)

Termination:

\( P(x, \pi^*) = \sum_k f_k(N) \)

Running time and space:

- Time: \( O(K^2N) \)
- Space: \( O(KN) \)

In practice:

- Sum of log scores is difficult
- \( \Rightarrow \) approximate \( \exp(1+p+q) \)
- \( \Rightarrow \) scaling of probabilities

Problem 3: Evaluation

Find the likelihood an emission is generated by a state

Calculate \( P(\pi_i = \text{CPG}^+ \mid x_i = \text{G}) \)

- With no knowledge (no characters)
  - \( P(\pi_i = k \mid x_i = \text{G}) = \text{most likely state (prior)} \)
  - Time spent in markov chain states

- With very little knowledge (just that character)
  - \( P(\pi_i = k \mid x_i = \text{G}) = (\text{prior}) \times (\text{most likely emission}) \)
  - Emission probabilities adjusted for time spent

- With knowledge of entire sequence (all characters)
  - \( P(\pi_i = k \mid x_i = \text{G}) = \text{AGCGCG...GATTATCGTCGTA} \)
  - Sum over all paths that emit 'G' at position 7
  - \( \Rightarrow \) Posterior decoding

Summary

- Generative model
  - Hidden states
  - Observed sequence
- 'Running' the model
  - Generate a random sequence
- Observing a sequence
  - What is the most likely path generating it?
    - Viterbi algorithm
  - What is the total probability generating it?
    - Sum probabilities over all paths
    - Forward algorithm
- Next: Classification
  - What is the probability that "CGGTACG" came from CpG+?
Motivation for the Backward Algorithm

We want to compute
\[ P(\pi_i = k \mid x) \]
the probability distribution on the \( i \)th position, given \( x \).

We start by computing
\[ P(\pi_i = k, x) = P(x_1 \ldots x_i, \pi_i = k, x_{i+1} \ldots x_N) \]
\[ = P(x_1 \ldots x_i, \pi_i = k) P(x_{i+1} \ldots x_N \mid x_1 \ldots x_i, \pi_i = k) \]
Forward, \( f_k(i) \) 
Backward, \( b_k(i) \)

The Backward Algorithm – derivation

Define the backward probability:
\[ b_k(i) = P(x_{i+1} \ldots x_N \mid \pi_i = k) \]
\[ = \sum_{\pi_{i+1}} \ldots \sum_{\pi_N} P(x_{i+1}, x_{i+2}, \ldots, x_N, \pi_{i+1}, \ldots, \pi_N \mid \pi_i = k) \]
\[ = \sum_{\pi_{i+1}} P(x_{i+1} \mid \pi_{i+1} = l) b_{l(i+1)} \]
\[ = \sum_{l} e_l(x_{i+1}) a_{kl} b_{l(i+1)} \]
Calculate total end probability recursively

• Assume we know \( b_l \) for the next time step \( (i+1) \)

• Calculate
\[ b_k(i) = \sum_{l} (e_l(x_{i+1}) a_{kl} b_{l(i+1)}) \]

Putting it all together: Posterior decoding

• \( P(k) = P(\pi_i = k \mid x) = (i)b_k(i) / P(x) \)
  – Probability that \( i \)th state is \( k \), given all emissions \( x \)
• Posterior decoding
  – Define most likely state for every of sequence \( x \)
  – \( x_i = \arg \max_k P(\pi_i \mid x) \)
• Posterior decoding ‘path’ \( \pi^* \)
  – For classification, more informative than Viterbi path \( \pi^* \)
  – More refined measure of “which hidden states” generated \( x \)
  – However, it may give an invalid sequence of states
  – Not all \( j \rightarrow k \) transitions may be possible

Summary

• Generative model
  – Hidden states
  – Observed sequence
• ’Running’ the model
  – Generate a random sequence
• Observing a sequence
  – What is the most likely path generating it?
    – Viterbi algorithm
  – What is the total probability generating it?
    – Sum probabilities over all paths
    – Forward algorithm
• Classification
  – What is the probability that “CGGTACG” came from Cpg+?
    – Forward + backward algorithm
  – What is the most probable state for every position
    – Posterior decoding
Problem 3: Learning

Estimate model parameters based on training data

Case 1. When the right answer is known

Given \( x = x_1, ..., x_N \) for which the true \( \pi = \pi_1, ..., \pi_N \) is known,

Define:

\[
A_{il} = \text{# times } k \to l \text{ transition occurs in } \pi \\
E_k(b) = \text{# times state } k \text{ in } \pi \text{ emits } b \text{ in } x
\]

We can show that the maximum likelihood parameters \( \theta \) are:

\[
a_{il} = \frac{A_{il}}{\sum_i A_{il}} \\
e_k(b) = \frac{E_k(b)}{\sum_c E_k(c)}
\]

Solution for small training sets:

Add pseudocounts:

\[
A_{il} = \text{# times } k \to l \text{ transition occurs in } \pi + r_{il} \\
E_k(b) = \text{# times state } k \text{ in } \pi \text{ emits } b \text{ in } x + r_k(b)
\]

\( r_{il}, r_k(b) \) are pseudocounts representing our prior belief

Larger pseudocounts \( \Rightarrow \) Strong prior belief

Small pseudocounts (\( c < 1 \)): just to avoid 0 probabilities

Case 2. When the right answer is unknown

Intuition: When we know the underlying states, the best estimate is the average frequency of transitions & emissions that occur in the training data

Drawback:

Given little data, there may be overfitting: \( P(x|\theta) \) is maximized, but \( \theta \) is unreasonable if \( 0 \) probabilities – VERY BAD

Example:

Given 10 casino rolls, we observe \( x = 2, 1, 5, 6, 1, 2, 3, 6, 2, 3 \)


Then:

\( a_{FF} = 1; a_{FL} = 0 \)

\( e_F(1) = e_F(3) = .2; e_F(2) = .3; e_F(4) = 0; e_F(5) = e_F(6) = .1 \)

Pseudocounts

Example: dishonest casino

We will observe player for one day, 500 rolls

Reasonable pseudocounts:

\( r_{FF} = r_{FL} = r_{LF} = r_{LL} = 1; \)

\( r_{FL} = r_{LF} = r_{FF} = r_{LL} = 1; \)

\( r_{F}(1) = r_{F}(2) = ... = r_{F}(6) = 20 \) (strong belief fair is fair)

\( r_{F}(1) = r_{F}(2) = ... = r_{F}(6) = 5 \) (wait and see for loaded)

Above #s pretty arbitrary – assigning priors is an art
Case 2. When the right answer is unknown

We don’t know the true $A_{k,b}$, $E_{j}(b)$

Idea:

- We estimate our “best guess” on what $A_{k,b}$, $E_{j}(b)$ are
- We update the parameters of the model, based on our guess
- We repeat

Starting with our best guess of a model $M$, parameters $\theta$:

Given $x = x_1 \ldots x_N$ for which the true $\pi_1 \ldots \pi_N$ is unknown,

We can get to a provably more likely parameter set $\theta$.

Principle: EXPECTATION MAXIMIZATION

1. Estimate $A_{k,b}$, $E_{j}(b)$ in the training data
2. Update $\theta$ according to $A_{k,b}$, $E_{j}(b)$
3. Repeat 1 & 2, until convergence

Estimating new parameters

To estimate $A_{k,b}$:

At each position $i$ of sequence $x$,

Find probability transition $k \rightarrow l$ is used:

$$P(\pi_i = k, \pi_{i+1} = l | x) = \frac{Q}{P(x)} = \frac{P(\pi_i = k, \pi_{i+1} = l, x_1 \ldots x_N)}{P(x)}$$

where $Q = P(x_1 \ldots x_i, \pi_i = k) = P(x_1 \ldots x_{i-1}, \pi_{i-1} = k) P(x_i | \pi_{i-1} = k)$

So: $P(\pi_i = k, \pi_{i+1} = l | x, \theta) = \frac{f_k(i) a_{k,l} e(x_{i+1}) b_{l}(i+1)}{P(x | \theta)}$

Similarly,

$$E_{j}(b) = \frac{1}{P(x)} \sum_i \{i | x_i = b\} f_j(i) b_j(i)$$

The Baum-Welch Algorithm

Initialization:

- Pick the best-guess for model parameters (or arbitrary)

Iteration:

1. Forward
2. Backward
3. Calculate $A_{k,b}$, $E_{j}(b)$
4. Calculate new model parameters $a_{k,l}$, $e_j(b)$
5. Calculate new log-likelihood $P(x | \theta)$

GUARANTEED TO BE HIGHER BY EXPECTATION-MAXIMIZATION

Until $P(x | \theta)$ does not change much
The Baum-Welch Algorithm – comments

Time Complexity:

\[ \# \text{iterations} \times O(KN) \]

- Guaranteed to increase the log likelihood of the model

\[ P(\theta | x) = P(x, \theta) / P(x) = P(x | \theta) / (P(x) P(\theta)) \]

- Not guaranteed to find globally best parameters
  - Converges to local optimum, depending on initial conditions

- Too many parameters / too large model: Overtraining

Alternative: Viterbi Training

Initialization: Same

Iteration:
1. Perform Viterbi, to find \( \pi^* \)
2. Calculate \( \alpha_k, \beta_k(b) \) according to \( \pi^* \) + pseudocounts
3. Calculate the new parameters \( a_{kl}, e_k(b) \)

Until convergence

Notes:
- Convergence is guaranteed – Why?
- Does not maximize \( P(x | \theta) \)
- In general, worse performance than Baum-Welch

What have we learned?

- Generative model
  - Hidden states / Observed sequence
- 'Running' the model
  - Generate a random sequence
- Observing a sequence
  - What is the most likely path generating it?
    - Viterbi algorithm
  - What is the total probability generating it?
    - Sum probabilities over all paths
    - Forward algorithm
- Classification
  - What is the probability that "CGTGACG" came from CpG+?
    - Forward + backward algorithm
  - What is the most probable state for every position
    - Posterior decoding
- Training
  - Estimating parameters of the HMM
    - When state sequence is known
      - Simply compute maximum likelihood A and E
  - When state sequence is not known
    - Baum-Welch: Iterative estimation of all paths / frequencies
    - Viterbi training: Iterative estimation of best path / frequencies