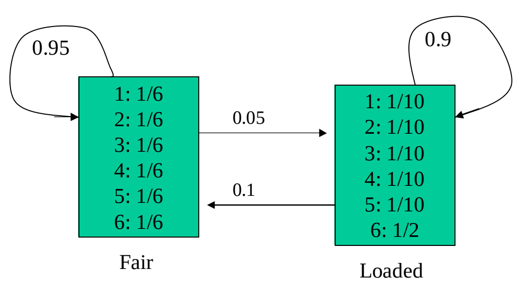
CISC 889 Bioinformatics (Spring 2004)

Hidden Markov Models (II)

a. Likelihood: forward algorithm
b. Decoding: Viterbi algorithm
c. Model building:
   – Baum-Welch algorithm
   – Viterbi training

Hidden Markov models

- A Markov chain of states
- At each state, there are a set of possible observations
- E.g.,



- Three major problems
  – Most probable state path
  – The likelihood
  – Parameter estimation for HMMs

A biological example: CpG islands

- Higher rate of Methyl-C mutating to T in CpG dinucleotides → generally lower CpG presence in genome, except at some biologically important ranges, e.g., in promoters, **--** called CpG islands.
- The conditional probabilities are collected from ~ 60,000 bps human genome sequences, + stands for CpG islands and – for non CpG islands.

+	A	C	G	T
A	.180	.274	.426	.120
C	.171	.368	**.274**	.188
G	.161	.339	.375	.125
T	.079	.355	.384	.182

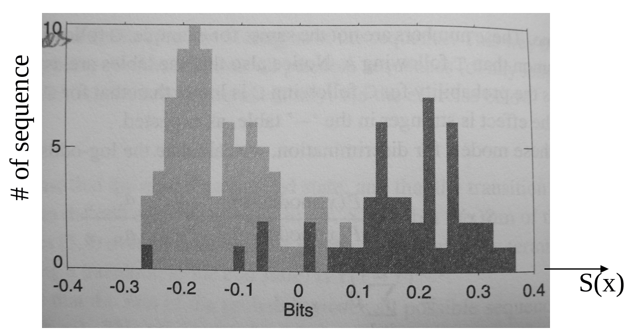
–	A	C	G	T
A	.300	.205	.285	.210
C	.322	.298	**.078**	.302
G	.248	.246	.298	.208
T	.177	.239	.292	.292

Task 1: given a sequence x, determine if it is a CpG island.

Solution: compute the log-odds ratio scored by the two Markov chains:

$$S(x) = \log [\, P(x \mid \text{model } +) / P(x \mid \text{model } -)]$$

Histogram of the length-normalized scores



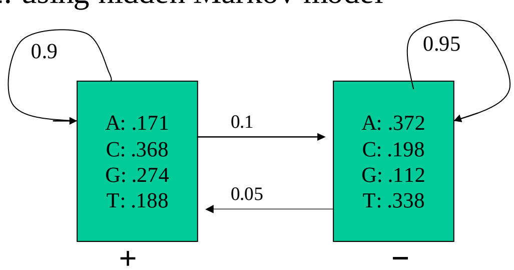
Task 2: For a *long* genomic sequence x, label these CpG islands, if there are any.

Approach 1: Adopt the method for Task 1 by calculating the log-odds score for a window of, say, 100 bps around every nucleotide and plotting it.

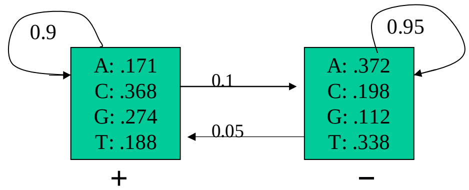
Problems with this approach:

- Won't do well if CpG islands have sharp boundary and variable length
- No effective way to choose a good Window size.

Approach 2: using hidden Markov model



- The model has two states, "+" for CpG island and "-" for non CpG island. Those numbers are made up here, and shall be fixed by learning from training examples. A reasonable assignment for emission frequencies may use the respective eigenvectors of the two Markov chains in approach 1.
- Use the same notations as in the text: a_{kl} is the transition probability from state k to state l; $e_k(b)$ is the emission frequency – probability that symbol b is seen when in state k.



Joint probability of an observed sequence x and a state path π:

$$P(x, \pi) = \prod_{i=1 \text{ to } L} e_{\pi i}(x_i)\, a_{\pi i\, \pi i+1}$$

```
i:123456789
x:TGCGCGTAC
π:--++++---
```

$P(x, \pi) = 0.388 \times 0.95 \times 0.112 \times 0.05 \times 0.368 \times 0.90 \times 0.274 \times 0.90 \times 0.368 \times 0.90 \times 0.274 \times 0.10 \times 0.338 \times 0.95 \times 0.372 \times 0.95 \times 0.198.$

Then, the probability to observe sequence x in the model is

$$P(x) = \Sigma_\pi P(x, \pi),$$

which is also called the likelihood of the model.

Q: Given a sequence x of length L, how many state paths do we have?

A: N^L, where N stands for the number of states in the model.

As an exponential function of the input size, it precludes enumerating all possible state paths for computing P(x).

Example: Let's assume the model has a set of two states $S = \{\alpha, \beta\}$. For L=3,

there are $2^3 = 8$ paths: $\Phi = \{\alpha\alpha\alpha, \alpha\alpha\beta, \alpha\beta\alpha, \beta\alpha\alpha, \alpha\beta\beta, \beta\alpha\beta, \beta\beta\alpha, \beta\beta\beta\}$.

$P(x) = \Sigma_\pi P(x, \pi)$

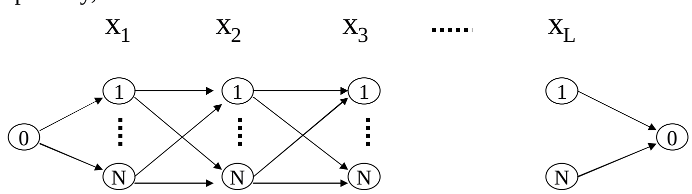
Specifically, let $f_k(i)$ be the intermediate result computed up to the k-th term in the i-th bracket,

$$\underbrace{[e_1(x_1) + \ldots + e_N(x_1)] \wedge_a \ldots \wedge_a [e_1(x_i) + \ldots e_k(x_i)}_{f_k(i)} + \ldots + e_N(x_i)] \wedge_a \ldots \wedge_a [e_1(x_L) + \ldots + e_N(x_L)]$$

which is the probability contributed by all paths from the beginning up to (and include) position i with state at position i being k.

And it can be written recursively as $f_k(i) = [\Sigma_j f_j(i-1)\, a_{jk}]\, e_k(x_i)$

Graphically,



A silent state 0 is introduced for better presentation

Forward algorithm

Initialization: $f_0(0) = 1$, $f_k(0) = 0$ for $k > 0$.

Recursion: $f_k(i) = e_k(x_i)\, \Sigma_j f_j(i-1)\, a_{jk}$.

Termination: $P(x) = \Sigma_k f_k(L)\, a_{k0}$.

Time complexity: $O(N^2L)$, where N is the number of states and L is the sequence length.

Similarly, we can compute P(x) backwards.

Backward algorithm

Initialization: $b_k(L) = a_{k0}$ for all k.

Recursion: $b_k(i) = \Sigma_j a_{kl}\, e_l(x_{i+1})\, b_l(i+1)$.

Termination: $P(x) = \Sigma_k a_{0k}\, e_k(x_1)\, b_k(1)$.

Therefore, $f_k(i)b_k(i)$ gives $P(x, \pi_i = k)$, the probability contributed from all paths that go through state k at position i.

Note: $b_k(i)$ does not include emitting x_i, this avoids double counting $e_k(x_i)$ in $f_k(i)b_k(i)$.

Decoding: Given an observed sequence x, what is the most probable state path, i.e.,

$$\pi^* = \mathrm{argmax}_{\pi}\, P(x, \pi)$$

$$\underbrace{[e_1(x_1) \vee \ldots \vee e_N(x_1)] \wedge_a \ldots \wedge_a [e_1(x_i) \vee \ldots e_k(x_i)}_{v_k(i)} \vee \ldots \vee e_N(x_i)] \wedge_a \ldots \wedge_a [e_1(x_L) \vee \ldots \vee e_N(x_L)]$$

Viterbi Algorithm

Initialization: $v_0(0) = 1$, $v_k(0) = 0$ for $k > 0$.

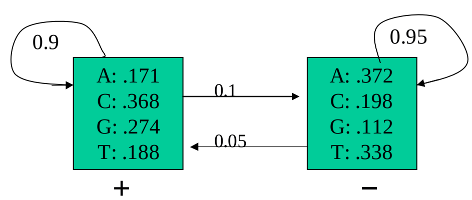
Recursion: $v_k(i) = e_k(x_i) \max_j (v_j(i-1)\, a_{jk})$;

$ptr_i(k) = \mathrm{argmax}_j (v_j(i-1)\, a_{jk})$;

Termination: $P(x,) = \max_k (v_k(L)\, a_{k0})$;

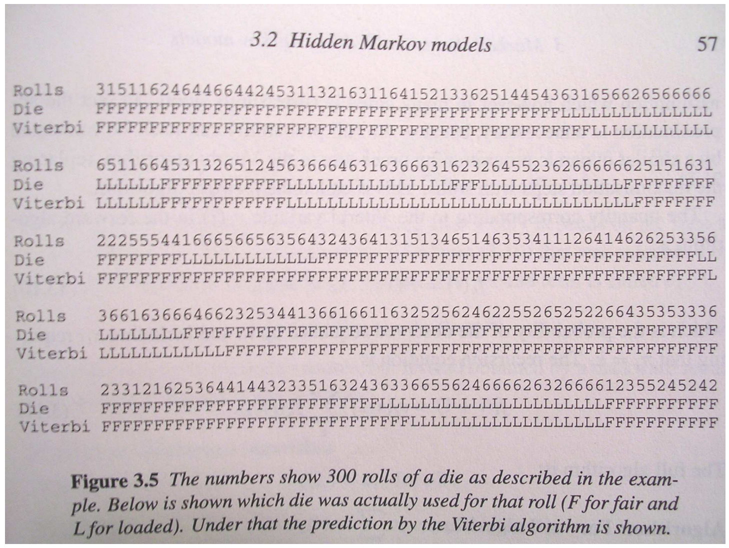
$\pi^*_L = \mathrm{argmax}_j (v_j(L)\, a_{j0})$;

Traceback: $\pi^*_{i-1} = ptr_i(\pi^*_i)$.



$$v_k(i) = e_k(x_i) \max_j (v_j(i-1)\, a_{jk});$$

$V_k(i)$	ε	T	G	C	T	A
0	1	0	0	0	0	0
+	0	.094	.023	.0076	.0013	.00020
−	0	.169	.018	.0033	.0011	.00038



3.2 *Hidden Markov models* 57

```
Rolls   315116246446644245311321631164152133625144543631656626566666
Die     FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLL

Rolls   651166453132651245636664631636663162326455236266666625151631
Die     LLLLLLFFFFFFFFFFFFLLLLLLLLLLLLLLLLLFFLLLLLLLLLLLLLLLFFFFFFFF
Viterbi LLLLLLFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFF

Rolls   222555441666566563564324364131513465146353411126414626253356
Die     FFFFFFFFLLLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFLL
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFL

Rolls   366163666466232534413661661163252562462255265252266435353336
Die     LLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Viterbi LLLLLLLLLLLLFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

Rolls   233121625364414432335163243633665562466662632666612355245242
Die     FFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFFF
Viterbi FFFFFFFFFFFFFFFFFFFFFFFFFFLLLLLLLLLLLLLLLLLLLLLLLLFFFFFFFFFF
```

Figure 3.5 *The numbers show 300 rolls of a die as described in the example. Below is shown which die was actually used for that roll (F for fair and L for loaded). Under that the prediction by the Viterbi algorithm is shown.*

Posterior decoding

$$P(\pi_i = k \mid x) = P(x, \pi_i = k) / P(x) = f_k(i) b_k(i) / P(x)$$

Algorithm:

for i = 1 to L

do $\operatorname{argmax}_k P(\pi_i = k \mid x)$

Notes: 1. Posterior decoding may be useful when there are multiple almost most probable paths, or when a function is defined on the states.

2. The state path identified by posterior decoding may not be most probable overall, or may not even be a viable path.

Model building

- Topology
  - Requires domain knowledge
- Parameters
  - When states are labeled
    - Simple counting:

$$a_{kl} = A_{kl} / \Sigma_{l'} A_{kl'} \text{ and } e_k(b) = E_k(b) / \Sigma_{b'} E_k(b')$$

  - When states are not labeled

    Method 1 (Viterbi training)

    1. Use Viterbi algorithm to label
    2. Do counting to collect new a_{kl} and $e_k(b)$;
    3. Repeat steps 1 and 2 until stopping criterion is met.

    Method 2 (Baum-Welch algorithm)

Baum-Welch algorithm (Expectation-Maximization)

- A iterative procedure similar to Viterbi training
- $P(\pi_i = k, \pi_{i+1} = l \mid x, \theta) = f_k(i)\, a_{kl}\, e_l(x_{i+1})\, b_l(i+1) / P(x)$

$$A_{kl} = \Sigma_j \{(1/P(x^j) [\Sigma_i f_k^j(i)\, a_{kl}\, e_l(x^j_{i+1})\, b_l^j(i+1)] \}$$

$$E_k(b) = \Sigma_j \{(1/P(x^j) [\Sigma_{\{i|x\_i\hat{}j = b\}} f_k^j(i)\, b_k^j(i)] \}$$